**MongoDB**

MongoDB is a cross-platform document-oriented database program. It is a NoSQL database program which uses JSON-like documents with schema. A NoSQL database is a non-relational database.

Mongo Compass

Similar to phpMyAdmin for MySQL, mongo compass is a GUI for mongoDB administration. It is an alternative to using the terminal.

Shell

|  |  |
| --- | --- |
| **Command** | **Description** |
| Mongo | Enter mongo shell |
| Show dbs | Show existing database |
| Use *dbname* | Select/create a database |
| Show collections | Show collections (like tables), of selected database |
| Db.dropDatabase() | Delete the currently selected database |
| Db.createCollection(‘name’) | Create a collection in the selected database, pass the name of the collection as the argument |
| Db.*collectionName.*insert() | Insert a document in the specified collection, pass an object as the argument with key/value pairs to reference columns and values |
| Db.*collectionName.*insertMany() | Insert many rows, argument should be an array of objects |
| Db.*collectionName.*find() | Display all records in specified collection if no argument passed, or display records based in criteria passed to argument, e.g. find({name: ‘Ali’}) |
| Db.*collectionName.*find().pretty() | Display documents in shell in pretty format |
| Db.*collectionName.*find().sort({*field*: 1}) | Sort data alphabetically: 1 = asc, -1 = desc |
| Db.*collectionName.*count() | Count documents in a collection. Count takes an argument to count records matching criteria |
| Db.*collectionName.*find().limit(*n*) | Limit number of records |
| Db.*collectionName.*findOne() | Return first document matching argument |
| Db.*collectionName.*update() | Arg1 = document to update, e.g. {id: 1},  Arg2 = new data to override the document |
| Db.*collectionName.*remove() | Remove document/s from collection based on arg passed to remove method |

Foreach

Db.*collectionName.*find().forEach(function(document) {

Print(‘this person’s name is ‘ + document.name)

})

Updating documents

Instead of overwriting the full document when using the update method, you can just update the field/s you want to change using the $set operator:

Db.*collectionName.*update({id: 1}, {

$set: {

*fieldToChange*: *value*

}

})

Operators

*Query operators*

|  |  |  |
| --- | --- | --- |
| $eq | $gt | $gte |
| $in (matches vals in array) | $lt | $lte |
| $ne (not equal) | $and {$and: [{}, {}, …]} | $not |
| $or |  |  |

*Update operators*

|  |  |  |
| --- | --- | --- |
| $set | $inc (increment) | $min |
| $max | $rename (change field name) | $unset (remove field) |

Indexes

Indexes are special data structures which allow for more efficient structures by mongoDB and should be used on fields which you will commonly reference (e.g. for selecting, updating…).

By default, the \_id field is already indexed.

Db.*collectionName.*createIndex({*fieldname*: ‘*indexType’*})

Db.posts.createIndex({title: ‘text’})

Index Types:

* Text
* Hashed
* Multikey
* Geospatial
* Compound

Mongoose

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more.

*Npm install mongoose*

*App.js*

Const mongoose = require(‘mongoose’);

Mongoose.connect(‘mongodb://localhost/*dbName*’, {useNewUrlParser: true});

Const db = mongoose.connection;

Db.on(‘error’, console.error.bind(console, ‘connection error’);

Db.once(‘open’, () => console.log(‘connected to local mongoDB’);

// schema

Var catSchema = new mongoose.Schema({

Name: String

})

// compile schema into a model

Var Cat = mongoose.model(‘cat’, catSchema);

// A model is a class with which we construct documents

Var patchy = new Cat({name: ‘Patchy’});

// save document to mongoDB

Patchy.save((err, patchy) => {

If (err) return console.log(err);

Console.log(‘cat with name ‘ + patchy.name + ‘ created.’);

});

Example

*App.js*

Const express = require(‘express’);

Const path = require(‘path’);

Const mongoose = require(‘mongoose’);

Mongoose.connect(‘mongodb://localhost/blog’);

Const db = mongoose.connection;

Db.on(‘error’, (err) => console.error.bind(console, err));

Db.once(‘open’, () => console.log(‘connected to mongoDB’));

Const app = express();

// set up template engine and location

App.set(‘views’, path.join(\_\_dirname, ‘views’);

App.set(‘view engine’, ‘pug’);

Const Article = require(‘./models/article’);

App.get(‘/’, (req, res) => {

Article.find({}, (err, articles) => {

Res.sender(‘index’, { // send data to index.pug view

Title: ‘articles’,

articles

})

})

})

App.listen(5000, () => console.log(‘listening on port 5000’));

*Models > articles.js*

Const mongoose = require(‘mongoose’);

// article schema

Const articleSchema = new mongoose.Schema({

Title: {

Type: String,

Required: true

},

Author: {

Type: String,

Required: true

},

Body: {

Type: String,

Required: true

}

});

Const Article = module.exports = mongoose.model(‘Article’, articleSchema);

**TEST**

1. Create, edit, delete, view db/collections in mongo compass GUI
2. Create, edit, delete, view db/collections/documents in mongo shell
3. Find documents in a collection by an argument
4. Sort collection in alphabetically desc order of a certain field
5. Count number of rows in a collection
6. Update one field of a document
7. Show all collections belonging to a database
8. Use foreach in an example
9. Give a field a text index
10. Create an express server with mongoose, and connect local mongoDB database
11. Create a schema on an express server
12. Save a document in a collection and check mongo shell to see if it inserted
13. Display records in the browser from express server