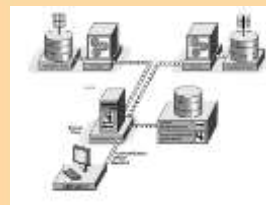


Secteur Tertiaire Informatique  
Filière « Etude et développement »

Séquence « Mettre en place une base de données »

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Apprentissage



Mise en situation



Evaluation

Version	Date	Auteur(s)	Action(s)
1.0	08/10/19	Lécu Régis	Création du document

# TABLE DES MATIERES

Table des matières .....	2
1. Introduction .....	6
2. Découvrir NoSQL .....	7
2.1 Définition du <i>big data</i> : les « trois V » .....	7
2.1.1 Le volume .....	8
2.1.2 La variété .....	8
2.1.3 La vitesse .....	9
2.1.4 Un critère supplémentaire : la véracité .....	9
2.2 Théorème de CAP ( <i>Consistency, Availability, Partition tolerance</i> ) .....	9
2.2.1 Aucun système n'est parfait : sur un exemple .....	9
2.2.2 Aucun système n'est parfait : cas général .....	10
2.3 Classification des SGBD SQL et NoSQL selon le théorème de CAP .....	11
2.3.1 AC : <i>Availability, Consistency</i> .....	11
2.3.2 PC : <i>Partition Tolerance, Consistency</i> .....	11
2.3.3 AP : <i>Availability, Partition Tolerance</i> .....	12
2.4 Les familles de NoSQL .....	12
2.4.1 Couple clé/valeur .....	12
2.4.2 Base documentaire .....	13
2.5 Que choisir pour ses données : SQL ou NoSQL ? .....	13
3. Mettre en pratique NoSQL avec <i>MongoDB</i> .....	14
3.1 Installer le serveur <i>MongoDB</i> .....	14
3.2 Installer le client <i>Robo3T</i> .....	15
3.3 Créer et remplir une collection de test .....	18
3.4 Interroger <i>MongoDB</i> en <i>JavaScript</i> depuis le client <i>Robo3T</i> .....	21
3.4.1 Interrogations simples .....	22
3.4.2 Projection .....	23
3.4.3 Filtre .....	24
3.4.4 Filtre et Projection .....	25
3.4.5 Filtrer avec des expressions régulières .....	26
3.4.6 Filtrer avec des opérateurs .....	27
3.4.7 Le pipeline d'agrégation : <i>aggregate</i> .....	31
3.4.8 Utiliser des variables <i>JavaScript</i> pour simplifier le code des requêtes .....	32
3.4.9 Le pipeline avec group by et fonctions d'agrégation .....	32

Persisten des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

3.4.10	Utiliser d'autres fonctions d'agrégation : moyenne, min, max.....	34
3.4.11	Modifier la base : ajouter un champ, modifier un champ, supprimer un champ sur un objet existant .....	34
3.4.12	Modifications multiples (sur plusieurs objets d'une collection) .....	36
3.4.13	Renommer un attribut.....	36
3.4.14	Manipuler des attributs de type tableau .....	37
3.4.15	Insérer de nouveaux enregistrements dans une collection .....	37
3.4.16	Supprimer des enregistrements, selon un critère.....	38
3.4.17	Utiliser JavaScript pour des opérations plus complexes. ....	38
3.4.18	Faire une "jointure" entre plusieurs collections .....	39
3.5	Appeler MongoDB depuis un client JAVA .....	40
3.5.1	Création du projet Maven.....	40
3.5.2	Les premiers pas : valider l'environnement.....	41
3.5.3	Une première requête simple.....	41
3.5.4	Une requête paramétrée.....	42
3.5.5	Insérer de nouveaux objets et les afficher.....	43
3.5.6	Supprimer des objets dans une collection.....	44

## Objectifs

A l'issue de cette séance, le stagiaire sera capable de :

- Comparer SQL et NoSQL et faire le meilleur choix pour ses données
- Comprendre les différences entre les familles de NoSQL
- Découvrir NoSQL par la pratique avec *MongoDB*

## Pré requis

Cette séance intervient après l'apprentissage des bases de données relationnelles et du langage SQL, qui demeurent des repères pour l'apprentissage.

## Méthodologie

Ce document peut être utilisé en présentiel ou à distance.

Il précise la situation professionnelle visée par la séance, la situe dans la formation, et guide le stagiaire dans son apprentissage et ses recherches complémentaires.

## Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

## Ressources

- Documentation en ligne de MongoDB : <https://docs.mongodb.com/manual/>
- Présentation en français : <https://www.mongodb.com/fr>
- Introduction au NoSQL : <https://fr.wikipedia.org/wiki/NoSQL>
- Livre NoSQL et le Big Data :  
<https://www.eyrolles.com/Chapitres/9782212141559/9782212141559.pdf>

# 1. INTRODUCTION

**NoSQL** (« *Not Only SQL* ») désigne une nouvelle approche de la persistance de données, qui ne s'appuie plus sur le modèle relationnel.

- Les bases de données NoSQL, telles que *MongoDB*, ne sont donc pas des SGBDR. Elles peuvent si nécessaire être interrogées et mises à jour en SQL, mais ce n'est pas leur langage natif.
- L'approche NoSQL répond au problème des *Big Data* : par exemple, des moteurs de recherche comme *Google*, ou des réseaux sociaux comme *Facebook* ou *Twitter* doivent stocker et manipuler d'énormes quantités de données très diversifiées (structurées, non structurées ou semi-structurées), qui doivent être distribuées sur de nombreux serveurs.

Or les SGBDR et le langage SQL n'ont pas été prévus à l'origine pour :

- Manipuler des données hétérogènes et évolutives : il faut prévoir à l'avance les entités et leurs attributs, les tables et leurs colonnes, les clés primaires et étrangères etc.
  - Distribuer de gros volumes de données sur différents serveurs synchronisés.
- Nous allons préciser les insuffisances du langage SQL face aux *Big Data* et présenter les différentes réponses de l'approche NoSQL. Il s'agit d'une approche et pas d'une norme commune :
    - Il n'est pas si simple de remplacer SQL : chaque SGBD NoSQL répond bien à un ou plusieurs problèmes posés par les *Big Data* mais pas à tous
    - Il faudra donc savoir choisir le bon SGBD NoSQL en fonction du type de données à manipuler, ou au moins comprendre le choix effectué par un expert.
  - Nous allons présenter quelques-unes des familles NoSQL et leurs avantages respectifs, sans chercher à devenir nous-même expert dans ce domaine.
  - Nous mettrons en pratique cette nouvelle approche, avec la base *MongoDB* :
    - avec un requêteur JSON pour l'apprentissage
    - puis en réalisant un client Java.

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

## 2. DECOUVRIR NoSQL

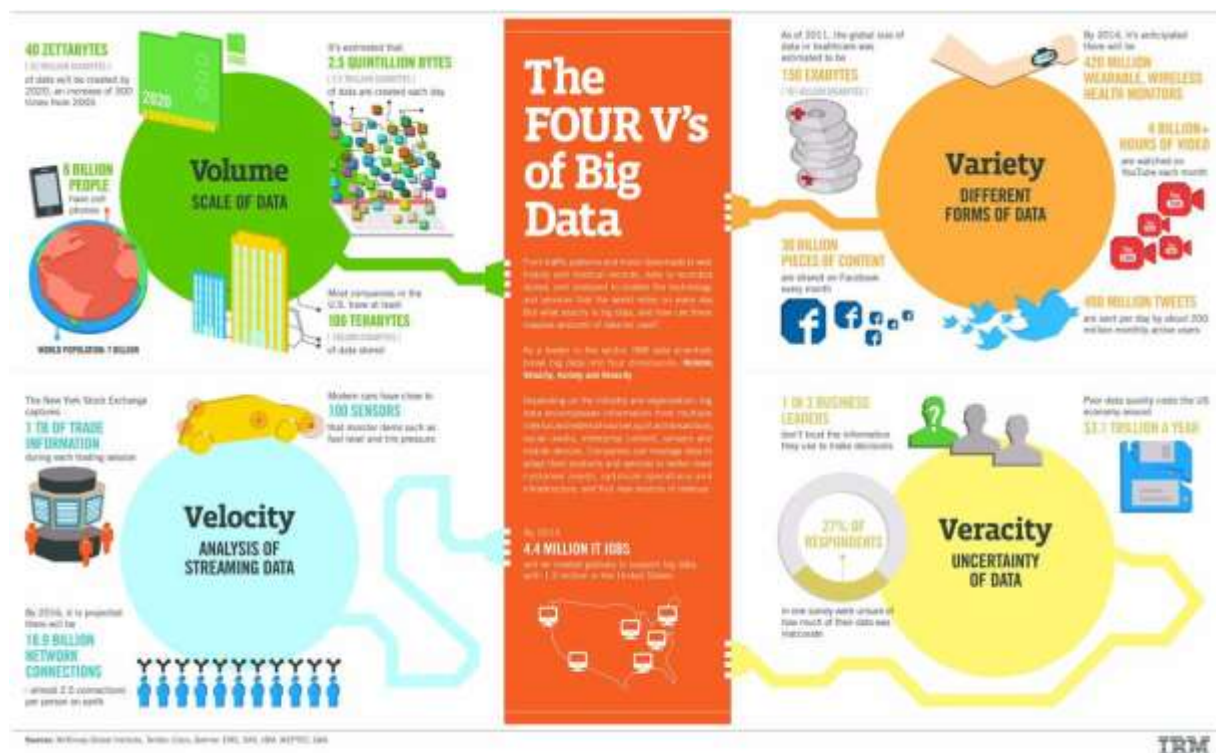
NoSQL est une approche qui répond aux problèmes techniques du *Big Data*.

### 2.1 DEFINITION DU BIG DATA : LES « TROIS V »

Le *Big Data* est apparu avec le développement des techniques numériques : réseaux sociaux, appareils mobiles, objets connectés, multiplication des recherches et des transactions sur Internet etc.

Il en a résulté une profonde évolution de la nature des données à traiter, que l'on peut résumer par le critère des « trois V » : **Volume**, **Vitesse**, **Variété**.

IBM ajoute un quatrième critère : la **Véracité** des données.



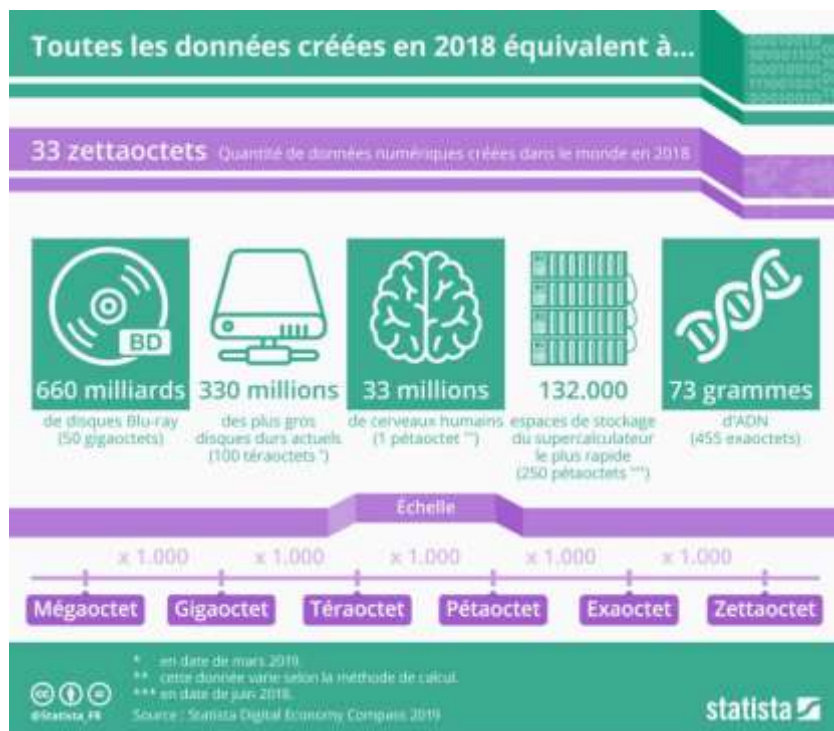
Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »



### 2.1.1 Le volume

C'est la principale caractéristique du *Big Data*.



33 Zettaoctet ( $10^{21}$ ) de données numériques ont été créées dans le monde en 2018, soit pour donner une idée concrète, 33 milliards de disque de 1 Tera.

Avec une prévision de 40 Zettaoctet en 2020.

Ce volume a été multiplié par 300 depuis 2005.

Les données proviennent à la fois des entreprises et des particuliers.

Pour revenir à la technique, retenons un point précis : ces énormes volumes de données posent des problèmes de stockage et de temps de réponse dans les traitements.

Ceci exige de nombreux serveurs avec des traitements distribués, qui n'étaient pas prévus au départ par le modèle relationnel et sont difficiles à gérer.

### 2.1.2 La variété

Au-delà du volume, c'est aussi la diversité des données qui a augmenté : provenance, format, domaine.

Cette diversité exige des traitements différenciés selon les domaines, avec des niveaux de fiabilité et de sécurité variés. Deux cas extrêmes :

- Les données du domaine médical, sur des appareils connectés de « monitoring » : grande fiabilité et disponibilité, exigence de sécurité
- Les données publiques sur Facebook, les Tweets : peu d'exigence de sécurité, mais grande variété dans les formats à traiter.

Point technique : cette grande diversité des données du *Big Data* gêne l'approche relationnelle, qui exige de prévoir à l'avance les entités et leurs attributs.

Dans un même domaine (par exemple le marketing), les données évoluent sans cesse, en introduisant de nouveaux indicateurs de tendance etc.

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

### 2.1.3 La vitesse

Les données doivent être collectées plus rapidement, souvent en temps réel, en provenance d'objets connectés par Internet (*IoT*).

Selon le premier schéma (IBM), une voiture récente possède environ 100 capteurs connectés. Chaque session de la bourse de New York réalise en moyenne 1 To de transactions, qui doivent bien sûr être traitées en temps réel.

### 2.1.4 Un critère supplémentaire : la véracité

Ces trois critères (volume, variété, vitesse) suffisent à définir le *Big Data*.

IBM leur ajoute la « véracité », qui est un critère classique pour les bases de données et constitue un enjeu majeur pour le *Big Data*.

Toujours selon le schéma d'IBM, 27% des entreprises reconnaissent qu'elles ne sont pas certaines de l'exactitude des données qu'elles ont collectées.

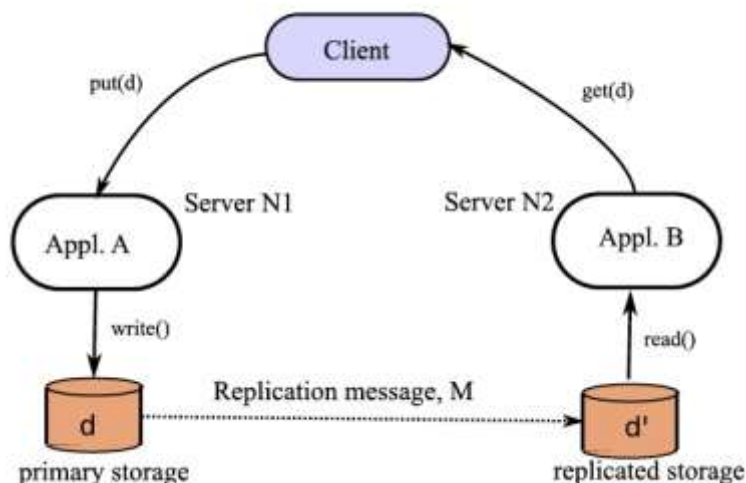
## 2.2 THEOREME DE CAP (CONSISTENCY, AVAILABILITY, PARTITION TOLERANCE)

Pour faire un choix, il faut connaître les limites des systèmes SQL et NoSQL, qui sont décrites selon trois critères par le théorème de CAP.

### 2.2.1 Aucun système n'est parfait : sur un exemple

Commençons par un cas particulier, avec deux serveurs N1 et N2 :

- Le serveur N1 est « maître », il est le seul à pouvoir écrire sur le disque, suite aux demandes d'écriture du client
- Le serveur N2 est « esclave », il ne répond qu'aux demandes de lecture du client.



Dans cet exemple d'architecture distribuée, le serveur N1 duplique ses données sur le serveur N2 qui lui sert de miroir.

Pendant la duplication du message **M**, les deux serveurs ne sont pas synchronisés :

- Le disque **d** est différent de sa copie **d'**
- Cette architecture ne respecte pas la propriété de « Cohérence » (*Consistency*).

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Si un serveur tombe en panne, l'ensemble ne fonctionne plus, car on perd soit la lecture, soit l'écriture : l'architecture ne respecte donc pas les propriétés de « Disponibilité » (*Availability*) et de « Tolérance au partitionnement » (*Partition Tolerance*).

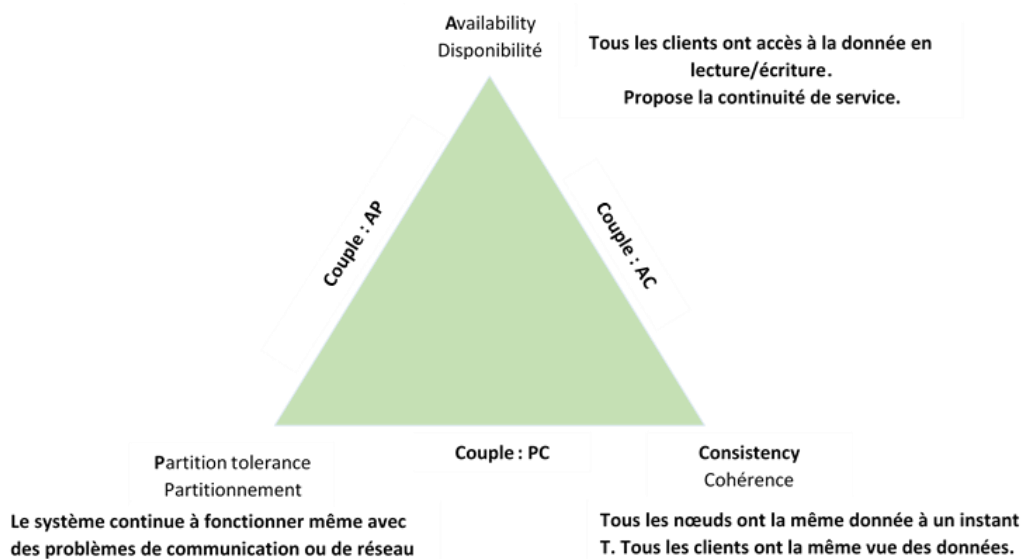
Ces deux propriétés seraient garanties par une duplication bidirectionnelle, avec un accès en lecture/écriture sur les deux serveurs, mais toujours pas la cohérence.

### 2.2.2 Aucun système n'est parfait : cas général

En prenant le cas général (théorème de CAP), un système distribué ne peut garantir à la fois que deux de ces 3 propriétés :

- **Consistency** (cohérence) : tous les nœuds voient exactement les mêmes données au même moment
- **Availability** (disponibilité) : garantie que toutes les requêtes reçoivent une réponse. La perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement
- **Partition tolérance** (partitionnement) : aucune panne autre que la coupure totale du réseau ne doit empêcher le système de répondre correctement ; en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome.

Tout système distribué ne peut garantir qu'un couple d'exigences parmi trois : AP, AC ou PC

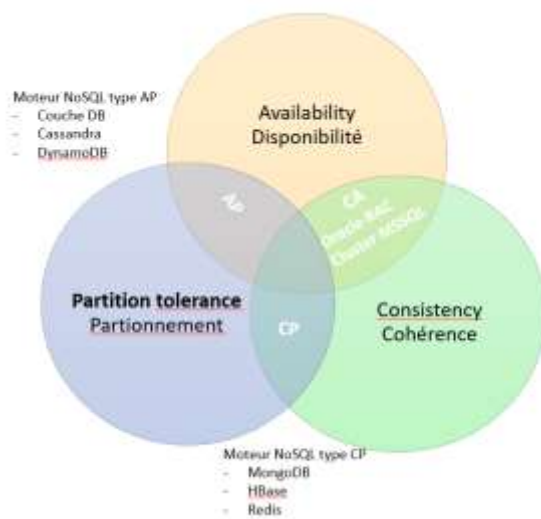


Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

## 2.3 CLASSIFICATION DES SGBD SQL ET NoSQL SELON LE THEOREME DE CAP

Les SGBDR sont en général conformes à AC, les moteurs NoSQL à AP ou PC



### 2.3.1 AC : Availability, Consistency

Cluster de site unique, donc tous les nœuds sont toujours en contact :

- Le cluster est redondant, donc si un nœud tombe en panne, le cluster continue à fonctionner en lecture/écriture (*Availability*).
- Les nœuds sont synchronisés dans le cluster et renvoient tous la même donnée (*Consistency*).
- Mais lorsqu'une partition du système se produit, suite à un problème réseau, le système s'arrête.

Les SGBDR fournissent des versions en cluster (« grappe ») de serveurs, qui peuvent aussi traiter des *Big Data* (d'après leurs éditeurs !).

Ces clusters privilégient la disponibilité et la cohérence, au détriment de la résistance à la partition du système.

Deux références :

**Oracle RAC** (*Real Application Clusters*, version cluster du SGBDR Oracle) :

<https://www.oracle.com/fr/assets/real-application-clusters-11g-070323-fr.pdf>

**SQL Server 2019** en cluster :

<https://docs.microsoft.com/fr-fr/sql/sql-server/what-s-new-in-sql-server-ver15?view=sqlallproducts-allversions>

### 2.3.2 PC : Partition Tolerance, Consistency

Certaines données peuvent ne pas être accessibles, mais les données restent cohérentes, et le système supporte le partitionnement (coupure réseau...).

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

**MongoDB** : <https://www.mongodb.com/>

C'est le SGBD NoSQL que nous allons utiliser dans ce document.

**hbase** : <https://www.lebigdata.fr/apache-hbase-tout-savoir>

**redis** : <https://www.w3resource.com/slides/redis-nosql-database-an-introduction.php>

### 2.3.3 AP : Availability, Partition Tolerance

Le système est toujours disponible même en cas de partitionnement, mais certaines données retournées peuvent être inexactes.

**cassandra** : <http://cassandra.apache.org/>

**dynamodb** : <https://aws.amazon.com/fr/dynamodb/>

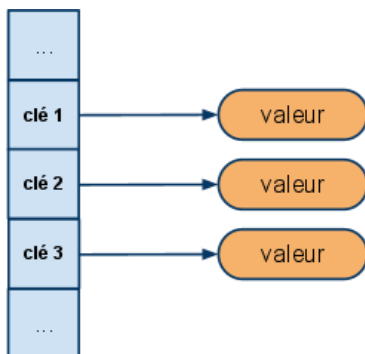
**couchdb** : <http://couchdb.apache.org/>

## 2.4 LES FAMILLES DE NoSQL

Les familles NoSQL ne remplissent pas toutes les mêmes critères, mais elles diffèrent aussi par leur structure et mode d'utilisation : stockage de couple (clé, valeur), base documentaire, base orientée colonnes, base de graphes.

Nous allons décrire les deux premières familles, qui sont répandues et simples à mettre en œuvre.

### 2.4.1 Couple clé/valeur

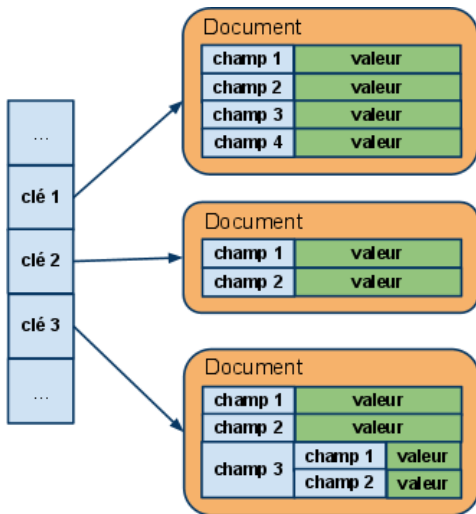


- C'est la catégorie de base : chaque objet est identifié par une clé unique qui constitue la seule manière de le requêter.
- On dispose des quatre opérations CRUD (*Create, Read, Update, Delete*) qui sont toutes basées sur la clé de l'objet.
- La base utilise une table de hachage distribuée qui permet de retrouver les objets par leur clé.
- La structure de l'objet est libre et le plus souvent laissée à la charge du développeur (formats XML, JSON...).

Persiste des données non structurées (semi-structurées) dans un SGBD NoSQL

- Exemples : **dynamodb, redis**

### 2.4.2 Base documentaire



- Les bases de données documentaires sont constituées de collections de document.
- Un document est composé de champs et de leurs valeurs associées : soit des types simples (entier, chaîne de caractère, date...), soit des types composés de plusieurs couples clé/valeur.
- La structure des documents ne doit pas être définie à l'avance comme dans une base relationnelle. Elle peut évoluer en cours d'utilisation, ce qui la rend très pratique pour des données complexes et hétérogènes.
- Par rapport au modèle du couple (clé, valeur), le modèle documentaire a l'avantage de permettre des requêtes complexes sur toutes les valeurs (et pas seulement sur la clé de l'objet).
- Le modèle documentaire reste simple, et possède des meilleures performances que le modèle relationnel.
- Exemples : **mongodb, couchdb**

### 2.5 QUE CHOISIR POUR SES DONNEES : SQL OU NoSQL ?

- Il faut choisir un SGBDR avec SQL quand :
  - Les données sont connues à l'avance, liées entre elles, pas trop diversifiées et stables
  - L'intégrité des données est essentielle (gérée par le SGBDR)
  - Les transactions sont essentielles (sites de vente, de réservation...)
- A l'inverse, NoSQL peut être une bonne solution quand :
  - Les données sont indépendantes, difficiles à déterminer à l'avance et évolutives
  - La principale caractéristique du projet est l'agilité : il faudra intégrer de nombreux types de données, structurées ou non.
  - Il y a une forte exigence de performance : volume des données, rapidité d'accès, nécessité de distribuer la base sur différents serveurs.

Persisten des données non structurées (semi-structurées) dans un SGBD NoSQL

### 3. METTRE EN PRATIQUE NoSQL AVEC *MONGODB*

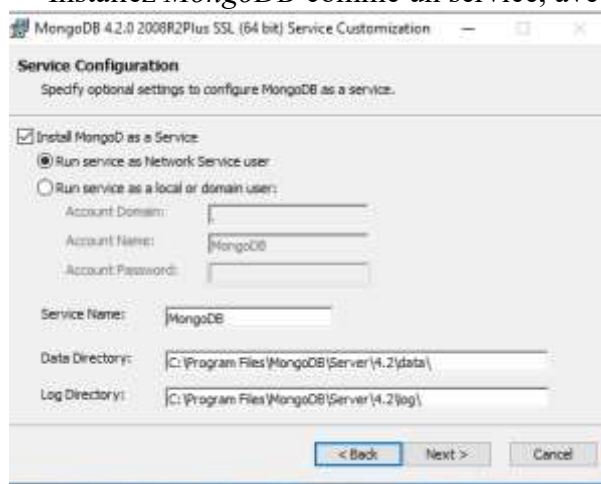
#### 3.1 INSTALLER LE SERVEUR *MONGODB*

- Téléchargez la version gratuite de *MongoDB* sur :  
<https://www.mongodb.com/download-center/community>

- Lancez le fichier d'installation :



- Choisissez l'installation complète
- Installez *MongoDB* comme un service, avec la configuration standard :



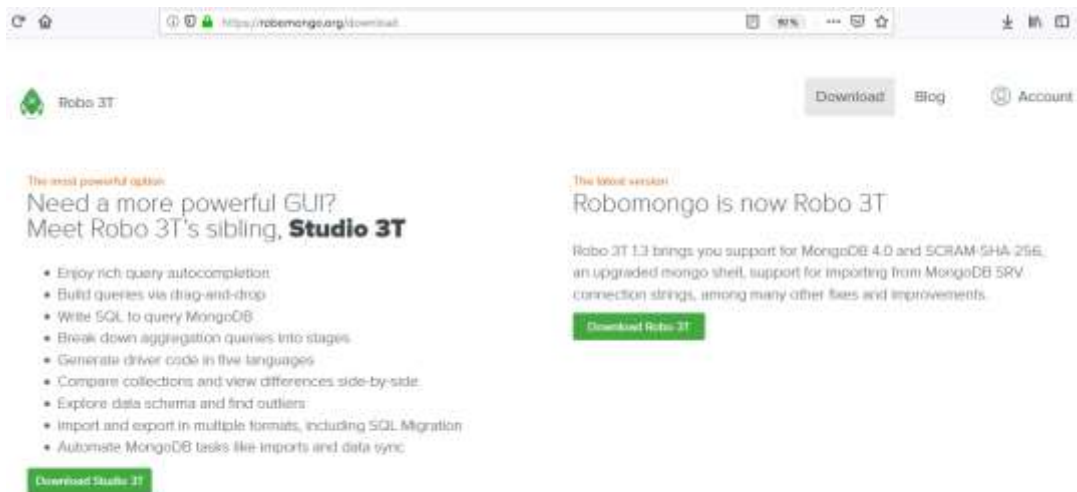
- Installez *MongoDB Compass* qui est l'interface graphique standard de *MongoDB*
- A la fin de l'installation, *MongoDB Compass* vous propose de créer une connexion par défaut non sécurisée vers l'hôte.
- Confirmez en cliquant sur *Connect*.

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

### 3.2 INSTALLER LE CLIENT **ROBO3T**

- Nous allons effectuer nos requêtes vers MongoDB avec le client ROBO3T.
- Téléchargez la distribution de ROBO3T, sur le site :

<https://robomongo.org/download>

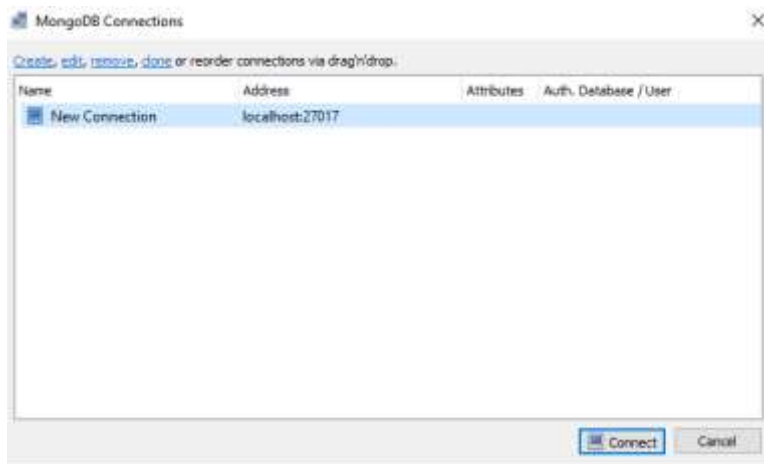


- Choisissez « *Download Robo 3T* » pour obtenir la version gratuite, téléchargez l'installation pour Windows, et lancez l'exécutable de l'installateur
- A la fin de l'installation, Robo3T vous propose de vous connecter sur le serveur MongoDB.
- Cliquez sur *Connect*

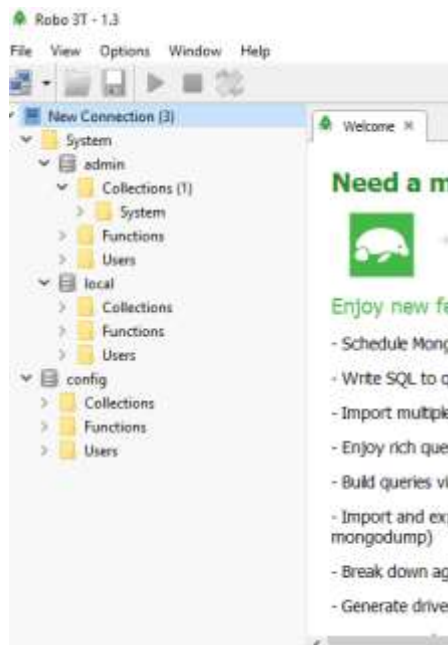
Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

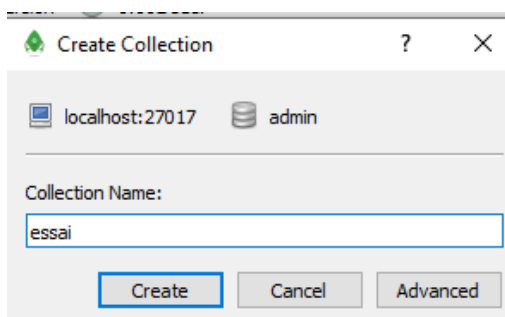




- Robo3T vous affiche une vue sur les trois bases de données par défaut de MongoDB : **admin, local et config** avec leurs collections, fonctions et utilisateurs.



- Pour valider l'outil nous allons créer une collection « *essai* », dans la base *admin*, avec un premier document
- Cliquez avec le bouton droit sur *Collection* -> *Create Collection*

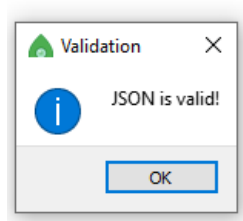


Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

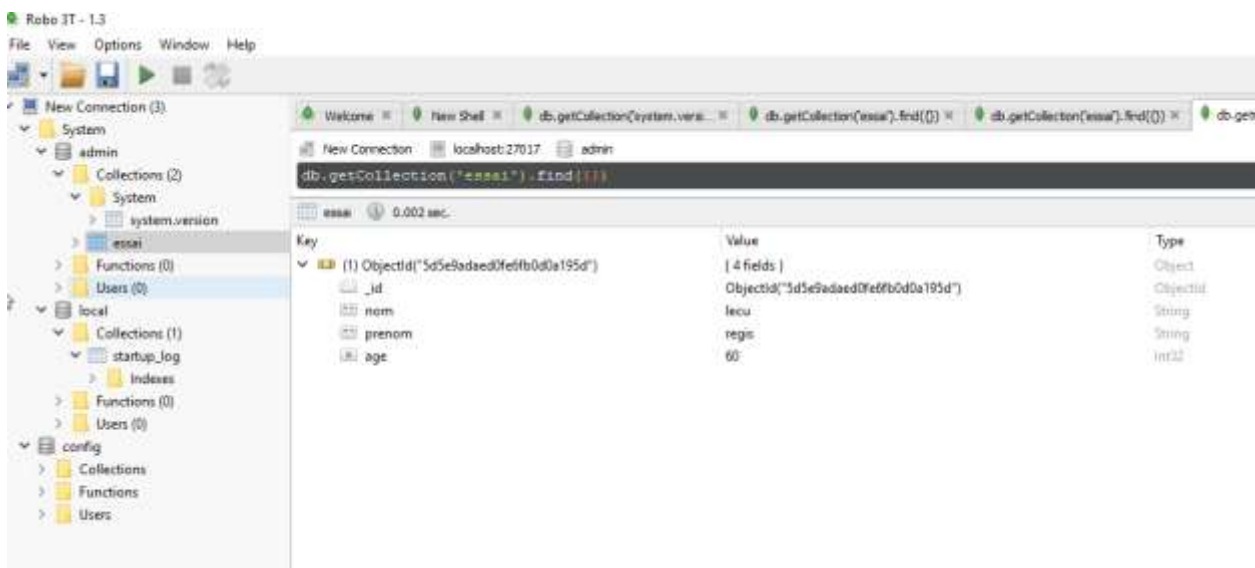
- Cliquez avec le bouton droit sur la collection « *essai* » -> *Insert Document*
- L'insertion, comme les interrogations, de *MongoDB* va se faire en JSON
- Par exemple, je crée un document avec mon nom, prénom et âge



- Validez la syntaxe de l'objet JSON à insérer, en cliquant sur *Validate*



- Puis exécutez la requête par *Save*
- Double-cliquez sur la collection « *essai* » pour visualiser ses documents :



- Le document est un objet avec une clé automatique *\_id* du type *ObjectId*, qui contient trois autres champs : deux String *nom* et *prenom* et un Int32 *age*
- Votre serveur *MongoDB* est opérationnel

Persistez des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

### 3.3 CREER ET REMPLIR UNE COLLECTION DE TEST

- Pour notre apprentissage de NoSQL, nous allons manipuler des données publiques, qui sont maintenant fournies par les administrations, sur les communes, le cadastre, les bibliothèques, musées etc. :

<https://www.data.gouv.fr/fr/>

- Nous allons nous intéresser aux bibliothèques publiques :

<https://www.data.gouv.fr/fr/datasets/adresses-des-bibliotheques-publiques/>

- Le site permet d'exporter les données aux formats CSV, JSON :



- On peut prévisualiser les données :

Prévisualisation

	Librairie	Librairie	Nom	Type	code_in	code_p	code	Localisation	CP	CEDEX	Ville	Profil département
1	Bibliothèque Municipale	Bibliothèque municipale	Bibliothèque municipale	Bibliothèque ouverte au public	4.007	4.007	31 rue Desbrières		91290		Acquies	91000000
2	Bibliothèque Municipale	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.207	5.207	1 rue du Parc	Station d'Acquies	91010		Acquies	91000000
3	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.208	5.208	4 rue du Parc		91010		Acquies	91000000
4	BIBLIOTHEQUE COMMUNALE	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.209	5.209	1 place du Centre du Village		91010		Acquies	91000000
5	BIBLIOTHEQUE COMMUNALE	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.210	5.210	1 rue du Village		91010		Acquies	91000000
6	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.211	5.211	1 rue du Village		91010		Acquies	91000000
7	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.212	5.212	1 rue du Village		91010		Acquies	91000000
8	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.213	5.213	1 rue du Village		91010		Acquies	91000000
9	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.214	5.214	1 rue du Village		91010		Acquies	91000000
10	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.215	5.215	1 rue du Village		91010		Acquies	91000000
11	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.216	5.216	1 rue du Village		91010		Acquies	91000000
12	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.217	5.217	1 rue du Village		91010		Acquies	91000000
13	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.218	5.218	1 rue du Village		91010		Acquies	91000000
14	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.219	5.219	1 rue du Village		91010		Acquies	91000000
15	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.220	5.220	1 rue du Village		91010		Acquies	91000000
16	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.221	5.221	1 rue du Village		91010		Acquies	91000000
17	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.222	5.222	1 rue du Village		91010		Acquies	91000000
18	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.223	5.223	1 rue du Village		91010		Acquies	91000000
19	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.224	5.224	1 rue du Village		91010		Acquies	91000000
20	Bibliothèque Municipale Georges Des	Bibliothèque municipale	Bibliothèque ouverte au public	Bibliothèque ouverte au public	5.225	5.225	1 rue du Village		91010		Acquies	91000000

- Pour une importation dans une base NoSQL, le plus pratique sera le format JSON
- Vous trouverez dans *Livrables*, le fichier *Biblio.json* qui provient de ces données publiques. Il a été adapté pour servir de support à nos exercices.
- Chaque objet JSON commence par { et comprend :
  - un attribut **\_id** qui est la clé de l'objet, du type *ObjectID*

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL  
Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

- un attribut **fiche**, de type objet, qui décrit la bibliothèque : **catégorie**, **nom**, **nom\_dept** etc.

Ces données sont issues des données publiques. Certains noms d'attributs ont été modifiés pour clarifier. L'attribut **ouverture** a été ajouté.

- un attribut **notation**, de type objet, qui a été ajouté pour les besoins de la série d'exercices. Il comprend une appréciation globale sur la bibliothèque (**appreciation**) donnée par l'administration, et un tableau **notes** : série de notes entre 1 et 5 donnée par le public.

```
{
  "_id" : ObjectId("5d4054ba3b447b3ecda889cc"),
  "fiche" : {
    "categorie" : "Bibliothèque municipale",
    "région" : "Ile-de-France",
    "code_bib" : 4067.0,
    "insee" : "91021",
    "nom_dept" : "ESSONNE",
    "code_ua" : 4067.0,
    "voie" : "31 rue Dauvilliers",
    "coordonnees_finales" : [ 48.589699, 2.252239],
    "coordonnees_ban" : "48.589699, 2.252239",
    "adresse_ville" : "Arpajon",
    "dept" : "91",
    "regi" : "11",
    "nom" : "Bibliothèque Municipale",
    "cp" : "91290",
    "coordonnees_insee" : "48.5908921647, 2.24349117607",
    "ouverture" : "Bibliothèque ouverte hors congés d'été"
  },
  "notation" : {
    "appreciation" : "ok",
    "notes" : [
      5.0,
      4.0,
      5.0
    ]
  },
  "record_timestamp" : "2018-06-08T15:12:13+02:00"
}
// etc. enregistrement suivant
```

- Les données ne sont pas homogènes : beaucoup de bibliothèques n'ont pas encore de notation. Certaines ont des données supplémentaires : l'attribut **local** qui précise le lieu.

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

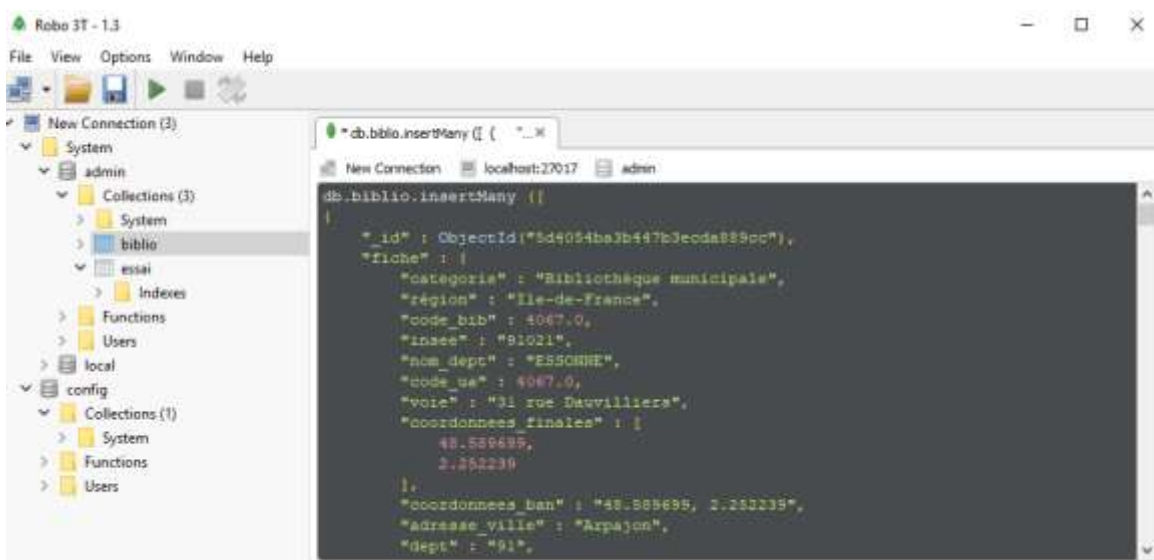
Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

- Créez comme dans l'essai précédent une collection « *biblio* »
- Puis double-cliquez sur la collection et remplacez la requête *getCollection* qui liste la collection, par la requête d'insertion contenue dans le fichier *Biblio.json*, en sélectionnant tout le fichier

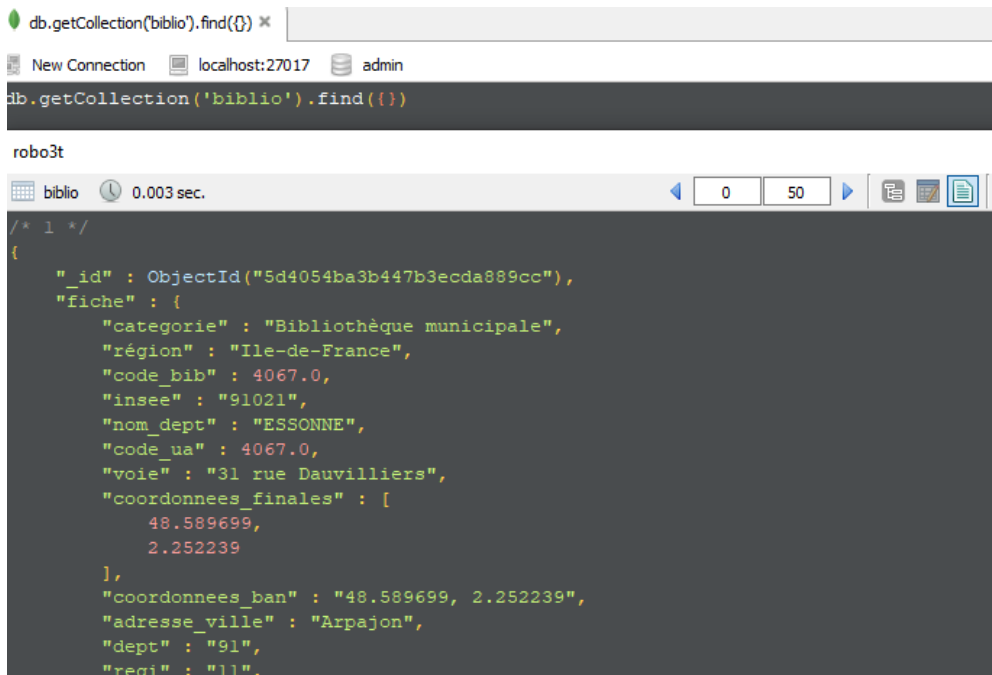
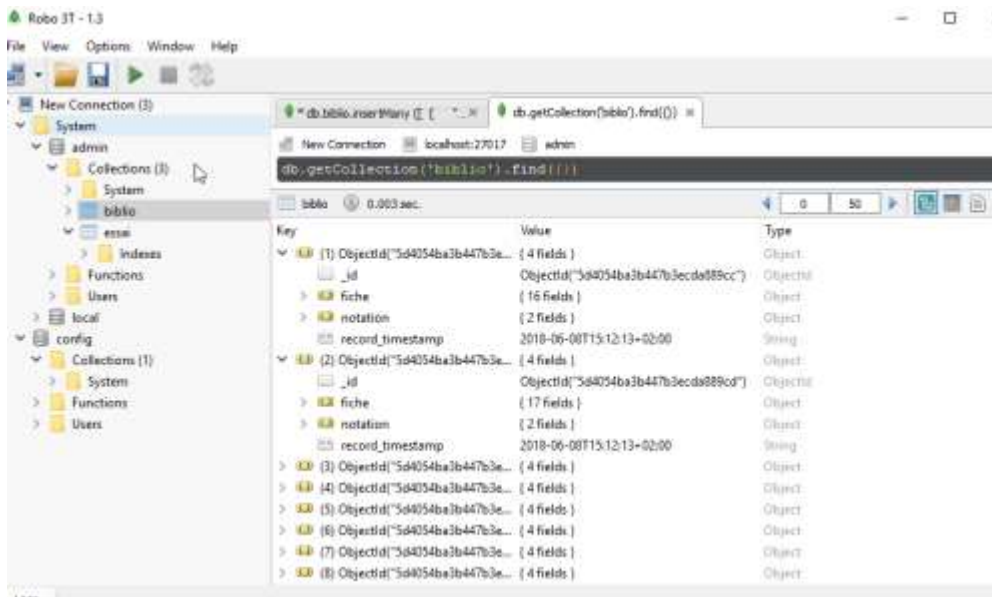
```
db.biblio.insertMany ( [
{
  "_id" : ObjectId("5d4054ba3b447b3ecda889cc"),
  "fiche" : {
    "categorie" : "Bibliothèque municipale",
    "région" : "Ile-de-France",
    "code_bib" : 4067.0,
    "inssee" : 91021,
    "nom_dept" : "ESSONNE",
    "code_us" : 4067.0,
    "voie" : "31 rue Dauvilliers",
    "coordonnees_finales" : [
      48.589695,
      2.282239
    ],
    "coordonnees_ban" : "48.589695, 2.282239",
    "adresse_ville" : "Arpajon",
    "dept" : "91",
  }
},

```

- Toutes les requêtes vers MongoDB utilisent des paramètres en JSON : ici on passe une collection d'objets (commençant par [ ) à la méthode *insertMany* sur la collection *biblio*.



- Lancez la requête en cliquant sur la flèche verte
- Vérifiez que votre collection a été remplie, en cliquant à nouveau sur *biblio*
- Essayez les différents modes d'affichage qui seront pratiques pour parcourir les données : en arbre montrant les objets imbriqués, en table et en JSON.



### 3.4 INTERROGER MONGODB EN JAVASCRIPT DEPUIS LE CLIENT ROBO3T

Vous trouverez les corrections de tous les exercices proposés dans le dossier **Livrables**, fichier **corriges.json**

- Nous allons utiliser cette base de test pour découvrir les possibilités de NoSQL, comme nous l'avons fait en SQL :
    - d'abord des interrogations simples
    - puis des requêtes avec regroupement (équivalent du GROUP BY en SQL)
    - des insertions, modifications, suppressions
    - des requêtes portant sur plusieurs collections (comme les jointures en SQL).
- Persister des données non structurées (semi-structurées) dans un SGBD NoSQL
- Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

- La connaissance du langage SQL n'est en théorie pas nécessaire, mais cela aidera souvent comme point de comparaison !
- Toutes les requêtes vont manipuler des objets en JSON : pour les paramètres et pour les valeurs retournées.
- Les requêtes dans le client interactif *Robo3T* sont écrites en *JavaScript* : nous pourrons utiliser du code *JavaScript* pour enchaîner nos requêtes et organiser nos traitements.
- Pour chaque exercice, nous renverrons à la documentation en ligne de *MongoDB* qui contient de nombreux exemples :

<https://docs.mongodb.com/manual/reference/>

- Pour chaque point :
  - Exécutez les codes d'exemple (en bleu)
  - Consultez la documentation, en suivant les liens
  - Réalisez et testez les exercices proposés
  - Comparez vos codes à ceux du fichier *corriges.json* (dans *Livrables*)

### 3.4.1 Interrogations simples

*Lister toutes les bibliothèques avec toutes leurs caractéristiques (c'est-à-dire tous les objets de la collection biblio)*

**db.biblio.find()**

La méthode *find* sans paramètre retourne toute la collection.

Pour mémoire, en SQL : `select * from biblio`

Dans les exemples suivants, nous allons le plus souvent abandonner SQL et « penser en JSON ».

<https://docs.mongodb.com/manual/reference/method/db.collection.find/>

*Récupérer le nombre de bibliothèques :*

**db.biblio.find().count()**

Le *find* retourne une collection sur laquelle on peut faire un *count*.

*Lister les noms de toutes les bibliothèques, sans doublon*

**db.biblio.distinct("fiche.nom")**

Remarquez la notation objet : le *nom* est un attribut de l'objet *fiche*.

Comme son nom l'indique, le *distinct* n'affiche que des objets distincts, en les listant à « plat » dans un tableau JSON, sans les structures qui les contiennent :

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

```
New Connection  localhost:27017  admin
db.biblio.distinct("fiche.nom")

0.005 sec.

/* 1 */
[
  "Bibliothèque Municipale",
  "Bibliothèque Municipale Georges Brassens",
  "BOUSSY-Bibliothèque Communautaire",
  "BRUNOY-Bibliothèque Communautaire",
  "Bibliothèque Jean-Jacques SEMPE",
  "Mediatheque Municipale De Cheptainville",
  "Mediatheque Municipale",
  "Bibliothèque",
  etc.
]
```

etc.

*Lister toutes les catégories, sans doublon*

```
db.biblio.distinct("fiche.categorie")
```

La méthode *distinct* est pratique pour explorer une base NoSQL et visualiser des informations utiles, afin de bien construire ses requêtes.

```
* db.biblio.distinct("fiche.categ...
New Connection  localhost:27017  admin
db.biblio.distinct("fiche.categorie")

0.002 sec.

/* 1 */
[
  "Bibliothèque municipale",
  "Bibliothèque départementale"
]
```

### 3.4.2 Projection

*Lister toutes les bibliothèques, en affichant uniquement les champs : categorie, nom, région (sans l'id de l'objet) :*

```
db.biblio.find( {},
  {"fiche.categorie":1, "fiche.nom":1, "fiche.région":1, "_id":0 } )
```

La méthode *find* a deux paramètres, de type objet.

Le deuxième paramètre liste les attributs que l'on veut afficher : c'est une « projection », l'équivalent des colonnes derrière un SELECT en SQL.

Pour afficher un attribut, on le déclare à 1 (catégorie, nom et région).

Persistez des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »



Pour exclure un attribut, on le déclare à 0 (\_id)

```
db.biblio.find ( {
  "fiche.dept": 0,
  "fiche.ouverture": 1,
  "fiche.nom": 1,
  "fiche.adresse": 1,
  "fiche.note": 1,
  "fiche.appréciation": 1,
  "fiche.type": 1
})
```

etc.



Exercice 1 : Lister toutes les bibliothèques, en affichant uniquement leurs nom, ville et notation (appréciation et notes), sans l'id de l'objet

```
db.biblio.find ( {
  "fiche.dept": "91",
  "fiche.ouverture": "Bibliothèque ouverte au public"
})
```

etc.

### 3.4.3 Filtre

Lister uniquement les bibliothèques du 91, qui sont ouvertes au public

```
db.biblio.find ( {"fiche.dept": "91", "fiche.ouverture": "Bibliothèque ouverte au public"}
```

La méthode *find* a deux paramètres, de type objet (ici, le deuxième est omis).

Le premier paramètre liste les critères de sélection : c'est un « filtre », l'équivalent d'une clause WHERE en SQL.



Exercice 2 : Lister uniquement les bibliothèques municipales, qui ont reçu une appréciation

Pour tester l'existence d'un attribut, utilisez l'opérateur *\$exists* avec une valeur 1 (ou 0 pour l'inexistence) : *nom\_attribut : {\$exists : 1}*

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

Key	Value
<ul style="list-style-type: none"> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> <li>[2] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> <li>[3] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> <li>[4] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> <li>[5] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> <li>[6] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> <li>[7] ObjectID("544254ba3a447b3ecda889ca") <ul style="list-style-type: none"> <li>_id</li> <li>fiche</li> <li>notation <ul style="list-style-type: none"> <li>appreciation</li> <li>notes</li> <li>record_timestamp</li> </ul> </li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>( 2 fields )</li> <li>ok</li> <li>( 3 elements )</li> <li>2018-06-08T15:12:13+02:00</li> <li>( 4 fields )</li> <li>ObjectID("544254ba3a447b3ecda889ca")</li> <li>( 17 fields )</li> <li>( 2 fields )</li> <li>ok</li> <li>( 3 elements )</li> <li>2018-06-08T15:12:13+02:00</li> <li>( 4 fields )</li> <li>ObjectID("544254ba3a447b3ecda889ca")</li> <li>( 16 fields )</li> <li>( 2 fields )</li> <li>ok</li> <li>( 3 elements )</li> <li>2018-06-08T15:12:13+02:00</li> <li>( 4 fields )</li> <li>( 4 fields )</li> <li>( 4 fields )</li> <li>( 4 fields )</li> </ul>

etc. (49 bibliothèques en tout).

### 3.4.4 Filtre et Projection

Filtrer les bibliothèques du 91 qui ne sont pas ouvertes au public, en affichant leur nom, ville, et le motif de non ouverture (sans l'id de l'objet) :

```
db.biblio.find( {"fiche.dept":"91", "fiche.ouverture":{"$ne:"Bibliothèque ouverte au public"} },
               { "fiche.nom":1, "fiche.adresse_ville":1, "fiche.ouverture":1, "_id":0 } )
```

Notez l'utilisation de l'opérateur *not equal* : `$ne`

Le premier paramètre du *find* est un objet JSON, qui contient une liste de critères (sous la forme clé : valeur), séparés par des virgules.

Chaque valeur est soit une constante ("**91**"), soit elle-même un objet composé d'un opérateur et d'une valeur : `{"$ne:"Bibliothèque ouverte au public"}`

Les opérateurs de MongoDB (« *Query Selectors* ») sont précédés d'un \$, qui permet à l'interpréteur de les distinguer des constantes :

<https://docs.mongodb.com/manual/reference/operator/query/#query-selectors>



Autre syntaxe avec l'opérateur `$regex` :

```
db.biblio.find({"fiche.nom":{"$regex :/Médiathèque/i}})
```

<https://docs.mongodb.com/manual/reference/operator/query/regex/#behavior>



Exercice 4 : Lister les bibliothèques du 95 dont la ville commence par « Arr », en affichant uniquement le département et la ville

```
{
  "fiche" : {
    "adresse_ville" : "Arronville",
    "dept" : "95"
  }
}
```

(1 seule bibliothèque : « Arronville »)



Exercice 5 : Lister les bibliothèques dont la ville commence par "B", contient au moins un "o", et se termine par "le", en affichant uniquement la ville

Dans une expression régulière, `.*` désigne une suite de caractères quelconques

```
{
  "fiche" : {
    "adresse_ville" : "Bondoufle"
  }
}
```

(1 seule bibliothèque : « Bondoufle »)

### 3.4.6 Filtrer avec des opérateurs

Lister les bibliothèques qui ont reçu au moins une note inférieure ou égale à 1, avec leur nom, leur ville, leur appréciation et leurs notes

```
db.biblio.find( {
  "notation.notes":
  {
    $lte:1
  },
  {"fiche.adresse_ville":1, "fiche.nom":1, notation:1 }
})
```

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

(1) ObjectId("544254ba3b447b3ecda889cf")	{ 3 fields }
_id	ObjectId("544254ba3b447b3ecda889cf")
fiche	{ 2 fields }
notation	{ 2 fields }
appreciation	moyen
notes	[ 4 elements ]
[0]	1.0
[1]	2.0
[2]	3.0
[3]	5.0
(2) ObjectId("544254ba3b447b3ecda889d3")	{ 3 fields }
_id	ObjectId("544254ba3b447b3ecda889d3")
fiche	{ 2 fields }
notation	{ 2 fields }
appreciation	très moyen
notes	[ 4 elements ]
[0]	1.0
[1]	1.0
[2]	2.0
[3]	3.0
(3) ObjectId("544254ba3b447b3ecda889d9")	{ 3 fields }

MongoDB fournit des opérateurs de comparaison (*\$lte*, *\$lt*, *\$gt* etc.) et des opérateurs logiques (*\$or*, *\$not*, *\$and* etc.) qui permettent de combiner les tests.

<https://docs.mongodb.com/manual/reference/operator/query/#query-selectors>

Ces opérateurs peuvent être utilisés sur des attributs simples ou composés, comme des tableaux.

Dans le deuxième cas, ils fonctionnent comme des opérateurs ensemblistes en SQL (*EXISTS*, *ANY*, *ALL*, *IN*).

Dans l'exemple ci-dessus, *\$lte* est appliqué au tableau des notes : ce qui signifie qu'il existe au moins une note inférieure ou égale à 1.



Exercice 6 : Lister les bibliothèques qui n'ont aucune note strictement supérieure à 3 (y compris celles qui n'ont pas de notes), avec leur nom, ville, appréciation et notes

```

/* 2 */
{
  "fiche" : {
    "adresse_ville" : "Chilly-Mazarin",
    "nom" : "Médiathèque Municipale"
  },
  "notation" : {
    "appreciation" : "très moyen",
    "notes" : [
      1.0,
      1.0,
      1.0,
      3.0
    ]
  }
}
/* 3 */
{

```

etc. (970 bibliothèques en tout)



Exercice 7 : Lister le nombre de bibliothèques qui ont été notées (50 bibliothèques)

Et le nombre de bibliothèques sans aucune note

(956 bibliothèques)



Exercice 8 : Lister les bibliothèques qui n'ont aucune note strictement supérieure à 3 avec leur nom, ville et notes, en limitant la recherche aux bibliothèques qui ont au moins une note

```
/* 1 */
{
  "fiche" : {
    "adresse_ville" : "Chilly-Mazarin",
    "nom" : "Mediatheque Municipale"
  },
  "notation" : {
    "notes" : [
      1.0,
      1.0,
      2.0,
      3.0
    ]
  }
}

/* 2 */
{
  "fiche" : {
    "adresse_ville" : "Eourdan",
    "nom" : "Mediatheque Municipale"
  },
  "notation" : {
    "notes" : [
      2.0,
      3.0
    ]
  }
}
```

(14 bibliothèques en tout)



Exercice 9 : Lister les bibliothèques qui ont au moins une note strictement inférieure à 2 et aucune note strictement supérieure à 3

```
/* 1 */
{
  "fiche" : {
    "adresse_ville" : "Chilly-Mazarin",
    "nom" : "Mediatheque Municipale"
  },
  "notation" : {
    "notes" : [
      1.0,
      1.0,
      2.0,
      3.0
    ]
  }
}

/* 2 */
{
  "fiche" : {
    "adresse_ville" : "Egly",
    "nom" : "Bibliothèque"
  },
  "notation" : {
    "notes" : [
      1.0,
      1.0,
      2.0,
      3.0
    ]
  }
}
```

(2 bibliothèques en tout)



Exercice 10 : La ville, le nom, l'appréciation et les notes des bibliothèques qui ont une appréciation "moyen" ou "très moyen"

(utilisez l'opérateur \$in pour tester l'appartenance de l'appréciation à une de ces catégories)

1	{ 2 fields }	Object
2	{ 2 fields }	Object
fiche	{ 2 fields }	Object
notation	{ 2 fields }	Object
appréciation	titre moyen	String
notes	[ 4 elements ]	Array
3	{ 2 fields }	Object
4	{ 2 fields }	Object
5	{ 2 fields }	Object
6	{ 2 fields }	Object
7	{ 2 fields }	Object
fiche	{ 2 fields }	Object
notation	{ 2 fields }	Object
appréciation	notes	String
notes	[ 4 elements ]	Array
8	{ 2 fields }	Object
9	{ 2 fields }	Object
10	{ 2 fields }	Object
11	{ 2 fields }	Object
12	{ 2 fields }	Object
13	{ 2 fields }	Object

(15 bibliothèques en tout)



Exercice 11 : La ville, le nom et les notes des bibliothèques dont la plus ancienne note est 1 (c'est-à-dire le premier élément du tableau notes, "notation.notes.0")

1	{ 2 fields }
fiche	{ 2 fields }
notation	{ 1 field }
notes	[ 4 elements ]
0	1.0
1	2.0
2	3.0
3	5.0
2	{ 2 fields }
fiche	{ 2 fields }
notation	{ 1 field }
notes	[ 4 elements ]
0	1.0
1	1.0
2	2.0
3	3.0
3	{ 2 fields }
fiche	{ 2 fields }
notation	{ 1 field }
notes	[ 3 elements ]
0	1.0
1	2.0
2	3.0

(3 bibliothèques en tout)



Exercice 12 : La ville, le nom et les notes des bibliothèques qui ont une appréciation "ok" et exactement trois notes, dans l'ordre : 5, 5, 4 (on peut comparer un tableau à une liste de constantes : notes:[5,5,4] )

```
/* 1 */
{
  "fiche" : {
    "adresse_ville" : "Boissy-sous-Saint-Yon",
    "nom" : "Bibliothèque Municipale Georges Brassens"
  },
  "notation" : {
    "notes" : [
      5.0,
      5.0,
      4.0
    ]
  }
}
```

(1 seule bibliothèque)

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

### 3.4.7 Le pipeline d'agrégation : *aggregate*

La syntaxe de MongoDB permet d'enchaîner plusieurs opérations, dans un « pipeline » avec la méthode *aggregate* :

- des filtres (par *\$match*),
- des tris (par *\$sort*),
- des projections en choisissant les colonnes à afficher (par *\$project*),
- des groupements (équivalents du GROUP BY en SQL, par un *\$group*).

<https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/>

Par exemple, l'exercice précédent « *Chercher les bibliothèques dont la plus ancienne note est 1, en n'affichant que la ville, le nom et les notes* » peut se décomposer en un filtre (comme le WHERE en SQL) et une projection (comme le SELECT du SQL)

```
db.biblio.aggregate (  
[  
  {$match : {"notation.notes.0":1} },  
  {$project : {"fiche.adresse_ville":1, "fiche.nom":1, "notation.notes":1, _id : 0 } }  
]  
)
```

A noter : la liste des opérations à effectuer dans l'ordre, est passée à la méthode *aggregate* dans un tableau.



```
{  
  "fiche" : {  
    "adresse_ville" : "Boussy-Saint-Antoine",  
    "nom" : "BOUSSY-Bibliothèque Communautaire"  
  },  
  "notation" : {  
    "notes" : [  
      1.0,  
      2.0,  
      3.0,  
      5.0  
    ]  
  }  
}  
  
{  
  "fiche" : {  
    "adresse_ville" : "Egley",  
    "nom" : "Bibliothèque"  
  },  
  "notation" : {  
    "notes" : [  
      1.0,  
      2.0,  
      3.0  
    ]  
  }  
}
```

(3 bibliothèques en tout)



Exercice 13 : Réalisez le même filtre sur la première note, en triant les bibliothèques par ordre croissant des villes, et en n'affichant que les noms de la ville et de la bibliothèque.

Pour trier, ajouter un *\$sort* dans le tableau, après le *\$match* :

```
{ $sort : {"fiche.adresse_ville":1} }
```

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »



```

1 // 1.1
2 {
3   "fiche" : {
4     "adresse_ville" : "Boussy-Saint-Antoine",
5     "nom" : "BOUSSY-Bibliothèque Communautaire"
6   }
7 }
8
9 // 2.1
10 {
11   "fiche" : {
12     "adresse_ville" : "Chilly-Mazarin",
13     "nom" : "Médiathèque Municipale"
14   }
15 }
16
17 // 3.1
18 {
19   "fiche" : {
20     "adresse_ville" : "Egry",
21     "nom" : "Bibliothèque"
22   }
23 }

```

### 3.4.8 Utiliser des variables *JavaScript* pour simplifier le code des requêtes

La syntaxe des requêtes en JSON devient vite lourde, en particulier dans une méthode *aggregate*.

Comme les requêtes sont écrites en JavaScript, on peut utiliser des variables intermédiaires JavaScript pour simplifier la syntaxe.

```

varMatch = {$match : { "notation.notes.0":1 } };
varSort = {$sort : {"fiche.adresse_ville":1}};
varProject = { $project : {"fiche.adresse_ville":1, "fiche.nom":1, _id : 0 } } ;
// requête simplifiée
db.biblio.aggregate( [ varMatch, varProject, varSort ] )

```



Exercice 14 : Refaites l'exercice précédent avec cette syntaxe simplifiée, et vérifiez que l'on obtient le même résultat.

### 3.4.9 Le pipeline avec *group by* et fonctions d'agrégation

La méthode *\$group* dans le pipeline de l'*aggregate* est l'équivalent du GROUP BY en SQL avec la syntaxe :

```
{ $group : { _id : "critère de regroupement", "nom attribut" : {$fonctionAgregation :1} }
```

Le critère de regroupement est un des attributs de l'objet, qui va déterminer des sous-ensembles (par exemple *\$fiche.dept* si l'on regroupe par département)

Ou *null* si l'on veut travailler sur tout l'ensemble, sans le décomposer.

Les fonctions d'agrégation sont sensiblement les mêmes qu'en SQL, mais il faut tout de même faire attention :


- *\$sum* :
  - avec la syntaxe *{ \$sum :1 }* compte le nombre de documents, comme *count* en SQL
  - avec la syntaxe *{ \$sum :attribut }* fait la somme d'un attribut, comme *sum* en SQL

[https://docs.mongodb.com/manual/reference/operator/aggregation/sum/#grp.\\_S\\_sum](https://docs.mongodb.com/manual/reference/operator/aggregation/sum/#grp._S_sum)

- $\$min$  : valeur minimale de l'attribut
  - $\$max$  : valeur maximale de l'attribut
  - $\$avg$  : moyenne de l'attribut
- etc.

```
// on utilise un $group sur tout l'ensemble, sans critère de regroupement (_id = null) pour compter les
//bibliothèques dont la première note est 1
varMatch = {$match : { "notation.notes.0":1 } };
varGroup = {$group : { _id:null, "nombre" : {$sum:1} } }
db.biblio.aggregate( [ varMatch, varGroup ] )
```


```
/* 1 */
{
  "_id" : null,
  "nombre" : 3.0
}
```

 Exercice 15 : Le résultat obtenu est correct mais peu lisible (" $\_id$ " : null).

Retirez les affichages inutiles

(en ajoutant une projection dans le tableau, après le regroupement) :

```
/* 1 */
{
  "nombre" : 3.0
}
```

 Exercice 16 : Regroupez les bibliothèques par département, en comptant leur nombre par département. Affichez chaque numéro de département et le nombre de bibliothèques du département.

```
{ $group : { _id: "$fiche.dept", "nombre" : { $sum: 1 } } }
```

1 (1) 81	{ 2 Fields }
111 _id	81
111 nombre	163.0
2 (2) 84	{ 2 Fields }
111 _id	84
111 nombre	78.0
3 (3) 75	{ 2 Fields }
111 _id	75
111 nombre	58.0
4 (4) 82	{ 2 Fields }
111 _id	82
111 nombre	73.0
5 (5) 93	{ 2 Fields }
111 _id	93
111 nombre	83.0
6 (6) 95	{ 2 Fields }
111 _id	95
111 nombre	125.0
7 (7) 77	{ 2 Fields }
111 _id	77
111 nombre	223.0
8 (8) 78	{ 2 Fields }
111 _id	78
111 nombre	202.0

(8 objets en tout)

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

### 3.4.10 Utiliser d'autres fonctions d'agrégation : moyenne, min, max

```
// Calculer la moyenne des notes des bibliothèques, regroupées par appréciation
varMatch = { $match : { notation: { $exists: 1 } } };
varUnwind = { $unwind : "$notation.notes" }
varGroup = { $group : { _id: "$notation.appreciation",
                      "moyenne" : { $avg: "$notation.notes" } } }
db.biblio.aggregate( [ varMatch, varUnwind, varGroup ] )
```

(1) ok	{ 2 fields }
_id	ok
moyenne	4.12121212121212
(2) moyen	{ 2 fields }
_id	moyen
moyenne	2.33333333333333
(3) très moyen	{ 2 fields }
_id	très moyen
moyenne	1.85714285714286

A noter :

- Il faut vérifier qu'un objet (*notation*) existe par *\$exists*, avant de prendre un de ses attributs (*appreciation*) comme critère de regroupement
- On ne peut pas appliquer directement les fonctions d'agrégation sur des collections imbriquées dans un objet : ici sur la collection *notes* qui appartient à l'objet *notation*
- Il faut d'abord recréer une collection au premier niveau de l'objet principal en appliquant la méthode *\$unwind* (sur *\$notation.notes*)



Exercice 17 : Regroupez les bibliothèques par département, en affichant la note la plus haute et la note la plus basse par département.

✓ (1) 92	{ 3 fields }
_id	92
maximum	5.0
minimum	2.0
✓ (2) 91	{ 3 fields }
_id	91
maximum	5.0
minimum	1.0

Attention : les min et les max portent sur la collection *notes* qui est imbriquée dans l'objet *notation*. Pour accéder à ses éléments, il faut d'abord utiliser l'opérateur *\$unwind*

```
{ $unwind : "$notation.notes" }
```

### 3.4.11 Modifier la base : ajouter un champ, modifier un champ, supprimer un champ sur un objet existant

- Comme en SQL, on va pouvoir modifier la base, par la méthode *update*.
- Nous n'allons pas raisonner en tables et en lignes mais sur des objets et des attributs :

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

- Une base NoSQL contient des collections d'objets JSON que l'on peut modifier directement, ou modifier en JavaScript puis sauvegarder.
- Avantage du NoSQL : on peut modifier non seulement le contenu de l'objet (les valeurs des attributs) mais également sa structure, en ajoutant, supprimant des attributs ou en modifiant leur nom.
- Pour commencer, nous allons ajouter grâce à la méthode *update* et l'opérateur *\$set*, un commentaire avec la mention **\*\*\* Coup de cœur \*\*\*** à une bibliothèque avec une appréciation "ok" et des notes élevées (que vous choisirez vous-même dans votre base) :

```
db.biblio.update (
  { _id:ObjectId("5d4054ba3b447b3ecda889d0")},
  { $set : { "commentaire " : "*** Coup de coeur ***" } }
)
```

Le premier paramètre de la méthode *update* décrit le ou les objets à modifier (l'équivalent du WHERE)

Le deuxième paramètre décrit les différentes modifications à effectuer (par l'opérateur *\$set*)

- Puis, vous vérifierez que le nouvel attribut a bien été créé, en faisant un *find* :

```
db.biblio.find ( { _id:ObjectId("5d4054ba3b447b3ecda889d0")})
```

(1) ObjectId("5d4054ba3b447b3ecda889d0")	{ 5 fields }
_id	ObjectId("5d4054ba3b447b3ecda889d0")
fiche	{ 17 fields }
notation	{ 2 fields }
record_timestamp	2018-06-08T15:12:13+02:00
commentaire	*** Coup de coeur ***

- On pourra aussi modifier la valeur d'un champ existant (équivalent d'un « setCommentaire » en objet)

```
db.biblio.update (
  { _id:ObjectId("5d4054ba3b447b3ecda889d0")},
  { $set : { "commentaire " : " – En fait, assez moyen !!" } } )
```

(1) ObjectId("5d4054ba3b447b3ecda889d0")	{ 5 fields }	Object
_id	ObjectId("5d4054ba3b447b3ecda889d0")	ObjectId
fiche	{ 17 fields }	Object
notation	{ 2 fields }	Object
record_timestamp	2018-06-08T15:12:13+02:00	String
commentaire	– En fait, assez moyen !!	String

- Ou supprimer un champ existant :

```
db.biblio.update (
  { _id:ObjectId("5d4054ba3b447b3ecda889d0")},
  { $unset : { "commentaire " : 1 } }
)
```

(1) ObjectId("5d4054ba3b447b3ecda889d0")	{ 4 fields }
_id	ObjectId("5d4054ba3b447b3ecda889d0")
fiche	{ 17 fields }
notation	{ 2 fields }
record_timestamp	2018-06-08T15:12:13+02:00

Persistier des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »



Exercice 18 : Faites toutes les opérations précédentes sur un attribut de votre choix, en jouant à la fois sur le contenu et la structure des objets.

Pour chaque manipulation, vérifiez le résultat par un `find`

#### 3.4.12 Modifications multiples (sur plusieurs objets d'une collection)

- Les manipulations précédentes sont utiles, mais elles ne s'appliquent qu'à un seul objet désigné par son identifiant `_id`
- On a aussi besoin de traitements génériques : augmenter de 10% le prix de tous les ours en peluche...
- La méthode `update` doit alors comporter une clause « *multi* » qui autorise les modifications multiples dans une collection (par défaut, elles sont interdites, par mesure de sécurité) :

```
{"multi" : true}
```



Exercice 19 : En généralisant le cas précédent, ajoutez un commentaire « A visiter ... » à toutes les bibliothèques qui ont reçu une appréciation « ok » et une première note à 5.

(35 enregistrements modifiés).

Vérifiez le traitement par un `find`, en n'affichant que les noms, l'appréciation et les notes, et le commentaire

```
/* 1 */
{
  "fiche" : {
    "nom" : "Bibliothèque Municipale"
  },
  "notation" : {
    "appreciation" : "ok",
    "notes" : [
      5.0,
      4.0,
      5.0
    ]
  },
  "commentaire" : "A visiter ..."
}

/* 2 */
```

(etc. 35 objets en tout).

On voit qu'on peut enrichir notre base, sans « tout casser », de manière plus souple qu'avec une base relationnelle.

#### 3.4.13 Renommer un attribut



Exercice 20 : Dans toute la collection, renommer l'attribut `record_timestamp` en `date_maj`, pour faciliter la compréhension de la base.

Utilisez l'opérateur `$rename`, en n'oubliant pas la clause *multi* :

```
{ $rename: { "xxx": "yyy" } }, { multi: true }
```

Persistez des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

(1) ObjectId("5d4054ba3b447b3ecda889cc")	{ 5 fields }
_id	ObjectId("5d4054ba3b447b3ecda889cc")
fiche	{ 16 fields }
notation	{ 2 fields }
commentaire	A visiter ...
date_maj	2018-06-08T15:12:13+02
(2) ObjectId("5d4054ba3b447b3ecda889cd")	{ 5 fields }
_id	ObjectId("5d4054ba3b447b3ecda889cd")
fiche	{ 17 fields }
notation	{ 2 fields }
commentaire	A visiter ...
date_maj	2018-06-08T15:12:13+02
(3) ObjectId("5d4054ba3b447b3ecda889ce")	{ 5 fields }
(4) ObjectId("5d4054ba3b447b3ecda889cf")	{ 4 fields }
(5) ObjectId("5d4054ba3b447b3ecda889d0")	{ 5 fields }
(6) ObjectId("5d4054ba3b447b3ecda889d1")	{ 5 fields }

(etc. 1006 enregistrements modifiés)

#### 3.4.14 Manipuler des attributs de type tableau

Les tableaux peuvent être manipulés comme des listes, où l'on peut ajouter ou supprimer le premier ou le dernier élément

*// On considère que les premières notes sont trop anciennes et doivent être retirées de la liste des notes.*

**db.biblio.update({}, { \$pop: { "notation.notes" : -1 }, { multi: true })**

L'opérateur \$pop permet de supprimer un élément en tête de tableau (avec -1) ou en fin de tableau (avec 1)

<https://docs.mongodb.com/manual/reference/operator/update/pop/#definition>



Exercice 21 : Faites l'essai ci-dessus et vérifiez le fonctionnement par un find.

#### 3.4.15 Insérer de nouveaux enregistrements dans une collection

L'insertion se fait par la méthode *insertOne* :

*// directement*

**db.biblio.insertOne ({categorie:"Bibliothèque Nationale", nom:"BNF Richelieu", ville:"Paris" })**

*// ou en passant par un objet de travail*

**nouvBibli = {categorie:"Bibliothèque Nationale", nom:"BNF François Mitterrand", ville:"Paris" }  
db.biblio.insertOne (nouvBibli)**

Mais on peut aussi créer un enregistrement à partir d'un enregistrement existant, en utilisant JavaScript.

*// rechercher l'enregistrement : ici "BNF Richelieu" qui va servir de modèle*

*// on crée "Beaubourg" sur ce modèle*

**db.biblio.find ({nom:"BNF Richelieu"}).forEach (**  
**function (bib)**  
**{**  
*// print (bib.categorie, bib.ville)*  
**}**

```
nouvBibli = {categorie:bib.categorie, nom:"Beaubourg", "ville":bib.ville }  
db.biblio.insertOne (nouvBibli)  
  
}  
  
)
```

La méthode *forEach* utilisée sur la collection renvoyée par le *find* permet de parcourir cette collection et d'exécuter sur chaque élément une fonction JavaScript passée en paramètre.

Le paramètre « *bib* » passé à la fonction est l'élément courant dans la collection.

#### 3.4.16 Supprimer des enregistrements, selon un critère

```
// toutes les bibliothèques de la catégorie "Bibliothèque Nationale"  
db.biblio.deleteMany( {categorie:"Bibliothèque Nationale"} )
```



Exercice 22 : Faites différents essais d'insertion, avec des constantes et en se basant sur des enregistrements existants. Puis supprimez tous les nouveaux enregistrements, pour remettre la base dans son état initial.

#### 3.4.17 Utiliser JavaScript pour des opérations plus complexes.

Le parcours de collection par *forEach* permet de faire des traitements complexes en JavaScript sur les éléments.



Exercice 23 : Ajouter une moyenne des notations à chaque bibliothèque :

- Attention : toutes les bibliothèques n'ont pas de notation
- Dans un objet JavaScript « *bib* », pour créer un nouvel attribut « moyenne », il suffit de l'initialiser :

*bib.moyenne = laMoyenne ;*

et de sauver l'objet « *bib* » dans la base de données :

*db.biblio.save (bib)*



Exercice 24 : Faire une extraction de données pour peupler une autre collection.

(Cette exercice suit le précédent : consultez d'abord le corrigé précédent, si vous avez un doute)

- Créez une collection « *appreciation* » dont les objets comprendront 4 attributs :
  - *idbibli* : id de la bibliothèque,
  - *appreciation* (tirée de *notation.appreciation*),
  - *moyenne* (tirée de la moyenne calculée précédemment)
  - *notes* (tirées de *notation.notes*)

(50 objets en tout)

- Supprimez ensuite les attributs *notation* et *moyenne* de la collection d'origine, pour éviter la redondance.

Persistez des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

Pour peupler la nouvelle collection, on utilisera la méthode *insertOne* avec des objets définis en JSON, à partir des attributs de l'objet courant « bib » dans la liste des bibliothèques :

*{idbibli : bib.\_id, appreciation:bib.notation.appreciation, etc. }*



Pour supprimer les champs *notation* et *moyenne* dans la collection « *biblio* », faire un *\$unset* sur ces champs, avec l'option « *multi* »

### 3.4.18 Faire une "jointure" entre plusieurs collections

On peut déjà faire beaucoup de choses avec une seule collection, en NoSQL.

Mais il restera des cas où une jointure ou son équivalent sera nécessaire entre deux collections, pour éviter ou limiter la redondance de données.

L'exercice précédent a préparé le travail : la collection "*appreciation*" contient maintenant les notes, moyenne et appréciation des bibliothèques, qui sont référencées par leur id (champ *idbibli*, qui tient lieu de clé étrangère).

Ces attributs ne sont plus dans la collection d'origine.

Nous allons maintenant croiser les données de ces deux collections par un *\$lookup* (équivalent du *join* sur MongoDB) :

```
{
  $lookup:
  {
    from: <collection to join>,
    localField: <field from the input documents>,
    foreignField: <field from the documents of the "from" collection>,
    as: <output array field>
  }
}
```

<https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/>

```
db.appreciation.aggregate([
{
  $lookup:
  {
    from: "biblio",
```

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »



```

    localField: "idbibli",
    foreignField: "_id",
    as: "bibliotheque"
  } })

```



Exercice 25 : Faites le traitement ci-dessus et observez le résultat

```

{
  "_id" : ObjectId("5d8c4948657cd00e68d771a2"),
  "idbibli" : ObjectId("5d4034ba1b447b3ecda889cc"),
  "appreciation" : "ok",
  "moyenne" : 4.66666666666667,
  "notes" : [
    5.0,
    4.0,
    3.0
  ],
  "bibliotheque" : [
    {
      "_id" : ObjectId("5d4034ba1b447b3ecda889cc"),
      "fiche" : {
        "categorie" : "Bibliothèque municipale",
        "region" : "Ile-de-France",
        "code bib" : 4067.0,
        "insee" : "91021",
        "nom_depr" : "ESSONNE",
        "code ua" : 4067.0,
        "voie" : "31 rue Dauvilliers",

```

(50 objets en tout)

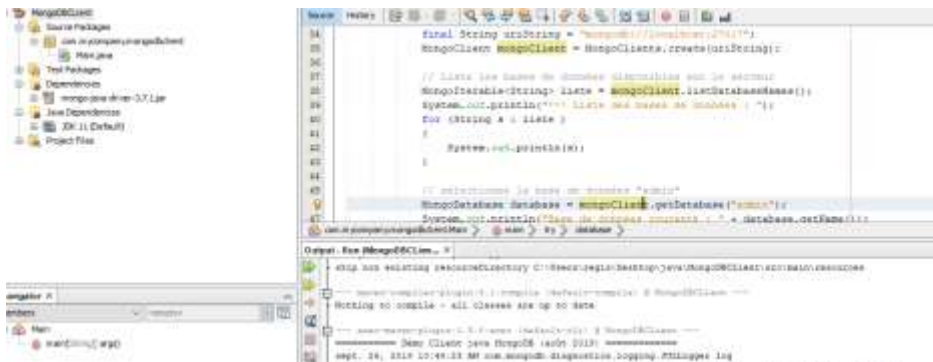
Les attributs de la bibliothèque « *idbibli* » sont stockés dans un tableau déclaré par la clause « *as* » du lookup : « *bibliotheque* »

### 3.5 APPELER MONGODB DEPUIS UN CLIENT JAVA

Nous allons maintenant laisser de côté l'interpréteur JavaScript et écrire quelques requêtes vers MongoDB avec une bibliothèque Java.

#### 3.5.1 Création du projet Maven

- Nous allons vous guider dans la création d'un premier client lourd Java vers MongoDB (fourni dans *Livrables/MongoClient*).
- Vous allez vous approprier le code commenté, réaliser et tester chaque étape.
- Nous travaillons en NetBeans mais les sources peuvent être utilisés sur l'IDE de votre choix (Eclipse etc.)



- Créez un projet Java « maven » avec le fichier pom suivant, qui permet de charger le driver de mongo-db :

Persistez des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

[illegible]

### 3.5.2 Les premiers pas : valider l'environnement

Pour vérifier que notre client parvient à interroger le serveur MongoDB local, créez une classe *Main* dans votre projet, en vous inspirant de l'extrait de code qui suit :

```
package com.mycompany.mongoclient;

import com.mongodb.client.*;
public class Main {
    public static void main(String[] args)
    {
        try
        {
            System.out.println("===== Démo Client java MongoDB (oct 2019) =====");

            final String uriString = "mongodb://localhost:27017";
            MongoClient mongoClient = MongoClient.create(uriString);

            // Liste les bases de données disponibles sur le serveur
            MongoIterable<String> liste = mongoClient.listDatabaseNames();
            System.out.println("*** Liste des bases de données : ");
            for (String s : liste )
            {
                System.out.println(s);
            }
        }
        catch (Exception e)
        {
            System.out.println("Exception : " + e.getMessage());
        }
    }
}
```

- Le programme se connecte au serveur MongoDB local grâce à la bibliothèque *MongoClient*
- Puis il liste les bases de données présentes sur ce serveur : *admin*, *config*, *local* etc.

```

Output - Run [Mal... X]
--- exec-maven-plugin:1.5.0:exec (default-cli) @ MongoClient ---
----- Demo Client java MongoDB (oct 2019) -----
sept. 26, 2019 11:20:34 AM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, required
... Liste des bases de données :
sept. 26, 2019 11:20:34 AM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster description not yet available. Waiting for 30000 ms before timing out
sept. 26, 2019 11:20:34 AM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId[localValue:1, serverValue:39]] to localhost:2
sept. 26, 2019 11:20:34 AM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescri
sept. 26, 2019 11:20:35 AM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId[localValue:2, serverValue:40]] to localhost:2
admin
config
local
-----

```

### 3.5.3 Une première requête simple

Nous allons commencer classiquement par compter le nombre d'objets dans notre collection « biblio »

Persister des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

```
// sélectionne la base de données "admin"
MongoDatabase database = mongoClient.getDatabase("admin");
System.out.println("Base de données courante : " + database.getName());

// affiche le nombre de documents de la collection "biblio"
MongoCollection<Document> col = database.getCollection("biblio");
System.out.println("Nombre de bibliothèques : " + col.count());
```

Le programme sélectionne la base de donnée *admin* et charge la collection *biblio* sur laquelle il fait un *count* :

```
Base de données courante : admin
Nombre de bibliothèques : 1006
```

### 3.5.4 Une requête paramétrée

Nous allons faire un *find* avec un paramètre en JSON pour ne récupérer que les bibliothèques du département de l'Essonne.

```
// affiche le nombre de documents de la collection "biblio"
MongoCollection<Document> col = database.getCollection("biblio");
System.out.println("Nombre de bibliothèques : " + col.count());

// Affiche les bibliothèques de l'ESSONNE présentes dans la collection
Bson bson = (Bson) JSON.parse("{fiche.nom_dept:'ESSONNE'}");
System.out.println("*** Bibliothèques de l'essonne : \n");
FindIterable<Document> listeEssonne = col.find(bson);
for (Document d : listeEssonne)
{
    Document fiche = (Document) d.get("fiche");
    System.out.println("Département : " + fiche.get("nom_dept")
        + "\tNom : " + fiche.get("nom"));
}
```

- Le paramètre pour filtrer est un objet JSON, comme dans les appels en JavaScript :  
`{fiche.nom_dept:'ESSONNE'}`
- Pour l'API de MongoDB en Java, le format JSON doit au préalable être converti en format Bson (*Binary Json*) qui est aussi le format de stockage de MongoDB.
- Dans chaque objet de la liste (de type *org.bson.Document*), on extrait d'abord l'objet "fiche" par un *get* sur le nom de l'attribut, puis dans cet objet, les attributs "nom\_dept" et "nom" de type String.

```
*** Bibliothèques de l'essonne :
```

```
Département :ESSONNE    Nom : Bibliothèque Municipale
Département :ESSONNE    Nom : Bibliothèque Municipale
Département :ESSONNE    Nom : Bibliothèque Municipale Georges Brasser
Département :ESSONNE    Nom : BOUSSY-Bibliothèque Communautaire
Département :ESSONNE    Nom : BRUNOY-Bibliothèque Communautaire
Département :ESSONNE    Nom : Bibliothèque Jean-Jacques SEMPE
Département :ESSONNE    Nom : Mediatheque Municipale De Cheptainville
Département :ESSONNE    Nom : Mediatheque Municipale
Département :ESSONNE    Nom : Bibliothèque
Département :ESSONNE    Nom : Médiathèque Municipale
```

(etc. 163 bibliothèques en tout).

<https://howtodoinjava.com/mongodb/mongodb-find-documents/>

Persisten des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

### 3.5.5 Insérer de nouveaux objets et les afficher

- En NoSQL, nous ne sommes pas limités par un modèle de données préalable.
- Pour illustrer ce point, nous allons insérer dans la collection « *biblio* » de nouveaux objets qui ne seront pas des bibliothèques.
- Dans la vie réelle, on n'insère pas n'importe quoi dans une collection existante. Rappelons la règle d'or de l'objet qui reste valable en NoSQL : pas de fourre-tout ; la souplesse ne doit pas conduire à une absence de structuration.
- Mais l'on pourra ajouter de nouveaux attributs à l'objet bibliothèque, d'autres types d'objet proche de la bibliothèque ou en lien fonctionnel avec elle (bibliobus, points d'échange de livres, administration en charge des bibliothèques etc.)
- Pour la démonstration technique, nous allons ajouter deux objets simples, avec un seul attribut « *essai* » de type entier, et les réafficher :

```
// création de deux documents "essai" (pour démo)
System.out.println("*** Création de documents : ");
for (int i = 1; i <= 2; i++)
{
    Document nouveau = new Document().append("essai", i);
    col.insertOne(nouveau);
}

System.out.println("*** Documents créés : ");
bson = (Bson) JSON.parse("{essai:{exists:true}}");
for (Document d : col.find(bson))
{
    System.out.println(d);
}
```

- La méthode *append* de la classe *Document* permet d'ajouter un attribut (clé, valeur) à un document.
- La méthode *insertOne* permet d'ajouter un nouveau document dans une collection de documents existante.
- On filtre la collection (par un *find*) sur les objets qui possèdent un attribut « *essai* » et on affiche la collection filtrée.

```
/* 1 */
{
  "_id" : ObjectId("5d8c97cb24007f0cc02efc2f"),
  "essai" : 1
}

/* 2 */
{
  "_id" : ObjectId("5d8c97cb24007f0cc02efc30"),
  "essai" : 2
}
```

Persistier des données non structurées (semi-structurées) dans un SGBD NoSQL

Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

### 3.5.6 Supprimer des objets dans une collection

- On supprimera les objets dans une collection « *col* » par la méthode *deleteMany* avec le même filtre BSON que pour un *find*
- On réaffichera les objets avec le même filtre pour vérifier leur suppression.

```
col.deleteMany(bson);

// après suppression
System.out.println("==> Après suppression, liste des nouveaux documents : ");
for (Document d : col.find(bson))
{
    System.out.println(d);
}
```

```
*** Création de documents :
*** Documents créés :
Document({_id=5d8c97cb24007f0cc02efc2f, essai=1})
Document({_id=5d8c97cb24007f0cc02efc30, essai=2})
Document({_id=5d970dca24007f1fa4268682, essai=1})
Document({_id=5d970dca24007f1fa4268683, essai=2})
==> Après suppression, liste des nouveaux documents :
-----
```

## CRÉDITS

### OEUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la DIIP  
et du centre sectoriel Tertiaire

### EQUIPE DE CONCEPTION

Régis Lécu – Formateur AFPA Pont de Claix

## Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »