UV Distributed Information Management
Summer semester 2022

Assignment 02

Summary:

Deadline: May 18, 2022, 11:55 pm (aka 23:55) CET.

Extended Deadline: May 25, 2022, 11:55 pm (aka 23:55) CET.

Submission: Submit a compressed archive (e.g., a `zip` or a `tar.gz` file) that contains your Python3 code and the answers to the questionnaire via Blackboard.

Grading: 55% Python3 code, 45% answers (incl. meeting; cf. Section 5 for details).

---

1 General Remarks

The purpose of this assignment is to get in touch with a widely used document-based NoSQL database system named *MongoDB* [1], which is used in many modern applications. Like PostgreSQL, MongoDB is open-source, rather easy to install and use, and also accessible using the Python programming language (i.e., with the `pymongo` [2] module). In contrast to PostgreSQL, MongoDB is based on the document-based data model [3].

Please submit your final Python code and the answers to the questionnaire until May 18, 2022, 11:55 pm (aka 23:55) CET via Blackboard [4] (late submission until May 25, 2022, 11:55 pm (aka 23:55) CET). Furthermore, please keep in mind that the exams contribute 46% to your final grade, hence you need to submit at least one assignment (partially) to pass the course.

1.1 Formatting Conventions

Commands (for the Linux and the `mongo` command-line tool) and Python3 code are written in `TrueType` font. In addition, `mongo` terminal commands are in a box that specifies the used `mongo` command-line tool at the beginning of the title (separated by a dash – and with a `(.exe)` suffix to denote the Windows version) and the database we are currently working on separated with the > symbol). The following listing shows an example that executes the command `show dbs` on a database named `test` in MongoDB's command-line tool:

Listing 1: `mongo(.exe)` – Show all databases.

```
1    test> show dbs
```

[1] The name refers to the *humongous* amounts of data MongoDB is able to manage.
[2] `https://pymongo.readthedocs.io/en/stable/` and `https://pypi.org/project/pymongo/`
[3] `https://en.wikipedia.org/wiki/Relational_model`
[4] `https://elearn.sbg.ac.at`

Linux and Windows terminal commands are formatted in a similar manner, i.e., framed with the command-line tool at the beginning of the title (`terminal` and `cmd.exe` for Linux and Windows, respectively; again separated by a dash –).

Listing 2: `cmd.exe` – Show directories.

```
dir
```

Listing 3: `terminal` – Show directories.

```
ls -lah
```

Python3 code is simply wrapped in a box without specifying any command-line tool.

For the remainder of this assignment description, we assume that the commands/queries are executed using one of the following command-line tools:

- A system command-line tool (Windows: `cmd.exe`, Linux/MacOS: `terminal`).
- The MongoDB command-line tool (Windows: `mongo.exe`, Linux/MacOS: `mongo`).
  Caveat: Interested students may also want to check out `mongosh(.exe)`, which supports syntax highlighting.
- The MongoDB import command-line tool (Windows: `mongoimport.exe`, Linux/MacOS: `mongoimport`).

1.2 Support

If you have troubles understanding the assignment, please use one of the following communication channels to get help (in this order):

1. **Lecture:** mondays 10:15 - 12:00 am CET, wednesdays 12:00 - 14:00 pm CET (exception: lecture-free periods).
2. **Slack:** `https://dbteaching.slack.com/archives/C01QQHPLFB2`.
3. **Email:** dkocher@cs.sbg.ac.at (as a last resort).

We recommend to start the assignment early. In case of a problem, it is easier for the instructor and other students to provide help in time if you identify the problem early.

Remark: Please notify the instructor as soon as possible if the assignment description is (partially) unclear or if there is a problem with the submission.

2 Assignment Description

We *highly recommend* that you read this section (including all the subsections) to the end before you start working (this should be less error-prone).

Similar to assignment 1, we split this assignment into five parts:

1. Setting up MongoDB, cf. Section 2.2.
2. Create collections and fill them with data, cf. Section 2.3.
3. Familiarize yourself with MongoDB, cf. Section 2.4.
4. Write an example application in Python that uses the database, cf. Section 2.5.

5. Answer the questionnaire, cf. Section 2.6.

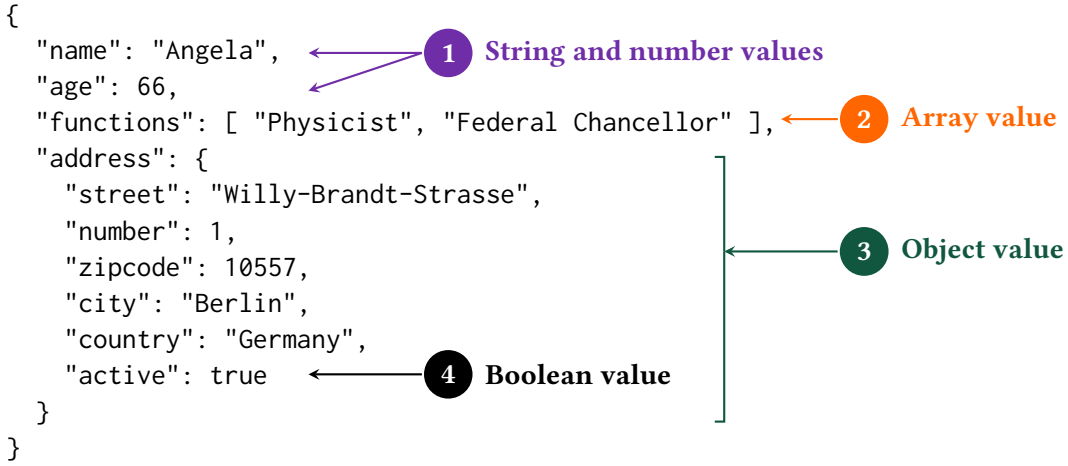
Only parts 1 (depending on your choice), 4, and 5 contribute to the overall grade of this assignment. We recommend to follow these steps and refer to the corresponding sections for details. In addition, we briefly introduce MongoDB and the JSON format in Section 2.1.

2.1 Introduction to MongoDB and JSON

MongoDB is a NoSQL database system that complies to the document-based data model. Since we did not cover many details in the lecture with regard to this data model, we will briefly introduce MongoDB and the underlying data model in more detail.

From the lecture, we know that the document-based data model stores data in a semi-structured and nested format, so-called *documents*. As a document format, MongoDB uses the JavaScript Object Notation (JSON) [5], which is a human-readable text format for data structures that consists of (possibly nested) key-value pairs. A JSON document contains a JSON object (enclosed by curly braces, "{ ... }"). Keys and values in JSON are separated by a colon (":"), and multiple key-value pairs are separated by a comma (","). The curly braces are also used to structure the JSON object. In fact, any data enclosed by curly braces is a JSON object itself. Keys are strings (i.e., enclosed by double quotes "...") but values can be numbers (cf. 1), strings (cf. 1), booleans (`true` or `false`; cf. 4), arrays (an ordered, comma-separated list of values enclosed by squared brackets, "[...]"; cf. 2), or JSON objects themselves (again enclosed by curly braces; cf. 3). The fact that a value can be a JSON object itself allows to nest JSON objects arbitrarily (i.e., introduce a notion of hierarchy in a JSON document, cf. 3).

JSON Example:



MongoDB does not maintain tables but so-called *collections*. A collection stores multiple documents in JSON format and can be regarded as the equivalent of a table in relational database systems. Although MongoDB uses a binary JSON format (called BSON) as internal representation, it is sufficient for this assignment to understand the concept of the JSON format. For interested students, a link to a detailed description of MongoDB's document format and the grouping as collection is provided in Section 4.

[5] `https://www.json.org/json-en.html`

2.2 MongoDB Setup

Similar to assignment 1, you have two possible ways to set up MongoDB such that the assignment can be solved properly:

MongoDB Installation Install MongoDB on your own system (or virtual machine) using an operating system of your choice. This implies that you "pollute" your system with this installation and that the instructor may only be partially helpful (we will still try to help, but we mostly work on Linux systems). Nonetheless, we would like to emphasize that it is an *excellent exercise* for students to perform the installation of such a system on their own (at least once) – cf. Sections 2.2.1 and 2.2.3 ff.

Use MongoDB in a VM Use a pre-installed MongoDB in a virtual machine (VM) that runs Debian Linux. This implies that you will not experience the process of installing MongoDB on your own, but the instructor can more helpful since the VM runs Debian Linux. On the one hand, it may be cumbersome if you do not have any experience with Linux systems (and VMs), but on the other hand it may also be a nice opportunity to familiarize yourself and play around with a Linux system (and a VM) – cf Section 2.2.2 and 2.2.3 ff.

From our experience, however, the installation of MongoDB should not cause many troubles. Hence, we encourage everyone to give it a try. All subsequent steps (starting from the configuration, cf. Section 2.2.3) should be identical in the VM and your local setting.

2.2.1 MongoDB Installation

The first step is to install a MongoDB server (Community Edition) locally on your machine. This should be possible on almost all operating systems, but the specific steps may diverge. However, the documentation of MongoDB provides all the details on how to install MongoDB on different operating systems (cf. links given in Section 4). Once MongoDB is installed, you can create a database and import data in the JSON format. Like PostgreSQL, MongoDB also provides a graphical user interface named *Compass*. Nonetheless, we encourage you to use MongoDB's command-line tools to interact with the database system. In particular the `mongo` command-line tool (aka `mongo` *terminal* or *shell*) provides the most basic way to use the database, and the `mongoimport` command-line tool provides an easy way to import data into the database.

Remarks: (i) Please note that MongoDB's command-line tools may have to be downloaded/installed separately, for example, on Windows systems. (ii) Please use the command-line tool of your operating system to call MongoDB's command-line tools. On Windows systems, for example, use the `cmd.exe` (and navigate into the folder that contains MongoDB's command-line tools).

2.2.2 MongoDB in a VM

We also provide you with a so-called *virtual machine image*, which consists of a pre-installed Debian Linux as operating system and a pre-installed MongoDB installation. In this case, you may have to familiarize yourself with the concept of a virtual machine (VM) and how to host the corresponding image. Essentially, a virtual machine [6] is a software that is designed to provide you with a substitute of a real machine, that is, it runs another operating system on top of your regular operating system – it *virtualizes* another system. For our assignment, it suffices to install a software named VirtualBox [7] that acts as a host for virtual machines. After you successfully installed VirtualBox, a so-called *image* can be used to set up the virtual machine that can be

[6] `https://en.wikipedia.org/wiki/Virtual_machine`
[7] `https://www.virtualbox.org/manual/ch02.html`

used for this assignment. The image (as `ova` file) can be downloaded from our Nextcloud [8] and runs Debian Linux [9] as operating system with an installation of MongoDB.

After downloading the `debian-mongodb-vm.ova` file, this file must be imported into VirtualBox. There are many online tutorials [10] on how to accomplish this and it primarily consists of two steps: (1) Choosing the image to import (navigate to the downloaded file) and (2) checking/-modifying the settings of the virtual machine (typically no modifications are necessary, but sometimes, for example, you may have to disable the USB port). Once you click on the *Import* button, VirtualBox should import the image and set up the virtual machine that can be used for this assignment. VirtualBox then shows a new VM on the left-hand side, which can be started/booted with a double-click.

For this VM, there exist two users: (1) `dbtutorial` and (2) `root`. If you are not familiar with the concept of a `root` user (or superuser), please check out the corresponding article [11] on Wikipedia. In essence, the `root` user has all privileges whereas the `dbtutorial` user has not. By default, we will work with the `dbtutorial` user, but we will temporarily switch to the `root` user if we need additional privileges (e.g., to install a software package). The password for both users is the same: `dbpwd1`

Once you logged into the VM and before you start working on the actual assignment, try to familiarize yourself a bit with Debian Linux (unless you already have experience using Linux systems). For example, open a *terminal* (aka *shell* or *command-line tool*) and try some basic commands (some links can be found in the supplementary material, cf. Section 4). In particular, we will use the `mongo` command-line tool (aka *Mongo terminal* or *Mongo shell*), which provides the most basic way to use the database.

2.2.3 MongoDB Configuration

After installation, the first step is to check whether the MongoDB server is running on your system. This is most likely the case if you installed MongoDB as a daemon (or service) and can be checked using the following command (on Linux systems):

Listing 4: `terminal` – Check whether MongoDB daemon is running.

```
ps aux | grep -v grep | grep mongod
```

`mongod` denotes the so-called *daemon* (or service) for MongoDB. A daemon is a process that runs in the background. In the case of MongoDB, it manages the data and the requests [12]. On Windows systems, one way to see whether this daemon is running is to look for the process `mongod` in the Task Manager.

Once the MongoDB daemon is running, we can connect to our local database without credentials as follows:

Listing 5: `cmd.exe` – Connect to the local MongoDB server without credentials.

```
mongo.exe "mongodb://localhost"
```

Listing 6: `terminal` – Connect to the local MongoDB server without credentials.

```
mongo "mongodb://localhost"
```

[8] `https://kitten.cosy.sbg.ac.at/index.php/s/M4rAJFsM7iCN8oq`
[9] `https://en.wikipedia.org/wiki/Debian`
[10] `https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html`
[11] `https://en.wikipedia.org/wiki/Superuser`
[12] `https://docs.mongodb.com/manual/reference/program/mongod/`

If you configured your MongoDB server to use credentials and/or a custom port, the connection string [13] must be adjusted accordingly:

Listing 7: `terminal` – Connect to the local MongoDB server with credentials (user: `dkocher`, password: `mypw`) and a custom port (`4242`).

```
mongo "mongodb://dkocher:mypw@localhost:4242"
```

After the connection has been established, we can start to interact with the database. Similar to PostgreSQL, MongoDB has a default database named "`test`", and we can use the following command to display the database that is currently in use (i.e., the database we operate on; also indicated by the "`test>`" prefix).

Listing 8: `mongo(.exe)` – Show the database we are currently on.

```
test> db
```

This should report that we are currently on the database named "`test`". To see all databases on our MongoDB server, we can use the following command:

Listing 9: `mongo(.exe)` – Show all databases on our server.

```
test> show dbs
```

In contrast to PostgreSQL, in MongoDB we can switch to a new database on the fly, meaning that MongoDB creates the database as soon as we fill the database with data. Therefore, we can use the following commands to switch to a new database named "`assignment2`" (and verify it).

Listing 10: `mongo(.exe)` – Switch database (and verify it).

```
test> use assignment2
assignment2> db
```

As database for this assignment, please use `assignment2`.

2.3 Data Initialization

After validating that our MongoDB daemon is running, we can proceed to fill our database with some data – by now it is empty. We can verify that it contains no data by executing the following command, which shows all collections in our database.

Listing 11: `mongo(.exe)` – Show all collections in our database.

```
assignment2> show collections
```

Therefore, the next step is to populate the database with some data.

In the course of this assignment, we will use data of two publicly available archives for scientific publications in the computer science domain:

- The DBLP Computer Science Bibliography [14], which provides open bibliographic information on computer science publications.

[13] `https://docs.mongodb.com/manual/reference/connection-string/`
[14] `https://dblp.uni-trier.de/`

- A small portion of the arXiv repository [15], which is a document server for pre-prints of publications.

First, we must download the data from our Nextcloud [16]. Unlike in assignment 1, we do not have to create tables (or documents) before we import the data. Instead, the schema of each single document is encoded in the document itself and we simply import the documents directly into the database. This is the first benefit of MongoDB's schema independence. The data will again be provided through two plain files (one JSON document per line), which are then imported such that their documents fill the respective collections (one collection per file).

MongoDB provides a command-line tool called `mongoimport(.exe)`, which can be used to import the data into the `assignment2` database as follows (note that `mongoimport(.exe)` is called from the system command-line tool):

Listing 12: `terminal` – Import the plain JSON files.

```
mongoimport --db assignment2 --collection dblp --file dblp.json
mongoimport --db assignment2 --collection arxiv --file arxiv.json
```

Listing 13: `cmd.exe` – Import the plain JSON files.

```
mongoimport.exe --db assignment2 --collection dblp --file dblp.json
mongoimport.exe --db assignment2 --collection arxiv --file arxiv.json
```

This creates two collections named `dblp` and `arxiv`, and we can verify this by executing the following commands in the `mongo` command-line tool:

Listing 14: `mongo(.exe)` – Show all collections in our database.

```
assignment2> show collections
```

Remarks: (i) The `--file` option of these commands also work with full paths, for example, `--file "C:\Users\dkocher\Desktop\dblp.json"`. (ii) Importing the DBLP data into the corresponding collection will take some time, hence please wait for the commands to finish. (iii) On Windows systems, `mongoimport.exe` should be executed from the system command-line tool and *not* via double click on the `mongoimport.exe` (it will close immediately since the parameters are missing). Therefore, please execute the system command-line tool in the folder in which the `mongoimport.exe` resides (or navigate into it using `cd`).

2.4 Start Using MongoDB

After the data import, try to further familiarize yourself with the `mongo` command-line tool. Unlike PostgreSQL, MongoDB does not support SQL statements but uses the MongoDB query language (MQL). MQL is an intuitive query language that is designed for application developers. MongoDB supports four so-called CRUD operations: **C**reate, **R**ead, **U**pdate, and **D**elete. In the course of this assignment, we will only use read operations, and the MongoDB documentation provides a comparison of various statements in SQL and MQL (cf. Section 4). In the following, we briefly cover some basics of the MongoDB query language and provide four queries, Q1-Q4, that can be executed out of the box by entering them into the `mongo` command-line tool once the data was imported.

First, the collection on which the operation should be executed must be specified. To refer to a specific collection, MongoDB uses a Dot notation. For example, `db.dblp` refers to the DBLP

[15] `https://arxiv.org/`

[16] `https://kitten.cosy.sbg.ac.at/index.php/s/854BWTPdSGPFFKM`

collection in our database. Similarly, we can execute an operation on this collection using the Dot notation by specifying the name of the operation. The following query executes the `find()` operation on the DBLP collection and uses the `pretty()` operation to display the results in a human-readable format (otherwise each JSON document is printed as a single line, which is rather unreadable). Note that MongoDB does not return all documents immediately, but a cursor that can be used to iterate over the documents in the result set of the query.

Listing 15: `mongo(.exe)` – Execute the `find()` operation on the DBLP collection.

```
assignment2> db.dblp.find().pretty()
```

The above query corresponds to the SQL query `SELECT * FROM dblp`, i.e., we request all documents of the DBLP collection (without any constraints). Similar to the `WHERE` clause in SQL, we can define one or more criteria to filter the documents before they are returned. This can be accomplished by passing a JSON object to the `find()` operation. In this JSON object, you are able to specify the criteria that must be met by a document to be in the result set of the query.

Query Q1: `mongo(.exe)` – Find all documents authored by "Michael Stonebraker".

```
assignment2> db.dblp.find({ "author": "Michael Stonebraker"}).pretty()
```

In query Q1, we pass a JSON object (mind the enclosing curly braces) with a single key-value pair, `author: "Michael Stonebraker"`, to the `find()` operation. Consequently, only documents that have been authored by "Michael Stonebraker" are returned by the database.

Query Q2: `mongo(.exe)` – Find all documents authored by "Michael Stonebraker" in books titled "ICDE" (which is one of the top conferences in database research).

```
assignment2> db.dblp.find({
  "author": "Michael Stonebraker",
  "booktitle": "ICDE"
}).pretty()
```

Query Q2 passes a JSON object with two key-value pairs to the `find()` operation. This is similar to the `WHERE` clause of query Q2 of assignment 1: Only documents that satisfy both criteria (i.e., the documents that satisfy both key-value pairs) are returned.

Query Q3: `mongo(.exe)` – Join documents of the arXiv collection and the DBLP collection that have the same title (i.e., find identical publications that appear in both collections).

```
assignment2> db.arxiv.aggregate({
  "$lookup": {
    "from": "dblp",
    "localField": "title",
    "foreignField": "title",
    "as": "arxivdblp"
  }
}).pretty()}
```

MongoDB also supports joins to link documents of two (or more) collections, which is depicted in query Q3. Although it is not as intuitive as the `JOIN` operation in PostgreSQL, it follows the same principle. We specify the first collection using the Dot notation (`db.arxiv`) and then use the `aggregate()` operation to define the second collection (using `from: "dblp"`).

To perform the join, we use `$lookup` [17], which adds a new field to each input document (note that we can omit the double quotes around `$lookup` because this is a special key). The name of the new field (i.e., its key) is specified using the "as" field in our JSON object. The value of the new field contains all documents that satisfy the join criterion (i.e., have the same title). To this end, we specify the fields of the respective collections that are used to link the documents. `localField` defines the field to be used in the arXiv collection and `foreignField` defines the field to be used in the DBLP collection. In other words, each document of the arXiv collection is extended with a new key "`arxivdblp`" that is associated with a list of all DBLP documents that have the same title.

To conclude, the database returns documents that appear in both collections and have the same title.

Query Q4: `mongo(.exe)` – Explain query Q2.

```
assignment2> db.dblp.find({
  "author": "Michael Stonebraker",
  "booktitle": "ICDE"
}).explain()
```

The last query, Q4, is basically Q2 with an additional operation: We put the `explain()` operation [18] before the actual operation we want to execute (in this case, the `find()` operation). Like the `EXPLAIN` keyword in PostgreSQL, this instructs MongoDB to return the *query plan*, that is, information about the steps MongoDB plans to execute in order to determine the result. As an exercise (not part of this assignment), the other queries could also be extended with a trailing `explain()` operation.

2.5 Access the Data Using Python

Remark for students that use the VM: For technical reasons, you may experience an error when you try to install the `python-pip3` package in our virtual Debian machine (which in turn is used to install the `pymongo` module for Python3). Please try executing the following command in case Debian reports an error when you try to install `python-pip3`:

Listing 16: `terminal` – Fix missing dependencies (VM users only).

```
su -
apt-get install --fix-missing
```

Although the `mongo` command-line tool can be used to execute queries, a database system is typically accessed by an application. Therefore, the fourth part of this assignment is to write a small Python. We recommend to use the `pymongo` module (or driver) for Python3 to (a) establish a connection to your local database, (b) execute the queries and retrieve the results, and (c) close the connection to your local database. Your application should execute the queries Q1, Q2, and Q4 as provided in Section 2.4 and print the respective results (i.e., all the documents that are returned by MongoDB; output format does not matter as long as it is human-readable).

Like in assignment 1, you are asked to adapt query Q3: Q3 as given in Section 2.4 returns a list of documents that satisfy the condition in the `$lookup` aggregation operator. In your Python3 application, Q3 should only return the *number* of documents that satisfy the condition that is expressed using the `$lookup` operator. This can either be accomplish by modifying the MQL statement itself to count the number of documents (hint: extend the MQL query with the `$count`

[17] `https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/`
[18] `https://docs.mongodb.com/manual/reference/method/db.collection.explain/`

aggregation operator [19]) or by adapting the Python3 code such that not all the documents are printed but only the number of documents (hint: this can probably be done using the `len()` function [20]).

Remark: If you cannot execute all queries (for whatever reason), please contact the instructor *before* the submission. We will find a reasonable solution together.

Template Code There are many tutorials regarding the installation [21] and the usage of pymongo [22] to access a MongoDB database. Nonetheless, we provide a minimum template code in Python3 that can be used as a starting point.

Listing 1: Template code to access a database using pymongo.

```python
#!/usr/bin/python3

import pymongo as pym
# A module that formats the output in a readable format, which is quite useful
# in case of JSON documents.
import pprint

def main():
  try:
    # Establish a connection to the local MongoDB server without credentials.
    connection = pym.MongoClient("mongodb://localhost")

    # If you have credentials, we refer to the documentation for more details:
    # https://pymongo.readthedocs.io/en/stable/examples/authentication.html
  except:
    print("Unable to connect to {}".format('mongodb://localhost'))

  try:
    db = connection["assignment2"]

    # Create a dict (aka JSON object) that is used in the find() operation
    json_query = {
      "author": "Michael Stonebraker",
      "booktitle": "ICDE"
    }

    # Execute the find() operation and retrieve a cursor to the result set
    cursor = db.dblp.find(json_query)

    # Print the documents in the result set to the command line. We use the
    # pprint module to print the JSON document in a human-readable format (the
    # standard print function prints the JSON documents in a single line). It is
    # up to you whether you want to use pprint or not.
    for i, x in enumerate(cursor):
      pprint.pprint({i: x})
  except Exception as e:
    print("Unable to execute simple find() query: {}".format(e))
  finally: # The finally-branch is executed independently of an exception.
    if cursor is not None:
      # Close the cursor.
      cursor.close()

    if connection is not None:
      # Close the connection.
      connection.close()

if __name__ == "__main__":
  main()
```

[19] https://www.mongodb.com/docs/manual/reference/operator/aggregation/count/
[20] https://docs.python.org/3/library/functions.html#len
[21] https://pymongo.readthedocs.io/en/stable/installation.html and https://pypi.org/project/pymongo/
[22] https://pymongo.readthedocs.io/en/stable/tutorial.html

2.6 Questionnaire

The questionnaire contains questions about the assignment. These questions are potentially discussed during the after-assignment meetings. The questionnaire can be found in a separate textfile called `assignment2-questionnaire.txt`.

3 Submission

Please submit a single compressed archive (e.g., `.zip` or `.tar.gz`) that contains two files: (a) The code of your Python3 application and (b) the answers to the questionnaire.

Code Please submit a single Python3 file (`.py`) that contains the full code for this assignment, i.e., the connection to the database and the execution of the four queries. The code must print the results of *all* four queries (one after another) when executed *as submitted*. We will not change your code, for example, change some variable to see the result of a certain query. Therefore, please double-check that *all* four queries are executed (and also that query Q3 is modified as described in Section 2.5).

Remark: Please consider removing the database credentials (i.e., username and password) before you submit your code (in case you did not use the default ones as described in Section 2.2).

Questionnaire You can answer the questions directly in the textfile `assignment2-questionnaire.txt`. If you prefer to use a different application to answer the questions (e.g., Microsoft Word and the likes), you are welcome to do so. In any case, the submitted file must be in one of the following formats: `.txt`, `.pdf`, `.odt`, `.doc`, or `.docx`. **Remark:** The recommended formats are `.txt` and `.pdf`.

4 Supplementary Material

This section provides a list of pointers to material that may be helpful to solve the assignment.

- MongoDB: `https://www.mongodb.com/`
- The full MongoDB documentation: `https://docs.mongodb.com/manual/`
- MongoDB introduction: `https://docs.mongodb.com/manual/introduction/`
- MongoDB "Getting Started": `https://docs.mongodb.com/manual/tutorial/getting-started/`
- MongoDB resources regarding the installation of MongoDB on
  - Linux systems: `https://docs.mongodb.com/manual/administration/install-on-linux/`
  - MacOS systems: `https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/`
  - Windows systems: `https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/`
- Other resources regarding installation and usage of MongoDB:
  - Two of the many MongoDB "Getting Started" guides:
    1. Windows: `https://www.youtube.com/watch?v=RsWdj7V2Ojg`
    2. Linux/MacOS: `https://www.youtube.com/watch?v=bKjH8WhSu_E`
  - SQL to MongoDB Mapping Chart: `https://docs.mongodb.com/manual/reference/sql-comparison/`

- MongoDB documents: `https://docs.mongodb.com/manual/core/document/`
- MongoDB collections: `https://docs.mongodb.com/manual/core/databases-and-collections/`
- Reference for the `mongo` command-line tool: `https://docs.mongodb.com/manual/reference/mongo-shell/`
- Python Modules: `https://docs.python.org/3/installing/index.html`
- The `pymongo` module: `https://pymongo.readthedocs.io/en/stable/`
- Installation of the `pymongo` module for Python:
  - Official website: `https://pymongo.readthedocs.io/en/stable/installation.html`
  - The Python Package Index: `https://pypi.org/project/pymongo/`
- The `pymongo` tutorial: `https://pymongo.readthedocs.io/en/stable/tutorial.html`
- One of the many `pymongo` "Getting Started" guides: `https://www.youtube.com/watch?v=rE_bJl2GAY8`
- One of the many introductions to the Linux terminal: `https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal`

5 Grading

For the sake of transparency, this section provides more details on the grading of this assignment, i.e., which part contributes how many points to the total number of 18 points for this assignment.

Code The code contributes at most 10 points and is evaluated based on the following criteria (if the code is executed as submitted; disregarding the credentials):

Max. Points	Criterion
2	Q1 is executed and the correct result is printed to the command line.
2	Q2 is executed and the correct result is printed to the command line.
2	Modified Q3 is executed and the correct result is printed to the command line.
2	Q4 is executed and the correct result is printed to the command line.
2	Answer 2 questions regarding your submission in the after-assignment meeting.
10	

Questionnaire The questionnaire contributes at most 8 points and is evaluated based on the following criteria (taking the discussion in the after-assignment into account):

Max. Points	Criterion
2	Correctness of answer A1.
2	Correctness of answer A2.
2	Correctness of answer A3.
2	Correctness of answer A4.
8	