MongoDB

07

Presented by
Subhash EP

Subhash

mongoDB

{ name: mongo, type: DB }

Subhash

What datatype is this?

	BIKE	COUNTER	LOVESMONGO	LOVESSQL	NAME	TS	WIFE
1	Felt	1	true	true	Marc	{ts '2010-10-20 18:12:21'}	Heather
2	Trek	2	true	false	Steve	{ts '2010-10-20 18:12:21'}	Paige
3	Giant	3	false	false	Dan	{ts '2010-10-20 18:12:21'}	Katie

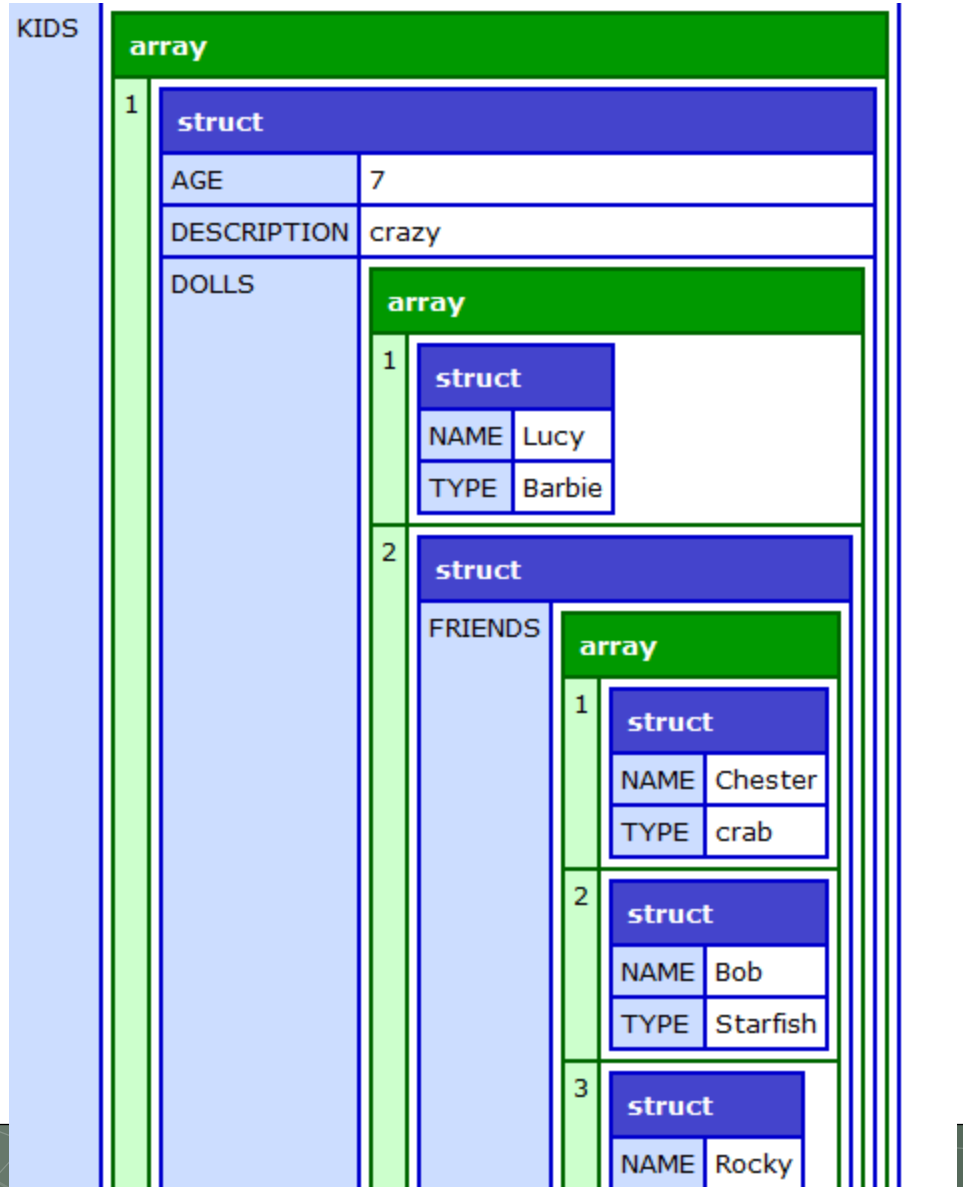
And This?

BIKE	Felt
COUNTER	1
LOVESMONGO	true
LOVESSQL	true
NAME	Marc
TS	{ts '2010-10-20 21:03:45'}
WIFE	Heather

And This?

	struct	
1	BIKE	Specialized
	COUNTER	1
	KIDS	
	NAME	Cool Dude 1
	TS	{ts '2010-10-20 18:20:01'}
	WIFE	Smokin hot wife 1
2	BIKE	Specialized
	COUNTER	2
	KIDS	
	NAME	Cool Dude 2
	TS	{ts '2010-10-20 18:20:01'}
	WIFE	Smokin hot wife 2
3	BIKE	Specialized
	COUNTER	3
	KIDS	
	NAME	Cool Dude 3

And This?

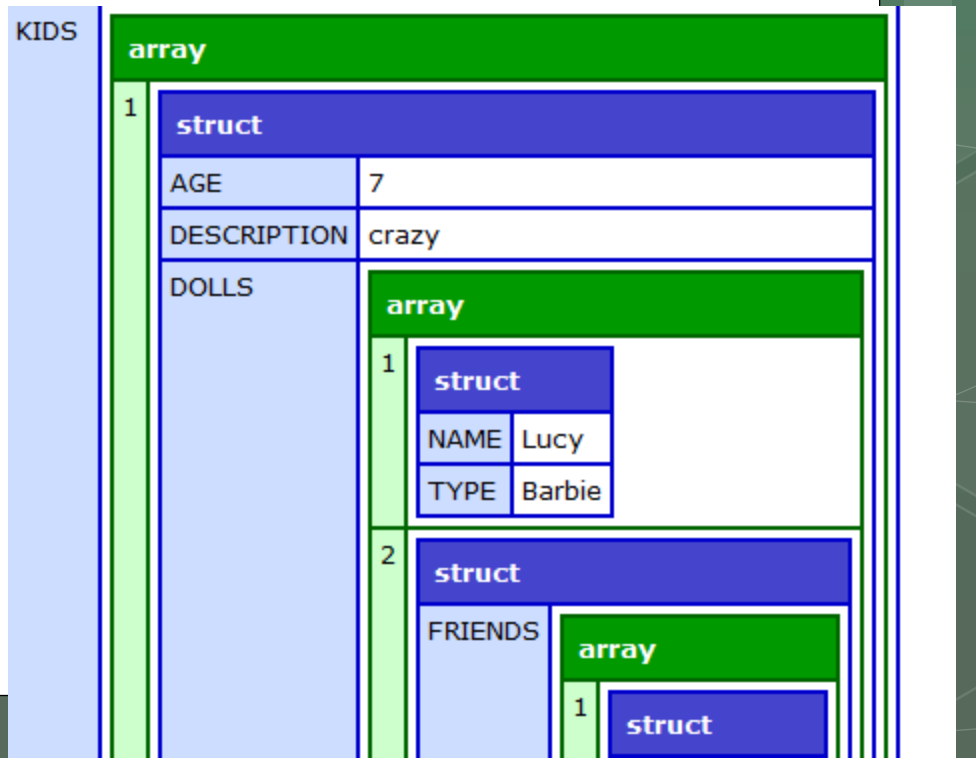

KIDS
array
1
struct
AGE
7
DESCRIPTION
crazy
DOLLS
array
1
struct
NAME
Lucy
TYPE
Barbie
2
struct
FRIENDS
array
1
struct
NAME
Chester
TYPE
crab
2
struct
NAME
Bob
TYPE
Starfish
3
struct
NAME
Rocky

And the first picture is different from that last one how?

	BIKE	COUNTER	LOVESMONGO	LOVESSQL	NAME	TS	WIFE
1	Felt	1	true	true	Marc	{ts '2010-10-20 18:12:21'}	Heather
2	Trek	2	true	false	Steve	{ts '2010-10-20 18:12:21'}	Paige
3	Giant	3	false	false	Dan	{ts '2010-10-20 18:12:21'}	Katie

VS

.



Mindshift

From Queries

To Arrays of Structures

MongoDB is…

- A Schema-less, Document-Oriented datastore
- Documents are roughly equivalent to KV structs:
  - Keys, values
  - Values can be other documents (i.e. embedded)
- Documents are **searchable**

MongoDB's Data Model

- A Database has "Collections"
  - Collections have "Documents"
    - Documents have "Fields"
      - Fields are key = value pairs

- A Collection does not enforce the structure of its documents*

*i.e. Schemaless

Install and Run MongoDB

Download from mongodb.org

Unzip

Create data directory

>mkdir c:\data\db

Run MongoDB (mongod):

>cd c:\mongodb-1.6.3\bin

>mongod

Run Mongo shell (mongo):

>mongo

Yes, that's it

The Mongo Shell

>mongo

>help()

>show dbs

>use <dbname>

>show collections

>db.collectionName.findOne()

>db.collectionName.find()

>db.help()

>db.collectionName.help()

Insert

```
db.people.insert( {NAME: "Steve",  BIKE: "Trek", WIFE: "Paige", TS: new Date() } )

db.people.findOne( {NAME: "Steve"} )
{
        "_id" : ObjectId("4cbcefb22d710000000051cc"),
        "NAME" : "Steve",
        "BIKE" : "Trek",
        "WIFE" : "Paige",
        "TS" : "Mon Oct 18 2010 21:09:06 GMT-0400 (Eastern Daylight Time)"
}
```

Update

```
db.people.update( {NAME: "Steve"}, { $set: { DOG:  "Apollo"  }  })

db.people.findOne( {NAME: "Steve"} )
{
        "BIKE" : "Trek",
        "DOG" : "Apollo",
        "NAME" : "Steve",
        "TS" : "Mon Oct 18 2010 21:09:06 GMT-0400 (Eastern Daylight Time)",
        "WIFE" : "Paige",
        "_id" : ObjectId("4cbcefb22d710000000051cc")
}
```

```
db.people.update( {NAME: "Steve"}, { $unset: { DOG: true  }  })

db.people.findOne( {NAME: "Steve"} )
{
        "BIKE" : "Trek",
        "NAME" : "Steve",
        "TS" : "Mon Oct 18 2010 21:09:06 GMT-0400 (Eastern Daylight Time)",
        "WIFE" : "Paige",
        "_id" : ObjectId("4cbcefb22d710000000051cc")
}
```

Updating Components

```
> db.Components.find();
{ "_id" : 1, "SerialNumber" : "1-002" }
> db.Components.update({"_id":1}, {"$set":{"ModelNumber":"Model1"}});
> db.Components.find();
{ "ModelNumber" : "Model1", "SerialNumber" : "1-002", "_id" : 1 }
>
```

- $set keyword used for partial updates
- Without $set keyword entire document is replaced
- {multi : true} to update multiple documents

```
$set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $pop,
$addToSet, $rename, $bit, $ positional operator
```

mongoDB

Remove

```
>
> db.people.remove( {NAME: "Steve"} )
>
> db.people.findOne( {NAME: "Steve"} )
null
>
```

Deleting Components

```
db.Components.remove();
```

Works like .find()

```
db.Components.drop();
```

Drops collection

```
db.dropDatabase();
```

Drops database

Finding by nested fields

```
> db.people.insert( {NAME: "Bob", KIDS: [ {NAME: "Sally", AGE: 3}, {NAME: "Billy", AGE: 7} ] })
```

Finding by nested fields

```
> db.people.insert( {NAME: "Bob", KIDS: [ {NAME: "Sally", AGE: 3}, {NAME: "Billy", AGE: 7} ] })
>
> db.people.findOne({NAME: "Bob"}
... )
{
        "_id" : ObjectId("4cbcf1632d710000000051cf"),
        "NAME" : "Bob",
        "KIDS" : [
                {
                        "NAME" : "Sally",
                        "AGE" : 3
                },
                {
                        "NAME" : "Billy",
                        "AGE" : 7
                }
        ]
}
```

Finding by nested fields

```
> db.people.insert( {NAME: "Bob", KIDS: [ {NAME: "Sally", AGE: 3}, {NAME: "Billy", AGE: 7} ] })
> db.people.findOne({ "KIDS.NAME": "Billy"})
{
        "_id" : ObjectId("4cbcf1632d710000000051cf"),
        "NAME" : "Bob",
        "KIDS" : [
                {
                        "NAME" : "Sally",
                        "AGE" : 3
                },
                {
                        "NAME" : "Billy",
                        "AGE" : 7
                }
        ]
}
>
```

Finding with Conditionals

```
> db.people.insert( {NAME: "Shaggy", KIDS: [ {NAME: "Scrappy", AGE: 3}, {NAME: "Scooby", AGE: 5} ] })
> db.people.insert( {NAME: "Fred", KIDS: [ {NAME: "Biff", AGE: 13}, {NAME: "Muffy", AGE: 10} ] })
> db.people.insert( {NAME: "Thelma", KIDS: [ {NAME: "Simon", AGE: 4}, {NAME: "Stella", AGE: 2} ] })
>
```

Finding with Conditionals

```
> db.people.insert( {NAME: "Shaggy", KIDS: [ {NAME: "Scrappy", AGE: 3}, {NAME: "Scooby", AGE: 5} ] })
> db.people.insert( {NAME: "Fred", KIDS: [ {NAME: "Biff", AGE: 13}, {NAME: "Muffy", AGE: 10} ] })
> db.people.insert( {NAME: "Thelma", KIDS: [ {NAME: "Simon", AGE: 4}, {NAME: "Stella", AGE: 2} ] })
>
> db.people.find({ "KIDS.AGE" : { $in: [3,4] } })
{ "_id" : ObjectId("4cbcf3f92d7100000000051d3"), "NAME" : "Shaggy", "KIDS" : [
        {
                "NAME" : "Scrappy",
                "AGE" : 3
        },
        {
                "NAME" : "Scooby",
                "AGE" : 5
        }
] }
{ "_id" : ObjectId("4cbcf4012d7100000000051d5"), "NAME" : "Thelma", "KIDS" : [
        {
                "NAME" : "Simon",
                "AGE" : 4
        },
        {
                "NAME" : "Stella",
                "AGE" : 2
        }
] }
>
```

What happens if I search on a field a Document doesn't have?

```
db.people.insert( {NAME: "Daphne", SWINGER: true })

db.people.find({ "KIDS.AGE" : { $in: [3,4] }  })
```

What happens if I search on a field a Document doesn't have?

```
> db.people.insert( {NAME: "Daphne", SWINGER: true })
> 
> db.people.find({ "KIDS.AGE" : { $in: [3,4] }  })
{ "_id" : ObjectId("4cbcf3f92d7100000000051d3"), "NAME" : "Shaggy", "KIDS" : [
        {
                "NAME" : "Scrappy",
                "AGE" : 3
        },
        {
                "NAME" : "Scooby",
                "AGE" : 5
        }
] }
{ "_id" : ObjectId("4cbcf4012d7100000000051d5"), "NAME" : "Thelma", "KIDS" : [
        {
                "NAME" : "Simon",
                "AGE" : 4
        },
        {
                "NAME" : "Stella",
                "AGE" : 2
        }
] }
>
```

What happens if I search on a field a Document doesn't have?

```
> 
> 
> db.people.find({SWINGER: true})
{ "_id" : ObjectId("4cbcf5392d7100000000051d6"), "NAME" : "Daphne", "SWINGER" : true }
> 
```

Can I Find Documents without XXX?

```
> db.people.find({KIDS: {$exists: false} })
{ "_id" : ObjectId("4cce10cbff350000000034a2"), "NAME" : "Daphne", "SWINGER" : true }
>
```

Sorting? Limiting? Skipping?

Have you ever had to "Page" in a web app?


```
>
> db.people.insert( {NAME: "Thelma", AGE: 25  })
> db.people.insert( {NAME: "Daphne", AGE: 33  })
> db.people.insert( {NAME: "Shaggy", AGE: 19  })
> db.people.insert( {NAME: "Scrappy", AGE: 3  })
>
```

Sorting?

```
> db.people.insert( {NAME: "Thelma", AGE: 25  })
> db.people.insert( {NAME: "Daphne", AGE: 33  })
> db.people.insert( {NAME: "Shaggy", AGE: 19  })
> db.people.insert( {NAME: "Scrappy", AGE: 3  })
>
>
> db.people.find().sort({AGE: -1})
{ "_id" : ObjectId("4cbcf8042d710000000051d8"), "NAME" : "Daphne", "AGE" : 33 }
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }
{ "_id" : ObjectId("4cbcf8262d710000000051d9"), "NAME" : "Shaggy", "AGE" : 19 }
{ "_id" : ObjectId("4cbcf82e2d710000000051da"), "NAME" : "Scrappy", "AGE" : 3 }
>
```

1 = true, 1 = asc, 1 = include

-1 = false, -1 = desc, -1 = exclude

(depends on context)

Sorting? Limiting?

```
> db.people.insert( {NAME: "Thelma", AGE: 25  })
> db.people.insert( {NAME: "Daphne", AGE: 33  })
> db.people.insert( {NAME: "Shaggy", AGE: 19  })
> db.people.insert( {NAME: "Scrappy", AGE: 3  })


> db.people.find().sort({AGE: -1})
{ "_id" : ObjectId("4cbcf8042d710000000051d8"), "NAME" : "Daphne", "AGE" : 33 }
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }
{ "_id" : ObjectId("4cbcf8262d710000000051d9"), "NAME" : "Shaggy", "AGE" : 19 }
{ "_id" : ObjectId("4cbcf82e2d710000000051da"), "NAME" : "Scrappy", "AGE" : 3 }

> db.people.find().sort({AGE: -1}).limit(2)
{ "_id" : ObjectId("4cbcf8042d710000000051d8"), "NAME" : "Daphne", "AGE" : 33 }
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }
>
```

sort(), skip(), limit()

Sorting + Limiting + Skipping =

```
> db.people.insert( {NAME: "Thelma", AGE: 25  })
> db.people.insert( {NAME: "Daphne", AGE: 33  })
> db.people.insert( {NAME: "Shaggy", AGE: 19  })
> db.people.insert( {NAME: "Scrappy", AGE: 3  })


> db.people.find().sort({AGE: -1})
{ "_id" : ObjectId("4cbcf8042d710000000051d8"), "NAME" : "Daphne", "AGE" : 33 }
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }
{ "_id" : ObjectId("4cbcf8262d710000000051d9"), "NAME" : "Shaggy", "AGE" : 19 }
{ "_id" : ObjectId("4cbcf82e2d710000000051da"), "NAME" : "Scrappy", "AGE" : 3 }

> db.people.find().sort({AGE: -1}).limit(2)
{ "_id" : ObjectId("4cbcf8042d710000000051d8"), "NAME" : "Daphne", "AGE" : 33 }
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }


> db.people.find().sort({AGE: -1}).skip(1).limit(2)
{ "_id" : ObjectId("4cbcf7f12d710000000051d7"), "NAME" : "Thelma", "AGE" : 25 }
{ "_id" : ObjectId("4cbcf8262d710000000051d9"), "NAME" : "Shaggy", "AGE" : 19 }
>
```

sort(), skip(), limit()

count() and distinct()

```
> db.people.insert({NAME: 'Shaggy', AGE: 25})
> db.people.insert({NAME: 'Fred', AGE: 28})
> db.people.insert({NAME: 'Daphne', AGE: 27})
> db.people.insert({NAME: 'Thelma', AGE: 24})
>
>
> db.people.count()
10
```

count() and distinct()

```
> db.people.insert({NAME: 'Shaggy', AGE: 25})
> db.people.insert({NAME: 'Fred', AGE: 28})
> db.people.insert({NAME: 'Daphne', AGE: 27})
> db.people.insert({NAME: 'Thelma', AGE: 24})
>
>
> db.people.count()
10
>
> db.people.count({AGE: {$gt: 24}})
3
>
```

count() and distinct()

```
> db.people.insert({NAME: 'Shaggy', AGE: 25})
> db.people.insert({NAME: 'Fred', AGE: 28})
> db.people.insert({NAME: 'Daphne', AGE: 27})
> db.people.insert({NAME: 'Thelma', AGE: 24})
>
>
> db.people.count()
10
>
> db.people.count({AGE: {$gt: 24}})
3
>
> db.people.distinct( 'AGE' )
[ 24, 25, 27, 28 ]
```

count() and distinct()

```
> db.people.insert({NAME: 'Shaggy', AGE: 25})
> db.people.insert({NAME: 'Fred', AGE: 28})
> db.people.insert({NAME: 'Daphne', AGE: 27})
> db.people.insert({NAME: 'Thelma', AGE: 24})
>
>
> db.people.count()
10
>
> db.people.count({AGE: {$gt: 24}})
3
>
> db.people.distinct( 'AGE' )
[ 24, 25, 27, 28 ]
>
> db.people.distinct( 'NAME' )
[
        "Cool Dude 1",
        "Cool Dude 2",
        "Cool Dude 3",
        "Cool Dude 4",
        "Cool Dude 5",
        "Daphne",
        "Fred",
        "Marc",
        "Shaggy",
        "Thelma"
]
>
```

Schema Design Guideline

Embed when you can:

```
{ NAME = “Marc”, KIDS = [{NAME=“Lexie”},  {NAME=“Sidney”}] }
```

Relate when you must:

```
{ NAME = “Marc”,
   KIDS = [
         ObjectId(4cd0bcc754503766b7dcfd6a),
         ObjectId(4cd0bcc754503766b7dcfd7a)
   ]
 }
```

Why MongoDB: The Official Word

MongoDB is Big and Fast

(just like every other NoSQL Datastore)

Why MongoDB: Performance

- No Joins + No multi-row transactions
  - = Fast Reads
  - = Fast Writes (b/c you write to fewer tables, no trans. log)
- Async writes
  - = you don't wait for inserts to complete
  - (optional, though)
- Secondary Indexes
  - = Index on embedded document fields for superfast ad-hoc queries
  - Indexes live in RAM

Why MongoDB: Scalability / Availability

- No Joins + No multi-row transactions
  - = Horizontally Scalable
  - = Built for Distribution of data and computation
    - Sharding for distributed data
    - MapReduce for distributed computation
- Built-in Replication
  - Easy to configure
  - Reads can be distributed across slaves
  - Automatic failover when using Replica Sets

Shard for scalability, **replicate** for availability

Why MongoDB: R.A.D.

- Documents (think: objects) with embedded documents feel more natural compared with DB / ORM
- Ad-hoc queries don't require "view" creation or other config… code-n-go
- No penalty for schema evolution (true for most NoSQL stores)
- Atomic Modifiers for safe in-place updates (no locking code around single documents)
- Built-in functions for Mongo-as-a-queue (I LOVE these!)

Why MongoDB: Friendly License

- You can use MongoDB in your corporate app
- You do not have to open source your corporate app

- From the docs:

" To reiterate, you **only** need to provide the source for the MongoDB server and not your application"

http://www.mongodb.org/display/DOCS/Licensing

Why Not MongoDB? Vs. Itself

- Still kinda new
  - Though used in production at major sites
- Evolving rapidly
  - This is a good thing, though you'll want to keep up with new releases
- Not meant for 32Bit
  - You'll have a 2GB limit per mongod process if you do
- No single-server durability
  - If you're not running replication, and it crashes, you will lose data (they're working on this)
  - Just replicate, period

Awesome: Query Operators

$ne:

db.people.find({ NAME: { $ne: “Shaggy”} })

$nin:

db.people.find({ NAME: { $nin: [“Shaggy”, “Daphne”] } })

$gt and $lt:

db.people.find(AGE: { {$gt: 30, $lt: 35} })

$size (eg, people with exactly 3 kids):

db.people.find(KIDS: {$size: 3})

regex: (eg, names starting with Ma or Mi)

db.people.find({NAME: /^M(a|i)/})

Awesome: Atomic Modifiers

$inc

```
db.pageviews.update( {URL: 'http://myurl.com'}, {$inc: {N: 1}} )
```

$set

```
db.people.update( {NAME: 'Steve'}, {$set: {Age: 35}} )
```

$push (for atomically adding values to an array)

```
db.people.update( {NAME: 'Steve'}, {$push: {KIDS: {NAME: 'Sylvia', AGE: 3}}})
```

findAndModify()

```
db.tasks.findAndModify(
        query: {STATUS: 'pending'},
        sort: {PRIORITY: -1},
        update: {$set: {STATUS: 'running', TS: new Date()}}
)
```

Awesome: ObjectId

- BSON (binary-encoded JSON)
- Unique: combination of
  - Time
  - Machine
  - Process id
  - Incrementer
- Mongo will automatically add this _id field

That's all

End of Session

©

subhash.ep@gmail.com
linkedin.com/in/subhashep