Node.js + MongoDB

Ashok Dudhat
Germany Startup Jobs
www.germanystartupjobs.com

Installation

Install MongoDB Driver

$ npm install mongodb

Import require package

Node.js can use this module to manipulate MongoDB databases:

```
var mongo = require('mongodb');
```

Creating a Database

To create a database in MongoDB, start by creating a **MongoClient** object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/testdb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

Creating a Collection

To create a collection in MongoDB, use the **`createCollection()`** method:

Create a collection called "**customers**":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Important: A **collection** in MongoDB is the same as a **table** in MySQL. In MongoDB, a collection is not created until it gets content! MongoDB waits until you have inserted a document before it actually creates the collection.

Insert Document : Insert One

To insert a record, or *document* as it is called in MongoDB, into a collection, we use the **`insertOne()`** method.

The first parameter of the **`insertOne()`** method is an object containing the name(s) and value(s) of each field in the document you want to insert. It also takes a callback function where you can work with any errors, or the result of the insertion:

Insert a document in the "**customers**" collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Important: A **document** in MongoDB is the same as a **record** in MySQL. If you try to insert documents in a collection that do not exist, MongoDB will create the collection automatically.

Insert Document : Insert Many

To insert multiple documents into a collection in MongoDB, we use the **`insertMany()`** method.

The first parameter of the **`insertMany()`** method is an **array of objects**, containing the data you want to insert.

Insert multiple documents in the "**customers**" collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myobj = [
    { name: 'John', address: 'Highway 71'},
    { name: 'Peter', address: 'Lowstreet 4'},
    { name: 'Amy', address: 'Apple st 652'},
    { name: 'Hannah', address: 'Mountain 21'},
    { name: 'Michael', address: 'Valley 345'},
    { name: 'Sandy', address: 'Ocean blvd 2'},
    { name: 'Betty', address: 'Green Grass 1'},
    { name: 'Richard', address: 'Sky st 331'},
    { name: 'Susan', address: 'One way 98'},
    { 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'}
  ];
  dbo.collection("customers").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("Number of documents inserted: " + res.insertedCount);
    db.close();
  });
});
```

The _id Field

If you do not specify an **_id** field, then MongoDB will add one for you and assign a unique id for each document.

In the example above no **_id** field was specified, and as you can see from the result object, MongoDB assigned a **unique** _id for each document.

If you *do* specify the **_id** field, the value must be unique for each document:

Insert three records in a "**products**" table, with specified `_id` fields:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myobj = [
    { _id: 154, name: 'Chocolate Heaven'},
    { _id: 155, name: 'Tasty Lemon'},
    { _id: 156, name: 'Vanilla Dream'}
  ];
  dbo.collection("products").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log(res);
    db.close();
  });
});
```

When executing the **insertMany()** method, a result object is returned. The result object contains information about how the insertion affected the database.

Find

In MongoDB we use the **find** and **findOne** methods to find data in a collection.

Just like the **SELECT** statement is used to find data in a table in a MySQL database.

Find Data : Find One

To select data from a collection in MongoDB, we can use the **`findOne()`** method.
The **`findOne()`** method returns the first occurrence in the selection.

The first parameter of the **`findOne()`** method is a query object. In this example we use an empty query object, which selects all documents in a collection (but returns only the first document).

Find the first document in the **customers** collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.collection("customers").findOne({}, function(err, result) {
    if (err) throw err;
    console.log(result.name);
    db.close();
  });
});
```

Find Data : Find All

To select data from a table in MongoDB, we can also use the `find()` method.
The `find()` method returns all occurrences in the selection.

The first parameter of the `find()` method is a query object. In this example we use an empty query object, which selects all documents in the collection.

No parameters in the find() method gives you the same result as **SELECT *** in MySQL.

Find all documents in the **customers** collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.collection("customers").find({}).toArray(function(err, result)
{
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Find Data : Find Some

The second parameter of the `find()` method is an object describing which fields to include in the result.

This parameter is optional, and if omitted, all fields will be included in the result.

Return the fields "**name**" and "**address**" of all documents in the **customers** collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.collection("customers").find({}, { projection: { _id: 0, name: 1, address:
1 }}).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the _id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa: Set only { address: 0 } and check the result.

To exclude the _id field, you must set its value to 0: Set the { _id: 0, name: 1 } and check the result.

Also please check the result by set { _id: 0 } and { name: 1, address: 0 }.

Filter the Result

When finding documents in a collection, you can filter the result by using a **query object**.
The first argument of the `find()` method is a **query object**, and is used to limit the search.

Find documents with the **address** "**Park Lane 38**”:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true},function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var query = { address: "Park Lane 38" };
  dbo.collection("customers").find(query).toArray(function(err,
result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Filter With Regular Expressions

You can write regular expressions to find exactly what you are searching for.

Regular expressions can only be used to query *strings*.

To find only the documents where the **"address" field starts with the letter "S"**, use the **regular expression** `/^s/`:

Find documents where the **address starts** with the letter "**S**":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var searchStr = "S";
  var query = { address: new RegExp('^'+ searchStr, i) };
  dbo.collection("customers").find(query).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Sort the Result

Use the `sort()` method to sort the result in **ascending or descending order**.

The `sort()` method takes one parameter, an object defining the sorting order.

Sort the result alphabetically by **name**:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var mysort = { name: 1 };
  dbo.collection("customers").find().sort(mysort).toArray(function(err,
result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Also Check, Using the value -1 in the sort object to sort descending.
{ name: 1 } // ascending
{ name: -1 } // descending

Delete Document : Delete One

To delete a record, or document as it is called in MongoDB, we use the **deleteOne()** method.

The first parameter of the **deleteOne()** method is a query object defining which document to delete.

Note: If the query finds more than one document, only the first occurrence is deleted.

Delete the document with the address "Mountain 21":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true},function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myquery = { address: 'Mountain 21' };
  dbo.collection("customers").deleteOne(myquery, function(err, obj) {
    if (err) throw err;
    console.log("1 document deleted");
    db.close();
  });
});
```

Delete Document : Delete Many

To delete more than one document, use the deleteMany() method.

The first parameter of the deleteMany() method is a query object defining which documents to delete.

Delete all documents were the address starts with the letter "O":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, { useNewUrlParser:true},function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myquery = { address: /^O/ };
  dbo.collection("customers").deleteMany(myquery, function(err, obj) {
    if (err) throw err;
    console.log(obj.result.n + " document(s) deleted");
    db.close();
  });
})
```

The Result Object

The `deleteMany()` method returns an object which contains information about how the execution affected the database.

Most of the information is not important to understand, but one object inside the object is called "result" which tells us if the execution went OK, and how many documents were affected.

The result object looks like this:

```
{ n: 2, ok: 1 }
```

Update Document: Update One

You can update a record, or document as it is called in MongoDB, by using the **updateOne()** method.

The first parameter of the **updateOne()** method is a query object defining which document to update.

Note: If the query finds more than one record, only the first occurrence is updated.

The second parameter is an object defining the new values of the document.

Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });
});
```

When using the $set operator, only the specified fields are updated:

Update Document: Update Many

To update all documents that meets the criteria of the query, use the **updateMany()** method.

Update all documents where the name starts with the letter "**S**":

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, { useNewUrlParser:true},function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  var myquery = { address: /^S/ };
  var newvalues = {$set: {name: "Minnie"} };
  dbo.collection("customers").updateMany(myquery, newvalues,
function(err, res) {
    if (err) throw err;
    console.log(res.result.nModified + " document(s) updated");
    db.close();
  });
});
```

The Result Object

The `updateOne()` and the `updateMany()` methods return an object which contains information about how the execution affected the database.

Most of the information is not important to understand, but one object inside the object is called "result" which tells us if the execution went OK, and how many documents were affected.

The result object looks like this:

```
{ n: 1, nModified: 2, ok: 1 }
```

Limit the Result

To limit the result in MongoDB, we use the **limit()** method.

The **limit()** method takes one parameter, a number defining how many documents to return.

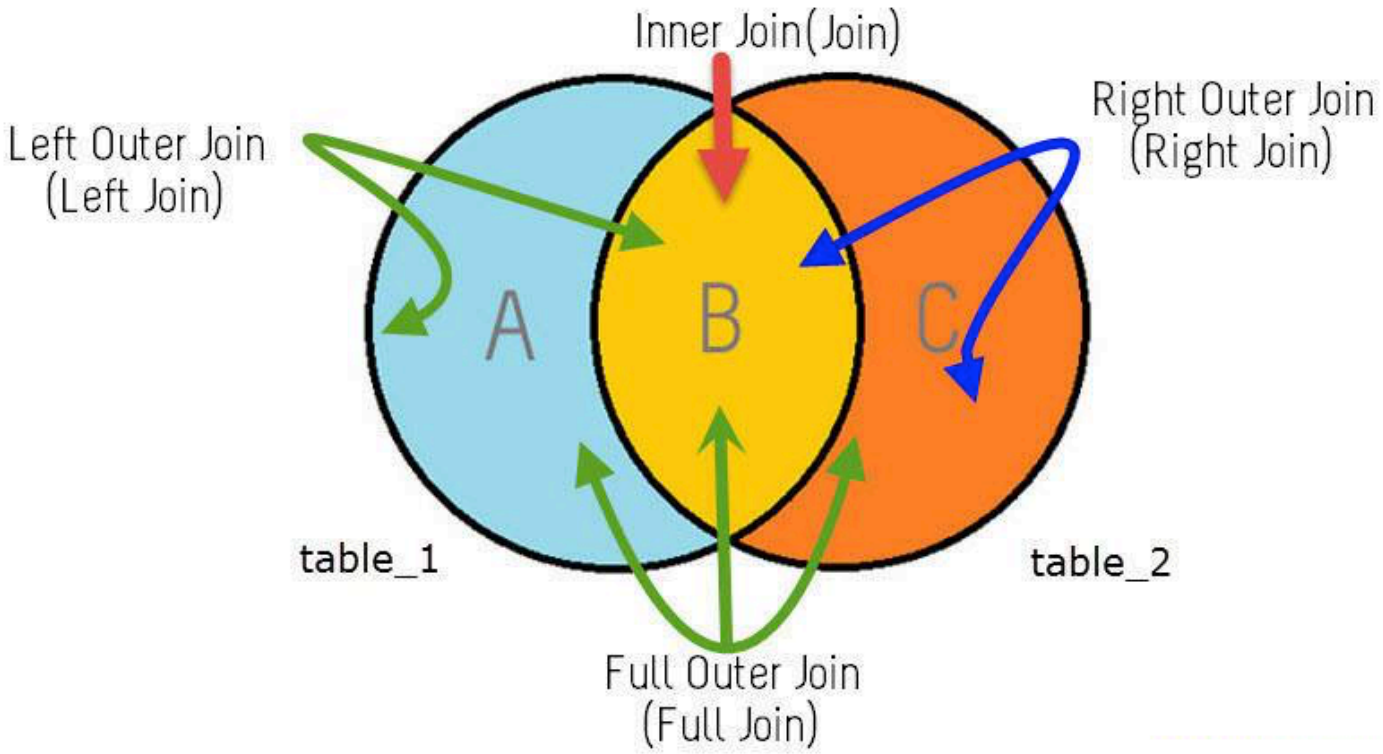
Limit the result to only return **5 documents**:

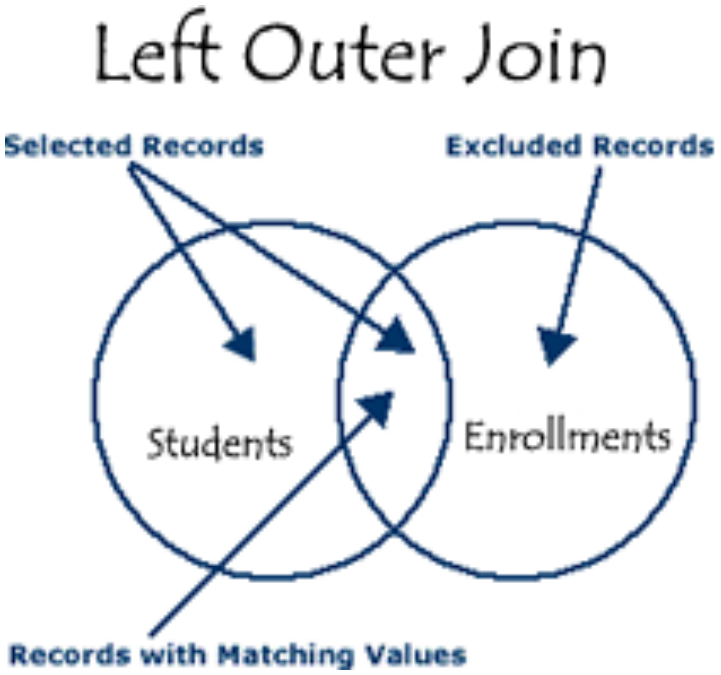
```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err, db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.collection("customers").find().limit(5).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
})
```

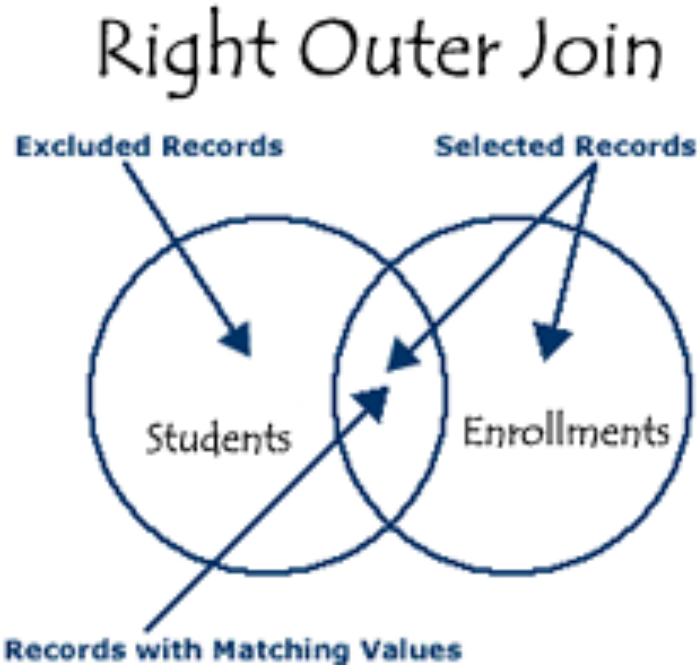
Join Collections

MongoDB is **not a relational database**, but you can perform a **left outer join** by using the `$lookup` stage.

Joins


Inner Join(Join)
Left Outer Join
(Left Join)
Right Outer Join
(Right Join)
A
B
C
table_1
table_2
Full Outer Join
(Full Join)


Left Outer Join
Selected Records
Excluded Records
Students
Enrollments
Records with Matching Values


Right Outer Join
Excluded Records
Selected Records
Students
Enrollments
Records with Matching Values

Join Collections

The `$lookup` stage lets you specify which collection you want to join with the current collection, and which fields that should match.

Consider you have a "**orders**" collection and a "**products**" collection:

orders :

```
[
  { _id: 1, product_id: 154, status: 1 }
]
```

products :

```
[
  { _id: 154, name: 'Chocolate Heaven' },
  { _id: 155, name: 'Tasty Lemons' },
  { _id: 156, name: 'Vanilla Dreams' }
]
```

Join the matching "**products**" document(s) to the "**orders**" collection:

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url,{ useNewUrlParser:true}, function(err,
db) {
  if (err) throw err;
  var dbo = db.db("testdb");
  dbo.collection('orders').aggregate([
    { $lookup:
       {
         from: 'products',
         localField: 'product_id',
         foreignField: '_id',
         as: 'orderdetails'
       }
     }
    ]).toArray(function(err, res) {
    if (err) throw err;
    console.log(JSON.stringify(res));
    db.close();
  });
});
```