

# Web Engineering II

## 08 Datenhaltung: mongoDB NoSQL

Johannes Konert



BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN

University of Applied Sciences



## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



Bildquelle:  
[http://static.zehn.de/image/3a2e5985e6bcc6c726abe9b2f9a1a335/1248105898.4.081fdc6b871a6e5f1134554af404e040.Sahnehaubchen\\_Fotolia\\_4150869\\_askaja.jpgthumb\\_187x141](http://static.zehn.de/image/3a2e5985e6bcc6c726abe9b2f9a1a335/1248105898.4.081fdc6b871a6e5f1134554af404e040.Sahnehaubchen_Fotolia_4150869_askaja.jpgthumb_187x141)



## Zusammenfassende Fragen

1. Welche **zwei wesentlichen Probleme** sollen durch Modularisierung gelöst werden? Wie gelingt das?
2. Was ist das **JavaScript Module Pattern**? Wie sieht die **Syntax** aus?
3. Wie gelingt die **Übergabe von Parametern** in das Modul hinein beim Modul-Pattern?
4. Wie kann eine Methode oder Eigenschaft vor dem **Zugriff** von außerhalb des Moduls **geschützt werden**, obwohl es kein `private` in JavaScript gibt?
5. Worin unterscheiden sich die Modularisierungslösungen **CommonJS** und **AMD**? Nennen Sie mindestens 2 Unterschiede.
6. Welche Modularisierungslösung kann **server-seitig** genutzt werden, welche **client-seitig**? Warum?
7. Bei welcher Modularisierungslösung stehen Ihnen die Funktionen `require(..)` und `define(..)` innerhalb eines Moduls zur Verfügung?
8. Wo liegt der Unterschied der Variablen `exports` und `module.exports` bei CommonJS?
9. Wie geben Sie bei einer Modul-Definition in **AMD-Stil** an, welche anderen Module geladen werden sollen, bevor ihr Code laufen kann?
10. Nach welchen Richtlinien sucht `require(..)` bei nodeJS die Module?
11. Welche Modularisierungslösung müssen Sie einsetzen, um **jQuery** damit verwenden (laden) zu können? Warum?
12. Welche Vor/Nachteile hat ein **Boilerplate Generator**?
13. Was ist die **Gemeinsamkeit** von `Object.create(...)` und einer Konstruktorfunktion, wie `Factory()` ?
14. Welche Vorteile hat **prototypische Vererbung**?

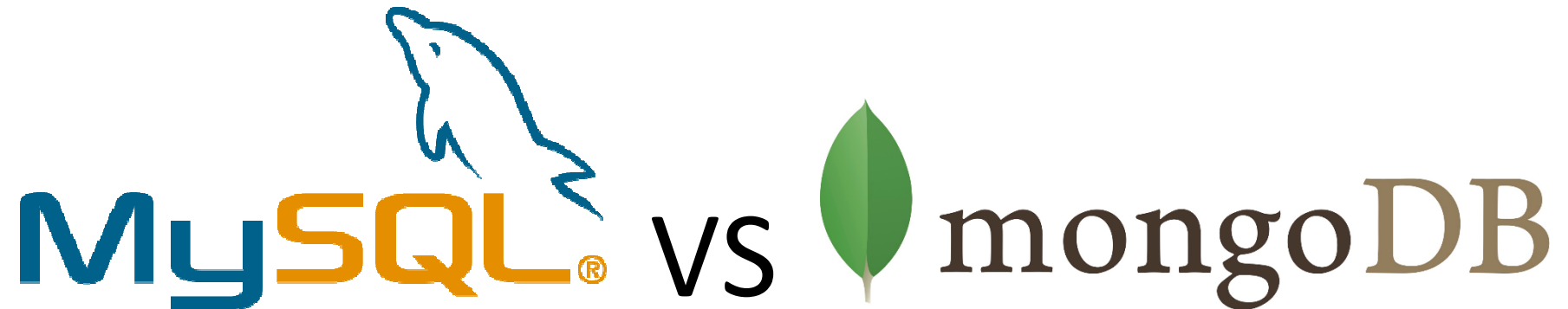
## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongodb**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_
  
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



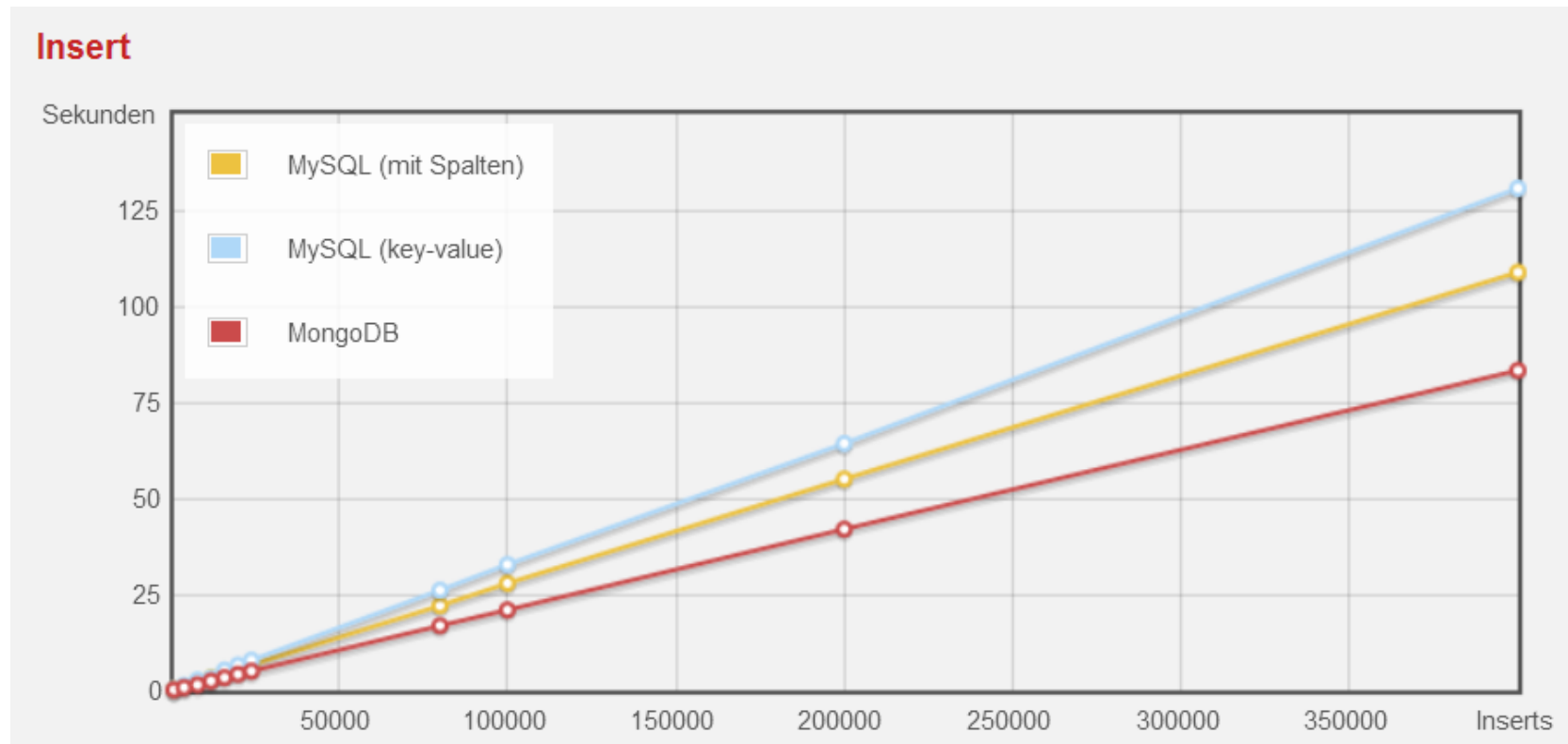
# Was heißt NoSQL?

## Not only SQL



# Performance

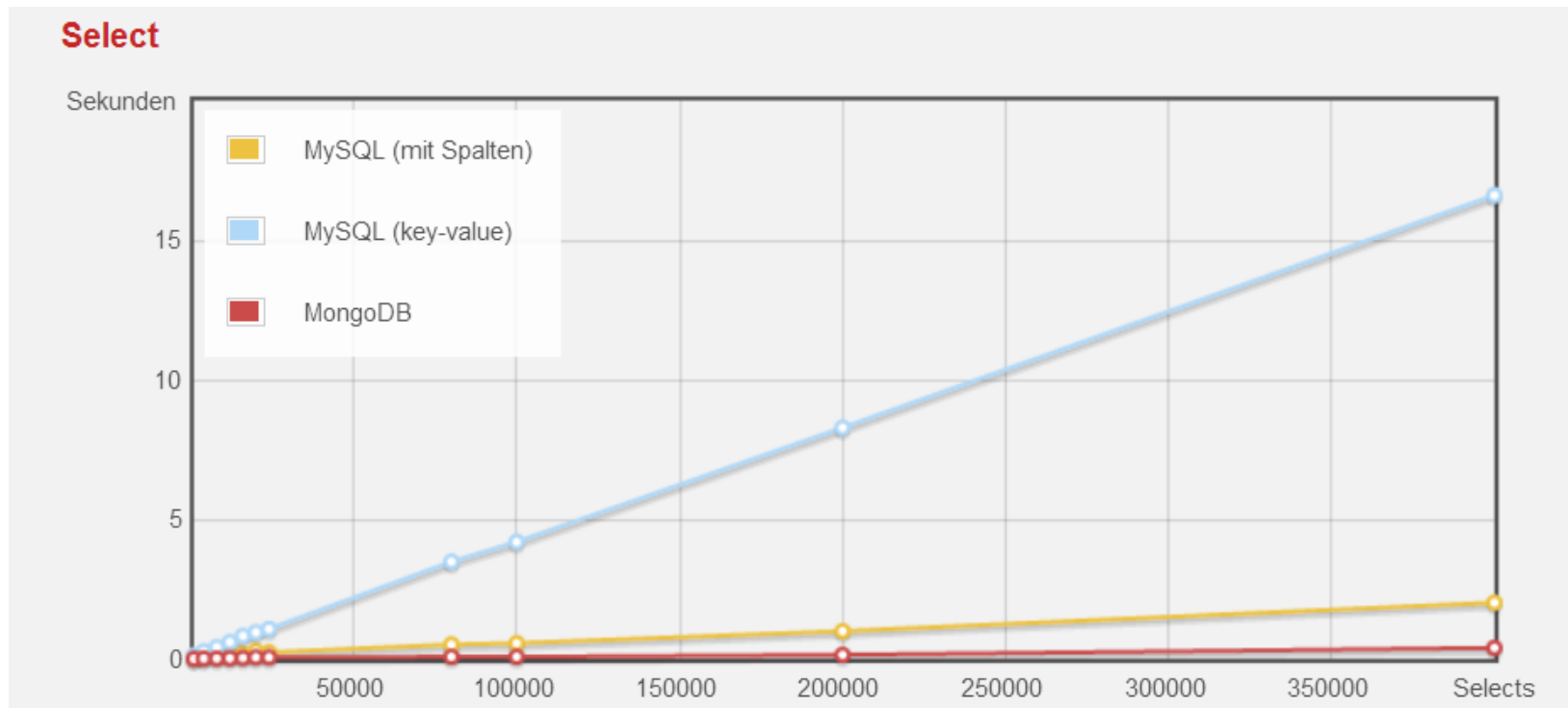
## NoSQL Performance



MongoDB 30,6% schneller als MySQL

<http://skowron.biz/artikel/mysql-mariadb-vs-mongodb/>

## NoSQL Performance



MongoDB 500% schneller als MySQL

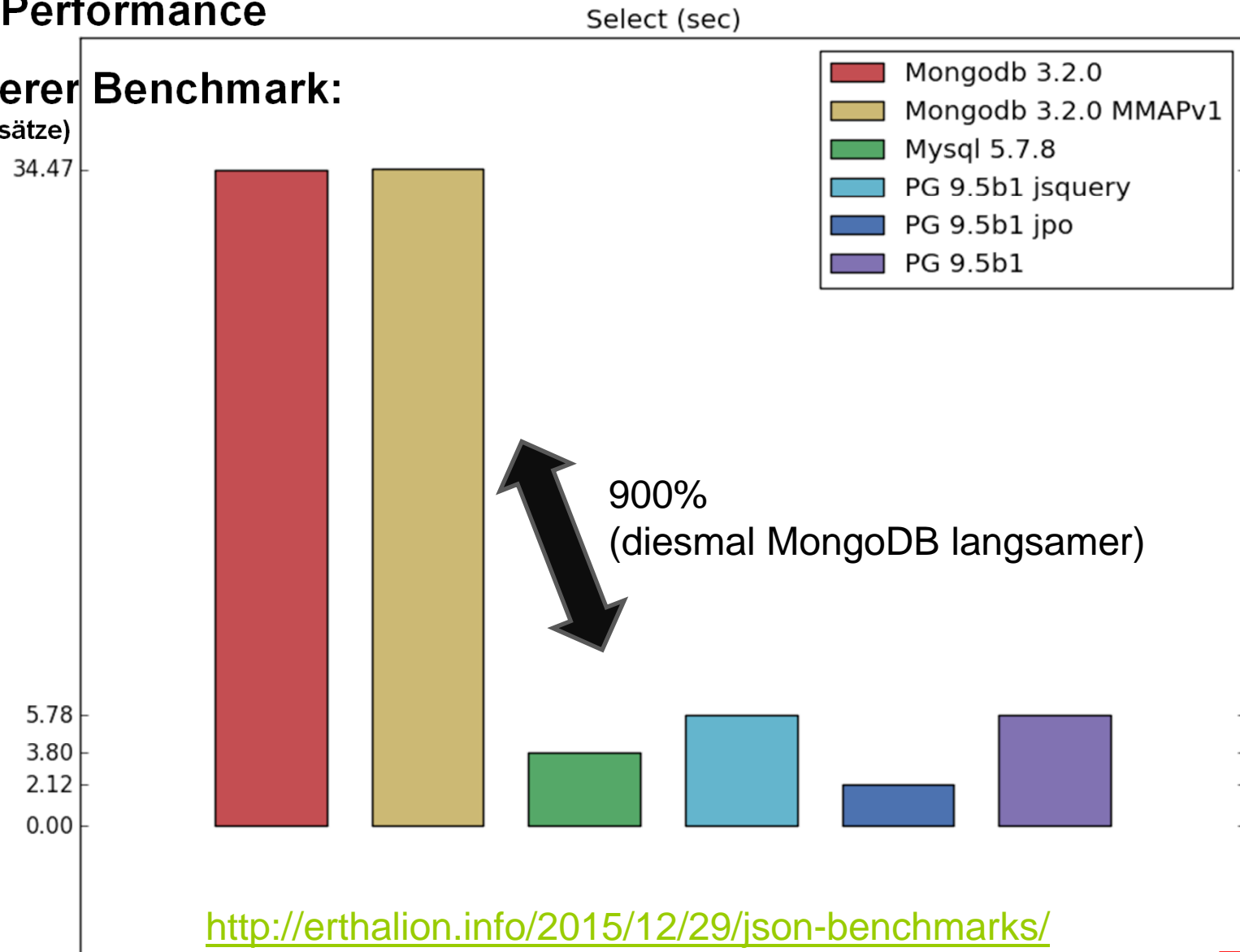
<http://skowron.biz/artikel/mysql-mariadb-vs-mongodb/>



## NoSQL Performance

### Ein anderer Benchmark:

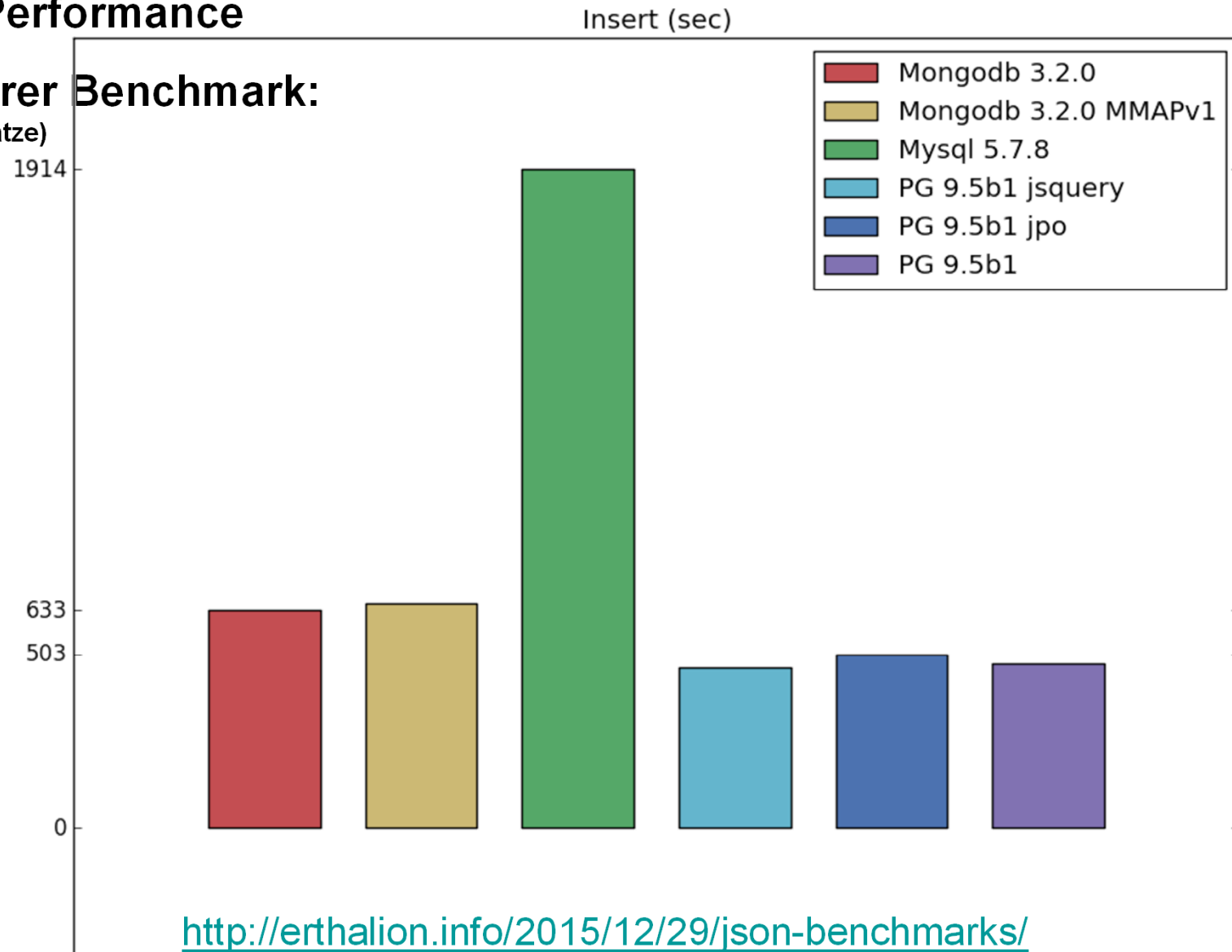
(1 Mio Datensätze)



## NoSQL Performance

### Ein anderer Benchmark:

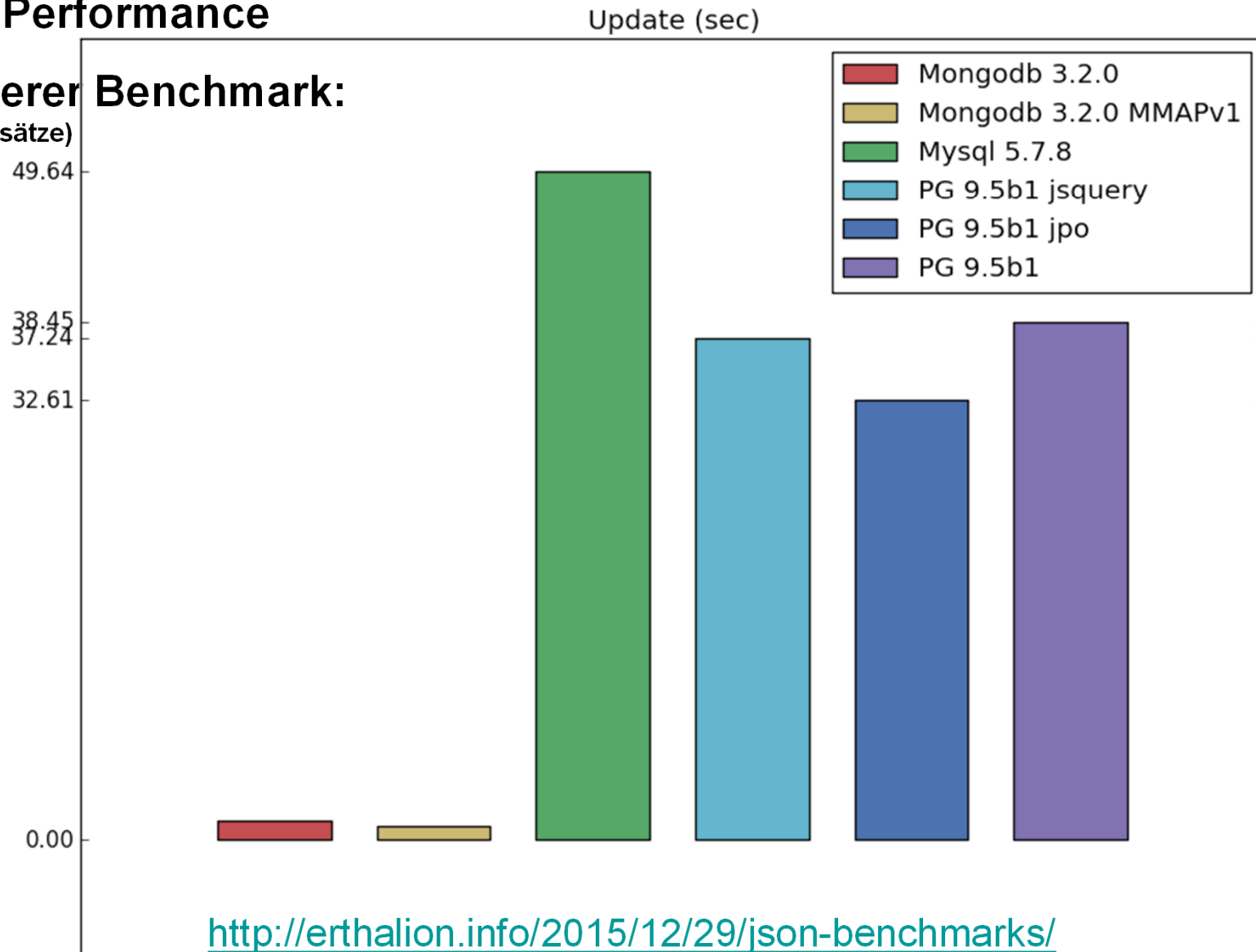
(1 Mio Datensätze)



## NoSQL Performance

### Ein anderer Benchmark:

(1 Mio Datensätze)



## NoSQL Performance

### Zwischenfazit

- **Es kommt auf den Test an**  
(welche Konfiguration? Welche einfachen/Komplexen Anfragen?)
- **PostgreSQL hat inzwischen auch schnelle JSON und key-value Datenbank-Formate**
- **Vorsichtige Faustregel: MongoDB punktet dann, wenn**
  - Mehr simple READ als WRITE
  - Einfache Inserts/Updates
  - Keine JOINS etc (keine Tabellenübergreifende Anfragen)
- **Besser: eigene Test mit eigener Datenstruktur durchführen!**

Siehe bei Interesse auch: <http://de.enterprisedb.com/postgres-plus-edb-blog/marc-linster/postgres-outperforms-mongodb-and-ushers-new-developer-reality> sowie <https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/>

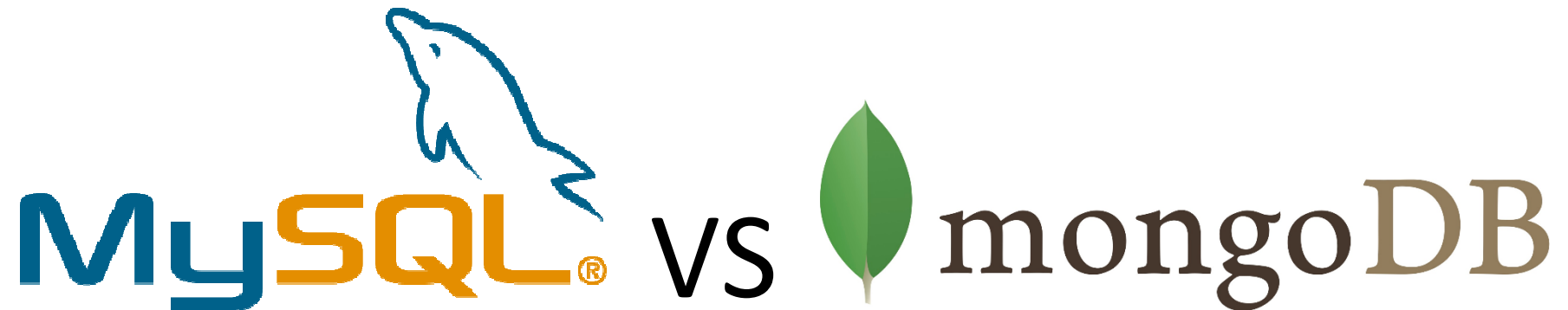
## Agenda

- **Wiederholung**
- **NoSQL und Performance**
- **Konzeptvergleich mit MySQL**

- **Alternativen**

- **mongoDB**
  - **Eigenschaften**
  - **Installation**
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- **\_\_\_\_\_**

- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



# Konzeptvergleich

## Konzeptvergleich

id	vorname	nachname	plz
0	Max	Mustermann	10133
1	Susi	Sonnenschein	10625
2	Bernd	Brot	10533

key	value
0	{ _id: 0, vorname: Max, nachname: Mustermann, plz: 10133 }
1	{ _id: 1, vorname: Susi, nachname: Sonnenschein, plz: 10625 }
2	{ _id: 2, vorname: Bernd, nachname: Brot, plz: 10533 }



SQL Terms/Concepts	MongoDB Terms/Concepts
database	<a href="#">database</a>
table	<a href="#">collection</a>
row	<a href="#">document</a> or <a href="#">BSON</a> document
column	<a href="#">field</a>
index	<a href="#">index</a>
table joins	embedded documents and linking
primary key Specify any unique column or column combination as primary key.	<a href="#">primary key</a> In MongoDB, the primary key is automatically set to the <a href="#">_id</a> field.
aggregation (e.g. group by)	aggregation pipeline See the <a href="#">SQL to Aggregation Mapping Chart</a> .

<http://docs.mongodb.org/manual/reference/sql-comparison/>



## SQL Schema Statements

```
CREATE TABLE users ( id MEDIUMINT NOT  
NULL AUTO_INCREMENT, user_id Varchar(30),  
age Number, status char(1), PRIMARY KEY (id) )
```

```
ALTER TABLE users ADD join_date DATETIME
```

## MongoDB Schema Statements

Implicitly created on first [insert\(\)](#) operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.users.insert( { user_id: "abc123", age: 55,  
status: "A" } )
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, [update\(\)](#) operations can add fields to existing documents using the [\\$set](#) operator.

```
db.users.update( { }, { $set: { join_date: new  
Date() } }, { multi: true } )
```

<http://docs.mongodb.org/manual/reference/sql-comparison/>

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document

level, [update\(\)](#) operations can remove fields from documents using the [\\$unset](#) operator.

```
db.users.update( { },  
                 { $unset: { join_date: "" } },  
                 { multi: true } )
```

**ALTER TABLE** users **DROP COLUMN** join\_date

**CREATE INDEX** idx\_user\_id\_asc **ON**  
users(user\_id)

```
db.users.ensureIndex( { user_id: 1 } )
```

**CREATE INDEX** idx\_user\_id\_asc\_age\_desc **ON**  
users(user\_id, age **DESC**)

```
db.users.ensureIndex( { user_id: 1, age: -1 } )
```

**DROP TABLE** users

```
db.users.drop()
```

<http://docs.mongodb.org/manual/reference/sql-comparison/>

## SQL INSERT Statements

```
INSERT INTO users(user_id, age, status)  
VALUES ("bcd001", 45, "A")
```

## MongoDB insert() Statements

```
db.users.insert( { user_id: "bcd001", age: 45,  
status: "A" } )
```

## SQL Update Statements

```
UPDATE users SET status = "C" WHERE age >  
25
```

## MongoDB update() Statements

```
db.users.update( { age: { $gt: 25 } },  
{ $set: { status: "C" } },  
{ multi: true } )
```

```
UPDATE users SET age = age + 3 WHERE  
status = "A"
```

```
db.users.update( { status: "A" } ,  
{ $inc: { age: 3 } },  
{ multi: true } )
```

<http://docs.mongodb.org/manual/reference/sql-comparison/>

## SQL Delete Statements

**DELETE FROM** users **WHERE** status = "D"

**DELETE FROM** users

## MongoDB remove() Statements

**db.users.remove( { status: "D" } )**

**db.users.remove( )**

<http://docs.mongodb.org/manual/reference/sql-comparison/>

SQL SELECT Statements	MongoDB find() Statements
<b>SELECT</b> * <b>FROM</b> users	<b>db.users.find()</b>
<b>SELECT</b> id, user_id, status <b>FROM</b> users	<b>db.users.find</b> ( { }, { <b>user_id</b> : 1, <b>status</b> : 1 } )
<b>SELECT</b> user_id, status <b>FROM</b> users	<b>db.users.find</b> ( { }, { <b>user_id</b> : 1, <b>status</b> : 1, <b>_id</b> : 0 } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A"	<b>db.users.find</b> ( { <b>status</b> : "A" } )
<b>SELECT</b> user_id, status <b>FROM</b> users <b>WHERE</b> status = "A"	<b>db.users.find</b> ( { <b>status</b> : "A" }, { <b>user_id</b> : 1, <b>status</b> : 1, <b>_id</b> : 0 } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status != "A"	<b>db.users.find</b> ( { <b>status</b> : { \$ne: "A" } } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A" <b>AND</b> age = 50	<b>db.users.find</b> ( { <b>status</b> : "A", <b>age</b> : 50 } )
<b>SELECT</b> * <b>FROM</b> users <b>WHERE</b> status = "A" <b>OR</b> age = 50	<b>db.users.find</b> ( { \$or: [ { <b>status</b> : "A" }, { <b>age</b> : 50 } ] } )

Viele weitere Beispiele siehe MongoDB-Referenz

<http://docs.mongodb.org/manual/reference/sql-comparison/>

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - **Konzeptvergleich mit MySQL**

### Alternativen

- **mongoDB**
  - **Eigenschaften**
  - **Installation**
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**

- **Alternativen**
  - Apache Cassandra
  - CouchDB
  - RethinkDB (Optimiert für Realtime Anwendungen)
  
- **Reine Key-Value-Datenbanken**
  - Google BigTable
  - Berkeley DB
  
- **Objekt-Datenbanken**
  - DB4O
  
- **Key-Value-Erweiterungen und JSON-Erweiterungen bei**
  - MySQL
  - PostgreSQL, ..

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen

- **mongoDB**

- Eigenschaften
- Installation

- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_

- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



mongoDB



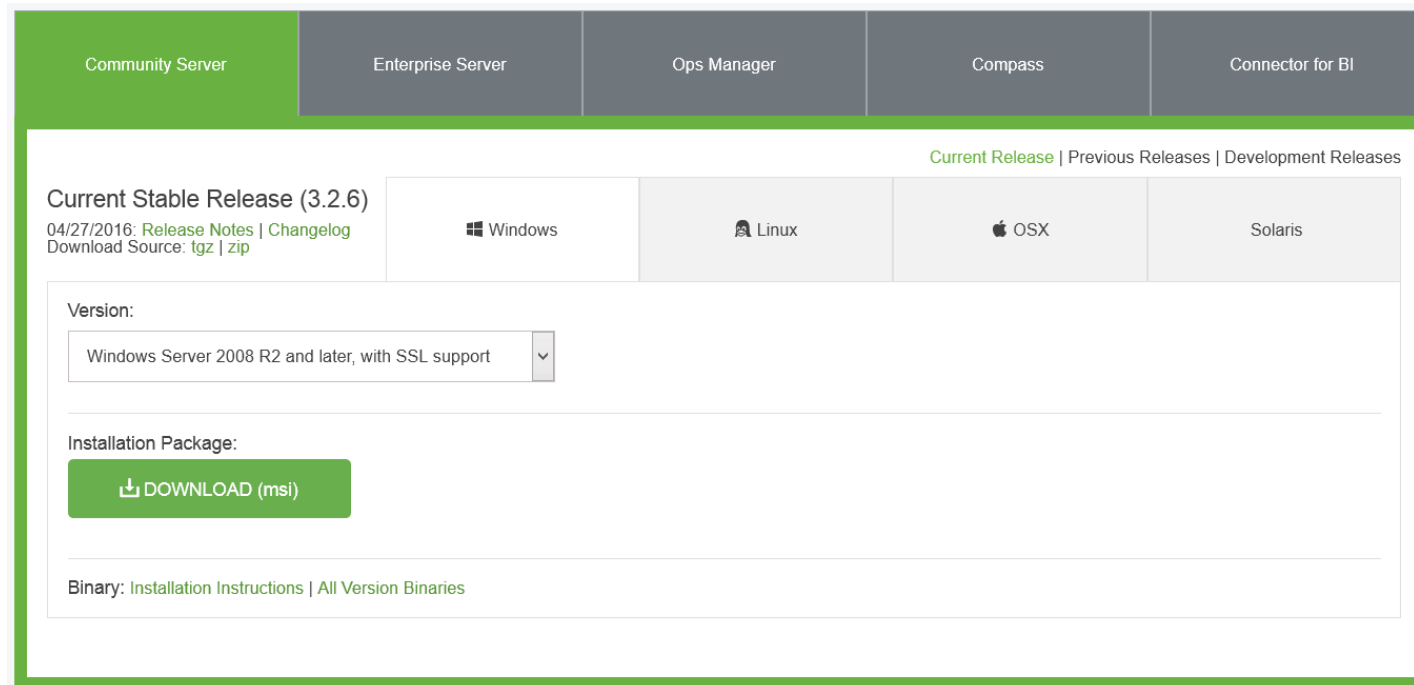
## mongoDB

- Kommt von „humongous“ \*
- führende NoSQL-Datenbank
- Open Source
- Dokumentenorientiert (JSON)
- Skalierbar (horizontal und vertikal)
- Syntax ist JavaScript-kompatibel!



## mongoDB - Installation

- 1. Install (bspw. Windows)
  - <https://www.mongodb.com/download-center#community>



<https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-windows/>

## mongoDB - Installation

### ■ 2. Datenbankordner erstellen

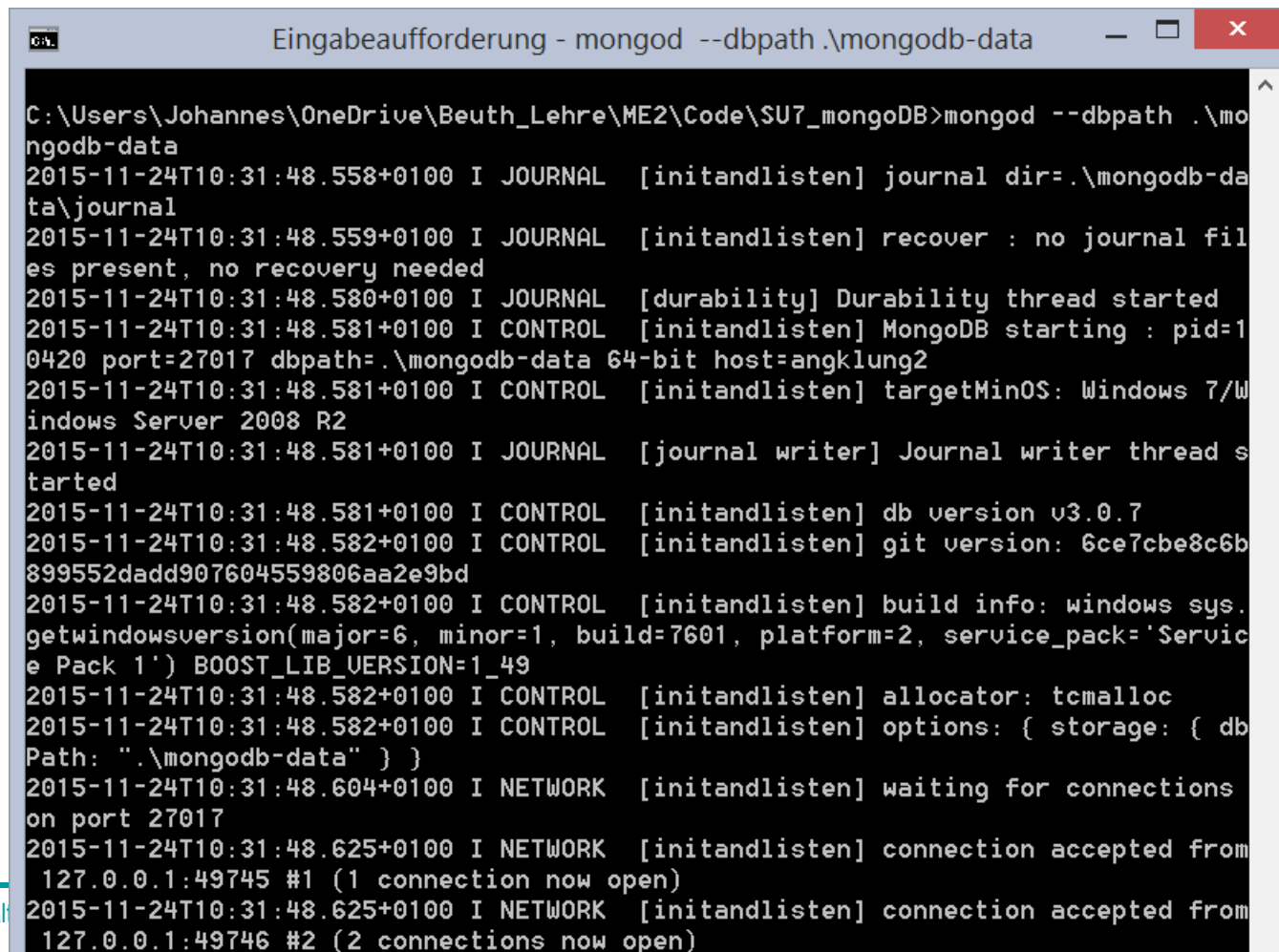
- data

z.B.:

- .\mongodb-data

## mongoDB - Installation

- 3. mongoDB starten (Terminal bleibt offen)
  - `mongod.exe --dbpath .\mongodb-data`



```
Eingabeaufforderung - mongod --dbpath .\mongodb-data

C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU7_mongoDB>mongod --dbpath .\mo
ngodb-data
2015-11-24T10:31:48.558+0100 I JOURNAL [initandlisten] journal dir=.\mongodb-da
ta\journal
2015-11-24T10:31:48.559+0100 I JOURNAL [initandlisten] recover : no journal fil
es present, no recovery needed
2015-11-24T10:31:48.580+0100 I JOURNAL [durability] Durability thread started
2015-11-24T10:31:48.581+0100 I CONTROL [initandlisten] MongoDB starting : pid=1
0420 port=27017 dbpath=.\mongodb-data 64-bit host=angklung2
2015-11-24T10:31:48.581+0100 I CONTROL [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2015-11-24T10:31:48.581+0100 I JOURNAL [journal writer] Journal writer thread s
tarted
2015-11-24T10:31:48.581+0100 I CONTROL [initandlisten] db version v3.0.7
2015-11-24T10:31:48.582+0100 I CONTROL [initandlisten] git version: 6ce7cbe8c6b
899552dadd907604559806aa2e9bd
2015-11-24T10:31:48.582+0100 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
2015-11-24T10:31:48.582+0100 I CONTROL [initandlisten] allocator: tcmalloc
2015-11-24T10:31:48.582+0100 I CONTROL [initandlisten] options: { storage: { db
Path: ".\mongodb-data" } }
2015-11-24T10:31:48.604+0100 I NETWORK [initandlisten] waiting for connections
on port 27017
2015-11-24T10:31:48.625+0100 I NETWORK [initandlisten] connection accepted from
127.0.0.1:49745 #1 (1 connection now open)
2015-11-24T10:31:48.625+0100 I NETWORK [initandlisten] connection accepted from
127.0.0.1:49746 #2 (2 connections now open)
```

## mongoDB - Test

- **mongo.exe** nutzen als Client (neues Terminal)

```
C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU7_mongoDB>mongo
MongoDB shell version: 3.0.7
connecting to: test
>
```

- **Auf dem Server: (altes Terminal)**

```
Sun Nov 10 13:32:11.798 [initandlisten] connection accepted from 127.0.0.1:54186
#1 <1 connection now open>
```

## mongoDB - Collection „Test“

```
C:\Users\Johannes\OneDrive\Beuth_Lehre\ME2\Code\SU7_mongoDB>mongo
MongoDB shell version: 3.0.7
connecting to: test
> db.test.insert( { "beuth-course": "me2" } )
WriteResult({ "nInserted" : 1 })
> db.test.find()
{ "_id" : ObjectId("56542f8f1a58a5bf5ef5732e"), "beuth-course" : "me2" }
>
```

### Merke: bei mongodb

- Ein **Verzeichnis** ist zentraler Speicherort aller Datenbanken
- Pro **Datenbank** gibt es eine gleichnamige Datei
- **Dokumentensammlungen** (Collections) haben einen eindeutigen Namen in einer solchen Datenbank
- und **einzelne Dokumente** sind JSON-konforme Objekte in diesen Collections

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen

- **mongoDB**

- Eigenschaften
  - Installation

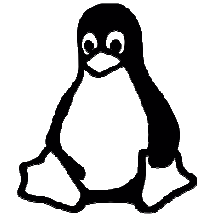
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**



- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



## mongoDB - Installation



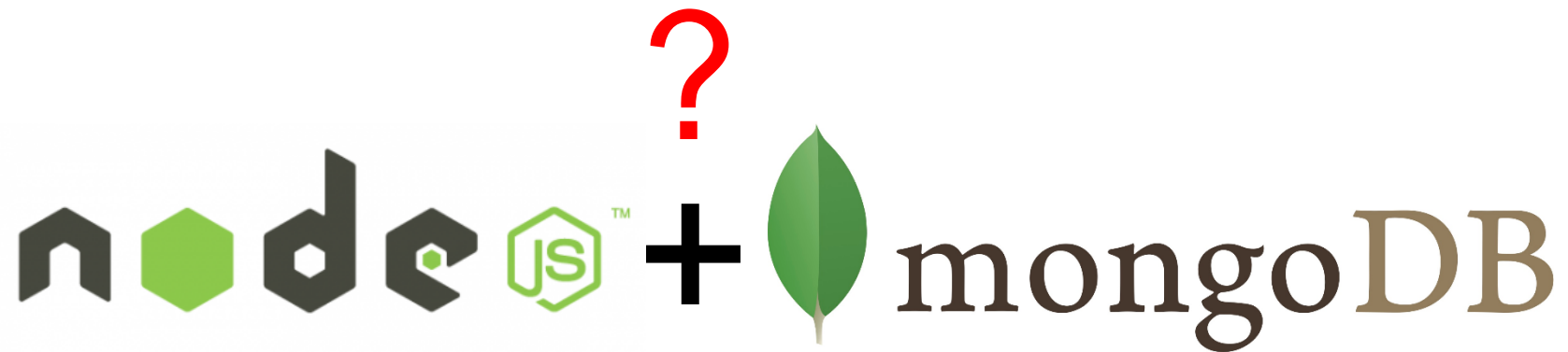
- Linux – APT vorbereiten (Key für Package Signatur)
  - `sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927`
- Source-List erstellen
  - `echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/3.2 main" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list`
- Installation
  - `sudo apt-get update`
  - `sudo apt-get install mongodb-org`

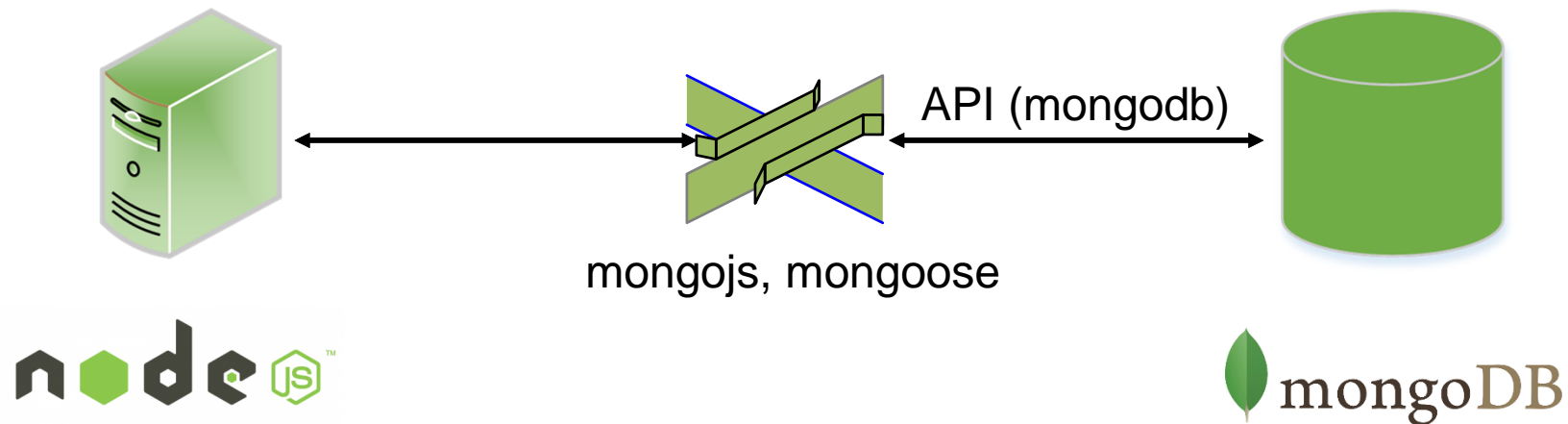
<https://docs.mongodb.com/manual/administration/install-on-linux/>



## mongoDB - Installation

```
root@viwitra:~# apt-get install mongodb-org
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  mongodb-10gen
1 upgraded, 0 newly installed, 0 to remove and 80 not upgraded.
Need to get 87.9 MB of archives.
After this operation, 829 kB of additional disk space will be used.
Get:1 http://downloads-distro.mongodb.org/repo/ubuntu-upstart/ dist/10gen mongodb-10gen amd64 2.4.8 [87.9 MB]
Fetched 87.9 MB in 11s (7572 kB/s)
(Reading database ... 37554 files and directories currently installed.)
Preparing to replace mongodb-10gen 2.4.4 (using .../mongodb-10gen_2.4.8_amd64.deb) ...
arg: upgrade
mongodb stop/waiting
Unpacking replacement mongodb-10gen ...
Processing triggers for ureadahead ...
ureadahead will be reprofiled on next reboot
Setting up mongodb-10gen (2.4.8) ...
mongodb start/running, process 13357
root@viwitra:~#
```





- Für die eigentliche Verbindung wird ein API-Modul benötigt. Natives Modul hierfür ist mongodb
  - <https://github.com/mongodb/node-mongodb-native>
- mongojs, mongoose sind Module, die eine Kommunikation zwischen nodeJS und mongoDB erleichtern.

# mongojs vs. mongoose

## mongojs

```
{
  "name": "mongojs",
  "version": "0.13.0",
  "repository": "git://github.com/mafintosh/mongojs.git",
  "author": "Mathias Buus Madsen <mathiasbuus@gmail.com>",
  "dependencies": {
    "thunky": "~0.1.0",
    "readable-stream": "~1.1.9",
    "mongodb": "1.4.0"
  },
  "scripts": {
    "test": "node ./tests"
  }
}
```

## mongoose

```
{
  "name": "mongoose"
, "description": "Mongoose MongoDB ODM"
, "version": "3.8.12-pre"
, "author": "Guillermo Rauch <guillermo@learnboost.com>"
, "dependencies": {
    "mongodb": "1.4.5"
  , "hooks": "0.2.1"
  , "ms": "0.1.0"
  , "sliced": "0.0.5"
  , "muri": "0.3.1"
  , "mpromise": "0.4.3"
  , "mpath": "0.1.1"
  , "regexp-clone": "0.0.1"
  , "mquery" : "0.7.0"
  }
...
}
```

## Kurzer Vergleich: mongojs und mongoose

### ■ **MongoJS**

- **Direkter Zugriff auf die Datenbank**
- **Flexibler Umgang mit Struktur**
- **Einfach in der Handhabung**
- **Native und einfache MongoDB-Syntax**

### ■ **Mongoose**

- **Arbeit mit Schemata und Modellen**
- **Komplexere Anforderungen können umgesetzt werden (z.B. Referenzen und JOINS)**
- **Umständlich für minimale Anforderungen**

## Agenda

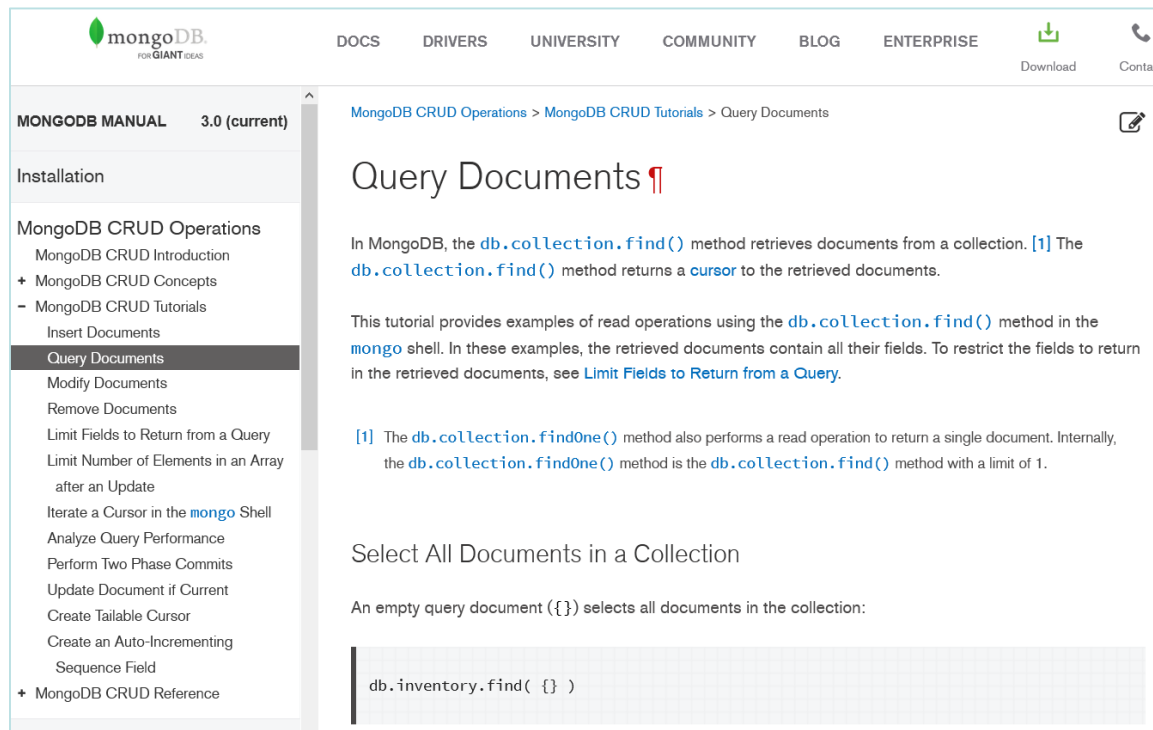
- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



# node.js + mongoDB mit mongojs

## Node Modul „mongojs“

- `yarn add mongojs`
- mongojs emuliert die offizielle MongoDB API
  - Also Syntax wie im Terminal, aber direkt in `node.js`



The screenshot shows the MongoDB Manual website. The left sidebar contains a table of contents for the 'MONGODB MANUAL 3.0 (current)'. The 'Query Documents' section is highlighted. The main content area is titled 'Query Documents' and contains text explaining the `db.collection.find()` method. It also includes a code block showing `db.inventory.find( {} )`.

MONGODB MANUAL 3.0 (current)

Installation

MongoDB CRUD Operations

- MongoDB CRUD Introduction
- + MongoDB CRUD Concepts
- MongoDB CRUD Tutorials
  - Insert Documents
  - Query Documents
  - Modify Documents
  - Remove Documents
  - Limit Fields to Return from a Query
  - Limit Number of Elements in an Array after an Update
  - Iterate a Cursor in the `mongo` Shell
  - Analyze Query Performance
  - Perform Two Phase Commits
  - Update Document if Current
  - Create Tailable Cursor
  - Create an Auto-Incrementing Sequence Field
- + MongoDB CRUD Reference

MongoDB CRUD Operations > MongoDB CRUD Tutorials > Query Documents

### Query Documents ¶

In MongoDB, the `db.collection.find()` method retrieves documents from a collection. [1] The `db.collection.find()` method returns a `cursor` to the retrieved documents.

This tutorial provides examples of read operations using the `db.collection.find()` method in the `mongo` shell. In these examples, the retrieved documents contain all their fields. To restrict the fields to return in the retrieved documents, see [Limit Fields to Return from a Query](#).

[1] The `db.collection.findOne()` method also performs a read operation to return a single document. Internally, the `db.collection.findOne()` method is the `db.collection.find()` method with a limit of 1.

### Select All Documents in a Collection

An empty query document (`{}`) selects all documents in the collection:

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

## Node Modul „mongojs“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

### ALT

```
var store = require('./blackbox/store.js');
```

### NEU

```
var mongojs = require('mongojs');  
var db = mongojs('mongodb://localhost:27017/we2',  
  ['tweets'],  
  {authMechanism: 'ScramSHA1'});
```

Standard-Port  
der mongoDB

Gewünschte  
Datenbank

Zu benutzende  
Collections

Letzter Parameter: Objekt mit Optionen.  
Bei MongoDB ^3.0 muss hier  
ScramSHA1 genommen werden

## Node Modul „mongojs“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

### ALT

```
store.select('tweets')
```

Zu benutzende Collections sind  
als Objekt in db verfügbar

### NEU

```
db.tweets.find({}, function(err, items) {  
  // check err, use items;  
});
```

Suchfilter-Objekt. Kann auch  
weggelassen werden.

## Node Modul „mongojs“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

**ALT**

```
var id = store.insert('tweets', req.body);  
  
store.replace('tweets', req.params.id, req.body);  
  
store.remove('tweets', req.params.id);
```

**NEU**

```
db.tweets.insert(req.body, function(err, item) {  
    id = item._id  
});  
  
db.tweets.findAndModify( {  
    query: { _id: mongojs.ObjectId(req.params.id) },  
    update: req.body,  
    new: true  
},  
function(err, item) {  
  
    });  
  
db.tweets.remove({ _id: mongojs.ObjectId(req.params.id) },  
function(err, result) {  
  
    });
```

## Node Modul „mongojs“: Programmierbeispiel

- **Ziel: Ersetzen von store.js durch mongodb**

```
ALT    var id = store.insert('tweets', req.body);

        store.replace('tweets', req.params.id, req.body);

        store.remove('tweets', req.params.id);
```

```

NEU db.tweets.insert(req.body, function(err, result) {
    id = item._id
  });

db.tweets.findAndModify( {
  query: { _id: mongojs.ObjectId(item._id) },
  update: req.body,
  new: true
},
function(err, item) {
  // item is the new tweet
});

db.tweets.remove( { _id: mongojs.ObjectId(item._id) },
function(err, result) {
  // item is the removed tweet
});

```

Wie bei store.js besteht das Problem, dass jedes beliebige Objekt (req.body) als Dokument eingefügt werden kann.

→ Konsistenzprüfung in der DB fehlt!

## Node Modul „mongojs“

- Weitere Funktionen, siehe Doku

<https://github.com/mafintosh/mongojs>

- Sortierung mit `.sort({name: 1})`
- Kriterien mit `.find({level: {$gt: 90}})`
- Limit, Offset mit `.limit(2).skip(1)` usw.

- Prinzipiell alle Funktionen, die mongoDB auch selbst hat, als JS-API

### mongojs

A `node.js` module for mongodb, that emulates [the official mongodb API](#) as much as possible. It wraps `mongodb-core` and is available through `npm`

```
npm install mongojs
```

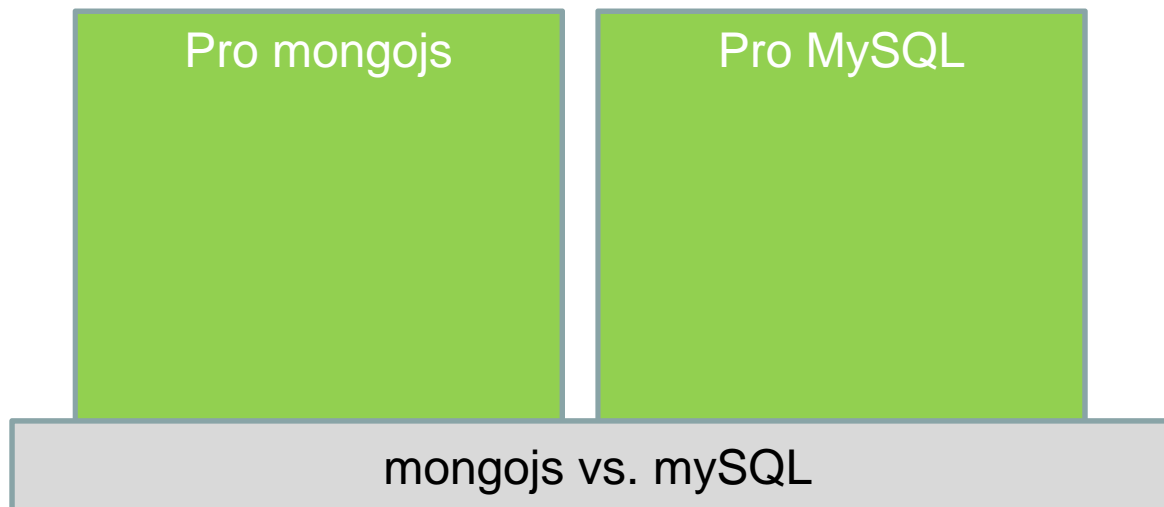
build passing code style standard

## Zusammenfassung:

- **Aufgabe: Vergleichen Sie mySQL und mongoDB mit mongojs.**  
Welche Vorteile hat mySQL für Sie. Welche Vorteile hat mongoDB mit mongojs-Modul?

**1. Sammeln Sie pro Team mind. zwei Vorteile von MySQL und zwei Vorteile von MongoDB/mongojs?(2-3min Zeit)**

**2. Anschließend: Tafelsammlung**





## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- \_\_\_\_\_
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**

# node.js + mongoDB mit mongoose

## Node Modul „mongoose“

- **yarn add mongoose**
  - Mongoose ist ein mongoDB Objekt-Mapper
  - Operationen
    - nicht direkt auf Collections via DB-Verbindung
    - Sondern: über Mongoose **Model** Instanzen



Was ist ein Model, was ein Schema?

In mongoose:

- **Model:** Eine Konstruktorfunktion, die Folgendes ermöglicht:
  - **Anlegen neuer Objekte basierend auf einem Schema**
  - **Zugriff auf die entsprechende DB-Collection des Models**
- **Schema:** Beschreibung der Datenstruktur

## Code-Beispiel: Vergleich von mongojs und mongoose

- **Mongojs: direkter Abruf von der Collection der DB**

```
var mongojs = require('mongojs');  
var db = mongojs('mydb', ['tweets']);  
db.tweets.find(function(err, docs) {  
    console.log(docs);  
});
```

- **Mongoose: ein Model wird erstellt (mittels eines Schemas) und darüber laufen die Anfragen**

```
var mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost/we2');  
  
var TweetModel = mongoose.model('Tweet', { message: String });  
  
TweetModel.find({}, function (err, tweets) {  
    console.log(tweets)  
});
```

```
var tweet = new TweetModel({ message: 'What a day..' });  
tweet.save(function (err) { });
```

## Node Modul „mongoose“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

### ALT

```
var store = require('./blackbox/store.js');
```

### NEU

```
var mongoose = require('mongoose');  
var db = mongoose.connect('mongodb://localhost:27017/we2');  
  
var TweetModel = require('./models/tweets');
```

## Node Modul „mongoose“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

### NEU (in Datei ./models/tweets.js)

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

```
var TweetSchema = new Schema({  
  message: { type: String, required: true },  
  creator: { type: Schema.Types.ObjectId, ref: 'User' },  
}, {  
  timestamps: { createdAt: 'timestamp' }  
});
```

```
module.exports = mongoose.model('Tweet', TweetSchema);
```

Schema ist eine  
Strukturdefinition (ähnlich einer  
Tabellenstruktur bei MySQL)

Options-Objekt erlaubt u.a.  
automatische Erstellung von  
Timestamps

.model erstellt eine  
Konstrukturfunktion, welche auch  
Methoden für find(), update usw. auf  
dieser Collection hat.

## Node Paket „mongoose“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

### ALT

```
store.select('tweets')
```

### NEU

```
TweetModel.find({}, function(err, items) {  
    res.json(items);  
});
```

## Node Paket „, mongoose“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

**ALT**    `var id = store.insert('tweets', req.body);`

**NEU**    `var tweet = new TweetModel(req.body);  
tweet.save(function(err) {  
 if (err) {  
 return next(err);  
 }  
 res.status(201).json(tweet);  
});`

### Vorteile der Model-Nutzung

1. Erstellung mit new übernimmt nur Felder, die im Schema definiert sind und wandelt Typen um
2. .save(..) prüft automatisch auf Konsistenz und liefert Error err falls was fehlt/falsch ist



## Node Paket „mongojs“: Programmierbeispiel

- Ziel: Ersetzen von store.js durch mongodb

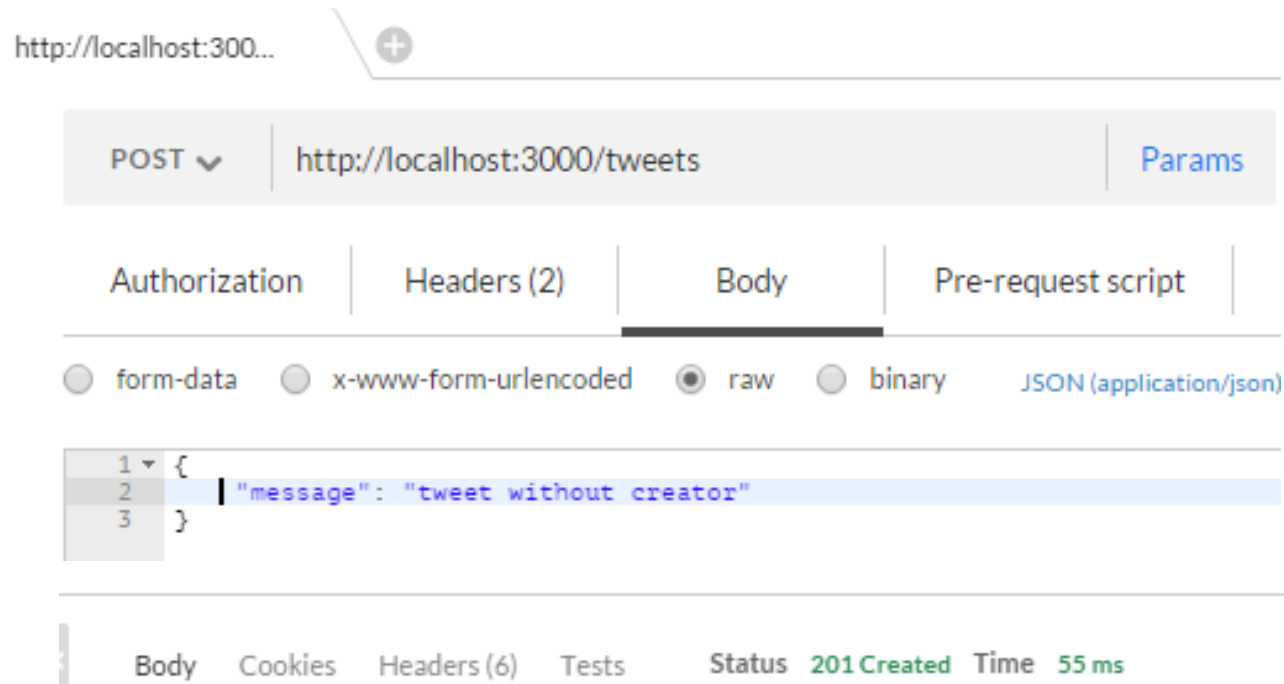
**ALT**

```
var id = store.insert('tweets', req.body);  
  
store.replace('tweets', req.params.id, req.body);  
  
store.remove('tweets', req.params.id);
```

**NEU**

```
var tweet = new TweetModel(req.body);  
tweet.save(function(err) {  
  if (err) {  
    return next(err);  
  }  
  res.status(201).json(tweet);  
});  
  
TweetModel.findByIdAndUpdate(req.params.id, req.body,  
  {new: true},  
  function(err, item) {  
  
  });  
  
TweetModel.findByIdAndRemove(req.params.id,  
  function(err, item) {  
  
  });
```

## Mongoose-basierte API ansprechen mit Postman



GET [http://localhost:3000/tweets](#) [Params](#)

Authorization Headers (2) Body Pre-request script

No Auth

Body Cookies Headers (6) Tests Status 200 OK

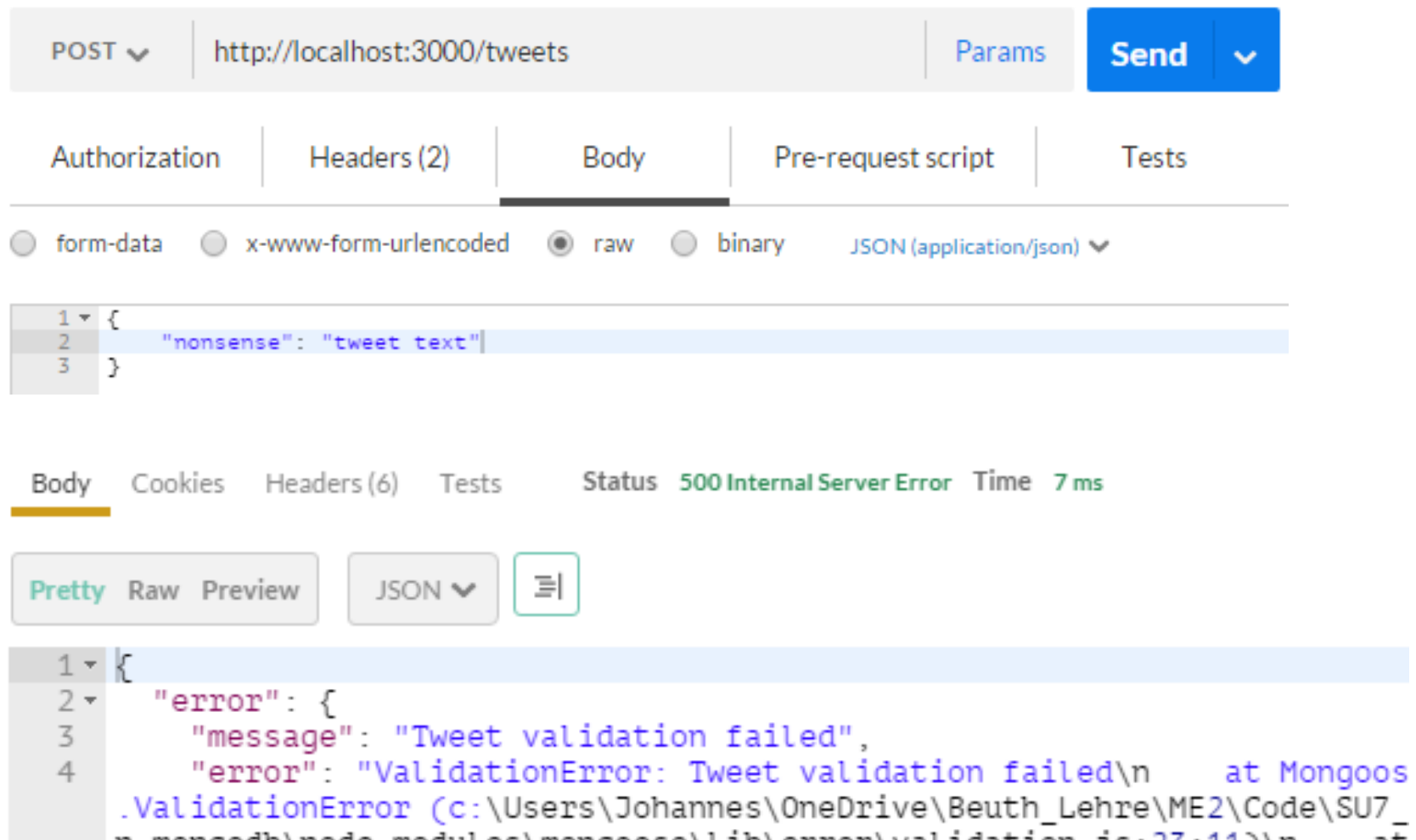
Pretty Raw Preview JSON

```
1 [
2   {
3     "_id": "565454b28a5666ec3aba5862",
4     "updatedAt": "2015-11-24T12:14:42.000Z",
5     "timestamp": "2015-11-24T12:14:42.000Z",
6     "message": "tweet without creator",
7     "__v": 0
8   }
9 ]
```

mongoDB vergibt \_id selbst

Version: wird hochgezählt bei Manipulation

## Mongoose-basierte API ansprechen mit Postman



POST ☐ http://localhost:3000/tweets Params Send ☐

Authorization Headers (2) **Body** Pre-request script Tests


☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ☐



```
1 {  
2   "nonsense": "tweet text"  
3 }
```

Body Cookies Headers (6) Tests **Status** 500 Internal Server Error Time 7 ms

Pretty Raw Preview JSON ☐

```
1 {  
2   "error": {  
3     "message": "Tweet validation failed",  
4     "error": "ValidationError: Tweet validation failed\n    at Mongoos  
.ValidationError (c:\\Users\\Johannes\\OneDrive\\Beuth_Lehre\\ME2\\Code\\SU7_  
n mongoose\\node_modules\\mongoose\\lib\\error\\validation.js:27:11)"
```


POST ▼ | http://localhost:3000/tweets | Params | Send ▼ |  ▼

Authorization | Headers (2) | **Body** | Pre-request script | Tests |  

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary | JSON (application/json) ▼

```
1 {  
2   "message": "tweet with creator",  
3   "creator": "8923145ghfo71326045123"  
4 }
```

**Body** | Cookies | Headers (6) | Tests | Status **500 Internal Server Error** | Time **11 ms**


Pretty | Raw | Preview | JSON ▼ | 


```
1 {  
2   "error": {  
3     "message": "Tweet validation failed",  
4     "error": "ValidationError: Tweet validation failed\n    at MongooseError  
.ValidationError (c:\\Users\\Johannes\\OneDrive\\Beuth_Lehre\\ME2\\Code\\SU7_tweets_i  
n_mongodb\\node_modules\\mongoose\\lib\\error\\validation.js:23:11)\n    at model  
.Document.invalidate (c:\\Users\\Johannes\\OneDrive\\Beuth_Lehre\\ME2\\Code\\SU7_twee  
ts_in_mongodb\\node_modules\\mongoose\\lib\\document.js:1280:32)\n    at model  
Document.set (c:\\Users\\Johannes\\OneDrive\\Beuth_Lehre\\ME2\\Code\\SU7_tweets_in m
```



## Mongoose Validation Errors: Example POST

```
var tweet = new TweetModel(req.body);
tweet.save(function(err) {
  if (err) {
    err.status = 400;
    err.message += ' in fields: '
                  + Object.getOwnPropertyNames(err.errors);
    return next(err);
  }
  res.status(201).json(tweet);
});
```

## Mongoose Validation Errors: Example POST





POST  Params Send 

Authorization Headers (2) **Body** Pre-request script Tests  

☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) ▼

```
1 {  
2   "nonsense": "tweet text"  
3 }
```

Body Cookies Headers (6) Tests **Status 400 Bad Request** Time 53 ms

Pretty Raw Preview JSON ▼  

```
1 {  
2   "error": {  
3     "message": "Tweet validation failed in fields: message",  
4     "error": "ValidationError: Tweet validation failed in fields: message\  
at MongooseError.ValidationError (c:\\Users\\Johannes\\OneDrive\\Beuth_Lehre\\ME2\\  
Code\\SU7_tweets_in_mongodb\\node_modules\\mongoose\\lib\\error\\validation.js:23:11"
```

Mehr Doku zu Validation Errors: <http://mongoosejs.com/docs/validation.html>

## Mongoose: Filtern





## Mongoose: Filtern

- Nur bestimmte Attribute zurückliefern

```
TweetModel.find({}, 'message', function(err, items) {  
    res.json(items);  
});
```

```
[  
  {  
    "_id": "565454b28a5666ec3aba5862",  
    "message": "tweet without creator"  
  }  
]
```

- Auch möglich als Query via Fluent Interfaces zusammenzustellen

```
var query = TweetModel.find({ });  
query.select('message')  
  .exec(function(err, items) { ... });
```

## Mongoose: Filtern

### ■ Weitere Filter

```
var query = TweetModel.find({ message: /first/ })  
  .where('likes').gt(0).lt(1000)  
  .limit(10)  
  .sort('-timestamp')  
  .select('message timestamp');  
query.exec(function(err, items) { });
```

### ■ Achtung: Bei sort() sollte ein index dafür angelegt worden sein!

```
TweetSchema.index({ timestamp: 1 });
```

### ■ Siehe auch Doku

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

## Zusammenfassung:

**Aufgabe:** Welche Aussagen treffen für Modul monojs zu, welche für mongoose, welche für beide/keinen?



1. Überlegen Sie erst alleine, dann im Team
2. Unklare Punkte diskutieren wir anschließend

mongo js	Beide/ keines	mong oose	Aussage
			unterstützt Validierung beim Speichern
			Ist ein Object-Document-Mapper (ODM)
			unterstützt das Speichern verschiedener Dokumentarten in einer Sammlung (collection)
			unterstützt Fremdschlüsselbeziehungen und JOINS
			unterstützt die Auswahl nur einzelner Dokumentenfelder (wie SELECT name, age from users;)
			unterstützt die komplette MongoDB API-Syntax

## Zusammenfassung:

**Aufgabe:** Welche Aussagen treffen für Modul monojs zu, welche für mongoose, welche für beide/keinen?

1. Überlegen Sie erst alleine, dann im Team
2. Unklare Punkte diskutieren wir anschließend



Mongo js	Beide/ keines	mong oose	Aussage
		x	unterstützt Validierung beim Speichern (Schemas)
		x	Ist ein Object-Document-Mapper (ODM)
x			unterstützt das Speichern verschiedener Dokumentarten in einer Sammlung (collection)
		x	unterstützt Fremdschlüsselbeziehungen und JOINS
	x		unterstützt die Auswahl nur einzelner Dokumentenfelder (wie SELECT name, age from users;)
x			unterstützt die komplette MongoDB API-Syntax

## Quellen für APIs

---

- **mongoDB**
  - <https://github.com/mongodb/node-mongodb-native>
- **MongoJS**
  - <https://github.com/mafintosh/mongojs>
- **Mongoose**
  - <https://github.com/learnboost/mongoose>

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- **node-restful**
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



## REST APIs mit node-restful



- **yarn add node-restful**
  - Eine Middleware
  - basierend auf Mongoose Models
  - Erstellt Handler für alle nötigen REST Methoden automatisch
  - Erstellt automatisch
    - GET /resources
    - GET /resources/:id
    - POST /resources
    - PUT /resources/:id
    - DELETE /resources/:id

## REST APIs mit node-restful



```
var restful = require('node-restful');  
var mongoose = require('mongoose');  
mongoose.connect('mongodb://localhost:27017/me2');  
  
var TweetSchema = require('./models/tweet-schema.js');  
  
var TweetModel = restful.model('Tweets', TweetSchema);  
TweetModel.methods(['get', 'post', 'put', 'delete']);  
TweetModel.register(app, '/tweets');  
  
logger('activated REST API for tweets');
```

- **Das ist alles.**

Mit register(..) hängt sich node-restful in app.use(..) entsprechend ein.

(Nutzung von MongoDB, mongoose oder node-restful für Übungsblatt 4 leider nicht erlaubt.  
node-restful auch nicht für Übungsblatt 5 erlaubt)



## REST APIs mit node-restful

### ■ Zum Vergleich: Der Code vorher

```
app.get('/tweets', function(req,res,next) {
    TweetModel.find({}, 'message' , function(err,
        res.json(items);
    });
});

app.post('/tweets', function(req,res,next) {
    var tweet = new TweetModel(req.body);
    tweet.save(function(err) {
        if (!err) {
            res.status(201).json(tweet)
        } else {
            err.status = 400;
            err.message += ' in fields: ' + Object
        }
        next(err);
    });
});
...
```

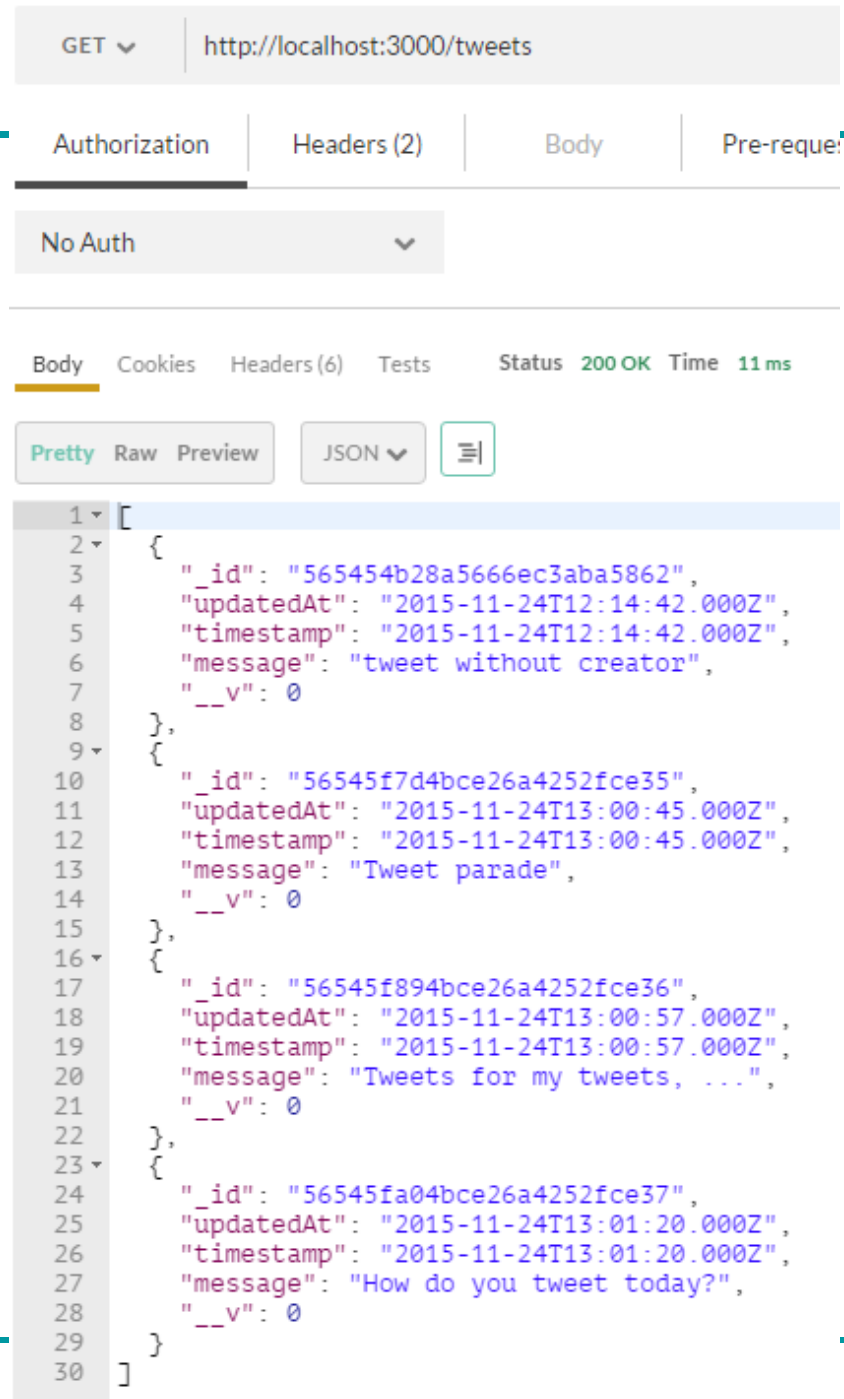
```
app.get('/tweets/:id', function(req,res,next) {
    res.json(store.select('tweets', req.params.id));
});

app.delete('/tweets/:id', function(req,res,next) {
    TweetModel.findByIdAndRemove(req.params.id, function(err, it
        if (err || !item) {
            err = err || new Error("item not found");
            err.status = 404;
        } else {
            res.status(200).end();
        }
        next(err);
    });
});

app.put('/tweets/:id', function(req,res,next) {
    TweetModel.findByIdAndUpdate(req.params.id, req.body, {new:
        if (err) {
            err.status = 400;
        }
        res.status(200).end();
        next(err);
    });
});
```

## REST APIs mit node-restful

- **yarn add node-restful**
- **Liefert Funktionen mit für**
  - Blättern
  - Filtern
  - Suche



GET ▼ | http://localhost:3000/tweets

Authorization | Headers (2) | Body | Pre-reqs

No Auth ▼

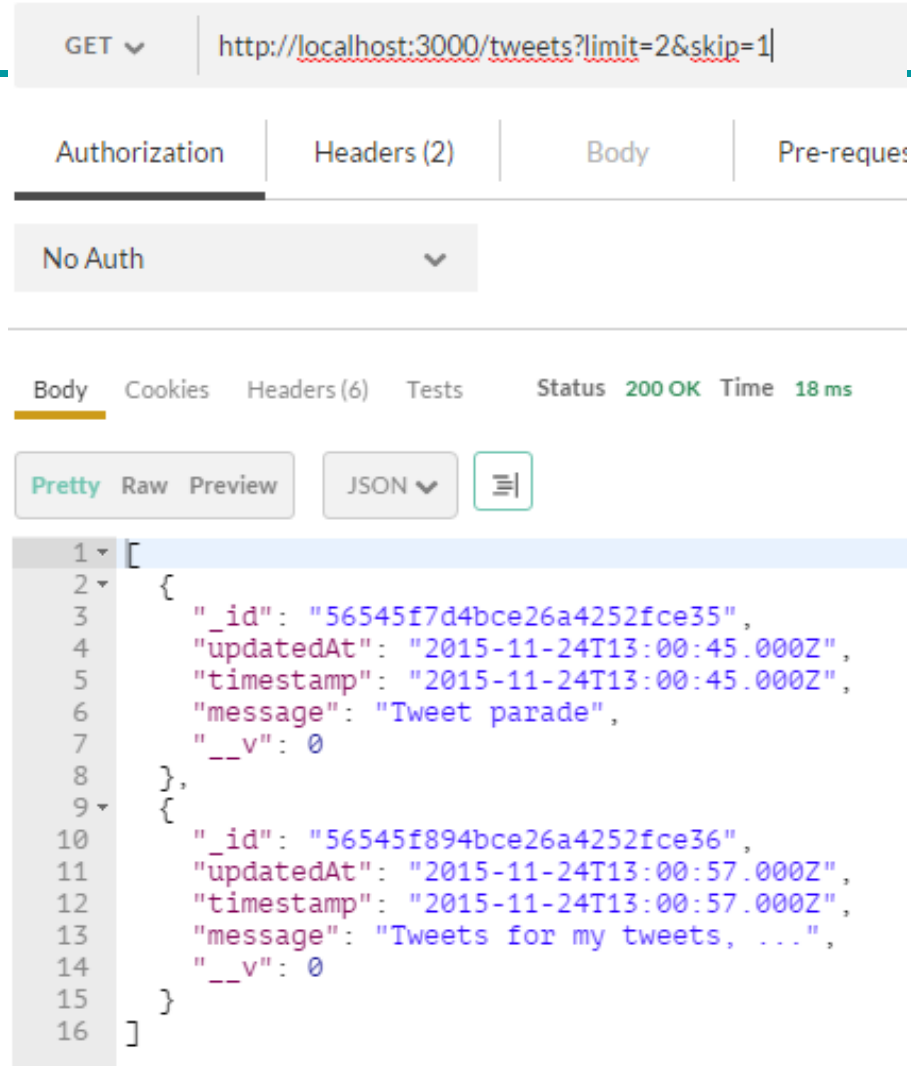
Body | Cookies | Headers (6) | Tests | Status 200 OK | Time 11 ms

Pretty | Raw | Preview | JSON ▼ | ≡

```
1 [
2   {
3     "_id": "565454b28a5666ec3aba5862",
4     "updatedAt": "2015-11-24T12:14:42.000Z",
5     "timestamp": "2015-11-24T12:14:42.000Z",
6     "message": "tweet without creator",
7     "__v": 0
8   },
9   {
10    "_id": "56545f7d4bce26a4252f35",
11    "updatedAt": "2015-11-24T13:00:45.000Z",
12    "timestamp": "2015-11-24T13:00:45.000Z",
13    "message": "Tweet parade",
14    "__v": 0
15  },
16  {
17    "_id": "56545f894bce26a4252f36",
18    "updatedAt": "2015-11-24T13:00:57.000Z",
19    "timestamp": "2015-11-24T13:00:57.000Z",
20    "message": "Tweets for my tweets, ...",
21    "__v": 0
22  },
23  {
24    "_id": "56545fa04bce26a4252f37",
25    "updatedAt": "2015-11-24T13:01:20.000Z",
26    "timestamp": "2015-11-24T13:01:20.000Z",
27    "message": "How do you tweet today?",
28    "__v": 0
29  }
30 ]
```

## REST APIs mit node-restful

- **Blättern:** ?limit=2&skip=1
- (liefert tweets 2 und 3)



GET `http://localhost:3000/tweets?limit=2&skip=1`

Authorization Headers (2) Body Pre-requests

No Auth

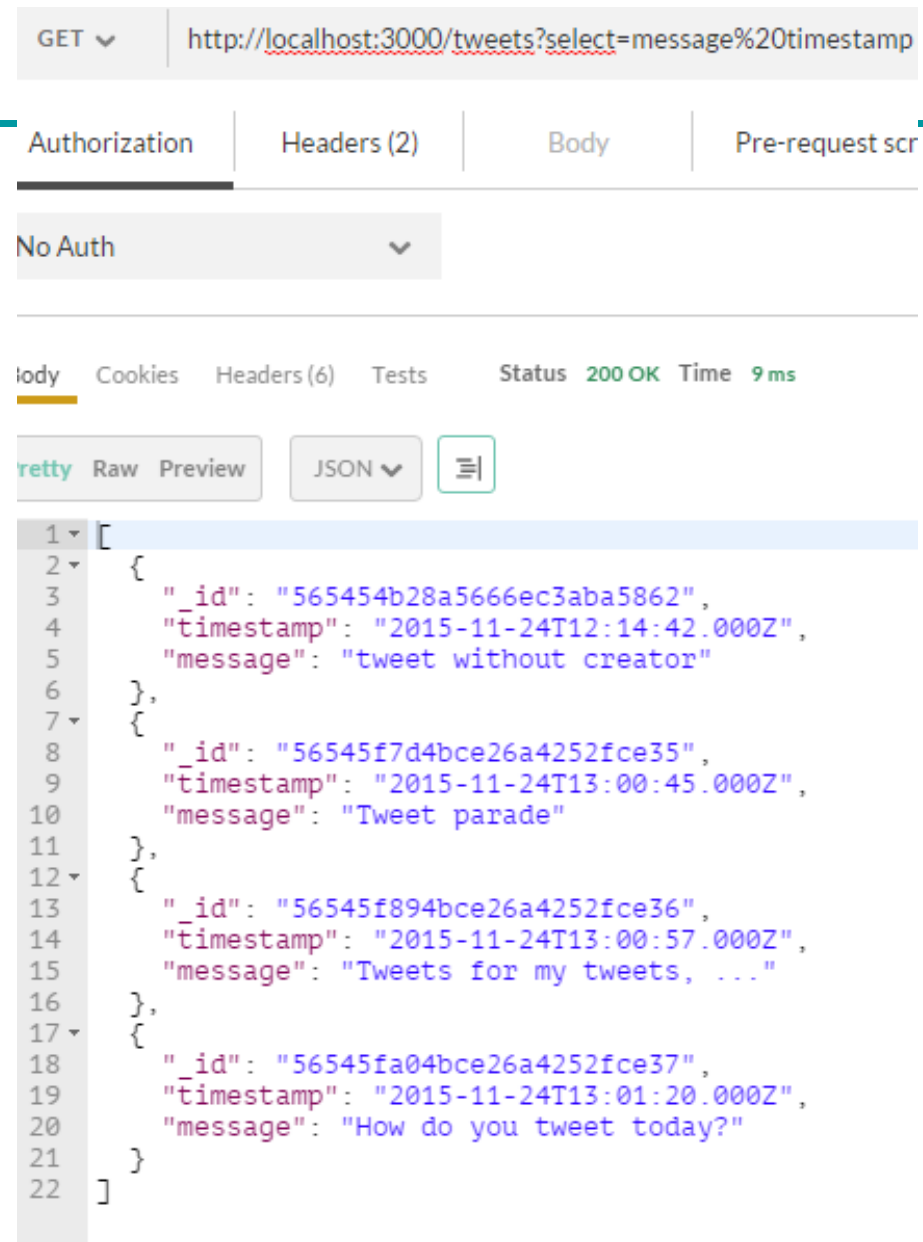
Body Cookies Headers (6) Tests Status 200 OK Time 18 ms

Pretty Raw Preview JSON

```
[
  {
    "_id": "56545f7d4bce26a4252fce35",
    "updatedAt": "2015-11-24T13:00:45.000Z",
    "timestamp": "2015-11-24T13:00:45.000Z",
    "message": "Tweet parade",
    "__v": 0
  },
  {
    "_id": "56545f894bce26a4252fce36",
    "updatedAt": "2015-11-24T13:00:57.000Z",
    "timestamp": "2015-11-24T13:00:57.000Z",
    "message": "Tweets for my tweets, ...",
    "__v": 0
  }
]
```

## REST APIs mit node-restful

- **Filtern:**  
**?select=message%20timestamp**



GET <http://localhost:3000/tweets?select=message%20timestamp>

Authorization Headers (2) Body Pre-request script

No Auth

body Cookies Headers (6) Tests Status 200 OK Time 9 ms

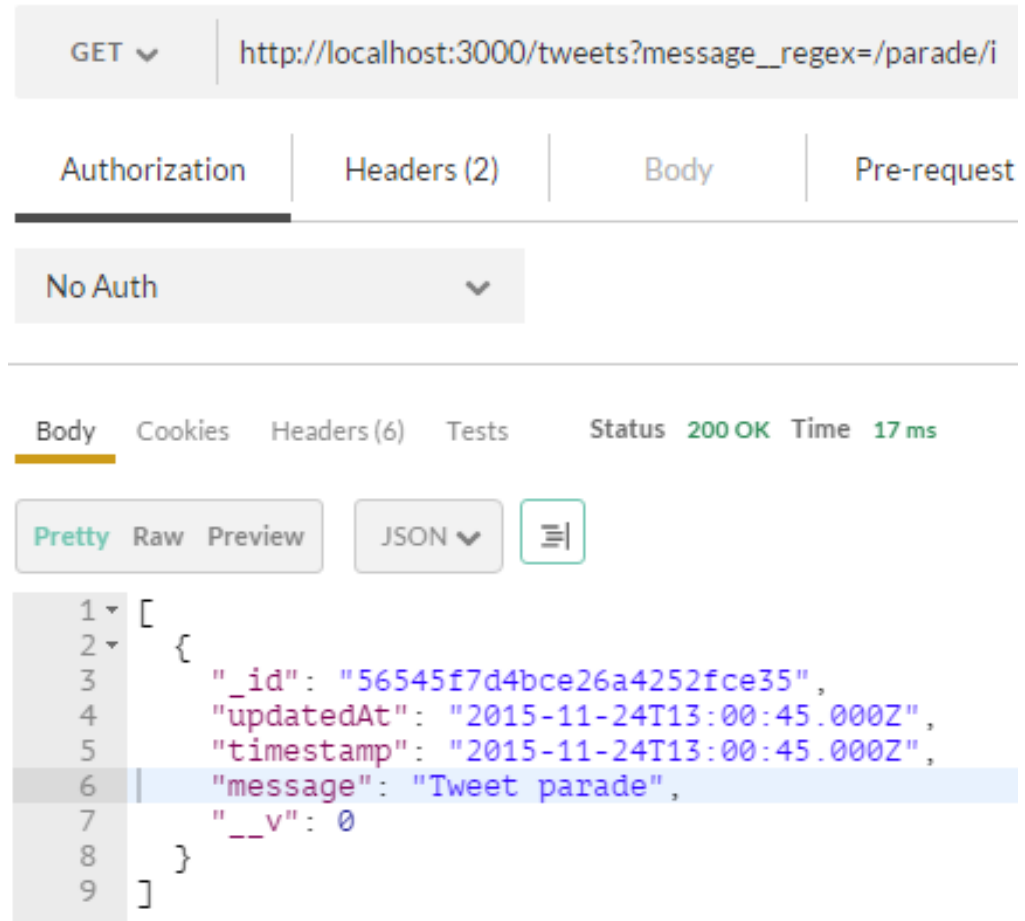
pretty Raw Preview JSON

```
[
  {
    "_id": "565454b28a5666ec3aba5862",
    "timestamp": "2015-11-24T12:14:42.000Z",
    "message": "tweet without creator"
  },
  {
    "_id": "56545f7d4bce26a4252fce35",
    "timestamp": "2015-11-24T13:00:45.000Z",
    "message": "Tweet parade"
  },
  {
    "_id": "56545f894bce26a4252fce36",
    "timestamp": "2015-11-24T13:00:57.000Z",
    "message": "Tweets for my tweets, ..."
  },
  {
    "_id": "56545fa04bce26a4252fce37",
    "timestamp": "2015-11-24T13:01:20.000Z",
    "message": "How do you tweet today?"
  }
]
```

## REST APIs mit node-restful

- **Suche:**  
? message\_\_regex=/parade/i

- ..und weiteres, siehe  
<https://github.com/baugarten/node-restful>




GET ▼ | http://localhost:3000/tweets?message\_\_regex=/parade/i

Authorization | Headers (2) | Body | Pre-request

No Auth ▼

Body | Cookies | Headers (6) | Tests | Status 200 OK Time 17 ms

Pretty Raw Preview | JSON ▼ | 

```
1 [
2   {
3     "_id": "56545f7d4bce26a4252fce35",
4     "updatedAt": "2015-11-24T13:00:45.000Z",
5     "timestamp": "2015-11-24T13:00:45.000Z",
6     "message": "Tweet parade",
7     "__v": 0
8   }
9 ]
```

## Agenda

- **Wiederholung**
  - **NoSQL und Performance**
    - Konzeptvergleich mit MySQL
    - Alternativen
  - **mongodb**
    - Eigenschaften
    - Installation
  - **Mongojs + Aufgabe**
  - **Mongoose + Aufgabe**
  - **node-restful**
- 
- **Zusammenfassende Fragen**
  - **Organisation des Inverted Classroom in 14Tg.**
  - **Ausblick**

## Zusammenfassende Fragen

1. Wann ist MongoDB die bessere Datenbank, wann MySQL? Wovon hängt das ab?
2. Was sind **collections** und wie unterscheiden die sich von relationalen Tabellen?
3. Welche **Alternativen zu MongoDB und mySQL** kennen Sie, die man ebenfalls für JSON-Daten verwenden könnte?
4. Wie unterscheiden sich **mongojs** und **mongoose**?
5. Wie legen Sie eine **neue Collection** in der MongoDB mittels mongojs an?
6. Wie geben Sie eine Fremdschlüssel-Beziehung (**1:n**) in Objekten ihrer Collections an?
7. Wie gelingt bei einer MongoDB die **Konsistenzprüfung** der gespeicherten Daten?
8. Unter welchen Bedingungen ist der Einsatz eines relationalen Datenbankschemas passender?
9. Was ist ein **Objekt Mapper**, was ein Schema, was ein Model?
10. Wo kommen bei mongoose **Konstruktorfunktionen** zum Einsatz?
11. Welche zwei wesentlichen Vorteile hat die **Verwendung eines Models** mit mongoose bei der Speicherung von Objekten in der Datenbank?
12. Wie konfigurieren Sie mongoose so, dass **automatisch Zeitstempel** verwaltet werden?
13. Wie können Sie mittels mongoose Anfragen (queries) an die DB mit **Filtern** versehen (bspw. Anzahl an Einträgen, Größer/kleiner, Offset usw.)?
14. Wozu dienen **Datenbankindizes** und wann sollten Sie welche für Ihre DB anlegen?
15. Was ist **node-restful**?
16. **Bonusfrage:** Wenn Sie eine idempotente PATCH-Implementierung vornehmen wollen, ist das dann leichter mit mongoose oder mit mongojs Modul umzusetzen? Warum?

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- **node-restful**
  
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**



## Semesterplan

	Datum	Thema
1	11.04.2017	Einführung, Ziele, Ablauf, Benotung usw.
2	18.04.2017	Wiederholung HTML/CSS/JS
3	25.04.2017	Client-Server Architekturen und WebStacks
4	02.05.2017	REST-APIs
5	09.05.2017	REST in node.js
6	16.05.2017	Debugging und Testen
7	23.05.2017	Strukturierung, Modularisierung
8	30.05.2017	Datenhaltung, SQL, NoSQL
9	06.06.2017	backbone.js als Gegenpart zu REST/Node.js
10	13.06.2017	Vertiefung (DB, Sicherheit, ..) als <b>Flipped Classroom (FC)</b>
11	20.06.2017	Nachbesprechung FC; Authentifizierung
12	27.06.2017	Mobile Development/Cross-Plattform-Development
13	<b>06.07.2017</b>	<b>Do</b> , Zusammenfassung/Semesterüberblick ( <b>alle Züge!</b> )
14	11.07.2017	Gastdozent(en) mit Anwesenheitspflicht
15	18.07.2017	<b>Klausur PZR1 (Di, 18.07. 10:00 Uhr, Raum ?)</b>
16	25.07.2017	Klausureinsicht ( <b>Di, 25.07. 10:00 Uhr, Raum ?</b> )
	21.09.2017	<b>Klausur PZR2 (Do, 21.09. 12:00 Uhr, Raum ?)</b>

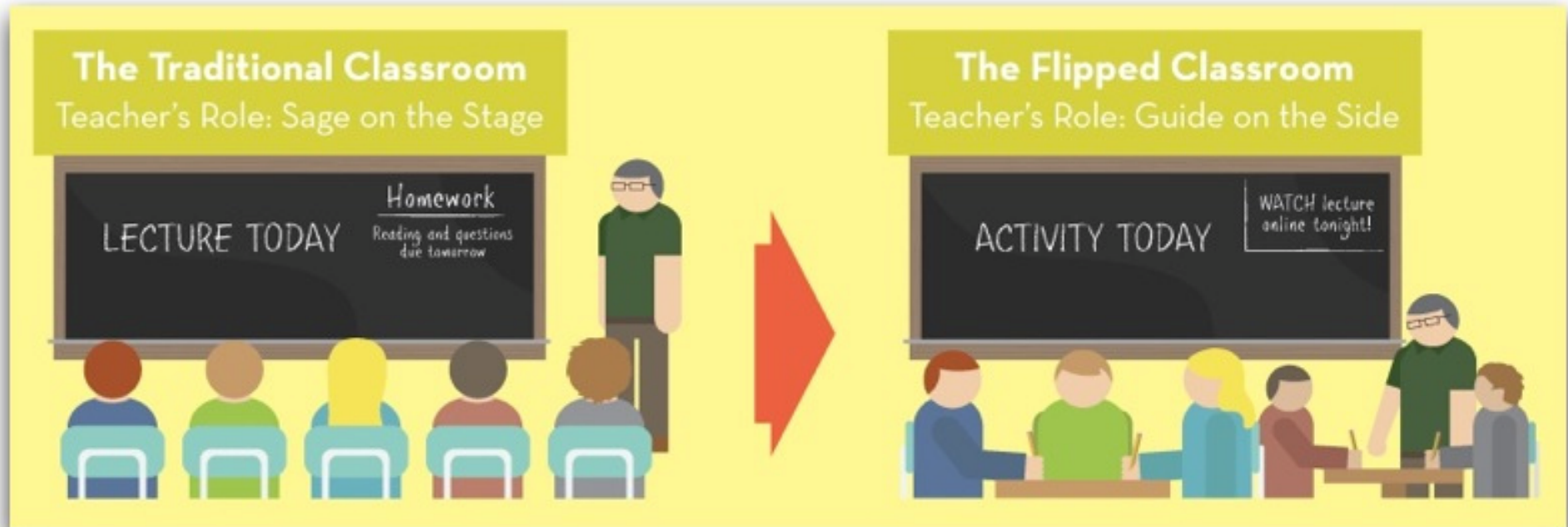
Zug 2

## Semesterplan

	Datum	Thema
1	06.04.2017	Einführung, Ziele, Ablauf, Benotung usw.
2	13.04.2017	Wiederholung HTML/CSS/JS
3	20.04.2017	Client-Server Architekturen und WebStacks
4	27.04.2017	REST-APIs
5	04.05.2017	REST in node.js
6	11.05.2017	Debugging und Testen
7	18.05.2017	Strukturierung, Modularisierung
8	<b>01.06.2017</b>	Datenhaltung, SQL, NoSQL
9	08.06.2017	backbone.js als Gegenpart zu REST/Node.js
10	15.06.2017	Vertiefung (DB, Sicherheit, ..) als <b>Flipped Classroom (FC)</b>
11	22.06.2017	Nachbesprechung FC; Authentifizierung
12	29.06.2017	Mobile Development/Cross-Plattform-Development
13	<b>06.07.2017</b>	Zusammenfassung/Semesterüberblick ( <b>alle Züge!</b> )
14	13.07.2017	Gastdozent(en) mit Anwesenheitspflicht
15	18.07.2017	<b>Klausur PZR1 (Di, 18.07. 10:00 Uhr, Raum ?)</b>
16	25.07.2017	Klausureinsicht ( <b>Di, 25.07. 10:00 Uhr, Raum ?</b> )
	21.09.2017	<b>Klausur PZR2 (Do, 21.09. 12:00 Uhr, Raum ?)</b>

Zug 1/3

## Flipped Classroom (am 13./15.06.) zur Vertiefung



- **SU passiv**
- **Nachbereitung** anhand von Zusammenfassungsfragen und Aufgaben
- **Vorbereitung** anhand von Material
- **SU aktiv** in Gruppen Inhalte zusammenfassen und den anderen vorstellen

## Vertiefungsthemen als FC + Expertengruppen (Jigsaw)

### A. Web Application Security

4 der Top10 Sicherheitsrisiken und  
Gegenmaßnahmen bei node/express Servern



### B. PostgreSQL + JSON statt MongoDB

Verwendung und Anbindung von PostgreSQL  
in nodejs/express

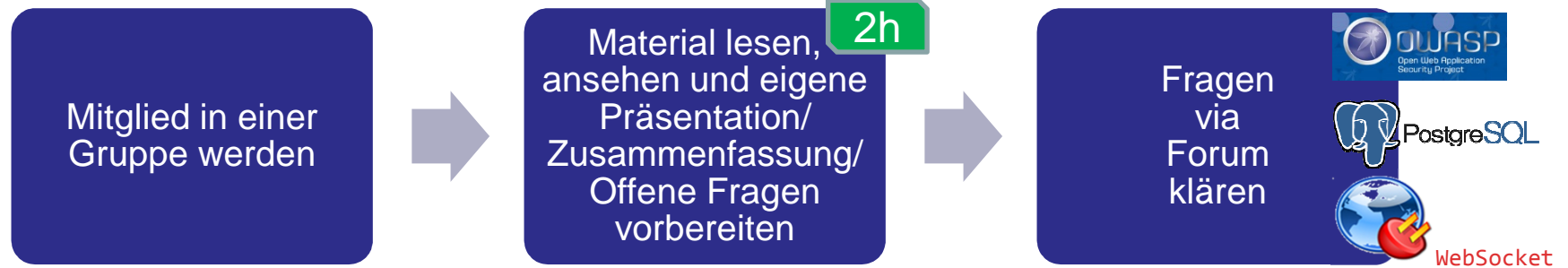


### C. WebSockets für Publish-Subscribe-Lösungen

Wie Websockets in node/express  
verwendet werden können



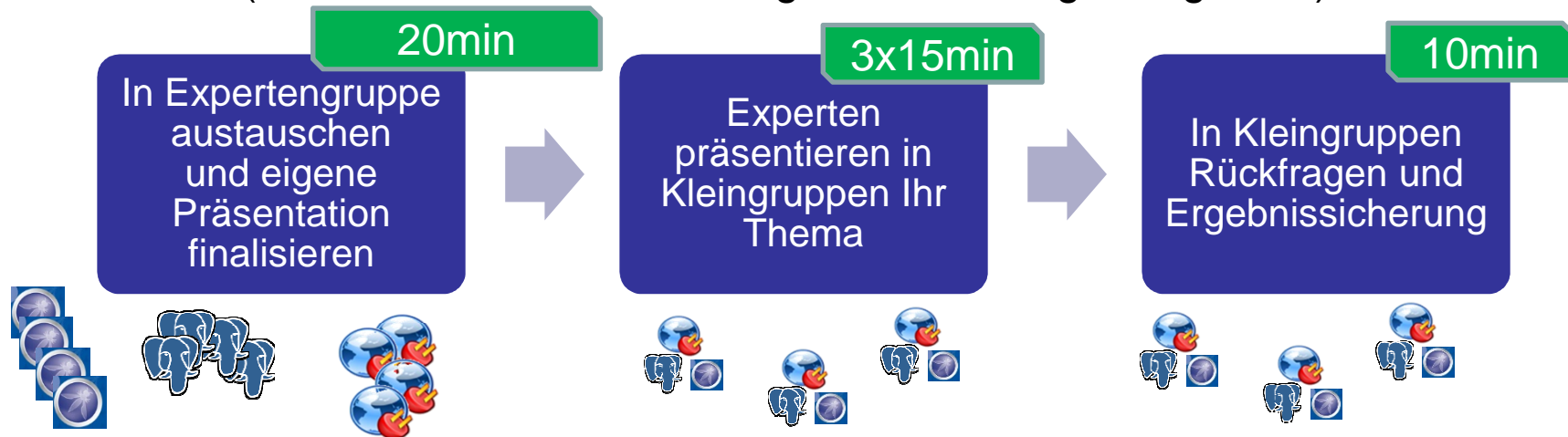
## Vertiefungsthemen



Diese und nächste Woche VOR dem 13./15.06.

In der SU 13./15.06.

(es wird KEINE Vorbereitung/Wiederholung dort geben!)



## Ihre Checkliste für die Vorbereitung

### Diese Woche:

- **Gruppe in Moodle beitreten (A, B oder C). Begrenztes Kontingent! Prof. legt Sie ggf. in andere Gruppe um.**

### Ab nächste Woche:

- **Material zum Gruppenthema lesen/ansetzen (Zugriff nur für Mitglieder)**
- **Offene Fragen via Forum klären (nur eigene Gruppe sichtbar)**
- **Zusammenfassende Fragen erstellen (5-10 Fragen)**
- **Eigene Präsentation/Demo/Zusammenfassung für Präsentation in Kleingruppe vorbereiten (Dauer 15min)**

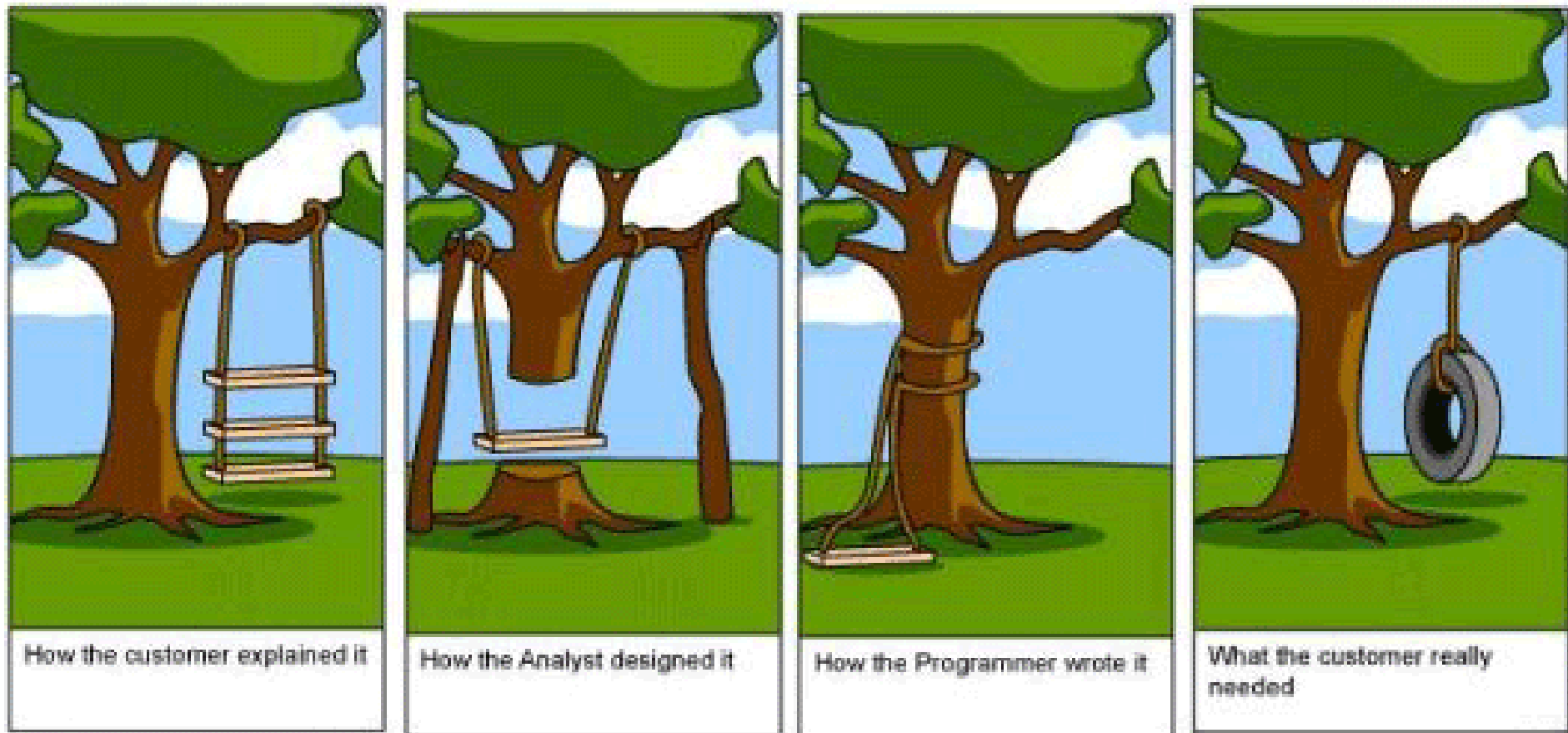
Sollte es in der SU nicht perfekt aufgehen mit der 1:1:1 Zuordnung wird Ihre Präsentation/Zusammenfassung trotzdem via Moodle verfügbar sein für die anderen (und den Prof)! (Upload nach der SU in Moodle)

## Agenda

- **Wiederholung**
- **NoSQL und Performance**
  - Konzeptvergleich mit MySQL
  - Alternativen
- **mongoDB**
  - Eigenschaften
  - Installation
- **Mongojs + Aufgabe**
- **Mongoose + Aufgabe**
- **node-restful**
  
- **Zusammenfassende Fragen**
- **Organisation des Inverted Classroom in 14Tg.**
- **Ausblick**

## Ausblick / Nächster Unterricht: REST APIs anzapfen mit Backbone.js

- (auch da gibt es Models)



# Vielen Dank und bis zum nächsten Mal





# Anhang

**Exkurs: MEAN Stack im Einsatz.  
Ein winziges Beispiel „in a nutshell“**

## Being MEAN (server-side)

- Starting a server  
and processing a request

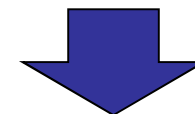


```
var express = require('express');  
var app = express();  
  
app.get('/', function(req, res){  
  res.send('Hello World');  
});  
  
app.listen(3000);
```

- Using the database to save JSON\*

```
var mongoDB = require('mongodb');  
  
var dbClient = mongoDB.MongoClient;  
dbClient.connect('mongodb://127.0.0.1:27017/mydb', function(err, db) {  
  if(err) throw err;  
  
  var tweet = {sender: 'John', text: 'Great lecture on MEAN stack'};  
  db.collection('tweets').insert(tweet, function(){  
    db.close();  
  });  
});
```

sender	John
text	Great lecture on MEAN stack



mongoose DB

## Being MEAN (server-side)

### ■ Together: using the database to send JSON



```
1 var express = require('express'), mongodb = require('mongodb');
2 var dbClient = mongodb.MongoClient;
3 var app = express();
4
5 app.get('/tweets/:name', function(req, res){
6   var name = req.params.name;
7
8   dbClient.connect('mongodb://127.0.0.1:27017/mydb', function(err, db) {
9     if(err) throw err;
10
11     db.collection('tweets').find({sender: name}).toArray(function(err, results) {
12       if(err) throw err;
13       res.send(results);
14       db.close();
15     });
16   });
17 });
```



sender	John	sender	John
text	Great lecture on MEAN stack	text	Yeah! My first tweet



DB

## Being MEAN (client-side)

- Dynamic binding of view to model

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4 <p>MyTweetApp</p>
5 <div ng-app="">
6
7 Name: <input type="text" ng-model="tweet.name"><br>
8 Tweet: <input type="text" ng-model="tweet.text"><br>
9 <br>
10 Preview: {{'"'+ tweet.text + '" said '+ tweet.name}}
11
12 </div>
13
14 <script src="./angular.js"></script>
15
16 </body>
17 </html>
```



MyTweetApp

Name:

Tweet:

Preview: "Cool" said John