**Report**

Assignment 3 - MongoDB

**Group**: 69

**Students**: Hauk Aleksander Olaussen, Vidar Michaelsen, Noran Baskaran

**Disclaimer**In Assignment 2 we used the wrong way of finding the size of an activity. When we were to check if the file length was over 2500 lines (trackpoints), we used pandas.DataFrame.size which does not give the proper line count, but the number of elements in the data frame. This caused our preprocessor to exclude a lot of data that should have been included. We have since fixed this issue in this assignment – and all the data should be present. You can see the place where we went wrong on line 117 in Preprocessor.py.  
**NOTE!**  
The queries and solutions work as intended when all the data is present.

**Introduction**

We used the same file architecture as we did in Assignment 2.  
  
The code for preprocessing of the data can be found in the python file Preprocessor.py. This class will read the data from the provided dataset, and place the preprocessed versions of it into two massive lists of dictionaries – which the DbHandler will later use to insert the data to MongoDB. We have decided to only use two collections this time – one for users, and one for trackpoints. As none of the activities are shared by multiple users, this makes sense. We initially tried to only use one collection (users) with the trackpoints being an attribute of the activities as well. More on this in the discussion part.

All the answers for part 2 can be found within the Queries.py. We solved the tasks by relying heavily on python. Most of the solutions fetches the data needed to solve the task and uses python code almost exclusively to find the result. Running the Queries.py file will run all the task functions and print their answers to the terminal. Images and runtimes for each function/query can be found in the next section.

For this assignment we have worked together physically at campus in a very similar fashion as Assignment 2. Hauk and Noran pair programmed the whole part 1 of the assignment and sent the data to a database which is (once again) located at his home desktop.

For part 2 of the assignment, we all worked together for all the solutions, with most of them solved using almost only python. We tried as best we could to find efficient and scalable solutions.

Link to repo: [GitHub](https://github.com/Olaussen/TDT4225/tree/main/assignment3)

To connect to the database, fill a .env file with the following data:

HOST=84.202.106.55

**Results from Part 1**

Image below shows the user collections. This collection also contains all the activities that belongs to the user.

**Et bilde som inneholder bord

Automatisk generert beskrivelse**

This image shows how the activity object is structured inside the users.

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

This image the first 10 trackpoints from our collection trackpoints.

**Et bilde som inneholder bord

Automatisk generert beskrivelse**

This image shows how the trackpoints are structured.

Et bilde som inneholder tekst

Automatisk generert beskrivelse

**Result from Part 2**

**Task 1:**

Screenshot of terminal showing result of function: count\_all\_entries().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 2:**

Screenshot of terminal showing result of function: average\_max\_min().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 3:**

Screenshot of terminal showing result of function: top\_10\_users().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 4:**

Screenshot of terminal showing result of function: started\_one\_day\_ended\_next().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 5:**

Screenshot of terminal showing result of function: duplicate\_activities().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 6:**

Screenshot of terminal showing result of function: covid\_19\_tracking().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 7:**

Screenshot of terminal showing result of function: user\_no\_taxi().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 8:**

Screenshot of terminal showing result of function: transportation\_mode\_usage().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 9:** Screenshot of terminal showing result of function: year\_month\_most\_activities() and user\_with\_most\_activities\_11\_2008().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 10:**

Screenshot of terminal showing result of function: user\_112\_walk\_2008().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 11:**

Screenshot of terminal showing result of function: top\_20\_users\_altitude().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

**Task 12:**

Screenshot of terminal showing result of function: year\_invalid\_activities().

**Et bilde som inneholder bord

Automatisk generert beskrivelse**

**Discussion**

**Part 1: Data preprocessing and insertion**As MongoDB is a NoSQL database using the document model we were – as stated in the introduction – able to combine the users and activities in one document. This differs from the SQL way of doing it, as the activities were located in their own table using foreign keys. The way of structuring the user and activity objects themselves is not changed however, but the activities are contained inside a user’s activities array. We have changed the way the trackpoints are structured. Instead of having lat and lon as their own attributes, we have combined them using MongoDB’s own GeoJSON objects. This allows for more intricate queries, where one can get information about distance and other geographical data directly in the queries. It is also possible to directly plot the data on a map using MongoDB charts using this datatype.

As stated in the introduction, we initially wanted to try to combine all the data into one collection – as the all the data has a one-to-many structure. We were not allowed to this however, as the maximum size of a document in MongoDB is 16MB. Hence, we decided to combine the users and activities, and leave the trackpoints in their own collection with a key to the activity it belongs to.

**Part 2: Queries and code**Compared to when using SQL we relied more on the application code in python when using MongoDB. This is because some of the complex queries that are easy to find in SQL, are very hard to do in MongoDB. This is because joins are a real pain when using MongoDB, and the NoSQL databases in general. However, this did not make the solutions harder to find, but simply relies more on the computational power of the computer running the code. It also means that most often, the actual MongoDB query returns more unnecessary data from the database that its SQL counterpart.

We know that we could have solved more of the tasks can be solved by making more use of aggregation and more complex queries but decided that writing a few lines of code in python made for a simpler solution as it is more readable, and understandable than a query. Task 1 and 7 are examples where we solved the task using just queries.

As in Assignment 2, for task 5, we assume that “duplicate activities” mean that they have the same start\_date\_time, end\_date\_time and transportation\_mode, as these are the only attributes with actual meaning for the activity itself. We do not check activities within one user, as no users has duplicates. We know this, as they would then share the \_id attribute, as this is the start date combined into a number. We also do not check activities two ways – meaning that an activity is only counted one time.

As tasks 11 and 12 made us fetch almost all the trackpoints, we created an index in the database on the activity\_id field, significantly reducing the time it took to solve these tasks.

Note that all the solutions are commented directly in the code. This will give a more rounded understanding of how the task at hand was solved.

­

For us, the biggest difference between using the MySQL and MongoDB was that this assignment involved a lot more application-level code than the last one. Because of this, we preferred a NoSQL database as we are more familiar with python code. Some tasks that where very easy to solve with the join operation in MySQL, was now taking a lot of resources and lines of code (task 11 and 12). With MongoDB we also did not have to define a schema like we did in MySQL, we just inserted the objects directly from the lists of dictionaries. For this assignment we think MySQL is better because it is more powerful when handling structured data with a clear schema. This fits the assignment dataset and tasks better in our opinion, as many of the tasks are highly relational.