DIT032 / DAT335 – Data Management

Assignment 4

Chalmers | University of Gothenburg Deadline: **So Mar 14th** 23:59

General Remarks

- There are in total 50 points to be reached for this assignment, and you need at least 35 points to pass the assignment. Over the entire course you need to pass 3 of the 4 assignments, and you need at least 25 points on *every* assignments.
- Assignment solutions are to be produced in teams of two students. Discussions with other colleagues (e.g., through Slack) are encouraged, but do not share your assignment solutions, or significant parts, with your colleagues (neither through Slack nor through other means). If we find that multiple groups submitted the same assignments, we are mandated to forward any suspected cases to the Disciplinary Committee (which may decide to suspend students from their studies).
- Remember to also hand in self- and peer assessment forms through Canvas *individually*.
- Shortly after the deadline, we grade the assignment and send you feedback. If you fail the assignment, you have the possibility to submit an improved version of the assignment. Please see Canvas for details on the re-submission procedure (you will also need to submit a detailed change log).

1 Setup MongoDB

As a first step, we again need to install some more software (in case you have not installed MongoDB yet).

1. Download and install the MongoDB community server (Versions <3.6 won't work. We recommend the latest version 4.4.x). Follow the detailed installation instructions for your OS here:
   https://github.com/joe4dev/dit032-setup

2. Import the Conferences and Delegates JSON documents (available from Canvas).
   Find some MongoDB Getting Started resources here:
   https://github.com/joe4dev/dit032-setup/blob/master/MongoDB.md
   To import, run the the following commands in your operating system shell (**not** in the mongo shell which you start by running the `mongo` command!):

```
# macOS shell
mongoimport -d conferences -c conferences conferences.json
mongoimport -d conferences -c delegates delegates.json
# Windows shell
mongoimport /d conferences /c conferences conferences.json
mongoimport /d conferences /c delegates delegates.json
```

   This creates a new database "conferences" with two collections ("conferences" and "delegates"), and imports the JSON documents from the file into them.

3. Start your mongo shell as per the tutorial linked above. For instance, run the command `mongo` in your operating system shell. You may have to first start the mongo database server itself (e.g., run `mongod` or `sudo mongod` in a separate terminal window). Validate that the new database has been created by running `show dbs` in the mongo shell. Switch to the "conferences" database with `use conferences`. Validate that two collections (conferences and delegates) have been created, and that there are some documents in both of them.

2 [MGDB1] Improving an Existing Semi-Structured Data Model (10 points)

Consider the example JSON file `server1.json` (available on Canvas). It is a faulty partial implementation of the EER model from Assignment 2 in JSON.

MGDB1.1 Which problems are in this JSON file? Explain in your own words which syntactical rules are not followed, and/or which parts do not correctly implement the domain described by the EER diagram. Then fix all problems *without* losing any information (i.e., don't just remove offending elements from the file). There are three types of problems in the file, but some come up more than once.
Submission: Save your fixed JSON as `server1_fixed.json` and your explanation in `mgdb1.txt`.

MGDB1.2 Extend the fixed file to also include information about one data center and at least two users and their roles (in the data center). Refer again to the EER diagram in Assignment 2 to decide which information you need to store. Document any notable design decisions. Particularly document and motivate if your design deviates from the EER model.
Submission: Save your extended JSON as `dc1.json` and your explanation in `mgdb1.txt`.

3 [MGDB2] Semi-Structured Data Representation (12 points)

Recall the Mondial schema and database that we used for the second and third assignments. Now think about how this relational data could be represented in a semi-structured data format.

MGDB2.1 Decide on a JSON format to represent continents (one JSON file should represent one continent). Continent files should contain basic information about the continent, the 5 largest countries in this continent, and the 10 biggest cities. Further store basic information about these countries and cities (you can decide what is most interesting to save). You don't need to implement every information available in the relational model, but the resulting JSON files should have at least 15 different fields. Make use of nested elements to represent relationships between the entities, and use array fields to represent lists. Create an example document for **two** continents. Use SQL to query the information that you need to construct your examples. Make sure that your JSON files are syntactically correct JSON. There are many free online services available that you can use to validate JSON (for example here[1]).
Submission: Save the examples into two JSON files (e.g., `europe.json` and `asia.json`).

MGDB2.2 Now create semantically equivalent XML representations for your two example files. Make use of the various syntactical features of XML, and ensure that the resulting XML files are clean. Ensure that all of the following is used at least once in a way that makes sense: attributes, subelements, and text nodes. *There are tools that automatically convert JSON to XML, but the resulting files are typically **not** a good XML, and will not be accepted for this task.* Again ensure well-formedness for your documents (for example using this service[2]).
Submission: Save your new representations to two new files (e.g., `europe.xml` and `asia.xml`).

4 [MGDB3] Querying MongoDB on a Simple Database (20 points)

Write the following queries against the conferences-and-delegates database with the *conferences* and *delegates* collections you imported. Use the query-by-example API (i.e., `db.<collection>.find()` and its variants[3]) in the mongo shell. You will need to refer to the documentation of MongoDB to assemble the queries (the few example statements covered in the lecture will not be sufficient to solve these tasks).

MGDB3.1 Count how many conferences are held in the year 2019.

MGDB3.2 Find all delegates from Canada, and return only their name and _id fields.

MGDB3.3 Find the countries of all delegates. Return a list of unique values (i.e., make sure that no duplicates are in the result if multiple delegates are from the same country).

MGDB3.4 Find all conferences with a research-oriented track (i.e., where the track topic is "research").

MGDB3.5 Find all conferences that started in April 2019.

MGDB3.6 Find all conferences that have a track with a two or more chairs.

MGDB3.7 Find all iterations of the ICSE conference. Hint: you can use regular expressions[4] for this task.

[1] https://jsonlint.com
[2] https://www.xmlvalidation.com
[3] https://docs.mongodb.com/manual/reference/method/db.collection.find/
[4] https://docs.mongodb.com/manual/reference/operator/query/regex/

MGDB3.8 Change the name of the delegate Joanne M. Atlee to "Joanne Atlee".

MGDB3.9 Remove the city information from all conferences in India.

MGDB3.10 Find the names and editions (but not the _id) of all conferences where the general chair is from India. *Hint:* you may need need to use the aggregation pipeline[5] to solve this query.

4.1 Submission Information

Create a simple text file called `mgdb3.js` containing all queries. Use comments in the style `// MGDB<number>` to mark each query.

```
// MGDB3.8
db.conferences.command1()

// MGDB3.9
db.delegates.command2()
...
```

4.2 Tips

- Both, the official documentation[6] and TutorialsPoint[7] have a lot of easy-to-follow examples for interacting with MongoDB and the mongo shell. Use these to get started.
- If you want to check out what the data in the database looks like, you can just open the files "conferences.json" and "delegates.json" (which you imported in the first step) using any plain text editor or IDE.
- If you have a hard time with the commandline client of MongoDB you can also check out Robo 3T[8], a GUI for MongoDB.
- For this part of the assignment you will need to use the operator reference list[9] and other parts of the MongoDB online documentation.

5 [MGDB4] Querying Real-Life MongoDB Data (8 points)

As a last task, we now write some more queries against a real-life database. Download the database dump from Canvas (`gh-issues.zip`), unzip the file, and import it into your MongoDB instance.

```
   # in your OS shell, should work the same on Mac and Windows
mongorestore <UNZIPPED_DIR>
```

[5] https://docs.mongodb.com/manual/reference/operator/aggregation/
[6] https://docs.mongodb.com/manual/
[7] https://www.tutorialspoint.com/mongodb/index.htm
[8] https://robomongo.org
[9] https://docs.mongodb.com/manual/reference/operator/

(replace `<UNZIPPED_DIR>` with the directory you just unzipped). If the import is successful, your MongoDB instance should now contain a new database called "gh-issues" with three collections (issues, events, and comments). The `issues` collection should have 27350 entries, the other two collections 62605 and 120079 respectively. This database contains some issues and pull requests, as well as comments on these issues for 50 large open-source projects from the platform GitHub[10] (including all metadata about issues and comments).

A common task in empirical software engineering research is to study datasets like this. Write the following queries, which are similar to tasks that a researcher studying this data may encounter. You will need to first explore the structure of the database by investigating example documents from each collection (e.g., by running `db.<collection>.find().pretty()`), and then combine various operators similar to Task 3.

Submission: Create a simple text file called `mgdb4.js` containing all queries. Label the queries as described for MGDB3.

MGDB4.1 Find the repository URLs of all distinct projects in the dataset. *Hints:* issues are associated to projects, check the structure of issue documents in the database to see where you could find project URLs; then use the `db.<collection>.distinct` function.

MGDB4.2 Find the login names of all users who have created an open issue. *Hint:* check the state of issues, and then use again the `distinct` function to avoid listing the same user multiple times.

MGDB4.3 Write a query that calculates how many percent of comments in the comments collection have been submitted by a bot (check if the field "user.type" has the value "Bot"). *Hint:* you don't have to write this in a single operation, you can also find out how many issues there are by bots and in total, and then have your code calculate it from there.

MGDB4.4 For each issue in the issue collection, create a document that contains all information of the issue and all comments on this issue from the comment collection (that is, "join" the two collections). *Hint:* if you inspect the documents you will see that they have fields that can serve as "foreign keys"; then use the aggregation pipeline[11] and the `$lookup` operator.

6 [MGDB5] MongoDB Replication Sets and Sharding (for the glory)

In the lecture, we discuss how distributed databases can be used for either, data replication (insurance against data loss) or load balancing (splitting up data between multiple machines). MongoDB uses the notion of replica sets for replication[12], and sharding[13] for load balancing.

MGDB4.1 First read up on the two articles linked above. Try to mentally map what you read in the documentation to what we learned in the lecture, and try to understand how replica sets and sharding could be used to support a large, real-life MongoDB database.

MGDB4.2 Now assume you are responsible for rolling out a much larger version of the conferences-and-delegates database for world-wide usage. What replica sets would make sense, and why? What sharding strategy would you suggest? Describe it textually (potentially using a diagram of hosts and the data mapped to them).

[10] https://github.com
[11] https://docs.mongodb.com/manual/reference/operator/aggregation/
[12] https://docs.mongodb.com/manual/replication/
[13] https://docs.mongodb.com/manual/sharding/

MGDB4.3 Finally, try to set up a small distributed database on your computer. This will require some configuration (you will need to have different MongoDB configurations bound to different ports to simulate multiple database instances on a single physical computer). After you have successfully set up multiple MongoDB instances, you can follow this example[14] to configure multiple shards and replica sets. Now load some example JSON files (you can just add some of the examples we used so far multiple times, potentially with some changed data) into your newly deployed local Mongo cluster, and experiment with it: what happens if you query specific instances? what happens if you turn specific MongoDB server instances on or off? Report on your experiences.

[14] https://www.howtoforge.com/tutorial/deploying-mongodb-sharded-cluster-on-centos-7/