Université de Caen Normandie
Institut Universitaire de Technologie Grand Ouest Normandie
Département Informatique

Bachelor Universitaire de Technologie

INFORMATIQUE

Rapport TP2

Apprendre les bases du Big Data et de MongoDB

Cyprien DUROY - Alexandre Le Roy

Résumé :

Ce rapport décrit la réalisation de deux séances de tp de 2H. Elles ont eu pour but de nous apprendre les bases de l'utilisation de mongodb. On y trouve une présentation de mongo db. comment mettre en place un serveur mongo db. un test des commandes de base. Un test des requêtes de projections, de mise à jour et d'agrégation. Et enfin comment utiliser mongo db avec du javascript grâce à node.

Université de Caen Normandie
Institut Universitaire de Technologie Grand Ouest Normandie
Département Informatique

Bachelor Universitaire de Technologie

INFORMATIQUE

Rapport TP2

Apprendre les bases du Big Data et de MongoDB

Rapport d'activité
sous forme de portefeuille de compétences

Cyprien DUROY - Alexandre Le Roy

Remerciements :

Merci beaucoup à M Porcq pour son aide précieuse lors des deux différents TP.

Sommaire :

1. Introduction

MongoDB est une base de données NoSQL orientée documents, conçue pour stocker et gérer de grandes quantités de données de manière flexible et performante. Contrairement aux bases relationnelles, elle ne repose pas sur des tables et des lignes fixes, mais sur des collections qui contiennent des documents au format JSON ou BSON. Chaque document est un ensemble de paires clé-valeur, et il peut avoir un schéma différent des autres documents de la même collection. Cette flexibilité permet de stocker des données hétérogènes sans avoir besoin de modifier une structure rigide à chaque changement. MongoDB utilise un identifiant unique _id pour chaque document, garantissant l'unicité au sein de la collection. Les documents peuvent contenir des sous-documents et des tableaux, ce qui facilite la représentation de structures complexes et de relations imbriquées sans avoir recours aux jointures classiques. Pour les relations entre documents, on peut utiliser des références ou des imbrications selon le besoin. MongoDB propose un moteur d'agrégation performant qui permet de réaliser des filtres, regroupements et calculs directement sur les documents. Il est conçu pour la scalabilité horizontale, ce qui facilite le partage des données entre plusieurs serveurs (sharding). Grâce à son modèle flexible, MongoDB est très utilisé dans les applications modernes nécessitant rapidité et adaptabilité, comme les sites web, les applications mobiles ou les systèmes de big data. Son interface en ligne de commande, mongosh, permet de manipuler les bases de manière intuitive et d'exécuter facilement des requêtes complexes. En résumé, MongoDB combine performance, flexibilité et simplicité pour gérer des données structurées ou semi-structurées, offrant une alternative moderne aux bases relationnelles traditionnelles.

2. Test Du Serveur

Pour que mongo db fonctionne il faut d'abord préparer le serveur qui va gérer et accueillir nos collections, voici les différentes étapes :

tout d'abord il faut créer une partition virtuelle du disque de travail pour que cela soit plus propre :

2.1 Créer une partition virtuelle du disque pour faire un travail propre :

```
PS C:\Users\Utilisateur> cd C:\Users\Utilisateur\Nextcloud\S5\R5.10
PS C:\Users\Utilisateur\Nextcloud\S5\R5.10> subst S:/D
Paramètre non valide - S:/D
PS C:\Users\Utilisateur\Nextcloud\S5\R5.10> subst S: .
PS C:\Users\Utilisateur\Nextcloud\S5\R5.10> S:
PS S:\>
```

2.2 Fabriquer le dossier data/db dans son dossier de travail (donc dans S:\) :

```
PS S:\> mkdir data/db


    Répertoire : S:\data


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         05/09/2025   12:03                db
PS S:\>
```

2.3 Lancer mongod mongod.exe --dbpath :

```
mongod.exe --dbpath S:\data\db --bind_ip 127.0.0.1
```

2.4 lancer mongosh :

```
mongosh
```

3. Test de différentes commandes de base

Maintenant que nous avons mis en place le serveur, nous allons pouvoir tester quelques commandes de base de mongo shell :

◦ db :

résultat :

```
test> db
test
test>
```

db est une variable de mongosh qui représente la base de données actuellement utilisé.

◦ show dbs ou show databases

résultat :

```
test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   40.00 KiB
test>
```

show databases (ou show dbs qui est son raccourci) permet de voir les différentes bases de données présentent sur le serveur ainsi que leur taille.

◦ use maBDDTP2

résultat :

```
test> use maBDDTP2
switched to db maBDDTP2
maBDDTP2>
```

use permet de sélectionner une base de données présente sur le serveur pour ensuite pouvoir exécuter les différentes commandes du shell sur celle-ci.

Remarque : si la base écrite à la suite du use n'existe pas encore, elle sera créée.

◦ db

résultat :

```
maBDDTP2> db
maBDDTP2
maBDDTP2>
```

Nous pouvons voir que la commande use a bien créé la base de données maBDDTP2 tout en la mettant en tant que base de données de travail actuel

◦ show dbs

résultat :

```
maBDDTP2> show dbs
admin   40.00 KiB
config  60.00 KiB
local   40.00 KiB
maBDDTP2>
```

Nous affichons encore les bases de données, cependant nous ne voyons pas encore apparaître maBDDTP2 car il n'y a pas encore de collections à l'intérieur de celle-ci.

◦ show collections

résultat :

```
maBDDTP2> show collections
maBDDTP2>
```

show collections permet de visualiser les collections contenus dans la base de données actuelles. Cependant comme nous le disions auparavant aucune collection est contenue dans la database maBDDTP2.

◦ db.createCollection("maCollection1")

résultat :

```
maBDDTP2> db.createCollection("maCollection1")
{ ok: 1 }
maBDDTP2>
```

db.createCollection permet de créer une collection dont on définit le nom en paramètre de la commande, ici maCollection1 dans la database courante. Cette commande affiche ensuite si l'opération s'est bien passé en affichant soit 1 si c'est bon soit si c'est 0 si cela n'a pas été possible

◦ show dbs

résultat :

```
maBDDTP2> show dbs
admin           40.00 KiB
config 60.00 KiB
local   40.00 KiB
maBDDTP2   8.00 KiB
maBDDTP2>
```

On peut maintenant bien observer que maBDDTP2 existe car elle contient maintenant une collection.

◦ db.createCollection("maCollection2")

résultat :

```
maBDDTP2> db.createCollection("maCollection2")
{ ok: 1 }
maBDDTP2>
```

Cette commande nous permet de créer une deuxième collection dans maBDDTP2 du nom de maCollection2. On peut observer que l'opération s'est bien déroulée.

◦ db.createCollection("maCollection3")

résultat :

```
maBDDTP2> db.createCollection("maCollection3")
{ ok: 1 }
maBDDTP2>
```

Cette commande nous permet de créer une troisiéme collection dans maBDDTP2 du nom de maCollection3. On peut observer que l'opération s'est bien déroulée.

◦ show collections

```
maBDDTP2> show collections
maCollection1
maCollection2
maCollection3
maBDDTP2>
```

On peut maintenant observer grâce à show collections que les 3 collections ont bien été créé dans la bdd maBDDTP2

◦ db.maCollection3.insertOne({ "_id" : "1", "nom" : "nom1" })

résultat :

```
maBDDTP2> db.maCollection3.insertOne( { "_id" : "1", "nom" : "nom1" } )
{ acknowledged: true, insertedId: '1' }
maBDDTP2>
```

Cette commande permet d'enregistrer un enregistrement dans la collection maCollection3. son _id qui est un attribut obligatoire pour tous les enregistrements mongo db a été assigné à . De plus un attribut nom a été créé est initialisé à nom1. La commande retourne si l'enregistrement a bien été pris en compte grâce à acknowledged (true si réussi false si non) ainsi que l'id de l'enregistrement.

◦ db.maCollection3.insertOne({ "_id" : "35", ville : "Moult" , pays : "France", pop : 2200 , loc : [05,05] })

résultat :

```
maBDDTP2> db.maCollection3.insertOne( { "_id" : "35", ville : "Moult" , pays :
"France", pop : 2200 , loc : [ 05,05] } )
{ acknowledged: true, insertedId: '35' }
maBDDTP2>
```

Cette commande a permis de créer un nouvel enregistrement dans maCollection3. On voit que l'id a été défini ainsi que 4 attributs qui sont ville, pays, pop et loc. On peut observer que l'insertion c'est bien passé et que son id est bien 35.

◦ db.maCollection3.insertOne ({ "nom" : "nom2" })

résultat :

```
maBDDTP2> db.maCollection3.insertOne ( { "nom" : "nom2" } )
{
  acknowledged: true,
  insertedId: ObjectId("68bab91edf888d1a3cf58605")
}
maBDDTP2>
```

Cette commande a permis d'ajouter un nouvel enregistrement dans maCollection3 avec un attribut qui est nom. On observe que la commande s'est bien effectuée et on peut observer l'id de l'enregistrement. Lorsqu'on ne définit _id lors de l'enregistrement, il est généré automatiquement.

◦ db.maCollection3.find()

résultat :

```
maBDDTP2> db.maCollection3.find()
```

```
  { _id: '1', nom: 'nom1' },
  {
          _id: '35',
          ville: 'Moult',
          pays: 'France',
          pop: 2200,
          loc: [ 5, 5 ]
  },
  { _id: ObjectId("68bab91edf888d1a3cf58605"), nom: 'nom2' }
]
maBDDTP2>
```

La commande find sans paramètre permet d'afficher tous les enregistrements d'une collection ici maCollection3.

◦ db.maCollection3.find({},{_id:1})

résultat :

```
maBDDTP2> db.maCollection3.find({}, { _id: 1 } )
[
  { _id: '1' },
  { _id: '35' },
  { _id: ObjectId("68bab91edf888d1a3cf58605") }
]
maBDDTP2>
```

Cette commande a permis tous les id des différents enregistrements de maCollection3. On peut en déduire que le deuxième paramètre de la fonction permet de définir les attributs à afficher dun enregistrement avec find.

◦ db.maCollection3.find({"nom":"nom1"},{_id:0, nom:1})

```
maBDDTP2> db.maCollection3.find({"nom":"nom1"},{_id:0, nom:1})
[ { nom: 'nom1' } ]
maBDDTP2>
```

Cette commande a permis d'afficher les enregistrements qui ont un attribut nom dont la valeur est égale à "nom1". On peut en déduire que le premier paramètre de la fonction find permet de mettre des conditions pour définir les enregistrements à afficher. De plus, lorsqu'on met la valeur 0 pour un attribut donné de l'enregistrement il ne s'affiche pas dans l'affichage du résultat de la commande.

◦ db.maCollection3.drop()

résultat

```
maBDDTP2> db.maCollection3.drop()
true
maBDDTP2>
```

La commande drop permet d’effacer une collection ainsi que toutes ses données à l'intérieur, ici pour l’exemple maCollection3.

◦ config.set("displayBatchSize", 300)

résultat:

```
maBDDTP2> config.set("displayBatchSize", 300)
Setting "displayBatchSize" has been changed
maBDDTP2>
```

Cette commande permet de définir le nombre maximum d'enregistrements à afficher lorsqu'on utilise un find ici 300 enregistrements.

◦ db.dropDatabase()

résultat:

```
maBDDTP2> db.dropDatabase()
{ ok: 1, dropped: 'maBDDTP2' }
maBDDTP2>
```

La commande dropDatabase permet d’effacer l'entièreté de la base de données courantes ici maBDDTP2. Lorsque la commande est exécutée on peut voir si l’opération s’est bien passé et quel database nous avons supprimé.

◦ show databases

résultat:

```
maBDDTP2> show databases
admin 40.00 KiB
config  108.00 KiB
local   40.00 KiB
maBDDTP2>
```

Grâce à show databases nous pouvons observer la suppression de maBDDTP2.

4. Création d'une collection

Dans ce chapitre nous allons voir comment initialiser les données d'une collection et avec quelles différentes manières.

4.1 (1.4.1) Utiliser une base de données nommée maBDDTP2

résultat :

```
test> use maBDDTP2
switched to db maBDDTP2
maBDDTP2>
```

Nous créons une bdd maBDDTP2 grâce à la commande use.

4.2 (1.4.2) Créer une collection nommée maCollec1. Vérifier que la création a fonctionné.

résultat :

```
maBDDTP2> db.createCollection("maCollec1")
{ ok: 1 }
maBDDTP2> show collections
maCollec1
maBDDTP2>
```

Nous créons la collection maCollec1 grâce à la commande createCollection. Et ensuite nous vérifions si elle a bien été créée grâce à la commande show collections.

4.3 (1.4.3) Étudier les fichiers fournis (p14_X.json) et indiquer l'erreur du p14_2_faux.json.

Le premier dossier est un dossier json classique qui permet de décrire différentes données, avec différent format. On peut y voir différents enregistrements avec différents attributs chacun. Chaque attribut est formaté de manières différentes et peut contenir des formats de données différents que d'autres enregistrements ayant un attribut avec le même nom.
Le 14.3 est basé sur le même principe seulement il peut être inséré quand utilisant la fonction load au lieu de la fonction mongoimport.
le problème du 14.2 faux est que l'enregistrement d'un des âges est fait en octal car il ya un 0 devant le int.

4.4 (1.4.4) Y insérer les documents suivants. 3 solutions :
• On peut insérer ligne par ligne avec db..insert

```
maBDDTP2> db.maCollec1.insertOne({ "nom" : "Dubois" , "prenom" : "Maurice" , "age": 25 })
```

```
{
  acknowledged: true,
  insertedId: ObjectId("68bea3849f09c22b885f9613")
}
maBDDTP2> db.maCollec1.find()
[
  {
        _id: ObjectId("68bea3849f09c22b885f9613"),
        nom: 'Dubois',
        prenom: 'Maurice',
        age: 25
  }
]
maBDDTP2>
```

Pour utiliser cette solution, il suffit d'utiliser la commande insertOne en inscrivant en paramètre les couples attributs valeurs pour l'enregistrement souhaiter
Cette solution utilisant insertOne peut être pratique pour enregistrer un petit jeu de données avec peu d'enregistrement.

• On peut utiliser l'utilitaire mongoimport (ce n'est pas une commande du shell mongo). Le fichier associé contient toutes les données json nécessaires.

```
mongoimport --db maBDDTP2 --collection maCollec1 --file
Z:\S5\R510\TP2\code\code\p14_2.json
```

l'utilitaire mongoimport s'utilise dans l'invit de commande qui gère l'interface du serveur mongodb. Il a 3 paramètres : la bdd cible se trouvant sur le serveur, la collection cible présente dans cette bdd et le fichier json contenant les différents enregistrements.

```
maBDDTP2> db.maCollec1.find()
[
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 },
  { _id: 3, nom: 'Dulong', prenom: 'Sylvie', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' },
  { _id: '5', nom: 'Delalune', prenom: 'Claire' },
  { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 },
  { _id: 11, type: 'casserolle', couleur: 'rouge', 'matière': 'fonte' },
  { _id: 12, type: 'casserolle', couleur: 'verte', 'matière': 'fer' },
  { _id: 2, nom: 'Dupont', prenom: 'Bertrand', age: 28 },
  {
        _id: 6,
        nom: 'Super',
        telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
  },
  {
        _id: 7,
        nom: 'Super',
        prenom: 'veronique',
```

```
        telephone: { portable: '06 80 23 48 12', fixe: '02 32 41 22 16' }
 },
 { _id: 8, nom: 'orange', prenom: 'Enpierre', age: 36 },
 { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] },
 {
        _id: ObjectId("68bea60b359258b2a2df77b8"),
        nom: 'Dubois',
        prenom: 'Maurice',
        age: 25
 },
 { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
 { _id: 22, fruit: [ 'kiwi', 'banane', 'pomme' ] },
 { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
 { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
 { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
 { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
 { _id: 5, nom: 'Troj', telephone: { portable: '06 85 45 48 10' } }
]
```

et nous pouvons voir grâce au find que cela marche très bien.

• On peut utiliser la commande load. Le fichier associé contient les requêtes « db..insert ».

```
maBDDTP2> load('Z:/S5/R510/TP2/code/code/p14_3.json')
true
maBDDTP2>
```

Cela se base sur le même principe que le mongo import cela se fait sur un terminal client et permet comme l'utilitaire précédent de transférer des enregistrements en json dans une collection mongodb. Cependant le format du Json est légèrement différent.

4.5 (1.4.5) Analyser la collection. Quelle est la différence avec les données d'une BDDR ?

Les collections de MongoDB diffèrent des tables d'une base de données relationnelle par leur flexibilité et leur structure. Dans MongoDB, chaque document peut avoir un schéma différent et des types de données variables pour un même attribut, tandis que dans une base relationnelle, les tables imposent des colonnes avec des types fixes et respectent strictement les règles de normalisation pour éviter les redondances et assurer la cohérence des données. De plus, les relations entre données sont explicites dans les bases relationnelles via des clés étrangères et des jointures, alors que MongoDB privilégie l'imbrication de documents ou les références, rendant les relations moins formelles mais souvent plus rapides à gérer pour certaines opérations. Ainsi, MongoDB offre plus de souplesse au détriment d'une stricte normalisation, tandis que le relationnel garantit la rigueur et la cohérence des données.

5. Test des projections en Mongodb

L'objectif est d'apprendre à effectuer des projections, des restrictions et des mises à jour de base. La commande find() en MongoDB permet de rechercher et d'afficher des documents dans une collection. Elle prend généralement deux paramètres. Le premier correspond au filtre, c'est-à-dire les conditions que doivent respecter les documents pour être sélectionnés, un peu comme la clause WHERE en SQL. Par exemple, { age: { $gt: 20 } } permet de récupérer uniquement les documents où l'attribut age est supérieur à 20. Le second paramètre correspond à la projection, qui définit quels champs seront affichés dans le résultat. Une valeur 1 signifie que le champ est affiché, tandis que 0 le masque. Par exemple, { nom: 1, age: 1, _id: 0 } affichera uniquement le nom et l'âge, sans l'identifiant _id. Ainsi, la méthode find() est très flexible car elle permet à la fois de sélectionner précisément les documents et de contrôler les informations affichées dans la réponse.

5.1 (1.5.1) Afficher les documents avec et sans la méthode pretty

requête :

```
maBDDTP2> db.maCollec1.find()
```

résultat :

```
[
  {
        _id: ObjectId("68beaa02abf0abb375ab782a"),
        nom: 'Dubois',
        prenom: 'Maurice',
        age: 25
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 },
  { _id: 3, nom: 'Dulong', prenom: 'Sylvie', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' },
  { _id: '5', nom: 'Delalune', prenom: 'Claire' },
  { _id: 2, nom: 'Dupont', prenom: 'Bertrand', age: 28 },
  { _id: 11, type: 'casserolle', couleur: 'rouge', 'matière': 'fonte' },
  { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 },
  { _id: 12, type: 'casserolle', couleur: 'verte', 'matière': 'fer' },
  { _id: 8, nom: 'orange', prenom: 'Enpierre', age: 36 },
  { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] },
  { _id: 5, nom: 'Troj', telephone: { portable: '06 85 45 48 10' } },
  {
        _id: 6,
        nom: 'Super',
        telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
  },
  {
```

```
        _id: 7,
        nom: 'Super',
        prenom: 'veronique',
        telephone: { portable: '06 80 23 48 12', fixe: '02 32 41 22 16' }
 },
 { _id: 22, fruit: [ 'kiwi', 'banane', 'pomme' ] },
 { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
 { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
 { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
 { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
 { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] }
]
```

explication :

Cette commande affiche tous les documents de la collection maCollec1.

remarque :

L'option pretty permettait dans les anciennes versions de mongodb d'indenter le résultat d'un find pour qu'il soit plus lisible car sinon il était affiché seulement en ligne. Cependant avec les versions récentes, le résultat est déjà indenté de manière lisible donc l'option est devenue obsolète.

5.2 (1.5.2) Afficher pour tous les documents la valeur (attribut) nom

requête :

```
maBDDTP2> db.maCollec1.find({}, {nom : 1 , _id : 0})
```

résultat :

```
[
 { nom: 'Dulong' },   { nom: 'Dubois' },
 { nom: 'Delalune' }, { nom: 'Dubois' },
 { nom: 'Frere' },      {},
 {},                  { nom: 'Dubois' },
 { nom: 'Dupont' },   { nom: 'Super' },
 { nom: 'orange' },   { nom: 'Troj' },
 { nom: 'Super' },      {},
 {},                  {},
 {},                  {},
 {},                  {}
]
```

explication :

Nous utilisons la commande find en utilisant le deuxième paramètre de la commande qui nous permet de définir les attributs que nous voulons voir, nous mettons donc nom:1, _id:0.

remarque :

L'id est toujours affiché de bases, il faut pour pouvoir l'enlever définir l'attribut _id à 0.

5.3 (1.5.3) Afficher l'attribut nom des documents qui en contiennent (utiliser $exists)

requête :

```
maBDDTP2> db.maCollec1.find({nom : {$exists:true}} ,{nom : 1, _id : 0})
```

résultat :

```
[
  { nom: 'Dulong' },
  { nom: 'Dubois' },
  { nom: 'Delalune' },
  { nom: 'Dubois' },
  { nom: 'Frere' },
  { nom: 'Dubois' },
  { nom: 'Dupont' },
  { nom: 'Super' },
  { nom: 'orange' },
  { nom: 'Troj' },
  { nom: 'Super' }
]
```

explication :

La commande utilisée ressemble quasiment à celle d'avant cependant nous avons rajouter $exist:true dans le premier paramètre de la commande ce qui permet de ne pas afficher des accolades vident pour les enregistrements qui ne contiennent pas l'attribut nom.

5.4 (1.5.4) Afficher l'attribut nom sans doublons

requête :

```
maBDDTP2> db.maCollec1.distinct("nom")
```

résultat :

```
[
  'Delalune', 'Dubois',
  'Dulong',   'Dupont',
  'Frere',    'Super',
```

```
  'Troj',          'orange'
]
```

explication :

Pour afficher les différentes valeurs sans doublon d'un attribut, il suffit de faire appel avec la fonction distinct qui prend en paramètre le nom de l'attribut concerné.

5.5 (1.5.5) Afficher tous les documents sans afficher l'attribut nom

requête :

```
maBDDTP2> db.maCollec1.find({}, {nom : 0})
```

résultat :

```
[
  { _id: 3, prenom: 'Sylvie', age: 34 },
  { _id: '4', prenom: 'Sylvian' },
  { _id: '5', prenom: 'Claire' },
  {
    _id: ObjectId("68beabf078e3d7ff6b6cfc59"),
    prenom: 'Maurice',
    age: 25
  },
  { _id: 80, prenom: 'Jacques', age: 22 },
  { _id: 12, type: 'casserolle', couleur: 'verte', 'matière': 'fer' },
  { _id: 11, type: 'casserolle', couleur: 'rouge', 'matière': 'fonte' },
  { _id: '1', prenom: 'Mauricette', age: 23 },
  { _id: 2, prenom: 'Bertrand', age: 28 },
  {
    _id: 6,
    telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
  },
  { _id: 8, prenom: 'Enpierre', age: 36 },
  { _id: 5, telephone: { portable: '06 85 45 48 10' } },
  {
    _id: 7,
    prenom: 'veronique',
    telephone: { portable: '06 80 23 48 12', fixe: '02 32 41 22 16' }
  },
  { _id: 22, fruit: [ 'kiwi', 'banane', 'pomme' ] },
  { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
  { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
  { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
  { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
  { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
  { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] }
]
```

explication :

Pour cette commande il suffit de faire un find avec dans le deuxième paramètre de la commande qui contient nom : 0 ce qui permet d'afficher tous les enregistrements sans jamais afficher aucun des attributs noms.

5.6 (1.5.6) - Afficher les documents sans l'attribut nom, les données qui n'ont pas de nom

requête :

```
maBDDTP2> db.maCollec1.find({nom : {$exists:false}} ,{nom : 0})
```

résultat :

```
[
  { _id: 12, type: 'casserolle', couleur: 'verte', 'matière': 'fer' },
  { _id: 11, type: 'casserolle', couleur: 'rouge', 'matière': 'fonte' },
  { _id: 22, fruit: [ 'kiwi', 'banane', 'pomme' ] },
  { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
  { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
  { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
  { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
  { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
  { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] }
]
```

explication :

Pour effectuer cette requête on utilise un find. Dans le premier paramètre de la commande on définit qu'on ne veut seulement que les enregistrements n'ayant pas l'attribut nom grâce au nom : $exist:false. Ensuite, on ajoute dans le deuxième paramètre nom:0 pour ne pas pouvoir afficher l'attribut nom.

remarque :

Le deuxième paramètre est inutile car on affiche seulement les enregistrements qui n'ont pas de l'attribut nom.

5.7 (1.5.7) Afficher les documents dont le nom est Dubois

requête :

```
maBDDTP2> db.maCollec1.find({nom :'Dubois'})
```

résultat :

```
[
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' },
  {
        _id: ObjectId("68beabf078e3d7ff6b6cfc59"),
        nom: 'Dubois',
        prenom: 'Maurice',
        age: 25
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 }
]
```

explication :

Nous avons fait simplement un find avec en paramètre de la commande nom : Dubois pour afficher seulement les enregistrements contenant un attribut nom avec la valeur Dubois

remarque :

5.8 (1.5.8) Afficher le nom et prénom des documents dont le nom est Dubois

requête :

```
maBDDTP2> db.maCollec1.find({nom :'Dubois'},{nom: 1 ,prenom: 1 , _id : 0})
```

résultat :

```
[
  { nom: 'Dubois', prenom: 'Sylvian' },
  { nom: 'Dubois', prenom: 'Maurice' },
  { nom: 'Dubois', prenom: 'Mauricette' }
]
```

explication :

Cette requête ressemble fortement à celle d'avant seulement nous avons rempli le deuxième paramètre pour préciser qu'il faut afficher seulement l'attribut nom et prénom des enregistrement ayant un attribut nom avec la valeur Dubois.

5.9 (1.5.9) Afficher les documents dont le nom est Dubois et le prénom Mauricette

requête :

```
maBDDTP2> db.maCollec1.find({nom :'Dubois', prenom : 'Mauricette'})
```

résultat :

```
[ { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 } ]
```

explication :

Cette requête est un find qui cherche tous les enregistrements possédant un attribut nom et prénom dont la valeur sont dans l'ordre Dubois et Mauricette

5.10 (1.5.10) Afficher les documents dont le nom contient un " r "

requête :

```
maBDDTP2> db.maCollec1.find({nom : /r/i })
```

résultat :

```
[
  { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 },
  {
    _id: 6,
    nom: 'Super',
    telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
  },
  { _id: 8, nom: 'orange', prenom: 'Enpierre', age: 36 },
  { _id: 5, nom: 'Troj', telephone: { portable: '06 85 45 48 10' } },
  {
    _id: 7,
    nom: 'Super',
    prenom: 'veronique',
    telephone: { portable: '06 80 23 48 12', fixe: '02 32 41 22 16' }
  }
]
```

explication :

Nous faisons un find avec en paramètre nom : /r/i qui permet de trouver tous les enregistrements contenant un attribut nom avec un r contenu dans sa valeur.

remarque :

Le i après le regex signifie qu'on rend insensible à la casse la recherche c'est à dire qu'on recherche aussi les R majuscule.

5.11 (1.5.11) Afficher les documents dont le nom ne commence pas par un " D "

requête :

```
maBDDTP2> db.maCollec1.find({nom : {$not: /^D/ }})
```

résultat :

```
[
 { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 },
 { _id: 12, type: 'casserole', couleur: 'verte', 'matière': 'fer' },
 { _id: 11, type: 'casserole', couleur: 'rouge', 'matière': 'fonte' },
 {
        _id: 6,
        nom: 'Super',
        telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
 },
 { _id: 8, nom: 'orange', prenom: 'Enpierre', age: 36 },
 { _id: 5, nom: 'Troj', telephone: { portable: '06 85 45 48 10' } },
 {
        _id: 7,
        nom: 'Super',
        prenom: 'veronique',
        telephone: { portable: '06 80 23 48 12', fixe: '02 32 41 22 16' }
 },
 { _id: 22, fruit: [ 'kiwi', 'banane', 'pomme' ] },
 { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
 { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
 { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
 { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
 { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
 { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] }
]
```

explication :

On utilise find pour trouver dans ma Collec1 en utilisant dans le premier paramètre $not: /^D/ ce qui signifie qu'on affiche tous les enregistrements qui n'ont pas un attribut nom avec sa valeur qui commence par D.

remarque :

On retrouve aussi dans les résultats les enregistrements qui n'ont pas d'attribut nom car ils n'ont donc pas d'attribut nom commençant par d.

5.12 (1.5.12) Afficher les documents d'_id > 23. Justifier les résultats

requête :

```
maBDDTP2> db.maCollec1.find({ _id: { $gt:23}})
```

résultat :

```
[
  { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ] },
  { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] },
  { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
  { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 }
]
```

explication :

On fait un find sur maCollec1 avec pour en premier paramètre de la commande _id : $gt: 23 qui signifie qu'on cherche tous les id supérieur à 23 (gt =greater than).

justification:

Nous récupérons seulement ces informations car on utilise un int pour faire le find ce qui fait qu'on cherche que dans les _id qui sont des int. Donc si un enregistrement à pour id "24" il ne sera pas compté comme supérieur. Donc le fait d'utiliser un id en int est très pratique pour pouvoir traiter les données via l'id.

5.13 (1.5.13) Afficher les documents d'_id < "32". Justifier les résultats

requête :

```
maBDDTP2> db.maCollec1.find({ _id: { $lt:"32"}})
```

résultat :

```
[
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 },
  { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] }
]
```

explication :

On fait un find sur maCollec1 avec pour en premier paramètre de la commande _id : $lt: "32" qui signifie qu'on cherche tous les id inférieur à "32" (lt =lower than).

justification :

On récupére ces informations car on fait une recherche sur un id avec comme valeur seulement des strings. De plus la comparaison en string se fait par ordre alphabétique, c'est à dire que par exemple "4" sera compté comme plus grand par rapport "32" car 4 est supérieur à 3 vu qu'on prend caractère par caractère. Donc cela est mauvais de mettre des id en string si c'est pour faire des tris en fonction de l'id, cependant cela peut être pratique si on veut seulement avoir des noms clairs et précis sans qu'on est besoin de faire des opérations dessus.

5.14 (1.5.14) Afficher les documents dont l'âge est soit 22, soit 25 ans

requête :

```
maBDDTP2> db.maCollec1.find({age: { $in:[22,25]}})
```

résultat :

```
[
  {
        _id: ObjectId("68beabf078e3d7ff6b6cfc59"),
        nom: 'Dubois',
        prenom: 'Maurice',
        age: 25
  },
  { _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22 }
]
```

explication :

On fait un find dans ma Collec1 avec en premier paramètre age : $in[22,25] cela signifie qu'on va afficher tous les enregistrements qui ont un attribut age qui a une valeur égal à une des valeurs du tableau utilisé dans in c'est à dire 22 ou 25.

5.15 (1.5.15) Afficher les documents dont les numéros de téléphones sont "06 85 45 48 10" et "02 45 38 45 33"

requête :

```
maBDDTP2> db.maCollec1.find({"telephone.portable" : "06 85 45 48 10",
"telephone.fixe" : "02 45 38 45 33"} )
```

résultat :

```
[
  {
        _id: 6,
        nom: 'Super',
        telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
  }
]
```

explication :

On utilise un find qui permet de trouver les enregistrements qui ont un attribut téléphone et qui dans celui-ci contient deux autres attributs qui sont portable et fixe avec les valeurs dans l'ordre 06 85 45 48 10 et 02 45 38 45 33.

remarque :

l'ordre dans lequel on met telephone.fixe et telephone. portable n’a aucune importance.

5.16 (1.5.16) Afficher les documents dont le numéro de mobile est "06 85 45 48 10"

requête :

```
maBDDTP2> db.maCollec1.find({"telephone.portable" : "06 85 45 48 10"} )
```

résultat :

```
[
 { _id: 5, nom: 'Troj', telephone: { portable: '06 85 45 48 10' } },
 {
        _id: 6,
        nom: 'Super',
        telephone: { portable: '06 85 45 48 10', fixe: '02 45 38 45 33' }
 }
]
```

explication :

Cette projection comme celle d’au-dessus, cependant cette fois on recherche seulement les enregistrements qui ont un attribut téléphone avec à l'intérieur un attribut portable égal à 06 85 45 48 10.

5.17 (1.5.17) Afficher les documents contenant le fruit" orange ".

requête :

```
maBDDTP2> db.maCollec1.find({"fruit" : "orange"} )
```

résultat :

```
[
 { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
 { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
 { _id: 23, fruit: [ 'kiwi', 'orange', 'poire' ] },
 { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] },
 { _id: 25, fruit: [ 'cerise', 'banane', 'orange' ] }
]
```

explication :

On fait un find sur maCollec1 ou on recherche les enregistrements qui ont un attribut fruit est qui ont pour valeur “orange”.

remarque :

Par rapport aux attributs avec des attributs à l'intérieur, les valeurs des attributs qui ont comme valeur un tableau avec plusieurs valeurs à l'intérieur peuvent être affichées si une valeur du tableau est présente dans le find si son nom d'attribut correspond au find.

5.18 (1.5.18) Afficher les documents dont le premier fruit est " orange ".

requête :

```
maBDDTP2> db.maCollec1.find({"fruit.0" : "orange"} )
```

résultat :

```
[
  { _id: '27', fruit: [ 'orange', 'prune', 'cerise' ] },
  { _id: 26, fruit: [ 'orange', 'pomme', 'raisin' ] },
  { _id: 21, fruit: [ 'orange', 'banane', 'pomme' ] }
]
```

explication :

On utilise un find sur maCollec1 qui recherche dans les attributs qui s'appellent fruit et qui sont des tableaux et que leur première valeur soit égal à "orange".

5.19 (1.5.19) Donner l'age de 34 ans à tous les Dubois.

requête :

```
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 25
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' }
]
maBDDTP2> db.maCollec1.updateMany({nom : "Dubois"},{ $set : {age: 34} } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 34
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian', age: 34 }
]
```

explication :

Les find de début et de fin sont justes pour montrer le bon de changement de données de l'âge des dubois. Ensuite, on utilise l'updateMany qui cherches dans les enregistrements de la collection ce qui ont un attribut nom avec la valeur Dubois et change leur attribut âge à 34.

remarque :

Si l’attribut age n'existe pas dans un enregistrement qui a l’attribut nom avec la valeur Dubois, il créera l’attribut age en initialisant à 34

5.20 (1.5.20) Donner l'age de 36 ans au premier Dubois.

requête :

```
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 34
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian', age: 34 }
]
maBDDTP2> db.maCollec1.updateOne({nom : "Dubois"},{ $set : {age: 36} } )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 36
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian', age: 34 }
]
```

explication :

Les deux find sont pour montrer l'évolution après l'update. Pour changer le premier Dubois, il suffit simplement de faire un update one avec comme premier paramètre nom:Dubois pour trouver un enregistrement avec l'attribut nom avec en valeur Dubois. Il change l'attribut age en le mettant à 36.

remarque :

5.21 (1.5.21) Donner l'age de 19 ans au second Dubois (merci chatGPT)

requête :

```
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
```

```
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 36
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 34 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian', age: 34 }
]
maBDDTP2>  db.maCollec1.updateOne ( { _id: {$in : [db.maCollec1.find({nom :
"Dubois"}).skip(1).limit(1).next()._id]}},{$set: {age : 19}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1, age : 1 })
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice',
          age: 36
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 19 },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian', age: 34 }
]
```

explication :

Pour pouvoir changer l'age du deuxième Dubois, on fait un update one sur le deuxième enregistrement de Dubois dans la collection. On se l'assure en récupérant tous les id et en sélectionnant le deuxième en ignorant le premier de la liste pour faire la modification du deuxième.

5.22 (1.5.22) Donner l'age de 35 ans à Dudouit. S'il n'existe pas, le créer.

requête :

```
maBDDTP2> db.maCollec1.find({nom: "Dudouit"})

maBDDTP2> db.maCollec1.updateOne({nom : "Dudouit"}, {$set: {age: 35}}, {upsert :
true})
{
  acknowledged: true,
  insertedId: ObjectId('68c0650641cf59832fef3db0'),
  matchedCount: 0,
```

```
  modifiedCount: 0,
  upsertedCount: 1
}
maBDDTP2> db.maCollec1.find({nom: "Dudouit"})
[
  {
          _id: ObjectId('68c0650641cf59832fef3db0'),
          nom: 'Dudouit',
          age: 35
  }
]
```

explication :

Alors pour s'assurer de bien donner l'age de 32 ans à l'enregistrement qui à le nom Dudouit, il faut faire lors de l'updateOne, ajouter un paramètre à la commande en mettant upsert à true. En effet cela permettra dans un premier temps s'il n'existe pas de créer directement le document en mettant un id de base et en créant l'attribut nom et l'age au bonne valeur. S'il existe déjà, l'attribut age sera juste simplement modifier.

remarque :

5.23 (1.5.23) Remplacer le document 24 par celui-ci {"_id" : 24 , fruit:["mandarine","citron","prune"] }). ◦ Quelle est la différence entre update et replace ?
requête :

```
maBDDTP2> db.maCollec1.find({_id:24})
[ { _id: 24, fruit: [ 'cerise', 'peche', 'citron' ], age: 36 } ]

maBDDTP2> db.maCollec1.updateOne({_id:24},{$set:{fruit :
["poire","citron","pomme"]}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
maBDDTP2> db.maCollec1.find({_id:24})
[ { _id: 24, fruit: [ 'poire', 'citron', 'pomme' ], age: 36 } ]

maBDDTP2> db.maCollec1.replaceOne({_id:24},{fruit :
["mandarine","citron","prune"]})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
```

```
  upsertedCount: 0
}
maBDDTP2> db.maCollec1.find({_id:24})
[ { _id: 24, fruit: [ 'mandarine', 'citron', 'prune' ] } ]
```

explication :

Alors la principale différence entre update et replace et que lorsqu'on utilise replace l'enregistrement et totalement remplacé par les attributs et les valeurs que l'on met dans le deuxième paramètre de la commande. Ce qui écrase totalement les anciennes données. Alors que lorsqu'on update un enregistrement, si l'attribut n'existe pas, la commande l'ajoute et n'écrase pas les autres attributs qui ne sont pas précisés dans l'update one. On peut le voir dans le premier exemple que l'on update que l'attribut fruit et garde l'age alors que lorsqu'on utilise replace il reste que l'id et le tableau de fruit.

remarque :

5.24 (1.5.24) Supprimer le document 4

requête :

```
maBDDTP2> db.maCollec1.find({_id:'4'})
[ { _id: 4, nom: 'Romuald' } ]
maBDDTP2> db.maCollec1.deleteOne({_id:'4'})
{ acknowledged: true, deletedCount: 1 }
maBDDTP2> db.maCollec1.find({_id:'4'})
```

Mais on pourrait aussi utiliser deleteMany pour en supprimer plusieurs :

```
maBDDTP2> db.maCollec1.find({_id:'4'})
[ { _id: 4, nom: 'Romuald' } ]
maBDDTP2> db.maCollec1.deleteMany({_id:'4'})
{ acknowledged: true, deletedCount: 1 }
maBDDTP2> db.maCollec1.find({_id:'4'})
```

résultat :

```
maBDDTP2> db.maCollec1.find( {_id:'4'})
[{ _id: '4', nom: 'Dubois', prenom: 'Sylvian' }]
maBDDTP2> db.maCollec1.deleteOne({_id:'4'})
{ acknowledged: true, deletedCount: 1 }
maBDDTP2> db.maCollec1.find( {_id:'4'})
```

explication :

On a commencé par rechercher la personne avec l'id 4 via la fonction find. On effectue ensuite un deleteOne vu qu'un id est unique, il n'y a qu'un enregistrement a supprimer. On a ensuite constaté que l'enregistrement à disparu via un find.

remarque :

On aurait pu faire un deleteMany pour supprimer tous les id 4. Mais vu qu'un id est unique il n'y en a qu'un. Cependant le deleteMany est utile si on aurait voulu par exemple supprimer tous les Dubois.

5.25 (1.5.25) Supprimer le premier Dubois

requête :

```
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1})
[
  {
          _id: ObjectId('68bf36e50570d0b42bec209f'),
          nom: 'Dubois',
          prenom: 'Maurice'
  },
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette' },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' }
]
maBDDTP2> db.maCollec1.deleteOne({nom : "Dubois"})
{ acknowledged: true, deletedCount: 1 }
maBDDTP2> db.maCollec1.find({nom:"Dubois"},{nom:1 , prenom : 1})
[
  { _id: '1', nom: 'Dubois', prenom: 'Mauricette' },
  { _id: '4', nom: 'Dubois', prenom: 'Sylvian' }
]
```

explication :

Pour supprimer le premier enregistrement, il suffit de faire un deleteOne ce qui aura pour effet de supprimer le premier enregistrement que la commande trouve. On peut l'obersver car il supprime le premier Dubois que l'on trouve de fin avant de faire le deleteOne.

6. Essai de requête de modification plus complexe

6.1 (1.6.1) Créer une nouvelle collection : maCollec2.

résultat :

```
maBDDTP2> show collections
maCollec1
maBDDTP2> db.createCollection("maCollec2")
{ ok: 1 }
maBDDTP2> show collections
maCollec1
maCollec2
```

Nous créons la collection maCollec2 grâce à la commande createCollection. On vérifie et observe la différence avec la commande show collections qui affiche les collections.

6.2 (1.6.2) Créer 7 documents à partir d'un fichier p16_X.Json :

requête :

```
myBDDTP2> load('C:/Users/Utilisateur/Nextcloud/S5/R5.10/code/p16_3.json')
```

résultat :

```
myBDDTP2> db.maCollec2.find()

myBDDTP2> load('C:/Users/Utilisateur/Nextcloud/S5/R5.10/code/p16_3.json')
true
myBDDTP2> db.maCollec2.find()
[
  {
    _id: '1',
    titre: 'Les misérables',
    auteur: 'Victor Hugo',
    nbExemplaires: 100
  },
  {
    _id: '2',
    titre: 'Notre-Dame de Paris',
    auteur: 'Victor Hugo',
    nbExemplaires: 30
  },
  {
    _id: '3',
    titre: 'La Bête humaine',
    auteur: 'Emile Zola',
    nbExemplaires: 19
  },
  {
    _id: '4',
    titre: 'Germinal',
    auteur: 'Emile Zola',
    nbExemplaires: 100
  },
  {
    _id: '6',
    titre: 'Notre-Dame de Paris le retour',
    auteur: 'Victor Hugo',
    nbExemplaires: 20
  },
  {
    _id: '7',
    titre: 'Notre-Dame de Paris contre les misérables',
    auteur: 'Victor Hugo',
    nbExemplaires: 40
  },
  {
    _id: '9',
    titre: "Le Dernier Jour d'un confiné",
    auteur: 'Victor Hugo',
    nbExemplaires: 64
  }
]
myBDDTP2>
```

explication :

Nous avons d'abord réalisé une première commande find() afin de voir que maCollec2 est vide. Puis nous avons fait la commande load afin de charger le fichier p16_3.json qui est fourni. Celui-ci nous a créé 7 documents. Nous pouvons

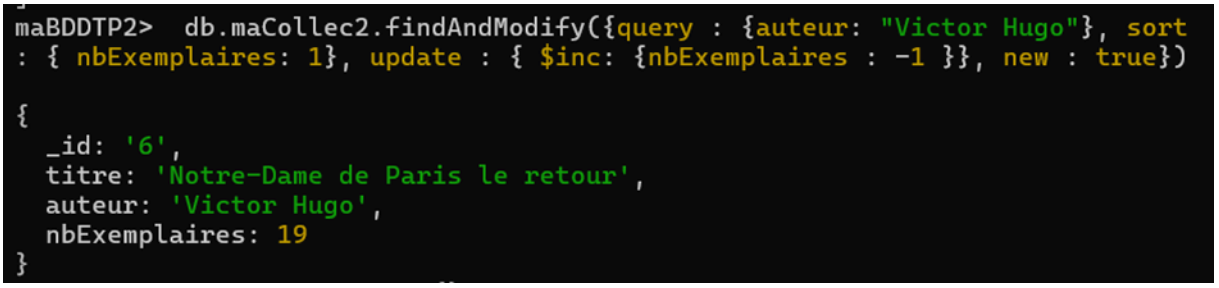
constater que la création est valide avec un second find, affichant le résultat de la fonction.

6.3 (1.6.3) En utilisant la fonction findAndModify (voir annexe), retirer un exemplaire du livre de Victor Hugo comportant le moins d'exemplaires.

requête :

```
maBDDTP2>  db.maCollec2.findAndModify({query : {auteur: "Victor Hugo"}, sort: { nbExemplaires: 1}, update : { $inc: {nbExemplaires : -1 }}, new : true})
```

résultat :



explication :

La fonction FindAndModify est une fonction cherche et modifie un document. Pour ce faire, ici nous utilisons les paramètres query qui sont les filtres, ici on veut les documents de l'auteur “Victor Hugo”. Ensuite le paramètre sort afin de les classer dans l’ordre du nombre d'exemplaires. La fonction FindAndModify ne modifiera que le premier document de la liste. On utilise le paramètre update afin de mettre à jour le nombre d'exemplaires en incrémentant d’un. Puis le paramètre new à true afin qu’après l'exécution de la modification le document est affiché.

6.4 (1.6.4) En utilisant la fonction findAndModify (voir annexe), ajouter un exemplaire au livre de Victor Hugo " les travailleurs de la mer" . L'exemplaire sera créé s'il n'existe pas. Répéter l’opération et tester.

requête :

```
maBDDTP2> db.maCollec2.findAndModify({query : {auteur: "Victor Hugo", titre : "Les travailleurs de la mer"}, update: {$inc: {nbExemplaires : 1}}, new : true, upsert : true})
{
  _id: ObjectId('68c08e69b7957397654cc1e8'),
  auteur: 'Victor Hugo',
  titre: 'Les travailleurs de la mer',
  nbExemplaires: 1
}
maBDDTP2> db.maCollec2.findAndModify({query : {auteur: "Victor Hugo", titre : "Les travailleurs de la mer"}, update: {$inc: {nbExemplaires : 1}}, new : true, upsert : true})
```

```
{
  _id: ObjectId('68c08e69b7957397654cc1e8'),
  auteur: 'Victor Hugo',
  titre: 'Les travailleurs de la mer',
  nbExemplaires: 2
}
```

résultat :

```
maBDDTP2> db.maCollec2.find({auteur : "Victor Hugo",titre : "Les travailleur
s de la mer"})

maBDDTP2> db.maCollec2.findAndModify({query : {auteur: "Victor Hugo", titre
: "Les travailleurs de la mer"}, update: {$inc: {nbExemplaires : 1}}, new :
true, upsert : true})
{
  _id: ObjectId('68c08e69b7957397654cc1e8'),
  auteur: 'Victor Hugo',
  titre: 'Les travailleurs de la mer',
  nbExemplaires: 1
}
maBDDTP2> db.maCollec2.findAndModify({query : {auteur: "Victor Hugo", titre
: "Les travailleurs de la mer"}, update: {$inc: {nbExemplaires : 1}}, new :
true, upsert : true})
{
  _id: ObjectId('68c08e69b7957397654cc1e8'),
  auteur: 'Victor Hugo',
  titre: 'Les travailleurs de la mer',
  nbExemplaires: 2
}
```

explication :

Nous avons d'abord procédé à un find afin de savoir s' il existe ou pas. Ce premier find était facultatif mais nous avons souhaité le faire afin de mieux comprendre le fonctionnement. Nous passons ensuite a la requete FindAndModify. Cette requete prend pour parametre query, les filtres avec l'auteur et le nom du livre. Elle prend aussi en parametre update pour mettre à jour en incrémentant de un le nombre d'exemplaires. Nous avons aussi mis le paramètre new afin d'afficher le retour de la fonction. Nous avons rajouté le paramètre upsert à true afin que si ce livre n'existe pas, il soit créé. Si il existe, il a juste à mettre à jour. Ici le premier appel créé le document et le second le met à jour.

remarque :

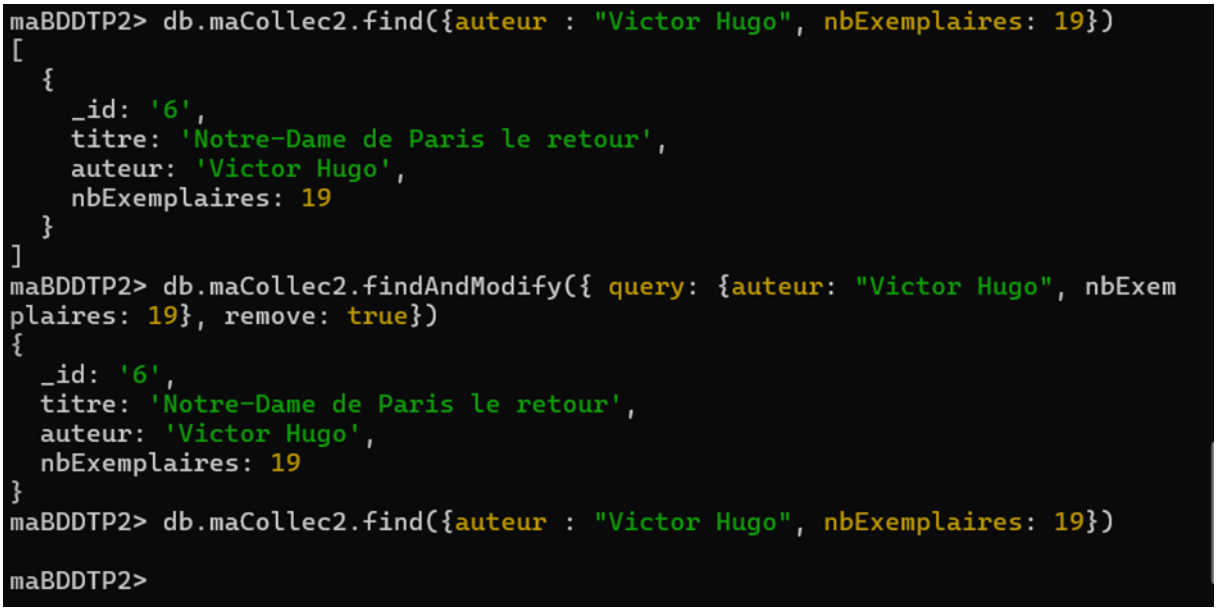
On observe que upsert permet de créer un document lors d'une modification si il ne connaît pas. De plus, il met dans ce document un maximum d'informations qu'il connaît, ici le titre, l'auteur et le nombre d'exemplaires. Cependant l'id est décidé par l'ordinateur, il n'est pas défini par une incrémentation de int.

6.5 (1.6.5) En utilisant la fonction findAndModify (voir annexe), supprimer le livre de Victor Hugo comportant 19 exemplaires.

requete :

```
maBDDTP2> db.maCollec2.findAndModify({ query: {auteur: "Victor Hugo",
nbExemplaires: 19}, remove: true})
{
  _id: '6',
  titre: 'Notre-Dame de Paris le retour',
  auteur: 'Victor Hugo',
  nbExemplaires: 19
}
```

résultat :



explication :

Nous avons commencé par réaliser un find afin de trouver le livre de Victor Hugo ayant 19 exemplaires. Ce find est facultatif, c'était pour prouver qu'il existait bien avant. Ensuite nous avons effectué le findAndModify. Dans ce findandmodify nous utilisons le parametre query pour filtrer avec l'auteur et le nombre d'exemplaires. Et le parametre remove à true afin de supprimer s' il trouve un exemplaire. Nous avons ensuite fait un nouveau find et nous remarquons que celui-ci ne renvoie rien, que le document a bien été supprimé.

remarques :

On remarque que le document supprimer est affiché par la fonction.

6.6 (1.6.6) Afficher les enregistrements triés du plus grand au plus petit nombre d'exemplaires.

requetes :

```
maBDDTP2> db.maCollec2.find().sort({ nbExemplaires: -1 })
```

resultat :

```
maBDDTP2> db.maCollec2.find().sort({ nbExemplaires: -1 })
[
  {
    _id: '1',
    titre: 'Les misérables',
    auteur: 'Victor Hugo',
    nbExemplaires: 100
  },
  {
    _id: '4',
    titre: 'Germinal',
    auteur: 'Emile Zola',
    nbExemplaires: 100
  },
  {
    _id: '9',
    titre: "Le Dernier Jour d'un confiné",
    auteur: 'Victor Hugo',
    nbExemplaires: 64
  },
  {
    _id: '7',
    titre: 'Notre-Dame de Paris contre les misérables',
    auteur: 'Victor Hugo',
    nbExemplaires: 40
  },
  {
    _id: '2',
    titre: 'Notre-Dame de Paris',
    auteur: 'Victor Hugo',
    nbExemplaires: 30
  },
  {
    _id: '3',
    titre: 'La Bête humaine',
    auteur: 'Emile Zola',
    nbExemplaires: 19
  },
  {
    _id: ObjectId('68c08e69b7957397654cc1e8'),
    auteur: 'Victor Hugo',
    titre: 'Les travailleurs de la mer',
    nbExemplaires: 2
  }
]
maBDDTP2>
```

explication :

Nous faisons la fonction find afin d'afficher les documents de la collection. Nous avons rajouter la fonction sort afin de les trié dans l'ordre du nombre d'exemplaire.

remarques :

On remarque que sort prend en paramètre le nom de l'attribut sur lequel il faut effectué le tri. Puis ici il prend -1 pour les classées dans l'ordre décroissant. Un 1 les aurait classé dans l'ordre croissant.

6.7 (1.6.7) Donner le nombre de documents de la collection.

Requête :

```
maBDDTP2> db.maCollec2.countDocuments()
```

Résultat :

Explication :

Pour donner le nombre de documents dans la collection maCollec2 pour ce faire, nous faisons appel à la fonction countDocuments(). Celle-ci compte le nombre de documents et renvoie cette valeur.

7. Essai de requête d'agrégation

7.1 (1.7.1) Créer une nouvelle collection : maCollec3

```
maBDDTP2> show collections
maCollec1
maCollec2
maBDDTP2> db.createCollection("maCollec3")
{ ok: 1 }
maBDDTP2> show collections
maCollec1
maCollec2
maCollec3
```

On crée une collection maCollec3 dans la base de données maBDDTP2 grâce à createCollection.

7.2 (1.7.2) Charger la collection p17_X.json

```
maBDDTP2> db.maCollec3.find()
```

```
maBDDTP2> load('C:/Users/Utilisateur/Nextcloud/S5/R5.10/code/p17_3.json')
true
maBDDTP2> db.maCollec3.find()
[
  {
        _id: '31',
        ville: 'Paris',
        pays: 'France',
        pop: 2187526,
        loc: [ 15, 20 ]
  },....
```

Nous avons choisi d'utiliser le load pour pouvoir le faire directement dans le mongo shell. On a observé ensuite que le load avait bien marché grâce à un find sur maCollec3.

7.3 (1.7.3) Afficher les documents

```
maBDDTP2> db.maCollec3.find()
[
{_id: '31',       ville: 'Paris',   pays: 'France',          pop: 2187526,         loc: [ 15, 20 ] },
{_id: '32',       ville: 'Marseille',      pays: 'France',          pop: 862211, loc: [ 16, 28 ] },
{          _id: '33',        ville: 'Rouen',pays: 'France',          pop: 515695, loc: [ 16, 28 ] },
{          _id: 32,          ville: 'Caen',   pays: 'France',          pop: 110000, loc: [ 10, 25 ] },
{_id: '34',       ville: 'Ifs ',      pays: 'France',          pop: 1111500,         loc: [ 1, 5 ] },
{          _id: '35',        ville: 'Moult', pays: 'France',          pop: 3200,    loc: [ 0, 0 ] },
{_id: '36',ville: 'Berlin',pays: 'Allemagne', pop: 3748148,         loc: [ 16, 30 ] },
{_id: '37',ville: 'Hambourg', pays: 'Allemagne',    pop: 1645095,         loc: [ 20, 30 ] },
{_id: '38',ville: 'Munich',pays: 'Allemagne',          pop: 1298941,         loc: [ 20, 35 ] },
{_id: '42',ville: 'Glasgow',    pays: 'Royaume Uni',          pop: 765030, loc: [ 0, 0 ] },
{_id: '43',ville: 'Birmingham',pays: 'Royaume Uni',pop: 1013395,      loc: [ 0, 0 ] },
{_id: '44',ville: 'Londres',pays: 'Royaume Uni',pop: 6574009,   loc: [ 55, 7 ] },
{_id: '45',       ville: 'Rome', pays: 'Italie',  pop: 2855397,         loc: [ 16, 15 ] },
{          _id: '46',        ville: 'Milan',  pays: 'Italie',  pop: 1378689,         loc: [ 34, 10 ] },
{          _id: '47',        ville: 'Naples',          pays: 'Italie',  pop: 957075, loc: [ 8, 7 ] }
]
```

7.4 (1.7.4) Tester et commenter les requêtes suivantes :
7.4.1 (1.7.4.1) db.maCollec3.aggregate({$group:{_id:null,population:{ $max:"$pop"} } })

résultat :

```
[ { _id: null, population: 6574009 } ]
```

explication :

Cette requête regroupe les différents enregistrements par leur attribut population et on retrouve ensuite la population maximum qu'on retrouve dans la collection.

On regroupe

7.4.2 (1.7.4.2) db.maCollec3.aggregate({$group:{_id:"$pays",population:{ $max:"$pop"} } })

résultat :

```
[
  { _id: 'Italie', population: 2855397 },
  { _id: 'Allemagne', population: 3748148 },
  { _id: 'Royaume Uni', population: 6574009 },
  { _id: 'France', population: 2187526 }
]
```

explication :

On regroupe les différents enregistrements par leur attribut pays. C'est-à-dire que chaque enregistrement est rangé par leur pays. Ensuite avec la fonction max on trouve la population max de la ville du pays

7.4.3 (1.7.4.3) db.maCollec3.aggregate({$group:{_id:"$pays",population:{ $avg:"$pop"} } })

résultat :

```
[
 { _id: 'Italie', population: 1730387 },
 { _id: 'Royaume Uni', population: 2784144.6666666665 },
 { _id: 'France', population: 798355.3333333334 },
 { _id: 'Allemagne', population: 2230728 }
]
```

explication :

Cette requête permet de ranger encore une fois les enregistrements de la maCollec3 par pays cependant au lieu de trouver la ville ayant la plus grande population du pays, on calcule la moyenne de la population par ville pour chaque pays distinct présent dans les enregistrements.

7.4.4 (1.7.4.4) db.maCollec3.aggregate({$group:{_id:"$pays",population:{ $sum:"$pop"} } })

résultat :

```
[
 { _id: 'Italie', population: 5191161 },
 { _id: 'France', population: 4790132 },
 { _id: 'Royaume Uni', population: 8352434 },
 { _id: 'Allemagne', population: 6692184 }
]
```

explication :

Cette requête permet de ranger encore une fois les enregistrements de la maCollec3 par pays. Cependant cette fois-ci on calcule le nombre total de population pour tous les enregistrements ayant le même pays.

7.4.5 (1.7.4.5) db.maCollec3.aggregate({$group:{_id:"$pays",population:{$sum:"$pop"}}},{$sort:{population:1}})

résultat

```
[
 { _id: 'France', population: 4790132 },
 { _id: 'Italie', population: 5191161 },
 { _id: 'Allemagne', population: 6692184 },
 { _id: 'Royaume Uni', population: 8352434 }
]
```

explication :

Cette requête calcule la somme du nombre d'habitants des enregistrements ayant la même valeur dans l'attribut pays. Ensuite cela tri, les pays par population croissante.

7.4.6 (1.7.4.6) db.maCollec3.aggregate({$match:{pays:"France"} })

résultat :

```
maBDDTP2> db.maCollec3.aggregate( {$match:{pays:"France"} } )
[
 {
        _id: '31',
        ville: 'Paris',
        pays: 'France',
        pop: 2187526,
        loc: [ 15, 20 ]
 },
 {
        _id: '32',
        ville: 'Marseille',
        pays: 'France',
        pop: 862211,
        loc: [ 16, 28 ]
 },
 {
        _id: '33',
        ville: 'Rouen',
        pays: 'France',
        pop: 515695,
        loc: [ 16, 28 ]
 },
 {
        _id: 32,
        ville: 'Caen',
        pays: 'France',
        pop: 110000,
        loc: [ 10, 25 ]
 },
 {
        _id: '34',
        ville: 'Ifs ',
        pays: 'France',
        pop: 1111500,
        loc: [ 1, 5 ]
 },
 {
        _id: '35',
```

```
        ville: 'Moult',
        pays: 'France',
        pop: 3200,
        loc: [ 0, 0 ]
}
]
```

explication :

Cette requête permet de trier les enregistrements d'une collection par rapport à des attributs déterminés ici pays : France. Cela est très utile pour appliquer une fonction d'agrégat à une certaine partie des enregistrements d'une collection.

7.5 (1.7.5) En déduire la requête affichant la ville française ayant le moins d'habitants.

requête :

```
maBDDTP2> db.maCollec3.aggregate([{ $match: {pays: "France"}}, {$sort: {pop:1}}, {$limit:1}])
```

explication :

On trie donc les enregistrements comme dans la requête précédente grâce à la fonction match pour récupérer toutes les villes de France. On les trie ensuite par ordre croissant grâce à sort puis on limite le nombre d'enregistrements grâce à limity ce qui permet de trouver la ville de France avec le moins d'habitant.

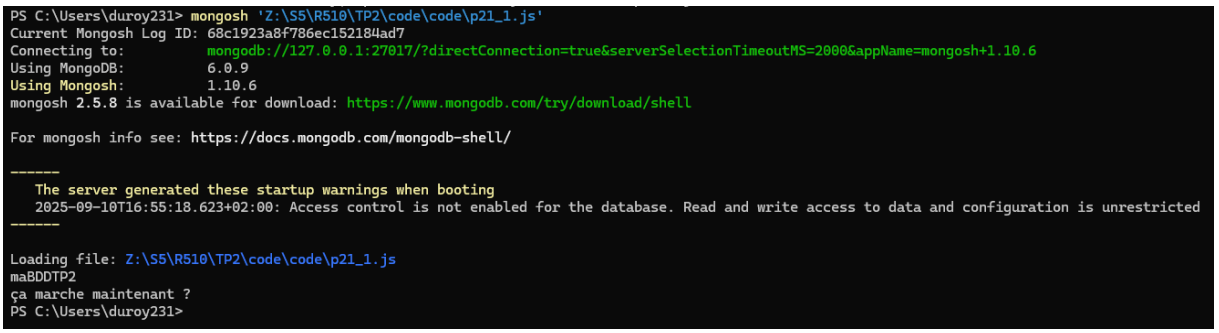
8. Comment accéder à mongodb avec un script ?

8.1 (2.1.1) Avec un autre interprète de commande lancer mongosh p21_1.js .

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_1.js'
```

resultat :



explication :

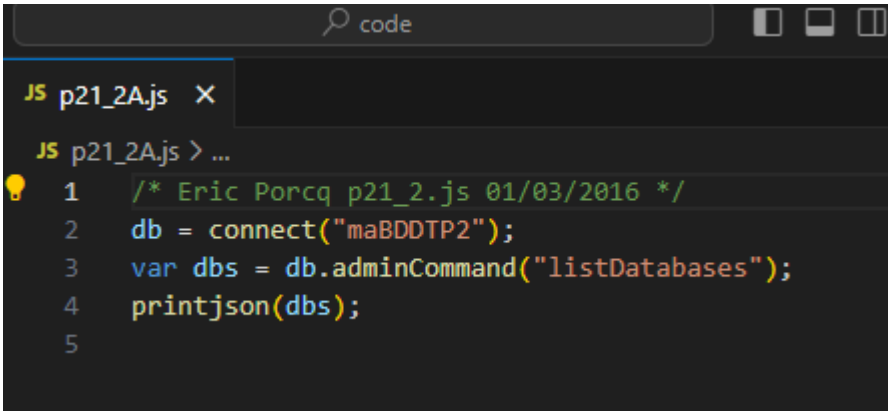
Cela permet de se connecter temporairement au serveur en lançant le script. Cette connexion reste ephemere le temps de l'exécution.

8.2 (2.1.2) Tester et étudier de la même façon p21_2A.js et p21_2B.js

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_2A.js'
```

fichier p21_2A.js :



resultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_2A.js'
Current Mongosh Log ID: 68c1934d7d361dcc91bf005a
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.6
Using MongoDB:          6.0.9
Using Mongosh:          1.10.6
mongosh 2.5.8 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_2A.js
{
  databases: [
    {
      name: 'admin',
      sizeOnDisk: Long("40960"),
      empty: false
    },
    {
      name: 'config',
      sizeOnDisk: Long("73728"),
      empty: false
    },
    {
      name: 'local',
      sizeOnDisk: Long("73728"),
      empty: false
    },
    {
      name: 'maBDDTP2',
      sizeOnDisk: Long("57344"),
      empty: false
    }
  ],
  totalSize: Long("245760"),
  totalSizeMb: Long("0"),
  ok: 1
}
```

explication :

Ici on lance le script p21_2A.js, celui se connecte à la base de donnée, effectue la demande de liste de base de données avec leur détails. Et l'affiche au format json dans le terminale.

requete :

```
mongosh 'Z:\S5\R510\TP2\code\code\p21_2B.js'
```

fichier p21_2B.js :

```
1  /* Eric Porcq p21_3.js 01/03/2016 */
2  db = connect("maBDDTP2");
3  var doc = db.maCollec2.find({},{auteur:1});
4  doc.forEach(function(val)
5  {
6    printjson(val);
7  });
```

resultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_2B.js'
Current Mongosh Log ID: 68c195a1733ed0f7697920046
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.6
Using MongoDB:          6.0.9
Using Mongosh:          1.10.6
mongosh 2.5.8 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_2B.js
{
  _id: '1',
  auteur: 'Victor Hugo'
}
{
  _id: '2',
  auteur: 'Victor Hugo'
}
{
  _id: '3',
  auteur: 'Emile Zola'
}
{
  _id: '4',
  auteur: 'Emile Zola'
}
{
  _id: '7',
  auteur: 'Victor Hugo'
}
{
  _id: '9',
  auteur: 'Victor Hugo'
}
{
  _id: ObjectId("68c19582b35533b3e9210d64"),
  auteur: 'Victor Hugo'
}
PS C:\Users\duroy231>
```

explication :

Ici on se connecte à la base de donnée, et on affiche les id par défaut et les auteurs de maCollec2. Le fichier utilise un foreach pour lire chaque ligne et finit par envoyer et afficher différents fichier json.

8.3 (2.1.3) En utilisant printjson, n'afficher que les documents possédant un champ " nom " (p21_3.js).

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_3.js'
```

fichier p21_3.js :

```
/* Cyprien Duroy et Alexandre Le Roy p21_3.js 10/09/2025 */
db = connect("maBDDTP2");
var doc = db.maCollec1.find({nom : { $exists: true }});
doc.forEach(function(val)
{
  printjson(val);
});
```

Resultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_3.js'
Current Mongosh Log ID: 68c197b2b8c8104bc2368cac
```

```
Connecting to:
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2
000&appName=mongosh+1.10.6
Using MongoDB:        6.0.9
Using Mongosh:        1.10.6
mongosh 2.5.8 is available for download:
https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database.
Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_3.js
{ _id: 3, nom: 'Dulong', prenom: 'Sylvie', age: 34}
{ _id: '4', nom: 'Dubois', prenom: 'Sylvian'}
{ _id: '5', nom: 'Delalune', prenom: 'Claire'}
{ _id: ObjectId("68beabf078e3d7ff6b6cfc59"), nom: 'Dubois', prenom: 'Maurice',
age: 25}
{ _id: 80, nom: 'Frere', prenom: 'Jacques', age: 22}
{ _id: '1', nom: 'Dubois', prenom: 'Mauricette', age: 23}
{ _id: 2, nom: 'Dupont', prenom: 'Bertrand', age: 28}
{ _id: 6, nom: 'Super', telephone: {   portable: '06 85 45 48 10',   fixe: '02 45 38 45
33' }}
{ _id: 8, nom: 'orange', prenom: 'Enpierre', age: 36}
{ _id: 5, nom: 'Troj', telephone: {   portable: '06 85 45 48 10' }}
{ _id: 7, nom: 'Super', prenom: 'veronique', telephone: {   portable: '06 80 23 48
12',   fixe: '02 32 41 22 16' }}
PS C:\Users\duroy231>
```

explication :

Dans le fichier javascript, on commence par se connecter à la base de donnée. Puis on fait une recherche avec un find et un filtre sur le fait qu'il y est un nom. Comme on aurait pu faire sur le serveur. Puis on affiche les données via une boucle foreach avec un printjson qui affiche les données json.

8.4 (2.1.4) Compléter ce programme pour n'afficher que l'id et le nom (p21_4.js)

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_4.js'
```

fichier p21_4.js :

```
/* Cyprien Duroy et Alexandre Le Roy p21_4.js 10/09/2025 */
db = connect("maBDDTP2");
```

```
var doc = db.maCollec1.find({nom : { $exists: true }},{nom:1});
doc.forEach(function(val)
{
  printjson(val);
});
```

resultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_4.js'
Current Mongosh Log ID: 68c19990323fb68508b0ba3b
Connecting to:
mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2
000&appName=mongosh+1.10.6
Using MongoDB:        6.0.9
Using Mongosh:        1.10.6
mongosh 2.5.8 is available for download:
https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
  The server generated these startup warnings when booting
  2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database.
Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_4.js
{ _id: 3,  nom: 'Dulong'}
{ _id: '4',  nom: 'Dubois'}
{ _id: '5',  nom: 'Delalune'}
{ _id: ObjectId("68beabf078e3d7ff6b6cfc59"),  nom: 'Dubois'}
{ _id: 80,  nom: 'Frere'}
{ _id: '1',  nom: 'Dubois'}
{ _id: 2,  nom: 'Dupont'}
{ _id: 6,  nom: 'Super'}
{ _id: 8,  nom: 'orange'}
{ _id: 5,  nom: 'Troj'}
{ _id: 7,  nom: 'Super'}
PS C:\Users\duroy231>
```

Explication :

Nous avons complété le code de la fonction précédente, en ajoutant un paramètre au find. Le second parametre “ {nom:1}”. Cela fait que ça affiche l’id automatiquement et le nom vu qu’il est demandé.

8.5 (2.1.5) En utilisant insert, ajouter dans macollec4, 10 enregistrements composés uniquement de i variant de 1 à 10 (p21_5.js). Vérifier avec le client connecté (mongosh)

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_5.js'
```

Dans mangosh :

```
maBDDTP2> db.maCollec4.find()
```

fichier p21_5.js :

```
/* Cyprien Duroy et Alexandre Le Roy p21_5.js 01/03/2016 */
for (let j = 1; j<=10; j++){
    db.maCollec4.insert({i : j})
}
```

résultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_5.js'
Current Mongosh Log ID: 68c19b5e0814fe93c634d838
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.6
Using MongoDB:          6.0.9
Using Mongosh:          1.10.6
mongosh 2.5.8 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_5.js
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
PS C:\Users\duroy231>
```

```
maBDDTP2> db.maCollec4.find()
[
  { _id: ObjectId("68c19c2b6300708704031eaf"), i: 1 },
  { _id: ObjectId("68c19c2b6300708704031eb0"), i: 2 },
  { _id: ObjectId("68c19c2b6300708704031eb1"), i: 3 },
  { _id: ObjectId("68c19c2b6300708704031eb2"), i: 4 },
  { _id: ObjectId("68c19c2b6300708704031eb3"), i: 5 },
  { _id: ObjectId("68c19c2b6300708704031eb4"), i: 6 },
  { _id: ObjectId("68c19c2b6300708704031eb5"), i: 7 },
  { _id: ObjectId("68c19c2b6300708704031eb6"), i: 8 },
  { _id: ObjectId("68c19c2b6300708704031eb7"), i: 9 },
  { _id: ObjectId("68c19c2b6300708704031eb8"), i: 10 }
]
maBDDTP2>
```

explication :

Pour le fichier JavaScript, on se connecte à la Base de données, on fait une boucle insérant à chaque fois i dans maCollec4. On effectue ensuite un find dans mangosh afin de voir le résultat et que les 10 ont bien été inséré.

remarque :

On remarque que maCollec4 est créé si elle n'existe pas.

8.6 (2.1.6) Supprimer les 8 premiers enregistrements (p21_6.js)

requete :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_6.js'
```

dans mangosh :

```
maBDDTP2> db.maCollec4.find()
```

fichier p21_6.js :

```
/* Cyprien Duroy et Alexandre Le Roy p21_6.js 01/03/2016 */
db = connect("maBDDTP2");
db.maCollec4.find().sort({ i: 1 }).limit(8).forEach(function(doc) {
    db.maCollec4.deleteOne({ _id: doc._id });
  });
```

resultat :

```
PS C:\Users\duroy231> mongosh 'Z:\S5\R510\TP2\code\code\p21_6.js'
Current Mongosh Log ID: 68c19d4b20988b88de7524c9
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.10.6
Using MongoDB:          6.0.9
Using Mongosh:          1.10.6
mongosh 2.5.8 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-09-10T16:55:18.623+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

Loading file: Z:\S5\R510\TP2\code\code\p21_6.js
PS C:\Users\duroy231>
```

```
maBDDTP2> db.maCollec4.find()
[
  { _id: ObjectId("68c19c2b6300708704031eb7"), i: 9 },
  { _id: ObjectId("68c19c2b6300708704031eb8"), i: 10 }
]
maBDDTP2>
```

explication :

Dans le fichier JavaScript, on sélectionne les 8 premier élément en classant les i, ensuite on ajoute la limite à 8 enregistrements. Puis sur chaque enregistrement un par un on effectue une deleteOne qui les supprime un par un. Pour finir sur mangosh on effectue une find afin de vérifier que cela à bien fonctionner.

8. Conclusion

L'utilisation de MongoDB se distingue par sa flexibilité et sa puissance à travers différentes familles de requêtes. Les requêtes de projection permettent de contrôler précisément les champs affichés, optimisant la lecture et la clarté des résultats. Les requêtes de mise à jour offrent la possibilité de modifier un ou plusieurs documents grâce à des opérateurs comme $set, $inc ou $unset, ce qui facilite la gestion et l'évolution des données. Avec les requêtes d'agrégation, MongoDB fournit un outil puissant pour analyser et transformer les données : filtrer ($match), regrouper ($group), trier ($sort) ou calculer des indicateurs (moyenne, maximum, somme, etc.), le tout de manière fluide et performante. Enfin, l'utilisation de scripts JavaScript dans l'environnement mongosh permet d'automatiser des traitements, de manipuler les résultats avec des variables, et d'enchaîner des opérations complexes, ce qui rend MongoDB particulièrement adapté aux besoins modernes en termes de big data et d'applications évolutives. Ainsi, MongoDB se révèle être une solution souple, efficace et intuitive pour stocker, interroger et transformer des données de manière dynamique.

On se rend donc compte que mongodb est très pratique pour gérer un grand nombre de données et de faire de lourds traitements de celle-ci grâce à l'utilisation de script javascript. Cependant, les données sont beaucoup moins strictes dans leur forme mais peuvent être beaucoup plus grandes et diverses.