Mongoose Cheat Sheet

Settings

```
npm install mongoose
```
// install mongoose

```
const mongoose = require('mongoose');
```
// import mongoose

```
mongoose.connect('mongodb://localhost:27017/dbName');
or
mongoose.connect('mongodb://127.0.0.1:27017/dbName');
```
// open connection for local hosted database

```
mongoose.connect('mongodb://username:password@host:port/database?options...');
```
// connection string parts

```
mongoose.connection.close();
```
// close connection

Schema

Simple Schema

```
const addressSchema = new mongoose.Schema({
    addressName:String,
    street:String,
    building:Number,
    city:String,
    isDefault:Boolean
},{
    timestamps:true,  // save add and edit time
})
```
// simple schema

With Validation

```
const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,          // Field is required
    unique: true,            // Field must be unique
    minlength: 3,            // Minimum length
    maxlength: 30,           // Maximum length
    trim: true,              // Remove leading/trailing whitespace
    match: /^[a-zA-Z0-9]+$/, // Regular expression pattern
    uppercase: true,         // Convert to uppercase
  },
  email: {
    type: String,
    required: true,
    unique: true,
    trim: true,
    lowercase: true,
    validate: {
      validator: (value) => {
        // Custom validator function
        return /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i.test(value);
      },
      message: 'Invalid email format', // Custom error message
    },
  },
  age: {
    type: Number,
    min: 18,   // Minimum value
    max: 120,  // Maximum value
    required: true,
    validate: {
      validator: Number.isInteger,  // Custom validator using a function
      message: '{VALUE} is not an integer', // Custom error message
    },
  },
  password: {
    type: String,
    required: true,
    minlength: [8, 'Password must be at least 8 characters long'], // Custom message
  },
  address:[addressSchema],
  createdAt: {
    type: Date,
    default: Date.now, // Default value
  }

});
```

link collections

```
const addressSchema = new mongoose.Schema({
    addressName:String,
    street:String,
    building:Number,
    city:String,
    isDefault:Boolean,
userType:{
    type: mongoose.SchemaTypes.ObjectId,
    ref:'type',
}
})
```
// link collection user with collection type

Pre, post

```
userSchema.pre("save", function(next) {
  if(this.username === 'MYNAME'){
    this.username = 'new name'
  }
  next() //move to the action in this case save
});
userSchema.post('save', function() {
    console.log(`${this.username} has been saved`);
});
```
// middleware to do somthing before action (you can change save by any other action like validate, find, findOne .. etc


should declared before model declaration

Model

Create Model

```
const userModel = mongoose.model('user',userSchema);
```
// create model mongoose.model('collection name','schema name'

CRUD

Create

create

```
User.create({username:"myName",email:"myemail@domain.com",age:40,password:"mypassword",address:
[{ addressName:"Home",
    street:"my street",
    building:5,
    city:"my city",
    isDefault:true}]})
```
// create document

save

```
const newUser = new User({username:"new user",email:"myemail@domain.com",age:40})
newUser.save();
```
// create new user object then save its document

Insert many

```
User.insertMany([{username:"new user 1",email:"myemail1@domain.com",age:40},
{username:"new user 2",email:"myemail2@domain.com",age:40}])
```
// it takes an array of user objects or it can take just one object

Retrieving

find

```
User.find({}).then((data)=>{
    console.log(data)
})
or
async function findUsers() {
    const users = await User.find({});
    console.log(users);
}
User.find({ username:"NEW NAME",age:40})
```
// find all data as a promise so use then or async / await returns array of objects

// find with conditions (equal)

```
User.find({$and:
[{username:"NEW NAME"},
{age:40}]})
User.find({$or:
[{username:"NEW NAME"},
{age:40}]})
```
// find with logical conditions and / or

```
User.find({age:{$eq:40}})
//equal
User.find({age:{$ne:30}}) // not equal
User.find({age:{$gt:30}})
//greater than
User.find({age:{$gte:30}})
//greater than or equal
User.find({age:{$lt:30}})
//less than
User.find({age:{$lte:30}})
//less than or equal
```
// find with conditions

```
User.find({ username: { $in: ['MYNAME', 'myname 1']}})
```
// find if the value in array

```
User.find({ username: { $nin: ['MYNAME', 'myname 1']}})
```
// find if the value not in array

```
User.find({ username: /MY/ }) // contains
User.find({ username: /^MY/ }) // starts with
User.find({ username: /NAMES/ }); // ends with
```
// using regular expression

```
User.find({},'username age')
//display username, email only (_id will display by default
User.find({},'username -_id')
//display only username
```
// display or not fields

where

```
User.where('age',40)
User.where('age').equals(40)
User.where('age').ne(40)
User.where('age').gt(40)
User.where('age').gte(40)
User.where('age').lt(40)
User.where('age').lte(40)
```
// find with conditions

findOne

```
User.findOne({age:40})
```
// find the first document meets conditions you can use all the conditions like find // returns Object of the document

findById

```
User.findById('652999f63004e85e97881c67')
```
// find document by its id // returns Object of document

count

```
User.countDocuments()
// it can use conditions too
User.countDocuments({ age: { $gte: 40 }})
```
// count the documents numbers // returns number

sort

```
User.find({}).sort('age') // sorting ascending
User.find({}).sort('-age') // sortind descending
// also you can sort by more than field
User.find({}).sort('-age username')
// also you can use it with where
User.where({}).sort('-age username')
```

limit

```
User.find({}).limit(1) // returns an array of 1 object
```
// returns only the number of limitation

skip

```
User.find({}).skip(4) // don't return first documents
```
// don't return first # documents

populate

```
User.find({}).populate('userType')
```
// returns data from linked collection type

```
Result:
{
    _id: new ObjectId('652994046d1a1712aeec84ae'),
    username: 'ahmed',
    email: 'myemail@domain.com',
    userType: {
        _id: new ObjectId('6529b97aef1842dfe4fc5c01'),
        typeName: 'admin'
    }
}
```

//You can use them as a chain functions like: User.find({}).skip(4).limit(2).sort('age') with any order

Retrieving hacks:

Find Maximum Value in a Field: User.findOne().sort('-age')
Find the minimum value in a field: User.findOne().sort('age')
Sum values in a field: User.aggregate([{ $group: { _id: null, total: {$sum: '$age' } } },])
Average Values in a Field: User.aggregate([{ $group: { _id: null, average: { $avg: '$age' } } },])

Update

save

```
async function myFun() {
    const myUser = await User.findById('652994046d1a1712aeec84ae')
    ;
    myUser.username = 'new name';
    myUser.save();
}
```
// when you have document or documents you can change them and call save method

find by id and update

```
User.findByIdAndUpdate('652994046d1a1712aeec84ae',{username:'another new name'})
```
// update one document

Find one and update

```
User.findOneAndUpdate({age:10},{username:'new username'})
```
// update first document that meets find conditions findOneAndUpdate({conditions},{new values})

Update many

```
User.updateMany({ age: { $lt: 30 } }, { $set: { status: 'young' } });
```
// update all documents that meet find conditions updateMany({conditions},{new values with $set})

Updates hacks:

User.updateOne({username: 'my username'}, {$inc:{ age: 1 }}) // Increment a numeric field
User.updateOne({username:'young'},{$push:{address:{ addressName:"work",
    street:"my street",
    building:5,
    city:"my city",
    isDefault:false}
} }) // push item inside an array
User.updateOne({username:'young'},{$pull:{address:{ addressName:"work",
    street:"my street",
    building:5,
    city:"my city",
    isDefault:false}
} }) // pull item from an array

Delete

Find by id and delete

```
User.findByIdAndDelete('652994a658646fdc14cc3d2e')
```

Find one and delete

```
User.findOneAndDelete({ username: 'john_doe' })
```

Delete many

```
User.deleteMany({ age: { $gte: 40 } })
```
// delete all documents meet conditions

Delete all

```
User.deleteMany({})
```