| | Course | **Datenbanken und verteilte Systeme 2015** | | |
|---|---|---|---|---|
| | | | | |
| | – | | | |
| | Release Date | **August 31 2015** | Due Date | – |

# Exercise 1: Tutorial

In this tutorial we are using Windows as operating system. If necessary, you have to adjust some operating system specific commands for your operating system and have to use the correct MongoDB version.

**Preparations**

First, download and extract the current MongoDB distribution[1] (e.g. into `C:/Users/<Username>/`), open a shell and navigate to the MongoDB binary directory. Start the client and connect to a remote database (URLs can be requested from your tutors):

```
cd "C:/Users/<Username>/mongodb/bin/"
mongo <URL>/db
```

You are now connected to a database with name `test`, but for this tutorial, switch to database `imdb`:

```
use imdb
```

Databases and collections in MongoDB are created implicitly while data is inserted. In this tutorial, you will create an `imdb` database with a collection of `films`.

You can list the available databases, . . .

```
show dbs
```

. . . but the `imdb` database does not exist, yet. There are also no collections, so far, . . .

```
show collections
```

. . . so create one by inserting a document:

```
db.films.insert({
  title: "Star Trek Into Darkness",
  year: 2013,
  genre: [
    "Action",
    "Adventure",
    "Sci-Fi",
    ],
  actors: [
    "Pine, Chris",
    "Quinto, Zachary",
    "Saldana, Zoe",
  ],
  releases: [
    {
      country : "USA",
      date : ISODate("2013-05-17"),
      prerelease: true
    },
    {
```

---

[1]https://www.mongodb.org/downloads.

```
      country : "Germany",
      date : ISODate("2003-05-16"),
      prerelease: false
    }
  ]
})
```

As you can verify by calling `show collections` again, now there is a `films` collection.

You can list the contents of the newly created collection by calling the `find()` function:

```
db.films.find()
```

If you prefer your result nicely formatted, use `pretty()`:

```
db.films.find().pretty()
```

As you can see, now there is an `_id` field which is unique for every document.

Now insert some more films:

```
db.films.insert({
  title: "Iron Man 3",
  year: 2013,
  genre: [
    "Action",
    "Adventure",
    "Sci-Fi",
    ],
  actors: [
    "Downey Jr., Robert",
    "Paltrow, Gwyneth",
  ]
}) // no releases

db.films.insert({
  title: "This Means War",
  year: 2011,
  genre: [
    "Action",
    "Comedy",
    "Romance",
    ],
  actors: [
    "Pine, Chris",
    "Witherspoon, Reese",
    "Hardy, Tom",
  ],
  releases: [
    {
      country : "USA",
      date : ISODate("2011-02-17"),
      prerelease: false
```

```
    },
    {
      country : "UK",
      date : ISODate("2011-03-01"),
      prerelease: true
    }
  ]
})

db.films.insert({
  title: "The Amazing Spider-Man 2",
  year: 2014,
  genre: [
    "Action",
    "Adventure",
    "Fantasy",
    ],
  actors: [
    "Stone, Emma",
    "Woodley, Shailene"
  ]
}) // also no releases
```

**Querying**

Now query your collection! Have MongoDB return all films with title `"Iron Man 3"` by calling:

```
db.films.find({title: "Iron Man 3"})
```

Using `findOne` instead of `find` produces at most one result (in pretty format):

```
db.films.findOne({title: "Iron Man 3"})
```

Regular expressions can also be used to query a collection. In this tutorial, a short notation is used where the actual regular expression is bounded by slashes (/). The following call yields all movies that start with the letter T:

```
db.films.find({title: /^T/})
```

If you are only interested in certain attributes, you can use projection to thin out the produced result. While the selection criteria are given by the first argument of `find`, the projection is given by the second argument. An example:

```
db.films.find({title: /^T/}, {title: 1})
```

By default, the `_id` is part of the output, so you have to explicitly suppress it, if you don't want to have it returned by MongoDB:

```
db.films.find({title: /^T/}, {_id: 0, title: 1})
```

You can also use conditional operators, for example to perform range queries. The following returns the titles of all films starting with the letter T where the year attribute is greater than 2009 and less than or equal to 2011:

```
db.films.find({
   year: {
      $gt:  2009,
      $lte: 2011
   },
   title: /^T/
},
{ _id: 0,
   title: 1
})
```

For a logical disjunction of the selection criteria, use the `$or` operator:

```
db.films.find({
   $or: [
      {year: {
         $gt:  2009,
         $lte: 2011
      }},
      {title: /^T/}
   ]
},
{ _id: 0,
   title: 1
})
```

There are also some options that can be appended to the regular expression, e.g. `i` to achieve case-insensitivity. The following call returns the titles of all movies whose title contains lowercase t, . . .

```
db.films.find({title: /t/}, {_id: 0, title: 1})
```

. . . whereas the following call also returns titles that contain a T (uppercase):

```
db.films.find({title: /t/i}, {_id: 0, title: 1})
```

You can query for exact matches in lists, . . .

```
db.films.find({genre: "Adventure"}, {_id: 0, title: 1, genre: 1})
```

. . . but you can also query for partial matches:

```
db.films.find({genre: /^A/}, {_id: 0, title: 1, genre: 1})
```

There are also more complex operators for more complex selection criteria, e.g. the `$all` operator. The following call prints the title and actors of every movie for which each of two given regular expressions matches at least one of its actors:

```
db.films.find({actors: {$all: [/pine/i, /zachary/i]}}, {_id: 0, title:
   1, actors: 1})
```

In contrast, the `$nin` operator checks for the lack of matching values, i.e. actor names that do not match either one of the given regular expressions:

```
db.films.find({actors: {$nin: [/pine/i, /zachary/i]}}, {_id: 0, title:
   1, actors: 1})
```

The `$exists` operator can be used to check for the existence of an attribute, e.g. to select only movies with undefined `releases`:

```
db.films.find({releases: {$exists: false}}, {_id:0, title: 1})
```

In MongoDB, it is also possible to query nested data, i.e. subdocuments. The following returns the title and releases of every movie that is known to be released in the UK:

```
db.films.find({'releases.country': "UK"}, {_id:0, title: 1, releases:
    1})
```

Please note that you have to use quotes to address nested fields.

Applying more complex selection criteria on a nested document, however, is a little tricky. For example, if you wanted MongoDB to return all movies that had their prerelease in the USA, you might try something like this:

```
db.films.find({'releases.country': "USA", 'releases.prerelease':
    true}, {_id:0, title: 1, releases: 1})
```

However, This Means War is also returned, but was prereleased in the UK. The call above actually returns all movies that have some prerelease *or* were released in the USA. To only select movies were both applies to the same release, the `$elemMatch` can be used:

```
db.films.find({
  releases: {
    $elemMatch: {
      country: "USA",
      prerelease: true
    }
  }
},
{_id: 0, title: 1, releases: 1}
)
```

Naturally, there are many other operators not covered by this tutorial.

**Update**

You can also add or update fields in a document by using the `$set` operator. For example, you can add a `rating` field to one of the movies:

```
db.films.update(
  {title: "Star Trek Into Darkness"},
  {$set: {rating: 6.4}}
)
```

If you do not use the `$set` operator, every document fulfilling the selection criteria will be replaced, so be careful!

To increment a number of value, you can use the `$inc` operator:

```
db.films.update(
  {title: "Star Trek Into Darkness"},
  {$inc: {rating: 0.1}}
)
```

Again, there are many other different operators for different purposes, e.g. $unset, $inc, $pop, $push, $pushAll or $addToSet.

### Delete

You can remove documents with the `remove` function. It actually works almost like the `find` function; you only don't use the projection parameter. If, for example, you want to remove all film documents whose title starts with the letter T, you can first query for all such movies. . .

```
db.films.find({title: /^T/})
```

. . . to verify that your selection criteria is correct and then replaced the `find` in your call by `remove`:

```
db.films.remove({title: /^T/})
```

### Speed Up Queries By Indexing

Quit MongoDB by typing

```
exit
```

Now copy the file `movies.json` into the `C:/Users/<Username>/mongodb/` directory and navigate to `C:/Users/<Username>/mongodb/bin/`:

```
cd "C:/Users/<Username>/mongodb/bin/"
```

The file `movies.json` contains many IMDb movies which you import into a new `movies` collection:

```
mongoimport.exe --db imdb --collection movies --file "../movies.json"
```

Now reconnect to the MongoDB instance and perform the following query:

```
mongo imdb
db.movies.find({
  rating: {
    $gt: 6.14,
    $lt: 7.78
  }
}).explain()
```

Note that `explain()` gives you information on the query execution. Write down the amount of time the execution of your query took!

To make the query execution of a little faster, create an index on the `rating` field. . .

```
db.movies.ensureIndex({rating: 1})
```

. . . and repeat the query! It now should have been performed much faster.