NoSQL DATABASES IN NODE.JS

Andrii Kochkin

JUNE 25, 2018

ABOUT ME

**ANDRII
KOCHKIN**

Senior Software Engineer

Andrii_Kochkin@epam.com

Development experience:

* 15+ years of development experience in Software Engineering with full-stack technical expertise;

* 5+ years of commercial experience as a JavaScript Developer;

Key Developer on CCC-ARCH project

Main Focus: Front-End and Backend Development

AGENDA OF THE LECTURE

- SQL vs NoSQL
- What is a MongoDB?
- MongoDB Node.js Driver
- Mongoose API overview
- CRUD operations
- Querying
- Data Validation and Manipulation

SQL vs NoSQL

NoSQL - ?

A history of NoSQL

1970: **NoSQL = We have no SQL**

1980: **NoSQL = Know SQL**

2000: **NoSQL = No SQL!**

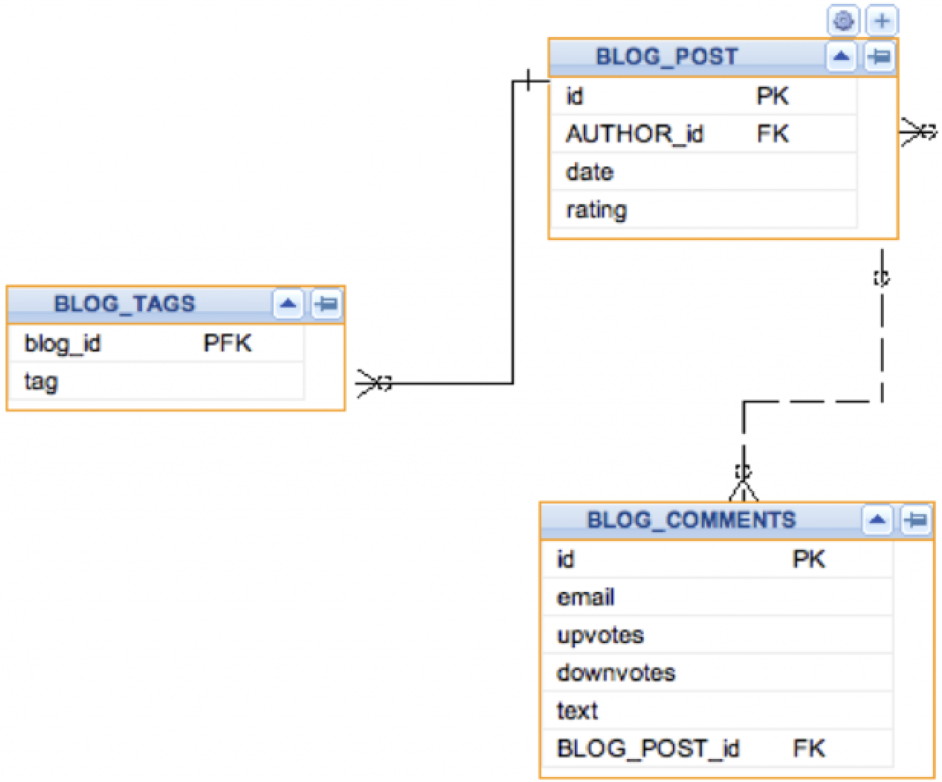
2005: **NoSQL = Not only SQL**

2016: **NoSQL = No, SQL!**

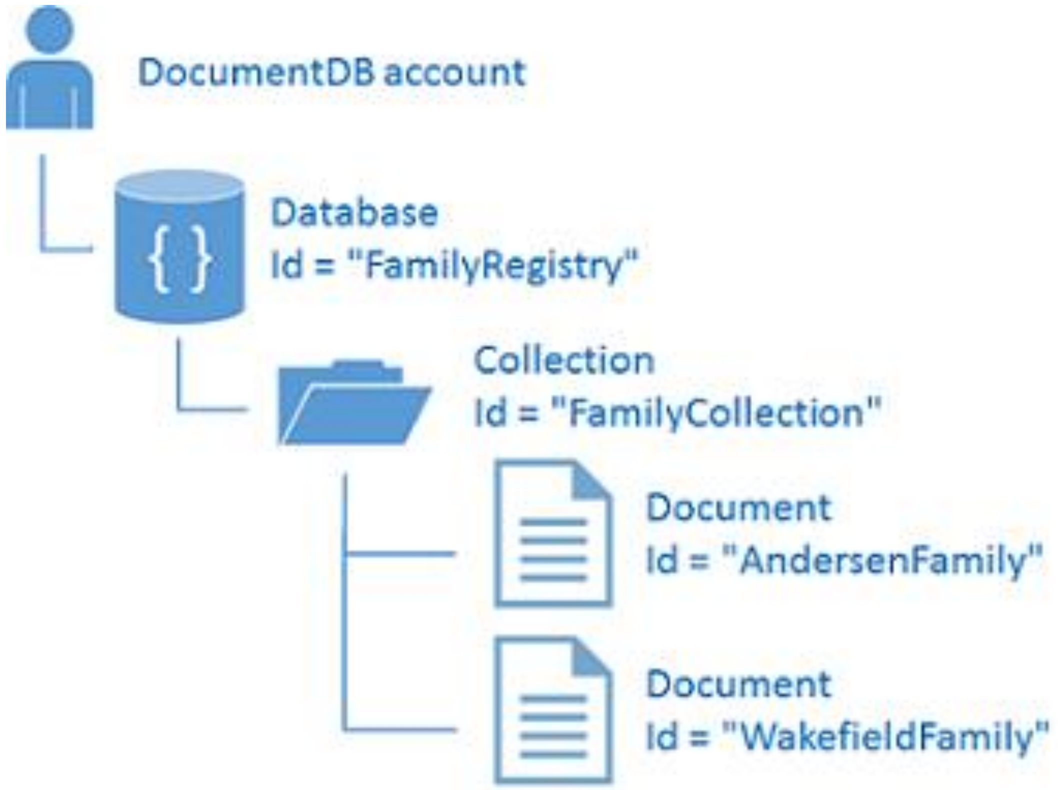
NoSQL

- “Not only SQL”
- Scalable by partitioning (sharding) and replication
- Distributed, fault-tolerant architecture
- Flexible schema – no fixed schema or structure
- Not a replacement for RDMBS but compliments it

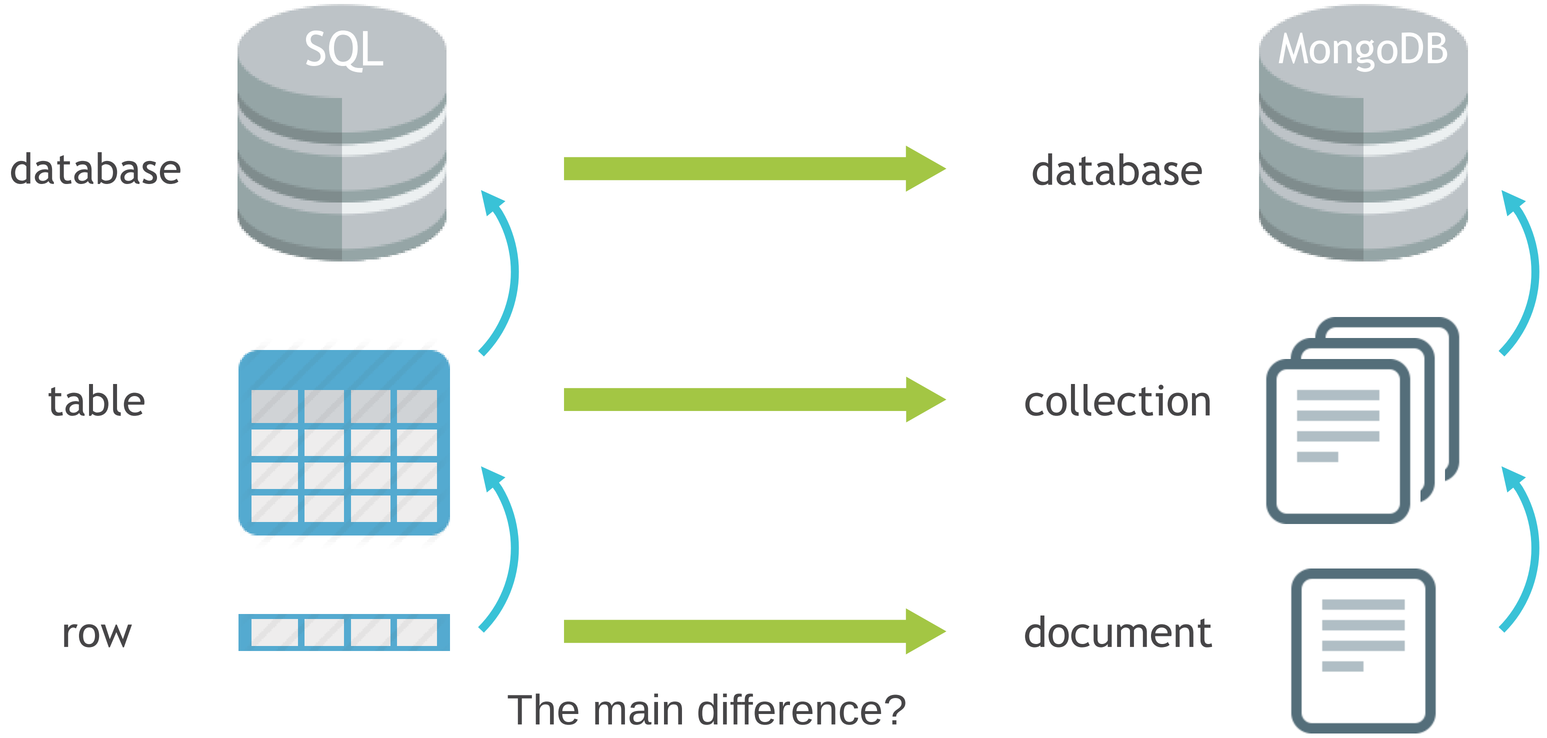
SQL DATA STRUCTURE



NoSQL DATA STRUCTURE

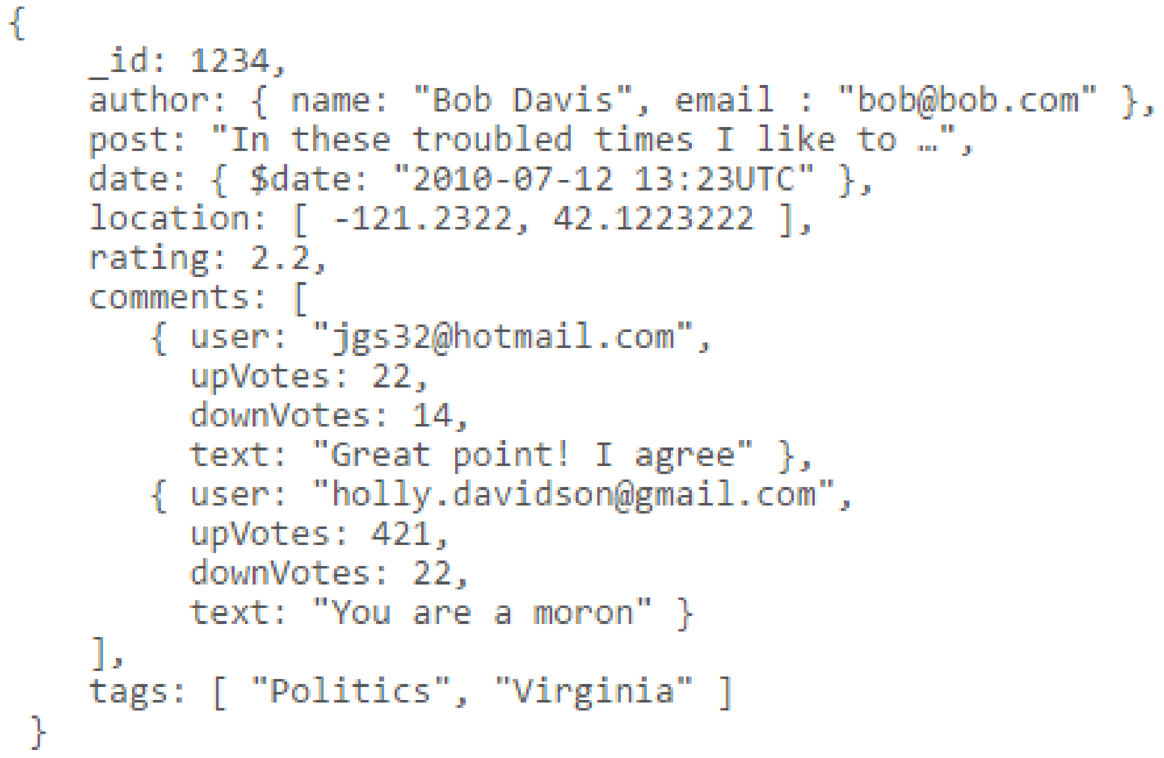


NoSQL Comparison to SQL



The main difference?
SQL is ***relational*** and MongoDB is ***document-oriented***

NoSQL DATA STRUCTURE

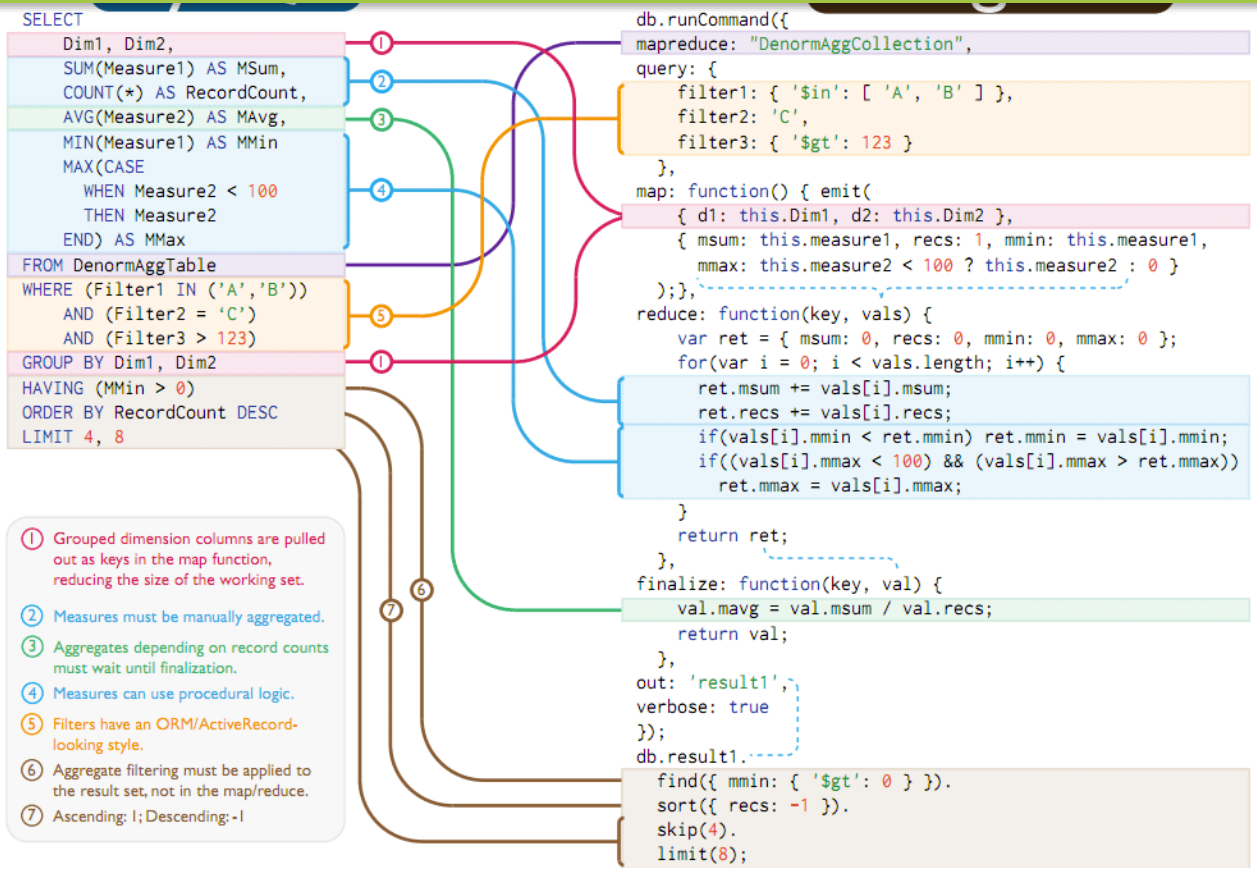


```
{
    _id: 1234,
    author: { name: "Bob Davis", email : "bob@bob.com" },
    post: "In these troubled times I like to …",
    date: { $date: "2010-07-12 13:23UTC" },
    location: [ -121.2322, 42.1223222 ],
    rating: 2.2,
    comments: [
       { user: "jgs32@hotmail.com",
         upVotes: 22,
         downVotes: 14,
         text: "Great point! I agree" },
       { user: "holly.davidson@gmail.com",
         upVotes: 421,
         downVotes: 22,
         text: "You are a moron" }
    ],
    tags: [ "Politics", "Virginia" ]
 }
```

SQL vs NoSQL DATA STRUCTURE

```
{    customer_id : 1,
     first_name : "Mark",
     last_name : "Smith",
     city : "San Francisco",
     phones: [ {
          type : “work”,
          number: “1-800-555-1212”
     },
     {    type : “home”,
          number: “1-800-555-1313”,
          DNC: true
     },
     {    type : “home”,
          number: “1-800-555-1414”,
          DNC: true
     }
   ]
}
```

SQL vs NoSQL DATA STRUCTURE



```sql
SELECT
    Dim1, Dim2,
    SUM(Measure1) AS MSum,
    COUNT(*) AS RecordCount,
    AVG(Measure2) AS MAvg,
    MIN(Measure1) AS MMin
    MAX(CASE
      WHEN Measure2 < 100
      THEN Measure2
    END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
    AND (Filter2 = 'C')
    AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
```

```javascript
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
    filter1: { '$in': [ 'A', 'B' ] },
    filter2: 'C',
    filter3: { '$gt': 123 }
  },
map: function() { emit(
    { d1: this.Dim1, d2: this.Dim2 },
    { msum: this.measure1, recs: 1, mmin: this.measure1,
      mmax: this.measure2 < 100 ? this.measure2 : 0 }
  );},
reduce: function(key, vals) {
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
    for(var i = 0; i < vals.length; i++) {
      ret.msum += vals[i].msum;
      ret.recs += vals[i].recs;
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
        ret.mmax = vals[i].mmax;
    }
    return ret;
  },
finalize: function(key, val) {
    val.mavg = val.msum / val.recs;
    return val;
  },
out: 'result1',
verbose: true
});
db.result1.
  find({ mmin: { '$gt': 0 } }).
  sort({ recs: -1 }).
  skip(4).
  limit(8);
```

① Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.

② Measures must be manually aggregated.

③ Aggregates depending on record counts must wait until finalization.

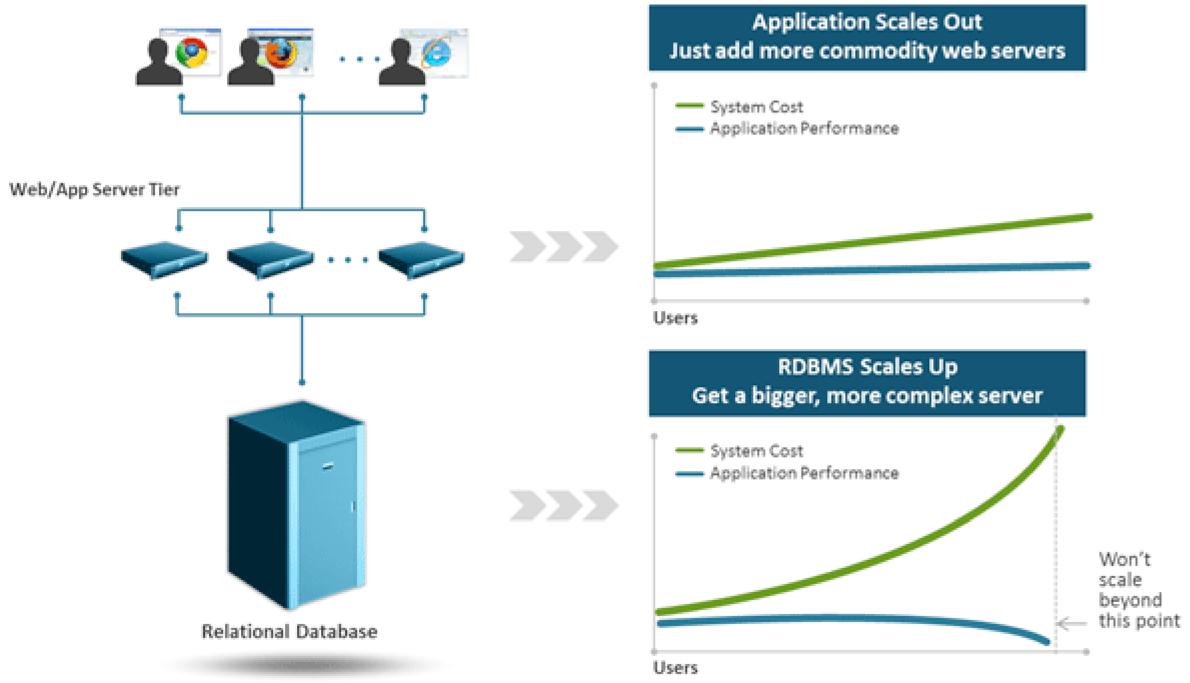
④ Measures can use procedural logic.

⑤ Filters have an ORM/ActiveRecord-looking style.

⑥ Aggregate filtering must be applied to the result set, not in the map/reduce.

⑦ Ascending: 1; Descending: -1

SQL vs NoSQL COST AND PERFOMANCE



TOP 5 NoSQL DB

1. CouchDB
2. MongoDB
3. Cassandra
4. Redis
5. HBase

WHAT IS MongoDB?

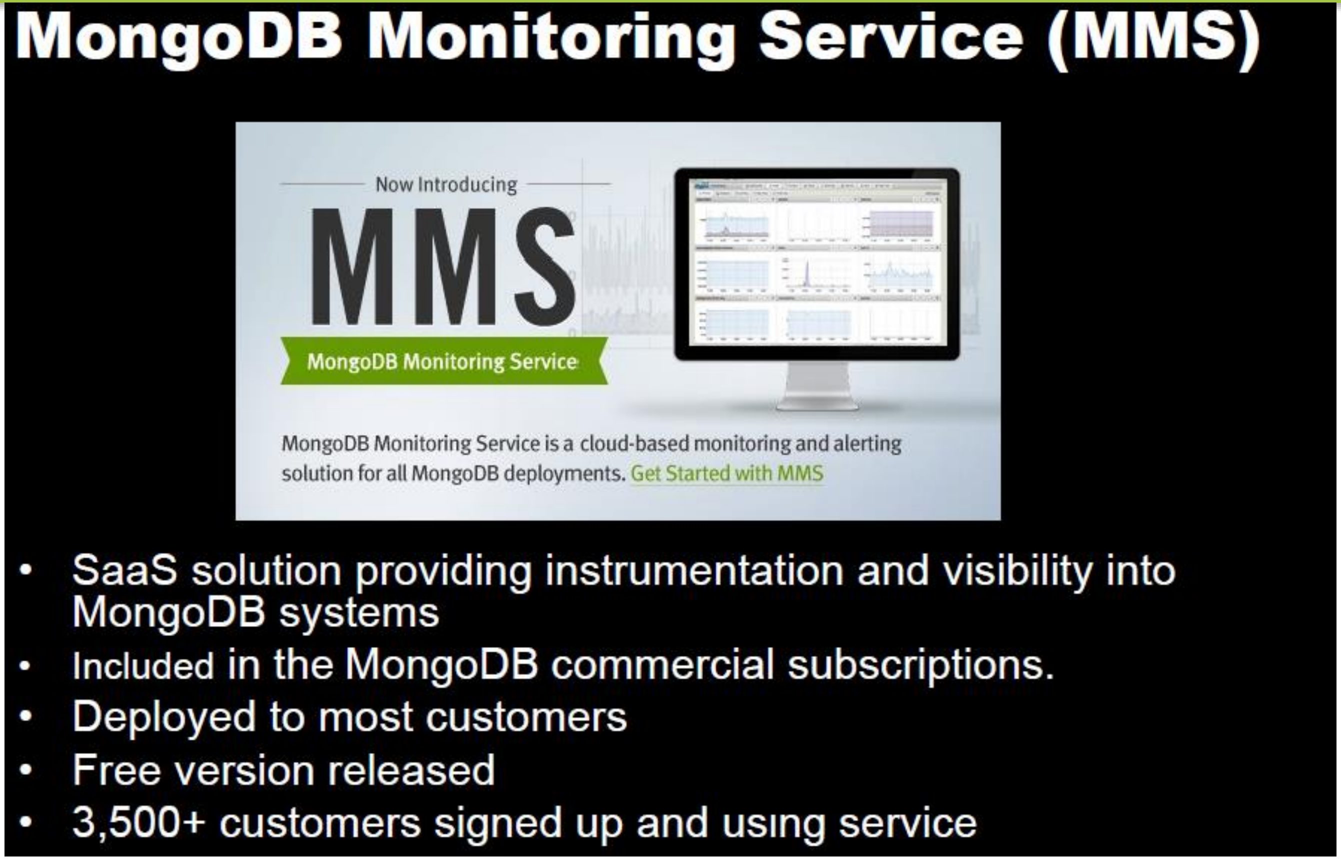
WHAT IS MONGODB?

MongoDB is an open source, document-oriented database designed with both scalability and developer agility in mind.

KEY HIGHLIGHTS

- Ad hoc queries
- Indexing
- Replication
- Load balancing
- File storage
- Aggregation
- Server-side Javascript
- Capped collections

MongoDB in PRODUCTION



MongoDB Monitoring Service (MMS)

Now Introducing

MMS

MongoDB Monitoring Service

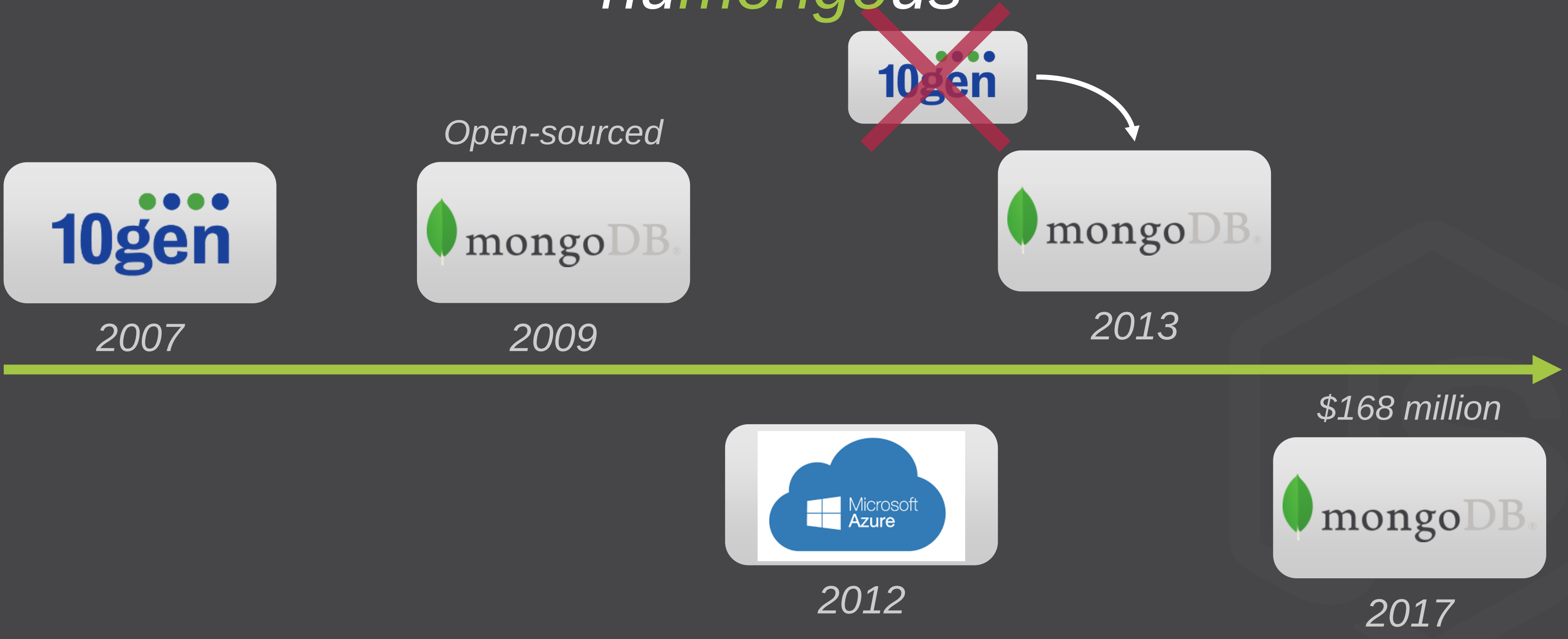
MongoDB Monitoring Service is a cloud-based monitoring and alerting solution for all MongoDB deployments. Get Started with MMS

- SaaS solution providing instrumentation and visibility into MongoDB systems
- Included in the MongoDB commercial subscriptions.
- Deployed to most customers
- Free version released
- 3,500+ customers signed up and using service

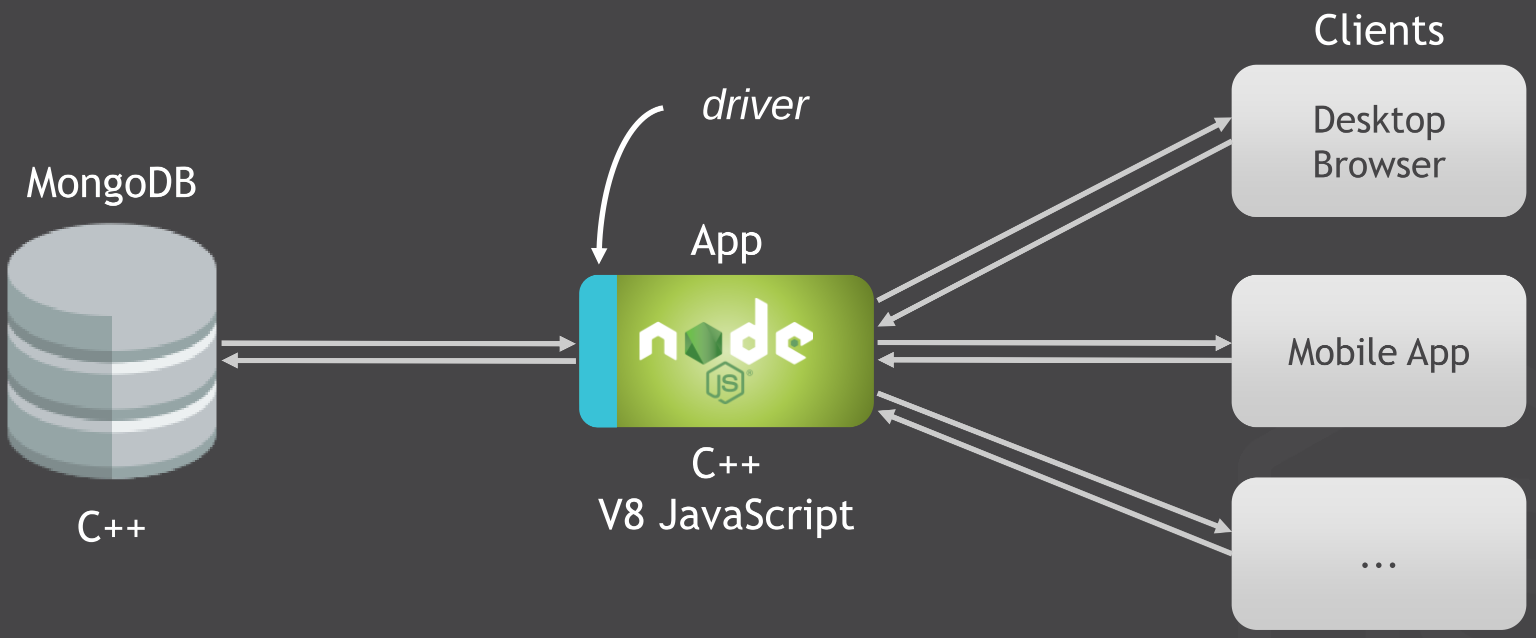
WHAT IS MONGODB?

HISTORY

Mongo stands for
humongous

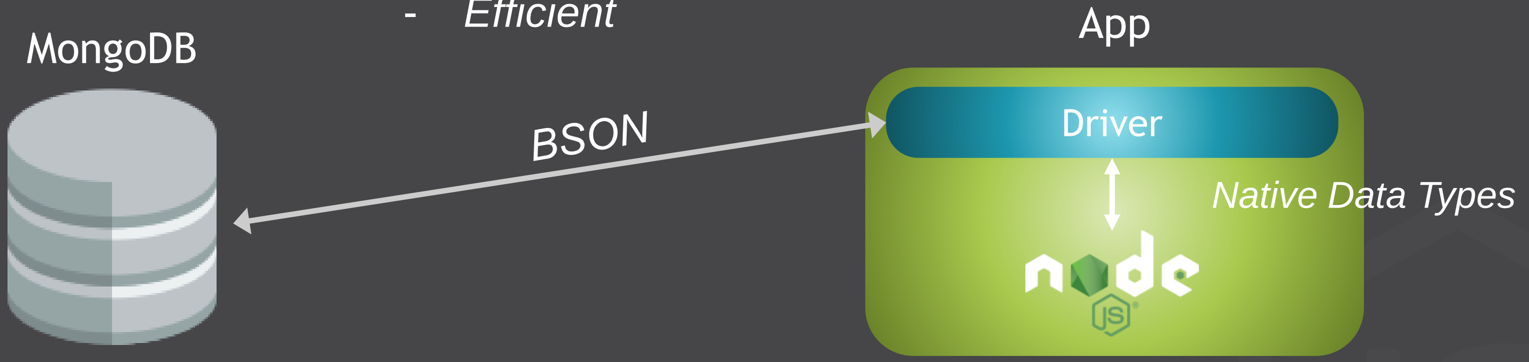


WHAT IS MONGODB?



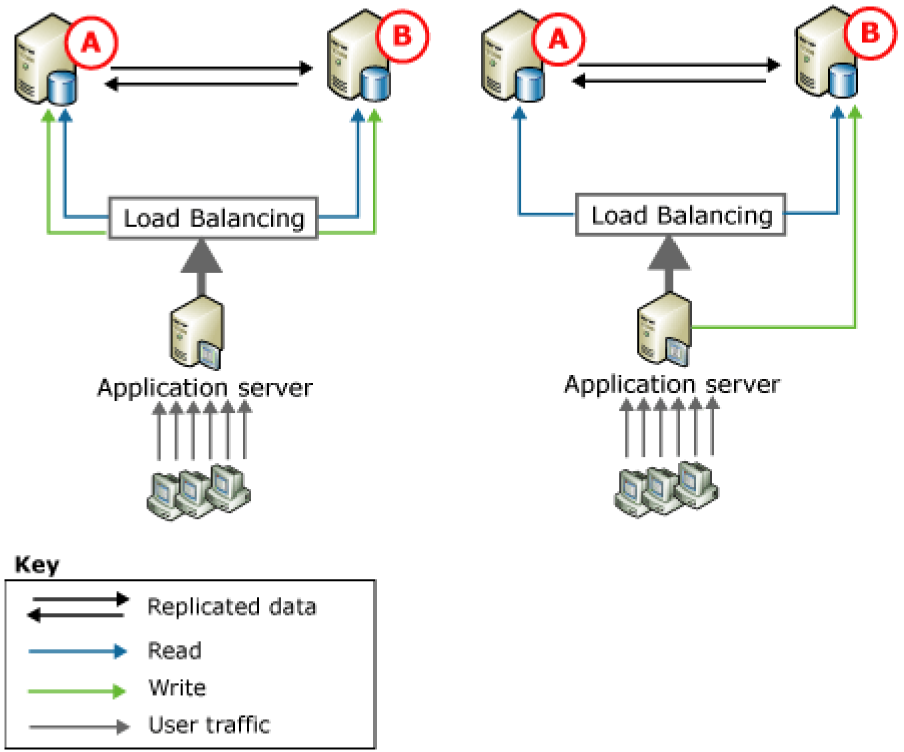
BSON – BINARY JSON

- *Lightweight*
- *Traversable*
- *Efficient*



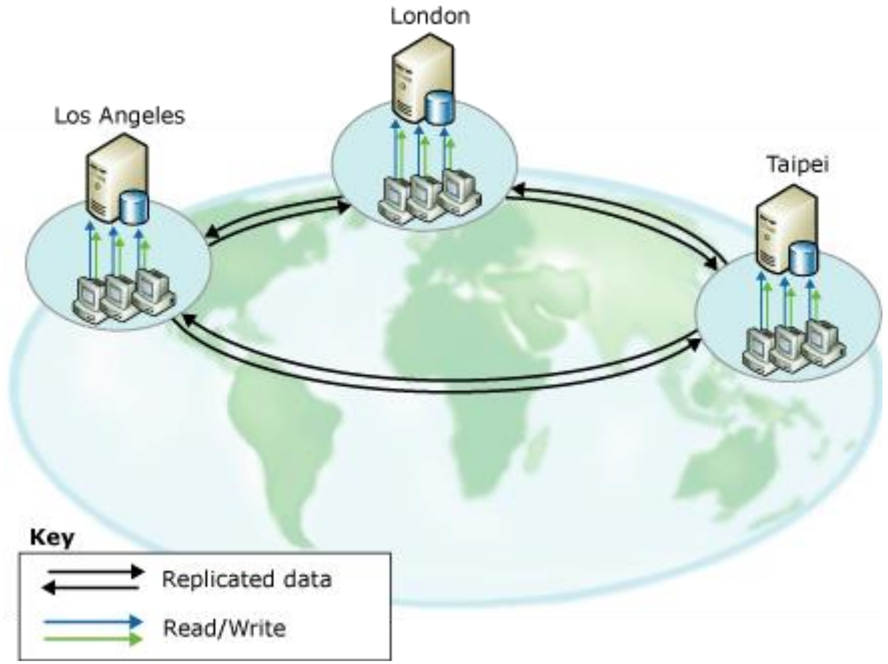
DB REPLICATION

Topology That Has Two Participating Databases

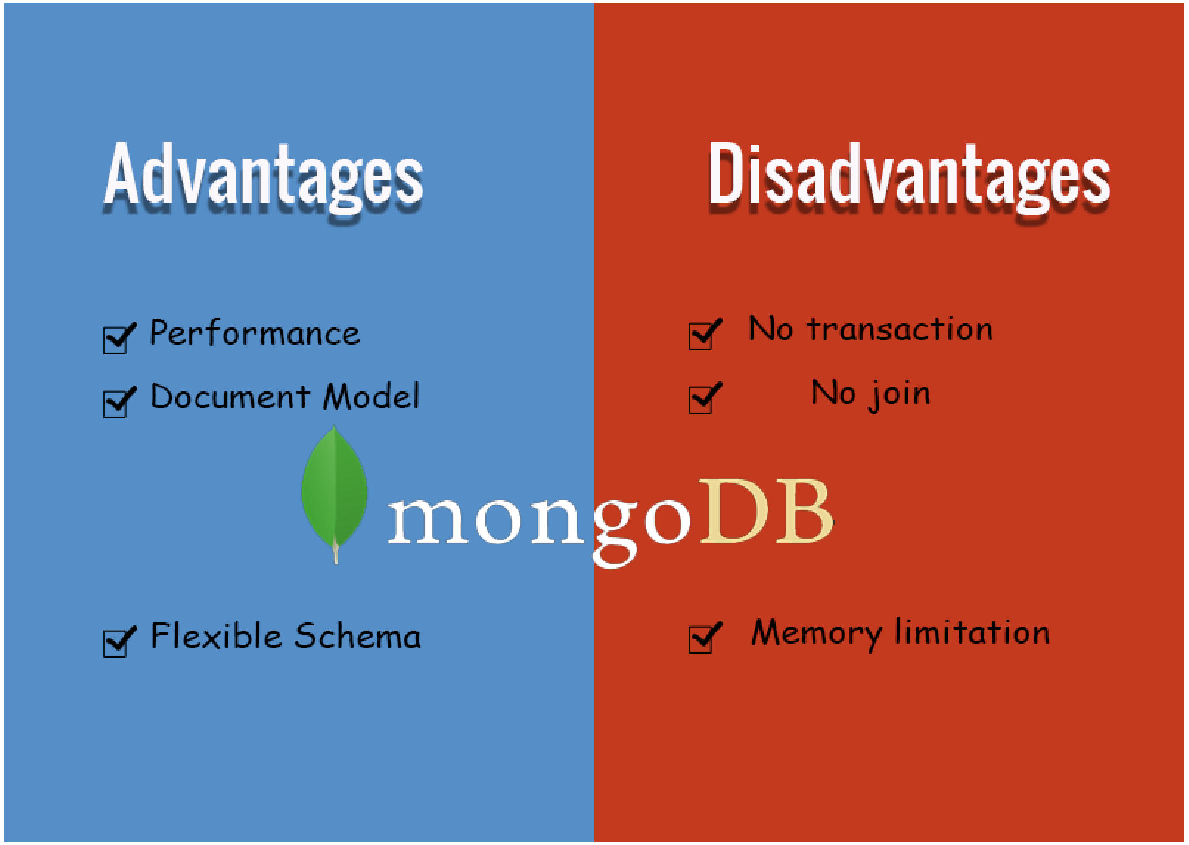


DB REPLICATION

Topologies That Have Three or More Participating Databases



MongoDB PROS and CONS



Advantages

- ✓ Performance
- ✓ Document Model
- ✓ Flexible Schema

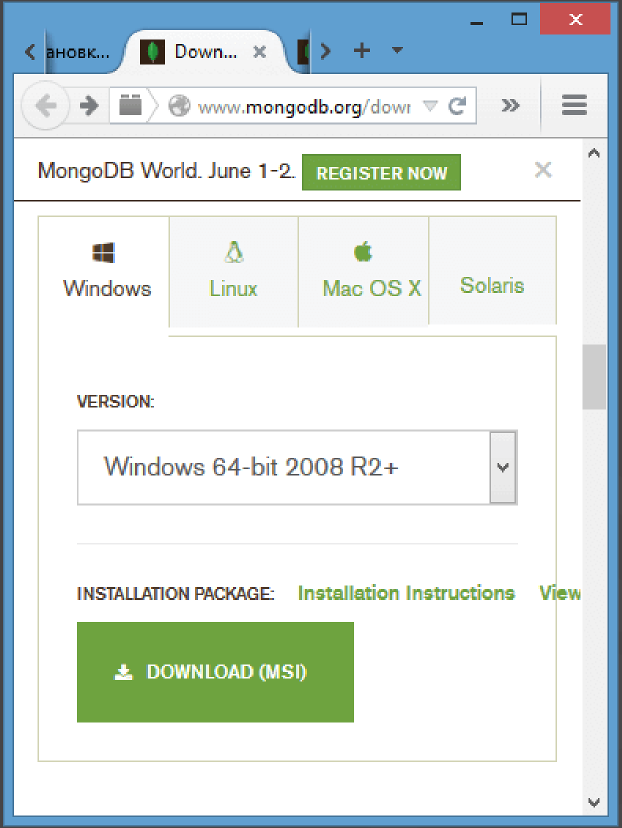
Disadvantages

- ✓ No transaction
- ✓ No join
- ✓ Memory limitation

INSTALL AND RUN MONGODB

Installation manual:

https://docs.mongodb.org/manual/installation/



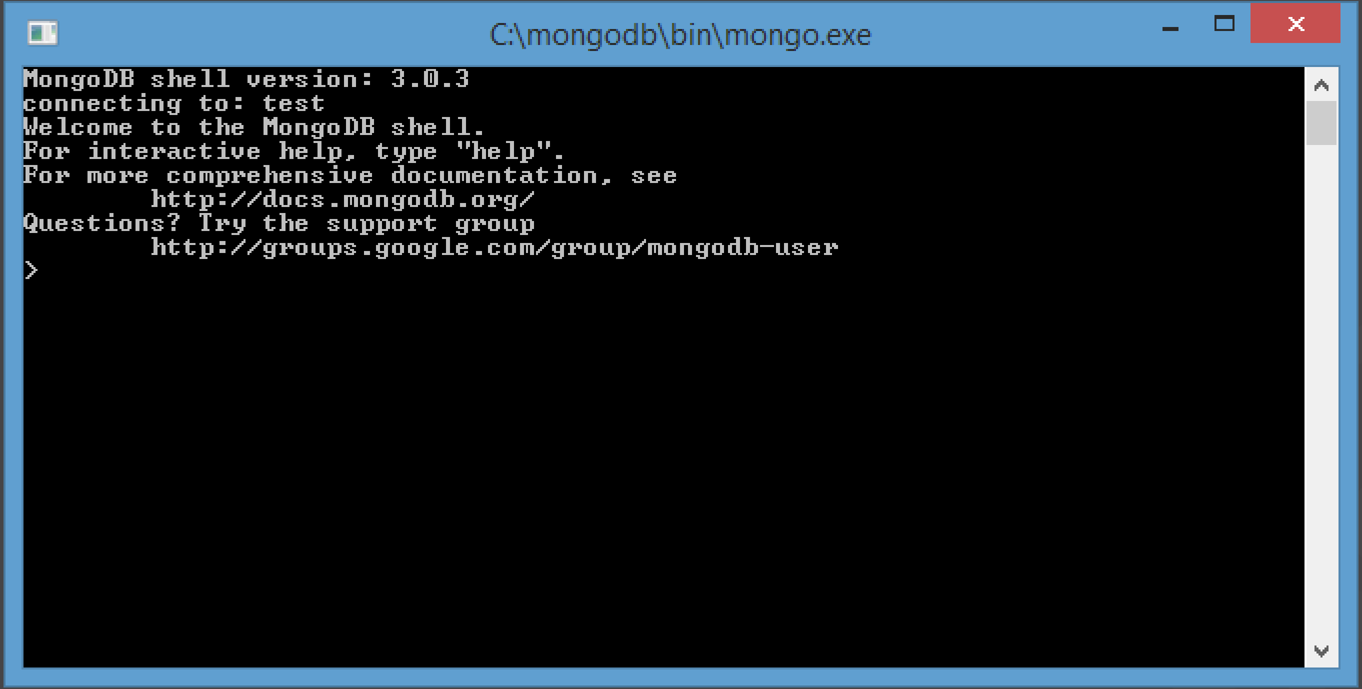
MONGO SERVER

Run server:

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

MONGODB CLI

Run CLI: mongo.exe



```
MongoDB shell version: 3.0.3
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
>
```

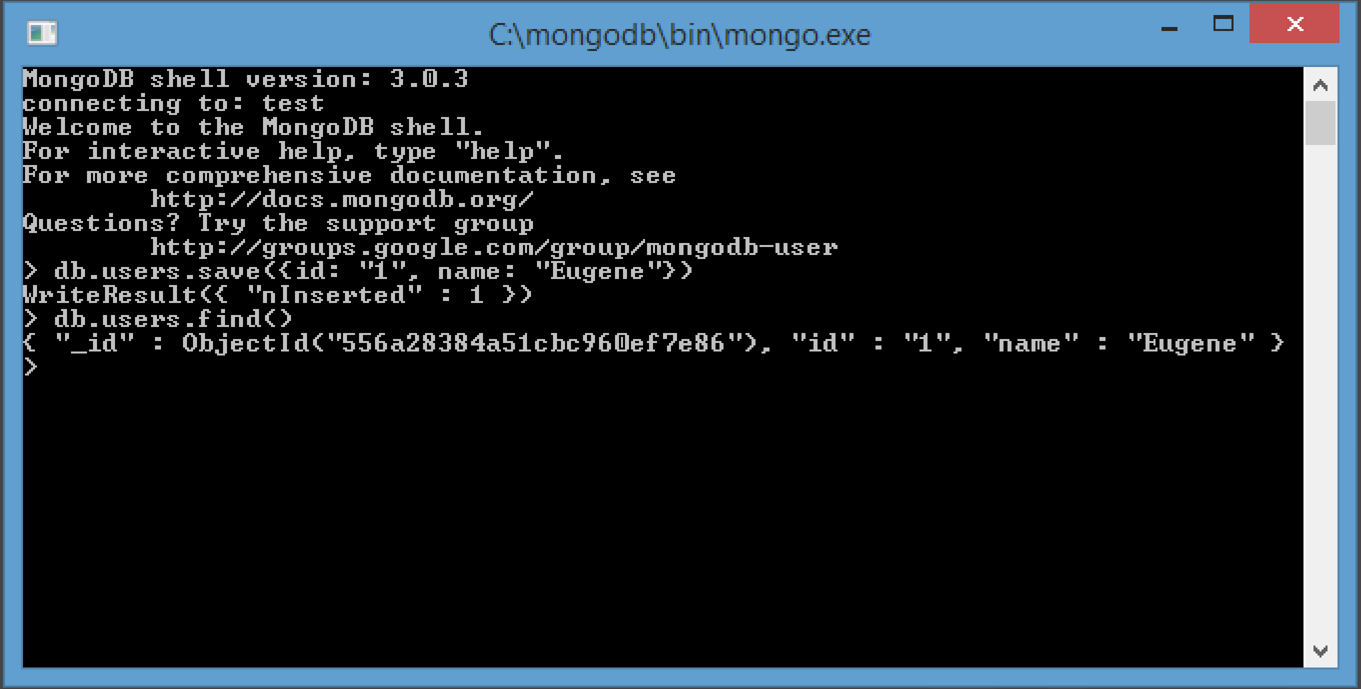
MongoDB CRUD Operations

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

MONGODB CLI

Working in mongo CLI

```
db.users.save({
   id: "1",
   name: "Eugene"
})
db.users.find()
```



C:\mongodb\bin\mongo.exe

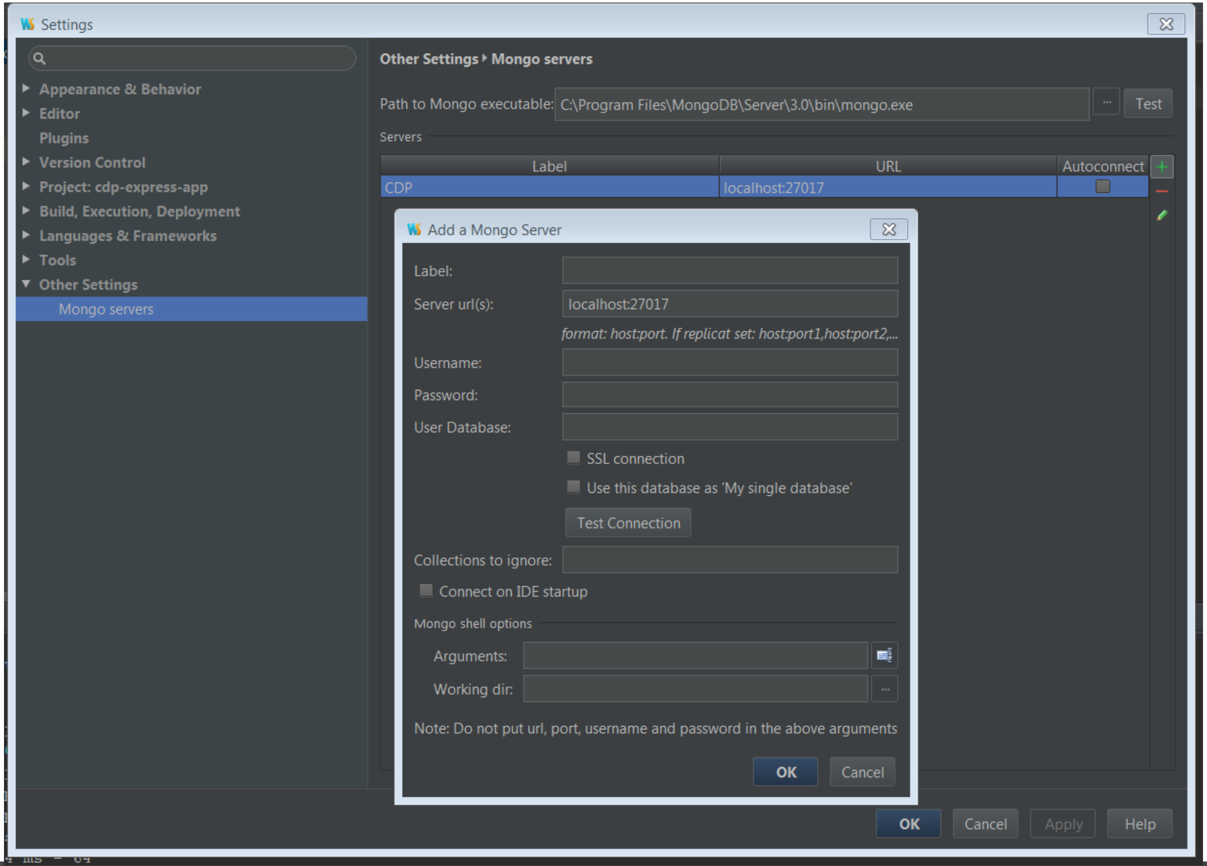
```
MongoDB shell version: 3.0.3
connecting to: test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
> db.users.save({id: "1", name: "Eugene"})
WriteResult({ "nInserted" : 1 })
> db.users.find()
{ "_id" : ObjectId("556a28384a51cbc960ef7e86"), "id" : "1", "name" : "Eugene" }
>
```

MONGO GUI PLUGIN FOR JETBRAINS PRODUCTS

Installation:

https://plugins.jetbrains.com/plugin/7141

MONGODB GUI PLUGIN CONFIGURATION



Other Settings ▸ Mongo servers

Path to Mongo executable: C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe

Servers

Label	URL	Autoconnect
CDP	localhost:27017	☐

Add a Mongo Server

Label:

Server url(s): localhost:27017

format: host:port. If replicat set: host:port1,host:port2,...

Username:

Password:

User Database:

☐ SSL connection

☐ Use this database as 'My single database'

Test Connection

Collections to ignore:

☐ Connect on IDE startup

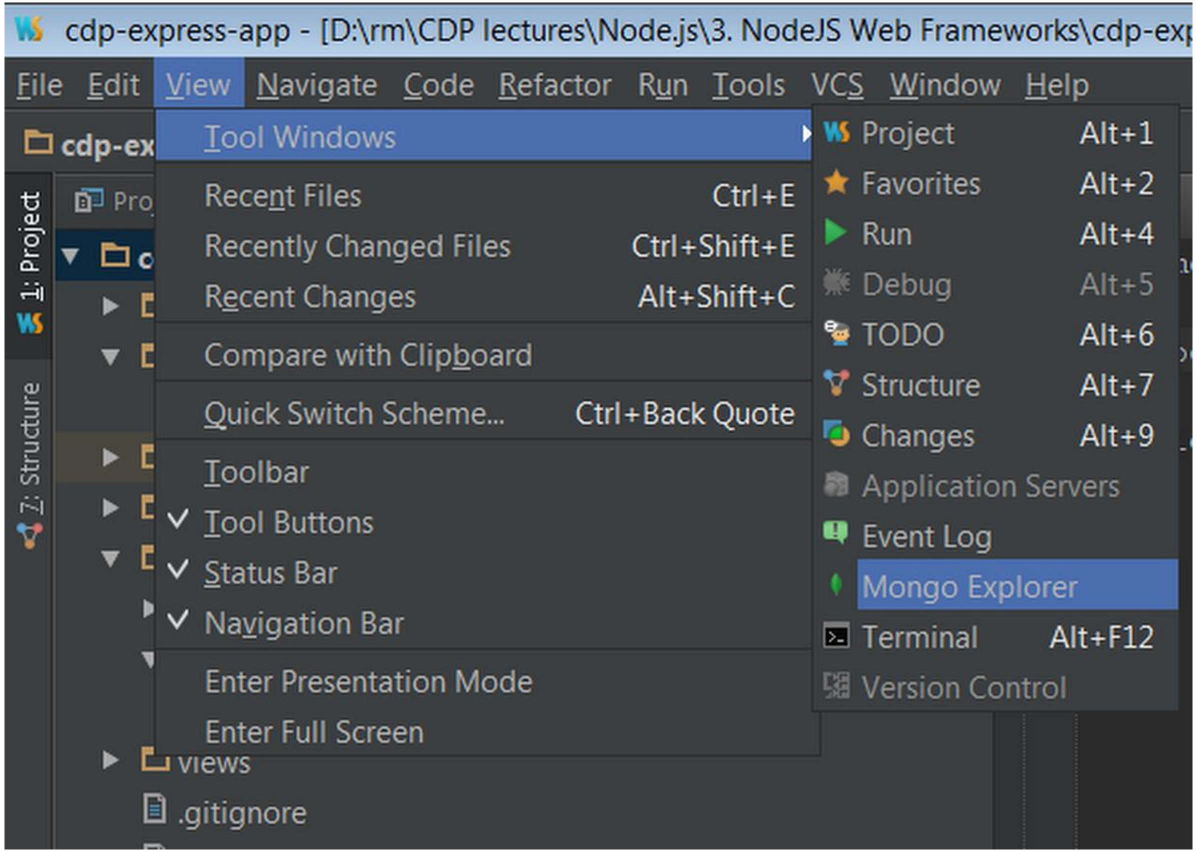
Mongo shell options

Arguments:

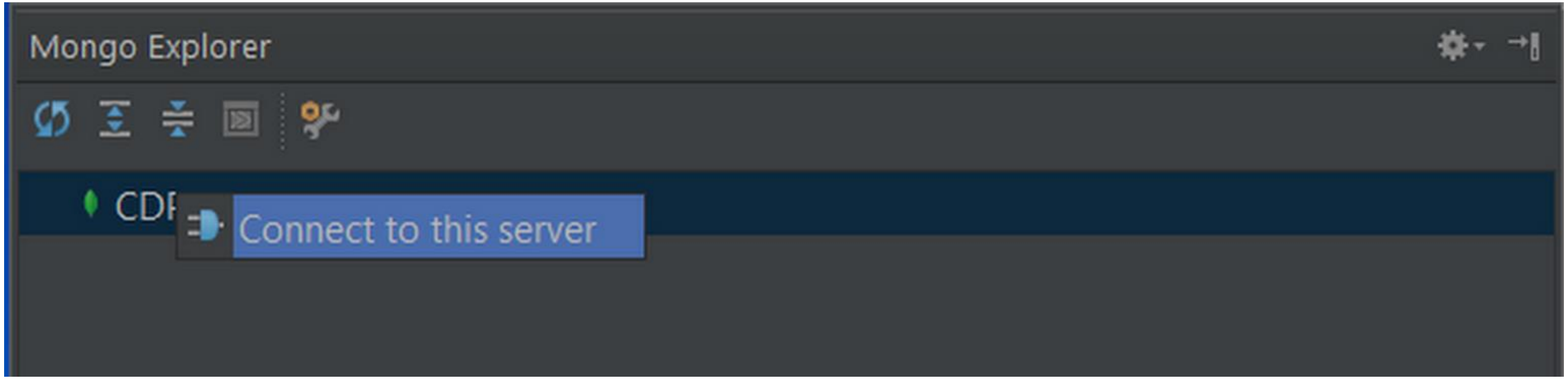
Working dir:

Note: Do not put url, port, username and password in the above arguments

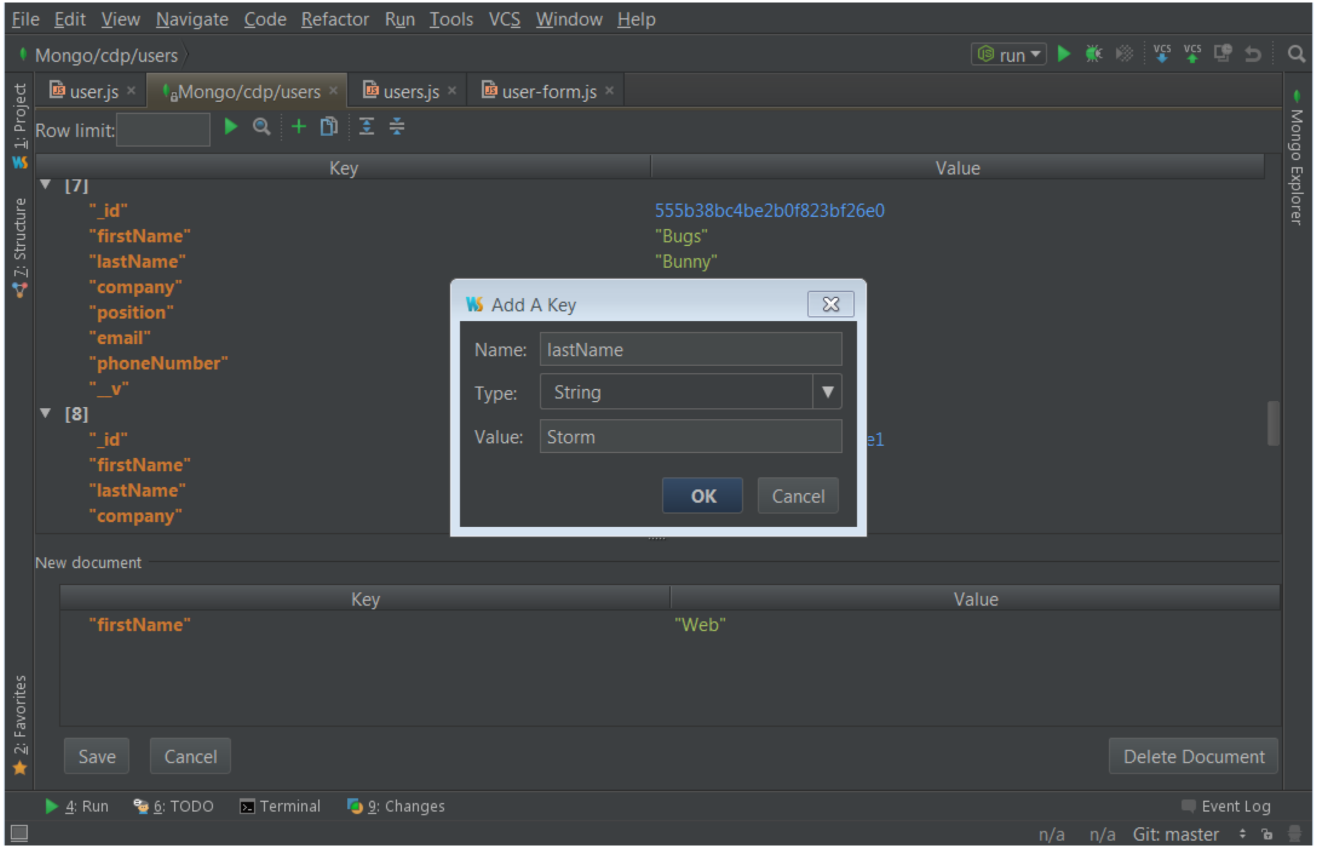
RUNNING MONGO GUI PLUGIN



CONNECTING MONGO GUI PLUGIN TO DB



EDITING DATA THROUGH MONGO GUI PLUGIN



Row limit:

Key	Value
[7]	
"_id"	555b38bc4be2b0f823bf26e0
"firstName"	"Bugs"
"lastName"	"Bunny"
"company"	
"position"	
"email"	
"phoneNumber"	
"__v"	
[8]	
"_id"	e1
"firstName"	
"lastName"	
"company"	

Add A Key

Name: lastName

Type: String

Value: Storm

OK Cancel

New document

Key	Value
"firstName"	"Web"

Save Cancel Delete Document

MONGODB NATIVE DRIVER

MongoDB Native Driver

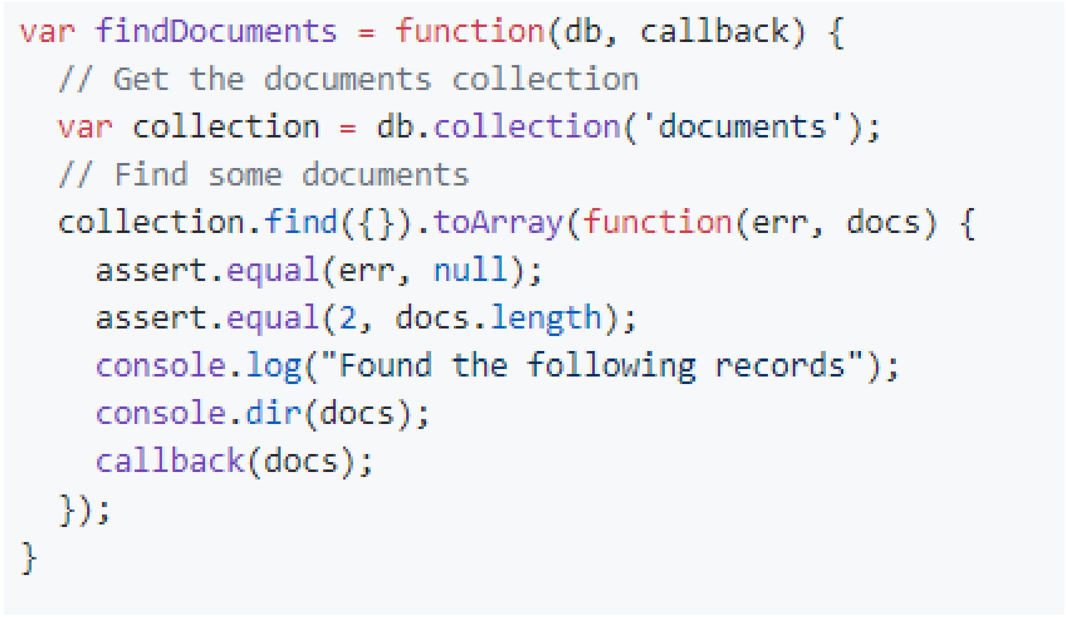
Getting started

```
var MongoClient = require('mongodb').MongoClient,
    assert = require('assert');

// Connection URL
var url = 'mongodb://localhost:27017/myproject‘;

// Use connect method to connect to the Server
MongoClient.connect(url, function(err, db) {
    assert.equal(null, err);
    console.log("Connected correctly to server");
    db.close();
});
```

MongoDB Native Driver

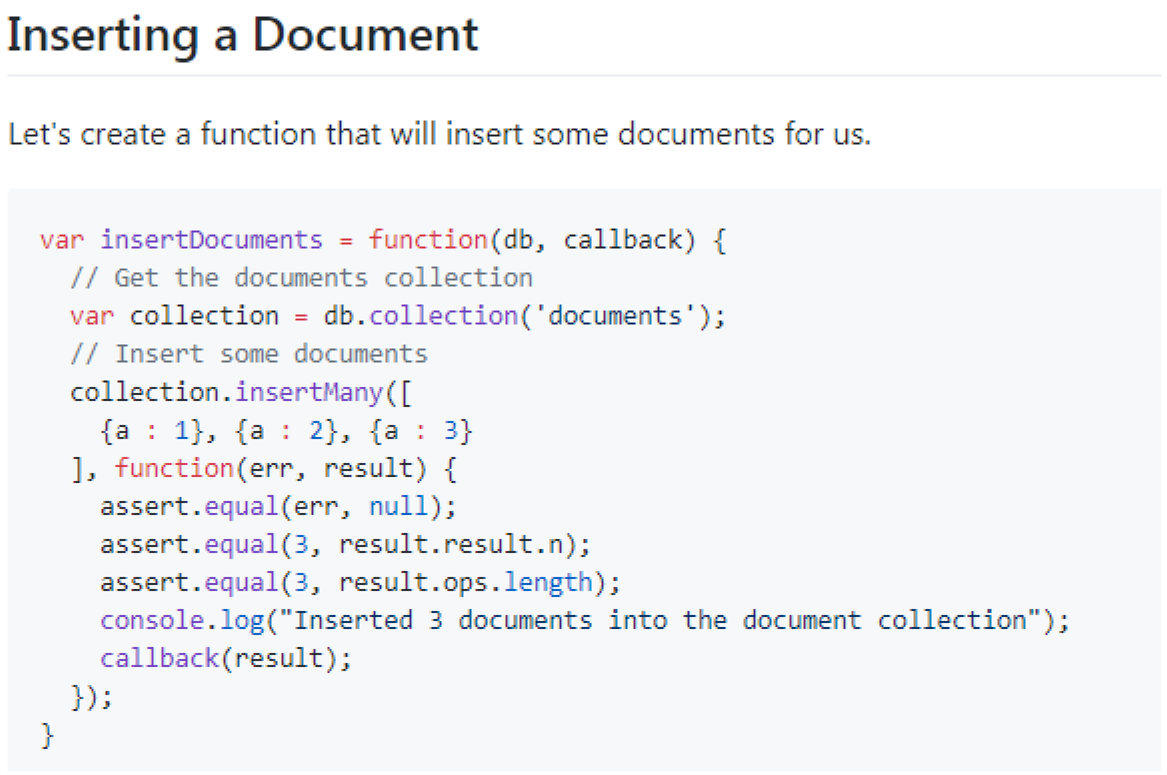
Find All Documents

a simple query that returns all the documents matching the query.



```
var findDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Find some documents
  collection.find({}).toArray(function(err, docs) {
    assert.equal(err, null);
    assert.equal(2, docs.length);
    console.log("Found the following records");
    console.dir(docs);
    callback(docs);
  });
}
```

MongoDB Native Driver



Inserting a Document

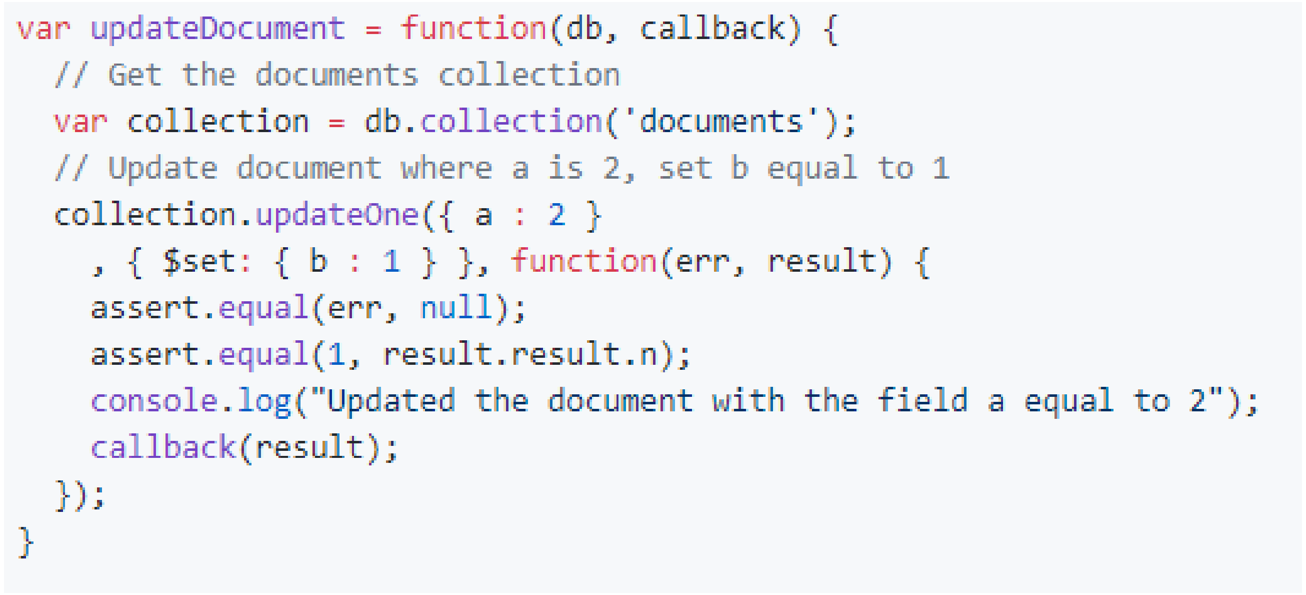
Let's create a function that will insert some documents for us.

```
var insertDocuments = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Insert some documents
  collection.insertMany([
    {a : 1}, {a : 2}, {a : 3}
  ], function(err, result) {
    assert.equal(err, null);
    assert.equal(3, result.result.n);
    assert.equal(3, result.ops.length);
    console.log("Inserted 3 documents into the document collection");
    callback(result);
  });
}
```

MongoDB Native Driver

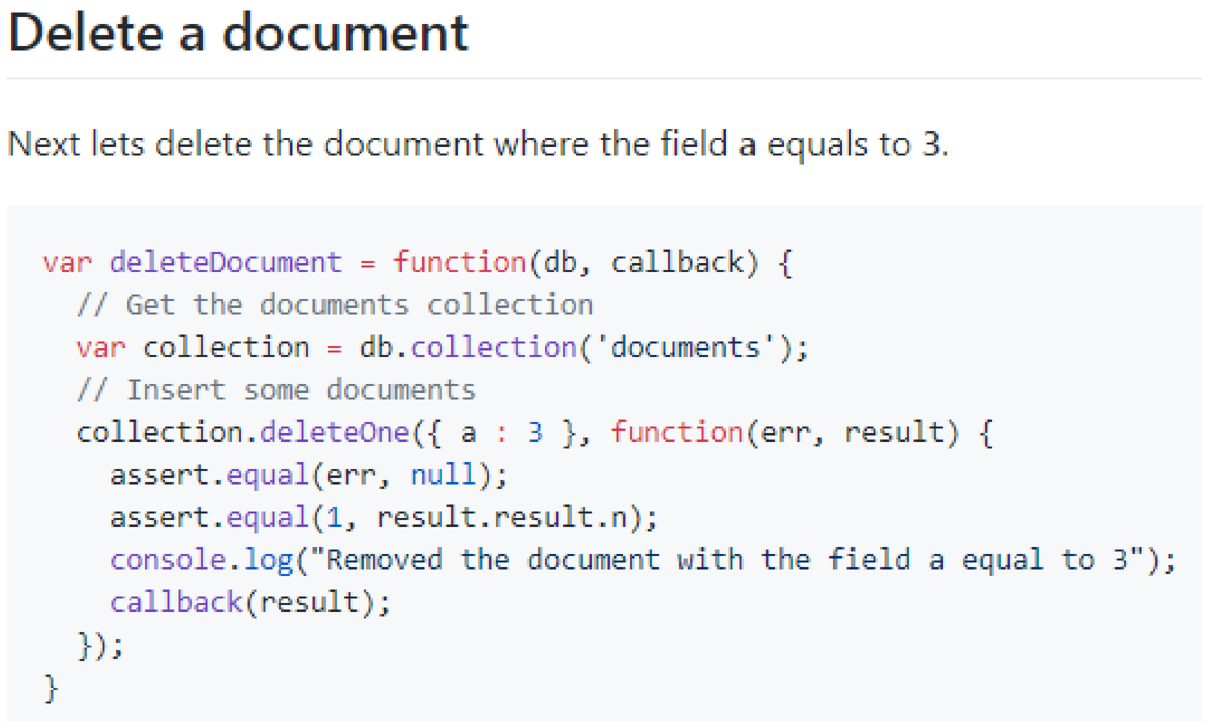
Updating a document

Let's look at how to do a simple document update by adding
a new field **b** to the document that has the field **a** set to **2**.



```
var updateDocument = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Update document where a is 2, set b equal to 1
  collection.updateOne({ a : 2 }
    , { $set: { b : 1 } }, function(err, result) {
    assert.equal(err, null);
    assert.equal(1, result.result.n);
    console.log("Updated the document with the field a equal to 2");
    callback(result);
  });
}
```

MongoDB Native Driver



Delete a document

Next lets delete the document where the field a equals to 3.

```
var deleteDocument = function(db, callback) {
  // Get the documents collection
  var collection = db.collection('documents');
  // Insert some documents
  collection.deleteOne({ a : 3 }, function(err, result) {
    assert.equal(err, null);
    assert.equal(1, result.result.n);
    console.log("Removed the document with the field a equal to 3");
    callback(result);
  });
}
```

ODM MONGOOSE

WHAT IS MONGOOSE?

mongoose

elegant mongodb object modeling for node.js

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

MONGOOSE FUNDAMENTALS

Getting started

The first thing we need to do is include mongoose in our project and open a connection

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
```

MONGOOSE FUNDAMENTALS

Getting started

We have a pending connection to the test database running on localhost. We now need to get notified if we connect successfully or if a connection error

```
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection
error:'));
db.once('open', function() {
    // we're connected!
});
```

MONGOOSE FUNDAMENTALS

With Mongoose, everything is derived from a **Schema**. Let's get a reference to it and define our kittens.occurs:

```
var kittySchema = mongoose.Schema({
    name: String
});
```

So far so good. We've got a schema with one property, name, which will be a String. The next step is compiling our schema into a Model.

```
var Kitten = mongoose.model('Kitten', kittySchema);
```

MONGOOSE FUNDAMENTALS

A model is a class with which we construct documents. In this case, each document will be a kitten with properties and behaviors as declared in our schema. Let's create a kitten document representing the little guy we just met on the sidewalk outside:

```
var silence = new Kitten({ name: 'Silence' });
console.log(silence.name); // 'Silence'
```

MONGOOSE FUNDAMENTALS

Kittens can meow, so let's take a look at how to add "speak" functionality to our documents:

```
// NOTE: methods must be added to the schema before compiling
it with mongoose.model()
kittySchema.methods.speak = function () {
    var greeting = this.name
        ? "My name is " + this.name
        : "I don't have a name";
    console.log(greeting);
}
var Kitten = mongoose.model('Kitten', kittySchema);
```

MONGOOSE FUNDAMENTALS

Functions added to the methods property of a schema get compiled into the Model prototype and exposed on each document instance:

```
var fluffy = new Kitten({ name: 'fluffy' });
fluffy.speak(); // "My name is fluffy"
```

MONGOOSE FUNDAMENTALS

Each document can be saved to the database by calling its save method. The first argument to the callback will be an error if any occured.

```
fluffy.save(function (err, fluffy) {
    if (err) return console.error(err);
    fluffy.speak();
});
```

We can access all of the kitten documents through our Kitten model.

```
Kitten.find(function (err, kittens) {
    if (err) return console.error(err);
    console.log(kittens);
});
Kitten.find({ name: /^fluff/ }, callback);
```

MONGOOSE FUNDAMENTALS

Using Query

```
var Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost',
// selecting the `name` and `occupation` fields
Person.findOne({ 'name.last': 'Ghost' }, 'name occupation', function (err, person) {
    if (err) return handleError(err);
    // Space Ghost is a talk show host.
    console.log('%s %s is a %s.', person.name.first, person.name.last, person.occupation)
})
```

MONGOOSE FUNDAMENTALS

Using Query without callback:

```
var Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost'
var query = Person.findOne({ 'name.last': 'Ghost' });

// selecting the `name` and `occupation` fields
query.select('name occupation');

// execute the query at a later time
query.exec(function (err, person) {
    if (err) return handleError(err);
    // Space Ghost is a talk show host.
    console.log('%s %s is a %s.', person.name.first,
 person.name.last, person.occupation)
})
```

MONGOOSE FUNDAMENTALS

Using Query: build up a query using chaining syntax

```
// With a JSON doc
Person.find({
    occupation: /host/,
    'name.last': 'Ghost',
    age: {$gt: 17, $lt: 66},
    likes: {$in: ['vaporizing', 'talking']}
}).limit(10)
  .sort({occupation: -1})
  .select({name: 1, occupation: 1})
  .exec(callback);
```

MONGOOSE FUNDAMENTALS

Using Query: build up a query using chaining syntax

```
// Using query builder
Person
    .find({occupation: /host/})
    .where('name.last').equals('Ghost')
    .where('age').gt(17).lt(66)
    .where('likes').in(['vaporizing', 'talking'])
    .limit(10)
    .sort('-occupation')
    .select('name occupation')
    .exec(callback);
```

MONGOOSE FUNDAMENTALS

Built-in Validators

```
var breakfastSchema = new Schema({
    eggs: {
        type: Number,
        min: [6, 'Too few eggs'],
        max: 12
    },
    bacon: {
        type: Number,
        required: [true, 'Why no bacon?']
    },
    drink: {
        type: String,
        enum: ['Coffee', 'Tea']
    }
});
var Breakfast = db.model('Breakfast', breakfastSchema);
```

MONGOOSE FUNDAMENTALS

Custom Validators

```
var userSchema = new Schema({
    phone: {
        type: String,
        validate: {
            validator: function(v) {
                return /\d{3}-\d{3}-\d{4}/.test(v);
            },
            message: '{VALUE} is not a valid phone number!'
        },
        required: [true, 'User phone number required']
    }
});
```

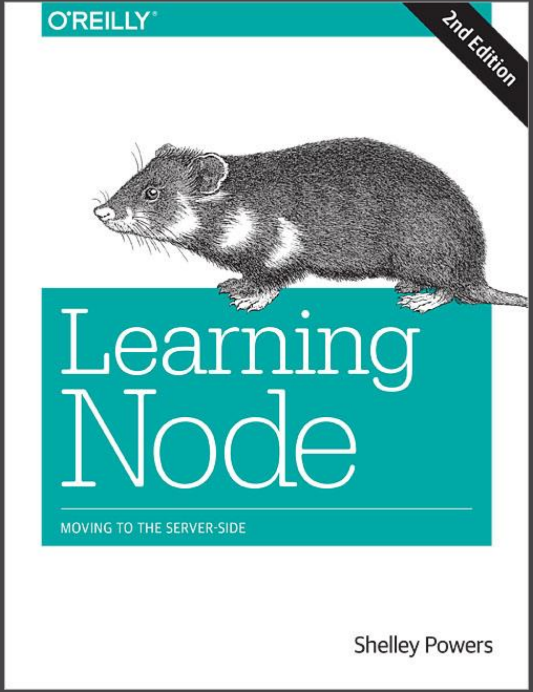
MONGOOSE FUNDAMENTALS

Middleware

```
var schema = new Schema(..);
schema.pre('save', function(next) {
    // do stuff
    next();
});
```

CONCLUSION

WHAT TO READ

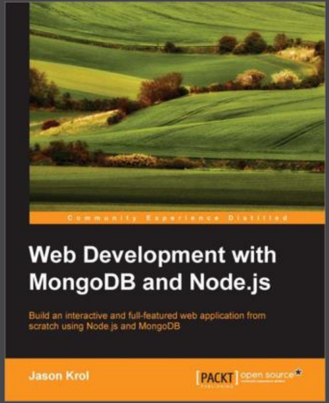

O'REILLY

2nd Edition

Learning Node

MOVING TO THE SERVER-SIDE

Shelley Powers

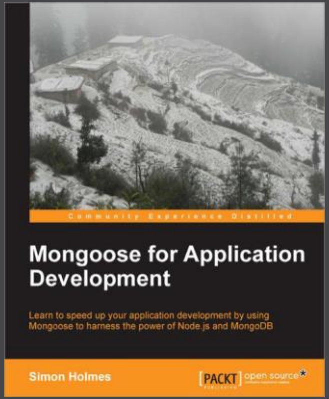

Community Experience Distilled

Web Development with MongoDB and Node.js

Build an interactive and full-featured web application from scratch using Node.js and MongoDB

Jason Krol

[PACKT] open source

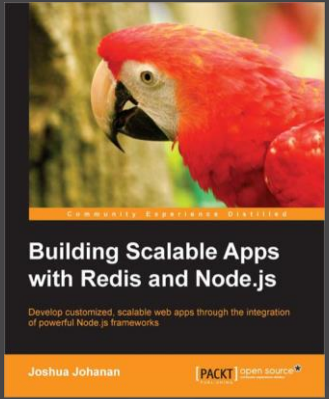

Community Experience Distilled

Mongoose for Application Development

Learn to speed up your application development by using Mongoose to harness the power of Node.js and MongoDB

Simon Holmes

[PACKT] open source


Community Experience Distilled

Building Scalable Apps with Redis and Node.js

Develop customized, scalable web apps through the integration of powerful Node.js frameworks

Joshua Johanan

[PACKT] open source

And others

FAQ

F. A. Q. - ?

THANKS!

Andrii Kochkin

JUNE 25, 2018