Hands-on MongoDB Development

Contents

Introduction

In this hands-on, you will be working with MongoDB – in the MongoDB shell on advanced query functionality, with Node.JS and Java to interact from those technologies with a MongoDB database and through MongoDB Compass – a GUI on top of MongoDB.

Each of the sections is optional and do not depend on each other.

The assumption is that you have a running MongoDB server in your workshop environment and that you can connect to that instance through the Mongo Shell.

Additional requirements apply to each of the sections – a Node.JS and npm installation, a Java and Maven installation and a Compass installation.

All sources, instructions and the presentations can be found in this GitHub Repository: https://github.com/lucasjellema/sig-nosql-mongodb . You can fetch the contents of this repository as a zip file (https://github.com/lucasjellema/sig-nosql-mongodb/archive/master.zip) or use a git client to clone the repository to your workshop environment.

1 Advanced Query – and a comparison between MongoDB Find and Aggregate vs. (Oracle) SQL

In this section, you will perform a number of query actions against a MongoDB database that contains two collections that are similar to data sets frequently used in samples and tutorials for relational databases – especially Oracle Database: Departments and Employees. You will perform a number of MongoDB search operations against these collections – and you can compare what you can do with MongoDB put next to (Oracle) SQL.

A presentation is available that presents 30 queries against MongoDB and Oracle SQL side by side: https://github.com/lucasjellema/sig-nosql-mongodb/blob/master/mongodb-find-agg-vs-oracle-sql.pptx - you can use this presentation for reference.

1. Load Data Set from CSV files into MongoDB

   Directory hr-queries contains the data sets in two CSV files: export_dept.csv and export_emp.csv. Two helper files - empFields.txt and deptFields.txt – specify how the mongoimport utility should map the fields in each record to document properties in the MongoDB collection.

   Using the following commands – running the monogimport tool that is installed in the bin directory of your MongoDB installation that also holds the server runtime and the shell – you import the datasets into collections emp and dept in database called hr. The database and the collections will be dropped (if they already exist) and (re)created through these commands.

   ```
   mongoimport --host 127.0.0.1:27017 --db hr --collection emp --drop --
   file export_emp.csv  --type csv --fieldFile empFields.txt
   ```

   ```
   mongoimport --host 127.0.0.1:27017 --db hr --collection dept --drop --
   file export_dept.csv  --type csv --fieldFile deptFields.txt
   ```

2. Run MongoDB Shell and Inspect Collections

   Start the MongoDB shell as you have done before. Switch to database hr:

   ```
   use hr
   ```

   Verify whether the collections were created:

   ```
   show collections
   ```

   Check the statistics for collection emp:

   ```
   db.emp.stats()
   ```

Check the contents for both collections:

```
db.emp.find()

db.dept.find()
```

3. Perform queries

To get started, let's find the names of all managers:

```
db.emp.find({"JOB":"MANAGER"},{ENAME:1})
```

And – find all salesmen, listed by salary descending; only show top 2

```
db.emp.find({"JOB":"SALESMAN"},{ENAME:1, SAL:1}).sort({'SAL':-
1}).limit(2)
```

Introducing lookup in MongoDB: list all employees (empno and ename) along with the looked up department

```
db.emp.aggregate(

[  {$lookup:

      {

         from:"dept",

         localField:"DEPTNO",

         foreignField:"deptno",

         as:"dept"

     }

   }

, {$project: {

        "EMPNO": 1,

        "ENAME": 1,

        "DEPT": { $arrayElemAt:["$dept", 0]},

      }

    }

]

)
```

4. The file sample-hr-queries.txt contains a number of queries, of increasing complexity, that you can try out to see what MongoDB can do in terms of retrieving information.

   You will come across:
   * aggregation
   * join & lookup
   * nested collections
   * geo spatial search (cities closest to…)
   * text search
   * materialized views
   * stored procedures
   * facet search

   Note: You can compare the MongoDB search commands with their SQL counterparts in the presentation: https://www.slideshare.net/lucasjellema/comparing-30-mongodb-operations-with-oracle-sql-statements.

2. MongoDB Compass (aka NoSQL Developer)

MongoDB Compass is the GUI visual explorer on top of MongoDB. Note that MongoDB Compass is a separate install that can run in any environment against a MongoDB server in a different environment. You can find the documentation for Compass at: https://docs.mongodb.com/compass/current/ .

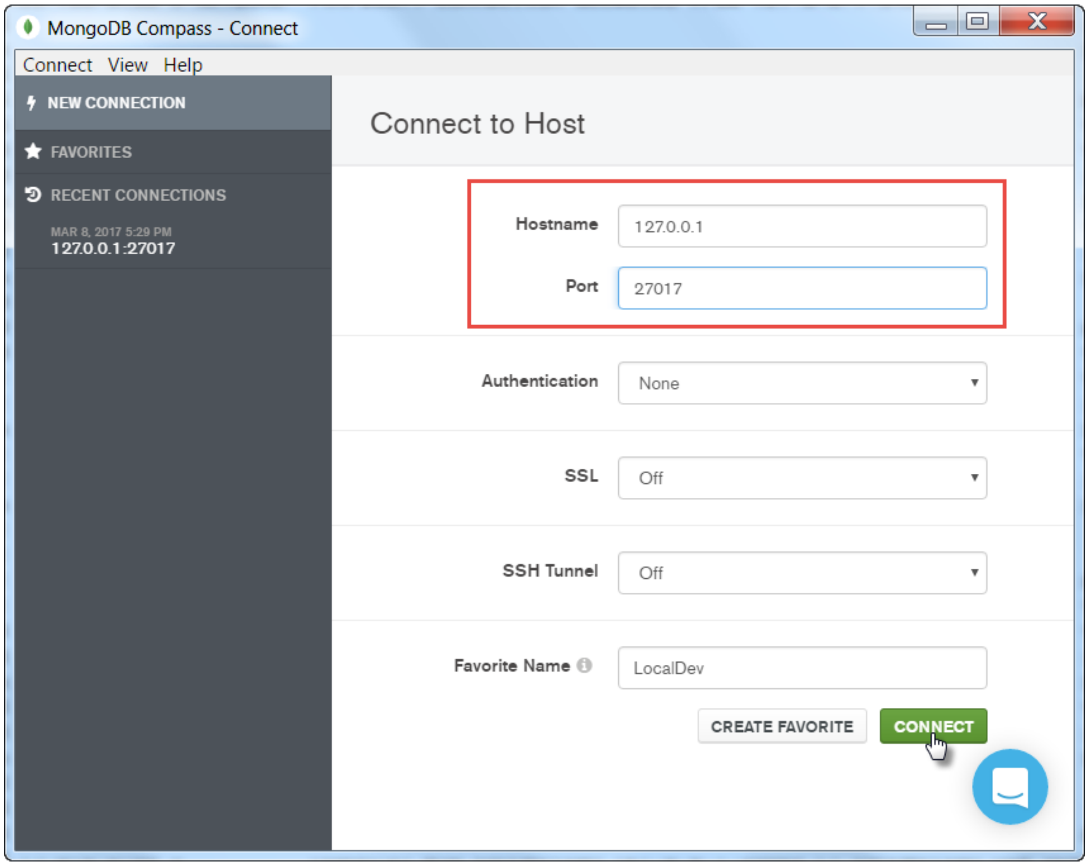
1. Install MongoDB Compass

   Download the MongoDB Compass installer for your environment from https://www.mongodb.com/download-center?jmp=hero#compass.

   Run the installer and complete the installation wizard.

2. Run MongoDB Compass and Connect to local MongoDB Server

   Start MongoDB Compass. Enter Hostname – 127.0.0.1 – and Port – 27017 – for your local MongoDB server. Then press Connect:

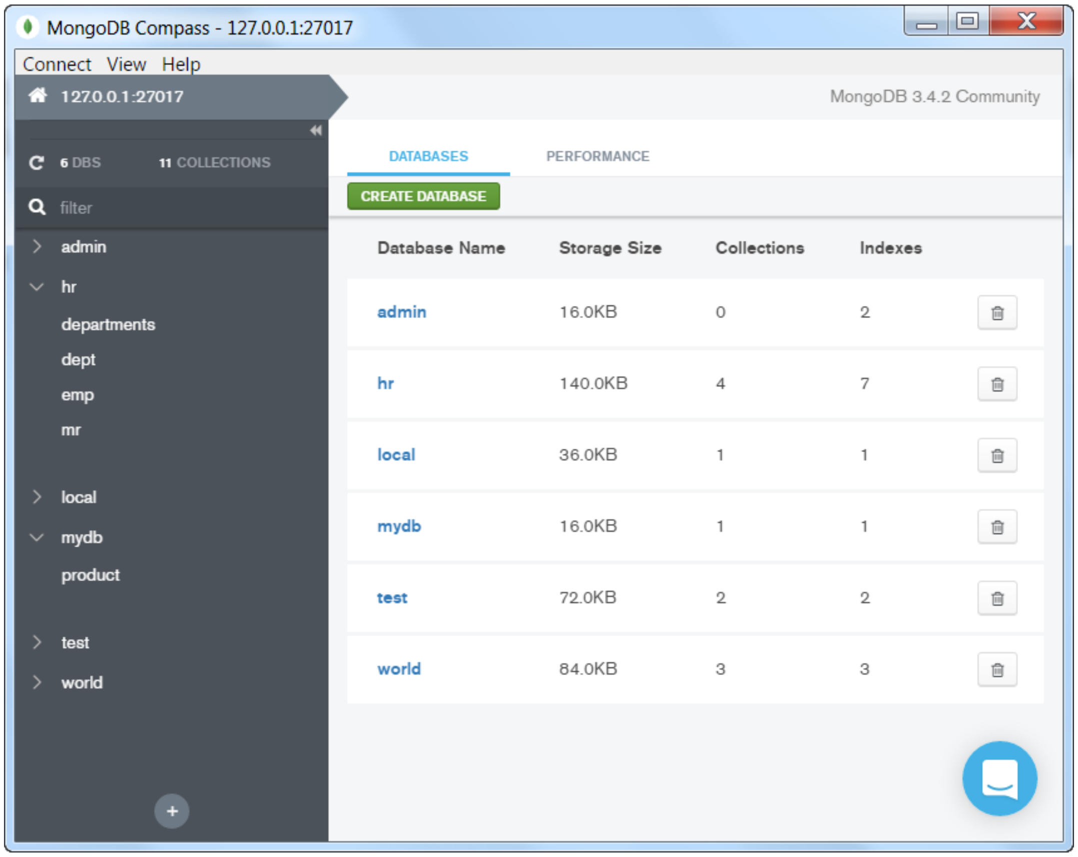


MAR 8, 2017 5:29 PM
127.0.0.1:27017

Connect to Host

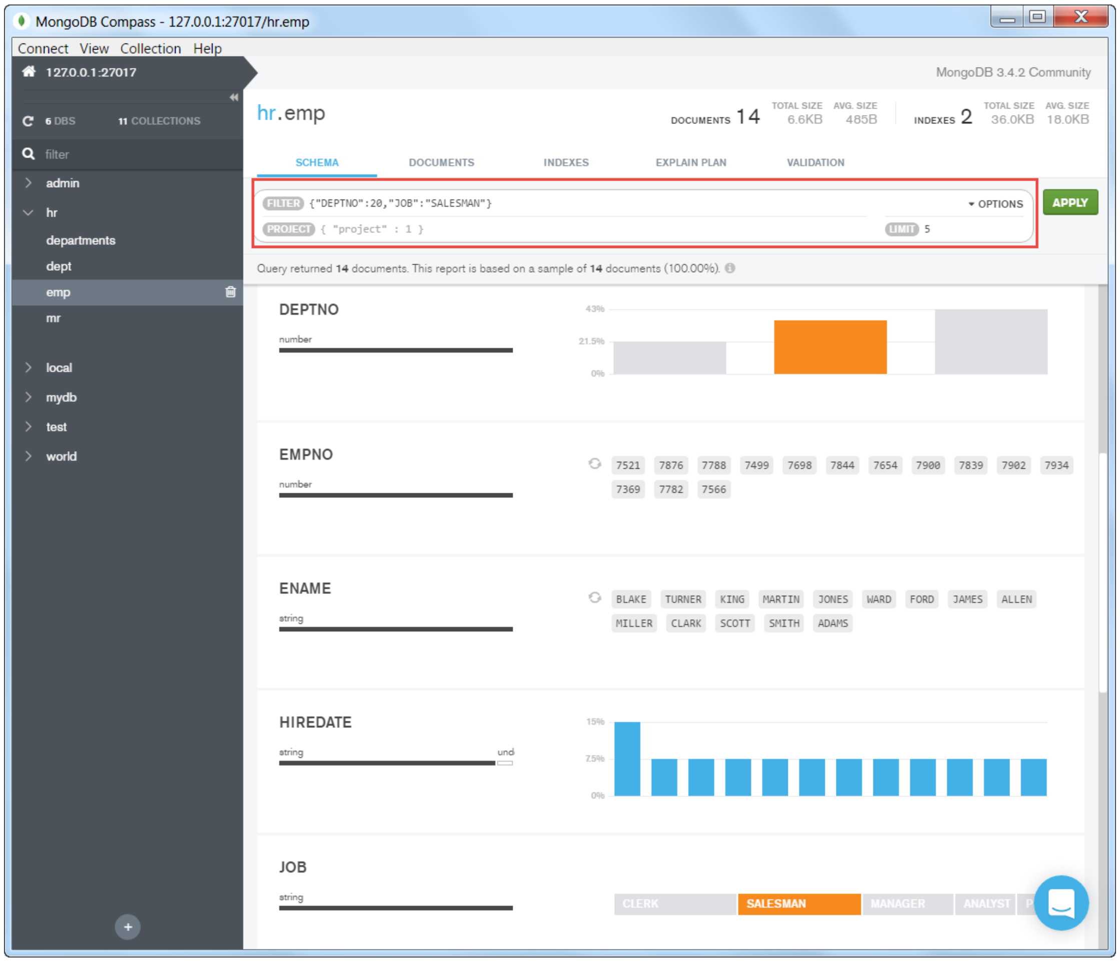
Hostname 127.0.0.1
Port 27017
Authentication None
SSL Off
SSH Tunnel Off
Favorite Name LocalDev

   The connection should be created and you should see a list of databases in your server:



Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	2
hr	140.0KB	4	7
local	36.0KB	1	1
mydb	16.0KB	1	1
test	72.0KB	2	2
world	84.0KB	3	3

3. Navigate to one of the collections in one of the databases. Explore the collection – the properties (and their values) in the documents. Try to build a query using the collection explorer.

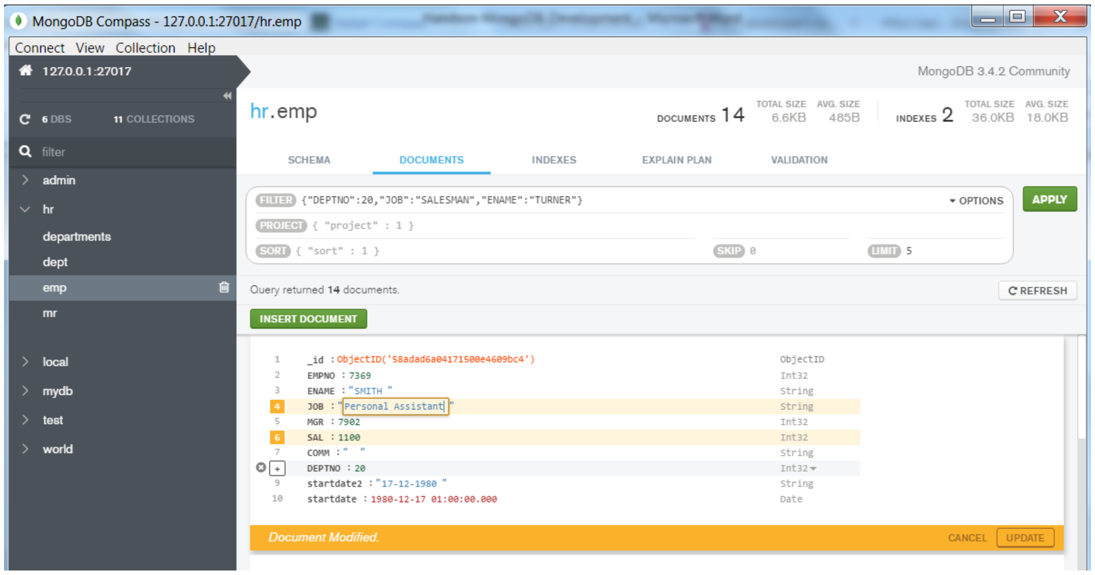


```
FILTER {"DEPTNO":20,"JOB":"SALESMAN"}
PROJECT { "project" : 1 }    LIMIT 5
```

Query returned 14 documents. This report is based on a sample of 14 documents (100.00%).

```
EMPNO: 7521 7876 7788 7499 7698 7844 7654 7900 7839 7902 7934 7369 7782 7566
ENAME: BLAKE TURNER KING MARTIN JONES WARD FORD JAMES ALLEN MILLER CLARK SCOTT SMITH ADAMS
```

4. Edit one of the records:



```
FILTER {"DEPTNO":20,"JOB":"SALESMAN","ENAME":"TURNER"}
PROJECT { "project" : 1 }
SORT { "sort" : 1 }    SKIP 0    LIMIT 5
```

Query returned 14 documents.

```
_id : ObjectID('58adad6a04171500e4609bc4')    ObjectID
EMPNO : 7369                                  Int32
ENAME : "SMITH "                              String
JOB : "Personal Assistant "                   String
MGR : 7902                                    Int32
SAL : 1100                                    Int32
COMM : "  "                                   String
DEPTNO : 20                                   Int32
startdate2 : "17-12-1980 "                    String
startdate : 1980-12-17 01:00:00.000           Date
```

Document Modified.

and/or add a new record.

3. Development with NodeJS and MongoDB

This section requires your workshop environment to include a NodeJS server – a fairly recent version at that, such as v6.10 that includes npm which is also required. You can download NodeJS from https://nodejs.org/en/download/ .

Note: these practices assume that the countries data set has been loaded into a Mongodb database called *world*. This can easily be done with the following steps:

- open a command line or terminal window in directory *country-queries*.
- import the file countries.csv into collection *countries* in database *world* using the following command (assuming that the bin directory of the MongoDB installation is in the PATH environment variable:

  ```
  mongoimport --host 127.0.0.1:27017 --db world --collection countries --
  drop --file countries.csv  --type csv --fieldFile countriesFields.txt
  ```

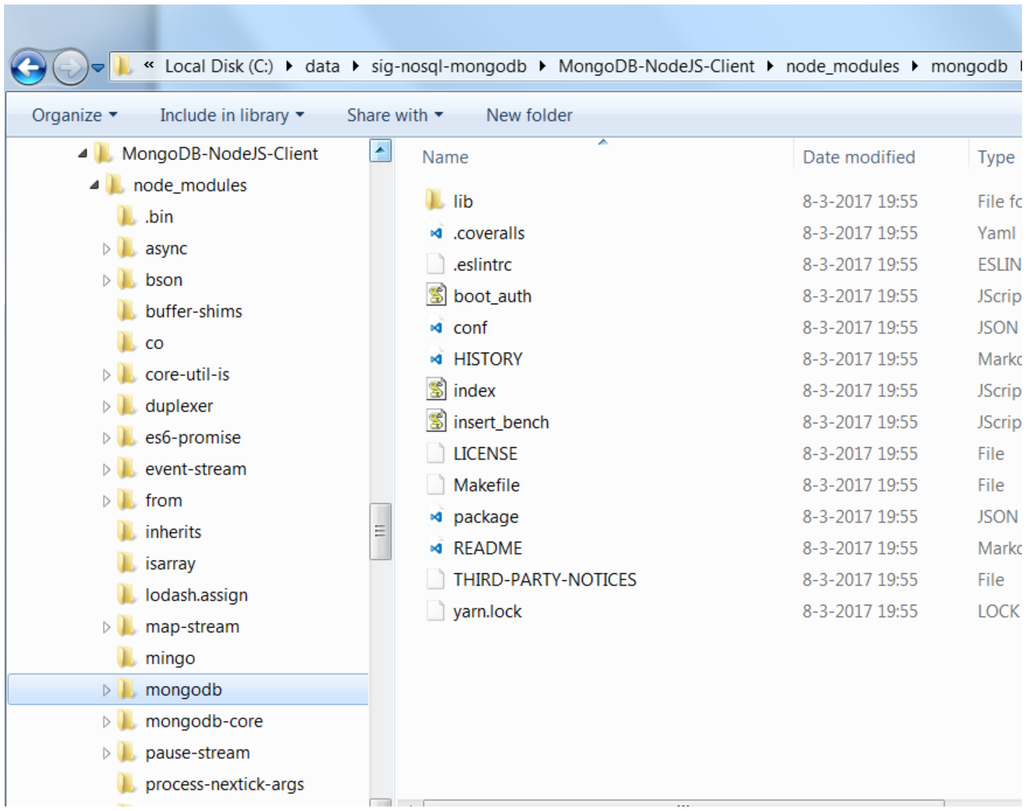
With NodeJS installed and the MongoDB database and collection created, you can turn to the sources in directory MongoDB-NodeJS-Client.

1. Open a command line or terminal window in directory MongoDB-NodeJS-Client.

2. Execute

   ```
   npm install
   ```

   This will have package.json interpreted by npm, resulting in the download of the MongoDB Driver for NodeJS module as well as a number of dependent modules, all into the node-modules subdirectory:


« Local Disk (C:) ▸ data ▸ sig-nosql-mongodb ▸ MongoDB-NodeJS-Client ▸ node_modules ▸ mongodb
Organize ▾
Include in library ▾
Share with ▾
New folder

MongoDB-NodeJS-Client
- node_modules
  - .bin
  - async
  - bson
  - buffer-shims
  - co
  - core-util-is
  - duplexer
  - es6-promise
  - event-stream
  - from
  - inherits
  - isarray
  - lodash.assign
  - map-stream
  - mingo
  - mongodb
  - mongodb-core
  - pause-stream
  - process-nextick-args

Name	Date modified	Type
lib	8-3-2017 19:55	File fc
.coveralls	8-3-2017 19:55	Yaml
.eslintrc	8-3-2017 19:55	ESLIN
boot_auth	8-3-2017 19:55	JScrip
conf	8-3-2017 19:55	JSON
HISTORY	8-3-2017 19:55	Markc
index	8-3-2017 19:55	JScrip
insert_bench	8-3-2017 19:55	JScrip
LICENSE	8-3-2017 19:55	File
Makefile	8-3-2017 19:55	File
package	8-3-2017 19:55	JSON
README	8-3-2017 19:55	Markc
THIRD-PARTY-NOTICES	8-3-2017 19:55	File
yarn.lock	8-3-2017 19:55	LOCK

3. Open file connect.js in directory MongoDB-NodeJS-Client in your favorite text editor. Verify and change if required the values of the variables mongodbHost, mongodbPort and mongodbDatabase.

   Then run this NodeJS module, using:
   ```
   node connect
   ```

4. Open the file *query6.js*. This file uses Promises instead of asynchronous callbacks. If you prefer the callbacks or have an older version of NodeJS that does not support the Promises, then use file query.js.

   The code performs the following steps – check how these steps are implemented in NodeJS, using the NodeJS driver for MongoDB
   * connect
   * fetch the top 20 largest countries (by area) into an array of country objects (and print two randomly selected countries from that array)
   * retrieve a cursor that returns all countries in Asia; get the number of records from the cursor; iterate through the cursor and write the name of each country returned from the cursor
   * retrieve through the aggregation framework using an aggregation query object a cursor that returns the largest country for each continent and write out the full country object

Note how well NodeJS can work with the JSON documents returned from MongoDB: these documents are native data objects in JavaScript and therefore in the NodeJS application.

5. Open the file *crud6.js*. This file uses Promises instead of asynchronous callbacks. If you prefer the callbacks or have an older version of NodeJS that does not support the Promises, then use file crud.js.

   The code performs the following steps – check how these steps are implemented in NodeJS:
   * connect
   * create a new collection (or work with an existing one) and add a list of two documents to it in a single statement (insertMany)
   * verify the creation of the documents by retrieving the newly created documents using a cursor
   * update one of the records, using a find filters and by applying a complex $set definition – with nested attributes and nested collections
   * verify the updated document
   * delete all documents
   * drop the collection

6. Please feel free to try out some more interaction between NodeJS and MongoDB. The documentation for the driver can be found here: http://mongodb.github.io/node-mongodb-native/2.0/ .

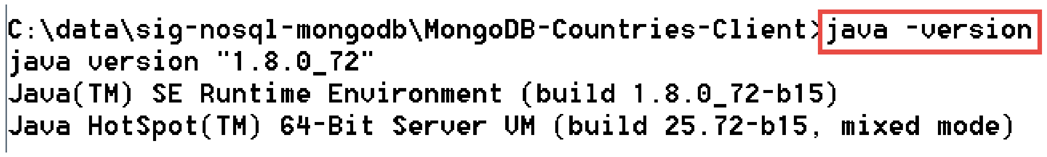
4. Development with Java and MongoDB

This section shows how MongoDB can be interacted with from Java applications. Because the interaction is done over TCP, it does not matter where the Java application runs relative to the MongoDB server – provided there is network connectivity. And whether the Java code runs as Java SE in a JVM or in a Java EE container does not matter. No special Java EE facilities are required.

Relevant resources:

- docs on MongoDB Java Driver: https://docs.mongodb.com/ecosystem/drivers/java/ and the Quickstart Guide: http://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/
- Java Driver API: http://api.mongodb.com/java/current/

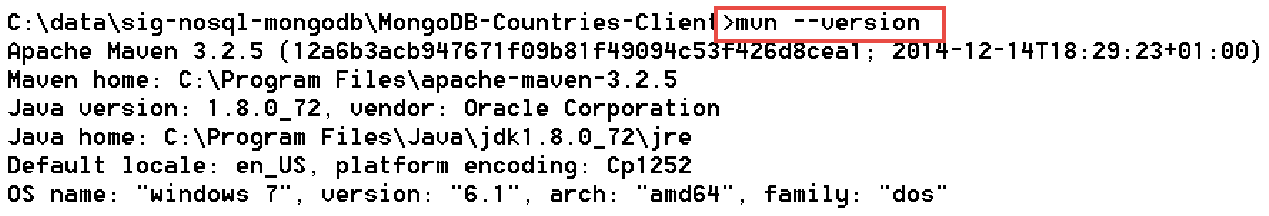
For this section, it is assumed that your workshop environment has Java 7 or 8 (or even 9) installed and running (java –version on the command line returns a response):



```
C:\data\sig-nosql-mongodb\MongoDB-Countries-Client>java -version
java version "1.8.0_72"
Java(TM) SE Runtime Environment (build 1.8.0_72-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.72-b15, mixed mode)
```

Installing the Java run time can be done through http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html .

It is also required to have Maven active. That means that mvn --version returns a meaningful response:



```
C:\data\sig-nosql-mongodb\MongoDB-Countries-Client>mvn --version
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-14T18:29:23+01:00)
Maven home: C:\Program Files\apache-maven-3.2.5
Java version: 1.8.0_72, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_72\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

If you do not have Maven 3 set up in your environment, you can download Maven from https://maven.apache.org/download.cgi and follow the installation instructions here: https://maven.apache.org/install.html (these basically amount to extracting the downloaded file to a specific directory and adding that directory to the PATH variable).

Once the Java Runtime and the Maven setup are complete, there is one final step: these practices assume that the countries data set has been loaded into a Mongodb database called *world* – the same database and data used in the previous section on NodeJS. This can easily be done with the following steps:

- open a command line or terminal window in directory *country-queries*.
- import the file countries.csv into collection *countries* in database *world* using the following command (assuming that the bin directory of the MongoDB installation is in the PATH

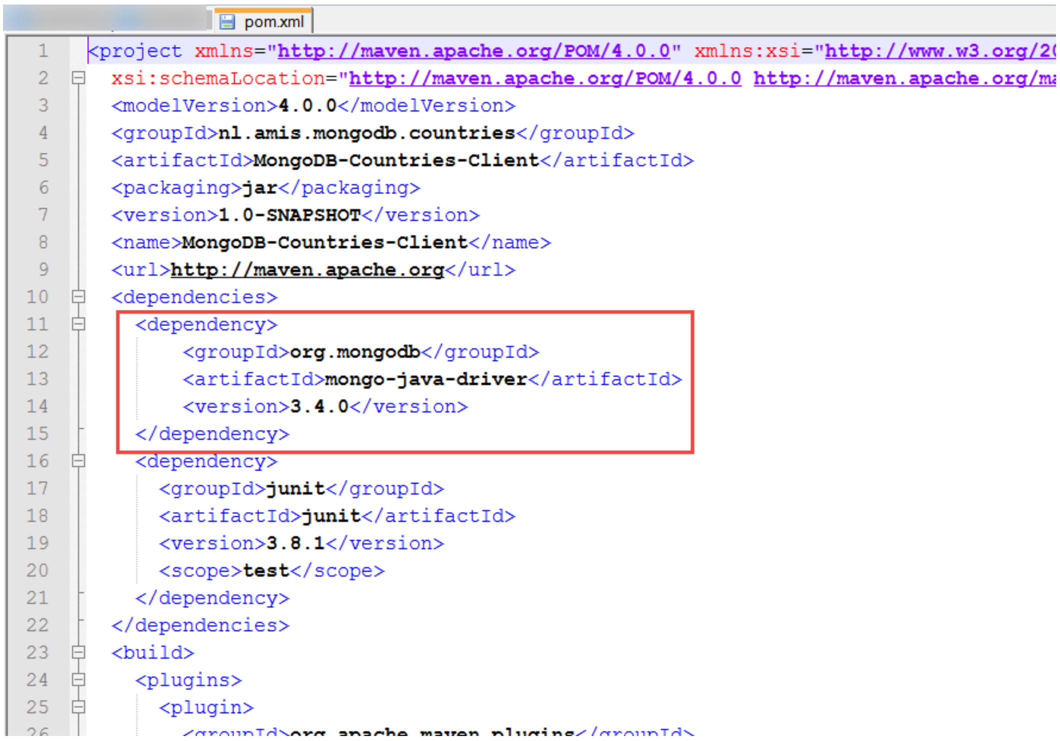
environment variable:

```
mongoimport --host 127.0.0.1:27017 --db world --collection countries --
drop --file countries.csv  --type csv --fieldFile countriesFields.txt
```

1. Open the file AppConnect.java in subdirectory src\main\java\nl\amis\mongodb\countries. This Class contains the code for connecting to a database on a locally running MongoDB server. Verify whether the connect details are applicable for your workshop environment – and modify them to the correct values if they are not.

2. Open a command line or terminal window in directory MongoDB-Countries-Client

3. Check the contents of file pom.xml. Crucial element in this file is the dependency on org.mongodb.mongo-java-driver:



```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/ma
  <modelVersion>4.0.0</modelVersion>
  <groupId>nl.amis.mongodb.countries</groupId>
  <artifactId>MongoDB-Countries-Client</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>MongoDB-Countries-Client</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
        <version>3.4.0</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
```

This dependency definition will retrieve the MongoDB Java Driver that handles the interaction with the MongoDB server.

4. To build the application, execute the following command:

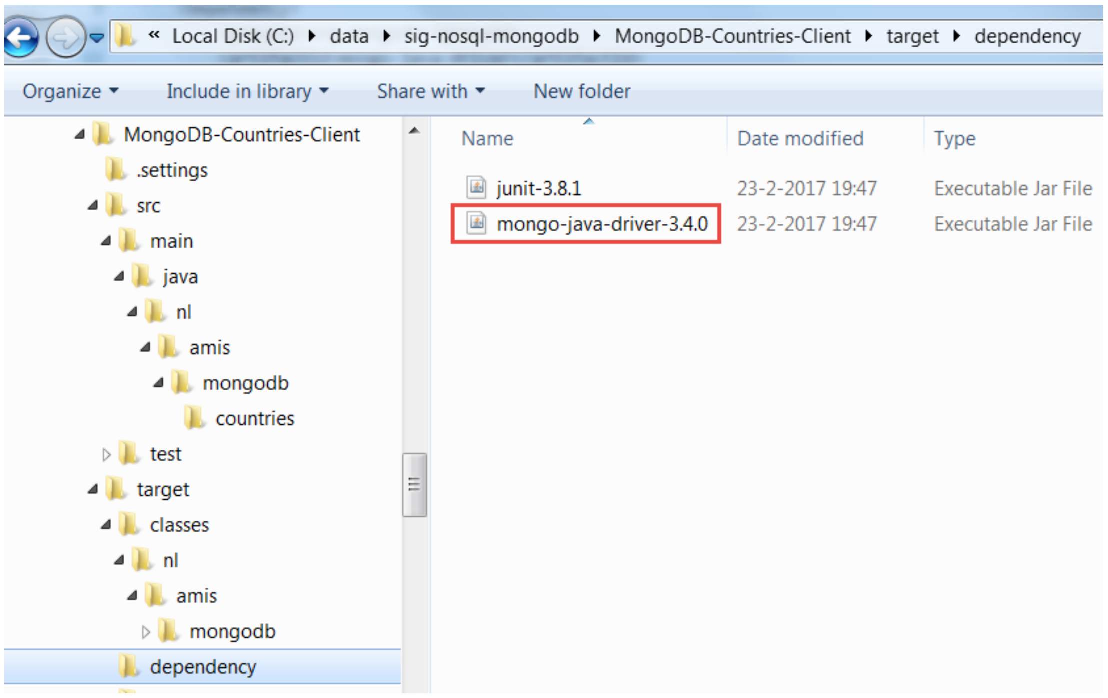
```
mvn package
```

This should produce some output regarding the build process. If the build is successful, the directory *target* will have been created and it should contain a JAR file: MongoDB-Countries-Client-1.0-SNAPSHOT.jar.

5. Before we can run any *main* methods in that JAR, we have to ensure that Java libraries that we depend on at runtime are available in the local environment. Execute this command:

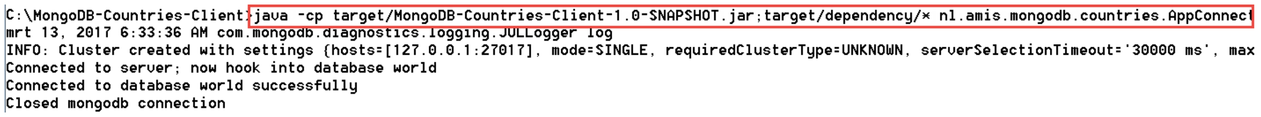
   ```
   mvn install dependency:copy-dependencies
   ```

   This takes care of fetching the JAR files that provide all runtime dependencies. These are downloaded to directory target\dependency.

6. At this point, we are ready to run something. Execute this command (still from directory MongoDB-Countries-Client) to run the main method in class AppConnect – which should establish a connection to database world on your local MongoDB server:

   ```
   java -cp target/MongoDB-Countries-Client-1.0-
   SNAPSHOT.jar;target/dependency/* nl.amis.mongodb.countries.AppConnect
   ```

   ```
   C:\MongoDB-Countries-Client>java -cp target/MongoDB-Countries-Client-1.0-SNAPSHOT.jar;target/dependency/* nl.amis.mongodb.countries.AppConnect
   mrt 13, 2017 6:33:36 AM com.mongodb.diagnostics.logging.JULLogger log
   INFO: Cluster created with settings {hosts=[127.0.0.1:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', max
   Connected to server; now hook into database world
   Connected to database world successfully
   Closed mongodb connection
   ```

   The output should indicate that the connection was successfully created and closed again.

7. Inspect Java Class AppQuery (in directory src\main\java\nl\amis\mongodb\countries). The connection details in this class are the same as in AppConnect; if you had to modify them in step 1, you need to apply that same change here.

   This class connects to the database *world* and subsequently queries data from collection *countries*. The construction of queries – with filter and sort constructions – and also the use of aggregations with projections, match, sort – is demonstrated. Note how data can be retrieved as MongoDBCollection<Document> and how a function can be passed to method forEach on the outcome of a call to find or aggregate to perform an action on each document returned from the query. Also note that the query results are JSON documents – not (mapped) Java objects. There are several libraries – Jongo, Morphia – that can help perform mapping from JSON documents to Java objects.

   Now run AppQuery with the following command:

   ```
   java -cp target/MongoDB-Countries-Client-1.0-
   SNAPSHOT.jar;target/dependency/* nl.amis.mongodb.countries.AppQuery
   ```

   Check the output produced by this call and try to relate every section of output to the contents of class AppQuery.

   Change for example the query that fetches all countries in Asia sorted by name ascending to a query for all countries in Europe, sorted by area descending. After making the change in the code in AppQuery, you need to run *mvn package* again to rebuild the application.

8. Class AppCrud contains code to first connect to the MongoDB server and subsequently create, update and delete data and drop the collection. The connection details in this class are the same as in AppConnect; if you had to modify them in step 1, you need to apply that same change here.

   Verify in the code how the following actions are implemented:
   * create a (nested) document
   * create a list of documents and insert them in a single statement into a MongoDB collection (insertMany)
   * update several attributes (new and existing ones, top level and nested) in a single document using updateOne
   * delete all documents in the collection
   * drop the collection

   Feel free to make changes in the code – to have your own data created for example. If you do make changes, make sure that you run *mvn package* again to rebuild the code.

Run AppCrud using the following command:

```
java -cp target/MongoDB-Countries-Client-1.0-
SNAPSHOT.jar;target/dependency/* nl.amis.mongodb.countries.AppCrud
```

Check the output produced by this call and try to relate every section of output to the contents of class AppCrud.