**Report**

Assignment 2 - MySQL

**Group**: 69

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

**Introduction**The code for preprocessing of the data can be found in the python file Preprocessor.py. This file will read the data from the provided dataset, and dump the cleaned and preprocessed data into three separate files – namely users.pickle, activities.pickle and trackpoints.pickle. As you might notice, we make use of the python library pickle. To run the preprocessing, run the Preprocessor.preprocess() function. The insertion of all the data into the database from the provided folder took around 7 minutes on the desktop that hosts the database.

All the answers for part 2 can be found within the Queries.py. Many were done using only SQL, but some needed more work with python. One function exists for each task, clearly commented in the file to easily find the one you’re after. Other comments explain parts of the code not necessarily easily understood. Running the Queries.py file will run all the 12 queries and print their answers to the terminal. Images and runtimes for each function/query can be found in the next section.

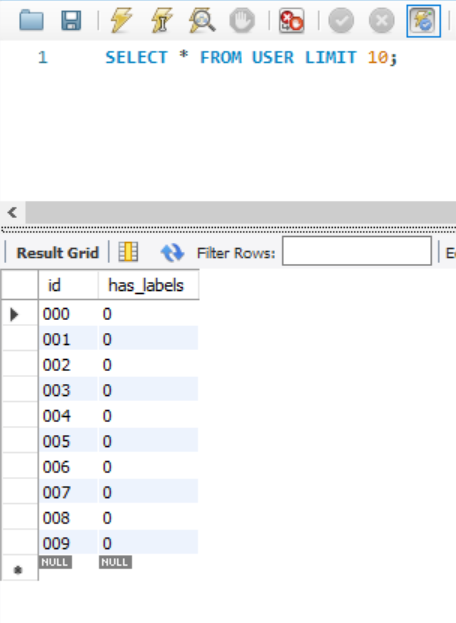
For this assignment we have worked together physically at campus. Hauk did the whole part 1 of the assignment and sent the data to a database which is located at his home desktop, and the rest of the group did a code review to confirm and check if this process was done efficiently and correct.

For part 2 of the assignment, we created the queries together and Hauk put them in the queries script and modified them to print the results in a readable format.

Link to repo: [GitHub](https://github.com/Olaussen/TDT4225/tree/main/assignment2)

**Results  
  
Results from part 1**

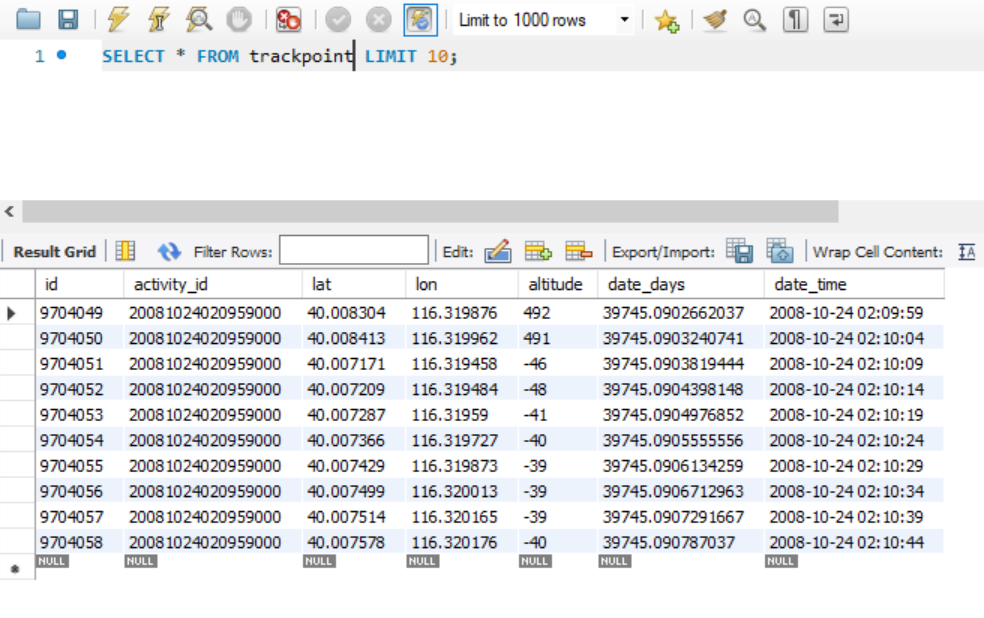
The table containing the users.

****

The table containing the activities. **Et bilde som inneholder tekst

Automatisk generert beskrivelse**

The table containing the trackpoints.

****

**Results from part 2**

**Task 1:**Screenshot of terminal showing the result of a function total\_amount\_of\_entries().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**  
**Runtime**: ca. 19.5 seconds

**Task 2:**Screenshot of terminal showing result of function min\_max\_avg\_activites().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**

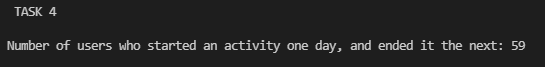
**Runtime**: ca. 60ms

**Task 3:**Screenshot of terminal showing result of function top\_ten\_users\_by\_activites().

**Et bilde som inneholder tekst, elektronikk

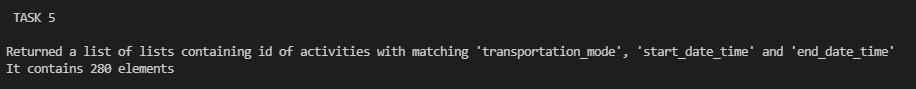
Automatisk generert beskrivelse  
Runtime**: ca. 36ms

**Task 4:**Screenshot of terminal showing result of function users\_start\_on\_one\_day\_end\_the\_next\_day().

****

**Runtime**: ca. 32ms

**Task 5:**Screenshot of terminal showing result of function find\_duplicate\_activities(). What this does is check if two activities has the same transportation\_mode, start\_date\_time and end\_date\_time, making them “equal”.

**  
Runtime**: ca. 45ms

**Task 6:**Screenshot of terminal showing result of function covid\_19\_tracking().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime**: ca. 3 minutes

**Task 7:**Screenshot of terminal showing result of function never\_taken\_taxi().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime**: ca. 280ms

**Task 8:**Screenshot of terminal showing result of function transportation\_mode\_count().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse**  
**Runtime**: ca. 31ms

**Task 9:**Screenshot of terminal showing result of function most\_active\_year()

and the result of function user\_with\_most\_activities\_from\_9a().  
The user with the most activities (025) have also the most total hours this month.

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime:** ca. 0.31s for 9a

ca. 0.32s for 9b

**Task 10:**Screenshot of terminal show result of function distance\_walked\_in\_2008().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime**: ca. 535ms

**Task 11:**Screenshot of terminal show result of function most\_altitude\_gained().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime**: ca. 3 seconds

**Task 12:** Screenshot of terminal show result of function invalid\_activities().

**Et bilde som inneholder tekst

Automatisk generert beskrivelse  
Runtime**: ca. 3.5 seconds

**Discussion**

For part one of the project, we learned a lot. At first we did not use the cursor.executemany() function and only used cursor.execute(), where we inserted by executing and committing one record and at a time, which took an ungodly amount of time. But after optimizing our preprocessor (Preprocessor.py) and our database handler (DbHandler.py) the process of cleaning and dumping the data into different files then inserting it, got sped up from painful 20 hours to basically 5 minutes. This is because when we tried to commit entries one at a time it wound take one trip to the database and back each insertion, but with executemany(), we do a batch insertion where we only do one round trip total – saving a lot a time. We could not get the executemany() function to work with the whole list of trackpoints (over 1 million entries). This might be because of how MySQL is set up at the home desktop – and we were not able to alter this from our laptops from campus. Because of this, we needed to split this list into separate chunks and insert these separately. We batch insert 10 000 at a time, for a total of 120 chunks/commits. All the code for inserting data can be found in DbHandler.py.

For part two, almost all the tasks went smoothly except for task 5 and 6.   
For task 5 we assume the task as finding the ids of the task where the fields transportation\_mode, start\_date\_time and end\_date\_time are equal – as this is the only data in the activity not being a unique id or foreign key. We would not get matches otherwise since they all have unique ids. Therefore, we think this was the most sensible interpretation.   
Task 6 was not hard to solve, but heavy computational. We did find a solution in pure SQL but dropped it. This is because the solution we found by finding activities with overlapping times in SQL and distance calculation in python performed a lot better.

Some of the data needed to be added to the database were not explicitly written to the files given in the assignment. An example of this is the start and end times for an activity. To find this data, we needed to read the date and time for the first and last trackpoint for each activity – and use these values for the fields in the activity table.