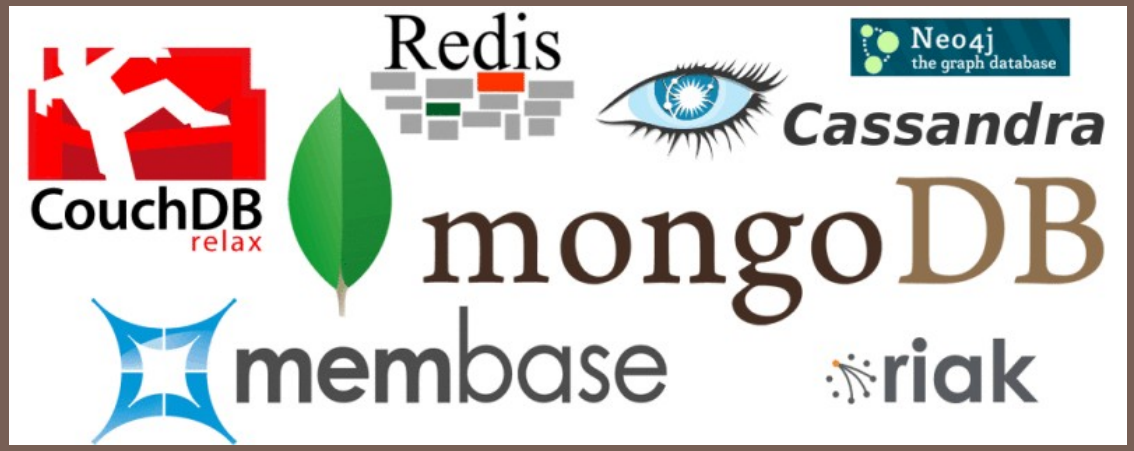


NoSQL : mongoDB

2021/2022

Frédéric Moal BD & SI

Plan de la présentation

- MongoDB
- JSON
- Requêtes en client Mongo
- Map-Reduce

MongoDB

Pourquoi MongoDB ?

- Une BDD mature
  - Doodle, github, Ebay, McAfee, Adobe, Craiglist, ...
  - Une doc très fournie et une grande communauté
- Adaptation facile
  - Orientée document - format JSON
  - SQL > MongoDB : Easyyy ! mouaip...
  - Utilise Javascript
- Scalable
  - MapReduce en natif ou avec le connecteur ;
  - Réplication, sharding automatique
- Et plein d’autres trucs cools ...

JSON

Rapide rappel

JSON

JSON « **J**ava**S**cript **O**bject **N**otation » est un format d'échange de données, facile à lire par un humain et interpréter par une machine.

Basé sur JavaScript, il est complètement indépendant des langages de programmation mais utilise des conventions qui sont communes à toutes les langages de programmation (C, C++, Perl, Python, Java, C#, VB, JavaScript,….)

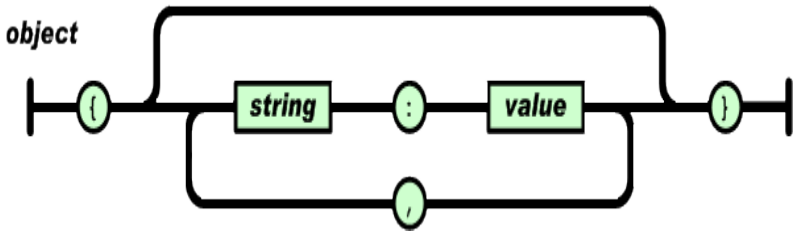
Deux structures :

- Une collection de clefs/valeurs : Object
- Une collection ordonnée d'objets : Array

JSON

Objet

Commence par un « { » et se termine par « } » et composé d'une liste non ordonnée de paire clefs/valeurs. Une clef est suivie de « : » et les paires clef/valeur sont séparés par « , »

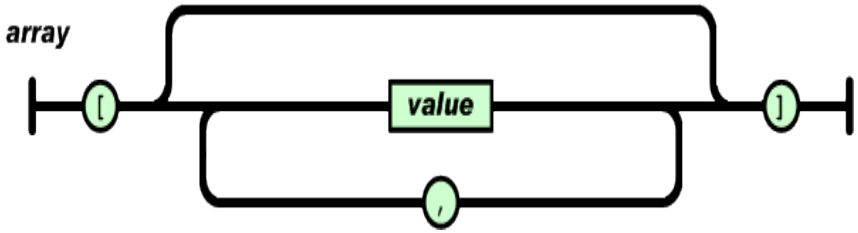



```
{ "id": 51,
"nom": "Mathematiques 1", "resume":
"Resume of math ", "isbn": "123654",
"categorie":
     {
     "id": 2, "nom": "Mathematiques",
     "description": "Description of
     mathematiques "
     },
"quantite": 42,
"photo": ""
}
```

JSON

ARRAY

Liste ordonnée d'objets commençant par « [« et se terminant par «] », les objets sont séparés l'un de l'autre par « , ».

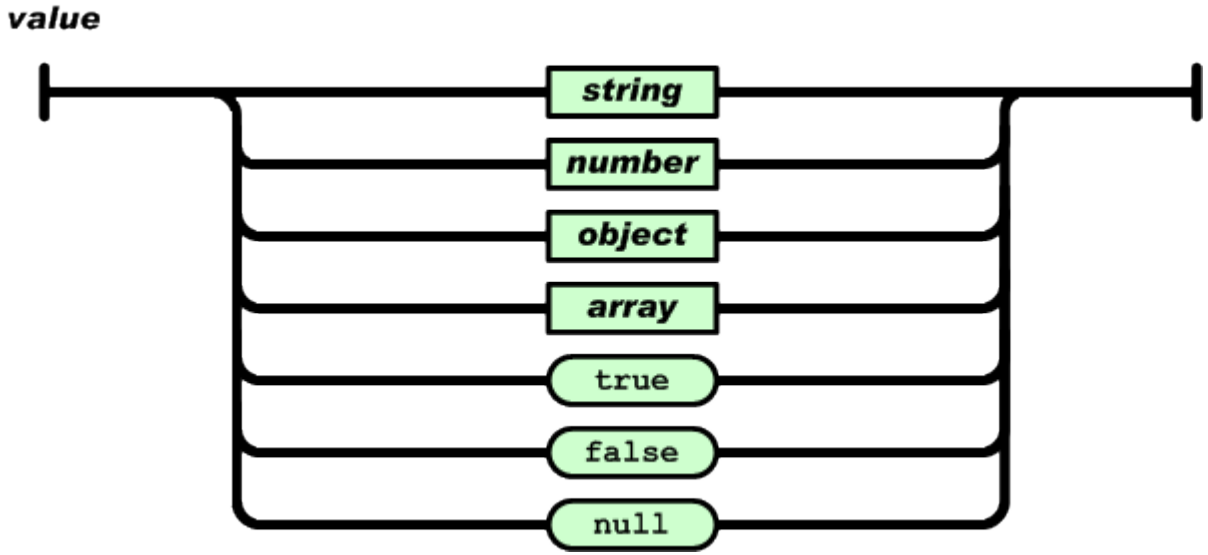



```
[
     { "id": 51,
     "nom": "Mathematiques 1",
     "resume": "Resume of math ",
     "isbn": "123654",
     "quantite": 42,
     "photo": ""
     },
     { "id": 102,
     "nom": "Mathematiques 1",
     "resume": "Resume of math ",
     "isbn": "12365444455",
     "quantite": 42,
     "photo": ""
     }
]
```

JSON

Value

Un objet peut être soit un string entre « ""» ou un nombre (entier, décimal) ou un boolean (true, false) ou null ou un objet.



MongoDB

retour

MongoDB

- MongoDB est fourni avec un client en ligne de commande
- 2 parties :
  - le serveur, écoute [en général] sur le port 20017
  - Le client qui s’y connecte

MongoDB avec docker

- On lance le serveur :

  docker run --name mongoserver -d mongo:tag

- On lance un client (mongoDB shell) :

  docker run -it --link mongoserver:mongoserv --rm mongo mongo --host mongoserv test

MongoDB avec docker

- Utilitaire externe : mongoimport (export) pour importer un jeu de données complet en JSON, csv, ...

MongoDB - les bases

- Le vocabulaire
  - database => database
  - table => collection
  - entrée, ligne, tuple => document
- Création, insertion
  - use db1
  - db.createCollection(”collection_name”)
  - db.collection_name.insert({field1 : ”value_string”, field2 : value_int, ..., fieldN: [arrayV1, ..., arrayVn]})

MongoDB - les bases

- Exemple :

  use *maBD;*

  db.createCollection('*users*');

  db.users.save ( ); //pas de quotes

- Requêtes orientées documents

  > db.users.find({ login : "james" });

  > db.users.find({ address.city : "London" }, {name : 1});

  > db.users.find({ age : { $gt : 40 } });

  //$gt, $gte, $lt, $lte, $ne, $in, $nin, $or, $and, $exists, $type…

  > db.users.find({ name : { $regex : "james", $options : "i" } });

  > db.users.count();

```
{
"_id": 100,
"name": "James Bond",
"login": "james",
"age": 50,
"address": {
"street" : "42 Class Street",
"city": "London"
}
}
```

MongoDB - MAJ

- Pas de transactions
- Format général : update({filtre},{operations}
- Opérations : modifications atomiques de documents
  - $set – Modifie une valeur
  - $unset – Supprime un attribut
  - $inc – Incrément
  - $push – Ajout dans un tableau
  - $pushAll – Plusieurs valeurs dans un tableau
  - $pull – Supprimer une valeur de tableau
  - $pullAll – Supprimer plusieurs valeurs

MongoDB - MAJ

```
> db.users.update( { “_id” :
ObjectId("4efa8d2b7d284dad101e4bc7") } ,
  { “$inc” : { “age” : 1 } }
> db.collection_name.update( { _id: 1 },
{
$inc: { field1: 5 },
$set: { field2: ”ABC123” }
} )
```

MongoDB - suppression

- Suppression : remove({filtre})

```
db.collection_name.remove({ })
//Delete the collection
db.collection_name.remove({field1 :
   value}) //Delete doc that match
```

MongoDB – les requêtes

- Exemple - Select, Projection

  db.collection_name.find({ }, {field1: 1})

  db.collection_name.find({ }, {field2: 0})

- Exemple – Where

```
db.collection_name.find({field: {$gt:v1,
   $lt:v2} });
// where v1 < field < v2
db.collection_name.find({field: value});
// where field == value
```

MongoDB – les requêtes

- Exemple - Select, Projection

  db.collection_name.find({ }, {field1: 1})

  db.collection_name.find({ }, {field2: 0})

- Exemple – Where

```
db.collection_name.find({field: {$gt:v1,
   $lt:v2} });
// where v1 < field < v2
db.collection_name.find({field: value});
// where field == value
```

MongoDB – les requêtes

- Order By équivalent

  db.collection_name.find().sort({field1: 1, field2: -1}) // Order by field1 ASC, field2 DESC

- Limit

  db.collection_name.find().limit(2)

  // les deux premiers docs

MongoDB – aggregate

- db.coll.aggregate() prend en paramètre un pipeline d'opération = array
- Le résultat de chaque opération (stage) est envoyé à l'opération suivante, jusqu'à la fin du tableau = résultat final
- Le résultat est un cursor : affiché par défaut, ou stocké (var variable=)

MongoDB – aggregate

- Syntaxe

```
db.collname.aggregate( [
    { ... },
    { ... }
] )
```

- Opérations principales :
  - $project : projection
  - $group : group by
  - $match : select
  - $unwind : applatissement array

MongoDB – aggregate

- $project
  - Inclusion ou non des champs du stage précédent (field:1 ou field:0)
  - Ajout de champs

    field : expression
  - Exemple :

```
db.books.aggregate( [
    { $project: { title:1, author: 1,
lastName: "$author.last" } }
] )
```

MongoDB – aggregate

- $group
  - _id : expression, group by expression
  - + champs avec accumulator
  - Min, max, sum, avg, first, push...

MongoDB – aggregate

- $unwind sur un tableau
  - Duplique les entrées par élément dans le tableau
  - Eg :

```
{ "_id" : 1, "item" : "ABC1", sizes: [ "S", "M", "L"] }

db.inventory.aggregate( [ { $unwind : "$sizes" } ] )

{ "_id" : 1, "item" : "ABC1", "sizes" : "S" }

{ "_id" : 1, "item" : "ABC1", "sizes" : "M" }

{ "_id" : 1, "item" : "ABC1", "sizes" : "L" }
```

MongoDB – les requêtes

- Exemple - Group By

```
db.collection_name.aggregate( [
    {$group : {_id : ”$field”} }
] ) //
```

- Group by field

```
db.collection_name.aggregate( [
   {$group : {_id : ”$user”, count: {$sum: 1} } } ,
   {$match : {count : {$gte : 10} } } }
] ) // Group by field having count > 10

db.collection name.aggregate( [
{$group : {_id : ”$user”, count: { $sum: 1} } },
{$match : {count : {$gte : 10 } } },
{$sort : {count : -1} } ] )
```

MongoDB – les requêtes

db.users.aggregate(**[{$sort : {age : 1}}]);**

db.users.aggregate(**[{$project : {login : 1, name:1}}]);**

```
db.users.aggregate([
  {$match : {address.city : “London”}},
  {$sort : {age : 1}
]);
db.users.aggregate([
{$group : {_id : “$address.city”, number : {$sum : 1}},
{$sort : {number : 1}]);
```

MongoDB – les requêtes

- Stockage possible des résultats dans des variables
- Attention au type de la variable ! Défaut = cursor = 1 seule lecture !
- Sinon next() si vous savez que vous avez 1 seul document

MongoDB – les requêtes

- Exemple : carte avec max de transaction

```
var searchMax = db.cartes.aggregate( [
    {$project:{ _id:0, nb:{$size:"$ventes"}}},
    {$group:{_id:0, maxNb:{$max:"$nb"}}}
    ]).next().maxNb
db.cartes.aggregate([
    {$project:{ _id:"$nomJoueur", nb:{$size:"$ventes"}}},
    {$match:{nb:{$gte:searchMax}}}
    ])
```

MongoDB – les requêtes

- Voir aussi :
  - distinct : valeurs distinctes
  - .concat : chainage
  - .filter : lambdas !

MongoDB – Index

- Comprendre l'exécution d'une requête : explain()

```
> db.articles.find({"auteurs.nom": "Dupont"}).explain()
{
"cursor" : "BasicCursor",
"isMultiKey" : false,
"n" : 1,
"nscannedObjects" : 2,
"nscanned" : 2,
"nscannedObjectsAllPlans" : 2,
"nscannedAllPlans" : 2,
"scanAndOrder" : false,
"indexOnly" : false,
"nYields" : 0,
"nChunkSkips" : 0,
"millis" : 0,
"indexBounds" : {
},
"server" : "mongo1:27017"
}
```

MongoDB – Index

- Créer un index avec createIndex

```
> db.articles.createIndex({"auteurs.nom": 1})
> db.articles.find({"auteurs.nom": "Dupont"}).explain()
{
"cursor" : "BtreeCursor auteurs.nom_1",
"isMultiKey" : true,
"n" : 1,
"nscannedObjects" : 1,
"nscanned" : 1,
"nscannedObjectsAllPlans" : 1,
"nscannedAllPlans" : 1,
"scanAndOrder" : false,
"indexOnly" : false,
"nYields" : 0,
"nChunkSkips" : 0,
"millis" : 0,
"indexBounds" : {
"auteurs.nom" : [
[
"Dupont",
"Dupont"
]
]
},
"server" : "mongo1:27017"
}
```

- En java : collection.createIndex(new Document("i", 1));

MongoDB – Types spécifiques

- Support natif pour des coordonnées géographiques
- Exemple :

```
{
  "_id" : ObjectId("4d291187888cec7267e55d24"),
  "city" : "New York City",
  "loc" : {
    "lon" : -73.9996
    "lat" : 40.7402,
  },
  "state" : "New York",
  "zip" : 10011
}
```

Création d'un index géographique :

```
> db.zips.createIndex({loc: '2dsphere'})
```

MongoDB – Types spécifiques

- Récupération des k plus proches voisins

```
> db.zips.find({'loc': {$near: [ -73.977842, 40.752315 ]}}).limit(3)
```

- Récupération des enregistrements à l'intérieur d'un disque

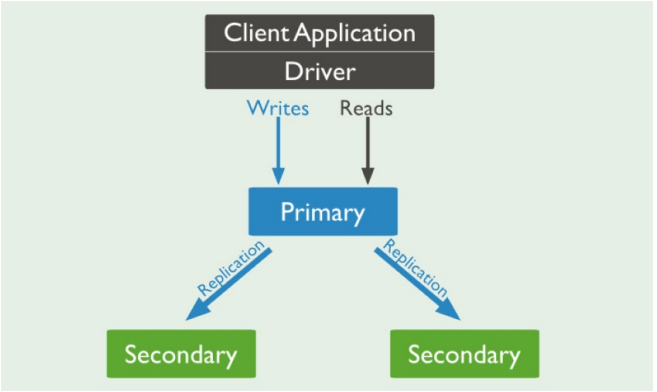
```
> center = [-73.977842, 40.752315]
> radius = 0.011
> db.zips.find({loc: {$within: {$center: [ center, radius ] }}})
```

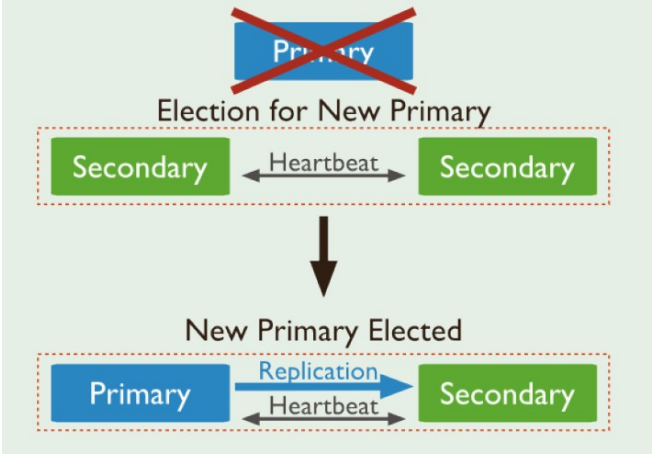
- Récupération des enregistrements à l'intérieur d'un rectangle

```
> lower_left = [-73.977842, 40.752315]
> upper_right = [-73.923649, 40.762925]
> db.zips.find({loc: {$within: {$box: [ lower_left, upper_right ] }}})
```

MongoDB – Replica Set

- Réplication d'un serveur
  - Asynchrone
    - Primary server : écritures
    - Secondary servers : lectures
      - Mise à jour via oPlog (fichier de log)
  - Distribue la charge de lecture (requêtes)
  - Tolérance aux pannes
    - Élection d'un nouveau serveur primaire
    - Besoin d'un serveur arbitre





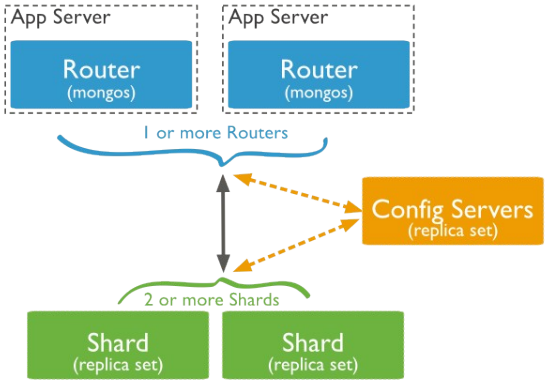
MongoDB – Replica Set

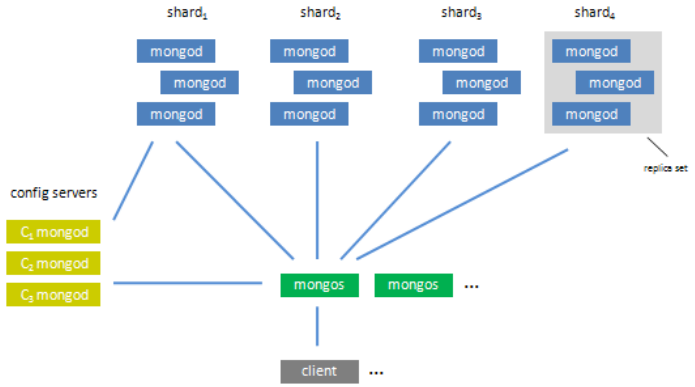
- Réplication d'un serveur
  - Avec Docker : on lance plusieurs serveurs, avec l'option --replSet monrs
  - On se connecte sur le primary avec mongo shell (client)
  - On lance la config du RS :

```
rs.initiate()

rs.add("ip:port")

…

rs.status()
```

MongoDB - Sharding

- Distribution sur un cluster de machines
  - Equilibrage de charge des données
  - Combinaison avec *Replica Set*
  - Nécessite :
    - *Config Servers*
    - *Mongos (routeur)*
  - Partitionnement sur attribut
    - Ranged-based
    - Hash-based





MongoDB - Monitoring

- Commande : mongostat
- Commande : mongotop
- ~~Interface HTTP : http://localhost:28017~~

MongoDB - cohérence

- Unacknowledged
  - On ne sait pas si le requête d'écriture est reçue
- Acknowledged
  - Mode **par défaut**
  - On ne sait que la requête d'écriture est reçue, on ne sait pas si elle est persistée
- Journaled
  - La requête est reçue et persistée dans le journal de la DB, elle est donc persistée, mais pas sur tout le cluster
- Replica Acknowledged
  - La requête d'écriture est forcément propagée aux noeuds du cluster répliqué

MongoDB – Outils

- Répertoire bin/
- **mongod : Lancement du serveur (daemon)**
- **mongo : shell pour exécuter les commandes**
- **mongoimport *: importation d'un fichier de données***
- **mongostat : Status du process en cours (inserts,** updates, queries, commands, memory…)
- **mongotop : Temps réparti en lecture vs écriture**
- **mongos : Service de routage (sharding)**

MongoDB

JOINTURES !

MongoDB - JOIN

- On peut [tout de même] réaliser des jointures : utilisation de l’opérateur $lookup:
- Toutes les opérations précédentes se focalisaient sur 1 seule collection
- Pour faire un JOIN entre deux collections il faut utiliser :
  - Une agrégation
  - Avec l’opérateur $lookup

MongoDB - JOIN

- utilisation de l’opérateur $lookup:

```
db.co1.aggregate([
{"$lookup" : {
"localField" : "title",          // champ de la collection co1
"from" : "co2",                  //  collection 2
"foreignField" : "title_book",   //  ’title_book’’ : champ de co2
"as" : "joined" }      // nom du champs AJOUTE
},
{"$out" : "resJoin"}
]);
```

Dans le pipeline, on génère un document égal aux documents de co1 **enrichis** d’un champ ‘’joined’’ contenant **un tableau** de tous les enregistrements de co2 de même clé

MongoDB - JOIN

- Résultat :

{ ‘’_id’’ : .....,

‘’title’’ : ‘’Cours de MongoDB’’,

‘’annee’’ : ‘’2021’’,

‘’joined’’ : [

{ ‘’_id’’ : ....

‘’title_book’’ : ‘’Cours de MongoDB’’,

‘’thematique’’ : ‘’informatique’’

‘’prerequis’’ : [‘’programmation’’, ‘’SQL’’]

},

...

]

}

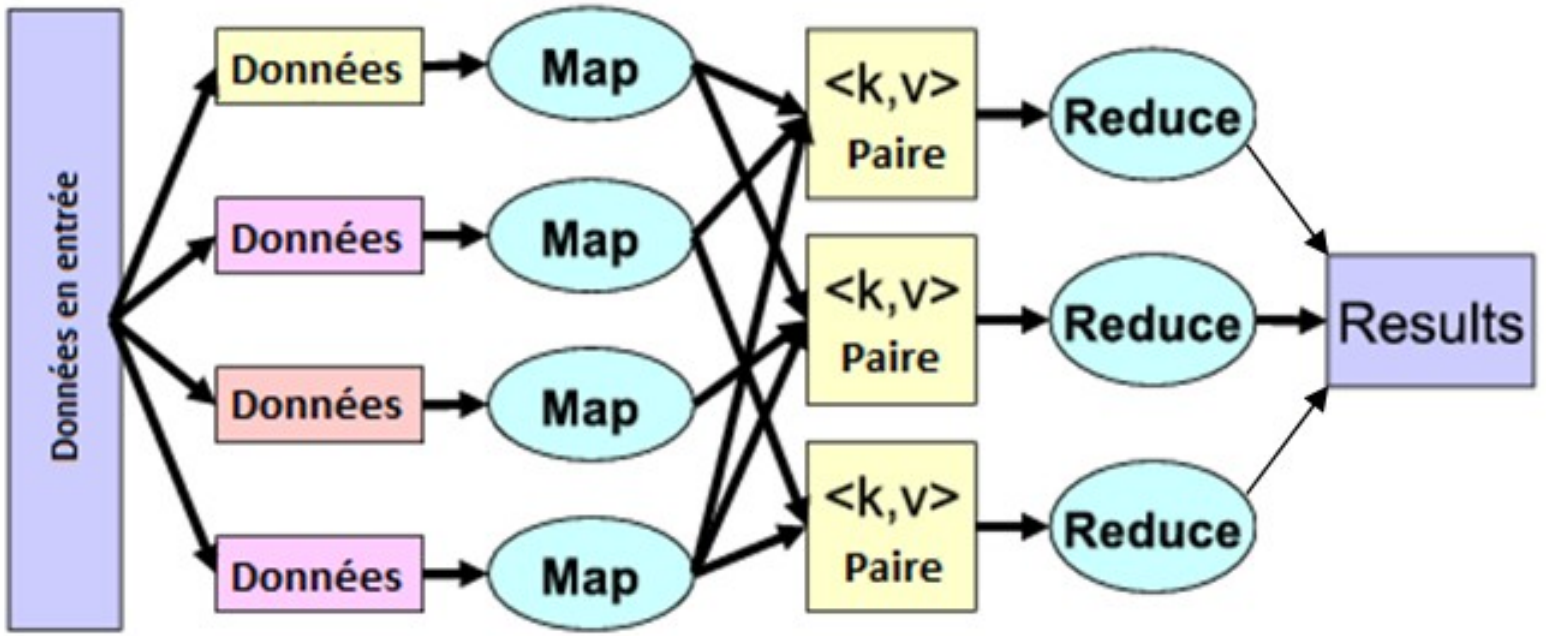
Utilisation de $unwind et $group ensuite

MongoDB

Map Reduce

MongoDB - MapReduce

- En natif



MongoDB - MapReduce

- Exemple : films & réalisateurs

- Quelle modélisation ?

MongoDB - MapReduce

- Exemple avec les films

```
{
    "title": "Pulp fiction",
    "year": "1994",
    "genre": "Action",
    "country": "USA",
    "director": "artist:3"
}

Artistes : {
_id : »artist »
            "last_name": "Tarantino",
            "first_name": "Quentin",
            "birth_date": "1963"
    }
```

MongoDB - MapReduce

- Exemple avec les films : la liste des films réalisés par chaque réalisateur
- Idée : pour chaque film, on envoie une clé-valeur, la clé étant le réalisateur, la valeur le titre du film

MongoDB - MapReduce

- Idée : pour chaque film, on envoie une clé-valeur, la clé étant le réalisateur, la valeur le titre du film

```
var mapRealisateur = function() {
        emit(this.director._id, this.title);
};
```

MongoDB - MapReduce

- Idée : pour la réduction, on se retrouve avec un réalisateur et le tableau JS de ses titres de films

```
var reduceRealisateur = function(directorId, titres) {
    var res = new Object();
    res.director = directorId;
    res.films = titres;
    return res;
};
```

MongoDB - MapReduce

- On lance :

  db.movies.mapReduce(mapRealisateur, reduceRealisateur, {out: {"inline": 1}})

```
{
        "_id" : "artist:3",
        "value" : {
                "director" : "artist:3",
                "films" : [
                        "Vertigo",
                        "Psychose",
                        "Les oiseaux",
                        "Pas de printemps pour Marnie",
                        "Fenêtre sur cour",
                        "La mort aux trousses"
                ]
        }
}
```

MongoDB - MapReduce

- Vous avez compris ?
  - Codez :

    select count(*) from movies group by genre

MongoDB - MapReduce

// Map : fonction appliquée à chaque document. Filtre et produit les éléments (clé/valeur) en sortie

var mapFunction = function () {**emit(this.address.city, this.login);}**

// Reduce : regroupe tous les documents sur la clé. Applique une fonction d'agrégat sur chaque ensemble.

var reduceFunction = function (**key, values) {**

return Array.count(values);}

//queryParam : Paramètres d'exécution

var queryParam = {query : {}, out : "**result_set"}**

//query : Permet de filtrer les documents AVANT le map, utilise l'API MongoDB. Utile pour l'utilisation des indexes

//out : collection de stockage du résultat

db.publis.mapReduce(mapFunction, reduceFunction, queryParam);

//Consulter le résultat

db.**result_set.find();**

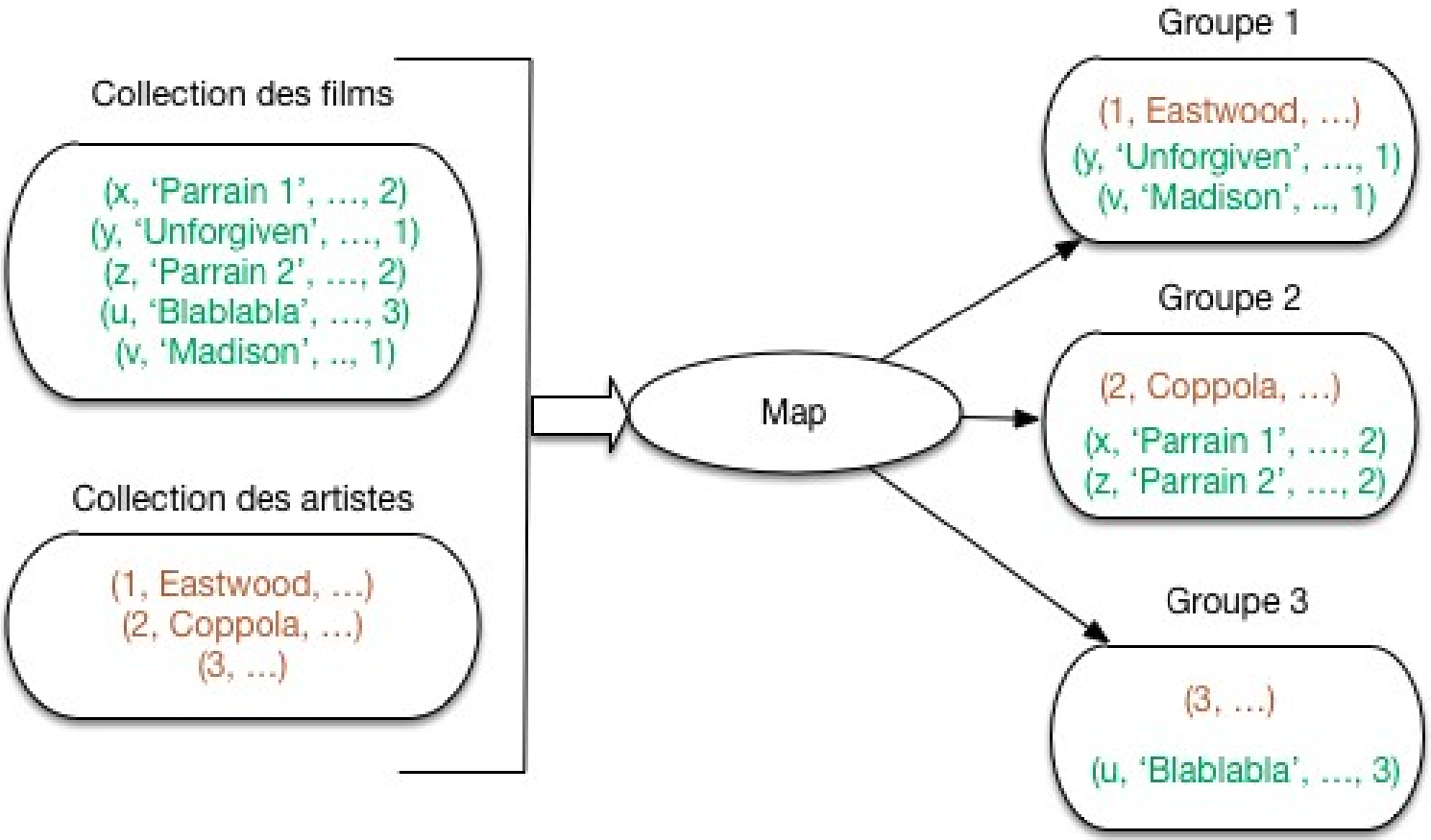
MongoDB - MapReduce

- Exemple :

```
var mapFunction = function () {
  if(this.genres.contains(“Action”)) emit(this.directors[0], this.rank);
  if(this.actors.contains(“Harrison Ford”)) emit(this.directors[0], 1);
}
var reduceFunction = function (key, values) {return Array.sum(values);}

{
"title" : "Star Wars: Episode VII",
"directors" : ["J.J. Abrams"],
"release_date" : "2015-12-16",
"genres" : ["Action","Adventure","Fantasy","Sci-Fi"],
"plot" : "A continuation of the saga created by George Lucas.",
"rank" : 168,
"year" : 2015,
"actors" : ["Mark Hamill","Harrison Ford","Carrie Fisher"]
}
```

MongoDB - MapReduce

- On peut également réaliser des jointures multi-collections, eg



MongoDB - MapReduce

- On peut également réaliser des jointures multi-collections : obligation de mélanger dans la même collection les données des 2 collections concernées :

  mongoimport -d moviesref -c jointure --file movies-refs.json –jsonArray

-

  mongoimport -d moviesref -c jointure --file artists.json --jsonArray

MongoDB - MapReduce

```
var mapJoin = function() {
  // Est-ce que la clé du document contient le mot "artist"?
  if (this._id.indexOf("artist") != -1) {
    // Oui ! C'est un artiste. Ajoutons-lui son type.
    this.type="artist";
    // On produit une paire avec pour clé celle de l'artiste
    emit(this._id, this);
  }
  else {
    // Non: c'est un film. Ajoutons-lui son type.
   this.type="film";
   // Simplifions un peu le document pour l'affichage
   delete this.summary;
   delete this.actors;
    // On produit une paire avec pour clé celle du metteur en scène
   emit(this.director._id, this);
 }
};
```

MongoDB - MapReduce

```
var reduceJoin = function(id, items) {

  var director = null, films={result: []}

  // Commençons par chercher l'artiste dans cette liste
  for (var idx = 0; idx < items.length; idx++) {
    if (items[idx].type=="artist") {
        director = items[idx];
    }
  }

  // Maintenant, 'director' contient l'artiste : on l'affecte aux films
  for (var idx = 0; idx < items.length; idx++) {
     if (items[idx].type=="film"  && director != null) {
        items[idx].director = director;
        films.result.push (items[idx]);
     }
   }
   return films;
};
```