NodeJS/MongoDB

EVERYTHING IS OBJECT

Summary

[Summary 1](#_Toc493719856)

[Preriquisite 2](#_Toc493719857)

[Install mongoDB on your computer 2](#_Toc493719858)

[Get mongoDB driver for nodeJS 2](#_Toc493719859)

[Run mongoDB 2](#_Toc493719860)

[Create database 2](#_Toc493719861)

[Create a Collection 3](#_Toc493719862)

[Insert document in Collection 4](#_Toc493719863)

[Find data in a Collection 6](#_Toc493719864)

[Find one item 6](#_Toc493719865)

[Find multiple items 6](#_Toc493719866)

[Query object 7](#_Toc493719867)

[Sorting datas 8](#_Toc493719868)

[Update item 8](#_Toc493719869)

[Update one item 8](#_Toc493719870)

[Update multiple items 9](#_Toc493719871)

[Limit the result 10](#_Toc493719872)

[Delete item in a collection 12](#_Toc493719873)

[Delete one item 12](#_Toc493719874)

[Delete multiple items 13](#_Toc493719875)

[Drop a Collection 13](#_Toc493719876)

[Drop 13](#_Toc493719877)

[dropCollection 14](#_Toc493719878)

# Preriquisite

NodeJS need to be installed ( <https://nodejs.org/en/> ).

# Install mongoDB on your computer

Go to <https://www.mongodb.com/> and install mongoDB (.msi file)

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\dalerte-attention-avertissement-icone-8189-32.png** | We are going to go for the local version, so do not choose Atlas version ! |

# Get mongoDB driver for nodeJS

Open a console and install mongoDB driver :

npm install –save mongodb

# Run mongoDB

First, create the following folder :

C:\data\db

Then go to your install folder (probably c:\program\mongodb) and run mongodb :

cd C:\program\mongodb\mongod

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | You have to keep this console window open, don’t touch it anymore, mongo is running and listening. |

# Create database

Let’s build a file called create\_database.js to create our database with the following :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/ToRead";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

console.log("Database created !");

db.close();

});

Then go on your console and run the file :

node create\_database.js

you should see « Database created ! » in the console. If not, then paste your error message in Google then find the answer !

# Create a Collection

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | A Collection is similar to a **record** in SQL |

Let’s create a file called create\_collection.js to create our Collection with the following :

const MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.createCollection("customers", (error, res) => {

if (error) throw error;

console.log("Collection created!");

db.close();

});

});

You should see « Collection created ! » in the console.

# Insert document in Collection

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | A Document is similar to a **record** in SQL |

To insert data, we will use **insertOne()** function. It takes an object as parameter.

const MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

let myobj = { name: "Company Inc", address: "Highway 37" };

db.collection("customers").insertOne(myobj, (error, res) => {

if (error) throw error;

console.log("1 document inserted");

db.close();

});

});

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | If you try to insert data in an inexistant collection, i twill be created automatically |

You can also insert more than one item per call. To do this, just wrap your object inside an array [] and call **insertMany()** function as following :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const myobj = [

{ name: 'John', address: 'Highway 71' },

{ name: 'Vicky', address: 'Yellow Garden 2' },

{ name: 'Ben', address: 'Park Lane 38' },

{ name: 'William', address: 'Central st 954' },

{ name: 'Chuck', address: 'Main Road 989' },

{ name: 'Viola', address: 'Sideway 1633' },

];

db.collection("customers").insertMany(myobj, (error, res) => {

if (error) throw error;

console.log(`Number of documents inserted: ${res.insertedCount}`);

db.close();

});

});

« res » is an object containing all previously inserted datas and few others importants things.

« res » (or whatever is his name, it’s on you after all, it’s just a variable !) looks like this :

{

result: { ok: 1, n: 14 },

ops: [

{ name: 'John', address: 'Highway 71' },

{ name: 'Vicky', address: 'Yellow Garden 2' },

{ name: 'Ben', address: 'Park Lane 38' },

{ name: 'William', address: 'Central st 954' },

{ name: 'Chuck', address: 'Main Road 989' },

{ name: 'Viola', address: 'Sideway 1633' },

],

insertedCount: 14,

insertedIds: [

58fdbf5c0ef8a50b4cdd9a84,

58fdbf5c0ef8a50b4cdd9a85,

58fdbf5c0ef8a50b4cdd9a86,

58fdbf5c0ef8a50b4cdd9a87,

58fdbf5c0ef8a50b4cdd9a88,

58fdbf5c0ef8a50b4cdd9a89,

]

}

You can see there is a key named « **insertedIds** ». Because we did not specify any id in our object before to insert datas, mongoDB automatically create a unique id for each of our items. You can obviously define an id by yourself, but every id need to be unique. Here our « myObj » oject containing our datas with ids :

let myobj = [

{ \_id: 154, name: 'Chocolate Heaven' },

{ \_id: 155, name: 'Tasty Lemon' },

{ \_id: 156, name: 'Vanilla Dream' },

];

With this setup, **insertedIds** will contains you ids.

# Find data in a Collection

## Find one item

To find data in a Collection, we will use the function **findOne()** which will return the first occurrence that match your condition :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.collection("customers").findOne({}, (error, result) => {

if (error) throw error;

console.log(result.name);

db.close();

});

});

## Find multiple items

To find multiple items from your collection, we will use **find()** function which take three parameters before the callback function :

* A query string « {} » which will **filter our result**. If empty, find() method will return every items.
* An optionnal object describing which key must be returned. if ommited, find() function will return items with all their keys.

Here is an example without the second parameter…

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.collection("customers").find({}).toArray((error, result) => {

if (error) throw error;

console.log(result);

db.close();

});

});

and here, another example with the second parameter :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.collection("customers").find({}, { \_id: false }).toArray((error, result) => {

if (error) throw error;

console.log(result);

db.close();

});

});

Objects returned with this configuration will looks like this :

{ name: 'John', address: 'Highway 71'}

There is no « \_id » key inside !

### Query object

About this query string… well it’s just another object to filter your result. It can be filtered by exiting value :

{ address: "Park Lane 38" }

or with regular expression (here, only get adress stating with the letter « S » :

{ address: /^S/ }

Of course, regular expression are **case sensitive**, so filtering with «  /^s/ » (lower case « s ») is different than filtering with « /^S/ » (upper case S).

# Sorting datas

You can sort your result by calling **sort()** function :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const mysort = { name: 1 };

db.collection("customers").find().sort(mysort).toArray((error, result) => {

if (error) throw error;

console.log(result);

db.close();

});

});

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | Here you can see that we are chaining functions. |

It takes an object as parameter, containing the key you want to sort, and the alphabetical order :

* « 1 » for ascending order (from A to Z)
* « -1 » for descending order (from Z to A)

# Update item

## Update one item

You can update an item by calling updateOne() function :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://127.0.0.1:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const myquery = { address: "Valley 345" };

const newvalues = { name: "Mickey", address: "Canyon 123" };

db.collection("customers").updateOne(myquery, newvalues, (error, res) => {

if (error) throw error;

console.log("1 document updated");

db.close();

});

});

updateOne takes two parameters before the callback function :

* A query string {} to find the item you want to update
* An object containing new values for the specified key

You can obviously target only one key to update. To do this, use the **$set** operator in the object of the second parameter :

const newvalues = { $set: { name: "Minnie" } };

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\dalerte-attention-avertissement-icone-8189-32.png** | Without this operator, all other keys that are **not specified** in the newValues object ill be **erased** ! |

## Update multiple items

As always, we can target multiple items to be updated thanks to the **updateMany()** function :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://127.0.0.1:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const myquery = { address: /^S/ };

const newvalues = { $set: { name: "Minnie" } };

db.collection("customers").updateMany(myquery, newvalues, (error, res) => {

if (error) throw error;

console.log(`${res.result.nModified} document(s) updated`);

db.close();

});

});

And again, we can use, or not, regular expression.

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | Both updateOne() and updateMany() functions will return an object describing how the execution affected the database. Important informations are inside result sub array (execution ok or not, number of change, …). |

# Limit the result

To limit the result returned, we can use **limit()** function :

var MongoClient = require('mongodb').MongoClient;

var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {

if (err) throw err;

db.collection("customers").find().limit(5).toArray(function(err, result) {

if (err) throw err;

console.log(result);

db.close();

});

});

It takes as parameter the number of items you want to get from the database.

Join Collections

Even if mongoDB is not a relational database, you can perform a **left outer join** with the **$lookup** operator.

Consider you have two Collection : Products and Orders :

**Products :**

[

{ \_id: 1, product\_id: 154, status: 1 }

]

**Orders :**

[

{ \_id: 154, name: 'Chocolate Heaven' },

{ \_id: 155, name: 'Tasty Lemons' },

{ \_id: 156, name: 'Vanilla Dreams' }

]

You can request both in once by joining them as following :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://127.0.0.1:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.collection('orders').aggregate([

{ $lookup:

{

from: 'products',

localField: 'product\_id',

foreignField: '\_id',

as: 'orderdetails',

},

},

], (error, res) => {

if (error) throw error;

console.log(JSON.stringify(res));

db.close();

});

});

$lookup is an object containing several keys :

* « from » : The collection source of the join. This collection will contains the aggregate
* « localField » : The key in the source collection that refer to the destination collection
* « foreignField » : The key in the destination collection that link our two collections
* « as » : A name of a key that will be added inside of the source collection (only in the result returned, not in the database of course).

So the result will be :

[

{ "\_id": 1, "product\_id": 154, "status": 1, "orderdetails": [

{ "\_id": 154, "name": "Chocolate Heaven" } ]

}

]

You can see « orderdetails » which is our key defined by « as » in the $lookup object, and it contain an array of all matched items.

# Delete item in a collection

## Delete one item

To delete an item, use the function **deleteOne()**.

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const myquery = { address: 'Mountain 21' };

db.collection("customers").deleteOne(myquery, (error, obj) => {

if (error) throw error;

console.log("1 document deleted", obj);

db.close();

});

});

## Delete multiple items

As you can do with insert, you can delete several items at the same times with **deleteMany()** function :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

const myquery = { address: /^O/ };

db.collection("customers").deleteMany(myquery, (error, obj) => {

if (error) throw error;

console.log(`${obj.result.n} document(s) deleted`);

db.close();

});

});

Once again, we use regular expression to target multiple items in our Collection.

# Drop a Collection

## Drop

Drop a collection with the **drop()** function :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.collection("customers").drop((error, delOK) => {

if (error) throw error;

if (delOK) console.log("Collection deleted");

db.close();

});

});

## dropCollection

There is another function available for this purpose : dropCollection(). Instead of calling another function to drop your collection, you can call dropFunction. It will takes the name of the collection as first parameter :

let MongoClient = require('mongodb').MongoClient;

const url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, (err, db) => {

if (err) throw err;

db.dropCollection("customers", (error, delOK) => {

if (error) throw error;

if (delOK) console.log("Collection deleted");

db.close();

});

});

|  |  |
| --- | --- |
| **C:\Users\Kotin\Downloads\info-icone-6785-32.png** | A quick search did not give me information about the difference between **drop()** and **dropCollection()** so feel free to do your own research about it.  **Both are working**. |