Polytech Annecy-Chambéry – Université Savoie Mont-Blanc

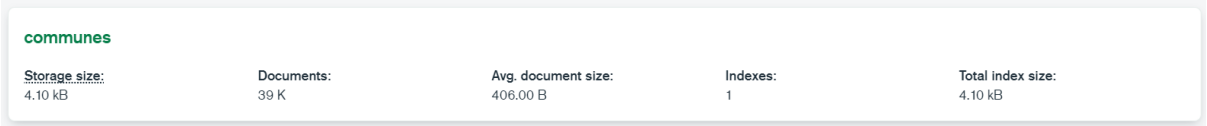
IDU 4 – INFO834 | PERROLLAZ Maverick & CAULLIREAU Dorian | TP3-4 (17/02/2022)

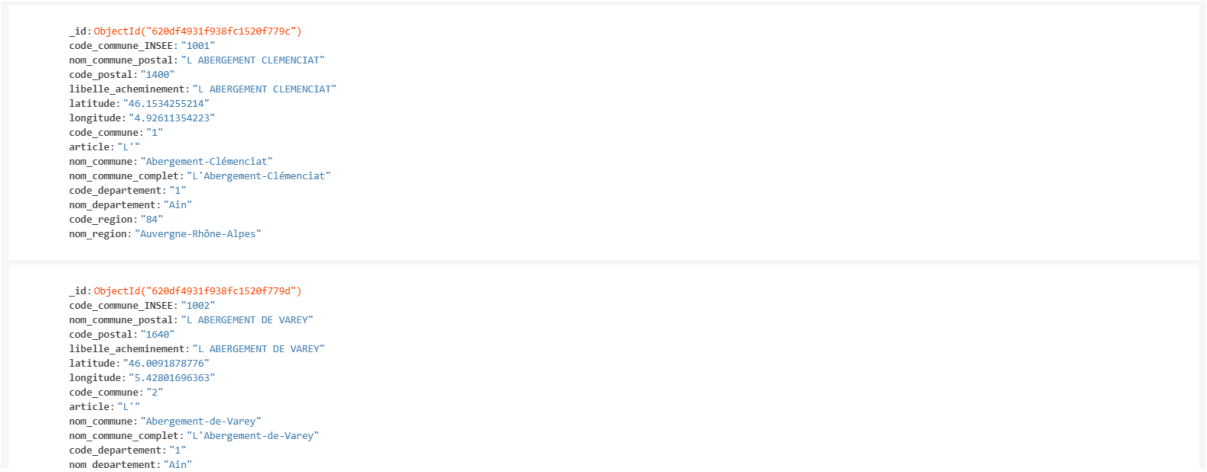
3. On insère de la donnée

Nous pouvons alors ajouter le fichier sous la forme suivante :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	code_commune_INSEE,nom_commune_postal,code_postal,libelle_acheminement,ligne_5,latitude,longitude,code_commune,article,nom_commune,nom_commune_complet,code_departement,nom_departement,code_region,nom_region																
2	1001,L ABERGEMENT CLEMENCIAT,1400,L ABERGEMENT CLEMENCIAT,,46.1534255214,4.92611354223,1,L',Abergement-ClÃ©menciat,L'Abergement-ClÃ©menciat,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
3	1002,L ABERGEMENT DE VAREY,1640,L ABERGEMENT DE VAREY,,46.0091878776,5.42801696363,2,L',Abergement-de-Varey,L'Abergement-de-Varey,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
4	1004,AMBERIEU EN BUGEY,1500,AMBERIEU EN BUGEY,,45.9608475114,5.3729257777,4,,AmbÃ©rieu-en-Bugey,AmbÃ©rieu-en-Bugey,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
5	1005,AMBERIEUX EN DOMBES,1330,AMBERIEUX EN DOMBES,,45.9961799872,4.91227250796,5,,AmbÃ©rieux-en-Dombes,AmbÃ©rieux-en-Dombes,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
6	1006,AMBLEON,1300,AMBLEON,,45.7494989044,5.59432017366,6,,AmblÃ©on,AmblÃ©on,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
7	1007,AMBRONAY,1500,AMBRONAY,,46.0055913782,5.35760660735,7,,Ambronay,Ambronay,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
8	1008,AMBUTRIX,1500,AMBUTRIX,,45.9367134524,5.3328092349,8,,Ambutrix,Ambutrix,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
9	1009,ANDERT ET CONDON,1300,ANDERT ET CONDON,,45.7873565333,5.65788307924,9,,Andert-et-Condon,Andert-et-Condon,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
10	1010,ANGLEFORT,1350,ANGLEFORT,,45.9093715116,5.79516005674,10,,Anglefort,Anglefort,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
11	1011,APREMONT,1100,APREMONT,,46.2054981558,5.65781475272,11,,Apremont,Apremont,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
12	1012,ARANC,1110,ARANC,,46.0015344029,5.51130637511,12,,Aranc,Aranc,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
13	1013,ARANDAS,1230,ARANDAS,,45.8908155275,5.49870439091,13,,Arandas,Arandas,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
14	1014,ARBENT,1100,ARBENT,,46.2834936005,5.690616512221,14,,Arbent,Arbent,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
15	1015,ARBOYS EN BUGEY,1300,ARBOYS EN BUGEY,ARBIGNIEU,45.7237621545,5.65263086429,15,,Arboys en Bugey,Arboys en Bugey,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
16	1015,ARBOYS EN BUGEY,1300,ARBOYS EN BUGEY,ST BOIS,45.7237621545,5.65263086429,15,,Arboys en Bugey,Arboys en Bugey,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
17	1016,ARBIGNY,1190,ARBIGNY,,46.4776762988,4.9599416863,16,,Arbigny,Arbigny,1,Ain,84,Auvergne-RhÃ´ne-Alpes																
18	1017,ARGIS,1230,ARGIS,,45.9337182132,5.48251100314,17,,Argis,Argis,1,Ain,84,Auvergne-RhÃ´ne-Alpes																

Nous pouvons donc ajouter ce fichier CSV dans notre MongoDB avec le mongoimport :





Nous avons donc bien nos 39201 documents dans notre base

4. Les commandes de base

show databases : permet de voir l'ensemble des database du serveur local

```
> show databases
France   0.004GB
admin    0.000GB
config   0.000GB
local    0.000GB
tdmongo  0.008GB
user     0.000GB
```

use France : permet de changer de répertoire

```
> use France
switched to db France
```

show collections : permet de récupérer l'ensemble des collections de la base actuel

```
> show collections
communes
```

db.communes.count() : permet de compter les éléments de la collection actuelle

```
> db.communes.count()
39201
```

db.communes.find() : permet de récupérer l'ensemble des communes

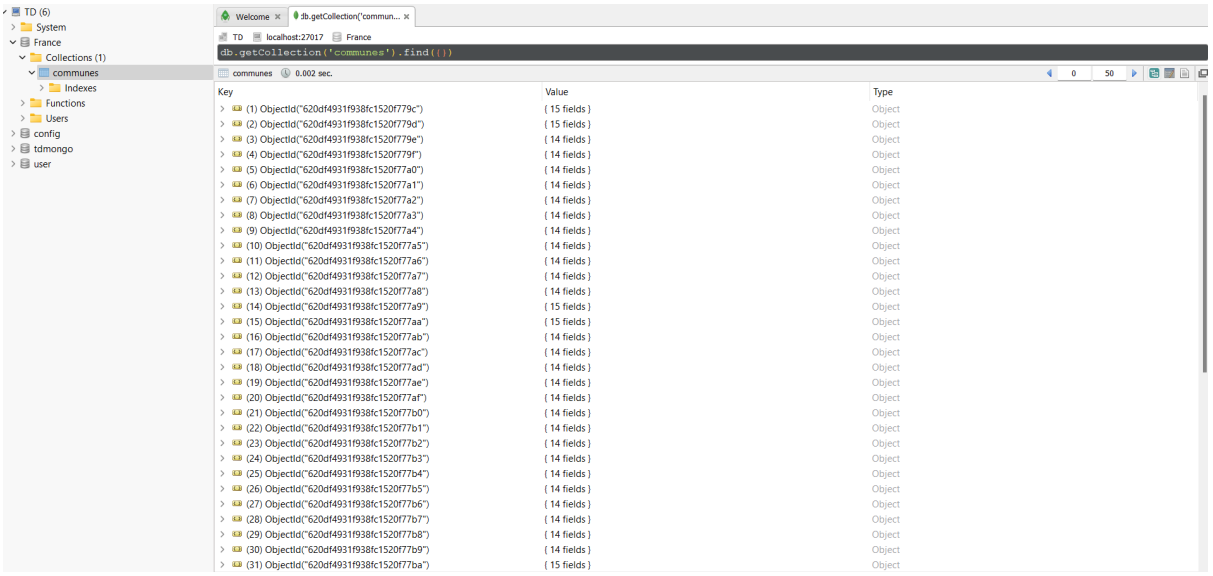

```
{ "_id" : ObjectId("620df4931f938fc1520f77ab"), "code_commune_INSEE" : "1016", "nom_commune_postal" : "ARBIGNY", "code_p
ostal" : "1190", "libelle_acheminement" : "ARBIGNY", "latitude" : "46.4776762988", "longitude" : "4.9599416863", "code_c
ommune" : "16", "nom_commune" : "Arbigny", "nom_commune_complet" : "Arbigny", "code_departement" : "1", "nom_departement
" : "Ain", "code_region" : "84", "nom_region" : "Auvergne-Rhône-Alpes" }
{ "_id" : ObjectId("620df4931f938fc1520f77ac"), "code_commune_INSEE" : "1017", "nom_commune_postal" : "ARGIS", "code_pos
tal" : "1230", "libelle_acheminement" : "ARGIS", "latitude" : "45.9337182132", "longitude" : "5.48251100314", "code_comm
une" : "17", "nom_commune" : "Argis", "nom_commune_complet" : "Argis", "code_departement" : "1", "nom_departement" : "Ai
n", "code_region" : "84", "nom_region" : "Auvergne-Rhône-Alpes" }
{ "_id" : ObjectId("620df4931f938fc1520f77ad"), "code_commune_INSEE" : "1019", "nom_commune_postal" : "ARMIX", "code_pos
tal" : "1510", "libelle_acheminement" : "ARMIX", "latitude" : "45.8541838459", "longitude" : "5.58357838695", "code_comm
une" : "19", "nom_commune" : "Armix", "nom_commune_complet" : "Armix", "code_departement" : "1", "nom_departement" : "Ai
n", "code_region" : "84", "nom_region" : "Auvergne-Rhône-Alpes" }
{ "_id" : ObjectId("620df4931f938fc1520f77ae"), "code_commune_INSEE" : "1021", "nom_commune_postal" : "ARS SUR FORMANS",
 "code_postal" : "1480", "libelle_acheminement" : "ARS SUR FORMANS", "latitude" : "45.9934611", "longitude" : "4.8219964
9494", "code_commune" : "21", "nom_commune" : "Ars-sur-Formans", "nom_commune_complet" : "Ars-sur-Formans", "code_depart
ement" : "1", "nom_departement" : "Ain", "code_region" : "84", "nom_region" : "Auvergne-Rhône-Alpes" }
{ "_id" : ObjectId("620df4931f938fc1520f77af"), "code_commune_INSEE" : "1022", "nom_commune_postal" : "ARTEMARE", "code_
postal" : "1510", "libelle_acheminement" : "ARTEMARE", "latitude" : "45.8695891748", "longitude" : "5.69065283943", "cod
e_commune" : "22", "nom_commune" : "Artemare", "nom_commune_complet" : "Artemare", "code_departement" : "1", "nom_depart
ement" : "Ain", "code_region" : "84", "nom_region" : "Auvergne-Rhône-Alpes" }
```


db.communes.find().pretty() : même principe, mais avec une mise en forme des données

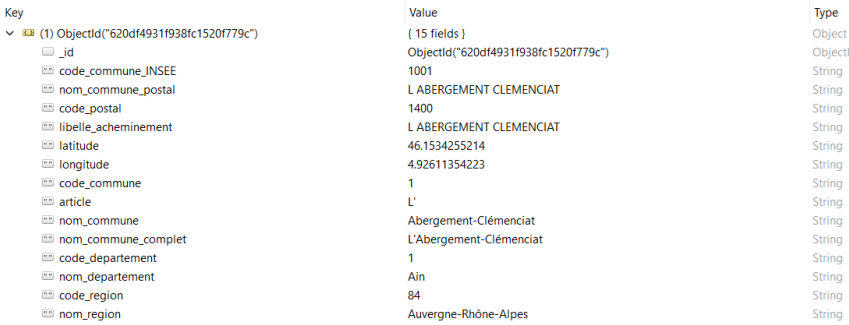

```
{
        "_id" : ObjectId("620df4931f938fc1520f77ae"),
        "code_commune_INSEE" : "1021",
        "nom_commune_postal" : "ARS SUR FORMANS",
        "code_postal" : "1480",
        "libelle_acheminement" : "ARS SUR FORMANS",
        "latitude" : "45.9934611",
        "longitude" : "4.82199649494",
        "code_commune" : "21",
        "nom_commune" : "Ars-sur-Formans",
        "nom_commune_complet" : "Ars-sur-Formans",
        "code_departement" : "1",
        "nom_departement" : "Ain",
        "code_region" : "84",
        "nom_region" : "Auvergne-Rhône-Alpes"
}
```

5. L'outil de visualisation Robo 3T

Pour faciliter la visualisation, nous pouvons utiliser des couches supérieures, pour avoir une interface plus agréable.



Nous pouvons retrouver l'ensemble des informations de notre table, dans la même idée que via le terminal ou le GUI de Compass.



5. Utilisation de curseurs

Pour faciliter la récupération d'un grand nombre de données, nous pouvons utiliser des curseurs, il permet la pagination de données et l'affichage de l'ensemble des données.

```
var myCursor = db.communes.find( {} );
while (myCursor.hasNext()) {
  print(tojson(myCursor.next()));
}
```

Nous pouvons voir directement l'ensemble des données.


```
{
        "_id" : ObjectId("620df4951f938fc1521010ba"),
        "code_commune_INSEE" : "98833",
        "nom_commune_postal" : "KOUAOUA",
        "code_postal" : "98818",
        "libelle_acheminement" : "KOUAOUA",
        "code_commune" : "833",
        "nom_commune" : "Kouaoua",
        "nom_commune_complet" : "Kouaoua",
        "code_departement" : "98"
}
{
        "_id" : ObjectId("620df4951f938fc1521010bb"),
        "code_commune_INSEE" : "98901",
        "nom_commune_postal" : "ILE DE CLIPPERTON",
        "code_postal" : "98799",
        "libelle_acheminement" : "ILE DE CLIPPERTON",
        "code_commune" : "901",
        "nom_commune" : "Ile de clipperton",
        "nom_commune_complet" : "Ile de clipperton",
        "code_departement" : "98"
}
```


7. Un peu de benchmark

Dans des projets avec un grand nombre de données, l'optimisation des temps d'exécution est très importante. Lorsque nous avons un très grand nombre de données, chaque ligne sur la donnée prend beaucoup de temps.

Nous avons essayé d'optimiser au maximum le délai d'exécution pour la recherche des informations d'une ville avec son nom.

```
(async () => {
      const { MongoClient } = require('mongodb');
      const moment = require('moment')
      const client = new MongoClient('mongodb://127.0.0.1:27017');
      await client.connect();

      const db = client.db('France');
      const communes = db.collection('communes');

      const nom_communes = (await communes.find().toArray()).map(c => c.nom_commune);

      const startTime = moment();

      for (let i = 0; i < 10000; i++) {
          await communes.findOne({ "nom_commune": nom_communes[i]})
              if ((i+1)%1000 === 0) {
                  let tmpTime = moment();
```

```
                    console.log((i+1) + " elements: " +
                            tmpTime.diff(startTime) + "ms")
                }
        }

        let tmpTime = moment();
        console.log("Total:" + tmpTime.diff(startTime) + "ms")

        process.exit()
})();
```

Pour comprendre notre programme, voici les différentes étapes d'exécution :

1. Connexion à la base de donnée MongoDB

```
const { MongoClient } = require('mongodb');
const moment = require('moment')
const client = new MongoClient('mongodb://127.0.0.1:27017');
await client.connect();
```

2. On switch notre connexion sur la bonne base et sur la bonne collection

```
const db = client.db('France');
const communes = db.collection('communes');
```

3. On stock dans une map l'ensemble des nom des communes

```
const nom_communes = (await communes.find().toArray()).map(c => c.nom_commune);
```

4. On lance notre timer pour le calcul du temps d'exécution

```
const startTime = moment();
```

5. On fait la recherche sur les 10000 premières communes, et on afficher le temps d'exécution toutes les 1000 communes

```
for (let i = 0; i < 10000; i++) {
          await communes.findOne({ "nom_commune": nom_communes[i]})
              if ((i+1)%1000 === 0) {
                  let tmpTime = moment();
                   console.log((i+1) + " elements: " +
                           tmpTime.diff(startTime) + "ms")
              }
      }
```

6. On affiche le temps total et on coupe le programme

```
let tmpTime = moment();
console.log("Total:" + tmpTime.diff(startTime) + "ms")

process.exit()
```

Nous pouvons donc essayer notre programme, et observer le temps d'exécution :

```
1000 elements: 637ms
2000 elements: 1609ms
3000 elements: 2800ms
4000 elements: 4276ms
5000 elements: 6111ms
6000 elements: 8404ms
7000 elements: 10863ms
8000 elements: 13889ms
9000 elements: 17129ms
10000 elements: 20620ms
Total:20620ms
```

Nous avons donc une exécution pour les 10000 premières villes de 20 620 ms, soit grossièrement **20 secondes**.

8. Optimisation avec indexage

Pour gagner du temps dans la recherche des éléments, l'indexage est très important, et permet de faire gagner un temps considérable.

Le plus cohérent est d'appliquer un index sur le champ que nous utilisons pendant la recherche, dans notre cas la recherche la plus efficace porterait sur le nom de la ville.

```
db.communes.createIndex( { nom_commune : 1 } )
```

Maintenant que nous avons un index sur le nom de la commune, nous pouvons essayer de refaire notre test de temps d'exécution.

```
maverick@DESKTOP-Maverick:~/Cours/S8/INFO834/TP3-TP4/Q7$ node main.js
1000 elements: 494ms
2000 elements: 934ms
3000 elements: 1284ms
4000 elements: 1618ms
5000 elements: 1928ms
6000 elements: 2231ms
7000 elements: 2549ms
8000 elements: 2850ms
9000 elements: 3161ms
10000 elements: 3464ms
Total:3465ms
```

Nous pouvons voir que le temps total est beaucoup plus faible que précédemment, nous avons un temps complet de 3.465 secondes.

Calcul du ratio

Temps d'exécution sans aucun index : *20,62 ms*
Temps d'exécution avec l'index sur le nom : *3,465 ms*

$$Ratio = \frac{Temps\ d'exécution\ sans\ aucun\ index}{Temps\ d'exécution\ avec\ l'index\ sur\ le\ nom} = \frac{20{,}62\ s}{3{,}465\ s} \simeq 5.95$$

Nous avons donc une exécution environ 5,95 fois plus rapide avec l'index sur le nom

L'indexage est possible sur le champ recherché, mais également sur l'ensemble des champs du modèle.

Nous pouvons alors essayer d'ajouter un index sur un autre champ : par exemple, ici sur le

champ **“code_commune_INSEE”** (nous avons préalablement supprimé l’index sur le nom, pour observer la différence sur les temps d’exécution.

```
db.communes.createIndex( { code_commune_INSEE: 1 })
```

```
1000 elements: 645ms
2000 elements: 1585ms
3000 elements: 2770ms
4000 elements: 4278ms
5000 elements: 6094ms
6000 elements: 8242ms
7000 elements: 10664ms
8000 elements: 13477ms
9000 elements: 16621ms
10000 elements: 20059ms
Total:20059ms
```

Nous pouvons voir aucun changement par rapport à l’exécution sans aucun index, nous avions un temps équivalent.

Nous pouvons voir que l’index est efficace uniquement sur le champ recherché.

9. Le nombre de visiteurs

Nous allons maintenant utiliser la variable *$inc* pour déterminer le nombre de visiteurs sur une page (globalement comparable à une variable d’incrémentation).

Nous utilisons une nouvelle base :

```
> use Analytics
switched to db Analytics
```

Nous pouvons maintenant créer un document directement via MongoShell :

```
> visit = { url : "https://www.google.com",
... ... nb : 1}
{ "url" : "https://www.google.com", "nb" : 1 }
```

Nous pouvons maintenant l’ajouter dans la collection :

```
> db.freq.insert(visit)
WriteResult({ "nInserted" : 1 })
```

Nous avons donc bien notre nouvelle base “Analytics” :

```
> show databases
Analytics  0.000GB
France     0.004GB
admin      0.000GB
config     0.000GB
local      0.000GB
tdmongo    0.008GB
test       0.000GB
user       0.000GB
```

Nous avons donc bien la nouvelle donnée que nous pouvons modifier :

```
> db.freq.find().pretty()
{
        "_id" : ObjectId("621c799c7c1e5e5424abdad3"),
        "url" : "https://www.google.com",
        "nb" : 1
}
```

```
> db.freq.update({_id : ObjectId("621c799c7c1e5e5424abdad3")}, {$inc:
... {nb : 1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Lorsque nous regardons une nouvelle fois nos données, nous pouvons voir qu'elle a bien été mise à jour.

```
> db.freq.find().pretty()
{
        "_id" : ObjectId("621c799c7c1e5e5424abdad3"),
        "url" : "https://www.google.com",
        "nb" : 2
}
```

10. Utilisation d'Array dans les documents

Nous allons maintenant voir l'utilisation des array dans les différents documents. Nous pouvons en effet ajouter des chaines de caractères et également des nombres, mais aussi des tableaux de données.

Pour l'exercice nous ajoutons une nouvelle base de données :

```
> use Mailing
switched to db Mailing
```

Nous créons et ajoutons le tableau de données dans notre base :

```
> list = {
... ... name : "Mailing-List 1",
... ... emails : []
... ... }
{ "name" : "Mailing-List 1", "emails" : [ ] }
```

```
> db.lists.insert(list)
WriteResult({ "nInserted" : 1 })
```

Nous pouvons voir et valider l'insertion de la donnée :

```
> db.lists.find().pretty()
{
        "_id" : ObjectId("621c7b444b1c292af6c1ae89"),
        "name" : "Mailing-List 1",
        "emails" : [ ]
}
```

Nous pouvons également modifier ces données :

```
> db.lists.update({_id : ObjectId("621c7b444b1c292af6c1ae89")},
... ... {$push : {emails: "wrongaddress@404"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Toujours en validant la bonne modification de la donnée :

```
> db.lists.find().pretty()
{
        "_id" : ObjectId("621c7b444b1c292af6c1ae89"),
        "name" : "Mailing-List 1",
        "emails" : [
                "Marc-Philippe.Huget@univ-smb.fr",
                "wrongaddress@404"
        ]
}
```

Comme dans tout modèle de donnée, nous pouvons également supprimer un enregistrement :

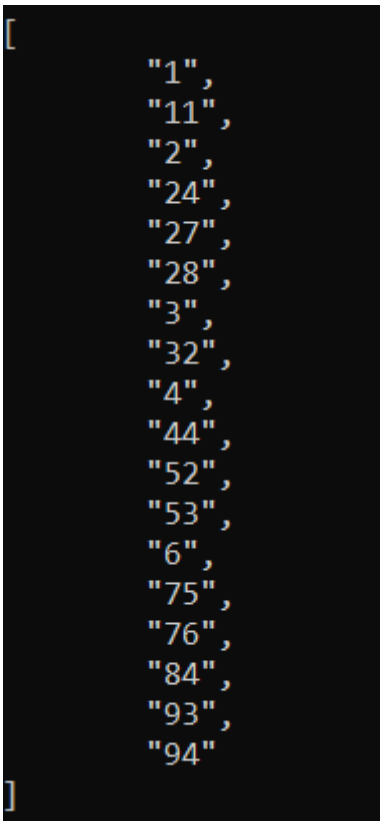
```
> db.lists.update({_id : ObjectId("621c7b444b1c292af6c1ae89")},
... {$pull : {emails : "wrongaddress@404"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.lists.find().pretty()
{
        "_id" : ObjectId("621c7b444b1c292af6c1ae89"),
        "name" : "Mailing-List 1",
        "emails" : [
                "Marc-Philippe.Huget@univ-smb.fr"
        ]
}
```

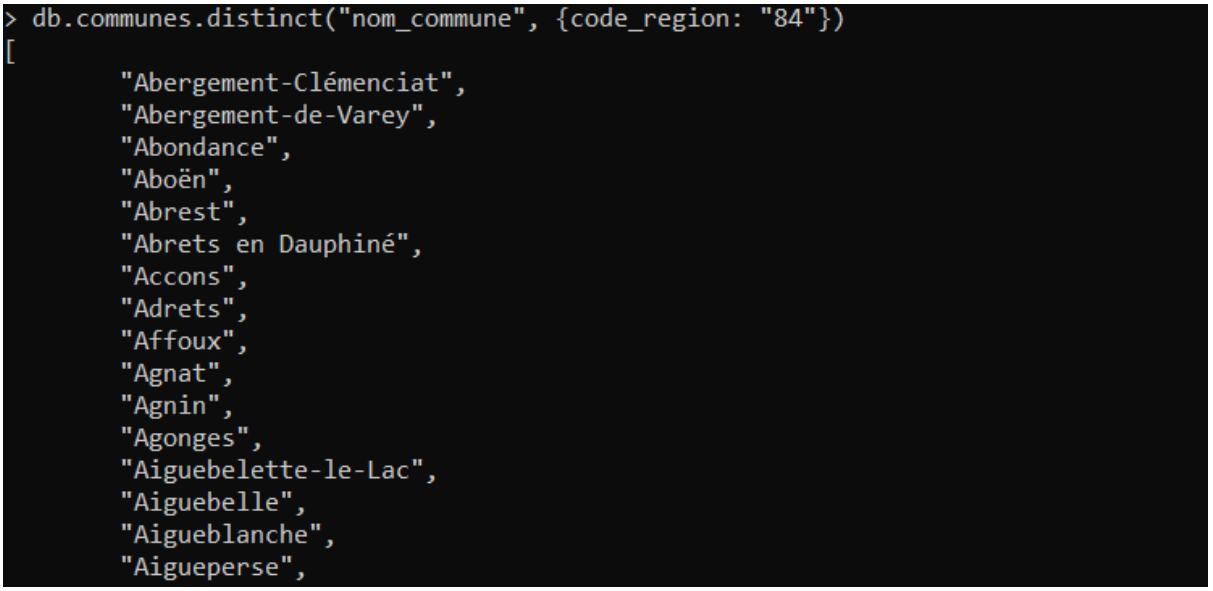
Nous avons donc avec cette utilisation l'ensemble des fonctionnalités classiques d'un modèle **CRUD (Create, Read, Update, Delete).**

11. Les valeurs distinctes

Nous pouvons voir l'ensemble des lignes correspondants aux régions :



Et une recherche et possible via ce champ (technique différente de l'indexage) :



Nous avons donc une partie des données en fonction de la recherche (ici l'ensemble des communes qui sont incluses dans le code région **84**.

12. Et si on se liait ?

Nous allons maintenant essayer de faire un lien ente les différents modèles et essayer de faire des relations entre chacun d'eux.

Nous ajoutons une nouvelle collection :

```
> use Mailing
switched to db Mailing
```

```
> user1 = { name : "Doe",
... ... firstname: "John",
... ... email: "John.Doe@email.org"}
{ "name" : "Doe", "firstname" : "John", "email" : "John.Doe@email.org" }
```

```
> db.users.insert(user1)
WriteResult({ "nInserted" : 1 })
```

```
> user2 = { name : "Claude",
... ... firstname : "Claude",
... ... email : "cloclo@email.org"}
{ "name" : "Claude", "firstname" : "Claude", "email" : "cloclo@email.org" }
```

```
> db.users.insert(user2)
WriteResult({ "nInserted" : 1 })
```

Nous pouvons observer et valider l'ensemble des insertions :

```
> db.users.find().pretty()
{
        "_id" : ObjectId("621c81144b1c292af6c1ae8a"),
        "name" : "Doe",
        "firstname" : "John",
        "email" : "John.Doe@email.org"
}
{
        "_id" : ObjectId("621c813e4b1c292af6c1ae8b"),
        "name" : "Claude",
        "firstname" : "Claude",
        "email" : "cloclo@email.org"
}
```

Nous allons maintenant voir l'idée sur la mailing-liste :

```
> db.lists.drop()
true
```

```
> db.lists.find().pretty()
> list = { name : "Mailing-List 1", users :
... [ObjectId("5e81f5cd1e95dfc6e9cd3613"),
... ObjectId("5e81f6041e95dfc6e9cd3614")]}
{
        "name" : "Mailing-List 1",
        "users" : [
                ObjectId("5e81f5cd1e95dfc6e9cd3613"),
                ObjectId("5e81f6041e95dfc6e9cd3614")
        ]
}
```

```
> db.lists.insert(list)
WriteResult({ "nInserted" : 1 })
```

```
> db.lists.find().pretty()
{
        "_id" : ObjectId("621c81a74b1c292af6c1ae8c"),
        "name" : "Mailing-List 1",
        "users" : [
                ObjectId("5e81f5cd1e95dfc6e9cd3613"),
                ObjectId("5e81f6041e95dfc6e9cd3614")
        ]
}
```

Avec NodeJS et l'utilisation de **.populate** nous allons voir l'ensemble du mailing et de l'ensemble des utilisateurs.

Le code ci dessous nous permet de récupérer les informations importantes :

```
maverick@DESKTOP-Maverick:~/Cours/S8/INFO834/TP3-TP4/programs$ node Q13.js
[
  {
    _id: new ObjectId("620e0992c1ae6bb1c2afdee6"),
    name: 'Mailing-List 1',
    users: [ [Object], [Object] ]
  }
]
```

```
(async () => {
    // Requiring module
    const mongoose = require('mongoose');
    const { Schema } = mongoose;

    // Connecting to database
    await mongoose.connect('mongodb://127.0.0.1:27017/Mailing');

    // Creating Schemas
    const UserSchema = new mongoose.Schema({
        id: Schema.Types.ObjectId,
        name: String,
        firstname: String,
        email: String,
    });

    const ListSchema = new mongoose.Schema({
        id: Schema.Types.ObjectId,
        name: String,
        users: [{
            type: mongoose.Schema.Types.ObjectId,
            ref: "User"
        }],
    })

    // Creating models from userSchema and postSchema
    const List = mongoose.model("Lists", ListSchema);
    const User = mongoose.model("User", UserSchema);

    module.exports = {
        List, User
    }

    List.find({"name" : "Mailing-List 1"})
        .populate("users")
        .exec(function (err, res){
            if (err) throw err;
            console.log(res)
        });

    // process.exit()
})();
```

13. Montrez patte blanche

Nous avons pu créer de nouveaux utilisateurs pour protéger nos bases de données, et permettre de contrôler de filtrer l'accès.

```
db.createUser(
  {
    user: "userTest",
    pwd:  "password",
    roles: [ { role: "read", db: "Analytics" } ]
  }
)
```

14. MapReduce

Nous allons maintenant voir ce qu'il est possible de faire avec MapReduce.
Dans un premier temps l'objectif est de rechercher l'ensemble des communes qui appartiennent à la région **"Auvergne Rhône Alpes"** *(c'est-à-dire avec le code 84).*

Nous avons donc une fonction map et reduce qui permet de faire la recherche ainsi que la récupération de l'ensemble des informations en fonction du paramètre :

```
> var mapFunction = function() { if (this.code_region == "84") emit(this.nom_commune, 1); }
> var reduceFunction = function(nom, index) {
... ... ... return Array.sum(index);
... ... ... }
```

Le concept de map/reduce ce fait à l'aide de la fonction mapReduce sur les différentes communes.

```
> db.communes.mapReduce(mapFunction, reduceFunction, {out : "map_reduce_example"});
{ "result" : "map_reduce_example", "ok" : 1 }
```

Nous pouvons donc avec la fonction find() récupérer l'ensemble des informations du Map/Reduce :

```
> db.map_reduce_example.find()
{ "_id" : "Châtel-en-Trièves", "value" : 1 }
{ "_id" : "Bully", "value" : 2 }
{ "_id" : "Saint-Gal-sur-Sioule", "value" : 1 }
{ "_id" : "Héry-sur-Alby", "value" : 1 }
{ "_id" : "Cressanges", "value" : 1 }
{ "_id" : "Saint-André-le-Coq", "value" : 1 }
{ "_id" : "Saint-Germain-Lembron", "value" : 1 }
{ "_id" : "Villeurbanne", "value" : 1 }
{ "_id" : "Tencin", "value" : 1 }
{ "_id" : "Maillat", "value" : 1 }
{ "_id" : "Savasse", "value" : 2 }
{ "_id" : "Rillieux-la-Pape", "value" : 2 }
{ "_id" : "Larringes", "value" : 1 }
```

Nous allons désormais refaire le même exercice en ajoutant une contrainte sur le nom de la commune, on doit commencer par “A” et être en région Rhône-Alpes (avec le code 84).

```
> var mapFunction = function() { if (this.code_region == "84" && this.nom_commune.charAt(0) == "A") emit(this.nom_commune, 1); }
> var reduceFunction = function(nom, index) { return Array.sum(index); }
> db.communes.mapReduce(mapFunction, reduceFunction, {out : "map_reduce_example"});
{ "result" : "map_reduce_example", "ok" : 1 }
```

Nous pouvons ensuite voir une partie des résultats du **Map/Reduce** :

```
> db.map_reduce_example.find()
{ "_id" : "Arronnes", "value" : 1 }
{ "_id" : "Ally", "value" : 3 }
{ "_id" : "Auberives-sur-Varèze", "value" : 1 }
{ "_id" : "Argis", "value" : 1 }
{ "_id" : "Abrets en Dauphiné", "value" : 3 }
{ "_id" : "Audes", "value" : 1 }
{ "_id" : "Arpajon-sur-Cère", "value" : 1 }
{ "_id" : "Avrieux", "value" : 1 }
{ "_id" : "Accons", "value" : 1 }
{ "_id" : "Annemasse", "value" : 1 }
{ "_id" : "Albepierre-Bredons", "value" : 1 }
{ "_id" : "Amions", "value" : 1 }
{ "_id" : "Ainay-le-Château", "value" : 1 }
{ "_id" : "Aizac", "value" : 1 }
{ "_id" : "Asnières-sur-Saône", "value" : 1 }
{ "_id" : "Allègre", "value" : 1 }
{ "_id" : "Alboussière", "value" : 1 }
{ "_id" : "Allemond", "value" : 1 }
{ "_id" : "Arenthon", "value" : 1 }
{ "_id" : "Andrézieux-Bouthéon", "value" : 1 }
Type "it" for more
```

15. Sauvegarde et restauration des données

Nous pouvons faire une sauvegarde des données :

```
maverick@DESKTOP-Maverick:~/Cours/S8/INFO834/TP3-TP4$ mongodump --out=./backup/
2022-02-28T11:54:13.975+0100    writing admin.system.users to backup/admin/system.users.bson
2022-02-28T11:54:13.982+0100    done dumping admin.system.users (1 document)
2022-02-28T11:54:13.982+0100    writing admin.system.version to backup/admin/system.version.bson
2022-02-28T11:54:13.986+0100    done dumping admin.system.version (2 documents)
2022-02-28T11:54:13.986+0100    writing Analytics.city to backup/Analytics/city.bson
2022-02-28T11:54:14.045+0100    writing Analytics.zips to backup/Analytics/zips.bson
2022-02-28T11:54:14.062+0100    writing France.communes to backup/France/communes.bson
2022-02-28T11:54:14.090+0100    writing France.map reduce example to backup/France/map+reduce+example.bson
2022-02-28T11:54:14.531+0100    done dumping France.map reduce example (33440 documents)
2022-02-28T11:54:14.532+0100    writing France.map_reduce_example to backup/France/map_reduce_example.bson
2022-02-28T11:54:14.546+0100    done dumping France.map_reduce_example (224 documents)
```

Récupération de la sauvegarde :

```
maverick@DESKTOP-Maverick:~/Cours/S8/INFO834/TP3-TP4$ mongorestore backup/
2022-02-28T11:55:21.378+0100    preparing collections to restore from
2022-02-28T11:55:21.454+0100    reading metadata for France.communes from backup/France/communes.metadata.json
2022-02-28T11:55:21.455+0100    reading metadata for France.map reduce example from backup/France/map+reduce+example.metadata.json
2022-02-28T11:55:21.456+0100    reading metadata for France.map_reduce_example from backup/France/map_reduce_example.metadata.json
2022-02-28T11:55:21.457+0100    reading metadata for Mailing.lists from backup/Mailing/lists.metadata.json
2022-02-28T11:55:21.458+0100    reading metadata for polyshop.categories from backup/polyshop/categories.metadata.json
2022-02-28T11:55:21.458+0100    reading metadata for polyshop.products from backup/polyshop/products.metadata.json
2022-02-28T11:55:21.459+0100    reading metadata for polyshop.shops from backup/polyshop/shops.metadata.json
2022-02-28T11:55:21.460+0100    reading metadata for Analytics.freq from backup/Analytics/freq.metadata.json
2022-02-28T11:55:21.461+0100    reading metadata for todo.todos from backup/todo/todos.metadata.json
2022-02-28T11:55:21.462+0100    reading metadata for Analytics.zips from backup/Analytics/zips.metadata.json
2022-02-28T11:55:21.463+0100    reading metadata for Mailing.users from backup/Mailing/users.metadata.json
2022-02-28T11:55:21.464+0100    reading metadata for Analytics.city from backup/Analytics/city.metadata.json
2022-02-28T11:55:21.467+0100    restoring to existing collection France.communes without dropping
2022-02-28T11:55:21.467+0100    restoring to existing collection Analytics.city without dropping
2022-02-28T11:55:21.467+0100    restoring to existing collection Analytics.zips without dropping
2022-02-28T11:55:21.467+0100    restoring to existing collection France.map reduce example without dropping
2022-02-28T11:55:21.468+0100    restoring France.communes from backup/France/communes.bson
2022-02-28T11:55:21.468+0100    restoring Analytics.zips from backup/Analytics/zips.bson
2022-02-28T11:55:21.468+0100    restoring France.map reduce example from backup/France/map+reduce+example.bson
2022-02-28T11:55:21.468+0100    restoring Analytics.city from backup/Analytics/city.bson
^C2022-02-28T11:55:21.511+0100  signal 'interrupt' received; attempting to shut down
```

16. Agrégation

MongoDB Aggregation permet de faire un grand nombre de traitements sur les données, nous allons essayer l'ensemble des agrégations existantes sur le tuto suivant : https://fr.blog.businessdecision.com/tutoriel-mongodb-agregation/

Fonction d'agrégation **count()** : Permet de compter le nombre d'éléments :

```
> db.zips.count()
29353
```

Fonction d'agrégation **findOne()** : Permet de trouver un élément :

```
> db.zips.findOne()
{
        "_id" : "01012",
        "city" : "CHESTERFIELD",
        "loc" : [
                -72.833309,
                42.38167
        ],
        "pop" : 177,
        "state" : "MA"
}
```

Fonction d'agrégation pour faire **la somme** des éléments :

```
> db.zips.aggregate({$group:{_id:"$state",population:{$sum:"$pop"}}})
{ "_id" : "MD", "population" : 4781379 }
{ "_id" : "CT", "population" : 3287116 }
{ "_id" : "NY", "population" : 17990402 }
{ "_id" : "RI", "population" : 1003218 }
{ "_id" : "MO", "population" : 5110648 }
{ "_id" : "MT", "population" : 798948 }
{ "_id" : "ME", "population" : 1226648 }
{ "_id" : "NE", "population" : 1578139 }
{ "_id" : "IN", "population" : 5544136 }
{ "_id" : "AR", "population" : 2350725 }
{ "_id" : "SC", "population" : 3486703 }
{ "_id" : "TN", "population" : 4876457 }
{ "_id" : "FL", "population" : 12686644 }
{ "_id" : "PA", "population" : 11881643 }
{ "_id" : "NH", "population" : 1109252 }
{ "_id" : "MN", "population" : 4372982 }
{ "_id" : "VT", "population" : 562758 }
{ "_id" : "GA", "population" : 6478216 }
{ "_id" : "IL", "population" : 11427576 }
{ "_id" : "IA", "population" : 2776420 }
Type "it" for more
```

Fonction d'agrégation sur **le maximum** des éléments :

```
> db.zips.aggregate({$group:{_id:"$state",population:{$max:"$pop"}}})
{ "_id" : "MD", "population" : 76002 }
{ "_id" : "RI", "population" : 53733 }
{ "_id" : "CT", "population" : 60670 }
{ "_id" : "NY", "population" : 111396 }
{ "_id" : "MO", "population" : 54994 }
{ "_id" : "MT", "population" : 40121 }
{ "_id" : "ME", "population" : 40434 }
{ "_id" : "NE", "population" : 35325 }
{ "_id" : "IN", "population" : 56543 }
{ "_id" : "AR", "population" : 53532 }
{ "_id" : "SC", "population" : 66990 }
{ "_id" : "TN", "population" : 60508 }
{ "_id" : "FL", "population" : 73194 }
{ "_id" : "PA", "population" : 80454 }
{ "_id" : "NH", "population" : 41438 }
{ "_id" : "MN", "population" : 51421 }
{ "_id" : "VT", "population" : 39127 }
{ "_id" : "GA", "population" : 58646 }
{ "_id" : "IL", "population" : 112047 }
{ "_id" : "IA", "population" : 52105 }
Type "it" for more
```

Fonction d'agrégation sur **la moyenne** des éléments :

```
> db.zips.aggregate({$group:{_id:"$state",population:{$avg:"$pop"}}})
{ "_id" : "NY", "population" : 11279.248902821317 }
{ "_id" : "RI", "population" : 14539.391304347826 }
{ "_id" : "CT", "population" : 12498.539923954373 }
{ "_id" : "MT", "population" : 2544.420382165605 }
{ "_id" : "MO", "population" : 5141.496981891348 }
{ "_id" : "ME", "population" : 2991.8243902439026 }
{ "_id" : "NE", "population" : 2749.3710801393727 }
{ "_id" : "IN", "population" : 8201.384615384615 }
{ "_id" : "SC", "population" : 9962.00857142857 }
{ "_id" : "TN", "population" : 8378.792096219931 }
{ "_id" : "AR", "population" : 4066.998269896194 }
{ "_id" : "UT", "population" : 8404.146341463415 }
{ "_id" : "NH", "population" : 5088.311926605505 }
{ "_id" : "PA", "population" : 8149.275034293552 }
{ "_id" : "FL", "population" : 15779.407960199005 }
{ "_id" : "IA", "population" : 3011.301518438178 }
{ "_id" : "MN", "population" : 4958.02947845805 }
{ "_id" : "GA", "population" : 10201.914960629922 }
{ "_id" : "VT", "population" : 2315.8765432098767 }
{ "_id" : "IL", "population" : 9238.137429264349 }
Type "it" for more
```

Fonction d'agrégation sur **le tri et la somme** des éléments :

```
> db.zips.aggregate({$group:{_id:"$city",population:{$sum:"$pop"}}},{$sort:{population:-1}})
{ "_id" : "CHICAGO", "population" : 2452177 }
{ "_id" : "BROOKLYN", "population" : 2341387 }
{ "_id" : "HOUSTON", "population" : 2123053 }
{ "_id" : "LOS ANGELES", "population" : 2102295 }
{ "_id" : "PHILADELPHIA", "population" : 1639862 }
{ "_id" : "NEW YORK", "population" : 1476790 }
{ "_id" : "BRONX", "population" : 1209548 }
{ "_id" : "SAN DIEGO", "population" : 1054316 }
{ "_id" : "DALLAS", "population" : 999042 }
{ "_id" : "DETROIT", "population" : 967468 }
{ "_id" : "PHOENIX", "population" : 902249 }
{ "_id" : "MIAMI", "population" : 848436 }
{ "_id" : "COLUMBUS", "population" : 825448 }
{ "_id" : "SAN JOSE", "population" : 817497 }
{ "_id" : "SAN ANTONIO", "population" : 813188 }
{ "_id" : "WASHINGTON", "population" : 780954 }
{ "_id" : "BALTIMORE", "population" : 738846 }
{ "_id" : "JACKSONVILLE", "population" : 735505 }
{ "_id" : "SAN FRANCISCO", "population" : 723993 }
{ "_id" : "CLEVELAND", "population" : 687451 }
```

Fonction d'agrégation sur **la somme, le tri et faire un filtre** sur une région particulière :

```
> db.zips.aggregate({$match:{state:"CA"}},{$group:{_id:"$city",population:{$sum:"$pop"}}},{$sort:{population:-1}})
{ "_id" : "LOS ANGELES", "population" : 2102295 }
{ "_id" : "SAN DIEGO", "population" : 1049298 }
{ "_id" : "SAN JOSE", "population" : 816653 }
{ "_id" : "SAN FRANCISCO", "population" : 723993 }
{ "_id" : "SACRAMENTO", "population" : 628279 }
{ "_id" : "FRESNO", "population" : 347905 }
{ "_id" : "OAKLAND", "population" : 314487 }
{ "_id" : "LONG BEACH", "population" : 299651 }
{ "_id" : "ANAHEIM", "population" : 272327 }
{ "_id" : "BAKERSFIELD", "population" : 271347 }
{ "_id" : "STOCKTON", "population" : 267258 }
{ "_id" : "RIVERSIDE", "population" : 253478 }
{ "_id" : "SANTA ANA", "population" : 234472 }
{ "_id" : "MODESTO", "population" : 216459 }
{ "_id" : "HUNTINGTON BEACH", "population" : 183542 }
{ "_id" : "SAN BERNARDINO", "population" : 177552 }
{ "_id" : "FREMONT", "population" : 173374 }
{ "_id" : "GLENDALE", "population" : 163666 }
{ "_id" : "SANTA ROSA", "population" : 158398 }
{ "_id" : "TORRANCE", "population" : 158183 }
Type "it" for more
```

Fonction qui permet de ***compte***r le nombre de codes postaux par états :

```
> db.zips.aggregate({$group:{_id:"$state",nb_zip:{$sum:1}}},{$sort:{nb_zip:-1}})
{ "_id" : "TX", "nb_zip" : 1671 }
{ "_id" : "NY", "nb_zip" : 1595 }
{ "_id" : "CA", "nb_zip" : 1516 }
{ "_id" : "PA", "nb_zip" : 1458 }
{ "_id" : "IL", "nb_zip" : 1237 }
{ "_id" : "OH", "nb_zip" : 1007 }
{ "_id" : "MO", "nb_zip" : 994 }
{ "_id" : "IA", "nb_zip" : 922 }
{ "_id" : "MN", "nb_zip" : 882 }
{ "_id" : "MI", "nb_zip" : 876 }
{ "_id" : "VA", "nb_zip" : 816 }
{ "_id" : "KY", "nb_zip" : 809 }
{ "_id" : "FL", "nb_zip" : 804 }
{ "_id" : "WI", "nb_zip" : 716 }
{ "_id" : "KS", "nb_zip" : 715 }
{ "_id" : "NC", "nb_zip" : 705 }
{ "_id" : "IN", "nb_zip" : 676 }
```

Fonction qui permet de générer un groupe avec les associations État/Villes :

```
> db.zips.aggregate({$group:{_id:{state:"$state",city:"$city"}}})
{ "_id" : { "state" : "MA", "city" : "CONWAY" } }
{ "_id" : { "state" : "OK", "city" : "BURLINGTON" } }
{ "_id" : { "state" : "WA", "city" : "LA CONNER" } }
{ "_id" : { "state" : "WA", "city" : "SEABECK" } }
{ "_id" : { "state" : "PA", "city" : "SEWARD" } }
{ "_id" : { "state" : "NC", "city" : "BOONE" } }
{ "_id" : { "state" : "NY", "city" : "SOUTH EDMESTON" } }
{ "_id" : { "state" : "PA", "city" : "BENTON" } }
{ "_id" : { "state" : "WV", "city" : "RIVESVILLE" } }
{ "_id" : { "state" : "SC", "city" : "SEABROOK" } }
{ "_id" : { "state" : "AL", "city" : "FOSTERS" } }
{ "_id" : { "state" : "VA", "city" : "AMELIA COURT HOU" } }
{ "_id" : { "state" : "MS", "city" : "NEW SITE" } }
```

Fonction qui fait la somme du nombre de villes par état :

```
> db.zips.aggregate({$group:{_id:{state:"$state",city:"$city"}}},{$group:{_id:"$_id.state",nb_villes:{$sum:1}}})
{ "_id" : "MD", "nb_villes" : 379 }
{ "_id" : "CT", "nb_villes" : 224 }
{ "_id" : "NY", "nb_villes" : 1370 }
{ "_id" : "MO", "nb_villes" : 901 }
{ "_id" : "RI", "nb_villes" : 52 }
{ "_id" : "MT", "nb_villes" : 308 }
{ "_id" : "ME", "nb_villes" : 408 }
{ "_id" : "NE", "nb_villes" : 520 }
{ "_id" : "IN", "nb_villes" : 598 }
{ "_id" : "AR", "nb_villes" : 563 }
{ "_id" : "SC", "nb_villes" : 313 }
{ "_id" : "TN", "nb_villes" : 505 }
{ "_id" : "FL", "nb_villes" : 463 }
{ "_id" : "PA", "nb_villes" : 1369 }
{ "_id" : "MN", "nb_villes" : 814 }
{ "_id" : "IL", "nb_villes" : 1148 }
{ "_id" : "VT", "nb_villes" : 243 }
{ "_id" : "GA", "nb_villes" : 561 }
{ "_id" : "WA", "nb_villes" : 397 }
{ "_id" : "IA", "nb_villes" : 889 }
Type "it" for more
```

Puis un dernier exemple pour la route, un peu plus complexe provenant du tutoriel :

“Amusons-nous alors un peu. Imaginons maintenant que nous souhaitions afficher les états ayant le plus de villes à plus de 100 000 habitants. Nous allons alors intercaler dans notre traitement un filtre sur le nombre d’habitants et rajouter à la fin un tri par ordre décroissant. Bien sûr, il ne faut pas oublier dans la première étape de calculer la population de chaque ville puisque nous allons en avoir besoin dans notre filtre à l’étape 2.”

```
> db.zips.aggregate({$group:{_id:{state:"$state",city:"$city"},population:
{$sum:"$pop"}}},{$match:{population:{$gt:100000}}},{$group:{_id:"$_id.stat
e",nb_villes:{$sum:1}}},{$sort:{nb_villes:-1}})
{ "_id" : "CA", "nb_villes" : 44 }
{ "_id" : "TX", "nb_villes" : 22 }
{ "_id" : "FL", "nb_villes" : 20 }
{ "_id" : "NY", "nb_villes" : 13 }
{ "_id" : "VA", "nb_villes" : 11 }
{ "_id" : "MI", "nb_villes" : 11 }
{ "_id" : "OH", "nb_villes" : 9 }
{ "_id" : "CO", "nb_villes" : 8 }
{ "_id" : "WA", "nb_villes" : 7 }
{ "_id" : "MO", "nb_villes" : 6 }
{ "_id" : "GA", "nb_villes" : 6 }
{ "_id" : "NC", "nb_villes" : 6 }
{ "_id" : "LA", "nb_villes" : 6 }
{ "_id" : "AZ", "nb_villes" : 6 }
{ "_id" : "NJ", "nb_villes" : 6 }
{ "_id" : "IN", "nb_villes" : 5 }
{ "_id" : "IL", "nb_villes" : 5 }
{ "_id" : "TN", "nb_villes" : 4 }
{ "_id" : "AL", "nb_villes" : 4 }
{ "_id" : "WI", "nb_villes" : 4 }
Type "it" for more
```