Node.js MongoDB

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB

To be able to experiment with the code examples, you will need access to a MongoDB database.

You can download a free MongoDB database at https://www.mongodb.com.

Or get started right away with a MongoDB cloud service at https://www.mongodb.com/cloud/atlas.

Install MongoDB Driver

Let us try to access a MongoDB database with Node.js.

To download and install the official MongoDB driver, open the Command Terminal and execute the following:

Download and install mongodb package:

```
C:\Users\Your Name>npm install mongodb
```

Now you have downloaded and installed a mongodb database driver.

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.

Example

Create a database called "mydb":

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

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:

Example

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("mydb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Insert Into Collection

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

A **document** in MongoDB is the same as a **record** in MySQL

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:

Example

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("mydb");
  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();
  });
});
```

Insert Multiple Documents

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.

It also takes a callback function where you can work with any errors, or the result of the insertion:

Example

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("mydb");
  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 Result Object

When executing the `insertMany()` method, a result object is returned.

The result object contains information about how the insertion affected the database.

The object returned from the example above looked like this:

```
{
  result: { ok: 1, n: 14 },
  ops: [
    { name: 'John', address: 'Highway 71', _id:
58fdbf5c0ef8a50b4cdd9a84 },
    { name: 'Peter', address: 'Lowstreet 4', _id:
58fdbf5c0ef8a50b4cdd9a85 },
    { name: 'Amy', address: 'Apple st 652', _id:
58fdbf5c0ef8a50b4cdd9a86 },
    { name: 'Hannah', address: 'Mountain 21', _id:
```

```
58fdbf5c0ef8a50b4cdd9a87 },
    { name: 'Michael', address: 'Valley 345', _id:
58fdbf5c0ef8a50b4cdd9a88 },
    { name: 'Sandy', address: 'Ocean blvd 2', _id:
58fdbf5c0ef8a50b4cdd9a89 },
    { name: 'Betty', address: 'Green Grass 1', _id:
58fdbf5c0ef8a50b4cdd9a8a },
    { name: 'Richard', address: 'Sky st 331', _id:
58fdbf5c0ef8a50b4cdd9a8b },
    { name: 'Susan', address: 'One way 98', _id:
58fdbf5c0ef8a50b4cdd9a8c },
    { name: 'Vicky', address: 'Yellow Garden 2', _id:
58fdbf5c0ef8a50b4cdd9a8d },
    { name: 'Ben', address: 'Park Lane 38', _id:
58fdbf5c0ef8a50b4cdd9a8e },
    { name: 'William', address: 'Central st 954', _id:
58fdbf5c0ef8a50b4cdd9a8f },
    { name: 'Chuck', address: 'Main Road 989', _id:
58fdbf5c0ef8a50b4cdd9a90 },
    { name: 'Viola', address: 'Sideway 1633', _id:
58fdbf5c0ef8a50b4cdd9a91 } ],
  insertedCount: 14,
  insertedIds: [
    58fdbf5c0ef8a50b4cdd9a84,
    58fdbf5c0ef8a50b4cdd9a85,
    58fdbf5c0ef8a50b4cdd9a86,
    58fdbf5c0ef8a50b4cdd9a87,
    58fdbf5c0ef8a50b4cdd9a88,
    58fdbf5c0ef8a50b4cdd9a89,
    58fdbf5c0ef8a50b4cdd9a8a,
    58fdbf5c0ef8a50b4cdd9a8b,
    58fdbf5c0ef8a50b4cdd9a8c,
    58fdbf5c0ef8a50b4cdd9a8d,
    58fdbf5c0ef8a50b4cdd9a8e,
    58fdbf5c0ef8a50b4cdd9a8f
    58fdbf5c0ef8a50b4cdd9a90,
    58fdbf5c0ef8a50b4cdd9a91 ]
}
```

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:

Example

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("mydb");
  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();
  });
});
```

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).

Example

Find the first 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("mydb");
  dbo.collection("customers").findOne({}, function(err, result) {
    if (err) throw err;
    console.log(result.name);
    db.close();
  });
});
```

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.

Example

Find all 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("mydb");
  dbo.collection("customers").find({}).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Find Some

The second parameter of the `find()` method is the `projection` object that describes which fields to include in the result.

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

Example

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

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.

Example

Find documents with the address "Park Lane 38":

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

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var query = { address: "Park Lane 38" };
  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.

Example

Sort the result alphabetically by name:

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

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

Sort Descending

Use the value -1 in the sort object to sort descending.

```
{ name: 1 } // ascending
{ name: -1 } // descending
```

Delete Document

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.

Example

Delete the document with the address "Mountain 21":

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

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  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 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.

Example

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

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

```
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  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();
  });
});
```

Drop Collection

You can delete a table, or collection as it is called in MongoDB, by using the `drop()` method.

The `drop()` method takes a callback function containing the error object and the result parameter which returns true if the collection was dropped successfully, otherwise it returns false.

Example

Delete the "customers" table:

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

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").drop(function(err, delOK) {
    if (err) throw err;
    if (delOK) console.log("Collection deleted");
    db.close();
  });
});
```

db.dropCollection

You can also use the `dropCollection()` method to delete a table (collection).

The `dropCollection()` method takes two parameters: the name of the collection and a callback function.

Example

Delete the "customers" collection, using dropCollection():

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

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.dropCollection("customers", function(err, delOK) {
    if (err) throw err;
    if (delOK) console.log("Collection deleted");
    db.close();
  });
});
```

Update Document

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.

Example

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

Update Many Documents

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

Example

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, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  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();
  });
});
```