

DÉBUTER AVEC MONGODB POUR NODE.JS

Thunderseb

14 avril 2016

Table des matières

1	Introduction	5
2	Fonctionnement de MongoDB et structure des données	7
2.1	Des collections et des documents	7
2.2	Formats spéciaux	9
2.2.1	ObjectID	9
2.2.2	ISODate	9
2.2.3	Nombres	9
3	Installer MongoDB et faire quelques tests	11
3.1	Installer MongoDB	11
3.2	Quelques opérations	12
3.2.1	Lister et choisir la base de données	12
3.2.2	Insérer des documents	13
3.2.3	Faire des recherches	13
4	Se connecter depuis Node.js	15
4.1	Installer le module	15
4.2	Connexion à la base de données	15
5	Opérations depuis Node.js	17
5.1	L'ObjectID	17
5.2	Récupérer tous les documents d'une collection	17
5.3	Récupérer le document correspondant à un identifiant	18
5.4	Insérer un document	19
5.5	Insérer et/ou éditer	19
5.6	Éditer un document	19
5.7	Supprimer un document	20
6	Requêtes avancées	21
6.1	Plusieurs critères (AND)	21
6.2	Soit l'un soit l'autre (OR)	21
6.3	Conditions < <= > >= !=	21
6.4	Expressions régulières	22
7	Conclusion	23

1 Introduction

Vous utilisez **Node.js** et vous souhaitez utiliser le gestionnaire de base de données **NoSQL MongoDB** ? Alors bienvenue dans ce petit cours !

Lorsque je me suis mis à utiliser MongoDB, j'ai eu quelques difficultés à rassembler certaines informations dont j'avais besoin, à savoir :

- Comment est structurée une “base de données” MongoDB ?
- Comment installer et manipuler le shell de MongoDB ?
- Comment faire le lien entre mon application Node.js et MongoDB ?
- Comment faire en sorte que ça fonctionne (important, ça !)

Bon, en réalité, toutes ces informations sont trouvables sur le Web, mais c'est disparate, et beaucoup en anglais. Ou alors ça propose d'utiliser des trucs comme Mongoose et autres modules, or moi, je veux “le faire moi-même”, sinon ce n'est pas amusant.

[[attention]] | Ce tutoriel est simplement une *mise en route* pour utiliser MongoDB avec Node.js : ce n'est pas un tutoriel sur MongoDB, ni sur Node.js !

So, venez, on va étudier tout ça !

*[NoSQL] : Not Only SQL

2 Fonctionnement de MongoDB et structure des données

Tout comme MySQL, MongoDB est un système de gestion de bases de données (SGBD). Il s'agit donc d'un programme qui va gérer nos différentes bases de données. Sauf qu'ici, il ne s'agit pas du bon vieux SQL, mais de bases de données **NoSQL** orientées documents !

[[attention]] | Notez qu'une base de données MongoDB n'est pas faite pour remplacer une base de données relationnelle classique (par exemple MySQL, PostgreSQL, Oracle...). Il est courant de voir cohabiter deux SGBD : un relationnel, et un dit NoSQL.

2.1 Des collections et des documents

Chaque entrée d'une base MongoDB est appelée **document**. Un document n'est rien d'autre qu'un objet JSON contenant une série de clefs/valeurs. Voici un exemple de document :

```
{
  _id :      ObjectId("53dfe7bbfd06f94c156ee96e"),
  name :    "Adrian Shephard",
  game :    "Half-Life: Opposing Force",
  serie :   [
    "Half-Life",
    "Half-Life: Blue Shift",
    "Half-Life: Decay"
  ],
  options : {
    type : ["solo", "multiplayer"],
    os :   "Windows"
  },
  release : ISODate("1999-11-10T00:00:00Z")
}
```

Un document MongoDB est comparable à une entrée au sein d'une table SQL. Les documents sont rassemblés au sein de collections, qui sont donc les homologues des tables SQL. Voici un petit schéma démonstratif :



#	title	stuff	moar
1	Bla bla	Mdr	xD
3	TEST	Lmfao	XML
4	Azerty	GUI	Lol

```
{ { title: "Bla bla", stuff: "Mdr", moar: "xD" }  
  { title: "TEST", stuff: "Lmfao", moar: "XML" }  
  { title: "Azerty", stuff: "GUI", moar: "Lol" } }
```

->

<-

Le fait de structurer les documents en utilisant une syntaxe JSON les rend faciles à manipuler. Vous verrez, tout se fera en JavaScript, nul besoin d'apprendre un langage comme le SQL pour faire des requêtes, et l'accès aux propriétés des documents se fera en JavaScript "natif", comme s'il s'agissait d'objets tout à fait normaux!

[[information]] | En réalité, les documents ne sont pas stockés sous la forme de JSON, mais de BSON, qui est la représentation binaire du JSON. Mais ça ne change rien pour nous.

Considérons la collection **Personnages**. Cette collection va recevoir divers documents, un document par personnage. Le contenu de la collection pourrait s'écrire comme ceci :

```
// Document n° 1  
{ name : "Gordon Freeman", game : "Half-Life" }  
  
// Document n° 2  
{ name : "Chell Johnson", game : "Portal", gender : "F" }  
  
// Document n° 3  
{  
  name : "Soldier",  
  game : "Team Fortress²",  
  weapons : [  
    "Lance-roquettes",  
    "Shotgun",  
    "Pelle"  
  ]  
}
```

Quelque chose de flagrant doit vous sauter aux yeux : la structure de chaque document est différente!

Et pour cause, à l'inverse des tables SQL, aucune structure n'est imposée par MongoDB. Chaque document d'une collection peut différer d'un autre ! C'est au développeur de faire attention à garder une structure cohérente pour ne pas se perdre. Cette spécificité permet évidemment de gagner de l'espace, puisque si une clef est vide, il suffit de ne pas la déclarer. C'est donc un avantage en terme de stockage et accessoirement en terme de rapidité.

2.2 Formats spéciaux

Le premier code d'exemple montrait deux objets spéciaux : `ObjectId` et `ISODate`. Ce sont des “types” de données, reconnus par MongoDB.

2.2.1 ObjectId

`ObjectId` représente l'identifiant d'un document, et est toujours contenu dans la propriété `_id`. Lorsque l'on insère un nouveau document dans une collection, MongoDB lui définit automatiquement une propriété `_id` qui va contenir un objet `ObjectId`, lequel représente l'identifiant du document.

C'est un peu l'équivalent d'un auto-incrément en SQL, sauf qu'ici c'est MongoDB qui gère le tout, et que l'identifiant est une chaîne de caractères aléatoire. Nous reviendrons sur les caractéristiques d'`ObjectId` quand nous aborderons les requêtes.

2.2.2 ISODate

Cet objet permet de stocker une date au format ISO. Cela simplifie la manipulation des dates puisque `ISODate` est un conteneur de `Date` et supporte les mêmes méthodes (`getHours()`, `getMinutes()`...) :

```
ISODate("1999-11-10T00:00:00Z").getHours();
ISODate("1999-11-10T00:00:00Z").getMinutes();
```

2.2.3 Nombres

Outre la gestion des ID et des dates ISO, MongoDB permet de gérer “manuellement” certains nombres. Pour ça, les objets `NumberLong` et `NumberInt` définissent respectivement des nombres en 64 bits et en 32 bits. Par défaut, MongoDB stocke les nombres sous la forme de nombres à **virgule flottante**.

[SGBD] : Système de gestion de base de données [NoSQL] : Not Only SQL

3 Installer MongoDB et faire quelques tests

3.1 Installer MongoDB

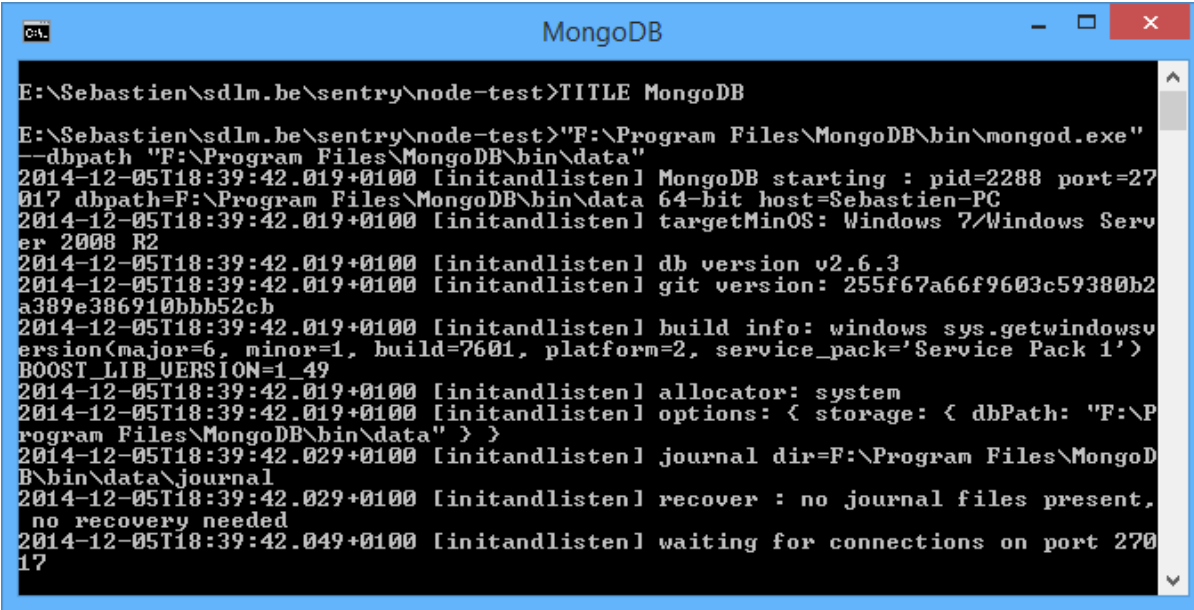
MongoDB est disponible pour Windows, Mac OS, Linux et Solaris. Rendez-vous [sur la page officielle de téléchargement](#) pour télécharger la version qui correspond à votre système d'exploitation. Dans mon cas, il s'agit d'un Windows 64 bits.

Pour démarrer MongoDB, exécutez le fichier *mongod.exe* (Windows) ou *mongod* (Unix), situé dans le dossier *bin* du dossier dans lequel MongoDB est installé. Lors de son lancement, MongoDB recherchera le dossier */data/bd* à la racine de votre disque. Si ce dossier n'existe pas, créez-le.

Vous pouvez évidemment créer le dossier */data/db* ailleurs ; dans ce cas, il faudra spécifier son emplacement en utilisant le paramètre `--dbpath`.

[[information]] | Pour plus de facilité, sous Windows, je vous conseille de vous créer un petit script *.bat*. Créez un nouveau fichier texte (.txt) avec le Bloc-Notes. Dedans, écrivez le chemin vers *mongod.exe*, suivi du paramètre `--dbpath`, comme ceci : `|| bash | TITLE MongoDB | "F:\Program Files\MongoDB\bin\mongod.exe" --dbpath "F:\Program Files\MongoDB\bin\data" |` Enregistrez le fichier, et changez l'extension en *.bat*. Un double clic sur le fichier *.bat* lancera la commande, et donc le démarrage de MongoDB.

Lancez donc MongoDB, et voici ce que vous devriez obtenir. Si la dernière ligne affiche *waiting for connections*, c'est bon, MongoDB fonctionne et attend vos consignes !

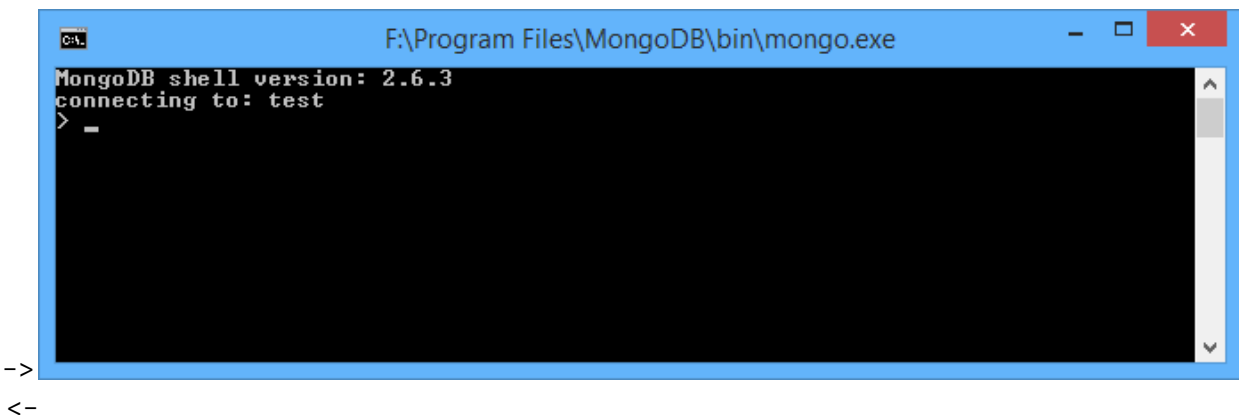


```
E:\Sebastien\sdlm.be\sentry\node-test>TITLE MongoDB
E:\Sebastien\sdlm.be\sentry\node-test>"F:\Program Files\MongoDB\bin\mongod.exe"
--dbpath "F:\Program Files\MongoDB\bin\data"
2014-12-05T18:39:42.019+0100 [initandlisten] MongoDB starting : pid=2288 port=27
017 dbpath=F:\Program Files\MongoDB\bin\data 64-bit host=Sebastien-PC
2014-12-05T18:39:42.019+0100 [initandlisten] targetMinOS: Windows 7/Windows Serv
er 2008 R2
2014-12-05T18:39:42.019+0100 [initandlisten] db version v2.6.3
2014-12-05T18:39:42.019+0100 [initandlisten] git version: 255f67a66f9603c59380b2
a389e386910bbb52cb
2014-12-05T18:39:42.019+0100 [initandlisten] build info: windows sys.getwindowsv
ersion(major=6, minor=1, build=7601, platform=2, service_pack='Service Pack 1')
BOOST_LIB_VERSION=1_49
2014-12-05T18:39:42.019+0100 [initandlisten] allocator: system
2014-12-05T18:39:42.019+0100 [initandlisten] options: { storage: { dbPath: "F:\P
rogram Files\MongoDB\bin\data" } }
2014-12-05T18:39:42.029+0100 [initandlisten] journal dir=F:\Program Files\MongoD
B\bin\data\journal
2014-12-05T18:39:42.029+0100 [initandlisten] recover : no journal files present.
no recovery needed
2014-12-05T18:39:42.049+0100 [initandlisten] waiting for connections on port 270
17
```

[[attention]] | Faites attention, il y a deux exécutables : *mongod.exe* (*mongod* pour Unix) qui démarre le “serveur” MongoDB, et *mongo.exe* (*mongo* pour Unix), que l’on va utiliser maintenant, qui permet de dialoguer avec le serveur. *mongo.exe* (*mongo* pour Unix) est une version console d’un utilitaire tel que PHPMyAdmin (pardonnez-moi pour cette analogie ^^).

3.2 Quelques opérations

Maintenant que MongoDB est lancé, nous allons faire quelques manipulations. Nous allons pour cela ouvrir le shell (la console) nous permettant de commander MongoDB : *mongo.exe* (Windows) ou *mongo* (Unix) :



```
->
<-
```

3.2.1 Lister et choisir la base de données

Par défaut, MongoDB se connecte à la base de données appelée test. La commande `show dbs` permet d’afficher toutes les bases de données gérées par MongoDB. Voici ce que j’obtiens :

```
> show dbs
admin    <empty>
local    0.078GB
test     0.078GB
```

Maintenant que nous connaissons toutes les bases de données accessibles, changeons de base, avec la commande `use <nom-de-la-base>` :

```
> use local
switched to db local
```

[[question]] | Mais comment créer une base de données ?

Pour créer une base de données, par exemple la base *tutoriel*, il suffit de la sélectionner (même si elle n’existe pas) avec `use`, puis d’y insérer des données. Lors de l’insertion, la base sera créée. Voyons donc comment insérer des données, mais auparavant, sélectionnez la base de données *tutoriel* :

```
> use tutoriel
switched to db tutoriel
```

3.2.2 Insérer des documents

Nous sommes connectés à la base tutoriel, donc toutes les opérations que nous allons réaliser ici porteront sur cette base de données-là. Gardez cela en tête !

Pour insérer un document, rien de plus simple :

```
db.personnages.insert( { name : "Gordon Freeman", game : "Half-Life" } );
```

Cette simple commande insérera le document `{ name: "Gordon Freeman", game: "Half-Life" }` au sein de la collection *personnages*. Cette collection n'existe pas, mais sera créée automatiquement.

3.2.3 Faire des recherches

Pour afficher tous les documents d'une collection, il suffit de faire :

```
> db.personnages.find()
```

Cette commande affichera tous les documents de la collection *personnages*.

Des critères peuvent être spécifiés en paramètre de la méthode `find()`. Ces paramètres prennent la forme d'un objet, dont les propriétés à analyser correspondent à celles des documents :

```
> db.personnages.find( { name : "Gordon Freeman" } )
```

Cette commande trouvera donc le document dont la clef `name` vaut *Gordon Freeman*.

4 Se connecter depuis Node.js

4.1 Installer le module

Afin de pouvoir se connecter à une base de données MongoDB depuis Node.js, il convient d'installer le module NPM **mongodb** :

```
npm install mongodb
```

4.2 Connexion à la base de données

La première étape est évidemment de requérir le module `mongodb` :

```
var MongoClient = require("mongodb").MongoClient;
```

[[information]] | `MongoClient` est **similaire** au `MongoClient` que l'on peut trouver pour d'autres langages comme PHP, Python, Ruby...

Une fois que c'est fait, une connexion peut être amorcée grâce à la méthode `connect()` :

```
MongoClient.connect("mongodb://localhost/tutoriel", function(error, db) {  
    if (error) return funcCallback(error);  
  
    console.log("Connecté à la base de données 'tutoriel'");  
});
```

`connect()` reçoit deux paramètres : une URI définissant l'adresse du serveur MongoDB ainsi que la base de données à utiliser (il s'agit ici de la base *tutoriel*), et une fonction de callback (j'utilise ici une fonction anonyme).

La fonction de callback recevra deux paramètres : `error` et `db`. Si `error` est défini, c'est qu'il s'est passé quelque chose empêchant la connexion. `db` est un objet qui représente la base de données, et qui va nous permettre de communiquer avec cette dernière.

*[NPM] : Node Package Manager

5 Opérations depuis Node.js

Maintenant que nous avons établi une connexion avec MongoDB depuis Node.js, voyons comment faire quelques manipulations.

5.1 L'ObjectID

Lorsqu'un document est ajouté à une collection, un identifiant lui est automatiquement attribué. Cet identifiant est stocké dans la propriété `_id` et contient un objet de type `ObjectID`. Ainsi, si on souhaite insérer ce document :

```
{
  name : "Adrian Shephard",
  game : "Half-Life: Opposing Force"
}
```

MongoDB lui définit alors un identifiant :

```
{
  _id : ObjectID("53dfe7bbfd06f94c156ee96e"),
  name : "Adrian Shephard",
  game : "Half-Life: Opposing Force"
}
```

C'est important pour la suite, surtout quand il va s'agir de récupérer le document correspondant à un ID donné !

5.2 Récupérer tous les documents d'une collection

Après s'être connecté, il suffit d'utiliser l'objet `db` pour effectuer une requête. Ici, on va faire une requête sur la collection `personnages` afin de lister tous les documents qui s'y trouvent. Pour ce faire, on utilise la méthode `find()` et, petit bonus, on demande de recevoir les résultats sous la forme d'un tableau, via la méthode `toArray()`. Grâce à ça, il sera aisé de parcourir les résultats :

```
MongoClient.connect("mongodb://localhost/tutoriel", function(error, db) {
  if (error) throw error;

  db.collection("personnages").find().toArray(function (error, results) {
    if (error) throw error;

    results.forEach(function(i, obj) {
      console.log(
```

```

        "ID : " + obj._id.toString() + "\n" // 53dfe7bbfd06f94c156ee96e
        "Nom : " + obj.name + "\n"         // Adrian Shephard
        "Jeu : " + obj.game                 // Half-Life: Opposing Force
    );
  });
});

```

Pour parcourir les résultats, une boucle `for` ou `forEach()` et le tour est joué.

[[information]] | Remarquez que pour récupérer l'identifiant, on accède directement à la propriété `_id`. Mais il faut penser à appliquer la méthode `toString()`, puisqu'il s'agit d'un objet `ObjectID`.

5.3 Récupérer le document correspondant à un identifiant

La liste complète des documents ayant été récupérée, on peut imaginer que votre application permettra ensuite d'afficher les informations relatives à un des documents. Pour ce faire, il faut le récupérer, en utilisant son identifiant.

[[information]] | On peut utiliser la méthode `find()`, mais comme un seul résultat sera retourné, autant privilégier `findOne()` qui ne retourne que le premier résultat.

L'identifiant sera reçu sous la forme d'une chaîne de caractères (imaginons qu'il a été transmis par GET ou POST). Pour l'utiliser au sein de la requête, il va falloir le transformer en une instance d'`ObjectID`. Il faut donc commencer par requérir cet objet ; j'ai choisi de le faire en utilisant :

```
var MongoDBObjectID = require("mongodb").ObjectID;
```

Souvenez-vous, pour définir une requête, il suffit de définir un objet dont les propriétés correspondent à ce que l'on souhaite trouver. Donc, si l'on souhaite rechercher le document dont l'identifiant est `53dfe7bbfd06f94c156ee96e`, on créera :

```
{ _id : new MongoDBObjectID("53dfe7bbfd06f94c156ee96e") }
```

Cet objet sera passé en paramètre de la méthode `findOne()` (ou `find()`). Voici le script complet :

```

var MongoDBObjectID = require("mongodb").ObjectID;           // Il nous faut ObjectID
var idToFind        = "53dfe7bbfd06f94c156ee96e";           // Identifiant, sous forme d
var objToFind        = { _id : new MongoDBObjectID(idToFind) }; // Objet qui va nous servir

db.collection("personnages").findOne(objToFind, function(error, result) {
  if (error) throw error;

  console.log(
    "ID : " + result._id.toString() + "\n" // 53dfe7bbfd06f94c156ee96e
    "Nom : " + result.name + "\n"         // Adrian Shephard
    "Jeu : " + result.game                 // Half-Life: Opposing Force
  );
});

```

5.4 Insérer un document

Insérer un nouveau document se fait avec la méthode `insert()` et requiert un argument : le document à insérer ou un tableau contenant plusieurs documents à insérer. Il est donc possible d'insérer plusieurs documents en une fois. Pratique !

```
var objNew = { name : "GLaDOS", game : "Portal" };

db.collection("personnages").insert(objNew, null, function (error, results) {
    if (error) throw error;

    console.log("Le document a bien été inséré");
});
```

La méthode `insert()` admet 3 paramètres :

1. Le document ou le tableau de documents à insérer ;
2. Un objet contenant les options (optionnel) : [voyez ici pour les options](#) `writeConcern`. Un exemple d'option serait `{ w: 0 }` ;
3. La callback, optionnelle

5.5 Insérer et/ou éditer

La méthode `save()` fonctionne comme `insert()`, à la différence que si `_id` est spécifié et qu'un document contenant cet ID existe dans la base, elle va le mettre à jour. Si la méthode ne trouve pas de document correspondant ou si `_id` n'est pas défini, le document est inséré, comme si `insert()` était utilisée.

```
db.collection("personnages").save(objNew, { w : 1 }); // Ce document sera inséré
```

5.6 Éditer un document

Comme vu précédemment, il suffit d'utiliser `save()`, mais `update()` peut être préférée dans certains cas. En particulier car `update()` autorise divers opérateurs, en particulier `$set`, qui permet de définir si le document trouvé est remplacé par le nouveau document, ou bien si les données du document trouvé sont mises à jour avec les données correspondantes du nouveau document. Et ça change tout !

```
// Exemple 1 : remplacement
db.collection("personnages").update(
    { name : "GladOS" },
    { name : "GladOS", game : "Portal 2" }
);
```

```
// Exemple 2 : mise à jour, via $set
db.collection("personnages").update(
    { name : "GladOS" },
```

```
    { $set : { game : "Portal 2" } }  
  );
```

1. Dans le premier exemple, le document trouvé est complètement remplacé par le nouveau document. C'est donc un remplacement !
2. Mais dans le second exemple, grâce à l'opérateur \$set, seule la propriété game est mise à jour.

5.7 Supprimer un document

Dernier point à voir, `remove()` qui permet de supprimer un document, ou tous les documents d'une collection si aucun argument n'est transmis. L'utilisation de `remove()` est aussi simple que les autres, le premier argument étant le document à supprimer.

[[attention]] | Si le document n'est pas spécifié, TOUS les documents présents dans la collection seront supprimés ! Soyez vigilants !

```
var MongoDBObjectID = require("mongodb").ObjectID;           // Il nous faut ObjectID  
var idToFind         = "53dfe7bbfd06f94c156ee96e";           // Identifiant, sous forme d  
var objToFind        = { _id : new MongoDBObjectID(idToFind) }; // Objet qui va nous servir  
  
db.collection("personnages").remove(objToFind, null, function(error, result) {  
    if (error) throw error;  
});
```

6 Requêtes avancées

Nous n'avons vu que des requêtes simples, comme récupérer un document comportant un identifiant donné. Il est évidemment possible de réaliser des requêtes plus détaillées. On peut même y inclure des expressions régulières !

6.1 Plusieurs critères (AND)

Pour définir plusieurs critères, il suffit juste d'utiliser plusieurs propriétés au sein de l'objet de recherche :

```
{ game : "Half-Life", gender : "M" }
```

6.2 Soit l'un soit l'autre (OR)

La syntaxe est ici quelque peu plus complexe. MongoDB met à disposition diverses propriétés *réservées* qui sont utilisées en tant que paramètres. Pour faire un OR, on utilise la propriété \$or. Cette dernière contient un tableau composé des critères de recherche. L'exemple montre un objet qui va récupérer tous les personnages du jeu Portal, ou, tous les hommes :

```
{
  $or : [
    { game : "Portal" },
    { gender : "M" }
  ]
}
```

6.3 Conditions < <= > >= !=

Des conditions peuvent être ajoutées. Voici les propriétés *réservées* correspondantes :

->

Condition	Propriété
<	\$lt
<=	\$lte
>	\$gt
>=	\$gte
!=	\$ne

<-

Trouve tous les personnages plus jeunes que 40 ans.

```
{ old : { $lt : 40 } }
```

Trouve tous les personnages qui ont entre 18 et 40 ans exclus

```
{ old : { $gt : 18, $lt : 40 } }
```

6.4 Expressions régulières

L'utilisation de regex est autorisée, de façon très simple. La requête suivante trouve tous les personnages de jeux dont le nom commence par *Half-Life*. Les personnages du jeu *Half-Life : Opposing Force* seront donc trouvés aussi :

```
{ game : /^Half-Life/ }
```

7 Conclusion

Voilà, j'espère que ce petit cours vous aura permis d'y voir plus clair !

Bien évidemment, ce tutoriel n'est pas complet et n'a pas pour vocation à remplacer la documentation.

Pour aller plus loin, n'hésitez pas à parcourir les documentations officielles :

1. [La documentation officielle de MongoDB](#) ;
2. [La documentation officielle du driver MongoDB pour Node.js](#).