Part III: MongoDB

MASTER SEP 2022-2023

(THANKS TO S. VIALLE – CENTRALESUPÉLEC WHO KINDLY PROVIDED ACCESS TO HIS COURSE MATERIALS)

Bart.Lamiroy@univ-reims.fr

Prerequisites

Install MongoDB

https://docs.mongodb.com/manual/

Make sure configuration is operational

https://docs.mongodb.com/manual/tutorial/getting-started/

Set `%PATH%` variables accordingly

We will be using publicly available Open Data

https://data.enseignementsup-recherche.gouv.fr

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Setting up a MongoDB Database

BASIC OPERATIONS AND COMMANDS

Basic Operation

Launch **`mongod`** (server daemon)

Launch **`mongo`** (client shell)

Within client shell

- Configure databases
- Access and query databases

Basic Operation

SHELL VIEW

```
> db
test
> show dbs
admin 0.000GB
local 0.000GB
> use db-films
switched to db-fims
> db
db-films
> show collections
```

`db` displays the current active database. By default, a virtual and empty `test` database always exists.

Other preconfigured databases are `admin` and `local`

`use` allows to switch databases (even to inexisting ones)

Basic Operation

RECORD INSERTION

```
> db.comedie.insert({titre : "Shrek", animation : true})
WriteResult({ "nInserted" : 1 })
> db.comedie.insert({titre : "Les visiteurs", acteurs :
["Clavier", "Reno"]})
WriteResult({ "nInserted" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
"titre" : "Shrek", "animation" : true }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
"titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Basic Operation

RECORD INSERTION

Standard instruction format:
db.*CollectionName*.*instruction*()

```
> db.comedie.insert({titre : "Shrek", animation : true})
WriteResult({ "nInserted" : 1 })
> db.comedie.insert({titre : "Les visiteurs", acteurs :
["Clavier", "Reno"]})
WriteResult({ "nInserted" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
"titre" : "Shrek", "animation" : true }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
"titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

Data representation format: JSON
{ label : value }

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Basic Operation

CONTROLLING DISPLAY

```
> db.comedie.find()
{ "_id" :
ObjectId("590f79cd646823f59510e489")
, "titre" : "Les visiteurs",
"acteurs" : [ "Clavier", "Reno" ] }

> db.comedie.find().pretty()
{
"_id" :
ObjectId("590f79cd646823f59510e489")
,
"titre" : "Les visiteurs",
"acteurs" : [
"Clavier",
"Reno"
]}
```

Adding **.pretty()** allows for easier to read data display

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Updating and Adding Fields

```
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
"titre" : "Shrek", "animation" : true, "date" : 2000 }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
"titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno"
] }
> db.comedie.update({titre : "Shrek"},
{$set : {date : 2001, avis : "****"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified"
: 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
"titre" : "Shrek", "animation" : true, "date" : 2001,
"avis" : "****" }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"), "titre" :
"Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
```

Solution

Add and update fields and values as a single collection using **{$set}**

Erasing Collection Content

```
> db.comedie.find()
{ "_id" : ObjectId("590f3bc1ae2448d877bbf8ea"),
"titre" : "Shrek", "animation" : true, "date" : 2001, "avis" :
"****" }
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"), "titre" :
"Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.remove({titre : "Shrek"})
WriteResult({ "nRemoved" : 1 })
> db.comedie.find()
{ "_id" : ObjectId("590f3bcaae2448d877bbf8eb"),
"titre" : "Les visiteurs", "acteurs" : [ "Clavier", "Reno" ] }
> db.comedie.remove({})
WriteResult({ "nRemoved" : 1 })
> db.comedie.find()
> show collections
comedie
```

- Selective removal of individual records
- Removal of full collection content
- Empty collection still exists

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Data Import (JSON)

mongod must be up and running for use of **mongoimport**

`--mode`

Allows for selecive adding/merging/replacing new data when existing data exists

`--jsonArray`

Specifies whether imported data is in JSON format

Multiple other options exist (*cf.* online documentation)

mongoimport is an OS command shell tool (not an internal Mongo shell instruction)

mongoimport --db *dbName* –collection *collectionName* --mode *importMode* --file *fileName.json* --jsonArray

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Data Import

When adding new data …

If new data has no `_id` field, a new `_id` will be created

If new data has `_id` field and the same `_id` already exists, import behaviour will depend on `--mode` parameter:

- `--mode` *`insert`*: raises an error on insertion
- `--mode` *`upsert`*: replaces old data by new ones
- `--mode` *`merge`*: adds new fields where appropriate and updates existing fields with new values.

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Interacting and Querying

OVERVIEW OF MONGODB COMMANDS

Javascript

```
> function fact(n) {
      if (n == 1) return 1
      else return (n*fact(n-1))
  }
> fact (2)
2
```

MongoDB client embeds a JavaScript interpreter

Allows for function definitions an execution

Querying using `find()`

Instruction **`db.collectionName.find({…},{…})`**

First argument: selection conditions

Only records statisfying conditions will be returned

Second argument (optionnal): projection conditions

Data from records to be retained after filtering

Examples :

`db.comedie.find({date: 2000})`

- All movies in `comedie` collection having been released in 2000
- Equivalent to: `SELECT * FROM comedie WHERE (date == 2000)`

`db.comedie.find({date: 2000},{titre: 1, _id: 0})`

- All movie titles in `comedie` collection having been released in 2000
- Equivalent to: `SELECT titre FROM comedie WHERE (date == 2000)`

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Querying : Access to Attributes

```
{entreprise: "Au bon produit",
 services: { direction: {etage: 5,
                         effectifs: 6}
             compatbilite: {etage: 5,
                            effectif:4}
             innovation: {etage: 4,
                          effectif: 12 }
                    …
           },
 adresse: "Nancy ARTEM",
}
```

A structured record containing structured sub-records

```
> find({"entreprise": "Au bon produit"},{"adresse": 1})

> find({entreprise: "Au bon produit"},{adresse: 1})

> find ({"entreprise": "Au bon produit"},
{"services.innovation.etage": 1})

> find ({entreprise: "Au bon produit"},{services.innovation.etage: 1})
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Querying : Access to Attributes

```
{entreprise : "Au bon produit",
 services : { direction : {etage : 5,
                           effectifs : 6}
              compatbilite : {etage : 5,
                              effectif :4}
              innovation : {etage : 4,
                            effectif 12 }
                    ….
            },
 adresse : "Nancy ARTEM",
}
```

A structured record containing structured sub-records

```
> find({"entreprise" : "Au bon produit"},{"adresse" : 1})

> find({entreprise : "Au bon produit"},{adresse : 1})

> find ({"entreprise" : "Au bon produit"},
{"services.innovation.etage" : 1})

> find ({entreprise : "Au bon produit"},{services.innovation.etage : 1})
```

Sub-field structure should be in quotes

Some Operators

Comparison:

- **$eq, $ne** ==, !=
- **$gt, $gte** >, >=
- **$lt, $lte** <, <=
- **$in, $nin** ∈

Logic:

- **$and, $or** AND, OR
- **$not, $nor** NOT, NOR

Array Test:

- **$all** array multiple value match
- **$elem** array conditions
- **$size** array size

Fields and Attibutes:

- **$exists** presence of a field
- **$type** type of an attribute

Others:

- **$mod** integer division modulo
- **$regex** regular expression
- **$text** text analysis
- **$where** record selection

Refer to on-line MongoDB documentation for further details.

AND Operators

There exist three distinct ways of expressing conjunctions of conditions:

1. Enumeration of ***independent*** conditions
2. Grouped ***dependent*** conditions
3. `$and` operator

AND Operators (1)

Enumerating serveral distinct attributes within the first argument of **`find()`** is interpreted as **`AND`**

```
db.comedie.find({pays:"France", date:2000}, {titre:1, _id:0})
```

Is equivalent to:

```
SELECT titre FROM comedie WHERE (pays="France" AND date"==2000)
```

The `AND` operator is implicit under the express condition to consider distinct attributes.

AND Operators (2)

In order to operate a conjunction of conditions on dependant attributes they need to be grouped.

```
db.comedie.find({pays : "France", date : {$gte : 2000, $lt : 2010}}, {titre : 1, _id : 0})
```

If not grouped, **only the last condition** on a given attribute **will be evaluated**.

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

AND Operators (3)

The explicit **$and** operator cannot be mixed with the previously described implicit methods.

```
db.comedie.find({$and : [{pays : ''France''}, {date : {$gte : 2000}}, {date : {$lt : 2010}}]}, {titre : 1, _id : 0})
```

OR Operator

There are 2 possible means of expressing a disjunction of conditions:

Using the **$in** operator (applicable to a set of values only)

```
db.comedie.find({date : {$in [2000, 2002, 2004]}},{titre : 1, _id : 0})
```

Is equivalent to

```
SELECT titre FROM comedie WHERE date in (2000, 2002, 2004)
```

Using the **$or** operator (applicable to logical expressions)

```
db.comedie.find({date : 2000, $or [budget < 100000, nbEntrees < 10000]},{titre : 1, _id : 0})
```

Is equivalent to

```
SELECT titre FROM comedie WHERE date = 2000 AND (budget < 100000 OR nbEntrees < 10000)
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Other Operators: `$exists` and `$type`

The **`$exists`** and **`$type`** evaluate attribute fields.

`$exists` tests whether an attribute field is present in a record

- Syntax: `{fieldName: {$exists: boolean }}`
- Example:

  `{nbEntrees : {$exists: true, $lt 10000}}` checks whether field `nbEntrees` exists and then tests if **`nbEntrees < 10000`**.

  `{nbEntrees : {$exists: false}}` returns `True` if the field `nbEntrees` is absent

`$type` tests whether an attribute field is of a given type

- Syntax: `{fieldName: {$type : BSON type number | String alias }}`
- Example:

  `{nbEntrees : {$type: 16}}` returns `True` if `nbEntrees` is of BSON type 16

  `{nbEntrees : {$type: "int"}}` returns `True` if `nbEntrees` is of the *`int`* type

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Other Operators: `$mod` and `$regex`

`$mod` checks the remainder of an integer division

- Syntax: `{fieldName: {$mod: [divisor, remainder]}}`
- Example : **`{nbEtudiants: {$mod: [3, 0]}}`** returns `True` if `nbEtudiants%3 == 0`

`$regex` evaluates regular expressions

- Syntax: `{fieldName: {$regex: regExp, $options: options}}`
- Example : **`{regionName: {$regex: /^lor/, $options: 'i'}}`**
  Returns `True` if `regionName` starts with either one of *lor*, *Lor*, *loR*, *LOR* …
  - **`/^lor/`**: starts with "lor"
  - **`$options: 'i'`**: case insensitive

Regular expressions have a large range of possible options and operators … refer to specialised documentation.

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Other Operators: `$text`

`$text` checks content of full text fields

- Syntax: `{fieldName: {$text: {$search: <string>,`

  `$language: <string>,`

  `$caseSensitive: <boolean>,`

  `$diacriticSensitive: <boolean> }}}`

- Example: **`{article: {$text: {"étudiant", "fr", false, false} }}`**

  Checks whether the term "*étudiant*" appears in `article`, the text of which is assumed to be French. The search is case insensitive and diacritic blind (*i.e.* terms *Etudiant, étudiant* and *etudiant* are considered equivalent)

  The `$language` parameter allows for adapting word stemming and filtering diacritics.

Other Operators: `$where`

`$where` allows for implementing a JavaScript selection operator in case the standard MongoDB built-in operators are insufficient.

Syntax: `{fieldName: {$where: JavaScript expression }}`

- `db.collectionName.find({active: true, $where: function() {return obj.credits - obj.debits < 0;} })`
- `db.collectionName.find({active: true, $where: "this.credits - this.debits < 0" })`

Array Operators 1

`$all` checks if an array typed field contains all specified ***values***

- Syntax: `{arrayFieldName: {$all: [val1,val2,val3…] }}`

  Returns *`True`* if *`arrayFieldName`* contains all provided listed values (*`val1`*, *`val2`* and *`val3`*).

- Example:

  If attribute field ***`cours`*** equals **`["info", "elec", "auto", "anglais"]`**

  then **`{cours: {$all: ["info", "anglais"] }}`** returns *`True`*

Array Operators 2

`$elemMatch` checks if an array typed field verifies all specified ***conditions***

- Syntax: `{arrayFieldName: {$elemMatch: {query1, query2…} }}`

  Returns `True` if array typed field `arrayFieldName` contains at least one element satisfying provided queries.

- Example: **Is there at least one grade (*note*) value within the [7, 10[range ?**

  If the array field `notes` equals `[18, 8, 17, 11]`

  then `{notes: {$elemMatch: {$gte: 7, $lt: 10}}}` returns `True`

`$size` checks if an array typed field is of a given size

- Syntax: `{arrayFieldName: {$size: theSize }}`

  Returns `True` if `arrayFieldName` a has `theSize` length.

What about testing if an array is of size > n ?

Handling Whole Collections

BESIDES “JUST” QUERYING

Applying Methods to Collections

General Syntax

`db.CollectionName.find(…).method(…)`

sometimes also

`db.CollectionName.method(…)`

Methods

- `sort(…)`
- `forEach(…)`
- …

- `count(…)`
- …

`sort()` method

Sorts records in a collection resulting from a `find()` command, according to multiple fields, in ascending or descending order.

Syntax: `db.`*`CollectionName`*`.find(…).`**`sort`**`({`*`fieldName1`*`:[-1,1],`
`fieldName2``:[-1,1] …})`

Example: `db.my`*`Collection`*`.find(…).`**`sort({"a.b.c.d":1, "a.b.x.y":-1})`**

Orders output of `find()` command according to `a.b.c.d` in **ascending order**, and uses field `a.b.x.y` in **descending order** as secondary sort key.

`count()` method

Returns the number of records in a collection.

Syntax:

- `db.`*`CollectionName`*`.find(…).`**`count()`**
- `db.`*`CollectionName`*`.`**`count()`**

`foreach()` method

Applies a specified function for each record in a collection.

Syntax: `db.`*`CollectionName`*`.find(…).forEach(function(doc) {…})`

The provided function is applied to all records of the collection. It can either have no side effects on the database, or can modify the database.

Examples:

- `db.myCollection.find().forEach(function(doc) {print(doc.a.b.c.d);})`
  Simply prints the specified field of each collection record.
- ```
  db.myCollection.find().forEach(function(doc) {
      db.myCollection.update({_id: doc._id}, {attribute: value })})
  ```
  Calls the `update()` function on each record, erasing any previous content
- ```
  db.laCollection.find().forEach(function(doc) {
      db.myCollection.update({_id: doc._id}, {$set: {attribute: value} })})
  ```
  Calls the `update()` function on each record, preserving any previous content

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Aggregating and pipelining

`aggregate()` framework

MongoDB allows to sequence and pipeline treatments using specific operators

These operators are similar to the built-in operators presented previously but are generally slower (since not taking advantage of internal optimisation like indexes)

Syntax: `db.collectionName.aggregate({op1}, {op2}, {op3}…)`

Sequentially pipelines the results of each operation into the next one without limit on the number of operations. ***Works on one single collection only.***

Available operators

- **`$match`**
- **`$project`**
- **`$group`**
- **`$unwind`**
- **`$sort`**
- **`$limit`**
- **`$lookup`**

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Example Data Base

We will be assuming the following record structure in all examples that follow.

The records represent publications by authors

```
{"_id": ObjectId("5911b9e9df7184e1c0778a8a"),
            …
 "paper": {
            …
            "authors": "Isabelle Kabla-Langlois,Florian Lezec",
            …
            "year": "2017"
 },
 "record_timestamp": "2017-04-14T16:04:46+02:00"
}
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

$match operator

$match is a selection operator on input records

Example:

Retrieve publications posterior to 2007

```
db.myCollection.aggregate(
    {"$match": {"paper.year": {$gt: 2007}}}
)
```

$project operator

$project is a projection operator on input records

It can also be used to rename or create new fields

Example:

Retain only authors and publication year

```
db.myCollection.aggregate(
    {"$match": {"paper.year": {$gt: 2007}}},
    {"$project": {"paper.authors": 1,
                  "paper.year": 1}}
)
```

$group operator

$group allows to group records by given id (to be defined) and to apply specific group functions and redefine new attribute fields

Example:

The author list becomes the new id and new fields expressing first common publication year and number of publications are computed.

```
db.myCollection.aggregate(
    {"$match": {"paper.year": {$gt: 2007}}},
    {"$project": {"paper.authors": 1, "paper.year": 1}},
    {"$group": {"_id": "$paper.authors",
                "start": {$min: "$paper.year"},
                "occurences": {$sum: 1}}}
)
```

$sort operator

$sort orders records according to specified fields, in ascending or descending order.

Example:

Sort by decreasing number of publications

```
db.myCollection.aggregate(
    {"$match": {"paper.year": {$gt: 2007}}},
    {"$project": {"paper.authors": 1, "paper.year": 1}},
    {"$group": {"_id": "$paper.authors",
                "start": {$min: "$paper.year"},
                "occurences": {$sum: 1}}},
    {"$sort": {"occurences": -1}}
)
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

$limit operator

$limit limits the number of returned results.

Example:

Return only 10 most productive authors

```
db.myCollection.aggregate(
    {"$match": {"paper.year": {$gt: 2007}}},
    {"$project": {"paper.authors": 1, "paper.year": 1}},
    {"$group": {"_id": "$paper.authors",
                "start": {$min: "$paper.year"},
                "occurences": {$sum: 1}}},
    {"$sort": {"occurences": -1}},
    {"$limit": 10}
).pretty()
```

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Results


```
{ "_id": "Joëlle Grille",
  "start": "2008",
  "occurences": 7 }
{ "_id": "Isabelle Kabla-Langlois, Louis-Alexandre Erb",
  "start": "2015",
  "occurences": 6 }
{ "_id": "Isabelle Kabla-Langlois ,Mathias Denjean",
  "start": "2016",
  "occurences": 5 }
{ "_id": "Annie Le Roux",
  "start": "2009",
  "occurences": 5 }
{ "_id": "Isabelle Kabla-Langlois, Claudette-Vincent Nisslé, Laurent Perrain",
  "start": "2015",
  "occurences": 5 }
```

$unwind operator

$unwind "flattens" a record containing a ***multiple-value array*** field ***into*** a set of records with a ***single value*** field each.

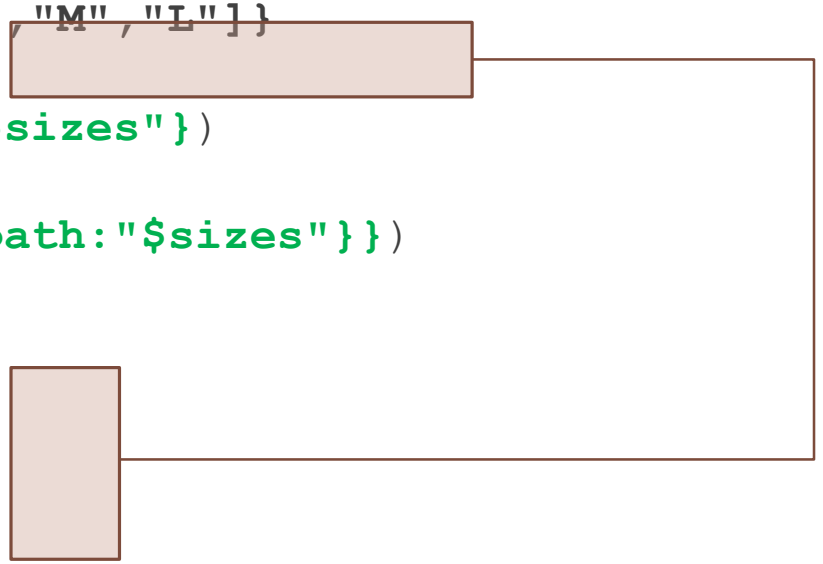
Records with an empty array field are discarded

Example:

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

db.theCollection.aggregate({$unwind: "$sizes"})
or
db.theCollection.aggregate({$unwind: {path:"$sizes"}})

{"_id": 1, "item": "ABC", "sizes": "S"}
{"_id": 1, "item": "ABC", "sizes": "M"}
{"_id": 1, "item": "ABC", "sizes": "L"}
```



UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

$unwind operator

$unwind "flattens" a record containing a ***multiple-value array*** field ***into*** a set of records with a ***single value*** field each.

Records with an empty array field can be forced

Example

```
{"_id": 1, "item": "EFG", "sizes": []}
```

```
db.theCollection.aggregate({$unwind: {path:"$sizes",
          preserveNullAndEmptyArrays: true}})
```

```
{"_id": 1, "item": "EFG"}
```

Temporary conclusions

`db.collectionName.find(…)` is similar in expressiveness and role to the SQL **`SELECT`** command

`db.collectionName.aggregate(…)` allows for pipelining sequential treatment of data and queries into more complex data handling

Limitations:

Operate on ***one single collection***

Relational algebra **JOIN** operations are **not possible**
(design patterns of NoSQL collections try to anticipate joins by storing the required data overhead in the collections themselves)

`aggregate()` is slower than `find()` and requires more resources

$lookup Operator and JOINs

$lookup is a specific operator for executing JOINs between collections of a same database.

It can only be executed within an **aggregate()** method.

It can handle complex JOIN conditions and sub-queries (refer to on-line MongoDB documentation for further details)

Syntax example:

```
db.collectionName1.aggregate([
    {"$lookup" : {
            "localField" : fieldName1,
            "from" : collectionName2,
            "foreignField" : fieldName2,
            "as" : "joined" }
    },
    {"$out" : outputCollectionName}
]);
```

- Will create a new field **"joined"** for each record of ***CollectionName1***.
- The field **"joined"** will contain all records of ***CollectionName2*** sharing the same value of ***fieldName1*** with ***fieldName2***.
- Resulting collection will be stored in ***outputCollectionName***.

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

$lookup Operator Example

```
db.co1.aggregate([
   {"$lookup": {
      "localField": "title",
      "from": "co2",
      "foreignField": "title_book",
      "as": "joined" }
   },
   {"$out": "resJoin"}
]);
```

Collection **resJoin**:

```
{
  {"_id": …,
   "title": "Cours de Big Data",
   "editeur": "CentraleSupelec",
   "annee": "2016-2017",
   "joined": [
      {"_id" : …,
       "title_book": "Cours de Big Data",
       "thematique": "informatique",
       "prerequis": ["programmation", "SQL"]
      },
      …
   ]
  }
  …
}
```

- Further transformation/filtering of the output is generally necessary
- The **$unwind** and **$project** operators will prove useful

Optimizations

USE OF INDEXES

Indexes

Indexes are a standard DataBase optimization technique

Speeds up many operations and significantly reduces memory overhead

In MongoDB **`aggregate()`** and **`mapreduce()`** ***do not take advantage*** of indexes (only **`find()`** and associated methods do)

Syntax:

```
db.collectionName.createIndex({"fieldName" : -1})
```

mapReduce

… IS NOT MAP-REDUCE

Basic Syntax

Syntax:

1. `res = db.runCommand({"mapreduce": collectionName, "map": mapFunction, "reduce": reduceFunction, "out": outputCollectionName, "finalize": finalizeFunction})`
2. `db.collectionName.mapReduce(mapFunction, reduceFunction, {"out": outputCollectionName, "finalize": finalizeFunction})`

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Example

```
map1 = function() {
  var key = this.book.title;
  var value = {"count" : 1} ;
  emit(key,value);
}
```

```
reduce1 = function(key,values) {
  var s = 0;
  for (var i in values)
    s += values[i].count;
  return( {"count" : s} );
}
```

```
finalize1 = function(key,values) {
  return( {"title" : key, "nb" : values.count} );
}
```

1. `res = db.runCommand({"mapreduce": co1, "map": map1, "reduce": reduce1, "out": "res1", "finalize": finalize1})`
2. `db.co1.mapReduce(map1, reduce1, {"out": "res1", "finalize": finalize1})`