NoSQL DATABASES IN NODE.JS

AGENDA

- SQL vs NoSQL
- MongoDB Overview
- MongoDB Node.js Driver
- Mongoose ODM

NoSQL

SQL

- SQL – Structured Query Language
- Relational data model

Microsoft SQL Server PostgreSQL MySQL ORACLE DATABASE

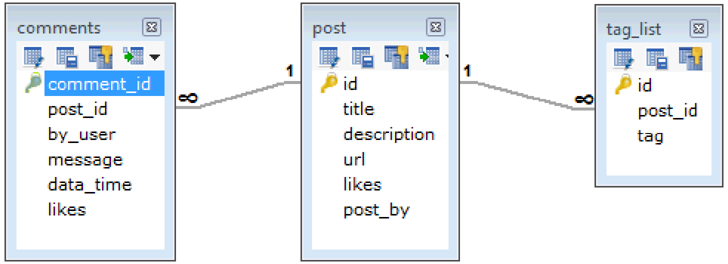
NoSQL

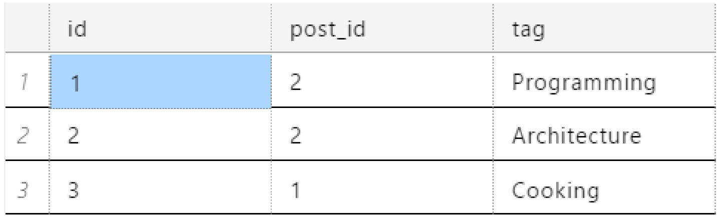
- No SQL - non-relational model
- Data models:
  - Column store
  - Document store
  - Key value/tuple store
  - Graph databases
  - Multimodel databases

APACHE HBASE mongoDB redis cassandra neo4j DynamoDB amazon web services Firebase

SQL Example

```
• CREATE TABLE tag_list (
•     id INT PRIMARY KEY NOT NULL,
•     post_id INT FOREIGN KEY REFERENCES post (id) NOT NULL,
•     tag VARCHAR(50) NOT NULL)
```





	id	post_id	tag
1	1	2	Programming
2	2	2	Architecture
3	3	1	Cooking

Document-Oriented NoSQL

- Key features:
- Each record is stored with its associated data in a single document
- Fast reads
- Query language
- Dynamic schema

```
{
  "_id": "5a27c5041b359b756d3c07c0",
  "address": {
    "building": "351",
    "coord": [ -73.98513559999999, 40.7676919 ],
    "street": "West   57 Street",
    "zipcode": "10019"
  },
  "borough": "Manhattan",
  "cuisine": "Irish",
  "grades": [
    {
      "date": "2014-09-06T00:00:00.000Z",
      "grade": "A",
      "score": 2
    },
    {
      "date": "2013-07-22T00:00:00.000Z",
      "grade": "A",
      "score": 11
    }
  ],
  "name": "Dj Reynolds Pub And Restaurant",
  "restaurant_id": "30191841"
}
```

- Use cases: cache, sessions, logging, CMS, blogging platforms and etc.

mongoDB. Firebase CouchDB relax MarkLogic™

SQL vs NoSQL

SQL
Defined schema
ACID compliant (atomicity, consistency, isolation, durability)
Scale up
SQL
Lots of tools for DB development

NoSQL
Schemaless
BASE compliant (basically available, soft state, eventual consistency)
Scale up and out
Object-based APIs
Applications are the primary interface to the DB

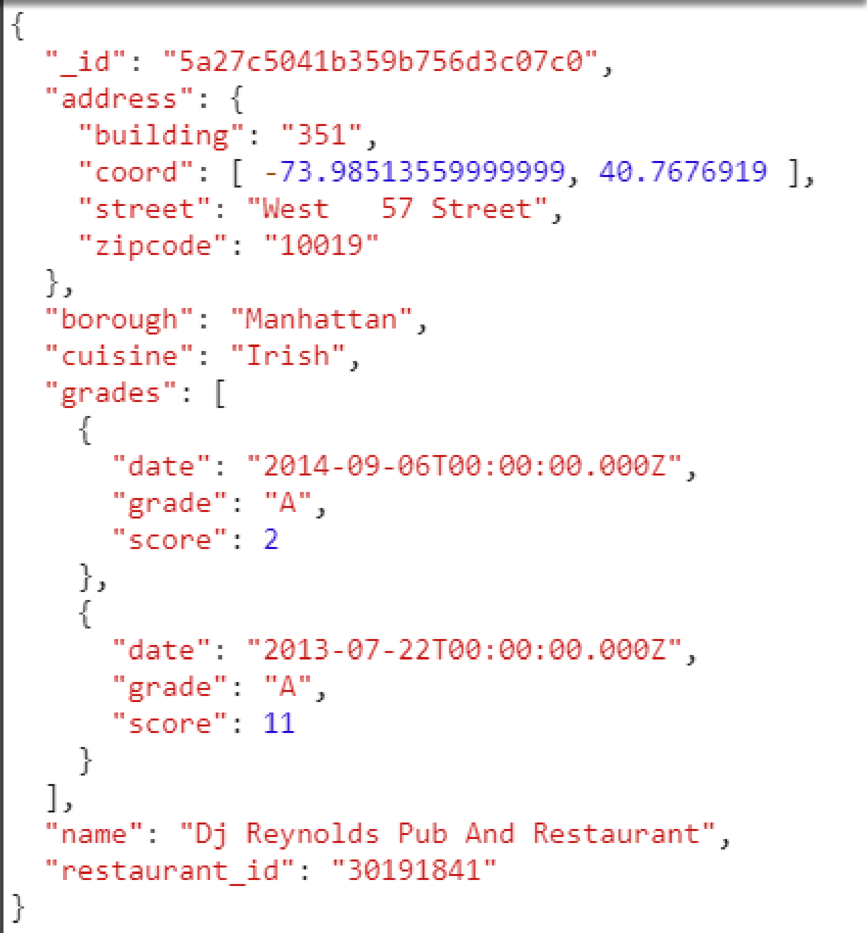
MongoDB Overview

MongoDB is a

- cross-platform
- open source
- document-oriented

database that provides

- high performance
- high availability
- horizontal scaling
- rich query language
- multiple storage engines (MMAPv1, In-Memory, WiredTiger)



```
{
  "_id": "5a27c5041b359b756d3c07c0",
  "address": {
    "building": "351",
    "coord": [ -73.98513559999999, 40.7676919 ],
    "street": "West   57 Street",
    "zipcode": "10019"
  },
  "borough": "Manhattan",
  "cuisine": "Irish",
  "grades": [
    {
      "date": "2014-09-06T00:00:00.000Z",
      "grade": "A",
      "score": 2
    },
    {
      "date": "2013-07-22T00:00:00.000Z",
      "grade": "A",
      "score": 11
    }
  ],
  "name": "Dj Reynolds Pub And Restaurant",
  "restaurant_id": "30191841"
}
```

MongoDB Overview

Document is a set of key-value pairs similar to JSON objects.

Collection is a group of documents.

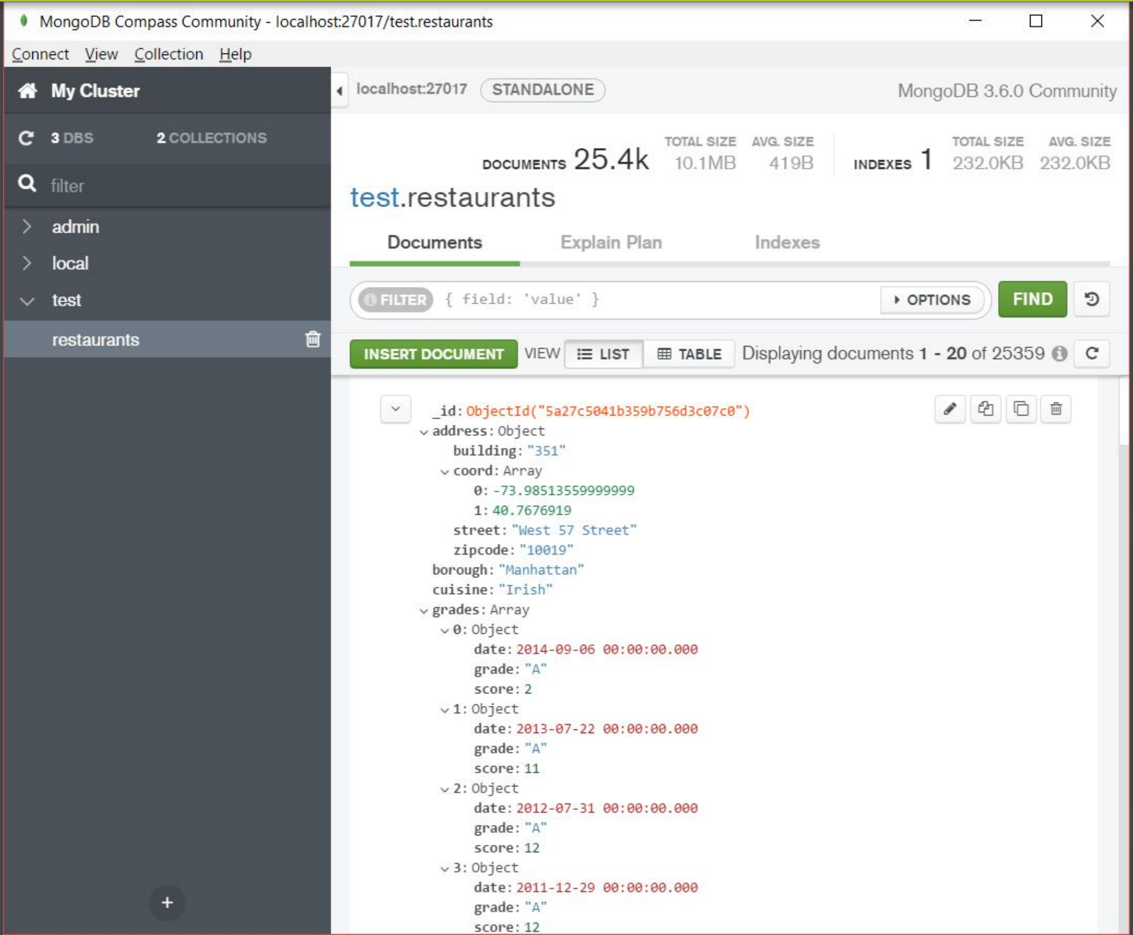
Databases hold collections of documents.

Advantages of documents:

- Native data types
- No expensive joins
- Dynamic schema

MongoDB Installation

https://docs.mongodb.com/manual/installation/


MongoDB Compass Community - localhost:27017/test.restaurants
Connect View Collection Help
My Cluster
3 DBS 2 COLLECTIONS
filter
admin
local
test
restaurants
localhost:27017 STANDALONE
MongoDB 3.6.0 Community
DOCUMENTS 25.4k TOTAL SIZE 10.1MB AVG. SIZE 419B
INDEXES 1 TOTAL SIZE 232.0KB AVG. SIZE 232.0KB
test.restaurants
Documents Explain Plan Indexes
FILTER
OPTIONS
FIND
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 20 of 25359

```
{ field: 'value' }
```

```
_id: ObjectId("5a27c5041b359b756d3c07c0")
address: Object
    building: "351"
    coord: Array
        0: -73.98513559999999
        1: 40.7676919
    street: "West 57 Street"
    zipcode: "10019"
borough: "Manhattan"
cuisine: "Irish"
grades: Array
    0: Object
        date: 2014-09-06 00:00:00.000
        grade: "A"
        score: 2
    1: Object
        date: 2013-07-22 00:00:00.000
        grade: "A"
        score: 11
    2: Object
        date: 2012-07-31 00:00:00.000
        grade: "A"
        score: 12
    3: Object
        date: 2011-12-29 00:00:00.000
        grade: "A"
        score: 12
```

MongoDB Node.js Driver

Getting started

```
const { MongoClient } = require('mongodb’);
const assert = require('assert');

// Connection URL
const url = 'mongodb://localhost:27017/db_name‘;

// Use connect method to connect to the Server
MongoClient.connect(url, (err, connection) => {
    assert.equal(null, err);
    console.log("Connected correctly to server");
    connection.close();
});
```

MongoDB Native Driver

Find All Documents

a simple query that returns all the documents.

```
db.users.find()
```

Find specific document using custom query

a simple query that returns a document matching the query.

```
✓  db.users.find({ _id: ObjectId("5a2fccbb98c538bdb299b98b") })
x  db.users.find({ _id: "5a2fccbb98c538bdb299b98b" })
✓  db.users.find({ lastLogin: ISODate("2018-10-10T00:00:00") })
```

MongoDB Native Driver

```
db.inventory.insertMany( [
   { item: "journal", price: 10 },
   { item: "notebook", price: 15 },
   { item: "paper", price: 9 },
   { item: "planner" },
   { item: "postcard" }
]);
```

MongoDB Native Driver

```
db.collection.updateOne(<filter>, <update>)
```

```
db.collection.updateMany(<filter>, <update>)
```

```
db.collection.replaceOne(<filter>, <replacement>)
```

```
db.inventory.updateOne({ status: "A" }, { $set: { deprecated: 1 } })
db.inventory.findOne({ status: "A" }, { size: 0, instock: 0 })
```

```
{ "_id" : ObjectId("5a42bcac64d214fe884232e1"), "item" : "journal", "status" : "A", "deprecated" : 1 }
```

```
db.inventory.replaceOne({ status: "A" }, { deprecated: 2 })
db.inventory.find ()
```

```
{ "_id" : ObjectId("5a42bcac64d214fe884232e1"), "deprecated" :2  }
```

MongoDB Native Driver

```
db.collection.deleteOne(<filter>)
```

```
db.collection.deleteMany(<filter>)
```

Remove all items with height less than 10:

```
db.inventory.deleteMany({ "size.h": { $lt: 10 } })
```

WHAT IS MONGOOSE?

mongoose

elegant mongodb object modeling for node.js

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, out of the box.

MONGOOSE FUNDAMENTALS

Getting started

The first thing we need to do is include mongoose in our project and open a connection

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');
```

MONGOOSE FUNDAMENTALS

Getting started

We have a pending connection to the test database running on localhost. We now need to get notified if we connect successfully or if a connection error

```
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection
error:'));
db.once('open', function() {
    // we're connected!
});
```

MONGOOSE FUNDAMENTALS

With Mongoose, everything is derived from a **Schema**. Let's get a reference to it and define our kittens.occurs:

```
const kittySchema = mongoose.Schema({
    name: String
});
```

So far so good. We've got a schema with one property, name, which will be a String. The next step is compiling our schema into a Model.

```
const Kitten = mongoose.model('Kitten',
kittySchema);
```

MONGOOSE FUNDAMENTALS

A model is a class with which we construct documents. In this case, each document will be a kitten with properties and behaviors as declared in our schema. Let's create a kitten document representing the little guy we just met on the sidewalk outside:

```
const silence = new Kitten({ name: 'Silence'
});
console.log(silence.name); // 'Silence'
```

MONGOOSE FUNDAMENTALS

Kittens can meow, so let's take a look at how to add "speak" functionality to our documents:

```
// NOTE: methods must be added to the schema before compiling
it with mongoose.model()
kittySchema.methods.speak = function () {
    const greeting = this.name
        ? "My name is " + this.name
        : "I don't have a name";
    console.log(greeting);
}
const Kitten = mongoose.model('Kitten', kittySchema);
```

MONGOOSE FUNDAMENTALS

Functions added to the methods property of a schema get compiled into the Model prototype and exposed on each document instance:

```
const fluffy = new Kitten({ name: 'fluffy' });
fluffy.speak(); // "My name is fluffy"
```

MONGOOSE FUNDAMENTALS

Each document can be saved to the database by calling its save method. The first argument to the callback will be an error if any occured.

```
fluffy.save((err, fluffy) => {
    if (err) return console.error(err);
    fluffy.speak();
});
```

We can access all of the kitten documents through our Kitten model.

```
Kitten.find((err, kittens) => {
    if (err) return console.error(err);
    console.log(kittens);
});
Kitten.find({ name: /^fluff/ }, callback);
```

MONGOOSE FUNDAMENTALS

Using Query

```
const Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost',
// selecting the `name` and `occupation` fields
Person.findOne({ 'name.last': 'Ghost' }, 'name occupation', (err, person) => {
    if (err) return handleError(err);
    // Space Ghost is a talk show host.
    console.log('%s %s is a %s.', person.name.first, person.name.last, person.occupation)
})
```

MONGOOSE FUNDAMENTALS

Using Query without callback:

```
const Person = mongoose.model('Person', yourSchema);

// find each person with a last name matching 'Ghost'
const query = Person.findOne({ 'name.last': 'Ghost' });

// selecting the `name` and `occupation` fields
query.select('name occupation');

// execute the query at a later time
query.exec((err, person) => {
    if (err) return handleError(err);
    // Space Ghost is a talk show host.
    console.log('%s %s is a %s.', person.name.first,
 person.name.last, person.occupation)
})
```

MONGOOSE FUNDAMENTALS

Using Query: build up a query using chaining syntax

```
// With a JSON doc
Person.find({
    occupation: /host/,
    'name.last': 'Ghost',
    age: {$gt: 17, $lt: 66},
    likes: {$in: ['vaporizing', 'talking']}
}).limit(10)
  .sort({occupation: -1})
  .select({name: 1, occupation: 1})
  .exec(callback);
```

MONGOOSE FUNDAMENTALS

Using Query: build up a query using chaining syntax

```
// Using query builder
Person
    .find({occupation: /host/})
    .where('name.last').equals('Ghost')
    .where('age').gt(17).lt(66)
    .where('likes').in(['vaporizing', 'talking'])
    .limit(10)
    .sort('-occupation')
    .select('name occupation')
    .exec(callback);
```

MONGOOSE FUNDAMENTALS

Built-in Validators

```
const breakfastSchema = new Schema({
    eggs: {
        type: Number,
        min: [6, 'Too few eggs'],
        max: 12
    },
    bacon: {
        type: Number,
        required: [true, 'Why no bacon?']
    },
    drink: {
        type: String,
        enum: ['Coffee', 'Tea']
    }
});
const Breakfast = db.model('Breakfast', breakfastSchema);
```

MONGOOSE FUNDAMENTALS

Custom Validators

```
const userSchema = new Schema({
    phone: {
        type: String,
        validate: {
            validator: v => /\d{3}-\d{3}-\d{4}/.test(v),
            message: '{VALUE} is not a valid phone number!'
        },
        required: [true, 'User phone number required']
    }
});
```

MONGOOSE FUNDAMENTALS

Middleware

```
const schema = new Schema(..);
schema.pre('save', function(next) {
    // do stuff
    next();
});
```

THANKS!