DAY 3

NODE.JS

Belkacem Alidra

June 28, 2018

nomades.ch

TODAY

- MongoDB
  - Collections and Documents
  - Setup
    - Local installation vs MLab
    - CLI client vs Robo 3T
  - MongoDB Shell
  - CRUD
  - Mongoose
  - Node MongoDB Native
  - API design
    - RESTful API
- Let's code !

MONGODB

NODAL AND MONGODB

NoSQL databases are *non relational* and usually lack [ACID transactions](#).

MongoDB is a NoSQL, *document-oriented* database Engine using *JSON-like* documents. It is popular in combination with Node.js/Express.js and AngularJS (aka the "MEAN" stack).

MongoDB is well documented at [docs.mongodb.com](#).

COLLECTIONS AND DOCUMENTS

In the MongoDB terminology, a *collection* is a group of *documents* within a single *database*.
It is the equivalent of an RDBMS table, although unlike a table a *collection* does not enforce a schema.

A *document* is a record in a MongoDB *collection* and the basic unit of data in MongoDB.
Documents are analogous to JSON objects but exist in the database in a more type-rich format known as BSON.

SETUP

LOCAL

Download and follow instructions from mongodb.com

Start local server:

% *mongod –dbpath=/Users/cas/mongodb/data*

MLAB

mLab

WELCOME PLANS & PRICING DOCS & SUPPORT ACCOUNT LOG OUT

{ user: "cas746", account: "Belkacem Alidra" }

Home

Database: yabe-development

Delete database

To connect using the mongo shell:

```
% mongo ds119651.mlab.com:19651/yabe-development -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds119651.mlab.com:19651/yabe-development
```

mongod version: 3.4.15 (MMAPv1)

Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.

Collections | Users | Stats | Backups | Tools

Collections

Delete all collections + Add collection

NAME	DOCUMENTS	CAPPED?	SIZE
posts	1	false	8.09 KB

System Collections

NAME	DOCUMENTS	SIZE
system.indexes	1	0.11 KB

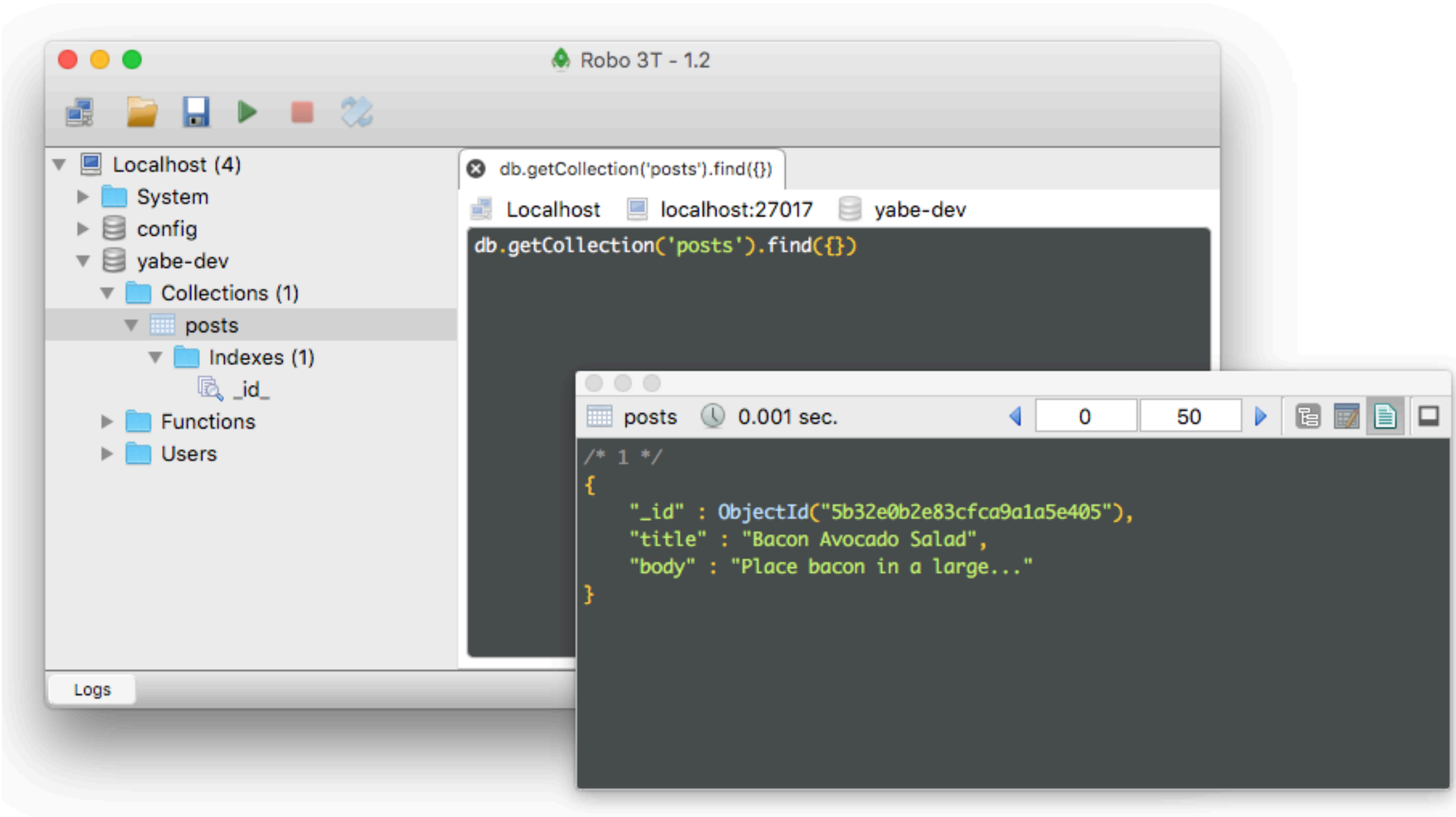
CLI

Use local client:

% *mongo [my_db_name]*

```
> db.posts.insert({"title" : "Bacon Avocado Salad", "body": "Place bacon in a large…"})
WriteResult({ "nInserted" : 1 })
> db.posts.find().pretty()
{
    "_id" : ObjectId("5b32e0b2e83cfca9a1a5e405"),
    "title" : "Bacon Avocado Salad",
    "body" : "Place bacon in a large..."
}
```

ROBO 3T


Robo 3T - 1.2
Localhost (4)
System
config
yabe-dev
Collections (1)
posts
Indexes (1)
_id_
Functions
Users
Logs
db.getCollection('posts').find({})
Localhost
localhost:27017
yabe-dev
db.getCollection('posts').find({})
posts
0.001 sec.
0
50
/* 1 */
{
"_id" : ObjectId("5b32e0b2e83cfca9a1a5e405"),
"title" : "Bacon Avocado Salad",
"body" : "Place bacon in a large..."
}

MONGODB SHELL

LOCAL

MongoDB has a interactive JavaScript command line shell (based on Mozilla SpiderMonkey) with ES6 features (v3.2+)

```
% mongo
MongoDB shell version v3.6.5
connecting to: mongodb://127.0.0.1:27017
> interpreterVersion()
MozJS-38
> console.log("Hello Mongo ?")
2018-06-27T03:11:08.880+0200 E QUERY    [thread1] ReferenceError:
console is not defined : @(shell):1:1
> print("Hello Mongo!")
Hello Mongo!
> [1,2,3].map(x => x * x)
[ 1, 4, 9 ]
```

CLI

You can also write Mongo Shell scripts and execute them:

```
% mongo dbname myscript.js
```

LOCAL

databases and *collections* are automatically created when the first document is inserted:

```
> show dbs
admin  0.000GB
local  0.000GB
> db
test
> show collections
> db.posts.insertOne({"title": "Homemade Brownie"})
{ "acknowledged" : true, … }
> show collections
posts
> show dbs
admin  0.000GB
local  0.000GB
test   0.000GB
```

CRUD

CREATE

Creating one document

```
db.posts.insertOne({                          # collection
 title: "Bacon Avocado Salad",                # field: value |
 body: "Place bacon in a large...", # field: value | document
 score: 5,                                    # field: value |
})
```

Creating many documents

```
db.posts.insertMany([
  {title: "Crispy Orange", body: "Lay beef...", score: 2},
  {title: "Simple BBQ Ribs", body: "Place ribs...", score: 4},
])
```

More at create-operations.

READ

MongoDB supports many query operators, like $lte, $eq, $neq, $in, etc.

```
db.posts.find(          # collection
 {score: {$gt: 3}},     # query criteria
 {title: 1, score: 1}, # projection
).limit(5)              # cursor modifier
```

Additionnally, it provides more powerful interfaces for queries like the $where operator, an Aggregation Pipeline, and Map-Reduce.

More at read-operations.

$WHERE

Write your query filter with a Javascript function !

```
db.foo.find( { $where: function() {
   return (hex_md5(this.name) == "9b53e667f30cd329d6a83e994")
}});
```

Be careful:

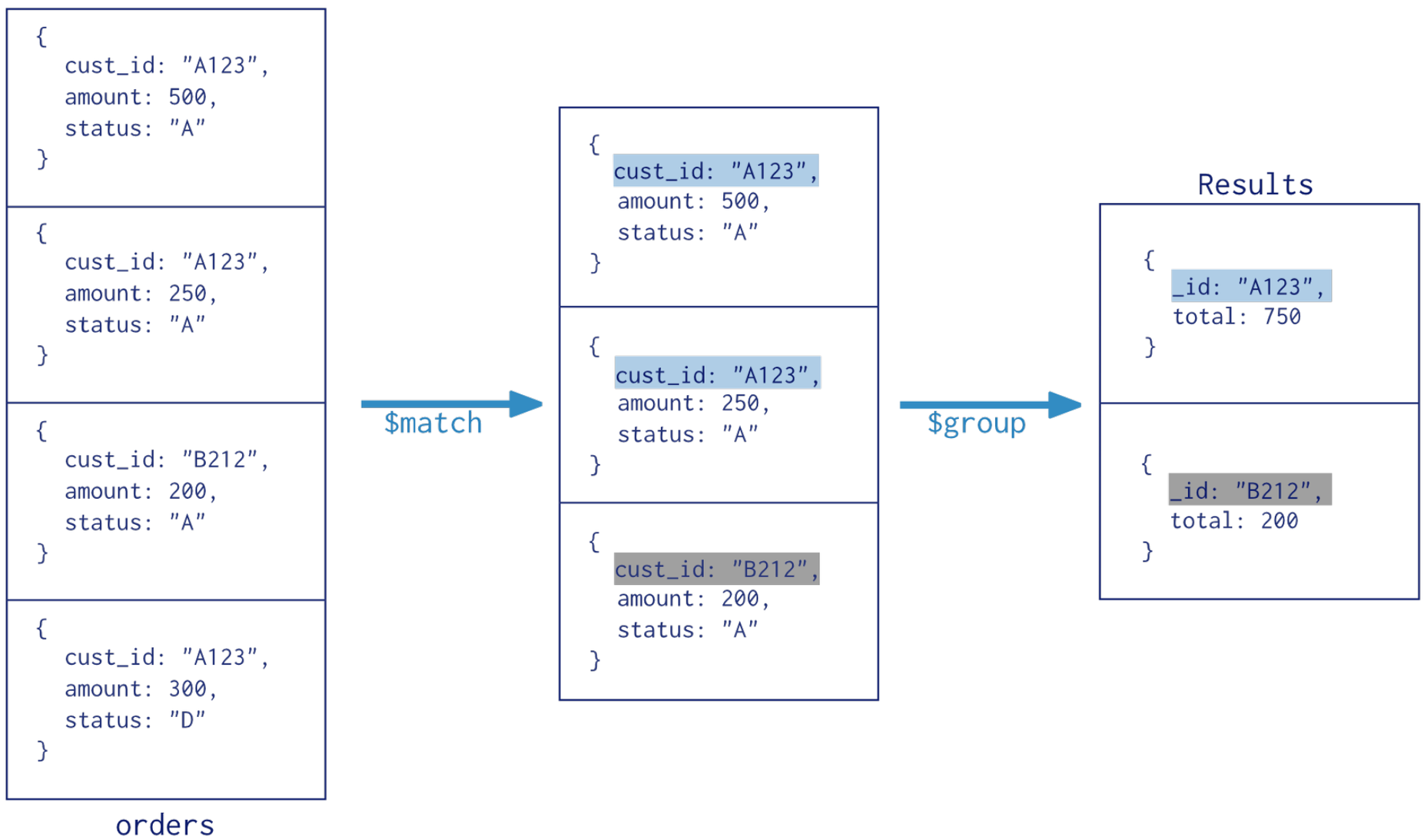
`$where` evaluates JavaScript and cannot take advantage of indexes.

More at $where operator.

AGGREGATION PIPELINES

Collection

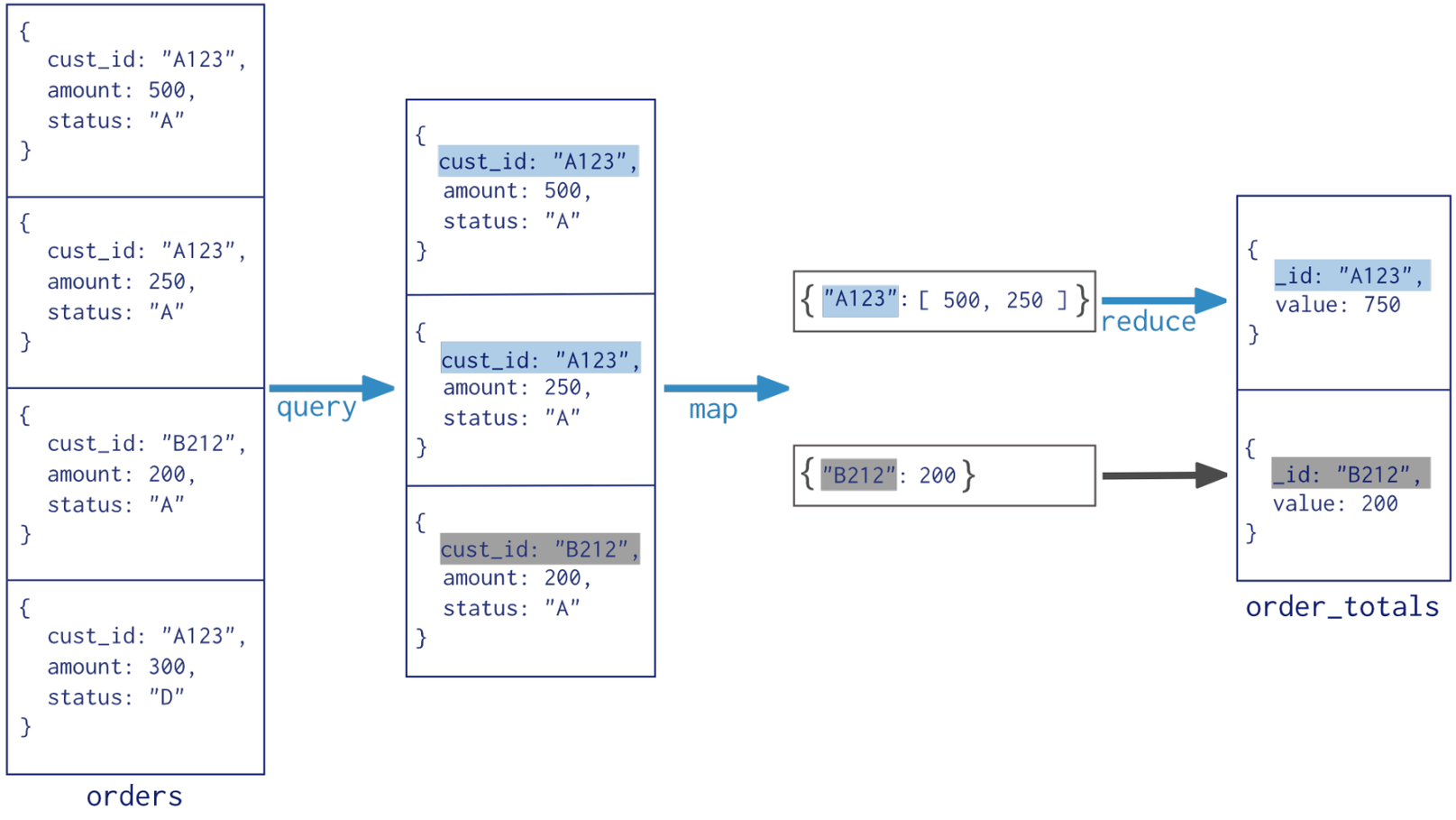
```
db.orders.aggregate( [
    $match stage →   { $match: { status: "A" } },
    $group stage →   { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
                   ] )
```



MAP REDUCE

```
db.orders.mapReduce(
        map      →  function() { emit( this.cust_id, this.amount ); },
        reduce   →  function(key, values) { return Array.sum( values ) },
                    {
        query    →    query: { status: "A" },
        output   →    out: "order_totals"
                    }
                )
```

Collection



UPDATE

Updating one document

```
db.posts.updateOne(                    # collection
 {title: "Bacon Avocado Salad"}, # update filter
 {$set: { score: 2}}                   # update action
)
```

```
db.posts.findAndModify({
 "query": {title: "Bacon Avocado Salad"},
 "update": {$set: { score: 2}},
 "new": true
})
```

Updating many documents

```
db.posts.updateMany(
  {title: /bacon/},
  {$set: { score: 2}}
)
```

More at update-operations.

DELETE

Deleting one document

```
db.posts.deleteOne(                    # collection
 {title: "Bacon Avocado Salad"}  # delete filter
)
```

Deleting many documents

```
db.posts.deleteMany(
  {score: {$lt: 5}}
)
```

More at delete-operations.

INDEXES

Indexes should be used in order to optimize queries, otherwise MongoDB must scan the whole collection:

```
db.posts.createIndex({score: 1}, {unique: false});
```

MongoDB automatically creates a unique index on the _id field that can not be dropped.

More at indexes and db.collection.createIndex().

MODEL DATA FOR ATOMIC OPERATIONS

in MongoDB, *all write operations* are atomic on the level of *a single document*.

See Model Data for Atomic Operations.

MONGOOSE

MONGOOSE

Mongoose is a MongoDB object modeling framework for Node.js. It provides mechanisms to describe *Schema*, *validations*, limited *versioning*, *relations* (!), and more.

```
1   var mongoose = require(‘mongoose');
2   mongoose.connect('mongodb://localhost/test');
3
4   var Cat = mongoose.model('Cat', { name: String });
5
6   var kitty = new Cat({ name: 'Zildjian' });
7   kitty.save(function (err) {
8     if (err) {
8       console.log(err);
9     }
10    else {
11      console.log(‘meow');
12    }
13  });
```

Documentation at mongoosejs.com

MONGOOSE

Query examples

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

// Create a new instance with new operator
var small = new Tk({ size: 'small' });

// Create a new instance using static method
Tk.create({ size: 'small' }, function (err, small) { … })

// Find
Tk.find({size:’sm’}).where('cDate').gt(oneYearAgo).exec(cbk);
Tk.findById(id, function (err, tank) { … })
Tk.findByIdAndUpdate(id, {$set:…}, {new: true}, function (err, tank) {…})

// Delete
Tk.deleteOne({size:'sm'}, function (err) { … })

// Update
Tk.updateOne({size:’sm’}, {size: ’xl’}, function (err, res) { … })
```

MONGOOSE

Validation

```
var userSchema = new Schema({
  username: {
    type: String,  // Built-in validator
    unique: true   // Not a validator
  },
  age: {
    type: Number,  // Built-in validator
    min: [7, ‘Too young’],   // Constraint with error message
    max: [77, ‘Too old’],  // Constraint with error message
  }
});
```

MONGOOSE

Custom validation

```
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’]
  }
});
```

NODE MONGODB NATIVE

NODE MONGODB NATIVE

The official MongoDB Node.js driver is the most direct way to talk to MongoDB. It is also the most similar to the MongoDB Shell interface.

Documentation at mongodb.github.io

```
const MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/test',
  function (err, db) {
    if (err)
      return console.error(err.message);

    const collection = db.collection(‘cats');
    collection.findOne({name: 'Zildjian'}, (err, kitty) => {

    if (err)
      return console.error(err.message);
    console.dir(kitty, {colors: true});
    collection.findAndModify(
      /* query */ {_id: kitty._id, __v: kitty.__v},
      /* sort */ [],
      /* update */ {
      $set: {name: 'Azrael'},
      $inc: {__v: 1} },
      /* options */ {'new': true},
      /* callback */ function (err, result) {
        if (err)
          return console.error(err.message);
        console.dir(result.value, {colors: true});
        db.close();
      });
    });
});
```

API DESIGN

API DESIGN

Designing an API is the most critical (and also probably the hardest) part of a project.

There is a ton of resource on the web to help, for example the five golden rules for designing a great Web API. Experience as both user and designer is key. Always keep in mind the Robustness principle.

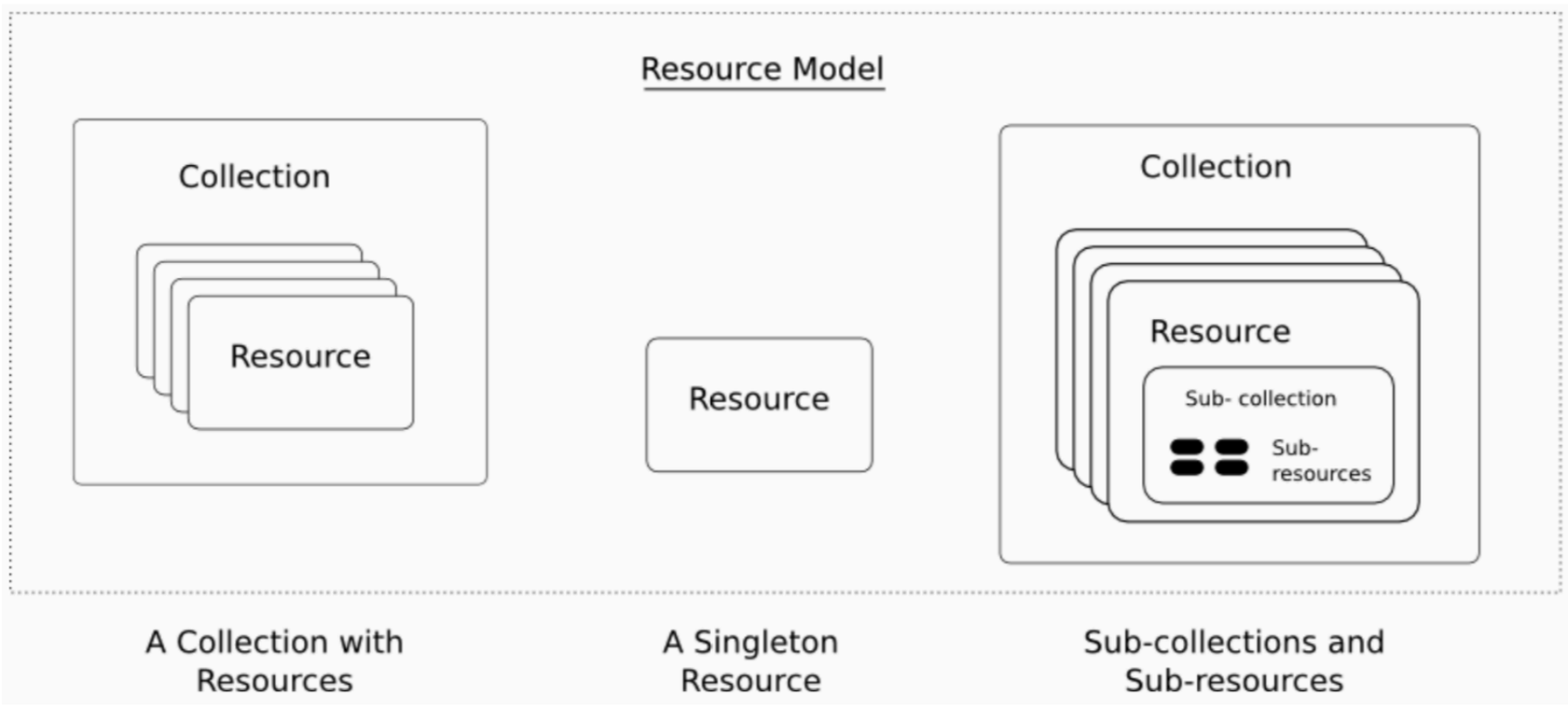
BE LIBERAL IN WHAT YOU ACCEPT, AND CONSERVATIVE IN WHAT YOU SEND.

Jon Postel

RESTFUL API

REST is an architectural style for API widely adopted in the world wide web. In other words, it is a set of constraints (rules) for API designers.

The fundamental concept in any RESTful API is the *resource*.



CRUD AND HTTP VERBS

- Create: POST
- Read: GET
- Update: PUT and PATCH
- Delete: DELETE

Note that GET is a "Safe method".
While all HTTP verbs are supported using AJAX, browsers are only able to perform GET and POST requests from a <form>. Thus, it is common to allow method overriding, see for example method-override for Express.js

RESTFUL API DESIGN

Guides for building beautiful RESTFul JSON API:

- Heroku's HTTP API Design Guide
- RESTful API design

LET'S CODE !

YET ANOTHER BLOG ENGINE

1. Replace the static hard-coded posts with MongoDB
2. **Ensure that your API is RESTful and well documented**
3. Add a state-changing action, for example publish / unpublish. Rational at [RESTful API Design](#).

```
app.post('/api/posts/:id/actions/publish', (req, res) => ...);
app.post('/api/posts/:id/actions/unpublish', (req, res) => ...);
```

Questions ?

READ ON LATER

You Are Not Google by Ozan Onay.