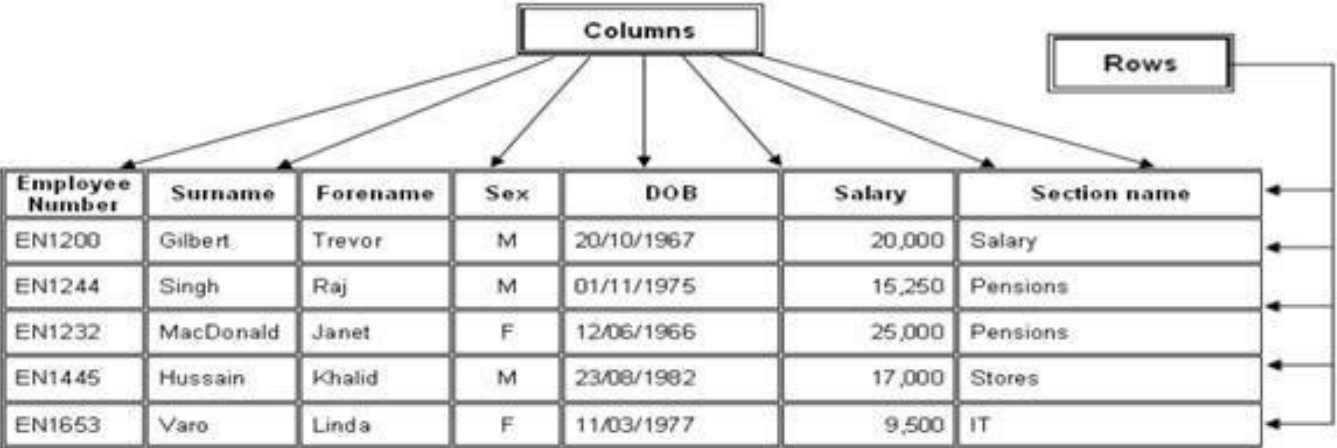# MongoDB and Mongoose

MongoDB is a variant of what is called a document-oriented database, part of a larger class of databases categorized as NoSQL, or Not-only-SQL. MongoDB takes a very different approach than the traditional, table-based, relational database structure in favor of JSON-like documents.

To provide context, a traditional RDBMS (Relational Database Management System) is one based on relationships defined by schemas. Data is stored in tables with rows and columns, and the data format must strictly adhere to that format.

Table example with rows and columns

This isn't how you typically model data in your application. This forces the use of an additional abstraction layer to map your data to how it will be stored in the database.

In MongoDB there is no schema to define - there are no tables and no relationships between collections of objects. Every document in Mongo can be as flat and simple or as deeply nested and complex as your application requires.

Connecting to MongoDB is straightforward and simple:

```javascript
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect('mongodb://localhost:27017/exampldDb', function (error, db) {
  if (!error) {
    console.log('We are connected');
  }
});
```

Because MongoDB is schema-less, the responsibility of schema enforcement is pushed to the application. This is where Mongoose comes in.

We'll be using Mongoose to easily store data in and retrieve documents from a MongoDB database in a structured manner. Even with Mongoose, we'll still have the flexibility to store whatever we want in Mongo, but Mongoose is designed to help structure our access to it.

Mongoose is what's known as a Object Document Mapper. With a relational database, for example, MySQL, you would use what's called an Object Relational Mapper (ORM) to allow you to interface with the database. To interface with MongoDB, a document-based database, you can use an ODM. In the case of MongoDB, it's not necessary, but it provides validations, associations, and other high-level data modelling functions.

Mongoose is an ODM for Node.js. It has a thriving open source community and includes advanced schema-based features such as async validation, casting, object life-cycle management, pseudo-joins, and rich query builder support.

- http://docs.mongodb.org/ecosystem/drivers/node-js/

First, installing MongoDB:

Installing MongoDB for MacOS and Windows

Importing some data into MongoDB

1. Clone the MongoDB-books repository locally
2. Make sure MongoDB is running
3. Go to the terminal/shell/command prompt and follow the instructions in the README

Installing Mongoose for our project

```
// mongoose is a project dependency, so inside our project directory
npm install --save mongoose
```

adding Mongoose to our project

```javascript
var express = require('express');

// mongoose is just another Node module
var mongoose = require('mongoose');
var bodyParser = require('body-parser');

var app = express();

// connect to the bookAPI database, creating it if it doesn't exist
var db = mongoose.connect('mongodb://localhost/bookAPI');
```

defining a model for Mongoose to use

```javascript
var express = require('express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');

// Book is the Mongoose model we'll use for our data
var Book = require('./models/bookModel');

var app = express();
var db = mongoose.connect('mongodb://localhost/bookAPI');
```

inside models/bookModel.js we'll define what structure
our model will use by employing another Mongoose
construct called a schema

```javascript
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// the schema describes what the data will look like
var bookSchema = new Schema({
    title: String,
    author: String,
    genre: String,
    read: {
        type: Boolean,
        default: false
    }
});
```

if you need to use "type" as a property of your schema,
set "type" to an object with a type value itself, otherwise
it will interpret the type of the entire schema

```javascript
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// the schema describes what the data will look like
var bookSchema = new Schema({
    title: String,
    author: String,
    genre: String,
    read: {
        type: Boolean,
        default: false
    },
    type: { type: String }
});
```

Mongoose schema allowed data types:

Mongoose Schema Type	JavaScript Data Type
String	String
Number	Number
Date	Object
Buffer	Object
Boolean	Boolean
Mixed	Object
ObjectId	Object
Array	Array (Object)

we'll export a 'Book' model based on this schema:

```javascript
var mongoose = require('mongoose');
var Schema = mongoose.Schema();

// the schema we'll use for the 'Book' model
var bookSchema = new Schema({
    title: {
        type: String,
        required: true
    },
    author: String,
    genre: String,
    read: { type: Boolean, default: false  }
});

module.exports = mongoose.model('Book', bookSchema);
```

now in app.js we have an instance of that bookModel as 'Book'

```javascript
var express = require('express');
var app = express();
var mongoose = require('mongoose');
var db = mongoose.connect('mongodb://localhost/bookAPI');
var bodyParser = require('body-parser');

// Book is the Mongoose model we'll use for our data
var Book = require('./models/bookModel');
```

mongoose models have helper methods, for example, "find"

```javascript
var express = require('express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');

// Book is the Mongoose model we'll use for our data
var Book = require('./models/bookModel');

var app = express();
var db = mongoose.connect('mongodb://localhost/bookAPI');

// when find is called without a callback parameter, it returns a query
// object that does not immediately connect to the MongoDB database
var query = Book.find();
```

calling find on our model and passing a
callback executes the query immediately

```
Book.find(function (error, books) {
  if (error) {
    response.status(500).send(error);
  } else {
    response.json(books);
  }
});
```

for our GET route for /books, we'll now
retrieve the data from our database

```javascript
app.get('/books', function (request, response) {

  // find called with a callback executes immediately
  Book.find(function (error, books) {
    if (error) {
        response.status(500).send(error);
    } else {
        response.json(books);
    }
  });
});
```

qualifying our search:

```javascript
app.get('/books', function (request, response) {

  // find called with a callback executes immediately
  Book.find({ author: 'Hugh Howey' }, function (error, books) {
    if (error) {
        response.status(500).send(error);
    } else {
        response.json(books);
    }
  });
});
```

qualifying returned fields:

```javascript
app.get('/books', function (request, response) {

  // find called with a callback executes immediately
  Book.find({ author: 'Hugh Howey' }, 'title genre', function (error, books) {
    if (error) {
        response.status(500).send(error);
    } else {
        response.json(books);
    }
  });
});
```

to query on specific Book properties, we can use the querystring

```javascript
// inside our /books route we can now retrieve data from our database
app.get('/books', function (request, response) {

  // the contents of the query-string is a hash on the request object
  var query = request.query;

  // that object is already formatted for the first parameter to find
  Book.find(query, function (error, books) {
    if (error) {
      response.status(500).send(error);
    } else {
      response.json(books);
    }
  });
});
```

sanitizing the query

```javascript
app.get('/books', function (request, response) {

  // control what kinds of data can be queried
  // via the query-string
  var query = {};

  if (request.query.genre) {
    query.genre = request.query.genre;
  }
  if (request.query.author) {
    query.author = request.query.author;
  }
  if (request.query.title) {
    query.title = request.query.title;
  }

  Book.find(query, function (error, books) {
    if (error) {
        response.status(500).send(error);
    } else {
        response.json(books);
    }
  });
});
```

retrieving a single document with findOne

```javascript
// like with find, calling without a callback defers execution
var query = Book.findOne({ author: 'Frank Herbert' });

// execution takes a callback of the same format as if it
// was passed directly to the findOne call
query.exec(function (error, results) {
    // do stuff
});
```

retrieving a single book with findById

```javascript
app.get('/books/:bookId', function (request, response) {

    // use the findById method available on the mongoose model
    // the id is the only parameter it takes (other than a callback)
    Book.findById(request.params.bookId, function (error, book) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.json(book);
        }
    });
});
```

another way to quality returned fields

```javascript
app.get('/books/:bookId', function (request, response) {

    // the second parameter indicates to return all fields but
    // the one specified with a '-'
    Book.findById(request.params.bookId, '-_id', function (error, book) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.json(book);
        }
    });
});
```

comparison query operators

operator	meaning
$gt	greater than
$gte	greater than or equal to
$in	exists in
$lt	less than
$lte	less than or equal to
$ne	not equal to
$nin	does not exist

using comparison query operators

```javascript
Book.find({ genre: { $ne: 'Fiction' }, function (error, book) {
    if (error) {
        response.status(500).send(error);
    } else {
        response.json(book);
    }
});
```

adding a new book

```
app.post('/books', function (request, response) {

    // the request.body is the new book object
    var book = new Book(request.body);

    book.save(function (error) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.status(201).send(book);
        }
    });
});
```

two approaches to updating an existing book

```javascript
app.put('/books/:bookId', function (request, response) {

    // like the GET route, use the findById method on the mongoose model
    Book.findById(request.params.bookId, function (error, book) {
        if (error) {
            response.status(500).send(error);
        } else {
            book.title = request.body.title;
            book.author = request.body.author;
            book.genre = request.body.genre;
            book.read = request.body.read;
            book.save(function (error) {
                if (error) {
                    response.status(500).send(error);
                } else {
                    response.send(book);
                }
            });
        }
    });
});
```

single call to updating an existing book

```
app.put('/books', function (request, response) {
    var condition = { author: 'Hugh Howey' };
    var update = { genre: request.body.genre };

    Book.update(condition, update, function (error, affected, book) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.send(book);
        }
    });
});
```

single call to updating an existing book by id

```javascript
app.put('/books/:bookId', function (request, response) {
    var bookId = request.params.bookId;
    var update = {
        title: request.body.title,
        author: request.body.author,
        genre: request.body.genre,
        read: request.body.read
    };

    Book.findByIdAndUpdate(bookId, update, function (error, affected, book) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.send(book);
        }
    });
});
```

two ways to delete a book

```javascript
app.delete('/books', function (request, response) {
    var condition = { genre: 'Fiction' };

    book.remove(condition, function (error) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.status(204).send('removed');
        }
    });
});
```

delete a book by Id

```javascript
app.delete('/books/:bookId', function (request, response) {
    var bookId = request.params.bookId;

    book.findByIdAndRemove(bookId, function (error) {
        if (error) {
            response.status(500).send(error);
        } else {
            response.status(204).send('removed');
        }
    });
});
```