MONGODB ET NODE.JS

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

**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.

**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*.

[Se connecter depuis Node.js](https://zestedesavoir.com/tutoriels/312/debuter-avec-mongodb-pour-node-js/#3-9640_se-connecter-depuis-nodejs)

**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@2.2.23](mailto:mongodb@2.2.23) (Ca installe la version 2.2.23 de mongodb)

**Connexion à la base de données**

La première étape est évidemment de requérir le module mongodb :

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

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.

[Opérations depuis Node.js](https://zestedesavoir.com/tutoriels/312/debuter-avec-mongodb-pour-node-js/#4-11691_operations-depuis-nodejs)

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

**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 !

**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 utiliser 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**(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.

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.

**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.

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** MongoObjectID = 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** MongoObjectID("53dfe7bbfd06f94c156ee96e") }

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

**var** MongoObjectID = require("mongodb").ObjectID; // Il nous faut ObjectID

**var** idToFind = "53dfe7bbfd06f94c156ee96e"; // Identifiant, sous forme de texte

**var** objToFind = { \_id: **new** MongoObjectID(idToFind) }; // Objet qui va nous servir pour effectuer la recherche

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

);

});

**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) :  *writeConcern*. Un exemple d'option serait { w: 0 } ;
3. La callback, optionnelle

**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

**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.

**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.

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

**var** MongoObjectID = require("mongodb").ObjectID; // Il nous faut ObjectID

**var** idToFind = "53dfe7bbfd06f94c156ee96e"; // Identifiant, sous forme de texte

**var** objToFind = { \_id: **new** MongoObjectID(idToFind) }; // Objet qui va nous servir pour effectuer la recherche

db.collection("personnages").remove(objToFind, **null**, **function**(error, result) {

**if** (error) **throw** error;

});

[Requêtes avancées](https://zestedesavoir.com/tutoriels/312/debuter-avec-mongodb-pour-node-js/#5-11770_requetes-avancees)

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 !

**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" }

**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" }

]

}

**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 }

**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/ }