MongoDB

Introduction

What is MongoDB

- Document-Oriented storage
  - The concept of a document replaces the row
- Utilizes “SQL” features
  - Index Support
- Easy to scale
  - Auto-Sharding
  - Auto-Balancing
- Querying
  - Native language : Javascript
- Map/Reduce

JSON

```
{
"interests": ["Exercise",     "Family
Time",    "Skiing", "Camping"],

"name": "Barbara",
"dob":10/01/1985,
"lastname": Simmons",
"sex": "F",
"favorites": {
            "color": "Red”,
            "sport":"Football",
            "music": "HipHop/Rap"

}
```

- Key-value representation in textual format
  - Human readable
  - Most programming languages translate it to dictionary-type data
    - In Javascript a json object works like a key-value dictionary
  - Order of keys not spesific

Document store

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row	Document (JSON,BSON)
Column	Field
Index	Index
Join	Embedded Document

```
Document:
{
      "_id" : ObjectId("5114e0bd42…"),
      "first" : "John",
      "last" : "Doe",
      "age" : 39,
     "interests" : [
            "Reading",
            "Mountain Biking ]
     "favorites": {
           "color": "Blue",
           "sport": "Soccer"}
}
```

Interacting With MongoDB

- Various APIs
  - Python, Ruby, Perl, Java, Java, Scala…
- MongoDB uses natively javascript
  - Simple language and easy to use
  - No datatypes
  - You can write Javascript in the shell or load a file to run
- UI: a lot off options
  - mongo-express, Edda, HumongouS, Umongo, MongoVision
- Here we are going to use (mostly)the shell
  - If you are writing a script in a file: field names should be in strings (single/double quoted)

Javascritpt

```
//variable declaration
var p=10;
//datatypes are decided on running time
p="text";
//while loop
var i=1;
while(i>0  && i<=10){
 ....
}
//if
if (condition) {
   .....
} else {
   .....
}
```

Logical operators :

- and: &&
- or: ||
- not : !

Javascritpt – Functions

```
function f(p1 , p2){
        return p1*p2;
}

var prod=f(5,6);
print (prod.toString());
//you need to convert a number
//to string  before you print it
```

```
//you can set a variable as  a
//function
var f=function(p1 , p2){
        return p1*p2;
};

var prod=f(5,6);
print (prod.toString());
```

Arrays

- Arrays can contain multiple types of data

```
text=''
var index;
var fruits = ["Banana", "Orange", "Apple", "Mango"];
for (index = 0; index < fruits.length; index++) {
    text += fruits[index];
}
```

```
text=''
var stuff = ["Banana", 5, [10,9]];
for (var index = 0; index < stuff.length; index++) {
    text += stuff[index].toString();
}
```

Starting MongoDB

- Standalone Server :

```
>mongod  -port 2020
```

- You can define the port
- Other wise the default one is 27017

```
>mongo 127.0.0.1:2020

// OR

>mongo 127.0.0.1:2020/db
```

- The port has to be the same
- **127.0.0.1** : the local host/machine
- **/db** : we can specify directly which database to use

```
//if we already have a script e.g. test.js

>mongo  127.0.0.1:2020/db  test.js
```

Looping over JSON

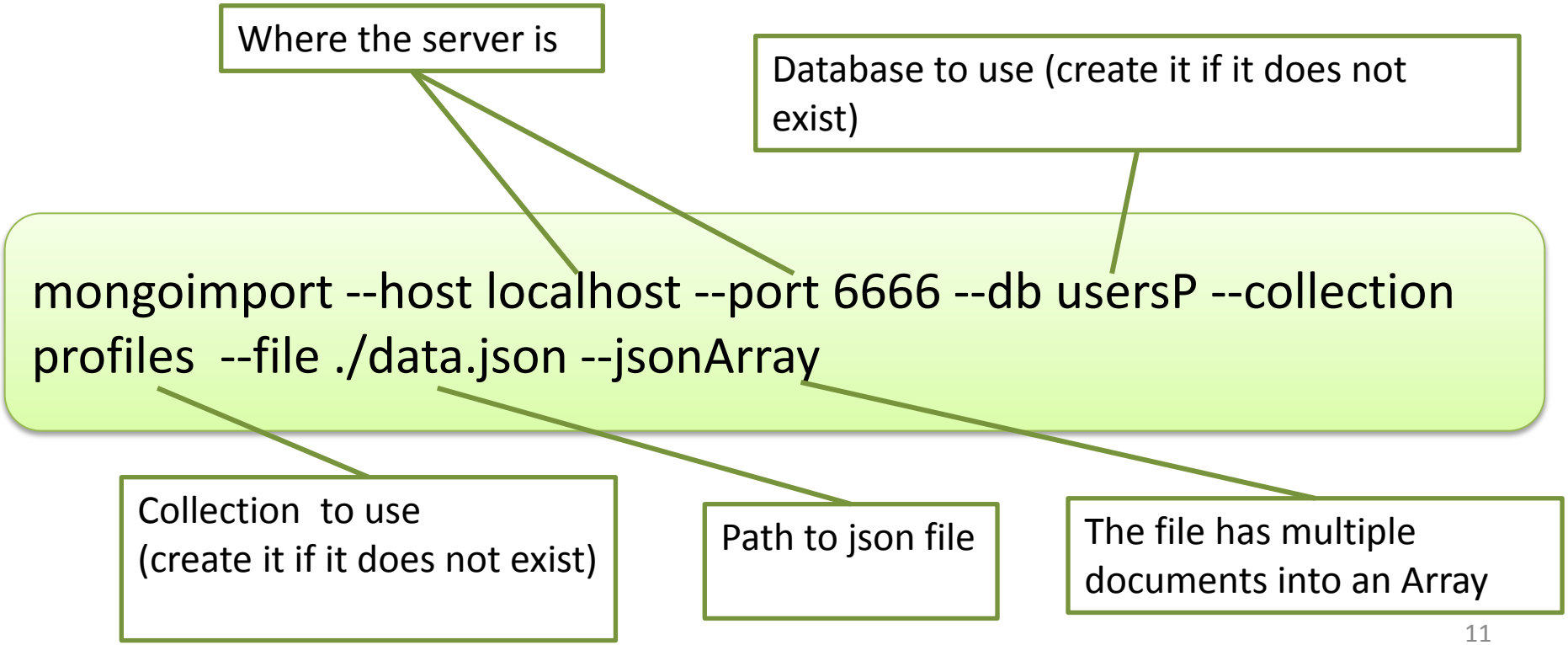
```
var json={ “name”:”John”,
  “age” : 19,
  “rating”: 1
  };
```

```
for (var key in json){
        print (key+" "+json[key]);
}
```

```
//you can print it fast and ‘nice’ with :
printjson(json);
```

Loading the database

- If you started the command line :
  - Exit with Ctrl+d



Insert

```
db.profiles.insert({name:"Rick",lastname:"Sanchez", dob:ISODate("1950-01-18T00:00:00Z")})
```

- You can insert stuff with completely different fields
  - A completely inconsistent “schema” would make the database management very hard

The cursor

```
var myCursor = db.profiles.find( { name: 'John' } );
while (myCursor.hasNext()) {
        printjson(myCursor.next());
}
```

```
var myCursor = db. profiles.find( { name: 'John' } );

myCursor.forEach(function (doc) { printjson(doc);});
```

Selection and Projection

- db.collection.find(**query, projection**)
  - Query and projection optional

```
db.profiles.find({name:{$eq:'John'}},{favorites:1})
```

```
db.profiles.find({name:{$eq:'John'}},{favorites:0})
```

```
db.profiles.find({},{_id:0})
```

```
db.profiles.find({},{_id:0}).limit(10)
```

```
db.profiles.find({$and:[{name:{$eq:'John'}},{"favorites.color":"Black"}]})
```

```
db.profiles.find({$and:[{name:{$eq:'John'}},{"favorites.color":{$ne:null}}]},
{favorites:1})
```

Delete documents

- Delete on query

```
db.profiles.remove({name:"Rick",lastname:"Sanchez"})
```

Sort and Count

```
db.profiles.find({name:{$eq:'John'}},{name:1,lastname:1}).sort({lastname:1})
```

```
db.profiles.find({name:{$eq:'John'}},{'favorites.color':1})
                                                    .sort({'favorites.color':-1})
```

```
db.profiles.find({name:{$eq:'John'}}).count()
```

Update

- db.collection.update(<query>, <update>, { upsert: <**boolean**>, multi: <**boolean**>,.})

```
//insert him again
db.profiles.insert({name:"Rick",lastname:"Sanchez", dob:ISODate("1950-01-18T00:00:00Z")})

db.profiles.update({name:"Rick",lastname:"Sanchez"},{name:"Rick",lastname:"Sanchez",dob:ISODate("1951-01-18T00:00:00Z"),nephew:"Morty"})
```

- $set/$unset

```
//this does not require to re-write the entire document
db.profiles.update({name:"Rick",lastname:"Sanchez"},{$set:{occupation:"Scientist"}})
```

Arrays in MongoDB (1)

```
//We can set an array
db.profiles.update({name:"Rick",lastname:"Sanchez"},{$set:{interests:['booze','science']}})
//or add new stuff
db.profiles.update({name:"Rick",lastname:"Sanchez"},{$push:{interests:'money'}})
```

```
db.profiles.update({name:"Rick",lastname:"Sanchez"},{$push:{interests:['unity','rebelion']}})

db.profiles.update({name:"Rick",lastname:"Sanchez"},{$pushAll:{interests:['unity','rebelion']}})
```

Arrays in MongoDB (2)

We can query on the values of an array
e.g. **How many people are interested in Reading ?**

```
>db.profiles.find({interests:'Reading'}).count()
```

We can also search based on the length of the array
e.g. **How many are interested in Reading AND have 2 or more interests?**

```
>db.profiles.find({interests:'Reading', $where:"this.interests.length>=2"}).count()
```

How many are interested in Reading OR Walking?

```
db.profiles.find({interests:{$in:['Reading','Walking']}}).count()
```

How many are interested in Reading AMD Walking?

```
db.profiles.find({interests:{$all:['Reading','Walking']}}).count()
```

Indexes

- By default there is an index on _id

```
//index in ascending order the values
db.profiles.createIndex({name:1})
```

```
//allow only unique values
db.profiles.createIndex({name:1}, { unique: true })
```

```
//build in background
db.profiles.createIndex({name:1}, {background: true})
```

```
//compound
db.profiles.createIndex({name:1,lastname:1})
```

Distinct values

- db.collection.distinct(**field, query**)

```
db.profiles.distinct('interests')
```

Aggregation in MongoDB

- db.collection.aggregate([{ <stage> }, ...])

```
db.profiles.aggregate([{$project:
                        {'age':
                            {$divide:[
                                {$subtract:[new Date(),'$dob']},
                                31558464000
                                ]
                            },
                        'name':1}
                        },
                    {$group:{
                            _id:'$name',
                            avgAge:{$avg:'$age'}
                        }
                     },
                    {$match:{avgAge:{$gt:18}
                }}])
```

Accumulators

$sum	Returns a sum for each group. Ignores non-numeric values.
$avg	Returns an average for each group. Ignores non-numeric values.
$first	Returns a value from the first document for each group.
$last	Returns a value from the last document for each group.
$max	Returns the highest expression value for each group.
$min	Returns the lowest expression value for each group.
$push	Returns an array of expression values for each group.
$addToSet	Returns an array of unique expression values for each group. Order of the array elements is undefined.

Operator (reminder)

Name	Description
$project	Manage the fields you want to use
$match	Apply a query to filter the data
$limit	Use only the first n documents
$skip	Skip n documents
$unwind	Applied in a array which flattens it .
$group	Groups input documents by a specified identifier expression and applies accumulator expression(s),.
$sort	Reorders the document stream by a specified sort key.
$out	Writes the resulting documents of the aggregation pipeline to a collection.

Aggregation in MongoDB

Find the top 5 most popular interests :

```
db.profiles.aggregate(
[
        {$unwind: "$interests"},
        { $group:
                { _id: "$interests",
                "count": { $sum: 1 } }
        },
        {$sort :{"count":-1}},
        {$limit: 5}
] )
```

If the aggregating field is an array we can expand so all possible values are used

Each appearance of a keyword counts as 1
Sum on the appearances

Sort in descending order by appearance

Keep only the first 5 results

Exercises

1. Count how many people are over 30
2. In the same aggregation:
   - For each music genre count the number of people who favorite it and group music genres in bin-counts of ’10s’
     - E.g. Pop and rock were like by [10-20) then they belong to bin 10
       - Helpful functions: $mod[a,b] =modulo(a,b), $subtract(a,b)=a-b
       - floor(n)=n - n mod 1
3. Find the top-5 music genres for people who like "Listening to Music"
4. For each favorite sport find the top interest

Resources

- http://docs.mongodb.org/manual/
- Python (almost exactly the same API as javascript):
  - https://api.mongodb.org/python/current/
- Java :
  - http://docs.mongodb.org/getting-started/java/
- UIs:
  - http://docs.mongodb.org/ecosystem/tools/administration-interfaces/