

NoSQL et MongoDB

Joan ORTEGA

NoSQL



No SQL (?) - Not only SQL (?)

Non-relationnelles, distribuées, open-source, scalables et haute disponibilité.

Taxonomie en fonction de leur façon de stockage des données

Taxonomie NoSQL

key-value



document



graphe



colonne

multi model

MongoDB



Supporte un riche langage de requetage :
opérations I/O, aggregations et requêtes géospatiales

Permet une scalabilité horizontale : SHARDS

Les données sont stockées en forme de documents :

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

MONGO SET UP

Installation du MongoDB

Plusieurs options en fonction du OS. (Normalement un tgz)

```
joan@jomaora:~$ ll /usr/bin/mongo
-rwxr-xr-x 1 root root 11957584 mars 23 18:58 /usr/bin/mongo*
joan@jomaora:~$ ll /usr/bin/mongo
mongo      mongodump   mongofiles  mongooplog  mongorestore  mongostat
mongod     mongoexport mongoimport  mongoperf   mongos        mongotop
```

mongod

Mongo daemon. Processus principal.

Par défaut, mongod...

écrit sur /data/db.

(--dbpath)

est lancé sur le port 27017

(--port)

peut avoir une fichier de config sur /etc/mongod.conf

(--config)

écrit les logs sur /var/log/mongodb/mongod.log

(--logpath)

```
> sudo mongod --config /etc/mongod.conf
```

```
> sudo mongod --dbpath tmp/data --logpath /tmp/logs
```

mongod

Il peut être exécuté / configuré comme un service (--fork)

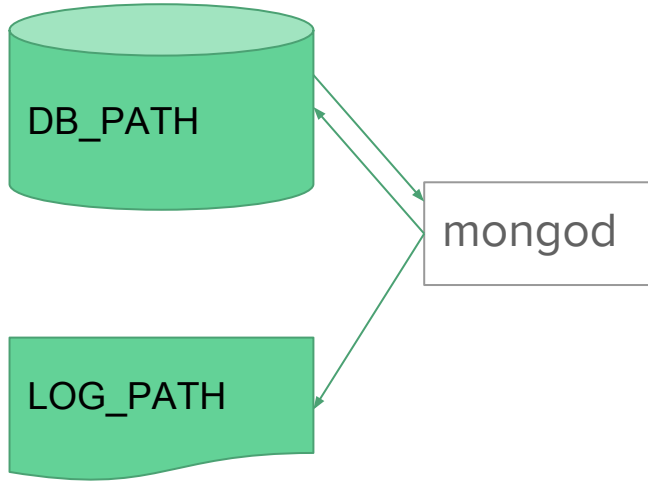
```
> sudo service mongod start
```

Pour arreter le processus :

```
> sudo service mongod stop -- si lancé en mode service
```

```
> mongod --shutdown --dbpath PATH -- pour arreter une instance
```


MongoDB

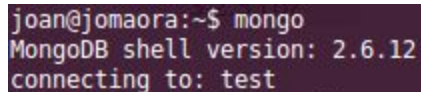


Mongo shell

Lancé par la commande **mongo**. (C'est le client de base)

Il indique la version du shell installé.

Et la bdd à laquelle il se connecte par défaut.



```
joan@jomaora:~$ mongo
MongoDB shell version: 2.6.12
connecting to: test
```

A terminal window with a dark purple background. The text is white. It shows the command 'mongo' being executed, followed by the output 'MongoDB shell version: 2.6.12' and 'connecting to: test'. A green arrow points from the text 'Et la bdd à laquelle il se connecte par défaut.' to the 'connecting to: test' line. Another green arrow points from the text 'Il indique la version du shell installé.' to the 'MongoDB shell version: 2.6.12' line.

Le shell est un interpreteur interactive de JS → comme NodeJS en mode REPL

Mongo Shell

```
> show databases
```

```
> use [NOM_DE_LA_DATABASE]
```

```
> show collections
```

Mongo Shell

Méthodes du Shell

```
Array(  
BinData(  
Boolean(  
BulkWriteError(  
BulkWriteResult(  
CountDownLatch(  
DBCommandCursor(  
DBExplainQuery(  
DBPointer(  
Date(  
ErrorCodeStrings(  
ErrorCodes(  
Explainable(  
Geo(  
HexData(  
ISODate(  
Infinity(  
JSON(  
MD5(  
MR(  
Map(  
MapReduceResult(  
Math(  
MaxKey(  
MinKey(  
Mongo(  
MongoBridge(  
MongoRunner(  
NaN(  
Number(  
NumberInt(  
NumberLong(  
ObjectId(  
PlanCache(  
QueryPlan(  
Random(  
RegExp(  
ReplSetTest(  
ReplTest(  
ShardingTest(  
String(  
SyncCCTest(  
TestData(  
Timestamp(  
ToolTest(  
UUID(  
WriteCommandError(  
WriteConcern(  
WriteResult(  
allocatePort(  
allocatePorts(  
argumentsToArray(  
assert(  
authutil(  
benchFinish(  
benchRun(  
benchRunSync(  
benchStart(  
bsonsize(  
cat(  
cd(  
chatty(  
checkProgram(  
clearRawMongoProgramOutput(  
compare(  
compareOn(  
connect(  
connectionURLTheSame(  
constructor(  
copyDbpath(  
copyFile(  
create(  
db(  
decodeURI(  
decodeURIComponent(  
defaultPrompt(  
defineProperties(  
defineProperty(  
doassert(  
encodeURI(  
encodeURIComponent(  
escape(  
eval(  
false(  
freeze(  
friendlyEqual(  
gc(  
getActiveCommands(  
getBuildInfo(  
getHostName(  
getMemInfo(  
getOwnPropertyDescriptor(  
getOwnPropertyNames(  
getPrototypeOf(  
hasOwnProperty(  
help(  
hex_md5(  
hostname(  
interpreterVersion(  
isExtensible(  
isFinite(  
isFrozen(  
isKeyTooLarge(  
isMasterStatePrompt(  
isNaN(  
isNumber(  
isObject(  
isSealed(  
isString(  
jsTest(  
jsTestFile(  
jsTestLog(  
jsTestName(  
jsTestOptions(  
jsTestPath(  
keys(  
listFiles(  
load(  
ls(  
md5sumFile(  
mkdir(  
module(  
myPort(  
null(  
parseFloat(  
parseInt(  
pathExists(  
preventExtensions(  
print(  
printShardingSizes(  
printShardingStatus(  
printStackTrace(  
printjson(  
printjsononline(  
propertyIsEnumerable(  
prototype(  
pwd(  
rawMongoProgramOutput(  
reconnect(  
removeFile(  
replSetMemberStatePrompt(  
resetDbpath(  
rs(  
run(  
runMongoProgram(  
runProgram(  
seal(  
setJsTestOption(  
setVerboseShell(  
sh(  
shellAutocomplete(  
shellHelper(  
shellPrint(  
shellPrintHelper(  
sleep(  
sortDoc(  
startMongoProgram(  
startMongoProgramNoConnect(  
startParallelShell(  
stopMongoProgramByPid(  
testingReplication(  
toLocaleString(  
toString(  
tojson(  
tojsonObject(  
tojsononline(  
true(  
undefined(  
unescape(  
validateIndexKey(  
valueOf(  
waitProgram(  

```

Mongo Shell

L'objet **db** est disponible...

```
Last login: Thu Apr 28 18:28:25 on ttys008
[jmortega@mbp ~]$ mongo
MongoDB shell version: 3.2.0
connecting to: test
Server has startup warnings:
2016-05-11T13:13:25.449+0200 I CONTROL [initandlisten]
2016-05-11T13:13:25.449+0200 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
> db
test
> db.
db.adminCommand(
db.auth(
db.changeUserPassword(
db.cloneCollection(
db.cloneDatabase(
db.commandHelp(
db.constructor
db.copyDatabase(
db.createCollection(
db.createRole(
db.createUser(
db.currentOp(
db.currentOp(
db.dbEval(
db.dropAllRoles(
db.dropAllUsers(
db.dropDatabase(
db.dropRole(
db.dropUser(
db.eval(
db.forceError(
db.fsyncLock(
db.fsyncUnlock(
db.getCollection(
db.getCollectionInfos(
db.getCollectionNames(
db.getLastError(
db.getLastErrorMsg(
db.getLastErrorObj(
db.getLogComponents(
db.getMongo(
db.getName(
db.getPrevError(
db.getProfilingLevel(
db.getProfilingStatus(
db.getQueryOptions(
db.getReplicationInfo(
db.getRole(
db.getRoles(
db.getSiblingDB(
db.getSisterDB(
db.getSlaveOk(
db.getUser(
db.getUsers(
db.getWriteConcern(
db.grantPrivilegesToRole(
db.grantRolesToRole(
db.grantRolesToUser(
db.group(
db.groupcmd(
db.groupeval(
db.hasOwnProperty(
db.help(
db.hostInfo(
db.isMaster(
db.killOp(
db.killOp(
db.listCommands(
db.loadServerScripts(
db.logout(
db.printCollectionStats(
db.printReplicationInfo(
db.printShardingStatus(
db.printSlaveReplicationInfo(
db.propertyIsEnumerable(
db.prototype
db.removeUser(
db.repairDatabase(
db.resetError(
db.revokePrivilegesFromRole(
db.revokeRolesFromRole(
db.revokeRolesFromUser(
db.runCommand(
db.runCommandWithMetadata(
db.runReadCommand(
db.serverBits(
db.serverBuildInfo(
db.serverCmdLineOpts(
db.serverStatus(
db.setLogLevel(
db.setProfilingLevel(
db.setSlaveOk(
db.setWriteConcern(
db.shutdownServer(
db.stats(
db.toLocaleString(
db.toString(
db.toJson(
db.unsetWriteConcern(
db.updateRole(
db.updateUser(
db.upgradeCheck(
db.upgradeCheckAllDBs(
db.valueOf(
db.version(
```

Mongo Shell

On peut interroger des infos sur les parametres utilisés par le processus mongod :

```
joan@jomaora:~$ mongo
MongoDB shell version: 2.6.12
connecting to: test
> db.serverCmdLineOpts();
{
  "argv" : [
    "mongod",
    "--dbpath",
    "/var/lib/mongodb/"
  ],
  "parsed" : {
    "storage" : {
      "dbPath" : "/var/lib/mongodb/"
    }
  },
  "ok" : 1
}
```

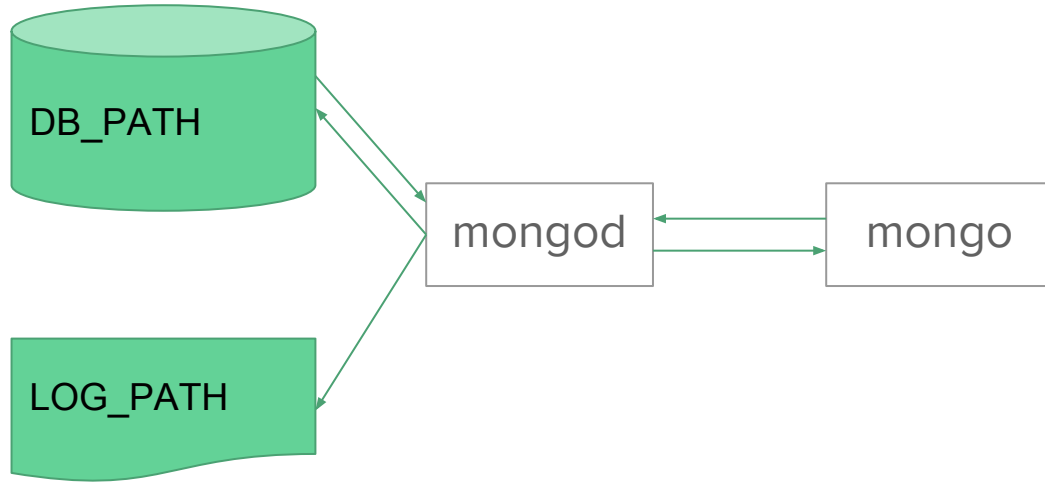
Mais sa principale utilisation est requeter les collections.

Mongo Shell

Il est possible de arreter le daemon depuis le shell

```
> use admin  
> db.shutdownServer()
```

MongoDB



Drivers

Mongo a développé divers drivers pour plusieurs langages :

C

NodeJS

Motor

C++

Perl

Ruby

C#

PHP

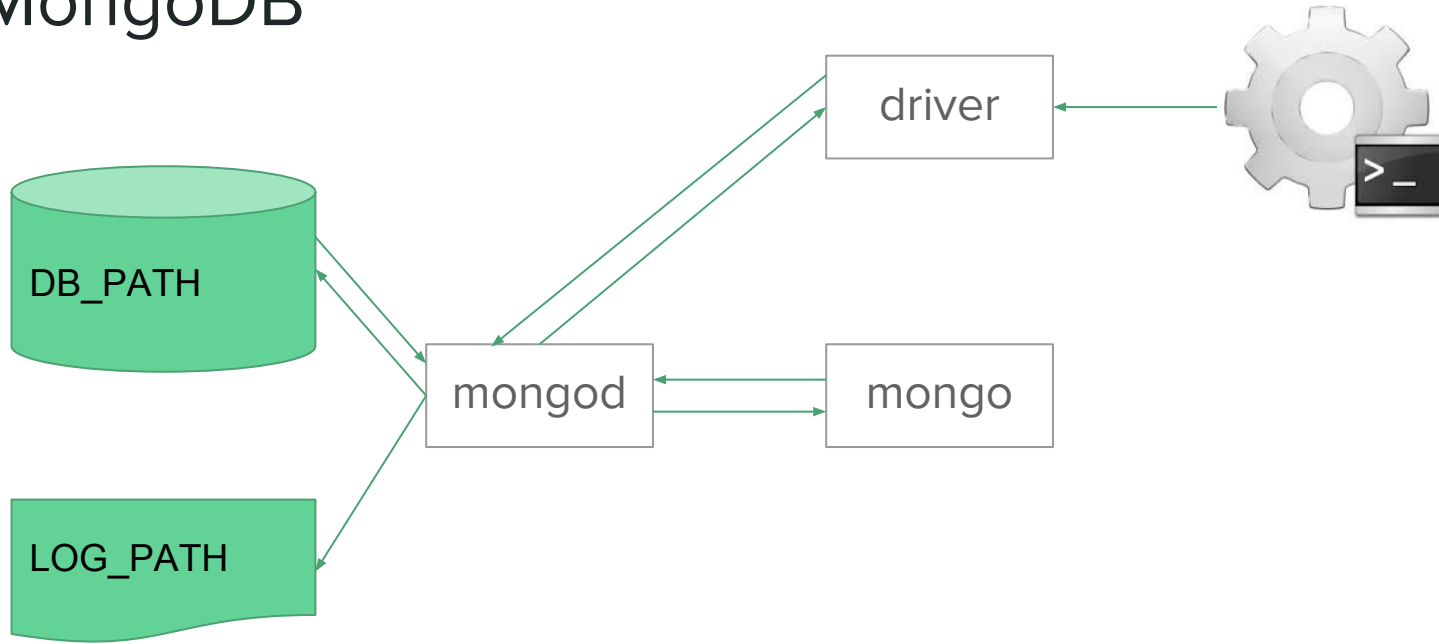
Scala

Java

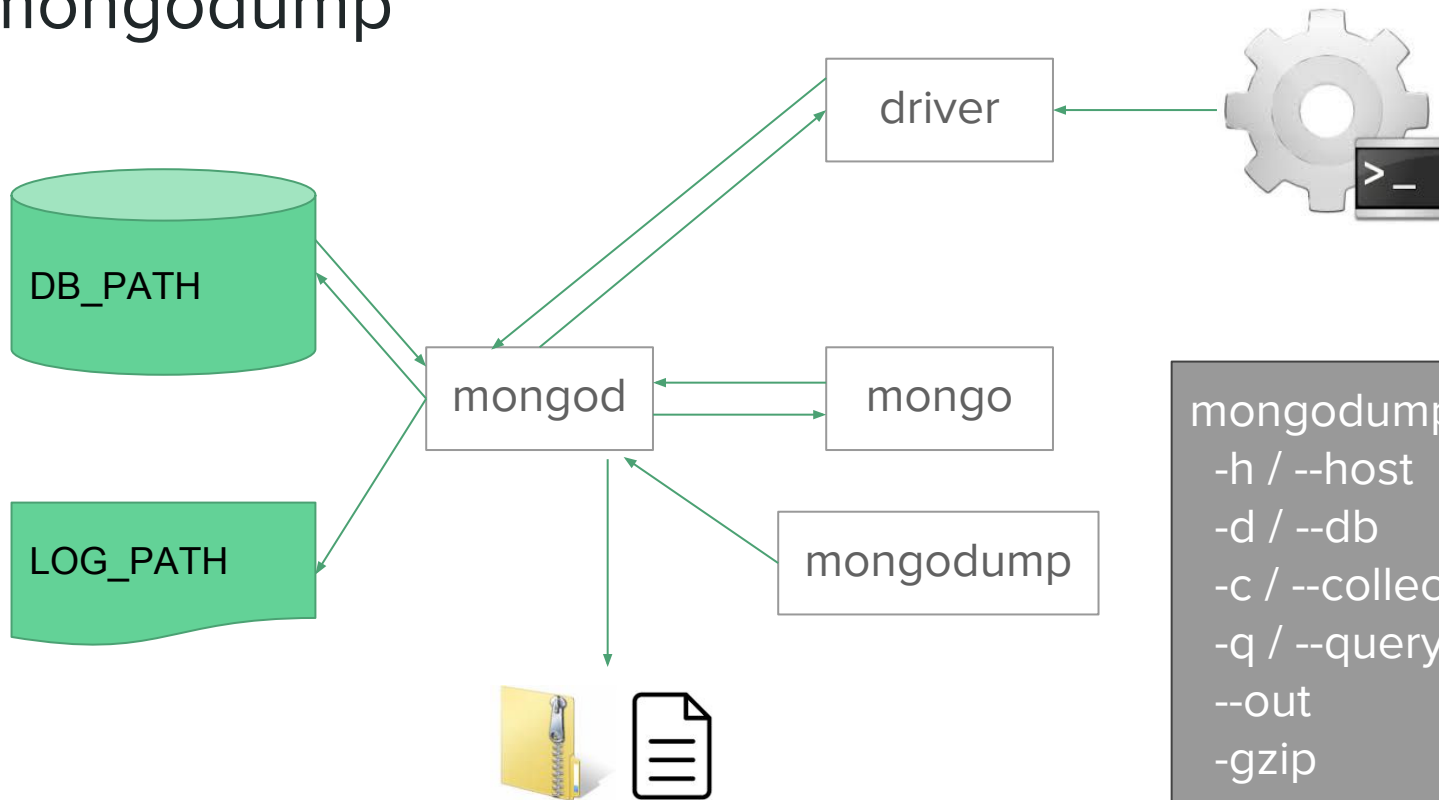
Phyton

Chaque driver assure l'interface entre le langage et MongoDB afin de pouvoir écrire des programmes et communiquer avec le processus mongod.

MongoDB

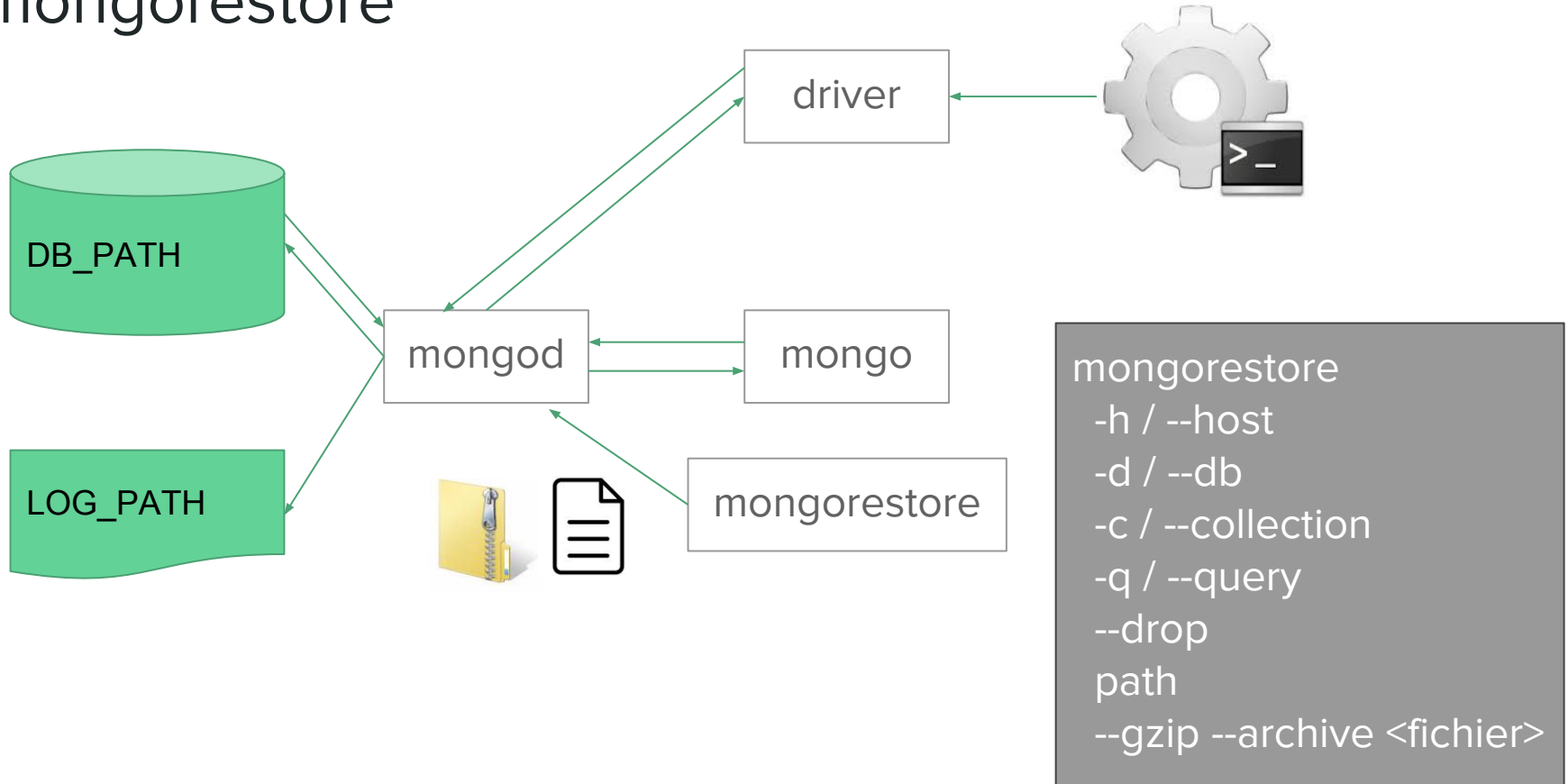


mongodump



```
mongodump
-h / --host
-d / --db
-c / --collection
-q / --query
--out
-gzip
```

mongorestore

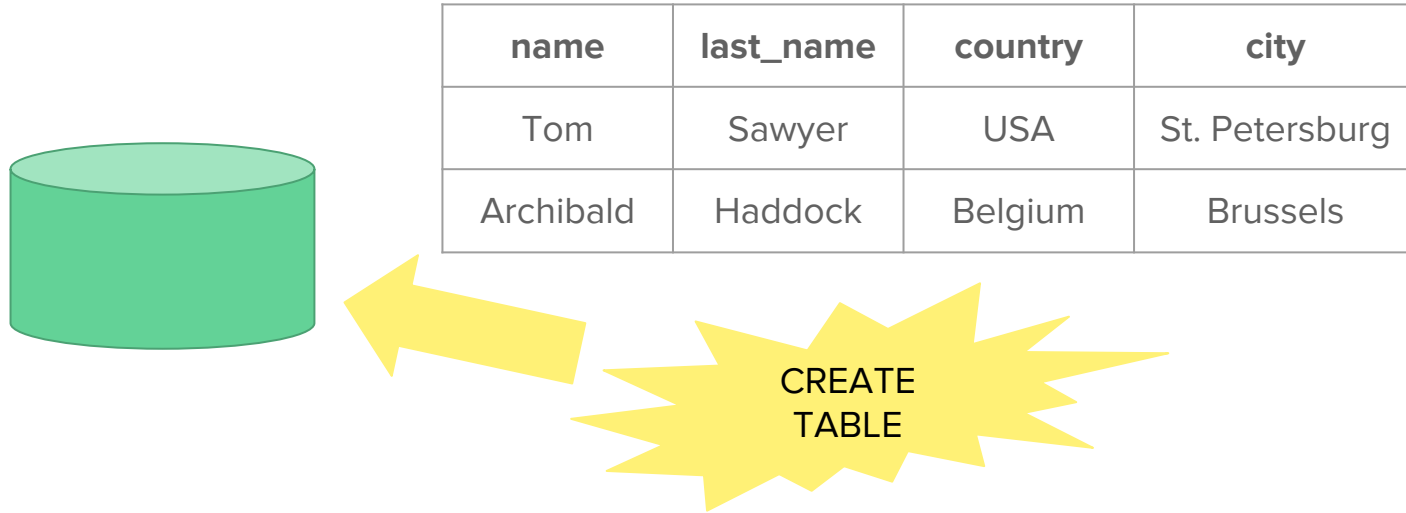


TP.

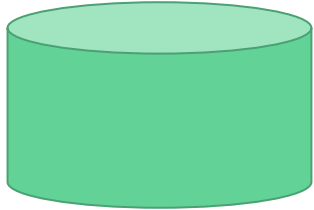
PREMIÈRE PARTIE

GÉNÉRALITÉS MONGODB

Schéma



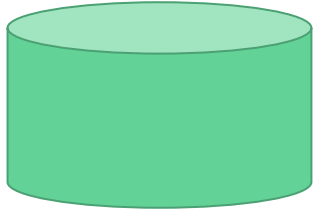
Schéma



name	last_name	country	city
Tom	Sawyer	USA	St. Petersburg
Archibald	Haddock	Belgium	Brussels

si j'ai besoin d'ajouter
des zipcodes ?

Schéma



name	last_name	country	city	zip_code
Tom	Sawyer	USA	St. Petersburg	33710
Archibald	Haddock	Belgium	Brussels	1000



Mongo est Schemaless

Plus besoin d'un langage de définition.

L'ajout des nouvelles données se font de façon transparente

Les documents stockés peuvent avoir différents attributs.

Schemaless vs Schema dynamique ?

Pourquoi schemaless c'est pas bon ?

- Si les documents n'ont pas une même structure, ça peut devenir difficile à comprendre le sens d'une collection.

Il y aura toujours un schema

- Même si un nouveau attribut n'a pas besoin d'être défini, il pourrait avoir besoin d'une initialisation sur la collection existante (faire job pour remplir les zipcodes)

Document

Les données sur MongoDB sont stockées sous forme des objets clé-valeur, inspiré du JSON de Javascript

Les valeurs ne sont pas seulement info tabular: array, un autre document :

```
{
  name: { first: "Alan", last: "Turing" },
  birth: new Date('Jun 23, 1912'),
  death: new Date('Jun 07, 1954'),
  contribs: [ "Turing machine", "Turing test", "Turingery" ],
  views : NumberLong(1250000)
}
```

Quelques contraintes :

- Les clés ne peuvent pas avoir des '.', null, ni commencer par \$

- Il y aura toujours un attribut _id, que sera le primary key

- La taille maximale d'un document est **16 MB**

BSON

Les documents sont stockées en format Binary JSON.

BSON est un sorte de JSON enrichi, qui supporte de valeurs du type :

Floating point

Arrays

Binary Data

UTC datetime

Timestamps

UTF-8 string

Boolean

Object Id

32-bit Int

et d'autres

Embedded documents

Null

Regular Expressions

64-bit Int

Collections

Les documents sont regroupés dans les collections.

Leurs définition n'est pas obligatoire :

```
db.myNewCollection2.insert( { x: 1 } )  
  
db.myNewCollection3.createIndex( { y: 1 } )
```

Mais ...

```
db.createCollection(name, { capped: <boolean>,  
                           autoIndexId: <boolean>,  
                           size: <number>,  
                           max: <number>,  
                           storageEngine: <document>,  
                           validator: <document>,  
                           validationLevel: <string>,  
                           validationAction: <string>,  
                           indexOptionDefaults: <document> } )
```

Collections

```
db.createCollection("contacts",
  {
    validator: { $or:
      [
        { phone: { $type: "string" } },
        { email: { $regex: /@mongodb\.com$/ } },
        { status: { $in: [ "Unknown", "Incomplete" ] } }
      ]
    }
  }
)
```

Collections

Il est possible de modifier certaines propriétés d'une collection existante :

```
db.runCommand( { "collMod" : <collection> , "<flag>" : <value> } )
```

Flags possibles :

- TTL
- validator
- validationLevel
- validationAction

Opérateurs

Il y a 3 familles des opérateurs :

- Queries
- Update
- Aggregation

Opérateurs - Queries

```
{ age: { $gte: 18 } }
```

COMPARAISON	
\$eq	===
\$neq	!==
\$lt	<
\$lte	<=
\$gt	>
\$gte	>=

Opérateurs - Queries

```
{ $and: [{ age: { $gt: 20 } }, { age: { $lt: 30 } }] }
```

COMPARAISON

\$eq	===
\$neq	!==
\$lt	<
\$lte	<=
\$gt	>
\$gte	>=

LOGICAL

\$and
\$or

Opérateurs - Queries

```
{ email: { $exist: true } }
```

COMPARAISON

\$eq	===
\$neq	!==
\$lt	<
\$lte	<=
\$gt	>
\$gte	>=

LOGICAL

\$and
\$or

SUR UNE PROPRIÉTÉ

\$exists
\$type

Opérateurs - Queries

```
{ firstName: { $regex: /^A(.*)a$/ } }
```

COMPARAISON

\$eq	===
\$neq	!==
\$lt	<
\$lte	<=
\$gt	>
\$gte	>=

LOGICAL

\$and
\$or

SUR une valeur

\$regex
\$mod

SUR UNE PROPRIÉTÉ

\$exists
\$type

Opérateurs - Queries

```
{ tags: { $in: ['food', 'vegan'] } }
```

COMPARAISON

\$eq	===
\$neq	!==
\$lt	<
\$lte	<=
\$gt	>
\$gte	>=

LOGICAL

\$and
\$or

SUR UNE PROPRIÉTÉ

\$exists
\$type

SUR une valeur

\$regex
\$mod

SUR UN TABLEAU

\$in	contains
\$nin	!contains
\$size	if length ===

Opérateurs - Update

SUR UNE PROPRIÉTÉ	
\$inc	=+
\$mul	=*
\$set	= assignation
\$unset	delete
\$currentDate	= new Date()

```
{ age: { $inc: 1 } }
```

```
{ price: { $mul: 0.01 } }
```

```
{ price: { $set: 300 } }
```

```
{ $unset: { $ref: "" } }
```

```
{ $currentDate: { createdAt: { $type: "date" } } }
```

Opérations

INSERT

`db.collection.insert(<document or array of documents>)`

`db.collection.insertOne(<document>)`

→ *new feature*

`db.collection.insertMany(<array document>)`

→ *new feature*

DELETE

`db.collection.remove(condition[, nombre de documents])`

`db.collection.findOneAndDelete(condition[, options])`

→ *new feature*

Opérations

UPDATE

`db.collection.findAndModify(document)`

`db.collection.findOneAndReplace(filter, replacement, options)` → *new feature*

`db.collection.findOneAndUpdate(filter, update, options)` → *new feature*

`db.collection.replaceOne(filter, replacement, options)` → *new feature*

`db.collection.update(query, update, options)`

`db.collection.updateOne(filter, update, options)` → *new feature*

`db.collection.updateMany(filter, update, options)` → *new feature*

Opérations

SELECT

```
db.collection.find(condition[, projection])
```

Cette opération retourne un cursor, sur lequel différentes opérations sont possibles :

- `pretty()`
- `toArray()`
- `sort({ attribute: [1|-1] })`
- `limit(value)`
- `count ~ db.collection.count(condition)`
- `map(callback)` → Renvoie les résultats dans un tableau

Opérations

DISTINCT

```
db.collection.distinct(attribute[, condition])
```

DROP

```
db.collection.drop()
```

AGGREGATION

```
db.collection.aggregate(pipeline, options)
```

Outils

Robomongo



Robomongo 0.9.0-RC8

File View Options Window Help

localhost (20)

- System
- aAa-BaUcis-ExAmPIE-AaA
- blog
- coopacademy
- coopacademy-digital-test
- enron
- exam-7
- lpdw
 - Collections (3)
 - System
 - songs
 - users
 - Functions
 - Users
- m101
- mymusic
- school
- social-music
- social-music-arnaud
- song

db.getCollection('us... x

localhost localhost:27017 lpdw

```
db.getCollection('users').find({})
```

users 0.215 sec. 0 50

Key	Value	Type
(1) Lyla_Marquardt	{ 3 fields }	Object
(2) Marquis_Weim...	{ 3 fields }	Object
(3) Jermey.Reinger	{ 3 fields }	Object
(4) Kamron37	{ 3 fields }	Object
(5) Jennie61	{ 3 fields }	Object
(6) Greyson77	{ 3 fields }	Object
(7) Asha_Jerde29	{ 3 fields }	Object
(8) Aimee.Kilback	{ 3 fields }	Object
(9) Federico_Rippin	{ 3 fields }	Object
(10) Jamison95	{ 3 fields }	Object
(11) Melyssa.Conn...	{ 3 fields }	Object
(12) Dan_Daniel	{ 3 fields }	Object
(13) Alessia51	{ 3 fields }	Object

Logs

Outils

MongoDB Compass



MongoDB Compass - Schema - ds045679.mlab.com:45679/mymusic.songs

ds045679.mlab.com:45679
Community version 3.0.7

1 DBs | 3 Collections |

filter

- mymusic
- ratings
- songs
- users

mymusic.songs

DOCUMENTS 43 total size 5.5 KB avg. size 130 B INDEXES 2 total size 16.0 KB avg. size 8.0 KB

{ "album": "Racine carrée" }

Query returned 3 documents.

album

string Racine carrée

artist

string Stromae

title

string Formidable Tous les memes Papaoutai

year

number

100% 50% 0%

Document 1

```
{
  "_id": "55328cb0f238ef5f0de2ad36",
  "title": "Tous les memes",
  "artist": "Stromae",
  "album": "Racine carrée",
  "year": 2013
}
```

Document 2

```
{
  "_id": "55328c56f238ef5f0de2ad35",
  "title": "Formidable",
  "artist": "Stromae",
  "album": "Racine carrée",
  "year": 2013
}
```

Document 3

```
{
  "_id": "55328bd3f238ef5f0de2ad33",
  "title": "Papaoutai",
  "artist": "Stromae",
  "album": "Racine carrée",
  "year": 2013
}
```

Outils

mLab






MongoDB Deployments

[Create from backup](#)[Create new](#)

Development and Utility

Single-node deployments intended for environments that do not require high availability.

NAME	PLAN	RAM	SIZE ?	FILE SIZE ?
<div>▼  ds045679/mymusic</div> <div>Ok: This database is up and running.</div>	Sandbox CLOUD: GCP US-CENTRAL-1 VERSION: 3.0.7	shared	1.86 MB	16.00 MB
<div>▼  ds049935/ws-places-api</div> <div>Ok: This database is up and running.</div>	Sandbox CLOUD: AWS US-EAST-1 VERSION: 3.0.8	shared	0.00 KB	0.00 KB
<div>▼  ds033170/ws-restapi</div> <div>Ok: This database is up and running.</div>	Sandbox CLOUD: AWS EU-WEST-1 VERSION: 3.0.7	shared	35.46 KB	16.00 MB

help

TP.

DEUXIÈME PARTIE

NODEJS MONGO DRIVER



Driver Mongo NodeJS

npm install mongodb --save

version": "2.1.X"

QuickStart Doc : <http://mongodb.github.io/node-mongodb-native/2.1/>

```
'use strict'
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect(url, function(err, db) {
  console.log("Connecté");
  db.close();
})
```



Driver Mongo NodeJS

npm install mongodb --save

version": "2.1.X"

QuickStart Doc : <http://mongodb.github.io/node-mongodb-native/2.1/>

```
'use strict'  
var MongoClient = require('mongodb').MongoClient;  
  
MongoClient.connect(url, function(err, db) {  
  console.log("Connecté");  
  db.close();  
})
```

```
var SongsDAO = db.collection("songs");
```





Driver Mongo NodeJS

Compatibilité entre le driver et les versions de Mongo

Node.js Driver	MongoDB 2.4	MongoDB 2.6	MongoDB 3.0	MongoDB 3.2
>= 2.1.0	✓	✓	✓	✓
>= 2.0.14	✓	✓	✓	
>= 1.4.29	✓	✓		
1.4.X	✓	✓		
1.3.X	✓			
1.2.X	✓			



Driver Mongo NodeJS

Compatibilité entre le driver Mongo NodeJs et les versions de NodeJS

Node.js Driver	NodeJS v0.8.X	NodeJS v0.10.X	NodeJS v0.12.X	NodeJS v4.X.X
>= 2.1.0	✓	✓	✓	✓
>= 2.0.14	✓	✓	✓	✓
>= 1.4.29	✓	✓	✓	
1.4.X	✓	✓		
1.3.X	✓	✓		
1.2.X	✓	✓		

Exemple : find sur une collection

```
'use strict'
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect(url, function(err, db) {
  console.log("Connecté");

  var Songs = db.collection('songs');

  Songs.find({year: {$gt: 2010}}).toArray(function(err, songs) {
    console.log(songs);
    console.log(songs.length + ' songs found');
    db.close();
  });
})
```

Exemple : find sur une collection

```
'use strict'
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect(url, function(err, db) {
  console.log("Connecté");

  var Songs = db.collection('songs');

  Songs.find({year: {$gt: 2010}}).toArray(function(err, songs) {
    console.log(songs);
    console.log(songs.length + ' songs found');
    db.close();
  });
})
```

Callbacks



Exemple : find sur une collection

```
'use strict'
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect(url, function(err, db) {
  console.log("Connecté");

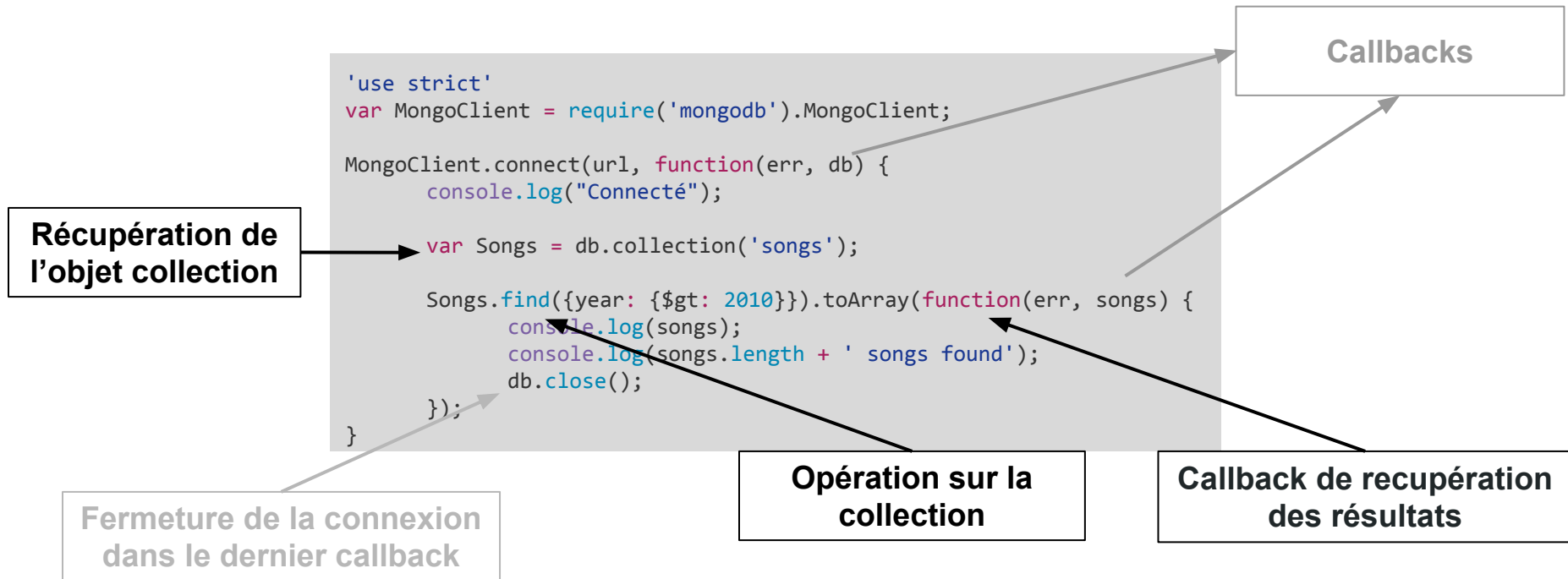
  var Songs = db.collection('songs');

  Songs.find({year: {$gt: 2010}}).toArray(function(err, songs) {
    console.log(songs);
    console.log(songs.length + ' songs found');
    db.close();
  });
})
```

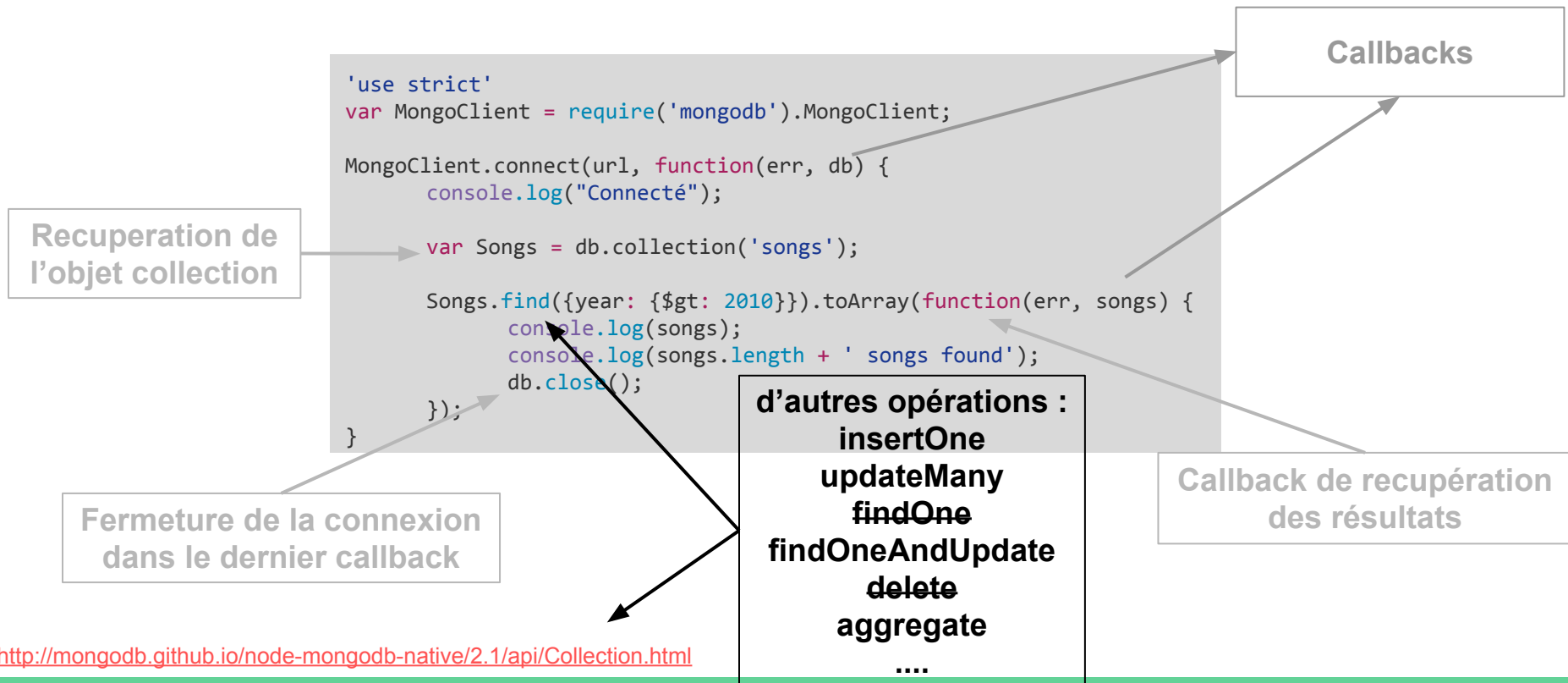
Callbacks

Fermeture de la connexion
dans le dernier callback

Exemple : find sur une collection



Exemple : find sur une collection



API

MongoDB	Driver
find	<code>find(query) → returns Cursor</code> <code><i>findOne(query, options, callback) → deprecated</i></code>
insert	<code><i>insert(docs, options, callback) → deprecated</i></code> <code>insertOne(doc, options, callback),</code> <code>insertMany(docs, options, callback),</code>
update	<code><i>findAndModify → deprecated</i></code> <code>findOneAndUpdate, findOneAndReplace</code> <code><i>update → deprecated</i></code> <code>updateOne(filter, update, options, callback)</code> <code>updateMany(filter, update, options, callback)</code>
remove	<code><i>remove(selector, options, callback) → deprecated</i></code> <code>delete(filter, options, callback),</code> <code>deleteMany(filter, options, callback),</code> <code><i>findAndRemove(query, sort, options, callback) → deprecated</i></code> <code>findOneAndDelete(filter, options, callback)</code>

Faker

npm install faker --save

```
const faker      = require('faker');
const moment     = require('moment');
const MongoClient = require('mongodb').MongoClient;
let product = {
  name:          faker.commerce.productName(),
  createdAt:     moment(faker.date.past()).format(),
  companyName:   faker.company.companyName(),
  shortDescription: faker.lorem.sentence(),
  imageUrl:      faker.image.imageUrl(),
  url:           faker.internet.url()
};

MongoClient.connect('mongodb://url', (err, db) => {
  db.collection('products').insertOne(product)
    .then(product => { console.log(product); })
    .catch(err => { console.log(err); })
    .then(() => db.close())
  ;
}):
```



<https://github.com/marak/Faker.js/>

TP.

TROISIÈME PARTIE

DATABASE DESIGN

BDD Relationelle

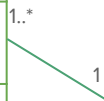
id	title	artist	album	year	bpm
1	Viva la Vida	Coldplay	Viva la Vida	2008	131.0
2	Formidable	Stromae	Racine carrée	2013	NULL
3	Violet Hill	Coldplay	Viva la Vida	2008	152.1
4	Fix You	Coldplay	X&Y	2005	111.6
5	She Will Be Loved	Maroon 5	Songs About Jane	2002	102.0
6	Alors on danse	Stromae	Cheese	2010	120.0

BDD Relationnelle: 3ème forme normale

id	title	bpm	album_id
1	Viva la Vida	131.0	1
2	Formidable	NULL	2
3	Violet Hill	152.1	1
4	Fix You	111.6	3
5	She Will Be Loved	102.0	4
6	Alors on danse	120.0	5

id	artist
1	Coldplay
2	Stromae
3	Maroon 5

id	name	year	artist_id
1	Viva la Vida	2008	1
2	Racine carrée	2013	2
3	X&Y	2005	1
4	Songs About Jane	2002	3
5	Cheese	2010	1



Sur mongo...

- pas des jointures, pas des contraintes ni transactions comme les RDBMS
- mais accepte de documents riches : documents imbriquées
- On doit plutôt réfléchir à :
 - quels morceaux de données sont utilisées ensembles
 - lesquels sont lues le plus souvent

Application Driven Schema

D'où sort le schéma ?

Specifications données par le client

Maquettes sur les vues qu'on doit construire (ceci permet de voir les données qui vont ensemble)

Deux approches sont possibles :

Embedded Design : Privilège les données au sein d'un même document

Separated Collections Design : plus familier au système des bases relationnelles

Embedded Design

OPTION 1 Collection songs

```
{
  "title" : "Alors on danse",
  "artist": {
    "name" : "Stromae",
    "type" : "SINGER"
  },
  "album" : {
    "name" : "Cheese",
    "year" : 2010
  },
  "bpm" : 120
},
{
  "title" : "She Will Be Loved",
  "artist": {
    "name" : "Maroon 5",
    "type" : "GROUP",
    "singers" : ["Adam Levine", "Mickey Madden", "..."]
  },
  "album" : {
    "name" : "Songs About Jane",
    "year" : 2002
  },
  "bpm" : 102
}
```

OPTION 2 Collection albums

```
{
  "name" : "Viva la Vida",
  "year" : 2008
  "artist": {
    "name" : "Coldplay",
    "type" : "GROUP",
    "singers" : ["Chris Martin", "Guy Berryman", "..."]
  },
  "titles": [
    {
      "name" : "Viva la Vida",
      "track": 7,
      "bpm" : 120
    },
    {
      "name" : "Violet Hill",
      "track": 8,
      "bpm" : 152.1
    }
  ]
}
```

Separated Collection Design

albums

```
{
  "id" : 1,
  "name" : "Cheese",
  "year" : 2010,
  "artist_id": 1
},
{
  "id" : 2,
  "name" : "Songs About Jane",
  "year" : 2002,
  "artist_id": 2
}
```

songs

```
{
  "title" : "Alors on danse",
  "album_id" : 1,
  "bpm" : 120
},
{
  "title" : "She Will Be Loved",
  "album_id" : 2,
  "bpm" : 102
}
```

artists

```
{
  "id" : 1,
  "name" : "Stromae",
  "type" : "SINGER"
},
{
  "id" : 2,
  "name" : "Maroon 5",
  "type" : "GROUP",
  "singers" : ["Adam Levine", "Mickey Madden", "..."]
}
```

Separated Collection Design

albums

```
{
  "id" : 1,
  "name" : "Cheese",
  "year" : 2010,
  "artist_id": 1
},
{
  "id" : 2,
  "name" : "Songs About Jane",
  "year" : 2002,
  "artist_id": 2
}
```

```
graph LR
    subgraph albums
        A1["{ 'id': 1, 'name': 'Cheese', 'year': 2010, 'artist_id': 1 }"]
        A2["{ 'id': 2, 'name': 'Songs About Jane', 'year': 2002, 'artist_id': 2 }"]
    end
    subgraph songs
        S1["{ 'title': 'Alors on danse', 'album_id': 1, 'bpm': 120 }"]
        S2["{ 'title': 'She Will Be Loved', 'album_id': 2, 'bpm': 102 }"]
    end
    subgraph artists
        AR1["{ 'id': 1, 'name': 'Stromae', 'type': 'SINGER' }"]
        AR2["{ 'id': 2, 'name': 'Maroon 5', 'type': 'GROUP', 'singers': ['Adam Levine', 'Mickey Madden', '...'] }"]
    end
    A1 -- "id" --> AR1
    A1 -- "artist_id" --> AR1
    S1 -- "album_id" --> A1
    S2 -- "album_id" --> A2
```

songs

```
{
  "title" : "Alors on danse",
  "album_id" : 1,
  "bpm" : 120
},
{
  "title" : "She Will Be Loved",
  "album_id" : 2,
  "bpm" : 102
}
```

artists

```
{
  "id" : 1,
  "name" : "Stromae",
  "type" : "SINGER"
},
{
  "id" : 2,
  "name" : "Maroon 5",
  "type" : "GROUP",
  "singers" : ["Adam Levine", "Mickey Madden", "..."]
}
```

Separated Collection Design

albums

```
{
  "id" : 1,
  "name" : "Cheese",
  "year" : 2010,
  "artist_id": 1
},
{
  "id" : 2,
  "name" : "Songs About Jane",
  "year" : 2002,
  "artist_id": 2
}
```

songs

```
{
  "title" : "Alors on danse",
  "album_id" : 1,
  "bpm" : 120
},
{
  "title" : "She Will Be Loved",
  "album_id" : 2,
  "bpm" : 102
}
```

artists

```
{
  "id" : 1,
  "name" : "Stromae",
  "type" : "SINGER"
},
{
  "id" : 2,
  "name" : "Maroon 5",
  "type" : "GROUP",
  "singers" : ["Adam Levine", "Mickey Madden", "..."]
}
```

Problème: vu qu'on n'a pas des joins, on va devoir faire les requêtes nous mêmes pour charger les infos des autres collections via les ids.

Consistance (Cohérence) de données

N'ayant pas ni des constraints, ni des foreign key, ni des joins...
Quelle est la garantie que les données soient cohérentes dans la base ?

AUCUNE !

Elle doit être garantie par la conception du développeur.

- Embedded Desing : Vu que les données sont imbriquées c'est une sorte de join
- Separated Collections : Il faut faire chaque enregistrement à part et valider son succès.

Relations 1:1

Pays - Capitale

OPTION 1



OPTION 2



OPTION 3



OPTION 4



countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": "Paris"
}
```

capitals

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000"
}
```

countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000
}
```

capitals

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": "France"
}
```

countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": {
    "name": "Paris",
    "population": 12405000,
    "zipcode": "75000"
  }
}
```

capitals

```
{
  "_id": "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": {
    "name": "France",
    "continent": "Europe",
    "population": 67222000
  }
}
```

Relations 1:1

OPTION 1



```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": "Paris"
}
```

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000"
}
```



Pays - Capitale

OPTION 2



```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000
}
```

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": "France"
}
```



Laquelle choisir ?

OPTION 3



OPTION 4



```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": {
    "name": "Paris",
    "population": 12405000,
    "zipcode": "75000"
  }
}
```

```
{
  "_id": "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": {
    "name": "France",
    "continent": "Europe",
    "population": 67222000
  }
}
```

PLUSIEURS CRITÈRES

Relations 1:1

OPTION 1



Pays - Capitale

OPTION 2



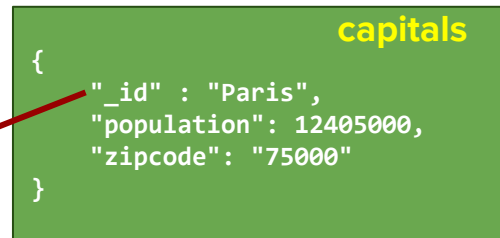
Laquelle choisir ?

CRITÈRE 1

Frequènce d'accès aux données...

Cas 1: On affiche rarement la capitale avec un pays : **OPTION 1**

Cas 2: On affiche un pays toujours avec sa capitale : **OPTION 3**



OPTION 3



OPTION 4



Relations 1:1

OPTION 1



Pays - Capitale

OPTION 2



Laquelle choisir ?

CRITÈRE 2

Tailles des items...

Cas 3: On stocke beaucoup d'infos sur un pays et une capitale :

OPTION 3 ❌ OPTION 1 ✅

OPTION 4 ❌ OPTION 2 ✅

countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": "Paris"
}
```

capitals

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000"
}
```

countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000
}
```

capitals

```
{
  "_id" : "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": "France"
}
```

OPTION 3



OPTION 4



countries

```
{
  "_id" : "France",
  "continent" : "Europe",
  "population": 67222000,
  "capitalCity": {
    "name": "Paris",
    "population": 12405000,
    "zipcode": "75000"
  }
}
```

capitals

```
{
  "_id": "Paris",
  "population": 12405000,
  "zipcode": "75000",
  "country": {
    "name": "France",
    "continent": "Europe",
    "population": 67222000
  }
}
```

Relations 1:1

OPTION 1



Pays - Capitale

OPTION 2



Laquelle choisir ?

CRITÈRE 3

Opérations Atomiques

Cas 4: On souhaite mettre à jour le pays avec sa capitale au même temps

OPTION 3 ✓

OPTION 1 ✗

OPTION 4 ✓

OPTION 2 ✗



OPTION 3



OPTION 4



Relations 1:Many

OPTION 1



Formation - Étudiant

```
{  
  program  
  "_id" : "LDPW",  
  "domain" : "informatics",  
  "students": [  
    ...  
  ]  
}
```

Relations 1:Many

OPTION 1



```
{  
  program  
  "_id" : "LDPW",  
  "domain" : "informatics",  
  "students": [  
    ...  
  ]  
}
```



Beaucoup d'
étudiants
dans l'array

Formation - Étudiant

OPTION 2



```
{  
  students  
  "_id": "1",  
  "name": "Thomas Cook",  
  "prom": 2013,  
  "course": {  
    "_id" : "LDPW",  
    "domain" : "informatics"  
  }  
}
```

Relations 1:Many

OPTION 1



```
{  
  "_id": "LDPW",  
  "domain": "informatics",  
  "students": [  
    ...  
  ]  
}
```



Beaucoup d'
étudiants
dans l'array

Formation - Étudiant

OPTION 2



```
{  
  "_id": "NDI_001",  
  "name": "Thomas Cook",  
  "prom": 2013,  
  "course": {  
    "_id": "LDPW",  
    "domain": "informatics"  
  }  
}
```



Duplication
de données
pour la
formation

OPTION 3



```
{  
  "_id": "LDPW",  
  "domain": "informatics",  
  "type": "license",  
  "duration": 1  
}
```



```
{  
  "_id": "NDI_001",  
  "name": "Thomas Cook",  
  "prom": 2013,  
  "course": "LDPW"  
}
```

Relations 1:Many

Formation - Étudiant

OPTION 1



```
{  
  "_id" : "LDPW",  
  "domain" : "informatics",  
  "students": [  
    ...  
  ]  
}
```



Beaucoup d'
étudiants
dans l'array

OPTION 2



```
{  
  "_id": "NDI_001",  
  "name": "Thomas Cook",  
  "prom": 2013,  
  "course": {  
    "_id" : "LDPW",  
    "domain" : "informatics"  
  }  
}
```



Duplication
de données
pour la
formation



OPTION 3



```
{  
  "_id": "LDPW",  
  "domain": "informatics",  
  "type": "license",  
  "duration": 1  
}
```



```
{  
  "_id": "NDI_001",  
  "name": "Thomas Cook",  
  "prom": 2013,  
  "course": "LDPW"  
}
```

Relations 1:Few

Blog Post - Comments

```
{
  "_id" : "1",
  "title" : "My Last Trip",
  "date" : "2015-09-15 20:30:02.684Z",
  "body" : "...",
  "comments": [
    {
      "user": "Tim Knight",
      "body": "I was also there!",
      "date" : "2015-09-16 10:20:52.875Z"
    },
    {
      "user": "Matt Davis",
      "body": "Me too!",
      "date" : "2015-09-16 11:41:35.457Z"
    },
    ...
  ]
}
```

Pas beaucoup
des comments
dans l'array

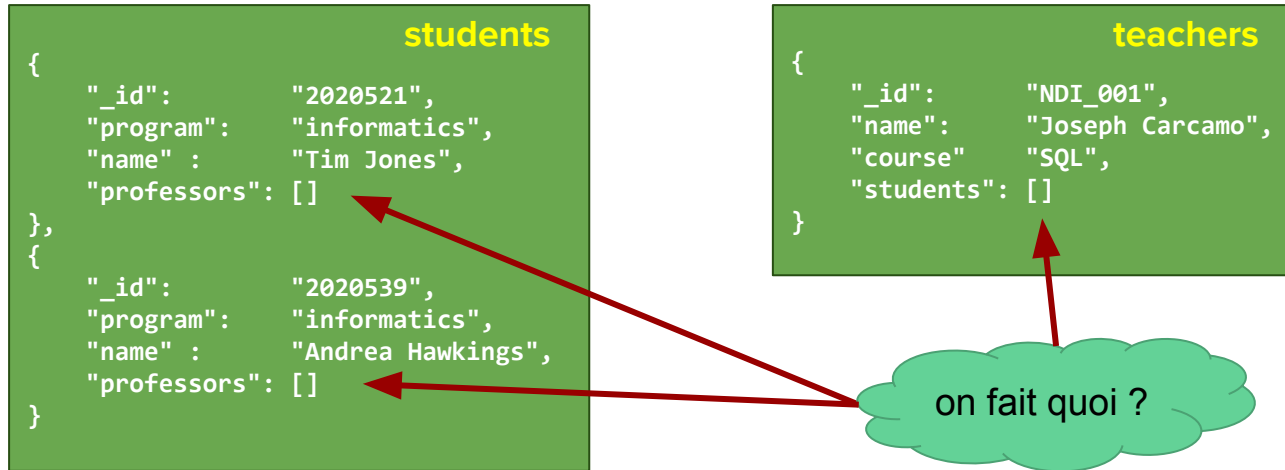


Pas de
duplication de
données pour
la formation



Relation Many:Many

Blog Étudiant - Enseignant



Relation Many:Many

Blog Étudiant - Enseignant

```
                                students
{
  "_id":      "2020521",
  "program":  "informatics",
  "name" :    "Tim Jones",
  "professors": ["NDI_001"]
},
{
  "_id":      "2020539",
  "program":  "informatics",
  "name" :    "Andrea Hawkings",
  "professors": ["NDI_001"]
}
```

```
                                teachers
{
  "_id":      "NDI_001",
  "name":     "Joseph Carcamo",
  "course"    "SQL",
  "students": ["2020521", "2020539"]
}
```

TP.

QUATRIÈME PARTIE
(HOMEWORK)

Mongoose

Mongoose



ODM (Object Document Mapper)

version 4.7.7

npm install mongoose --save

```
'use strict'

var mongoose = require('mongoose');

var mongolabStringConnexion = 'mongodb://localhost:27017/mymusic';

mongoose.connect(mongolabStringConnexion);
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function callback () {
  console.log('Connexion establish to ' + mongolabStringConnexion);
});
```

Mongoose



Definition du schéma

Types : String, Number, Date, Boolean, Buffer, ObjectId, Mixed, Array

```
'use strict'
var mongoose = require('mongoose');
var schema = mongoose.Schema({
  name: String,
  updated: { type: Date, default: Date.now }
  age: { type: Number, min: 18, max: 65, required: true }
  _someId: Schema.Types.ObjectId,
  arrayOfString: [String],
  nested: {
    stuff: { type: String, lowercase: true, trim: true }
  }
});

var Thing = mongoose.model('Thing', schema);
```

Mongoose



Operations avec le Model

```
Thing.find({name : 'toto'}, function(err, things) {...});
Thing.findOne({name : 'toto'}, function(err, thing) {...});
Thing.findById("55328bd3f238ef5f0de2ad33", function(err, thing) {...});

Thing.create(newThing, function(err, thing) {...});
Thing.update({name : 'toto'}, {age: 50}, {multi: true}, function(err, raw) {...});
Thing.remove({name : 'toto'}, function(err, thing) {...});

Thing.count(query, function(err, total){...});
Thing.aggregate({pipeline}, function(err, res){...});
```



M101JS: MONGODB FOR NODE.JS DEVELOPERS

NEXT SESSION

Start: 04 Aug 2015 at 17:00 UTC

End: 22 Sep 2015 at 17:00 UTC

REGISTER

<https://university.mongodb.com/courses/M101JS/about>