Multimedia im Netz
Online Multimedia
Winter semester 2015/16

Tutorial 09 – Major Subject
Today’s Agenda

• Discussion: Intellectual Property and Fair Use

• MongoDB:
  – Syntax
  – Statements: Parallels to SQL
  – NodeJS modules

• Break-Out:
  – Database queries

• Quiz
Fair Use – Pros and Cons

• Research for arguments for and against “fair use” of copyrighted material on the web.

• Starting articles:
  http://t3n.de/news/fair-use-klausel-kommentar-662049/
  https://www.youtube.com/yt/copyright/de/fair-use.html
  http://www.dmlp.org/legal-guide/fair-use

• Group A: Pros
  Group B: Cons

• Discussion after 15 minutes.

• Possible focus:
  – Can you take a picture at a museum and post it to your Facebook timeline?
  – Can we use copyrighted material in the tutorial slides?
MongoDB

• huMONGOus
• "NoSQL"
  – No SQL
  – Not only SQL → not meant to replace SQL entirely
• No “schemas”, i.e. no structured constraints regarding the data
• No “join” paradigm, but aggregation possible.
• Document Driven / Object
  – Different types of documents in the same collection
  – Deep Query ability
  – Index on any attribute
• High Scalability
• JSON Interface

https://www.mongodb.com/nosql-explained
Potential reasons for a shift to NoSQL

• NoSQL handles multi-structured data more easily
  – Example: Storing arrays in a table column
  – Especially important during the development, when data structures change all the time

• Capacity
  – Scaling out is easier with NoSQL databases

• Many relational systems are proprietary on larger scale

• Direct mapping to objects
## Terminology

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>database</td>
</tr>
<tr>
<td>table</td>
<td>collection</td>
</tr>
<tr>
<td>row</td>
<td>document</td>
</tr>
<tr>
<td>column</td>
<td>field</td>
</tr>
<tr>
<td>index</td>
<td>index</td>
</tr>
<tr>
<td>table joins</td>
<td>embedded documents and linking</td>
</tr>
<tr>
<td>primary key</td>
<td>primary key</td>
</tr>
<tr>
<td>UNIQUE column</td>
<td>Automatically generated _id field</td>
</tr>
</tbody>
</table>
Install MongoDB on your machine

- Download here and run locally:
  http://www.mongodb.org/downloads
- Start daemon:
  $ /path/to/your/mongo/installation/bin/mongod
- Launch mongo:
  $ mongo [-u username -p [password]]
- Create a database:
  $ use mmn
- Verify:
  $ show dbs
Alternative: mongolab.com

Attention: You need Mongo Version > 3.0 on your machine!
Verify by typing `mongo --version` in a terminal.
Using mongo client from CIP Pool

1. Open a terminal and perform steps 1-3 from this tutorial: https://docs.mongodb.org/v3.0/tutorial/install-mongodb-on-linux/

2. Navigate to the bin folder of the mongodb sources
   `cd ~/mongodb/…/bin`

3. Launch mongo client:
   `./mongo <address from mongolab.com> -u <username> -p`

If you have time and want to enable mongo permanently:
1. Type `pwd` to find out the full, absolute path of the folder containing the mongo binary
2. Open the .bashrc_local file in your home directory with a text editor, like so:
   `gedit ~/.bashrc_local`
3. Put a new line in there and replace `<...>` with your correct path:
   `export PATH=<output from pwd>:$PATH`
4. Save the file, close the terminal, re-open the terminal.
5. Type mongo --version to see if it works.
Try the connection to mongolab.com

Last login: Fri Dec 11 16:59:03 on ttys006
You have new mail.

$ mongo ds027345.mongolab.com:27345/mmn-1516 -u tobiasseitz -p
MongoDB shell version: 3.0.7
Enter password:
connecting to: ds027345.mongolab.com:27345/mmn-1516
rs-ds027345:PRIMARY> show collections
foos
system.indexes
users
rs-ds027345:PRIMARY>
Basics

• There is a global object named `db`
  – “collections” are accessible via `db`’s attributes
  – collections are also objects that have a number of methods

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

- global Object
- currently used database
- property
- name of the collection
- property
- method of the collection
Creating Collections

- Collections are created **implicitly in MongoDB** (as are databases)
- Alternative:
  
  ```javascript
  db.createCollection("collectionName")
  ```

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
</table>
| **CREATE TABLE** users ( id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id) ) | `db.users.insert(
  {
    user_id: "abc123",
    age: 55,
    status: "A"
  })` |
## Inserting Data

- Inserts are Javascript / JSON Objects
- Multiple objects can be wrapped into an array and then inserted

### SQL

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

### MongoDB

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

• Pull the changes from the tutorials repository
  git pull origin master

• Open the file here:
  tutorials-15-16/tutorial09/breakout/users.json

• Copy the file content and launch a mongo query in the
  console to insert the data to the “users” collection

• Timeframe: 5 Minutes
Multiple types in the same collection

- `db.foos.insert({foo: 'bar'})`;
- `db.foos.insert({foo: 847})`;
- `db.foos.insert({foo: new Date()})`;

```
{  
   "_id": ObjectId("566965c6ce9e741b3fe6b219"),
   "foo": "bar"
}
{  
   "_id": ObjectId("566965dbce9e741b3fe6b21a"),
   "foo": 847
}
{  
   "_id": ObjectId("566965e5ce9e741b3fe6b21b"),
   "foo": ISODate("2015-12-10T11:45:41.224Z")
}
```

- Con: Developers have to be careful to prevent inconsistencies
- Pro: Mongo is really flexible!
db.collection.find(query, projection)

• query:
  – Similar to SQL “where” clause
  – Optional
  – object (selection)
  – Example: { status : "A" }

• projection
  – Similar to column list in SQL
  – Indicate which fields of a document the query should return.
  – Example: { status : true, _id: false }
## Querying with `.find()`

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT * FROM users</strong></td>
<td><code>db.users.find()</code></td>
</tr>
</tbody>
</table>
| **SELECT id, user_id, status FROM users** | `db.users.find(
   {}),
   {
     user_id: 1, status: 1
   }
)` |
| **SELECT user_id, status FROM users** | `db.users.find(
   {}),
   {
     user_id: 1, status: 1, _id: 0
   }
)` |
| **SELECT * FROM users WHERE status = "A"** | `db.users.find(
   { status: "A" }
)` |
| **SELECT user_id, status FROM users WHERE status = "A"** | `db.users.find(
   { status: "A" },
   {
     user_id: 1, status: 1, _id: 0
   })` |

Pretty print the output: `db.collection.find().pretty()`
Operators

- Operators are “special keys” inside queries in MongoDB
- You cannot write ‘someKey’ != ‘someValue’.
- Most common operators:
  - $ne, $gt, $lt, $gte, $lte
  - $and, $or
  - $in
- Example:

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT * FROM users WHERE age &gt; 50</code></td>
<td><code>db.users.find({ age : { $gt : 50} })</code></td>
</tr>
</tbody>
</table>

http://docs.mongodb.org/manual/reference/operator/
Breakout: Querying

• Continue with the users collection from before.

• Do the following queries:
  1. Find all users with status A and who are older than 30 years.
  2. Find all users with either status B or who are older than 30 years.
  3. Only query the name of users who are younger than 30 years.

• Time frame: 5-10 minutes
## Update & Delete

**db.collection.update**(query, update, options)

### SQL vs. MongoDB

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPDATE</strong> users <strong>SET</strong> status = &quot;C&quot; <strong>WHERE</strong> age &gt; 25</td>
<td><code>db.users.update( { age: { $gt: 25 } }, { $set: { status: &quot;C&quot; } }, { multi: true } )</code></td>
</tr>
<tr>
<td><strong>UPDATE</strong> users <strong>SET</strong> age = age + 3 <strong>WHERE</strong> status = &quot;A&quot;</td>
<td><code>db.users.update( { status: &quot;A&quot; }, { $inc: { age: 3 } }, { multi: true } )</code></td>
</tr>
</tbody>
</table>

**db.collection.remove()**

<table>
<thead>
<tr>
<th>SQL</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DELETE FROM</strong> users <strong>WHERE</strong> status = &quot;D&quot;</td>
<td><code>db.users.remove( { status: &quot;D&quot; } )</code></td>
</tr>
<tr>
<td><strong>DELETE FROM</strong> users</td>
<td><code>db.users.remove({})</code></td>
</tr>
</tbody>
</table>
Breakout: Update

• Continue with the users collection from before

• Update Caroline’s age to 56.

• Insert a new user with age 100.

• Remove all users with age > 70.
NodeJS and MongoDB

• There are a couple of implementations for NoSQL/MongoDB drivers and middleware in NodeJS

• For MongoDB, the most prevalent examples are
  – monk (https://www.npmjs.com/package/monk)
  – mongoose (https://www.npmjs.com/package/mongoose)

• In the tutorial, we use monk because it is very simple. Mongoose is more sophisticated. If you plan to do a larger project, we suggest you consider mongoose instead of monk.
Using monk as MongoDB layer

• Connecting to the database:
  ```javascript
  var db = require('monk')('localhost/databasename');
  ```

• We can make the connection object available to all routes like this.
  ```javascript
  app.use(function(req, res, next) {
    req.db = db;
    next();
  });
  ```
  This has to come early in the middleware chain
Basic Operations

• Accessing a collection:
  ```javascript
  var users = db.get('users');
  ```

• Queries are asynchronous:
  ```javascript
  users.find({}, function(err, docs) {
    if (err) {
      // there was an error
    }
    // do something with the documents
  });
  ```
Accessing the DB from Middleware

```javascript
var express = require('express');
var router = express.Router();

router.get('/', function(req, res, next) {
  var users = req.db.get('users');
  users.find({}, function(e, docs) {
    if (!e) {
      res.json(docs);
    } else {
      next(e);
    }
  });
});

module.exports = router;
```

Note: it’s not necessary to require monk here! Why?
Round-Up

1. What will be logged first?
   
   ```javascript
   router.get('/', function(req, res, next) {
       var users = req.db.get('users');
       users.find({}, function(e, docs) {
           console.log("I found some users.");
           console.log("I'm feeling quizzical.");
       });
   });
   ```

2. Where is the conceptual error here?
   
   ```javascript
   router.get('/spottheerror', function(req, res) {
       req.db.get('users').find({}, function(e, docs) {
           res.send("Now I have some data");
           res.send("Ok, request received");
       });
   });
   ```
Thanks!
What are your questions?