Assignment 02

UV Distributed Information Management

Daniel Kocher, Summer semester 2026

Summary

- **Deadline:** June 07, 2026, 11:59 p.m. (aka 23:59) CET.
- **Submission:** Submit a compressed archive (`zip` or `tar.gz`) that contains your Python3 code and the answers to the questionnaire via Blackboard.
- **Grading:** 25% Python3 code, 25% answers, and 50% from the after-assignment meeting (cf. Section 5 for details).

1 Introduction

This assignment provides an introduction to a widely used document-based database system (DBS), namely **MongoDB** [1]. It is a NoSQL DBS, source-available [2], rather easy to install and use, and accessible using the Python3 programming language [3] (e.g., using the `pymongo` [4] module). MongoDB relies on the document-based data model [5].

1.1 Grading and Submission

Section 5 specifies the grading scheme for this assignment in detail. Please submit your final Python3 code **and** your answers to the questionnaire until June 07, 2026, 11:59 p.m. (aka 23:59) CET via Blackboard [6]. Recall that the assignments contribute 60% to your final grade, i.e., you need to participate in at least one assignment to pass this course.

1.2 Formatting Conventions

Commands for the Linux command-line tool (`terminal`) [7] and the MongoDB command-line tool (`mongosh`) are written in `TrueType` font [8]. In addition, all commands are in a box

[1] Referring to **humongous** amounts of data; MongoDB: `https://de.wikipedia.org/wiki/MongoDB`
[2] Source-available software: `https://en.wikipedia.org/wiki/Source-available_software`
[3] Python programming language: `https://en.wikipedia.org/wiki/Python_(programming_language)`
[4] `https://pymongo.readthedocs.io/en/stable/` and `https://pypi.org/project/pymongo/`
[5] Document-oriented database: `https://en.wikipedia.org/wiki/Document-oriented_database`
[6] Blackboard: `https://elearn.sbg.ac.at`
[7] Command-line tool: `https://en.wikipedia.org/wiki/Command-line_interface`
[8] `TrueType` font: `https://en.wikipedia.org/wiki/TrueType`

that specifies the command-line tool at the beginning of the title (separated by a dash –, e.g., `mongosh` for MongoDB). Listing 1 shows an example for the Linux `terminal`.

Listing 1: `terminal` – Show directories.

```
dbtutorial@database-tutorial:~# ls -l
total 32
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Mar 31 15:43 Desktop
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Documents
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Downloads
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Music
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Pictures
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Public
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Templates
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Videos
```

The Linux `terminal` shows a prefix `dbtutorial@database-tutorial:~#` that contains:

- The name of the **user** that executes the command. `dbtutorial` is the default user of the virtual machine (VM) (this may be different if you do not use the given VM).
- The name of the **machine**, where `database-tutorial` is the name of the given VM.
- A **delimiter** that separates the actual command from the "`user@machine`" string. "`:~#`" is the default delimiter of the given VM (~ denotes the `home` directory).

Moreover, we color-code commands and their output for readability:

- A **command** itself (that may be copied) is depicted in solid `black`.
- **Prefixes**, which are **not** part of a command itself, are shown in `gray`.
- **Output**, i.e., the result of a command, is shown in `green`.

Python3 [9] code is wrapped in a box without specifying any command-line tool, and we provide a `.py` file that contains the Python3 code.

Listing 2 exemplifies a command in MongoDB's command-line tool with output.

Listing 2: `mongosh` – Show all collections.

```
test> show collections
```

For commands that must be executed in the `mongosh` terminal, the prefix `test>` denotes the database we are currently connected to (cf. Section 2.3 for more details). The MongoDB query language (MQL) is different from SQL and keywords like `show` and `db` are typically **not** capitalized. Furthermore, MongoDB has a dedicated command-line tool called `mongoimport` to import data into the database. Additional information can be found in the supplementary material given in Section 4.

Remark: Note that on our server infrastructure, the command-line tool is called `mongo` (i.e., no `sh` suffix). Hence, please use `mongo` instead of `mongosh` to connect to the database.

1.3 Support

If anything is unclear or if there is a problem with the submission, please use one of the following communication channels to get help (in this order):

[9] Python Programming language: `https://en.wikipedia.org/wiki/Python_(programming_language)`

1. **Lecture:** Wednesday 11:30 a.m. - 01:30 p.m. CET (exception: lecture-free periods).
2. **Slack:** `https://dbteaching.slack.com/archives/C08GRUMJ97H` (I will check regularly and do my best to reply fast, but please do not expect me to be available 24/7).
3. **Email:** dkocher@cs.sbg.ac.at (as a last resort).

We recommend to start the assignment early. It is easier for the instructor (and other students) to provide help in time if you identify problems early.

---

2 Assignment Description

We **highly recommend to read the entire section** (incl. all subsections) before you start working (this should be less error-prone). Similar to Assignment 1, we split this assignment into five parts and 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 more details.

1. Set up and configure MongoDB; cf. Section 2.2 and Section 2.3.
2. Create collections and fill them with data, cf. Section 2.4.
3. Familiarize yourself with MongoDB, cf. Section 2.5.
4. Write an example application that accesses the database in Python3; cf. Section 2.6.
5. Answer the questionnaire; cf. Section 2.7.

2.1 Introduction to MongoDB and JSON

MongoDB is a NoSQL [10] 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.

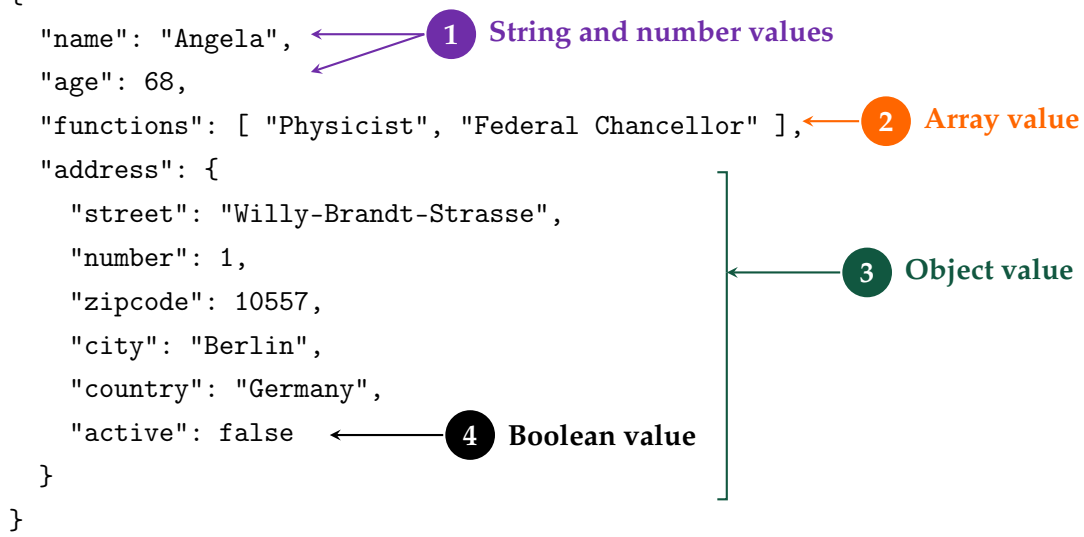
From the lecture, we know that the document-based data model stores data in a semi-structured [11] and nested format, so-called **documents**. As a document format, MongoDB uses the JavaScript Object Notation (JSON) [12], which is a human-readable text format for data structures that consists of (possibly 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 used to structure the JSON object, and any data enclosed by curly braces is a JSON object **itself**. Keys are strings [13] (i.e., enclosed by double quotes: "...") but values can be numbers (1), strings (1), booleans [14] (`true` or

[10] NoSQL: `https://en.wikipedia.org/wiki/NoSQL`
[11] Semi-structured data: `https://en.wikipedia.org/wiki/Semi-structured_model`
[12] JSON format: `https://en.wikipedia.org/wiki/JSON` and `https://www.json.org/json-en.html`
[13] The string data type: `https://en.wikipedia.org/wiki/String_(computer_science)`
[14] The boolean data type: `https://en.wikipedia.org/wiki/Boolean_datatype`

`false`; 4), arrays [15] (an ordered, comma-separated list of values enclosed by squared brackets, `[ 0, 1, 2, ..., 9 ]`; 2), or JSON objects [16] themselves (again enclosed by curly braces; 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; 3).

In contrast to the relational model, MongoDB does not maintain tables but **collections**. A collection stores multiple **documents** in the JSON format and is similar to a table in relational database systems. Although MongoDB uses a binary JSON format called BSON [17] 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.

JSON Example:



2.2 Options to Solve this Assignment

This section outlines four options to solve this assignment.

1. **Use our Server Infrastructure:** The Database Research Group runs a MongoDB server with a web interface that can be used to solve this assignment. On the one hand, you can focus on SQL and do not have to deal with technical details; on the other hand, you will not experience the MongoDB installation yourself. To continue with this option, you can jump to 🔗 Section 2.2.1.
2. **Use a Virtual Machine (VM):** Use a VM with Debian Linux and MongoDB pre-installed as you know it from Assignment 0. Again, you will not experience the MongoDB installation yourself, but it may be less cumbersome than setting up MongoDB. To continue with this option, you can jump to 🔗 Section 2.2.2.

[15] The array data type: `https://en.wikipedia.org/wiki/Array_data_structure`
[16] The object data type: `https://en.wikipedia.org/wiki/Object_(computer_science)`
[17] The Binary JSON format: `https://en.wikipedia.org/wiki/BSON`

3. **Install MongoDB:** Install MongoDB on your system. This implies that you may "pollute" your system with this installation and that the instructor may need additional information on your particular setup to provide help (since we mostly work on Linux systems). Nonetheless, we want to emphasize that it is an **excellent exercise** to install such a system yourself (at least once). To continue with this option, you can jump to 🔗 Section 2.2.3.
4. **Use Docker (experimental; bonus point opportunity):** Use a Docker image with MongoDB installed. For new chipsets (e.g., Apple Silicon [18]) with different architectural characteristics (compared to typical Intel and AMD chipsets), running VMs in VirtualBox is still problematic and buggy. Docker mimics a lightweight VM without a graphical user interface (GUI), i.e., this is the most advanced option. As this option is still experimental, you have the opportunity to receive **up to 1 bonus point** if you help to improve the Docker setup by giving (constructive) feedback on it. To continue with this option, you can jump to 🔗 Section 2.2.4.

You can choose your preferred option, but **be prepared for questions** on your choice.

2.2.1 Use our Server Infrastructure

We use the server infrastructure of the Database Research Group and you receive **personalized credentials** (i.e., do not expose them to others) to use it in your preferred browser (e.g., Firefox). Once you received your credentials, the **workflow** is as follows:

1. **Connect** to our interface: `https://terminal01.dbresearch.plus.ac.at/`
2. **Log in** with your personalized credentials and set up two-factor authentication (as you probably already know it from your student email).
3. You are redirected to a **web-based terminal interface** asking for your credentials once again, i.e., enter them.
4. You are logged into a **virtual machine (VM)** named `studentdbclient01` that you can use to work on this assignment.

Remark: If you run into problems, please contact the instructor as soon as possible and try to describe the problem in detail. We try to resolve it fast.

This option does **not** require `root` privileges. Consequently, there is **no possibility** to gain them (e.g., using `su`) and you can safely skip most steps of Section 2.3 and can proceed with Section 2.4, but take the following **important differences** into account:

– For security reasons, your VM is isolated and we cannot use `curl` or `wget` to download the data. However, the file `assignment2-data.zip` is already stored on your VM and you just need to `unzip` it as shown in Listing 3. Afterwards, you can proceed as described in Section 2.4.

[18] Apple Silicon chipset: `https://en.wikipedia.org/wiki/Apple_silicon`

– You do not have to create a new database. You can connect to an existing database as shown in Listing 4. Please replace `username` with your username. Afterwards, you are connected to your personalized database.

Overall, Listings 3-6 show the process to follow before jumping to Section 2.5.

Remark: To use the copy & paste functionality in the web interface, please use the shortcut `CRTL+ALT+SHIFT` to show/hide additional options.

Listing 3: `terminal` – Unzip the data for this assignment.

```
username@studentdbclient01:~$ unzip assignment2-data.zip
```

Listing 4: `terminal` – Connect to your personalized database on our server infrastructure and check for existing collections (replace `username` with your actual username).

```
username@studentdbclient01:~$ mongo --host studentdbserver01.dbresearch.plus.ac.at --port 27017 --authenticationMechanism PLAIN --authenticationDatabase '$external' -u username -p
ss2026-username> show collections
```

Listing 5: `terminal` – Import the plain JSON files to your personalized database on our server infrastructure (replace `username` with your actual username; `user@machine` string shortened).

```
username@..:~$ mongoimport --db ss2026-username --collection dblp --file dblp.json --host studentdbserver01.dbresearch.plus.ac.at --port 27017 --authenticationMechanism PLAIN --authenticationDatabase '$external' -u username
...
...1984049 document(s) imported successfully.  0 document(s) failed to import
username@..:~$ mongoimport --db ss2026-username --collection arxiv --file arxiv.json --host studentdbserver01.dbresearch.plus.ac.at --port 27017 --authenticationMechanism PLAIN --authenticationDatabase '$external' -u username
...
...3 document(s) imported successfully.  0 document(s) failed to import
```

Listing 6: `terminal` – Connect to your personalized database on our server infrastructure and show all collections (replace `username` with your actual username).

```
username@studentdbclient01:~$ mongo --host studentdbserver01.dbresearch.plus.ac.at --port 27017 --authenticationMechanism PLAIN --authenticationDatabase '$external' -u username -p
ss2026-username> show collections
arxiv
dblp
```

2.2.2 Use a Virtual Machine (VM)

Remark: In case you owe an Apple Silicon machine [19], you find online resources [20] on how to convert the `OVA` format into a format that is usable with, e.g., UTM [21].

We use the VM image of Assignment 0, and you may have to familiarize yourself with the concept of a VM and how to host the corresponding image (cf. Assignment

[19] Apple Silicon chipset: `https://en.wikipedia.org/wiki/Apple_silicon`
[20] OVA to UTM: `https://medium.com/@d_mechraoui/how-to-run-ova-virtual-machine-on-apple-silicon-using-utm-m1-m2-m3-7a2a883697d4`
[21] UTM – Virtual machines for MAC: `https://mac.getutm.app/`

0 for details). In 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 [22] on Wikipedia. Essentially, the `root` user has **all privileges**, whereas the user `dbtutorial` has only limited privileges. By default, we will work with the `dbtutorial` user, but we will temporarily switch to `root` if we need additional privileges. The password for both users is the same: `dbpwd1`

Once you logged into the VM, familiarize yourself with Debian Linux (unless you already have experience using Linux systems and/or solved Assignment 0). For example, open the **command-line tool** of Linux, i.e., the `terminal`, and try basic commands (cf. Assignment 0 and Section 4). In particular, we will use MongoDB's **command-line tool**, i.e., the `mongosh` (or `mongo`) terminal, which provides the most basic way to use MongoDB.

If you are using the VM, you may now directly jump to Section 2.3.

Remark: Note that you may not be able to use `CTRL+C` and `CTRL+V` to copy and paste between your regular system and the VM as the clipboard is not shared by default [23].

2.2.3 Install MongoDB

For this option, the first step is to install a MongoDB server (Community Edition) on your machine. This should be possible on any operating system, but the specific steps may differ. Sources on how to install MongoDB can be found in the supplementary material provided in Section 4. Like PostgreSQL, MongoDB also provides a graphical user interface named **Compass** [24]. Nonetheless, we recommend to use MongoDB's command-line tools to interact with MongoDB. In particular, the `mongosh` command-line tool provides the most basic way to use the database. Furthermore, you will need to use the `mongoimport` command-line tool to import JSON files into the database. Once MongoDB is installed, you can create a database and import data in the JSON format.

If you installed MongoDB natively on your system, you may now directly jump to Section 2.3.

2.2.4 Use Docker — Experimental + Opportunity for Bonus Point

Remark: You have the opportunity to receive **up to 1 bonus point** if you help to improve the Docker setup by providing (constructive) feedback.

First, install Docker Desktop [25] for your system. Installations exist for Linux, Windows, MacOS on Intel chipsets, and MacOS on Apple Silicon chipsets. The documentation provides information on how to install and start Docker for your particular system.

Docker [26] is a software package that also virtualizes at the level of an operating system (e.g., you can run Debian Linux within Windows or MacOS) but follows a different

[22] The `root` user: `https://en.wikipedia.org/wiki/Superuser`
[23] Shared clipboards in VirtualBox: `https://www.youtube.com/watch?v=fqrJ7qlhJu0`
[24] MongoDB Compass: `https://www.mongodb.com/products/compass`
[25] Getting Started with Docker: `https://www.docker.com/get-started/`
[26] The Docker software package: `https://en.wikipedia.org/wiki/Docker_(software)`

philosophy. Traditional VMs are **stateful**, i.e., the state of your VM can be stored and resumed later on. Contrarily, Docker is designed to be **stateless**, i.e., it is not as easy to store the state and resume from it. Hence, follow the instructions carefully not to lose **effects of your commands and/or data** when shutting down Docker (or the terminal it is running in). Furthermore, Docker operates at the (finer) granularity of **containers**, where a container is a lightweight building block with a single responsibility, e.g., a container can run a database. Then, multiple of these containers can be used to implement a broader functionality (containers are "stacked" like in a dock). Contrastingly, traditional VMs are full-fledged VMs that may run different services at once.

For this assignment, we use Docker as follows: We build a container that runs a MongoDB server and connect to it to work on the assignment. To this end, download the **Dockerfile** and the file **docker-compose.yml** from our Nextcloud [27], open a `terminal`, and navigate into the directory where both files are stored. Then, build the Docker image tagged (i.e., named) `mongodb-tutorial` as shown in Listing 7.

Listing 7: `terminal` – Build the Docker image (mind the trailing dot) and create required directories.

```
1  user@machine:~$ docker compose up -d -build
2  user@machine:~$ mkdir -p mongodb-data
```

Line 1 builds the Docker image using the file `docker-compose.yml` and `docker compose`. This command also runs the container after building it. Line 2 creates a new subdirectory that stores the MongoDB database persistently on your host system, i.e., it is **not** lost if you shut down the container, and you can reconnect to continue working.

To connect to the running container, we use the command shown in Listing 8.

Listing 8: `terminal` – Connect to the Docker container.

```
1  user@machine:~$ docker exec -it --user dbtutorial mongodb-tutorial bash
2  dbtutorial@6fb3eb1459c7:~$
```

We observe that the `user@machine` prefix changes to `dbtutorial@6fb3eb1459c7`. That is, the user `dbtutorial` is now connected to the container with the **unique hash** `6fb3eb1459c7` (this hash is most probably different on your system), and we can continue with the command shown in Listing 12 to connect to our database `assignment2`. From this point on, we can follow the instructions provided in Section 2.4, but keep in mind that we can use the shared directory between host system and Docker container. For example, you could write the Python3 code on your host system and execute it in the container. For completeness, we describe how to remove the Docker container (cf. Listing 9) **once you solved this assignment.**

Listing 9: `terminal` – Remove the Docker container.

```
1  user@machine:~$ docker compose down -v
```

[27] Dockerfile: `https://kitten.cosy.sbg.ac.at/index.php/s/obNicKdDEwNbXNe`

If you are using the Docker setup, you can now directly continue with Listing 12 (cf. Section 2.3).

2.3 MongoDB Configuration

Remark: If you are using our server infrastructure, you can safely skip this section.

After the installation, the first step is to check whether the MongoDB daemon (or service) is running. A **daemon** is a process that runs in the background. In the case of MongoDB, it manages the data and the requests [28]. Most likely, the MongoDB daemon named `mongod` is **not** running. Hence, we (re-)start it as shown in Listing 10.

In Listing 10, we first confirm that the MongoDB daemon is **not** running (line 1) using the `systemctl` command. This is a passive command that can be executed with the `dbtutorial` user. We observe that the state of our MongoDB daemon is `inactive (dead)`, i.e., it exists but is not running. Next, we need to execute an active command that **modifies** the state of our operating system by starting the MongoDB daemon. To this end, we must first switch to the `root` user (line 6) and use the `systemctl` command to (re-)start the MongoDB daemon (line 8). Do not be confused that nothing is shown when you type in your password (not even asterisks *****). Line 9 switches back to the `dbtutorial` user (as we do not want to stay in `root` mode). Finally, we confirm that the MongoDB daemon is running (line 10).

In this case, the `systemctl` command shows a lot of status information about the MongoDB daemon, e.g., the starting time, a process ID (PID), used memory, and the likes. This information may be different on your system, but the MongoDB daemon should be `active (running)`. Once the MongoDB daemon is running, we can connect to our local database without credentials using the MongoDB shell as shown in Listing 11.

Listing 10: `terminal` – (Re-)Start the MongoDB daemon.

```
1  dbtutorial@database-tutorial:~$ systemctl status mongod
2  ● mongod.service - MongoDB Database Server
3       Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor ...)
4       Active: inactive (dead)
5         Docs: https://docs.mongodb.org/manual
6  dbtutorial@database-tutorial:~$ su -
7  Password:
8  root@database-tutorial:~# systemctl restart mongod
9  root@database-tutorial:~# exit
10 dbtutorial@database-tutorial:~$ systemctl status mongod
11 ● mongod.service - MongoDB Database Server
12      Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor ...)
13      Active: active (running) since Tue 2023-05-02 13:12:12 CEST; 5s ago
14        Docs: https://docs.mongodb.org/manual
15    Main PID: 57756 (mongod)
16      Memory: 192.5M
17         CPU: 1.015s
18      CGroup: /system.slice/mongod.service
19              ÀÄ57756 /usr/bin/mongod -config /etc/mongod.conf
```

Listing 11: `terminal` – Connect to the local MongoDB server without credentials.

```
1  dbtutorial@database-tutorial:~$ mongosh "mongodb://localhost"
```

[28] The MongoDB daemon: `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 [29] must be adjusted accordingly (cf. Listing 12).

Listing 12: `terminal` – Directly connect to a specific database (`assignment2`) on a local MongoDB server with credentials (user: `dbtutorial`, password: `dbpwd1`) and a custom port (`27017`).

```
dbtutorial@database-tutorial:~$ mongosh "mongodb://dbtutorial:dbpwd1@localhost:27017/assignment2"
```

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 Listing 13 shows how to display the current database (indicated by the `test>` prefix).

Remark: The `test>` at the beginning is not part of the command.

Listing 13: `mongosh` – Show the database we are currently on.

```
test> db
test
```

This should report that we are currently on the `test` database. Listing 14 shows how to list all existing databases (the size may be slightly different on your system).

Listing 14: `mongosh` – Show all databases on our server.

```
test> show dbs
admin 8.00 KiB
config 12.00 KiB
local 8.00 KiB
```

In MongoDB, we can switch to a new database on the fly, i.e., without creating it explicitly. Listing 15 shows how to switch to a new database named `assignment2`.

Listing 15: `mongosh` – Switch database (and verify it).

```
test> use assignment2
switched to db assignment2
assignment2> db
assignment2
```

2.4 Data Initialization

Initially, the database contains no data (i.e., is empty). The `show collections` command returns no results, i.e., no collections have been found (cf. Listing 16).

Listing 16: `mongosh` – Show all collections in our database.

```
assignment2> show collections
```

Therefore, we have to populate the database with some data. We will use data of two publicly available archives for scientific publications in the computer science domain:

[29] MongoDB's connection string: `https://docs.mongodb.com/manual/reference/connection-string/`

– The DBLP Computer Science Bibliography [30] provides open bibliographic information on computer science publications and contains 1,984,049 JSON documents.
– A small portion of the arXiv repository [31], which is a document server for pre-prints of publications. This dataset contains 3 JSON documents.

Download the data from our Nextcloud [32] and unzip it. The following two JSON files will be extracted (each of which contains one JSON object per line):

– `dblp.json` represents the DBLP bibliography.
– `arxiv.json` represents the portion of the arXiv repository.

The data for this assignment can either be downloaded using a browser (e.g., Firefox) or using tools like `curl` or `wget` (cf. Listing 17). `curl` or `wget` are required if you use Docker.

Listing 17: `terminal` – Data download using `curl` or `wget` (`user@machine` string shortened).

```
~$ curl https://kitten.cosy.sbg.ac.at/index.php/s/sq6SNrpAyjQoNtt/download --output assignment2-data.zip
~$ wget https://kitten.cosy.sbg.ac.at/index.php/s/sq6SNrpAyjQoNtt/download -O assignment2-data.zip
```

The structure of each document is directly encoded in a document, and we simply import the documents into the database. Due to this schema independence, we do not have to create schemas before importing the data. Each JSON file results in one collection that contains all JSON documents of the corresponding JSON file. MongoDB provides a command-line tool called `mongoimport` to import the data into the `assignment2` database as depicted in Listing 18 (note that `mongoimport` is called from the Linux terminal).

Remark: Open another terminal to execute the `mongoimport` commands without disconnecting from your MongoDB server.

Listing 18: `terminal` – Import the plain JSON files (`user@machine` string shortened).

```
dbtut..@:~$ mongoimport -u dbtutorial -p dbpwd1 --db assignment2 --collection dblp --file dblp.json
...
...1984049 document(s) imported successfully.  0 document(s) failed to import
dbtut..@:~$ mongoimport -u dbtutorial -p dbpwd1 --db assignment2 --collection arxiv --file arxiv.json
...
...3 document(s) imported successfully.  0 document(s) failed to import
```

Afterwards, MongoDB provides feedback on the number of imported documents, i.e., 1,984,049 and 3 documents have been imported into the collections `dblp` and `arxiv`, respectively. This creates two collections named `dblp` and `arxiv` (cf. Listing 19).

Listing 19: `mongosh` – Show all collections in our database (after import).

```
assignment2> show collections
arxiv
dblp
```

[30] The DBLP Computer Science Bibliography: `https://dblp.uni-trier.de/`
[31] The arXiv repository: `https://arxiv.org/`
[32] Data for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/sq6SNrpAyjQoNtt`

Remarks: (i) The file import will take some time; please wait for it to finish (MongoDB shows the current progress). (ii) The `--file` option also works with full paths, e.g., `--file "C:\Users\dkocher\Desktop\dblp.json"`. (iii) On Windows systems, please execute `mongoimport.exe` from the Windows command-line tool (`cmd.exe` or Powershell) and **not** via double click on the `mongoimport.exe` (it will close immediately).

2.5 Introduction to MQL

Once the data has been imported, try to further familiarize yourself with the `mongosh` command-line tool. Unlike PostgreSQL, MongoDB does not support SQL statements but uses the MongoDB query language (MQL). MQL is an intuitive query language based on the JSON format. MongoDB supports four so-called **CRUD operations: C**reate, **R**ead, **U**pdate, and **D**elete, and we will use read, create, and update operations. The MongoDB documentation often contains equivalent SQL statements for given MQL statements (cf. Section 4). In the following, we briefly introduce the MongoDB query language.

First, the collection on which the operation should be executed must be specified. To refer to a specific collection, MongoDB uses a **Dot** notation [33]. For example, `db.dblp` refers to the DBLP collection in the current database (`db`). Similarly, we can execute an operation on this collection using the name of the operation. The query shown in Listing 20 executes the `find()` operation on the DBLP collection and uses the `pretty()` [34] 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 immediately return all documents, but a so-called *iterator* that can be used to iterate over the documents in the result (indicated by `Type "it" for more`). Consequently, we can retrieve additional results by typing `it` into `mongosh`.

Listing 20: `mongosh` – Execute the `find()` operation on the DBLP collection.

```
assignment2> db.dblp.find().pretty()
[
  {
    _id:  ObjectId("595c2c37a7986c0872f266e7"),
    mdate:  '2014-07-15',
    author:  [ 'John Meyer' ],
    ...
  }
]
Type "it" for more
```

The MQL query in Listing 20 corresponds to the SQL query `SELECT * FROM dblp`, i.e., we ask for all documents of the DBLP collection (without any constraints). Similar to the `WHERE` clause in SQL, we can define criteria to filter the documents. This can be accomplished by passing a JSON object to the `find()` operation. In this JSON object, we can specify criteria a document of the result has to satisfy.

We provide **four** queries (cf. Listings 21– 24) that can be executed out of the box by entering them into the `mongosh` command-line tool (one after another):

[33] The Dot notation: `https://en.wikipedia.org/wiki/Property_(programming)#Dot_notation`
[34] `pretty()`: `https://www.mongodb.com/docs/manual/reference/method/cursor.pretty/`

Listing 21: `mongosh` – Query **Q1**.

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

Listing 22: `mongosh` – Query **Q2**.

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

Listing 23: `mongosh` – Query **Q3**.

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

Listing 24: `mongosh` – Query **Q4**.

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

In query **Q1**, we pass a JSON object (enclosed in curly braces) with a single key-value pair, `"author": "Michael Stonebraker"`, to the `find()` [35] operation. This shows all documents in the collection `dblp` that have been authored by "Michael Stonebraker" [36].

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

Query **Q3** depicts a `JOIN`-like operation by linking documents of two (or more) collections and augmenting the collection that is specified first, i.e., `db.arxiv`. The principle is similar: 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"`). The linking is done using `$lookup` [37], which adds a **new** field to each input document. 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 is an array of all documents that satisfy the criterion, i.e., identical titles in both collections. To this end, we specify the fields of the respective collections that are used to link the documents: `localField` defines the field of the base collection `db.arxiv` and `foreignField` defines the field in the linked collection `dblp`. 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.

The last query, **Q4**, extends query **Q2** with the `explain()` operation [38] right after the

[35] `find()`: `https://www.mongodb.com/docs/manual/reference/method/db.collection.find/`
[36] Michael Stonebraker: `https://en.wikipedia.org/wiki/Michael_Stonebraker`
[37] `$lookup` operator: `https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/`
[38] `explain()`: `https://docs.mongodb.com/manual/reference/method/db.collection.explain/`

operation we actually want to execute (in this case, the `find()` operation). Like `EXPLAIN` in PostgreSQL, this instructs MongoDB to return the **query plan** with information about the steps MongoDB plans to execute. As an exercise (optional), you can also extend the other queries with a trailing `explain()` operation.

2.6 Access the Data Using Python3

The fourth part is to write a small Python3 application. 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 is supposed to execute the queries **Q1**, **Q2**, **Q4** as-is (cf. Section 2.5) and print the results (i.e., all documents; human-readable output format).

Like in Assignment 1, you will adapt query **Q3**: **Q3** as given in Section 2.5 returns a list of documents that satisfy the condition in the `$lookup` operator. In your Python3 code, **Q3** should only return the **number of documents** that satisfy the condition of the `$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` [39] operator) or by adapting the Python3 code such that it counts the documents.

Template Code

Remark: The details for connecting to the MongoDB database differ depending on your setup. Hence, read the comments in the template code and use it properly.

There are many tutorials on installating [40] and using the `pymongo` [41] in combination with MongoDB. Nonetheless, we provide a minimum Python3 template code as a starting point. The template code for this assignment is available as a separate `.py` file via the course website [42] and you can use `curl` or `wget` to download the file (cf. Listing 17).

2.7 Questionnaire

The questionnaire contains questions about this assignment in a separate text file called `assignment2-questionnaire.txt`, which are discussed in the after-assignment meetings.

---

3 Submission

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

[39] `$count` operator: `https://docs.mongodb.com/manual/reference/operator/aggregation/count/`
[40] Installation of the `pymongo` module: `https://pymongo.readthedocs.io/en/stable/installation.html` and `https://pypi.org/project/pymongo/`
[41] `pymongo` tutorial: `https://pymongo.readthedocs.io/en/stable/tutorial.html`
[42] Code: `https://dbresearch.uni-salzburg.at/teaching/2026ss/dim/assignment2-template.py`

Code Please submit a Python3 file (`assignment2.py`) that contains the full code for this assignment. The code **must** print the results of **all** four queries **Q1-4** (one after another) when executed **as submitted**. We will not debug your code, e.g., change a variable to make it work. Therefore, please double-check that your Python3 code works as expected.

Questionnaire Answer the questions directly in the file `assignment2-questionnaire.txt`. If you prefer to use a different application to answer the questions (e.g., Microsoft Word and the likes), you must use one of the following formats: `.txt`, `.pdf`, `.odt`, or `.docx`.

---

4 Supplementary Material

This section provides pointers to material that may be helpful to solve this 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 `mongosh` command-line tool: `https://www.mongodb.com/docs/mongodb-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 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 10 points.

Code The code contributes at most 5 points and is evaluated based on the following criteria (if the code is executed as submitted; disregarding the credentials):

Max. Points	Criterion
1.25	The result of Q1 is correctly printed to the command line.
1.25	The result of Q2 is correctly printed to the command line.
1.25	The result of modified Q3 is correctly printed to the command line.
1.25	The result of Q4 is correctly printed to the command line.
5	

Questionnaire The questionnaire contributes at most 5 points and is evaluated based on the following criteria:

Max. Points	Criterion
1.25	Correctness of answer A1.
1.25	Correctness of answer A2.
1.25	Correctness of answer A3.
1.25	Correctness of answer A4.
5	

After-Assignment Meeting The discussion in the after-assignment meeting contributes at most 10 points, which are awarded based on assignment-specific questions.
