

# Maîtrisez les bases de données NoSQL

🕒 15 hours 📶 Medium

License    

Last updated on 6/4/21



## Découvrez le fonctionnement de MongoDB



### Mise en place d'une base de données MongoDB

MongoDB est une base de données NoSQL orientée documents. Comme nous le verrons, l'ensemble du système tourne autour de cette gestion de documents, y compris le langage d'interrogation, ce qui en fait son point fort. Nous allons nous attaquer dès maintenant à la mise en place d'un serveur Mongo et comment intégrer vos données dans cet environnement.



### Installation

Pour bien fonctionner, une base *MongoDB* a besoin de trois choses :

- L'installation du serveur, que vous pouvez télécharger [ici](#) (version 3.4.7)



L'installation sous *Windows* est assez simple (appuyez sur suivant).

Sous Mac, décompressez l'archive et déplacez le répertoire dans vos applications "/Applications"

Sous Debian/Ubuntu, utilisez la commande "`sudo apt install mongodb`".

Le serveur sera alors installé dans le répertoire que je nommerai : **\$MONGO**.

Sous *Windows* : C:\Program Files\MongoDB\Server\3.4\

Sous *Linux* : /opt/local/bin/

Sous *Mac* : /Applications

- La création d'un répertoire pour stocker les données. Par exemple : C:\data\db (par défaut)



Sous *Windows*, il faut ouvrir l'explorateur de fichiers, aller à la racine "C:\", puis créer

<

Administrez vos données avec MongoDB

>

▶ 1. Découvrez le fonctionnement de MongoDB


2. Interrogez vos données avec MongoDB

3. Protégez-vous des pannes avec les ReplicaSet


4. Distribuez vos données avec MongoDB

5. Entraînez-vous à créer et à interroger une base de données MongoDB




Created by

  
CentraleSupélec

Grande École d'ingénieurs :  
cycle ingénieur, Master et  
École Doctorale, Mastère  
Spécialisé et formation  
continue



OpenClassrooms, Leading  
E-Learning Platform in  
Europe

le répertoire "data". Ensuite, aller dans ce répertoire et créer le répertoire "db".

Sous *Linux ou Mac*, il faut créer le répertoire data à la racine avec "sudo mkdir ~/data". Vous donnez les droits (ex. utilisateur 'toto') : "sudo chown toto ~/data". Et enfin, créer le répertoire db : "mkdir ~/data/db".

Sous *Mac*, décompressez l'archive et déplacez le répertoire dans vos applications "/Applications"

Il est possible de créer un autre répertoire dédié "\$REP" pour les données de MongoDB. Dans ce cas, le lancement du serveur ci-dessous devra utiliser le paramètre --dbpath \$REP. C'est le cas sous Mac comme effectué ci-dessus, cela donne : "mongod --dbpath ~/data/db"

- Le lancement du serveur, avec l'exécutable mongod (disponible sur \$MONGO/bin)


Si tout se passe bien, le serveur devrait tourner comme dans le screenshot ci-dessous (ne pas fermer la fenêtre sous peine d'éteindre la base de données). Le serveur attend une connexion sur le port 27017.

```
2017-06-20T23:40:42.406+0200 I CONTROL [initandlisten] MongoDB starting : pid=99067 port=27017 dbpath=/data/db 64-bit host=MacBook-Air, local
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] db version v3.6.3
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] git version: f87437f05adcca07c3b0fa78065456b10d5e1
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] OpenSSL version OpenSSL 1.0.2k 26 Jan 2017
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] allocator: system
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] modules: none
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] build environment:
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] distarch: x86_64
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] target_arch: x86_64
2017-06-20T23:40:42.408+0200 I CONTROL [initandlisten] options: {}
2017-06-20T23:40:42.409+0200 I - [initandlisten] Detected data files in /data/db created by the 'mmapv1' storage engine, so setting the active storage engine to 'mmapv1'.
2017-06-20T23:40:42.500+0200 I JOURNAL [initandlisten] Journal dir=/data/db/journal
2017-06-20T23:40:42.509+0200 I JOURNAL [initandlisten] recover: no journal files present, no recovery needed
2017-06-20T23:40:42.529+0200 I JOURNAL [initandlisten] (durability) durability thread started
2017-06-20T23:40:42.529+0200 I JOURNAL [journal writer] Journal writer thread started
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten]
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten]
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten]
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten] ** WARNING: soft rlimits too low. Number of files is 256, should be at least 1000
2017-06-20T23:40:42.529+0200 I CONTROL [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'
2017-06-20T23:40:42.878+0200 I NETWORK [thread1] waiting for connections on port 27017
```

Log de connexion du serveur mongod

## Interface utilisateur

Maintenant que le serveur tourne, on peut s'attaquer à son administration avec une interface graphique. Nous prendrons [Robo3T](#) version 1.1 (anciennement *robomongo*) qui est largement utilisé.

 L'installation est assez simple sur Windows et Mac, il suffit de suivre les instructions.

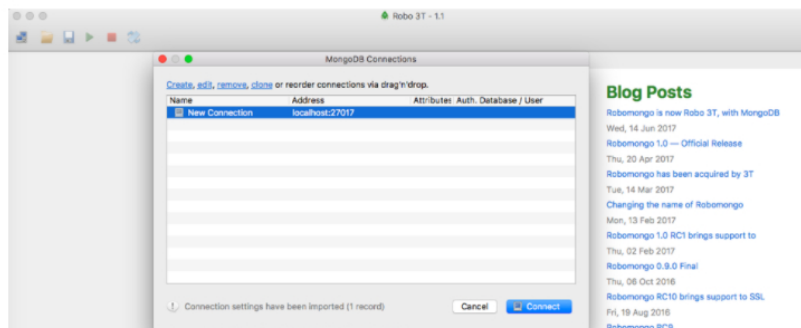
Sous *Windows*, l'application est installée par défaut dans le répertoire *C:\Program Files\3T\Robo3T*

Sous *Mac*, il suffit de placer Robo3T dans le répertoire "Applications"


Sous *Linux*, le fichier d'installation doit être décompressé (*tar xzvf robo3t-1.1.1-linux-x86\_64-c93c6b0.tar.gz*). Puis, vous pouvez exécuter le binaire "bin/robo3t"

Il peut être demandé la plateforme "Qt" avec le plugin "xcb". Dans ce cas, suivez la procédure indiquée [ici](#).

Après lancement de l'application, cliquez sur "create" comme dans l'image ci-dessous pour créer une nouvelle connexion à MongoDB (avec "localhost" et le port "27017" par défaut).



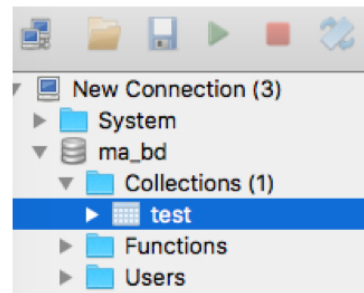
Création d'une connexion sous Robo3T

 Pour une utilisation plus intensive, je recommanderai de prendre une licence pour **Studio 3T** (Trial de 2 semaines). Il facilite les imports de données, la lecture de schéma, la conception de requêtes...

## Création d'une collection

Ca y est, nous pouvons manipuler la base de données et y ajouter une collection de documents. Une collection c'est l'équivalent d'une table pour une base de données relationnelle. Mais dans notre cas, une collection n'a pas de schéma donnant la structure d'une collection. Comme nous le verrons par la suite un document n'a pas de structure fixe, de fait, une base de données gère une 'collection' de documents, et non une table.

Créons tout d'abord une base de données 'ma\_bd' (bouton droit sur la connexion "Create database"), puis sur les collections de cette base, créer une collection "test" (bouton droit sur "Collections(0)")



Création d'une collection sous Robo3T

## Modélisation de documents

### Documents JSON

Maintenant que l'environnement est prêt, une question se pose : à quoi ressemblent mes données ? À des documents JSON. Le modèle est très simple :

1. Tout est clé/valeur : **"clé" : "valeur"**
2. Un document est encapsulé dans des accolades **{...}**, pouvant contenir des listes de clés/valeurs
3. Une valeur peut être un type scalaire (entier, nombre, texte, booléen, null), des listes de valeurs **[...]**, ou des documents imbriqués

On peut donc insérer un document dans notre collection "test" (double click sur la collection, champ de texte en noir pour exécuter une requête) :

```
1 db.test.save (
2 {
3   "cours" : "NoSQL",
4   "chapitres" : ["familles", "CAP", "sharding", "choix"],
5   "auteur" : {
6     "nom" : "Travers",
7     "prenom" : "Nicolas"
8   }
9 }
```

! Il faut savoir que MongoDB ne gère aucun schéma pour les collections. Donc pas de structure fixe ni de typage, ce qui fait la force des SGBDR. Du coup, il est difficile lors de l'interrogation de connaître le contenu de la base de données. Le logiciel **Studio3T** permet d'extraire des statistiques sur la structure des documents. Ce qui permet d'en avoir une vision globale.

### Relationnel vers JSON - comment faire ?

Cela semble simple, c'est à s'y méprendre ! En effet, les données utilisées de manière classiques sont en relationnel (Exportation en CSV pour faire simple). Mais voilà, nous devons faire face à un gros problème dans les bases NoSQL : Il faut proscrire les jointures !

De fait, si vous avez un SGBDR avec deux tables, vous ne pourrez pas faire de requêtes de jointure (ou alors, ça vous coûtera horriblement cher). Il faut dans ce cas trouver une solution pour fusionner les deux tables pour n'en produire qu'une seule en sortie. C'est ce qu'on appelle la dénormalisation.

Reprenons notre exemple précédent, mais avec un schéma un peu plus complexe (schéma de gauche). Une personne peut avoir plusieurs domaine d'expertise, des emplois successifs, et une habitation :



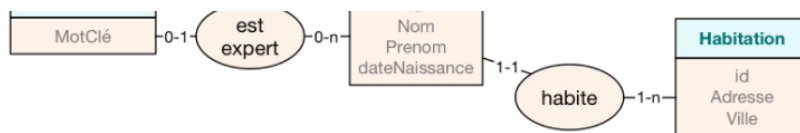


Schéma Entité/Association original

Si je dois intégrer une base de données avec une collection par entité (rectangle) et association (ovales), le nombre de jointures pour les requêtes sur la base NoSQL risque de faire exploser le système. Du coup, des fusions sont nécessaires pour réduire le coût des requêtes. Mais quelles tables doit-on imbriquer ? Dans quel sens le faire ?

## Dénormalisation du schéma

Voici quelques étapes de modélisation qui vont vous permettre de produire des documents JSON qui répondront à votre demande, tout en minimisant les problèmes de jointures et d'incohérences :

- **Des données fréquemment interrogées conjointement.**

Par exemple, les requêtes demandent fréquemment le lieu d'habitation d'une personne. De fait, la jointure devient coûteuse. Accessoirement, cette information étant peu mise à jour, cela pose peu de problèmes. Résultat, l'entité 'Habitation' et l'association 'habite' sont intégrés à l'entité Personne. Habitation devient un document imbriqué à l'intérieur de Personne, représenté par : "[habite]"

- **Toutes les données d'une entité sont indépendantes.**

Prenons l'exemple des domaines d'expertise d'une personne, ils sont indépendants des domaines d'une autre personne. De fait, rapatrier les données de cette entité n'impacte aucune autre instance de Personne. Ainsi, la liste des domaines est importé dans Personne et représenté par : "[domaines]"

- **Une association avec des relations 1-n des deux côtés.**

Cette fois-ci, c'est plus délicat pour l'entité Etablissement. Une personne peut avoir plusieurs emplois et un employeur, plusieurs employés. De fait, une imbrication de l'employeur dans Personne peut avoir de gros impacts sur les mises à jour (tous les employés à mettre à jour !). Il est donc peu recommandé d'effectuer une fusion complète. Pour cela, seule l'association est imbriquée sous forme d'une liste de documents, intégrant les attributs (qualité et date), ainsi qu'une référence vers l'employeur. Ainsi : "[{emploi+ref}]"

- **Même taux de mises à jour.**

Dans le cas des emplois d'une personne, là également nous pourrions effectuer une fusion de l'association "emploi". En effet, le taux de mises à jour des emplois est équivalent à celui de la Personne, de fait, sans incidence sur les problèmes de cohérence de données.



Ces étapes de modélisation sont un résumé de plusieurs articles de recherches en Système d'Information. Si vous souhaitez en lire d'avantage : Query-Driven [Chebotko15], MDA-NoSQL [Abdelhedi et al. 17], Document-oriented NoSQL modelling [Mason15].

Au vu de ces critères, voici le schéma qui pourrait être obtenu par fusion successives :

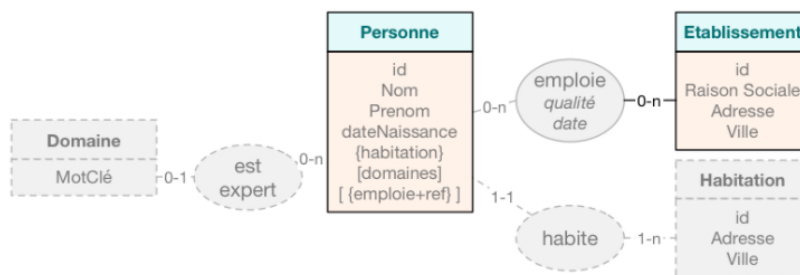


Schéma dénormalisé

Les listes sont représentées par des crochets et les imbrications par des accolades. Nous pourrions remarquer que les établissements sont stockés dans une association à part, mais que la référence est gardée dans une liste intégrée à l'entité "Personne". Voici un exemple d'instance de cette représentation. Vous pouvez remarquer que les établissements sont référencés et des informations peuvent ne pas être renseignées.

```
1 {
2   "id": 1, "nom": "Travers", "prenom": "Nicolas",
```

json

```

3  "domaines" : ["SGBD", "NoSQL", "RI", "XML"],
4  "emplois" : [
5    {"id_etablissement" : "100", "qualité" : "Maître de Conférences",
6     "date" : "01/09/2007"},
7    {"id_etablissement" : "101", "qualité" : "Vacataire",
8     "date" : "01/09/2012"}
9  ],
10 "Habite" : {"adresse" : "292 rue Saint Martin", "ville" : "Paris"}
11 }

```

Maintenant, nous sommes prêts pour concevoir des bases NoSQL orientées documents avec des fusions. Ne reste plus qu'à faire la conversion de données relationnelles vers ces documents. Pour cela, il est possible de créer un ETL (Extract-Transform-Load) avec votre langage de programmation préféré qui va, à la volée, chercher les informations nécessaires pour produire des données correspondant au schéma. Une seconde possibilité est d'utiliser l'outil [OpenRefine](#) (anciennement *GoogleRefine*) pour nettoyer et définir un format de sortie JSON. La dernière possibilité est d'intégrer chaque fichier dans MongoDB (chacun une collection distincte), puis d'utiliser les opérateurs \$lookup et \$redact pour produire la sortie souhaitée (les opérateurs MongoDB seront étudiés dans le chapitre suivant).

## Importer les données

Nous allons maintenant importer un jeu de données déjà formaté à notre base de données. Un jeu de données OpenData est disponible sur des restaurants de New York produits par la mairie sur les résultats des inspections. Pour l'importation :

1. Téléchargez l'archive suivante : [restaurants.zip](#)
2. Décompresser l'archive (j'appellerai le répertoire utilisé **\$RESTAURANT**)
3. Nous allons créer une base de données "**new\_york**" (paramètre `--db`) et une collection "**restaurants**" (paramètre `--collection`). Attention, il ne faut pas de majuscules !
4. Dans une console (Windows : invite de commande, Linux : Shell/Konsole), aller dans le répertoire `$MONGO/bin`
5. Exécutez la commande suivante :

```
$MONGO/bin/mongoimport --db new_york --collection restaurants $RESTAURANT/restaurants.json
```

Ca y est, vous avez une base de données de 25 357 restaurants que vous pouvez consulter directement avec Robo3T.

Key	Value	Type
['_id']	Objectid('594b9172c96c1e672dc06891')	Objectid
[address]	Objectid('594b9172c96c1e672dc06891')	Objectid
[building]	1007	String
[coord]	Point	Object
[type]	Point	Object
[coordinates]	[-73.856077, 40.848447]	Array
[street]	Morris Park Ave	String
[zipcode]	10462	String
[borough]	Bronx	String
[cuisine]	Bakery	String
[grades]	[5 elements]	Array
[date]	2014-03-03 00:00:00.000Z	Date
[grade]	A	String
[score]	2	Int32
[name]	Morris Park Bake Shop	String
[restaurant_id]	30075445	String

Screenshot de la collection "restaurants" sous Robo3T

Nous allons maintenant pouvoir interroger cette collection avec le langage proposé par MongoDB.

I FINISHED THIS CHAPTER. ONTO THE NEXT!

QUIZ: SAVEZ-VOUS VRAIMENT CE QU'EST LE NOSQL ?

INTERROGEZ VOS DONNÉES AVEC MONGODB

## Teachers



Régis Behmo

Expert en machine learning, développeur fullstack, grimpeur invétéré et gros, très gros amateur de nouilles

crinoides.



**Nicolas Travers**

Maitre de Conférences en informatique au CNAM Spécialités : Bases de données, Optimisation BDD, SQL, NoSQL

OPENCLASSROOMS

What we do

Apprenticeship

Path experience

Our blog

BUSINESS SOLUTIONS

Business

CONTACT



FAQ

LEARN MORE

Work at OpenClassrooms

Become a mentor [↗](#)

Our store

Terms of use

Privacy policy

Accessibility



English

