

Mongo DB and Mongoose

MONGODB ATLAS

DEPLOY A FULLY MANAGED CLOUD DATABASE IN MINUTES

My Cluster

4 DBS 3 COLLECTIONS

filter

- admin
- config
- diabudy
- courses**
- readings
- local

localhost:27017 STANDALONE

diabudy.courses

DOCUMENTS **3** TOTAL SIZE 458B AVG. SIZE 153B

INDEXES **1** TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

- Documents**
- Aggregations
- Schema
- Explain Plan
- Indexes
- Validation

FILTER ▶ OPTIONS **FIND** RESET ...

INSERT DOCUMENT VIEW LIST TABLE Displaying documents 1 - 3 of 3 < > C

```

  > _id: ObjectId("5c1f0fab0e7fd735dc29bb31")
  > tags: Array
  name: "Web Technologies"
  author: "Usman Akram"
  isPublished: true
  date: 2018-12-23 04:31:39.174
  __v: 0

  > _id: ObjectId("5c1f101cad41f2357072e895")
  > tags: Array
  name: "Web Technologies Lab"
  author: "Usman Akram"
  isPublished: false
  date: 2018-12-23 04:33:32.951
  __v: 0

  > _id: ObjectId("5c1f1951615878336c830aac")
  > tags: Array
  name: "OS"
  author: "Usman Akram"
  isPublished: true
  date: 2018-12-23 05:12:49.167
  __v: 0

```



NoSQL

_id

:5c1f0fab0e7fd735dc29bb31

tags

:Array

0

:"laravel"

1

:"React"

2

:"Node"

name

:"Web Technologies"

author

:"Usman Akram"

isPublished

:true

date

:2018-12-23 04:31:39.174

__v

:0

Connecting to Mongo use mongoose

```
const mongoose = require('mongoose');
mongoose.connect("mongodb://localhost/diabudy",
{ useNewUrlParser: true })
.then(() => console.log("Connected to Mongo
..."))
.catch((error) => console.log(error.message));
```

Schema

```
const courseSchema = mongoose.Schema({  
  name: String,  
  author: String,  
  tags: [String],  
  date: { type: Date, default: Date.now },  
  isPublished: Boolean  
});
```

Schema Types

String

Number

Date

Buffer

Boolean

Mixed

ObjectId

Array

Decimal128

Map

Schema Type Options

```
var schema2 = new Schema({  
  test: {  
    type: String,  
    lowercase: true // Always convert `test` to lowercase  
  }  
});
```

Indexes

```
var schema2 = new Schema({
  test: {
    type: String,
    index: true,
    unique: true // Unique index. If you specify `unique: true`
    // specifying `index: true` is optional if you do `unique: true`
  }
});
```


Define Model

```
const Course = mongoose.model("Course",  
courseSchema);
```

Create Course

```
async function createCourse() {  
  const course = new Course({  
    name: "OS",  
    author: "Usman Akram",  
    tags: ["React", "Node"],  
    isPublished: true  
  });  
  const result = await course.save();  
  console.log(result);  
}
```

Get Courses

```
async function getCourses() {  
  const courses = await Course.find({  
    isPublished: true })  
  .limit(10).sort({ name: 1 })  
  .select({ name: 1, tags: 1 });  
  console.log(courses);  
}
```

Comparison Operators

// eq (equal)

//neq (not equal)

//gt (greater than)

//gte (greater than or equal)

//lt (less than)

//lte (less than or equal)

//in

//nin (not in)

Course with price comparison

```
async function getCourses() {  
  const courses = await Course.  
  find({ price: { $gte: 10 } })  
  .limit(10).sort({ name: 1 }).select({ name:  
1, tags: 1 });  
  console.log(courses);  
}
```

Logical Comparison

```
const courses = await Course
  // .find({ author: 'Mosh', isPublished: true })
  .find()
  .or([ { author: 'Mosh' }, { isPublished: true } ])
  .limit(10)
  .sort({ name: 1 })
  .select({ name: 1, tags: 1 });
```

Regex

```
const courses = await Course
  // .find({ author: 'Mosh', isPublished: true })

  // Starts with Mosh
  .find({ author: /^Mosh/ })
  .limit(10)
  .sort({ name: 1 })
  .select({ name: 1, tags: 1 });
console.log(courses);
```

Count

```
const courses = await Course
  .find({ author: 'Mosh', isPublished: true })
  .limit(10)
  .sort({ name: 1 })
  .count();
console.log(courses);
```


api/courses?pagenumber=2&pagesize=10

```
async function getCourses() {
  const pageNumber = 2; const pageSize=10;
  const courses = await Course.
  find({ price: { $gte: 10 } })
  .skip((pageNumber-1)*pageSize)
  .limit(pageSize)
  .sort({ name: 1 }).select({ name: 1, tags: 1 });
  console.log(courses);
}
```

Exercise

Download Files from

<https://1drv.ms/f/s!AtGKdbMmNBGd0WY-xsTp6iY-9AS4>

Run from the folder

```
mongoimport --db mongo-exercises --collection courses --  
drop --file exercise-data.json --jsonArray
```

Get All the public backend courses sort them by their names
and pick only their names.

Updating

```
async function updateCourse(id){  
  //Approach: Query First  
  //findById(id)  
  //Modify  
  //save()  
  
  //Approach Update First  
  //Update Directly  
  //Optionally return updated Doc  
}
```

Query First Update

```
async function updateCourse(id) {  
  const course = await Course.findById(id);  
  if (!course) return;  
  course.isPublished = false;  
  //course.set({isPublished:false});  
  const result = await course.save();  
}
```

Update First

```
async function updateCourse1(id) {  
  const result = await Course.update(  
    { _id, id }, //selection  
    {  
      $set: { //update operations  
        isPublished: false  
      }  
    })  
}
```

Update First... Or

```
async function updateCourse1(id) {
  const result = await Course.findByIdAndUpdate(
    id, //id
    {
      $set: { //update operations
        isPublished: false
      }
    })
} // pass {new:true} to get updated Doc
```

Delete

```
const result = await Course.deleteOne({ _id: id });
```

```
const result = await Course.deleteMany({ _id: id });
```

```
const course = await Course.findByIdAndRemove(id);
```

Recap

MongoDB is an open-source document database.

It stores data in flexible, JSONlike documents. –

In relational databases we have **tables** and **rows**, in MongoDB we have **collections** and **documents**.

A document can contain sub-documents. - We don't have relationships between documents.

Validation

MongoDB

By Default every column or property is optional

You can do like this

```
const courseSchema = mongoose.Schema({  
  name: {type:String,required:true},  
});
```

```
//Promise will be rejected if name is not provided
```

Use try catch

```
try {  
  const result = await course.save();  
  console.log(result);  
} catch (err) {  
  console.log(err.message);  
}  
  
//try surround plugin of VS Code for //quick  
surrounding with try catch
```

Why Not use JOI

Use both

Mongoose Validation for DB

JOI for RESTFUL API

Mongoose Validation

```
const courseSchema = new mongoose.Schema({
  name: { type: String, required: true },
  author: String,
  tags: [ String ],
  date: { type: Date, default: Date.now },
  isPublished: Boolean,
  price: {
    type: Number,
    required: function() { return this.isPublished; }
  }
});
```

Built in Validators for Mongoose

```
eggs: {  
  type: Number,  
  min: [6, 'Too few eggs'],  
  max: 12  
},
```

Built in Validators for Mongoose

```
bacon: {  
  type: Number,  
  required: [true, 'Why no bacon?']  
},
```

Built in Validators for Mongoose

```
drink: {  
  type: String,  
  enum: ['Coffee', 'Tea'],  
  required: function() {  
    return this.bacon > 3;  
  }  
}
```


Custom Validator

```
phone: {  
  type: String,  
  validate: {  
    validator: function(v) {  
      return /\d{3}-\d{3}-\d{4}/.test(v);  
    },  
    message: props => ` ${props.value} is not a valid phone number!`  
  },  
  required: [true, 'User phone number required']  
}
```

For multiple Errors

```
try {  
  const result = await course.save();  
  console.log(result);  
}  
catch (ex) {  
  for (field in ex.errors)  
    console.log(ex.errors[field].message);  
}
```

Practice Project

https://1drv.ms/f/s!AtGKdbMmNBGd0Wobev4gA_rsjbQM

There are two folder

Before – Data is saved in an array

After – Data is handled in mongo db

Practice Project

Browse to folder vidly inside before and run

```
npm install
```

```
node index.js
```

Another Example Project (Customers API)

<https://1drv.ms/f/s!AtGKdbMmNBGd0XmKm8lAbQp58sd2>

Installation Instructions are the same

Final Restructured Solution

https://1drv.ms/f/s!AtGKdbMmNBGd0hFZG5QQ7v8LM_5I

Relationships

MONGOOSE

Using References (Normalization)

```
let author = {  
  name: "Usman Akram"  
}  
  
let course = {  
  title: 'Web Technologies',  
  author: 'id'  
}
```

Using Embedded Documents (Denormalization)

```
let course1 = {  
  title: 'Web Technologies',  
  author: {  
    name: "Usman Akram"  
  }  
}
```


Trade Off between Query Performance Vs Consistency

NORMALIZATION

A change in author would reflect every where

More Consistent but need extra query to get child records

DE NORMALIZATION

If you need to change the author you will have to modify in multiple records

Not Consistent but More Performance

Hybrid Approach

```
let author = {  
  name: "Usman Akram"  
  // 50 More properties  
}
```

```
let course = {  
  title: 'Web Technologies',  
  author: {  
    name: "Usman Akram",  
    id: 'reference to author'  
  }  
}
```

// Copy id and some of specific properties. Like facebook top comment should be beside post

Mongoose Recap and Code for This section

<https://1drv.ms/f/s!AtGKdbMmNBGd0ia66DfXulxFIM6m>

```
const mongoose = require('mongoose');
//npm init -yes
//npm install mongoose
mongoose.connect('mongodb://localhost/playground')
  .then(() => console.log('Connected to MongoDB...'))
  .catch(err => console.error('Could not connect to MongoDB...',
err));
```

Author

```
const Author = mongoose.model('Author', new
mongoose.Schema({
  name: String,
  bio: String,
  website: String
}));
```

Course

```
const Course = mongoose.model('Course', new
mongoose.Schema({
  name: String,
}));
```

Create Author

```
async function createAuthor(name, bio, website) {  
  const author = new Author({  
    name,  
    bio,  
    website  
  });  
  const result = await author.save();  
  console.log(result);  
}
```

Create Course

```
async function createCourse(name, author) {  
  const course = new Course({  
    name,  
    author  
  });  
  const result = await course.save();  
  console.log(result);  
}
```

List Courses

```
async function listCourses() {  
  const courses = await Course  
    .find()  
    .select('name');  
  console.log(courses);  
}
```


Actually Create an Author

```
createAuthor('Mosh', 'My bio', 'My Website');
```

```
createCourse('Web Technologies',  
'5c3e1c1d8b3a1c1d0c1ac03d')
```

```
// whatever id the create course has given to new doc put  
that in create Course
```

Author Doc

```
{  
  "_id": "5c3e1e0734ea7010842c96df",  
  "name": "Web Technologies",  
  "author": "5c3e1c1d8b3a1c1d0c1ac03d",  
  "__v": 0  
}
```

Population

```
async function listCourses() {  
  const courses = await Course  
    .find()  
    .populate('author')  
    .select('name author');  
  console.log(courses);  
}
```

Author populated

```
{_id: 5c3e1e0734ea7010842c96df,  
  name: 'Web Technologies',  
  author:  
    {_id: 5c3e1c1d8b3a1c1d0c1ac03d,  
     name: 'Usman',  
     bio: 'My bio',  
     website: 'usmanlive.com',  
     __v: 0 }  
}
```

Population only get author name

```
async function listCourses() {  
  const courses = await Course  
    .find()  
    .populate('author', 'name')  
    .select('name author');  
  console.log(courses);  
}
```

Population only get author name -id

```
async function listCourses() {  
  const courses = await Course  
    .find()  
    .populate('author', 'name -_id')  
    .select('name author');  
  console.log(courses);  
}
```

Multiple Populations

```
async function listCourses() {  
  const courses = await Course  
    .find()  
    .populate('author', 'name -_id')  
    .populate('category', 'name')  
    .select('name author');  
  console.log(courses);  
}
```

What if the author ID is changed

E.g. Course has an author ID which Actually doesn't exist in author collection

-Will Mongo Complain –No

-What will return –Null