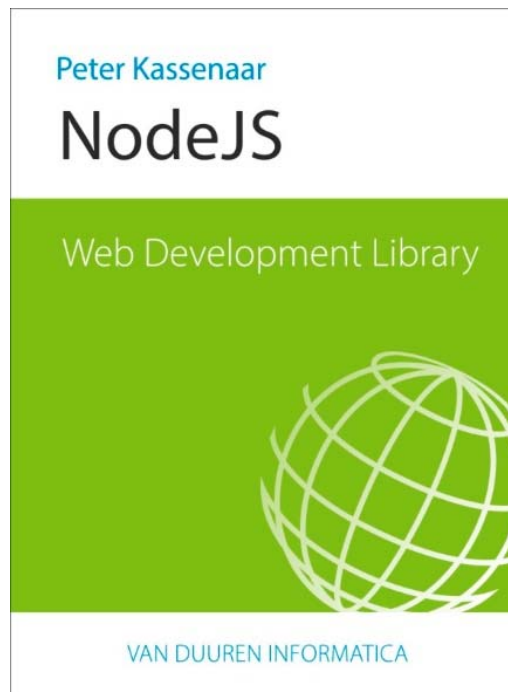


Node.js



Peter Kassenaar

Module 4 – MongoDB en Mongoose



Hoofdstuk 6, p.143 en verder

MongoDB - agenda

- Inleiding - SQL vs. NoSQL-databases
- Installatie & gebruik MongoDB 'los'.
- Handmatig CRUD-operations op MongoDB uitvoeren.
- Mongoose als Mongo-webinterface voor Node.js
 - Werken met schema's

SQL vs. NoSQL – de mythes

- NoSQL is een *alternatief* voor SQL-databases. Geen vervanging

MYTH: NoSQL supersedes SQL

That would be like saying boats were superseded by cars because they're a newer technology. SQL and NoSQL do the same thing: store data. They take different approaches, which may help or hinder your project. Despite feeling newer and grabbing recent headlines, NoSQL is not a replacement for SQL — *it's an alternative*.

<http://www.sitepoint.com/sql-vs-nosql-differences/>

“NoSQL is beter/slechter dan SQL”

- Hangt van je data & organisatie af

MYTH: NoSQL is better / worse than SQL

Some projects are better suited to using an SQL database. Some are better suited to NoSQL. Some could use either interchangeably. This article could never be a SitePoint Smackdown, because you cannot apply the same blanket assumptions everywhere.

“Er is duidelijk verschil tussen NoSQL- en SQL-databases”

- Ze nemen features van elkaar over. Er zijn al ‘NewSQL’, hybride, databases

MYTH: SQL vs NoSQL is a clear distinction

This is not necessarily true. Some SQL databases are adopting NoSQL features and vice versa. The choices are likely to become increasingly blurred, and NewSQL hybrid databases could provide some interesting options in the future.

“Het OS bepaalt de database”

MYTH: the language/framework determines the database

We've grown accustomed to technology stacks, such as —

- LAMP: Linux, Apache, MySQL (SQL), PHP
- MEAN: MongoDB (NoSQL), Express, Angular, Node.js
- .NET, IIS and SQL Server
- Java, Apache and Oracle.

There are practical, historical and commercial reasons why these stacks evolved — but don't presume they are rules. You can use a MongoDB NoSQL database in your [PHP](#) or [.NET](#) project. You can connect to [MySQL](#) or [SQL Server](#) in Node.js. You may not find as many tutorials and resources, but your requirements should determine the database type — *not the language*.

SQL tabellen vs NoSQL Documents

SQL databases provide a store of related data tables. For example, if you run an online book store, book information can be added to a table named `book`:

ISBN	title	author	format	price
9780992461225	JavaScript: Novice to Ninja	Darren Jones	ebook	29.00
9780994182654	Jump Start Git	Shaumik Daityari	ebook	29.00

Every row is a different book record. The design is rigid; you cannot use the same table to store different information or insert a string where a number is expected.

Traditionele SQL – tabelstructuur

NoSQL Documents

```
{
  ISBN: 9780992461225,
  title: "JavaScript: Novice to Ninja",
  author: "Darren Jones",
  format: "ebook",
  price: 29.00
}
```

Similar documents can be stored in a **collection**, which is analogous to an SQL table. However, you can store any data you like in any document; the NoSQL database won't complain. For example:

```
{
  ISBN: 9780992461225,
  title: "JavaScript: Novice to Ninja",
  author: "Darren Jones",
  year: 2014,
  format: "ebook",
  price: 29.00,
  description: "Learn JavaScript from scratch!",
  rating: "5/5",
  review: [
    { name: "A Reader", text: "The best JavaScript book I've ever read." },
    { name: "JS Expert", text: "Recommended to novice and expert developers alike." }
  ]
}
```

SQL Schema vs. NoSQL Schemaless

In an SQL database, it's impossible to add data until you define tables and field types in what's referred to as a *schema*. The schema optionally contains other information, such as —

- **primary keys** — unique identifiers such as the ISBN which apply to a single record
- **indexes** — commonly queried fields indexed to aid quick searching
- **relationships** — logical links between data fields
- functionality such as **triggers** and **stored procedures**.

SQL: alles upfront declareren

NoSQL: “just start using it”

In a NoSQL database, data can be added anywhere, at any time. There's no need to specify a document design or even a collection up-front. For example, in MongoDB the following statement will create a new document in a new `book` collection if it's not been previously created:

```
db.book.insert(  
  ISBN: 9780954182654,  
  title: "Jump Start Git",  
  author: "Shaumik Daityari",  
  format: "ebook",  
  price: 29.00  
);
```

(MongoDB will automatically add a unique `_id` value to each document in a collection. You may still want to define indexes, but that can be done later if necessary.)

Normalization

- SQL: Koppeltabellen, gemeenschappelijke velden

id	name	country	email
SP001	SitePoint	Australia	feedback@sitepoint.com

We can then add a `publisher_id` field to our `book` table, which references records by `publisher.id`:

ISBN	title	author	format	price	publisher_id
9780992461225	JavaScript: Novice to Ninja	Darren Jones	ebook	29.00	SP001
9780994182654	Jump Start Git	Shaumik Daityari	ebook	29.00	SP001

NoSQL: duplicatie van data in één document

```
{  
  ISBN: 9780992461225,  
  title: "JavaScript: Novice to Ninja",  
  author: "Darren Jones",  
  format: "ebook",  
  price: 29.00,  
  publisher: {  
    name: "SitePoint",  
    country: "Australia",  
    email: "feedback@sitepoint.com"  
  }  
}
```

This leads to faster queries, but updating the publisher information in multiple records will be significantly slower.

SQL Join vs. NoSQL

- Geen JOIN-statement beschikbaar

```
SELECT book.title, book.author, publisher.name  
FROM book  
LEFT JOIN book.publisher_id ON publisher.id;
```

This returns all book titles, authors and associated publisher names (presuming one has been set).

NoSQL has no equivalent of JOIN, and this can shock those with SQL experience. If we used normalized collections as described above, we would need to fetch all `book` documents, retrieve all associated `publisher` documents, and manually link the two in our program logic. This is one reason denormalization is often essential.

SQL vs. NoSQL Data Integrity

SQL: Schema dwingt integriteit af, door id's en FK's

Niet beschikbaar in NoSQL

The same data integrity options are not available in NoSQL databases; you can store what you want regardless of any other documents. Ideally, a single document will be the sole source of all information about an item.

SQL vs. NoSQL Transactions

SQL: transacties om conflicten tussen tabellen te voorkomen

NoSQL: niet beschikbaar

In a NoSQL database, modification of a single document is atomic. In other words, if you're updating three values within a document, either all three are updated successfully or it remains unchanged. However, there's no transaction equivalent for updates to multiple documents. There are [transaction-like options](#), but, at the time of writing, these must be manually processed in your code.

SQL vs. NoSQL CRUD Syntax

SQL: SQL-statements met gereserveerde keywords (SELECT, INSERT INTO, WHERE, SET, enzovoort)

NoSQL: JavaScript-like syntax.

SQL	NoSQL
insert a new book record	
<pre>INSERT INTO book (`ISBN`, `title`, `author`) VALUES ('9780992461256', 'Full Stack JavaScript', 'Colin Ihrig & Adam Bretz');</pre>	<pre>db.book.insert({ ISBN: "9780992461256", title: "Full Stack JavaScript", author: "Colin Ihrig & Adam Bretz" });</pre>

Update & Retrieve

update a book record

```
UPDATE book
SET price = 19.99
WHERE ISBN = '9780992461256'
```

```
db.book.update(
  { ISBN: '9780992461256' },
  { $set: { price: 19.99 } }
);
```

return all book titles over \$10

```
SELECT title FROM book
WHERE price > 10;
```

```
db.book.find(
  { price: { >: 10 } },
  { _id: 0, title: 1 }
);
```

The second JSON object is known as a **projection**: it sets which fields are returned (`_id` is returned by default so it needs to be unset).

Count & Delete

count the number of SitePoint books	
<pre>SELECT COUNT(1) FROM book WHERE publisher_id = 'SP001';</pre>	<pre>db.book.count({ "publisher.name": "SitePoint" });</pre> <p>This presumes denormalized documents are used.</p>
delete all SitePoint books	
<pre>DELETE FROM book WHERE publisher_id = 'SP001';</pre> <p>Alternatively, it's possible to delete the <code>publisher</code> record and have this cascade to associated <code>book</code> records if foreign keys are specified appropriately.</p>	<pre>db.book.remove({ "publisher.name": "SitePoint" });</pre>

MongoDB – 2 delen

Alternatieven: CouchDB, Cassandra, Riak.

Globale werking:



Mongo Shell

Opdrachtregelomgeving,
CLI, `mongo.exe`



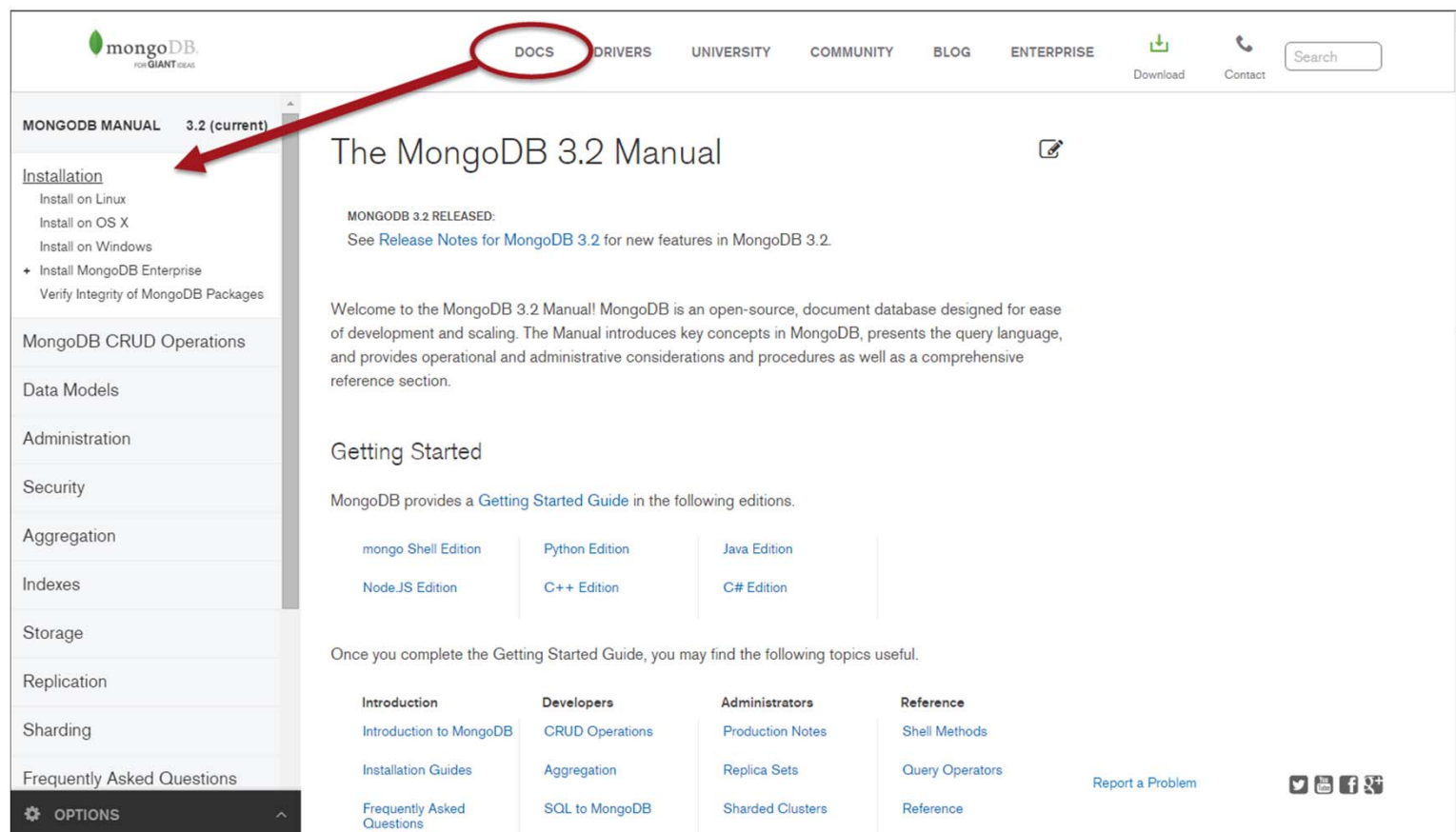
Mongo
Deamon

Databaseserver,
`mongod.exe`



p.143

Installatie



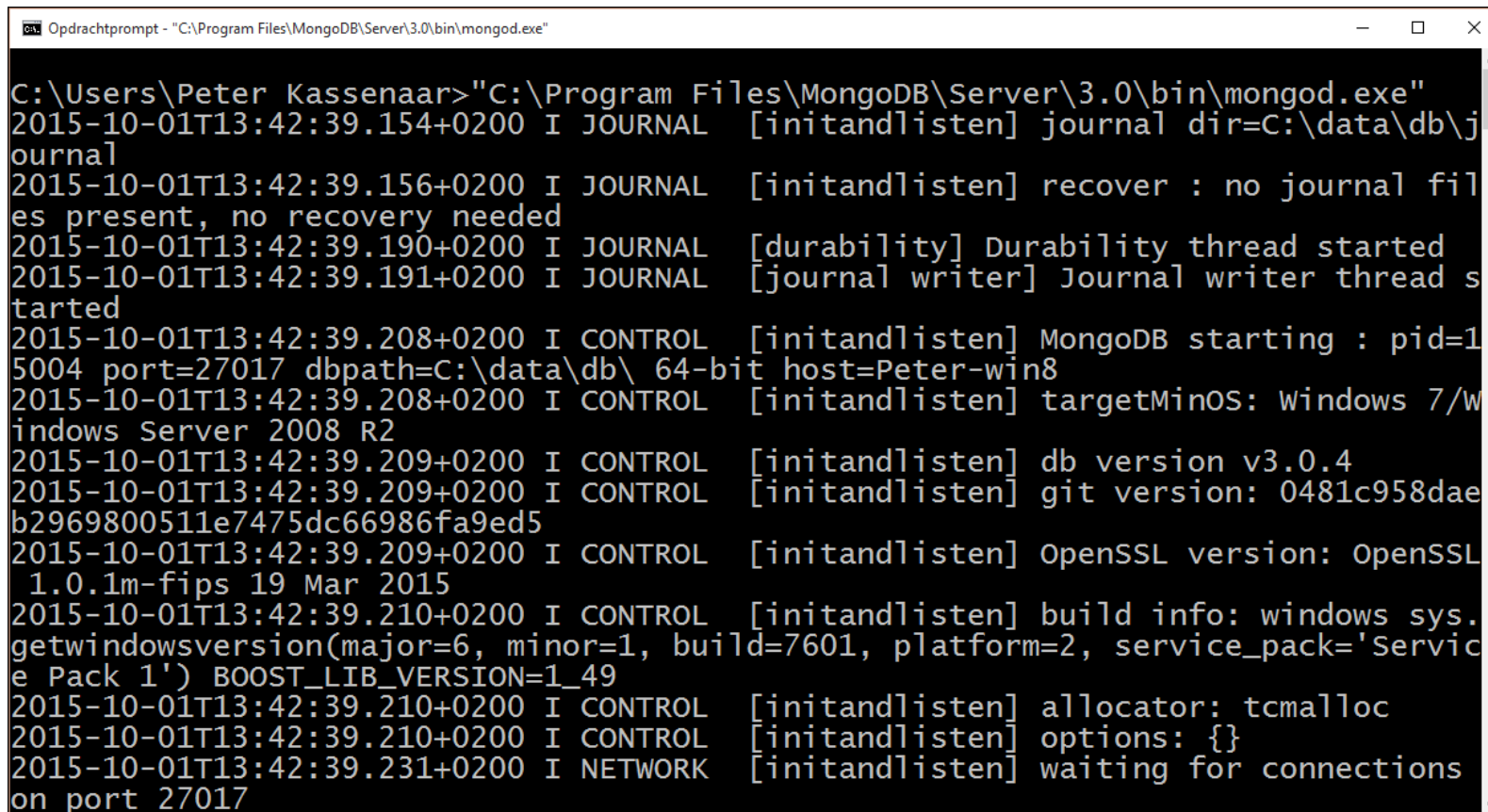
<https://docs.mongodb.org>

Data-directory

- Maak directory C:\Data\db voor opslag van MongoDB-databases.
- Op Mac: idem (maak directory \data\db)
- Lees de installatie-instructies per platform goed.
 - Lees ze nog een keer!

MongoDB Starten – 1. Deamon

"C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe"

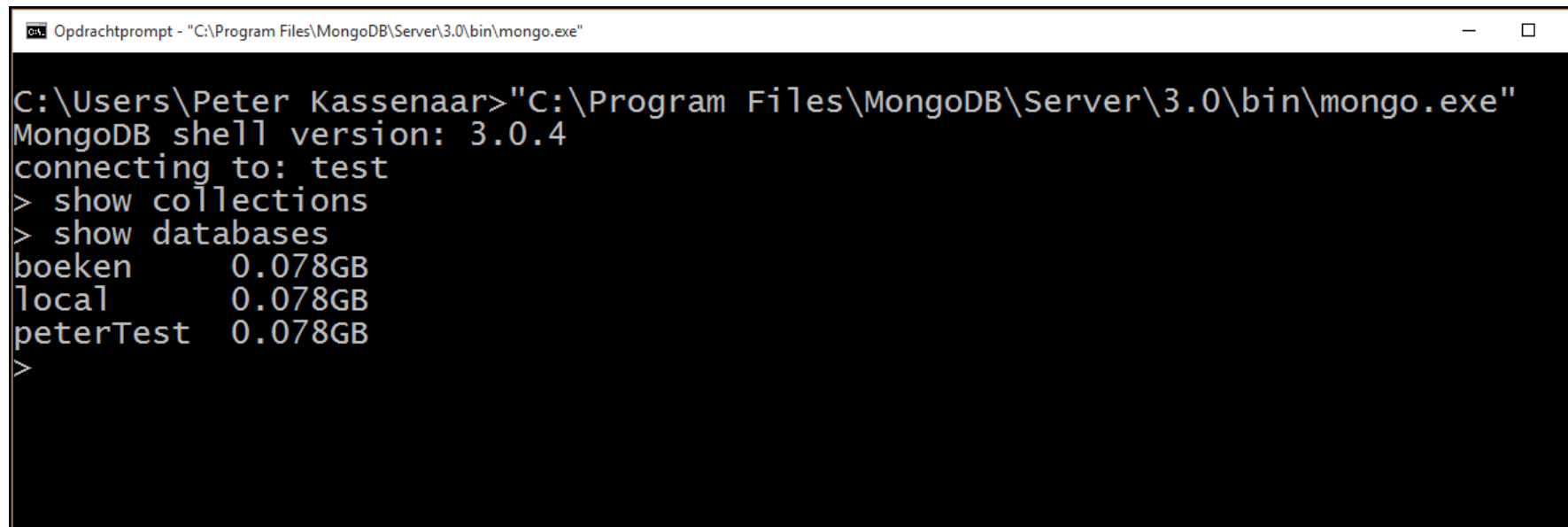


```
Opdrachtprompt - "C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe"

C:\Users\Peter Kassenaar>"C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe"
2015-10-01T13:42:39.154+0200 I JOURNAL [initandlisten] journal dir=C:\data\db\j
ournal
2015-10-01T13:42:39.156+0200 I JOURNAL [initandlisten] recover : no journal fil
es present, no recovery needed
2015-10-01T13:42:39.190+0200 I JOURNAL [durability] Durability thread started
2015-10-01T13:42:39.191+0200 I JOURNAL [journal writer] Journal writer thread s
tarted
2015-10-01T13:42:39.208+0200 I CONTROL [initandlisten] MongoDB starting : pid=1
5004 port=27017 dbpath=C:\data\db\ 64-bit host=Peter-win8
2015-10-01T13:42:39.208+0200 I CONTROL [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2015-10-01T13:42:39.209+0200 I CONTROL [initandlisten] db version v3.0.4
2015-10-01T13:42:39.209+0200 I CONTROL [initandlisten] git version: 0481c958dae
b2969800511e7475dc66986fa9ed5
2015-10-01T13:42:39.209+0200 I CONTROL [initandlisten] OpenSSL version: OpenSSL
1.0.1m-fips 19 Mar 2015
2015-10-01T13:42:39.210+0200 I CONTROL [initandlisten] build info: windows sys.
getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Servic
e Pack 1') BOOST_LIB_VERSION=1_49
2015-10-01T13:42:39.210+0200 I CONTROL [initandlisten] allocator: tcmalloc
2015-10-01T13:42:39.210+0200 I CONTROL [initandlisten] options: {}
2015-10-01T13:42:39.231+0200 I NETWORK [initandlisten] waiting for connections
on port 27017
```

MongoDB Starten – 2. Shell

"C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe"

A screenshot of a Windows command prompt window. The title bar reads "Opdrachtprompt - \"C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe\"". The command prompt shows the user running the command "C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe". The output shows the MongoDB shell version (3.0.4) and the connection to the 'test' database. The user then enters the commands 'show collections' and 'show databases'. The output for 'show databases' lists three databases: 'boeken' (0.078GB), 'local' (0.078GB), and 'peterTest' (0.078GB). The prompt is currently at the '>' line.

```
Opdrachtprompt - "C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe"

C:\Users\Peter Kassenaar>"C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe"
MongoDB shell version: 3.0.4
connecting to: test
> show collections
> show databases
boeken      0.078GB
local       0.078GB
peterTest   0.078GB
>
```


Using the CLI

Database maken/gebruiken:

```
use databaseNaam
```

Aanwezige collections tonen:

```
show collections
```

Nieuwe collection maken en document invoegen

```
db.users.insert({ name: 'Peter Kassenaar' });
```

Gegevens toevoegen

```
> db.users.find()  
> db.users.insert({name: 'Peter Kassenaar'});  
writeResult({ "nInserted" : 1 })  
> █
```

Veelgebruikte opdrachten:

```
db.<collectie>.find()  
db.<collectie>.insert()  
db.<collectie>.update()  
db.<collectie>.remove()
```

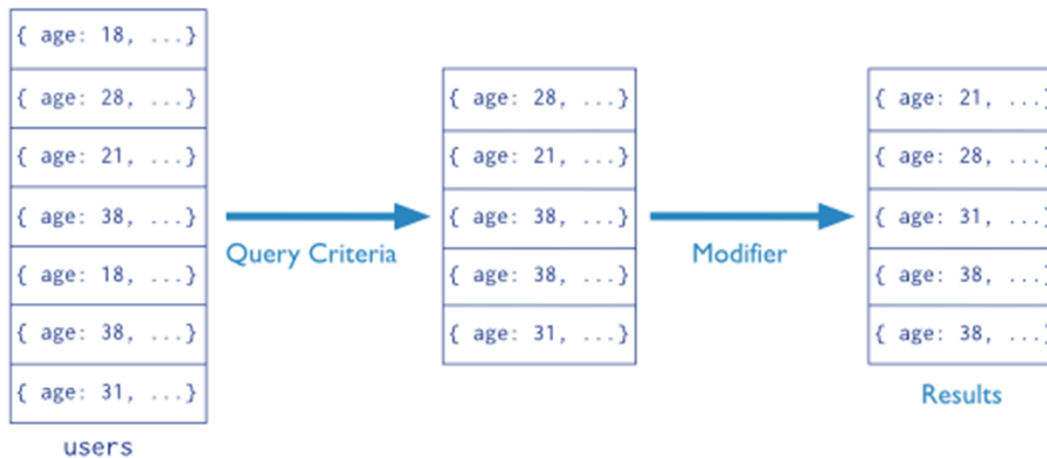
Ingewikkelder queries:

Query

In MongoDB a query targets a specific collection of documents. Queries specify criteria, or conditions, that identify the documents that MongoDB returns to the clients. A query may include a *projection* that specifies the fields from the matching documents to return. You can optionally modify queries to impose limits, skips, and sort orders.

In the following diagram, the query process specifies a query criteria and a sort modifier:

Collection Query Criteria Modifier
`db.users.find({ age: { $gt: 18 } }).sort({age: 1 })`



Meer over CRUD

MongoDB CRUD Introduction

MongoDB stores data in the form of *documents*, which are JSON-like field and value pairs. Documents are analogous to structures in programming languages that associate keys with values (e.g. dictionaries, hashes, maps, and associative arrays). Formally, MongoDB documents are *BSON* documents. BSON is a binary representation of *JSON* with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents. For more information, see [Documents](#).

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

MongoDB stores all documents in *collections*. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases.



Collection

<https://docs.mongodb.org/manual/core/crud-introduction/>

Database verwijderen

```
use <databasenaam>
```

```
db.dropDatabase( )
```

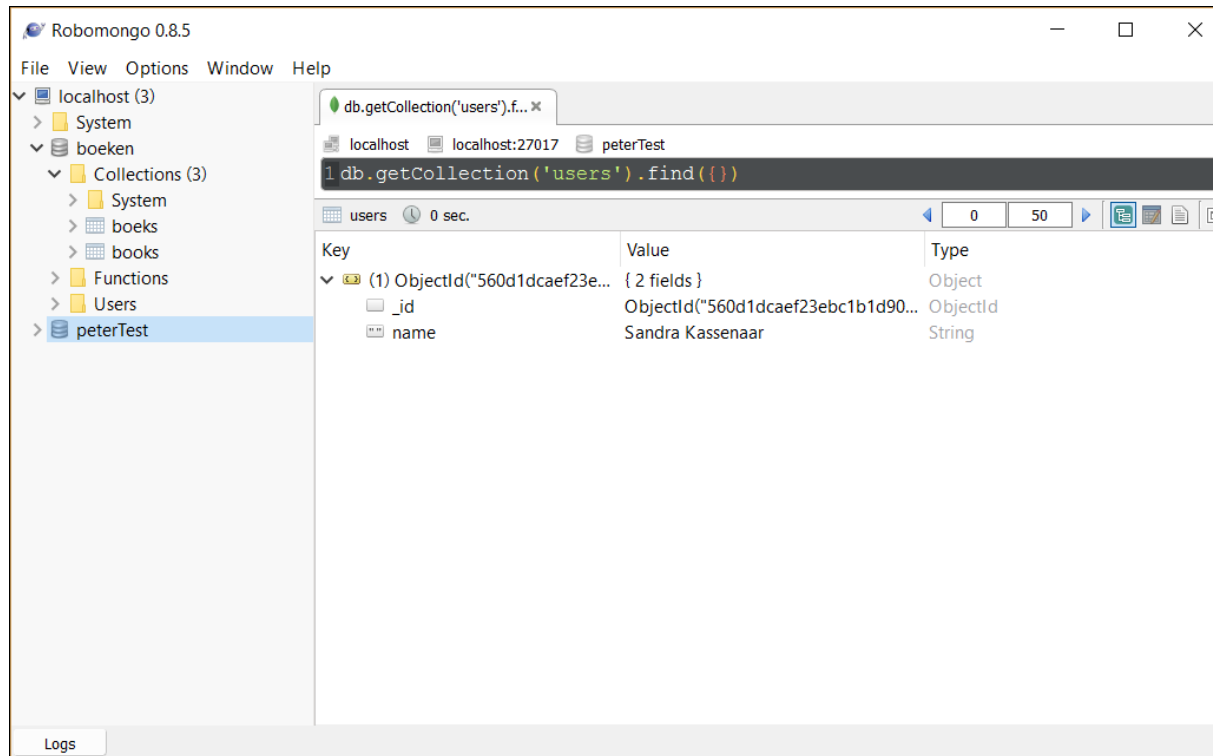
Query and Data Manipulation Collection Methods

Name	Description
<code>db.collection.count()</code>	Wraps <code>count</code> to return a count of the number of documents in a collection or matching a query.
<code>db.collection.distinct()</code>	Returns an array of documents that have distinct values for the specified field.
<code>db.collection.find()</code>	Performs a query on a collection and returns a cursor object.
<code>db.collection.findOne()</code>	Performs a query and returns a single document.
<code>db.collection.insert()</code>	Creates a new document in a collection.
<code>db.collection.remove()</code>	Deletes documents from a collection.
<code>db.collection.save()</code>	Provides a wrapper around an <code>insert()</code> and <code>update()</code> to insert new documents.
<code>db.collection.update()</code>	Modifies a document in a collection.

Meer referentie

Robomongo – visuele shell

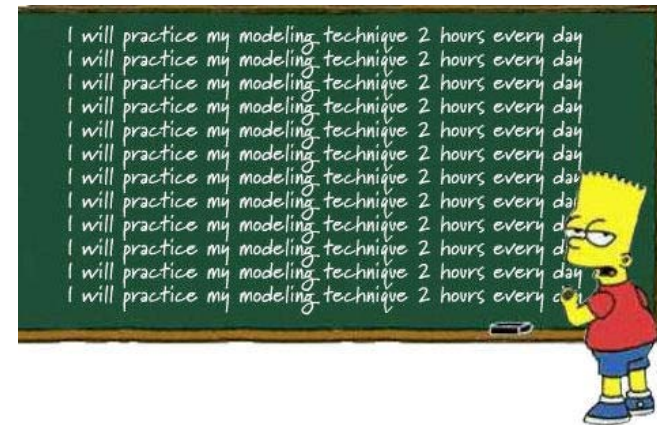
- Een IDE voor het werken met MongoDB
 - <http://robomongo.org/>

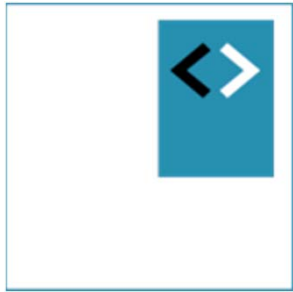


Checkpoint

- MongoDB is de meest gebruikte NoSQL-database in combinatie met Node.js.
- Installeer per platform, start Shell en CLI
- Leer de NoSQL CRUD-operations
- Gebruik eventueel Robomongo

Oefeningen...





Mongoose

Middleware layer voor Node.js, Express en MongoDB

Mongoose

"Mongoose provides a straight-forward, schema-based solution to model your application data"

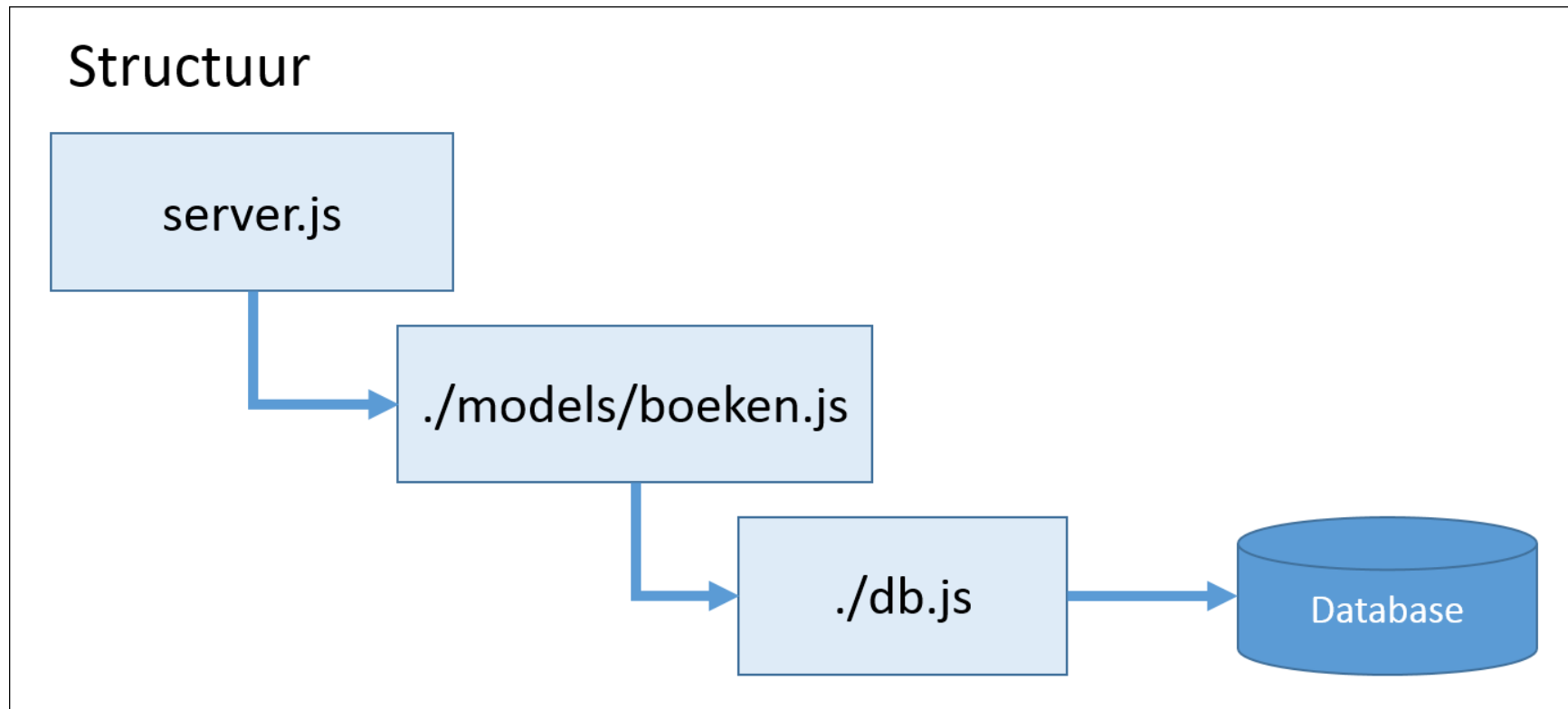


<http://mongoosejs.com/index.html>

ODM – Object Data Mapper

- `npm install mongoose --save`
- Modulair denken: meerdere lagen in model:
 - Centrale connectie met database: `db.js`
 - Model: `<entityName>.js`
 - Insluiten in applicatie: `require ('<entityName>.js')`

Architectuur mongoose



In code – connectie met database

//db.js - Logica voor verbinden met MongoDB

```
var mongoose = require('mongoose');  
var db = mongoose.connect('mongodb://localhost/boeken', function () {  
    console.log('mongoose connected');  
});  
  
module.exports = db;
```

Model

```
// boeken.js : Model voor boeken in MongoDB-database  
var db = require('../db');  
var Boek = db.model('Boek', {  
  titel : {type: String, required: true},  
  auteur: {type: String, required: true},  
  ISBN   : {type: String, required: true},  
  date   : {type: Date, required: true, default: Date.now}  
});  
  
module.exports = Boek;
```

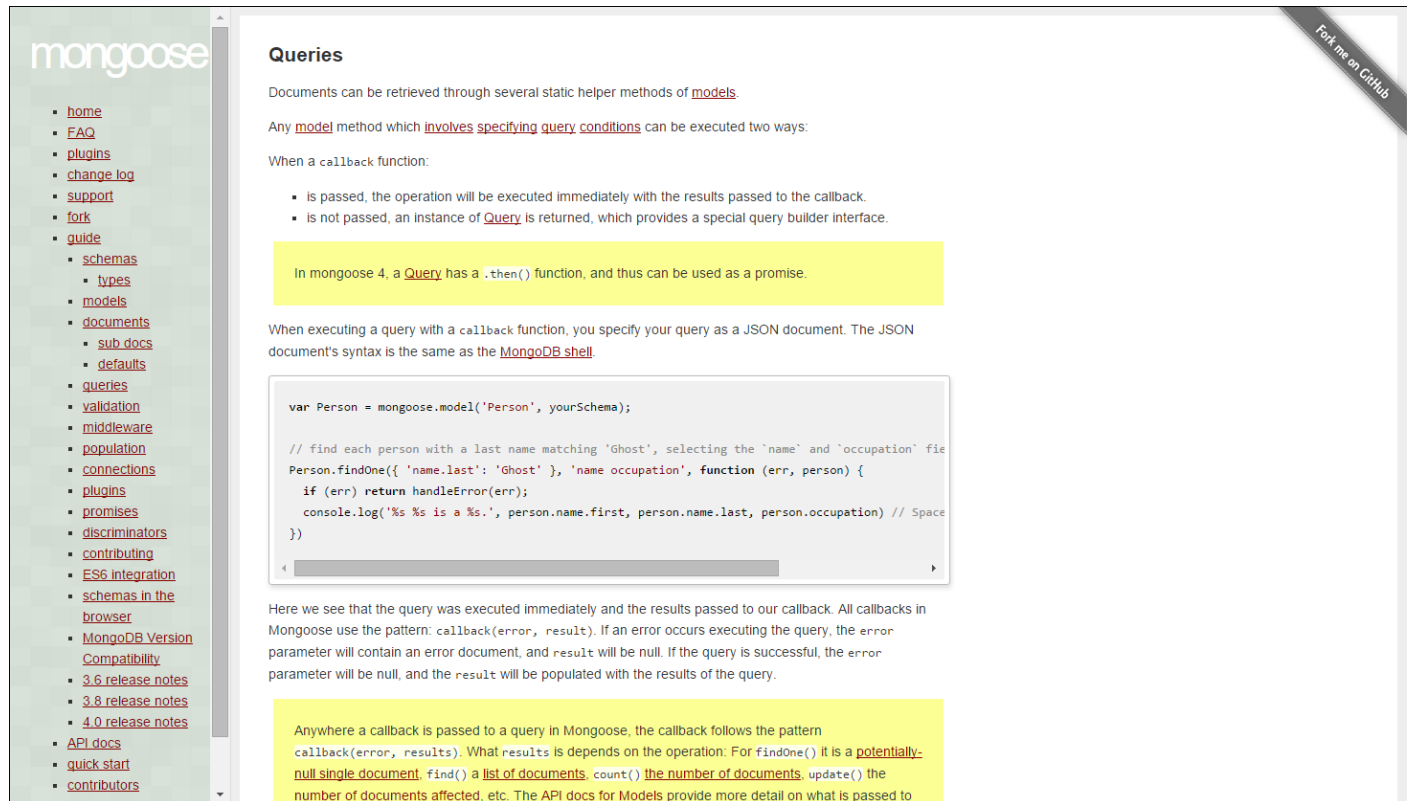
<http://mongoosejs.com/docs/schematypes.html>

Applicatie

```
// server.js - applicatie voor het ophalen en opslaan van boeken in MongoDB
var express = require('express');
var bodyParser = require('body-parser');
var Boek = require('./models/boeken');
...
// 2. POST-endpoint: nieuw boek in de database plaatsen.
app.post('/api/boeken', function (req, res, next) {
  // 2a. nieuw boekobject maken.
  var boek = new Boek({
    titel : req.body.titel,
    auteur: req.body.auteur,
    ISBN  : req.body.isbn
  });
  // 2b. Opslaan in database.
  boek.save(function (err, boek) {
    // indien error: teruggeven
    if (err) {
      return next(err);
    }
    // indien OK: status 201 (Created) en boekobject teruggeven
    res.status(201).json(boek);
  })
});
```

Mongoose CRUD-operations

- Queries: get, find, findOne, limit, sort, where, remove en tal van andere



The screenshot shows the Mongoose website's documentation for Queries. On the left is a sidebar with a navigation menu. The main content area is titled 'Queries' and explains how to retrieve documents using static helper methods of models. It details the use of the `find` method with a callback function and the `findOne` method. A code block shows a query for a person with the last name 'Ghost'. A yellow highlight box states that in Mongoose 4, a `Query` has a `.then()` function and can be used as a promise. Another yellow highlight box explains the callback pattern: `callback(error, results)`, where `error` is null on success and `results` is the document or list of documents.

mongoose

- [home](#)
- [FAQ](#)
- [plugins](#)
- [change log](#)
- [support](#)
- [fork](#)
- [guide](#)
- [schemas](#)
- [types](#)
- [models](#)
- [documents](#)
- [sub docs](#)
- [defaults](#)
- [queries](#)
- [validation](#)
- [middleware](#)
- [population](#)
- [connections](#)
- [plugins](#)
- [promises](#)
- [discriminators](#)
- [contributing](#)
- [ES6 integration](#)
- [schemas in the browser](#)
- [MongoDB Version Compatibility](#)
- [3.6 release notes](#)
- [3.8 release notes](#)
- [4.0 release notes](#)
- [API docs](#)
- [quick start](#)
- [contributors](#)

Queries

Documents can be retrieved through several static helper methods of [models](#).

Any [model](#) method which [involves specifying query conditions](#) can be executed two ways:

When a callback function:

- is passed, the operation will be executed immediately with the results passed to the callback.
- is not passed, an instance of [Query](#) is returned, which provides a special query builder interface.

In mongoose 4, a [Query](#) has a `.then()` function, and thus can be used as a promise.

When executing a query with a callback function, you specify your query as a JSON document. The JSON document's syntax is the same as the [MongoDB shell](#).

```
var 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', function (err, person) {
  if (err) return handleError(err);
  console.log('%s %s is a %s.', person.name.first, person.name.last, person.occupation) // Space
})
```

Here we see that the query was executed immediately and the results passed to our callback. All callbacks in Mongoose use the pattern: `callback(error, result)`. If an error occurs executing the query, the `error` parameter will contain an error document, and `result` will be null. If the query is successful, the `error` parameter will be null, and the `result` will be populated with the results of the query.

Anywhere a callback is passed to a query in Mongoose, the callback follows the pattern `callback(error, results)`. What results is depends on the operation: For `findOne()` it is a [potentially-null single document](#), `find()` a [list of documents](#), `count()` [the number of documents](#), `update()` the [number of documents affected](#), etc. The [API docs for Models](#) provide more detail on what is passed to

Fork me on GitHub

```
app.post('/api/boeken', function (req, res, next) {  
  // 2a. nieuw boekobject maken.  
  var boek = new Boek({  
    titel : req.body.titel,  
    auteur: req.body.auteur,  
    ISBN  : req.body.isbn  
  });  
  // 2b. Opslaan in database.  
  boek.save(function (err, boek) {  
    // indien error: teruggeven  
    if (err) {  
      return next(err);  
    }  
    // indien OK: status 201 (Created) en boekobject teruggeven  
    res.status(201).json(boek);  
  })  
});  
  
app.delete('/api/boeken/:id', function (req, res, next) {  
  Boek.remove({_id: req.params.id}, function (err, removed) {  
    if (err) {  
      return next(err);  
    }  
    console.log('removed: ', removed);  
    res.status(200).json(removed);  
  })  
});
```


Checkpoint

- Mongoose dient als extra layer om het werken vanuit Node.js met MongoDB te vergemakkelijken
- In mongoose kun je wél het werken met een model/schema afdwingen
- Een instantie van het model heeft methodes voor CRUD-operations op de database

Oefeningen...

