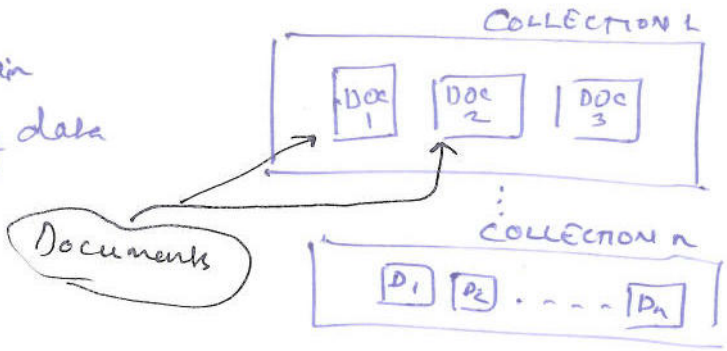
[18.07.18]

MONGO - DB — No SQL Database —

- No tables
- No relationships to maintain
- Documents are the driving data
- no schema



data/db → Data directory for mongodb
↳ is configurable ⇒ --dbpath <org>

#Start
* mongod → runs our mongo server
* mongo ↵ Starts our shell that connects to our server.

Configuration
→ data file in specific directory
→ Logging verbosity
→ Log file name & location

Running mongo as a service so that it runs on startup.

```
mongod -f C:\Pluralsight\
    mongod.conf --install ↵
net start MongoDB ↵
net start | findstr mongo ↵
net stop mongoDB
```

<mongod.conf> (need to give complete path)

```
dbpath = /Pluralsight/db
logpath = /Pluralsight/mongo-server.log
verbose = vvvvv   (1-5 ← least to most verbose)
```

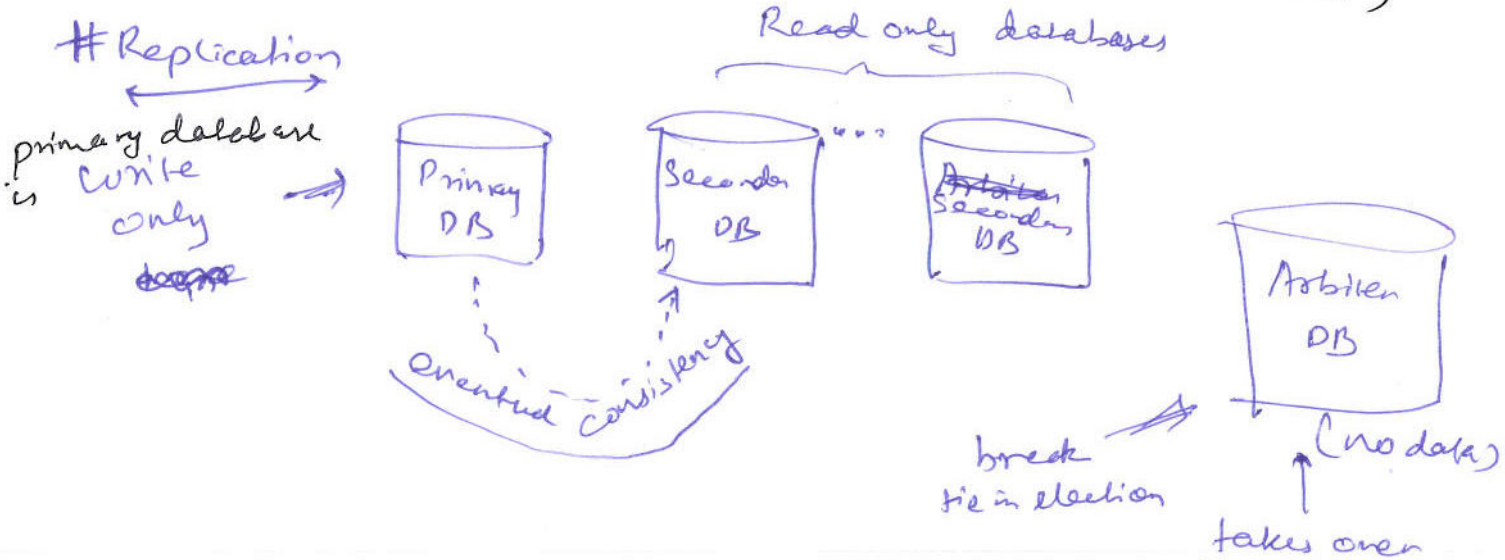
Using the above run the server

```
mongod -f C:\Pluralsight\mongod.conf
```

I$")
[illegible]

* Show dbs ↵ Show databases
* db ↵ Shows which database we are presene
* use foo ↵ Switched to database foo (If not present it will create one)

Replication



--port

--replSet "demo" ← name of the set

OR dbs path

Creates 3 replicas

Start "a" mongod --dbpath ./db1 --port 30000 --replSet "demo" ↵

Start "b" mongod --dbpath ./db2 --port 40000 --replSet "demo" ↵

Start "c" mongod --dbpath ./db3 --port 50000 --replSet "demo" ↵

~~dbs~~.

mongo --port 30000 ↵ ⇒ connects us to the server @ 30000

db.getMongo() ↵ connection to 127.0.0.1:30000

Configuring Replica set

```
var demoConfig = {
    "_id" : "demo",
    "members" : [
        { "_id":0, "host" : "localhost:30000",
          "priority" : 10 },
        { "_id" : 1, "host" : "loc:40000" },
        { "_id" : 2, "host" : "loc:50000",
          "arbiterOnly" : true }
    ]
```

1 Primary
1 Arbiter
2 Secondary

* Higher priority to get more votes.

rs.initiate(demoConfig)

(1) We are in the primary

db.foo.save({_id:1, value:'Hello World'})

Is object. (JSON)

Save a document to the collection

To switch to any db → mongo --port 40000

(2) Switch to Secondary to see if the data has been copied

mongo --port 40000 ↵ : switch to Secondary DB.

(now set slaveOk to be able to read the data)

db.setSlaveOk() ↵

db.foo.find() ↵

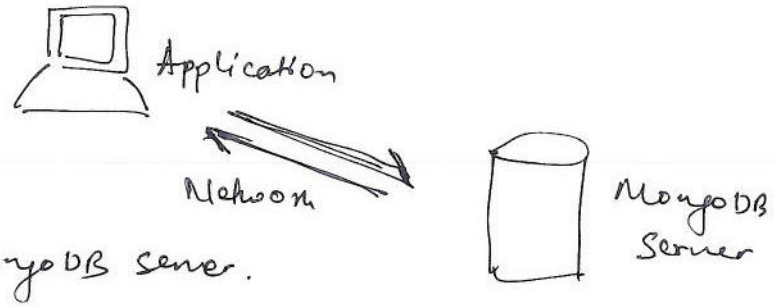
↳ will return us the object saved in 1° DB.

{ "_id" : 1, "value" : "hello world" }

rs.status()

20.07.18

Mongo Shell



→ It is a JS interpreter

→ Shell is a kind of app' that talks with the MongoDB server.

--shell → run script and return to shell.

To avoid dropping DB accidently in production

↳ drop <database> ⇒ do nothing

Safer.js

```
DB.prototype.dropDatabase = function () {
    print("Don't do anything");
}

db.dropDatabase = DB.prototype.dropDatabase;
```

mongo safer.js --shell ↵

db.dropDatabase() ↵

Don't do anything

* ctrl+L → clear screen.

.mongorc.js

Putting it here would not have to invoke it every time.

located in

users/(name)/.mongorc.js

imp ⇒ mongo --norc ↵

It will not run the .mongorc.js file

15.05.18

Mongo DB

Starting the Server

Set the dbpath

Goto the bin folder of mongodb

↳ mongod --dbpath = "<dbpath here>" ↵

It should starting in case the folder mentioned in the dbpath exists.

MongoDB Server → listens on the localhost 127.0.0.1

Starting the shell

Goto bin folder of the mongodb installed

↳ mongo ↵

Above command starts the shell used to connect to the mongodb server

Commands

Show dbs ↵ :- Displays the databases present.

Use myDatabase ↵ :- Creates a new database named "myDatabase"

Creating Collection

Collection will be a template for storing various information

To create collection we need to follow the below steps)

Use myDatabase ↵

db.employees.insert({ }) ← takes an object

employees → Name of the collection

Example → "name": "Suryakanna",
"age": 31,
"sex": "M"

To view the collection

generated by MongoDB (_id)

```
db.employees.find()
```

name of the collection

```
{ "_id" : ...
  "name" : "Suprataane",
  "age" : "32",
  "sex" : "M"
}
```

This o/p comes in a raw format.

This can be viewed in a pretty format

using → `db.employees.find().pretty()` : Output is as above

Adding document to collection

```
var employee1 = { };
employee1.name = "Bhanu";
employee1.age = 30;
employee1.sex = "M";
```

// Now need to add the employee1 to our collection "employees"

```
db.employees.save(employee1);
```

The above command adds the employee1 to our predefined collection 'employees'

Now, viewing employees will give two set of records.

```
db.employees.find().pretty();
```

```
{ 'name' : 'Suprataane'
  _id : ...
}
{ 'name' : 'Bhanu'
  _id : ...
}
```

[illegible] ≡ # Playing with Data

BSON → Binary JSON
Binary encoded serialization of JSON

Rules

#1 _id : must have an id
- Number : 1
- Decimal number : 3.14
- String : "Hello"
- Date : (ISODate())
- Complex object : {a:'x', b:2}

#2 No rules

⇒ 16Mb is the default size of the mongodb document.

| Cannot use array for ID |

⇒ If we don't provide mongo generates a unique ObjectID.
ObjectID has timestamp too.

```
ObjectID().getTimestamp() ↵
ISODate("2013-06-08T18:49:39Z")
```

→ If multiple documents are saved with same ID, then the latest one is saved.

insert()

```
db.bar.insert({_id:1, x:10}) ↵
db.bar.insert({_id:1, name:'bob'}) ↵
```

vs

db.bar. ↳ Duplicate key error

| Insert will prevent overwriting. | ⇒ Only helpful when providing the ID. |

Update

document(s)?

db.foo.update(query, update, options)

- collection name
- what change
- options:
  - One? → 1st occurrence
  - many? → multiple occurrences
  - Upsert? → new if not present

db.a.save({_id:1, x:10}) ↵

To increment

Find document (Query - JS object) | what change

db.a.update({_id:1}, {$inc:{x:1}}) ↵

To add variable

↳ db.a.update({_id:1}, {$set:{y:3}}) ↵

as → { "_id":1, "x":11, "y":3 }

To remove

↳ db.a.update({_id:1}, {$unset:{y:0}}) ↵

as → y is removed { _id:1, x:11 }

To rename

↳ db.a.update({_id:1}, {$rename:{"x":"X"}})

as → { _id:1, X:11 }

To add array

db.a.update({_id:1}, {$push:{things:'one'}})

as → { "_id":1, "X":11, "things":["one"] }

To push more items

↳ db.a.update({_id:1}, {$push:{things:'two'}})

as → { "_id":1, "X":11, "things":["one","two"] }

To push same items more than once

↳ db.a.update({_id:1}, {$push:{things:"two"}})

as → { "_id":1, "X":11, "things":["one","two","two"] }

To avoid pushing multiple times

↳ db.a.update({_id:1}, {$addToSet:{things:"two"}})

as → { "_id":1, "X":11, "things":["one","two"] }

To pull out of array

{ _id : 1 , X : 11 , things : ["one", "two", "four"] }

db.a.update({_id:1}, { $pop : { things : 1 } })

o/p = things : ["one", "two"] ← last element

o/p = things : ["two"] ← (-1) → first element

If "things" is on a string all the array operations will fail

{ "_id": 1 , "things" : [1 , 2 , 3] }
{ "_id": 2 , "things" : [2, 3] }
{ "_id": 3 , "things": [3] }
{ "_id": 4 , "things": [1, 3] }

Consider the four documents saved in our collection

If we want to update to all the documents, we need to provide {empty matching object query}

db.a.update({}, { $push : { things : 4 } })

→ This will update only 1 record.

To enable it to apply for all the records

→ db.a.update({}, { $push : { things : 40 } }, { multi : true })

To update only the document having "2"

→ db.a.update({things : 2}, { $push : { things : 42 } }, { multi : true })

FindAndModify

```
db.foo.findAndModify({
    query : <document>,      → which document
    update : <document>,     → What change?
    upsert : <boolean>,      → creat new if not present?
    delete : <boolean>       → delete?
    new : <boolean>          → returns old version before by default. setting this true will return document after change.
    sort : <document>        ← Query order
    fields : <document>      → return which fields?
});
```

example

```
{ "_id":1, "things":[1,2,3] }
{ "_id":2, "things":[2,3] }
{ "_id":3, "things":[3] }
{ "_id":4, "things":[1,3] }
```

Javascript object mod is defined

```
mod = {
    "query" : {
        "things" : 1
    },
    "update" : {
        $set : {
            "touched" : true
        }
    },
    "sort" : {
        "_id" : -1
    }
}
```

↑ descending order

findAndModify

```
db.a.update(mod)
```

ans:-

```
{ "_id" : 4, "things" : [1,3] }
```

~~"touched" : true~~

Document got before modified.

⇒ mod.new = true

ans:- db.a.findAndModify(mod)

```
{ "_id" : 4, "things" : [1,3], "touched" : true }
```

document got after modified as new field is set to true

Retrieving documents from DB

which fields?

db.foo.find(query, projection)

which documents?

(Project only id columns)

```
db.foo.find({_id:{$gt:5}}, {_id:1})
```

1 → indicates include.
0 → indicates exclude.

$gt → greater than.

$lt → Less than

$lte → Less than equal

$gte : greater than equal.

```
db.foo.find({_id:{$not:{$gt:2}}})
```

↳ returns documents whose id is not greater than 2.

```
db.foo.find({_id:{$in:[1,3]}})
```

↳ returns documents whose id field matches 1 or 3.

```
db.foo.find({_id:{$nin:[1,3]}})
```

↳ returns documents whose ID does not match 1 or 3.

Consider the document

```
{
  "_id" : 1,
  "name" : "cat",
  "tags" : [ "land", "cute" ],
  "intro" : { "type" : "mammal", "color" : "red" }
}
```

We cannot do both include and exclude with projections. Only "_id" is the limitation. By default it always show.

Projection :- shows name id is mandatory

```
db.animals.find({tags:'cute'}, {name:1})
```

↳ Shows document of animals who have tags as cute.

Strict equality & case sensitive

```
db.animals.find({tags:{$in:['cute','ocean']}})
```

↳ returns animals with tags 'cute' or 'ocean'

```
db.animals.find( { tags : { $all :['cute','ocean']}})
```

↳ will return only those animals that are both 'cute' & 'ocean'.
- AND condition is enforced (contains all)

Sub Document ⇒

```
db.animals.find( {"info.canFly" : true} )
```

↳ will return animals whose canFly property inside info is true

⇒ Dot notation " Always enclosed "

Important Note

Dot notation should be used strictly when matching different attributes within the parent object.

Eg:- as information : {
color : red,
type : rainbow,
age : 22
}

To access matching criteria

```
db.news.find( { information : { age:22, color:red }})
```

↳ will return nothing BSO * Not only about the order. But also all the fields should match

V/S

```
db.peeps.find( { "information.age" : 22, "information.color" : red })
```

↳ will return the matching data

Acts as AND

Deleting the database

Deleting the database is simple.

Head to the database and drop it.

↳ use myDatabase ↵

db.drop~~[illegible]~~Database();↵

Cursors

sort() skip() limit()

→ Limits the data set (result-set)

→ no of records to skip

↓

can be multiple.

Value is object.

sort({age: 1, raw: -1})

↑ ascending order

↖ descending order

find V/s findOne

↳ is a cursor

↳ hasNext() function is present.