

# NoSQL and MongoDB

NoSQL vs SQL, MongoDB, Mongoose



**SoftUni Team**  
**Technical Trainers**



**SoftUni**



**Software University**

<https://softuni.bg>

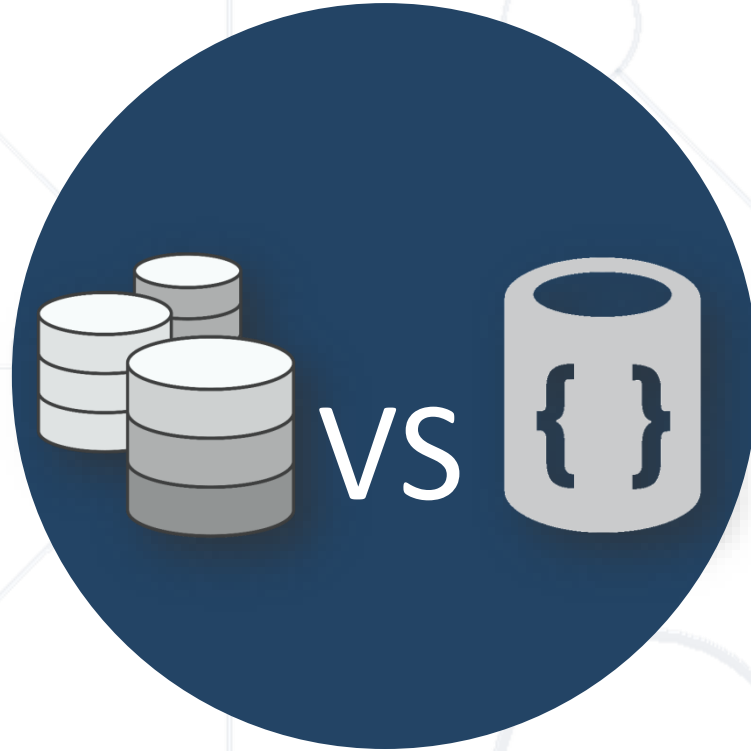
sli.do

**#js-back-end**

# Table of Contents

1. **Relational** and **Non-Relational** Databases
2. **MongoDB** and **Mongoose** Overview
3. **Mongoose** Models
4. **CRUD** with Mongoose
5. Mongoose **Querying**





# Relational and NoSQL Databases

# Relational Database

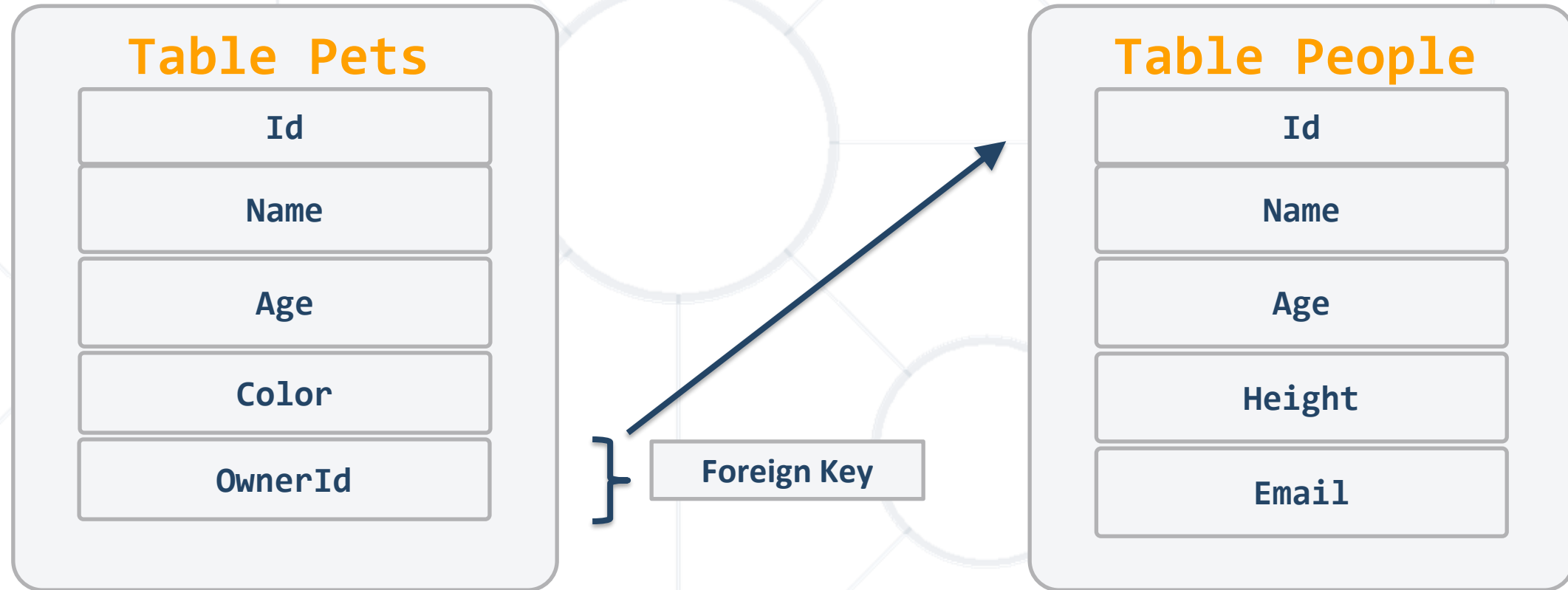
- Organize data into one or more **tables** of **columns** and **rows**
- Unique **key** identifying each **row** of data
- Almost all relational databases use **SQL** to **extract** data



```
SELECT * FROM Students
```

- **Relations** between tables are done using **Foreign Keys (FK)**
- Such databases are **Oracle**, **MariaDB**, **SQL Server**, etc...

# Relational Database – Example



- Key-value **stores**

```
{  
  "_id": ObjectId("59d3fe7ed81452db0933a871"),  
  "email": "peter@gmail.com",  
  "age": 22  
}
```

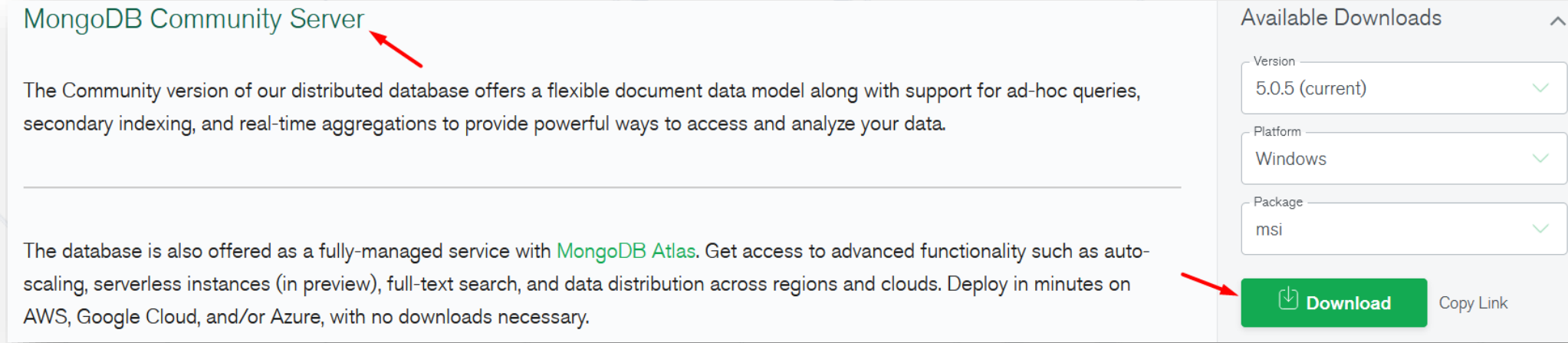
- **SQL** query is **not** used in NoSQL systems
- More **scalable** and **provide** superior **performance**
- Such databases are **MongoDB, Cassandra, Redis**, etc..



# MongoDB Overview



- Download from: [mongodb.com/try/download/community](https://mongodb.com/try/download/community)



**MongoDB Community Server**

The Community version of our distributed database offers a flexible document data model along with support for ad-hoc queries, secondary indexing, and real-time aggregations to provide powerful ways to access and analyze your data.

The database is also offered as a fully-managed service with [MongoDB Atlas](#). Get access to advanced functionality such as auto-scaling, serverless instances (in preview), full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

**Available Downloads**

Version: 5.0.5 (current) ✓

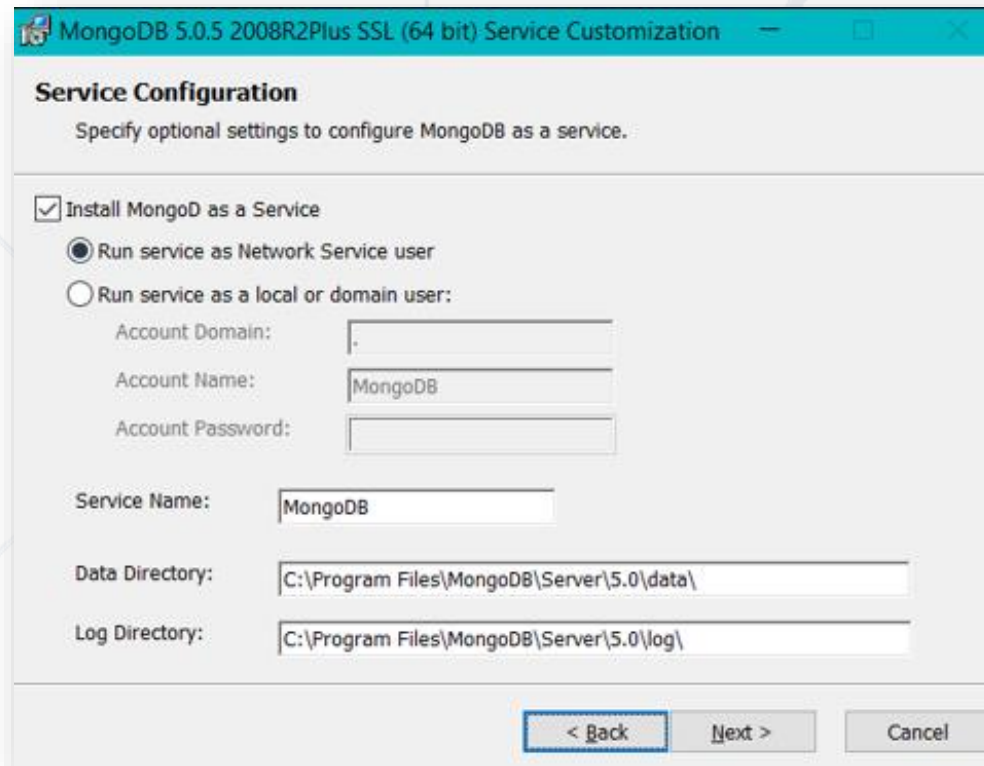
Platform: Windows ✓

Package: msi ✓

**Download** Copy Link

- The package includes **MongoDB Compass**
- When **installed**, MongoDB needs a **driver** (for every project)
  - Install MongoDB **driver** for Node.js `npm install mongodb`
  - We will be using **Mongoose** (includes a driver)

- During installation, configure the **MongoDB service**:



- Required if you **skipped** the service installation (and for Linux)
  - Go to installation folder and **run** a command prompt as an **administrator**
  - Type the following command

Usually in C:\Program Files\MongoDB\Server\3.4\bin

```
<path to mongod.exe> mongod --dbpath <path to store data>
```

- Additional information at <https://docs.mongodb.com/manual/tutorial/>

# Working with MongoDB Shell Client

- Start the shell from **another** CLI

- Type the command **mongo**

```
show dbs
```

```
use mytestdb
```

```
db.mycollection.insertOne({"name": "George"})
```

```
db.mycollection.find({"name": " George"})
```

```
db.mycollection.find({})
```

- Additional information at

- <https://docs.mongodb.com/manual/reference/mongo-shell/>

- Choose one of the many (**Compass** is included in the installer)
- For example
  - Compass - <https://www.mongodb.com/products/compass>
  - Robo 3T - <https://robomongo.org/download>
  - NoSQLBooster - <https://nosqlbooster.com>

```
const mongodb = require('mongodb');
const MongoClient = mongodb.MongoClient;
const connectionStr = 'mongodb://localhost:27017';
const client = new MongoClient(connectionStr,
{useUnifiedTopology: true});
client.connect(function(err) {
  const db = client.db('testdb');
  const people = db.collection('people');
  people.insertOne({ 'name': 'Ivan' }, (err, result) => {
    people.find({ name: 'Ivan' }).toArray((err, data) => {
      console.log(data);
    });
  });
});
});
```



# Mongoose Overview

# Mongoose Overview

- Mongoose is an object-document **model** module in Node.js for MongoDB
- It **provides** a straight-forward, **schema-based** solution to **model** your application data
- Includes build-in type **casting** and **validation**
- **Extends** the native **queries** (much **easier** to use)
- To **install** type in terminal/CMD (for every project)

```
npm install mongoose --save
```





- Load the following module

```
const mongoose = require('mongoose')
```

- **Connecting** to the database

```
async function main(){  
  await mongoose.connect('mongodb://localhost:27017/testdb', {  
    useNewUrlParser: true,  
    useUnifiedTopology: true  
  });  
  console.log("Database connected")  
}  
main();
```

- Host a **database** in the largest MongoDB **cloud** service
- Go to '**mongo atlas**' and register -  
<https://www.mongodb.com/cloud/atlas>
- You can **store** up to 500 MB of **content**



# Mongoose Models

# Mongoose Models

- Mongoose **supports** models
  - Fixed **types** of documents
    - Used like object **constructors**
- Needs a **mongoose.Schema** call

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
  firstName: String,
  lastName: String,
  facultyNumber: { type: String, required: true },
  age: Number
});
const Student = mongoose.model('Student', studentSchema);
```



# Mongoose Models - Example

```
const myPerson = new Student({
  firstName: "John",
  lastName: "Peterson",
  facultyNumber: "5014sa",
  age: 25
});

await myPerson.save();
const data = await Student.find({});
console.log(data); /* [{_id: new ObjectId("6139c6faf79365e5e54645bf"),
  firstName: 'John',
  LastName: 'Peterson',
  facultyNumber: '5014sa',
  age: 25, __v: 0}] */
```

- Since Mongoose models are just JavaScript **object constructors**, they can have **methods**

```
const studentSchema = new mongoose.Schema({...});

studentSchema.methods.getInfo = function() {
  return `I am ${this.firstName} ${this.lastName}`;
};
```

Do **not** use  
arrow functions

- Not all properties **need** to be **persisted** in the **database**
- Mongoose provides a way to **create** properties, that are accessible on all models, but are **not persisted** to the database
  - And they have both **getters** and **setters**

```
studentSchema.virtual('fullName').get(function () {  
  return this.firstName + ' ' + this.lastName  
});
```

- With Mongoose developers can **define** custom **validation** on their **properties**
- Validate records when trying to **save**

```
studentSchema.path('firstName')  
  .validate(function () {  
    return this.firstName.length >= 2  
    && this.firstName.length <= 10  
  }, 'First name must be between 2 and 10 symbols long!')
```

Error message as second param



- Mongoose has several built-in validators
  - All **Schema**-Types have the built-in required validator
  - Numbers have **min** and **max** validators
  - Strings have **enum**, **regex**, **minLength**, and **maxLength** validators

```
facultyNumber: {  
  type: String,  
  required: [true, 'FacultyNumber is required']  
}
```

Error message

- You can configure the error message for individual validators in your schema

- Mongoose replaces **{VALUE}** with the value being validated

```
const studentSchema = new mongoose.Schema({  
  age: {  
    type: Number,  
    min: [0, 'Must be at least 0, got {VALUE}'],  
    max: 50 }  
});
```

Error message as  
second param

```
const studentSchema = new mongoose.Schema({  
  facultyNumber: {  
    type: String,  
    enum: {  
      values: ['50121', '50122', '50123'],  
      message: '{VALUE} is not valid'  
    }  
  }  
});
```

- Having all model definitions in the **main** module is **no** good
  - That is the reason Node.js has **modules** in the first place
  - In folder **models**, file **Student.js**

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
  firstName: { type: String, required: true },
  age: { type: Number }
});

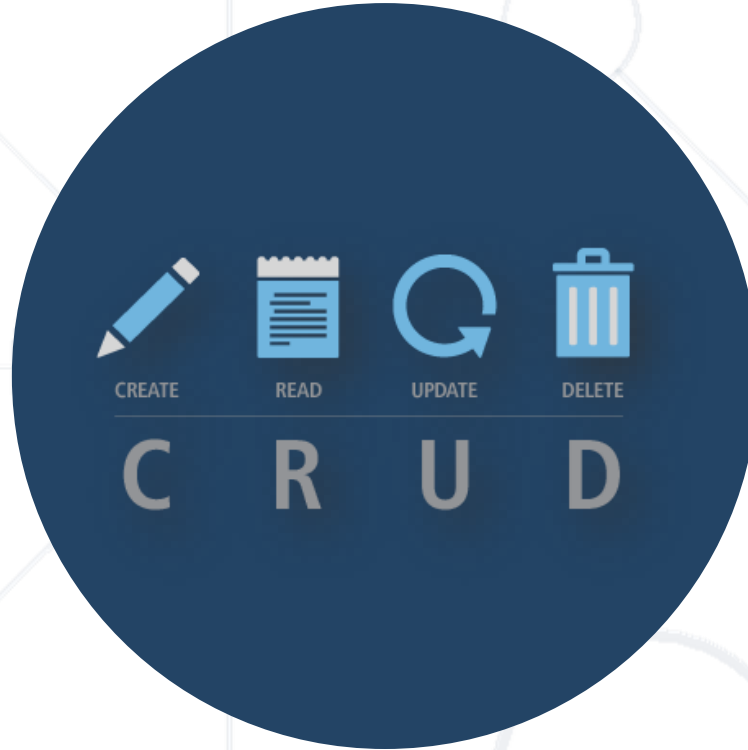
module.exports = mongoose.model('Student', studentSchema);
```

- Export the model definition:

```
const mongoose = require('mongoose');  
const studentSchema = new mongoose.Schema({ /* ... */ });  
  
module.exports = mongoose.model('Student', studentSchema);
```

- We can put each **model** in a different **module**, and **load** all models where they are needed

```
const Student = require('./models/Student');
```



# CRUD with Mongoose

# CRUD with Mongoose

- Mongoose supports **all** CRUD operations

- Create (Persist data)

```
new Student({}).save(callback)
```

- Read (Extract data)

```
Student.find({})
```

```
Student.findOne({conditions}, {options}, callback)
```

```
Student.findById(id, {options}, callback)
```



# CRUD with Mongoose

- Update (Modify data)

Student

```
.findByIdAndUpdate(id, {$set: {prop: newVal}}, callback)
```

Student

```
.updateOne({filter}, {$set: {prop: newVal}}, callback)
```

Student

```
.updateMany({filter}, {$set: {prop: newVal}}, callback)
```

- Delete (Remove data)

```
Student.findByIdAndDelete(id, callback)
```

```
Student.deleteOne({conditions}, {options}, callback)
```

```
Student.deleteMany({conditions}, {options}, callback)
```



# Create Example

```
const mongoose = require('mongoose');
const connectionStr = 'mongodb://localhost:27017/unidb';
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true, minlength: 3 },
  age: Number
});
const Student = mongoose.model('Student', studentSchema);
mongoose.connect(connectionStr).then(() => {
  new Student({ name: 'Petar', age: 21 })
    .save()
    .then(student => {
      console.log(student._id)
    });
});
```

You can also use  
`Student.create()`



Student

```
.find({})  
.then(students => console.log(students))  
.catch(err => console.error(err))
```

Student

```
.find({name: 'Petar'})  
.then(students => console.log(students))
```

Student

```
.findOne({name: 'Petar'})  
.then(student => console.log(student))
```

Always handle errors

# Update Example

Student

```
.findById('57fb9fe1853ab747b0f692d1')  
.then((student) => {  
  student.name = 'Stamat'  
  student.save()  
});
```

Student

```
.findByIdAndUpdate('57fb9fe1853ab747b0f692d1', {  
  $set: { name: 'Petar' }  
}).then(students => console.log(students))
```

Student

```
.updateOne(  
  { name: 'Petar' },  
  { $set: { name: 'Kiril' } }). then(students =>  
console.log(students))
```

# Remove and Count Example

Student

```
.findByIdAndDelete( '57fb9fe1853ab747b0f692d1' ).then()
```

Student

```
.deleteOne({ name: 'Stamat' }).then()
```

Student

```
.countDocuments()  
.then(console.log)
```

Remove by criteria

Student

```
.countDocuments({ age: { $gt: 19 } })  
.then(console.log)
```

Get the count by criteria



# Mongoose Queries

# Mongoose Queries

- Mongoose defines **all** queries of the native MongoDB driver in a more **clear** and **useful** way

- Instead of

```
{  
  $or: [  
    {conditionOne: true},  
    {conditionTwo: true}  
  ]  
}
```

- Do

```
.where({ conditionOne: true })  
.or({ conditionTwo: true })
```



# Mongoose Queries Example

- Mongoose supports **many** queries

- For equality / non-equality

```
Student.findOne({'lastName': 'Petrov'})
```

```
Student.find({}).where('age').gt(7).lt(14)
```

```
Student.find({}).where('facultyNumber').equals('12399')
```

- Selection of some properties

```
Student.findOne({'lastName': 'Kirilov'}).select('name age')
```

# Mongoose Queries Example 2

- Sorting

```
Student.find({}).sort({age:-1})
```

- Limit & skip

```
Student.find({}).sort({age:-1}).skip(10).limit(10)
```

- Different methods could be **stacked** one upon the other

```
Student.find({})  
  .where('firstName').equals('gosho')  
  .where('age').gt(18).lt(65)  
  .sort({age:-1})  
  .skip(10)  
  .limit(10)
```



**Model Population**



# Population Definition

- Population is the process of **automatically replacing** the **specified paths** in the document with document(s) from **other** collection(s)
- We may **populate** a single document, multiple documents, plain object, multiple plain objects, or all objects returned from a query



- We create **two models** that **reference** each other

```
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  facultyNumber: String,
  teacher: { type: Schema.Types.ObjectId, ref: 'Teacher' },
  subjects: [{ type: Schema.Types.ObjectId, ref: 'Subject' }]
});
const subjectSchema = new mongoose.Schema({
  title: String,
  students: [{ type: Schema.Types.ObjectId, ref: 'Student' }]
});
const Student = mongoose.model('Student', studentSchema);
const Subject = mongoose.model('Subject', subjectSchema);
```

- To load all the data **referenced** with the entity use **populate()**

```
Student.findOne({ name: 'Peter' })  
  .populate('subjects')  
  .then(student => {  
    console.log(student.subjects)  
  })
```

Will return an array of  
**objects** and **NOT** Id's

- You can also load **multiple** paths

```
Student.findOne({ name: 'Peter' })  
  .populate('subjects')  
  .populate('teacher')  
  .then(student => {  
    console.log(student.teacher)  
    console.log(student.subjects)  
  })
```

- Populate based on a **condition**

```
Subject.  
  find({})  
  .populate({  
    path: 'students',  
    match: { age: { $gte: 19 }},  
    select: 'name facultyNumber',  
    options: { limit: 3 }  
  })
```

- More on populate here - [mongoosejs.com/docs/populate.html](https://mongoosejs.com/docs/populate.html)

- NoSQL databases provide **superior** performance
- Mongoose gives us a **schema-based** solution

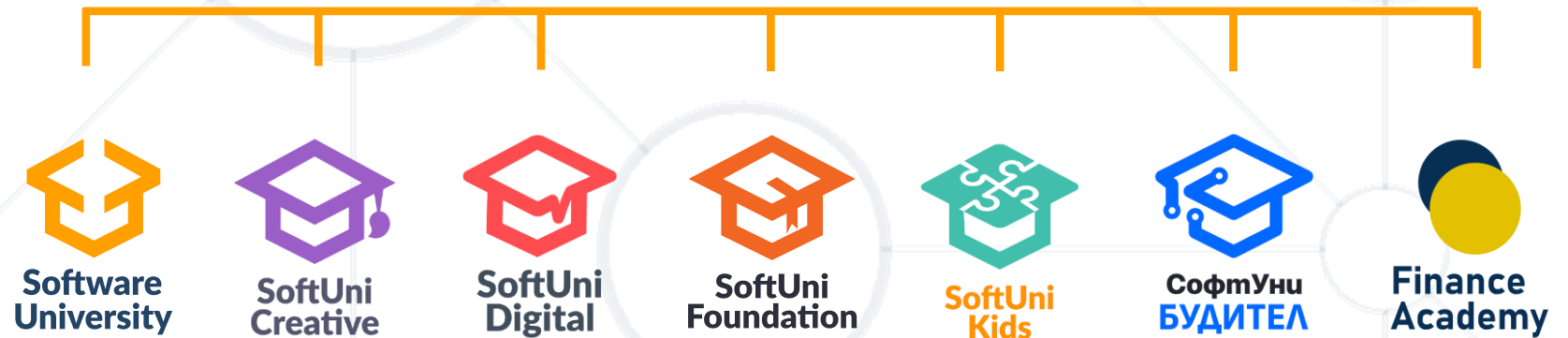
```
const modelSchema = new mongoose.Schema({  
  propString: String  
});
```

- Mongoose supports all **CRUD** operations
- Chaining queries with Mongoose is possible

```
Student.find({}).where('firstName').equals('gosho')  
.where('age').gt(18).lt(65).sort({age:1}).skip(10)  
.limit(10)
```



# Questions?



# SoftUni Diamond Partners





- Software University – High-Quality Education, Profession and Job for Software Developers
  - [softuni.bg](http://softuni.bg)
- Software University Foundation
  - [softuni.foundation](http://softuni.foundation)
- Software University @ Facebook
  - [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)





- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

