NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4225 - Very Large, Distributed Data Volumes

Assignment 2 - MySQL

Group:
6

Students:
Tord Johan Espe
Sivert Eggen
Kristoffer Grude

06.10.2023

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Intro to task and dataset

In this assignment we have used the given dataset of trajectories, which was based on the open source Geolife GPS Trajectory dataset from Microsoft. The modified dataset contained 182 users that in total had tracked 18 669 activities, where each activity contained a number of GPS trackpoints. The technical tasks in this assignment were divided into 2 parts:

1. Cleaning the data in the dataset based on rules that were given in the assignment, as well as inserting the data into the defined tables that made up our database. As part of this process we used the suggested table scheme given in the assignment, in addition to creating some additional columns for performance reasons, which we will discuss later.
2. Writing queries using SQL and Python on the database that we created, trying to answer questions about the dataset, and visualising the results of the queries.

1.2 Assumptions

In the cleaning process of the exercise we have done some assumptions about the users, the trajectory files and their relation to activities.

Firstly, we have included users without labels in the database, so we are not only looking at the users that had labeled their activities. Secondly, upon inspection of the data we saw that some trajectory files for the users that had labeled their activities contained trackpoints for different activities labeled on the same date. We could see this by looking at start- and end-times for the different activities labeled on the same date, and that all those start- and end-times were included in a trajectory file belonging to that date. Thus, for users with labeled activities, a trajectory file could contain more than one activity. For these users, we only added activities that match perfectly with the users labeled activities, and no other activities.

For the users that did not label their activities, we have still included their activities. However, since there is no way to divide a trajectory file into more activities for these users, we now assumed as the assignment sheet says; that every *.plt*-file corresponded to one activity. Thus, both users with and without labeled activities have activities in the database.

So to summarize:

- For users that labeled their activities, these activities were assumed to be the only activities these users had. A trajectory file could contain multiple activities
- For users without labeled activities, each trajectory file corresponded to one activity only

1.3 Method

The working method for this assignment mainly involved physical group work and pair programming, ensuring that every member of the group was up to date with the project at all times. Although work was distributed in some sense among the group members, it was important that every person was involved in all tasks, as part 1 and 2 depended on each other.

In order to get started we followed the steps in the setup guide given in the assignment description. When working on the project together, every group member connected to the same virtual machine where we initially created the database and project folder connected to a github repository. The repository is linked below, and was used mainly for backup.

1.4 Link to github

https://github.com/tjespe/bigdata/tree/master/assignment2

2 Results

2.1 Part 1

This section will elucidate our database design, covering the ER-diagram and table format, and showcase the outcomes of the created database tables. In the Appendix section, a complete set of SQL queries used for creating the database is attached, exactly as they were written. These queries can also be located in the accompanying code files submitted with this report.

2.1.1 Database and table structure

Under in Figure 1, an ER-diagram of the database and the relationships between the tables are illustrated for what we thought was the most appropriate structure for the database, based on our understanding of the dataset and tasks.

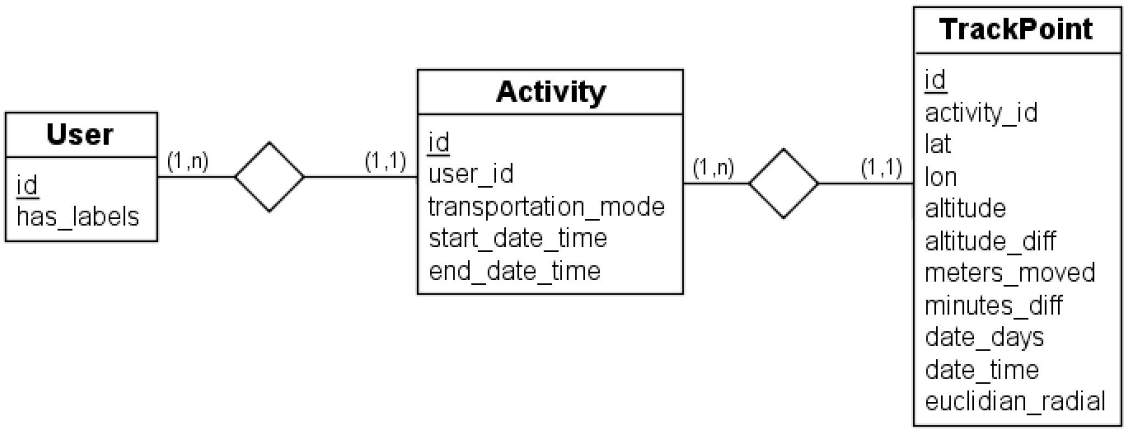


Figure 1: ER-diagram of the database

Table 1 shows the type for each column in the the three tables User, Activity and TrackPoint for our decided data structure.

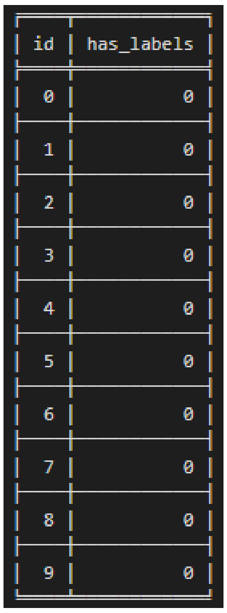
Table 1: Table format of the database

User	**Activity**	**TrackPoint**
id - int *(Primary key)* has_labels - boolean	id - int *(Primary key)* user_id - int *(Foreign Key)* transportation_mode - string start_date_time - datetime end_date_time - datetime	id - int *(Primary key)* activity_id - int *(Foreign Key)* lat - double lon - double altitude - double altitude_diff - double meters_moved - double minutes_diff - double date_days - double date_time - datetime euclidian_radial - double

Additional columns added or modified to the suggested data structure:

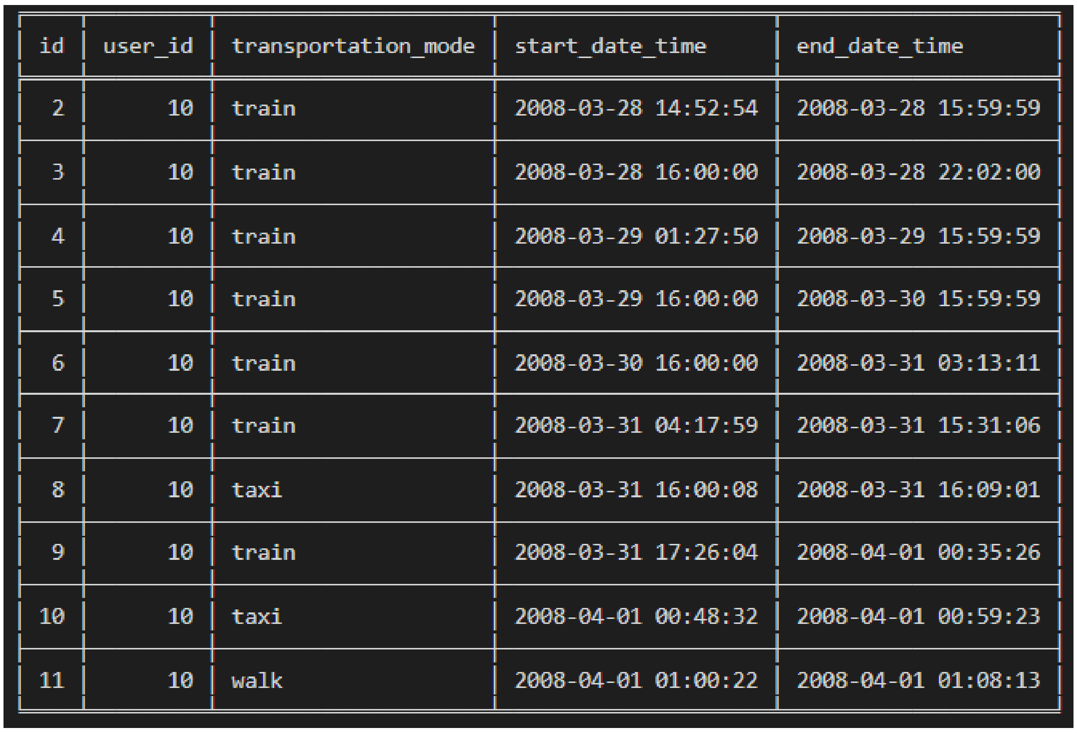
- **altitude**: Altitude converted from feet to meters
- **altitude_diff**: The difference between current altitude and previous altitude for consecutive TrackPoints
- **meters_moved**: Haversine distance moved between (lat, lon) and previous (lat, lon) for consecutive TrackPoints in meters
- **minutes_diff**: Time difference since last TrackPoint in minutes
- **euclidian_radial**: Radial distance in a Euclidean space with two dimensions (distance from center of world and time) used for querying for users that have been close to each other in time and space. See discussion for further explanation

2.1.2 Screenshots of database table entries



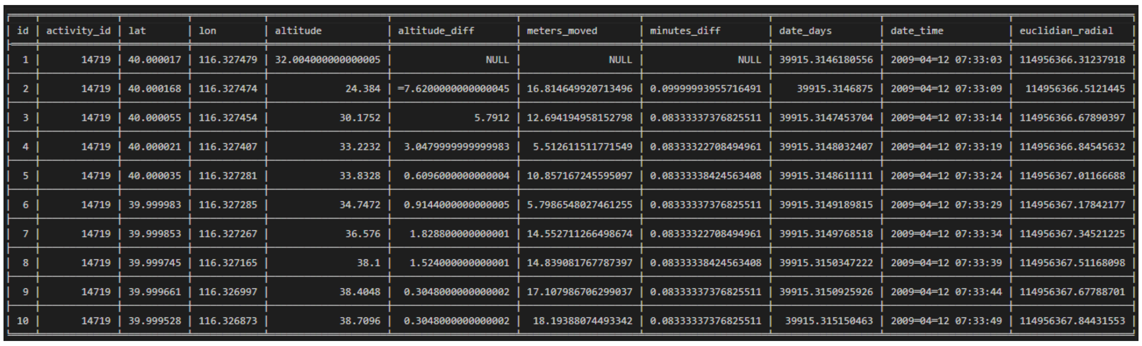
```
╒══════╤══════════════╕
│ id   │ has_labels   │
╞══════╪══════════════╡
│  0   │            0 │
├──────┼──────────────┤
│  1   │            0 │
├──────┼──────────────┤
│  2   │            0 │
├──────┼──────────────┤
│  3   │            0 │
├──────┼──────────────┤
│  4   │            0 │
├──────┼──────────────┤
│  5   │            0 │
├──────┼──────────────┤
│  6   │            0 │
├──────┼──────────────┤
│  7   │            0 │
├──────┼──────────────┤
│  8   │            0 │
├──────┼──────────────┤
│  9   │            0 │
╘══════╧══════════════╛
```

Figure 2: The 10 first rows of the User-table



id	user_id	transportation_mode	start_date_time	end_date_time
2	10	train	2008-03-28 14:52:54	2008-03-28 15:59:59
3	10	train	2008-03-28 16:00:00	2008-03-28 22:02:00
4	10	train	2008-03-29 01:27:50	2008-03-29 15:59:59
5	10	train	2008-03-29 16:00:00	2008-03-30 15:59:59
6	10	train	2008-03-30 16:00:00	2008-03-31 03:13:11
7	10	train	2008-03-31 04:17:59	2008-03-31 15:31:06
8	10	taxi	2008-03-31 16:00:08	2008-03-31 16:09:01
9	10	train	2008-03-31 17:26:04	2008-04-01 00:35:26
10	10	taxi	2008-04-01 00:48:32	2008-04-01 00:59:23
11	10	walk	2008-04-01 01:00:22	2008-04-01 01:08:13

Figure 3: The 10 first rows of the Activity-table



id	activity_id	lat	lon	altitude	altitude_diff	meters_moved	minutes_diff	date_days	date_time	euclidian_radial
1	14719	40.000017	116.327479	32.004000000000005	NULL	NULL	NULL	39915.3146180556	2009=04=12 07:33:03	114956366.31237918
2	14719	40.000168	116.327474	24.384	=7.6200000000000045	16.814649920713496	0.09999993955716491	39915.3146875	2009=04=12 07:33:09	114956366.5121445
3	14719	40.000055	116.327454	30.1752	5.7912	12.694194958152798	0.08333337376825511	39915.3147453704	2009=04=12 07:33:14	114956366.67890397
4	14719	40.000021	116.327407	33.2232	3.0479999999999983	5.512611511771549	0.08333322708494961	39915.3148032407	2009=04=12 07:33:19	114956366.84545632
5	14719	40.000035	116.327281	33.8328	0.6096000000000004	10.857167245595097	0.08333338424563408	39915.3148611111	2009=04=12 07:33:24	114956367.01166688
6	14719	39.999983	116.327285	34.7472	0.9144000000000005	5.7986548027461255	0.08333337376825511	39915.3149189815	2009=04=12 07:33:29	114956367.17842177
7	14719	39.999853	116.327267	36.576	1.8288000000000001	14.552711266498674	0.08333322708494961	39915.3149768518	2009=04=12 07:33:34	114956367.34521225
8	14719	39.999745	116.327165	38.1	1.5240000000000001	14.839081767787397	0.08333338424563408	39915.3150347222	2009=04=12 07:33:39	114956367.51168098
9	14719	39.999661	116.326997	38.4048	0.3048000000000002	17.107986706299037	0.08333337376825511	39915.3150925926	2009=04=12 07:33:44	114956367.67788701
10	14719	39.999528	116.326873	38.7096	0.3048000000000002	18.19388074493342	0.08333337376825511	39915.315150463	2009=04=12 07:33:49	114956367.84431553

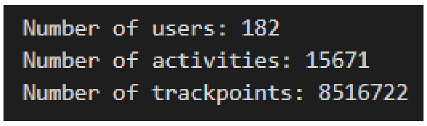
Figure 4: The 10 first rows of the TrackPoint-table

These results was produced by the queries in Appendix A.

2.2 Part 2

In the following section you can see the results outputted for each of the 12 queries in the assignment, both in raw output format and textual. In the Appendix all queries are also included the way they were written, which can also be found in the code files submitted alongside this report.

2.2.1 Number of users, activities and trackpoints in the dataset

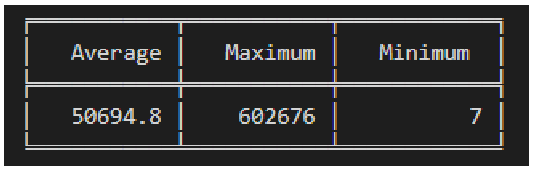


```
Number of users: 182
Number of activities: 15671
Number of trackpoints: 8516722
```

Figure 5: Result from SQL query for task 1

The SQL query gives the results shown in Figure 5 above. There are 182 users, 15 671 different activities and 8 516 722 trackpoints. This result was produced by query 1 in Appendix B.

2.2.2 Average, maximum and minimum number of trackpoints per user

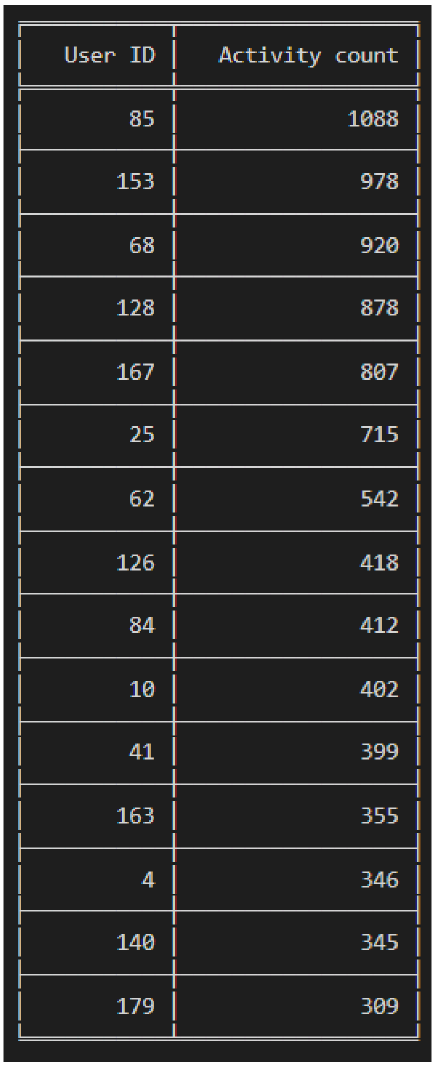


```
+---------+---------+---------+
| Average | Maximum | Minimum |
+---------+---------+---------+
| 50694.8 |  602676 |       7 |
+---------+---------+---------+
```

Figure 6: Result from SQL query for task 2

The SQL query gives the results shown in Figure 6 above. The average number of trackpoints per user is 50 694.8, the minimum is 7 and the maximum is 602 676 trackpoints. This result was produced by query 2 in Appendix C.

2.2.3 Top 15 users with the highest number of activities

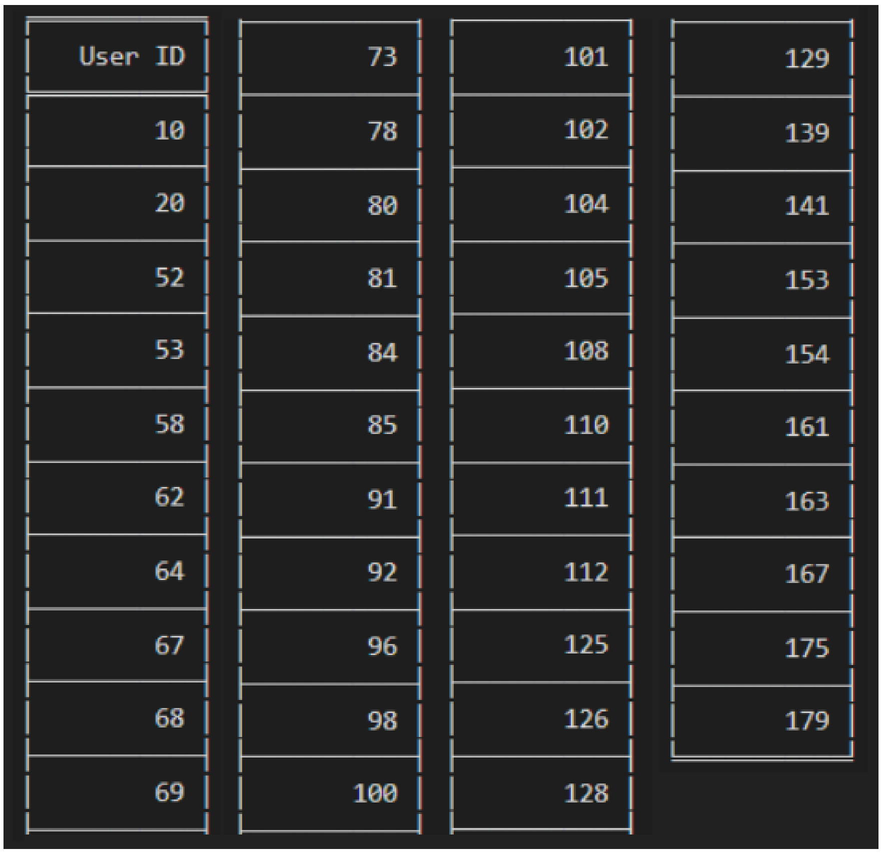


```
+---------+----------------+
| User ID | Activity count |
+---------+----------------+
|      85 |           1088 |
+---------+----------------+
|     153 |            978 |
+---------+----------------+
|      68 |            920 |
+---------+----------------+
|     128 |            878 |
+---------+----------------+
|     167 |            807 |
+---------+----------------+
|      25 |            715 |
+---------+----------------+
|      62 |            542 |
+---------+----------------+
|     126 |            418 |
+---------+----------------+
|      84 |            412 |
+---------+----------------+
|      10 |            402 |
+---------+----------------+
|      41 |            399 |
+---------+----------------+
|     163 |            355 |
+---------+----------------+
|       4 |            346 |
+---------+----------------+
|     140 |            345 |
+---------+----------------+
|     179 |            309 |
+---------+----------------+
```

Figure 7: Result from SQL query for task 3

The SQL query gives the results shown in Figure 7 above. The top 15 users with the highest number of activities are users with id 85, 153, 68, 128, 167, 25, 62, 126, 84, 10, 41, 163, 4, 140 and 179 in descending order respectively. This result was produced by query 3 in Appendix D.

2.2.4 All users who have taken a bus

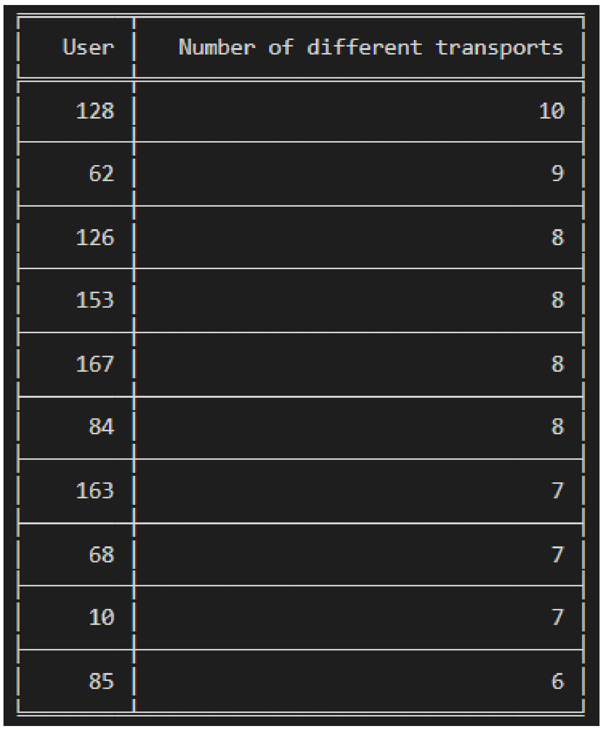


```
User ID     73    101    129
     10     78    102    139
     20     80    104    141
     52     81    105    153
     53     84    108    154
     58     85    110    161
     62     91    111    163
     64     92    112    167
     67     96    125    175
     68     98    126    179
     69    100    128
```

Figure 8: Result from SQL query for task 4

The SQL query gives the results shown in Figure 8 above. Users that have taken the bus are users with ID 10, 20, 52, 53, 58, 62, 64, 67, 68, 69, 73, 78, 80, 81, 84, 85, 91, 92, 96, 98, 100, 101, 102, 104, 105, 108, 110, 111, 112, 125, 126, 128, 129, 139, 141, 153, 154, 161, 163, 167, 175 and 179. In total there are 42 users that have taken the bus. This result was produced by query 4 in Appendix E.

2.2.5 Top 10 users by their amount of different transportation modes

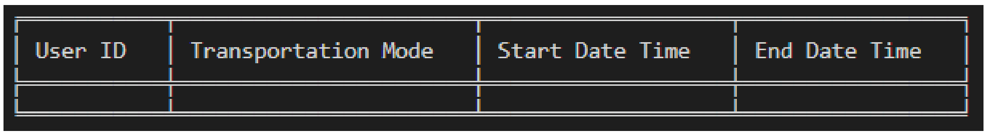


User	Number of different transports
128	10
62	9
126	8
153	8
167	8
84	8
163	7
68	7
10	7
85	6

Figure 9: Result from SQL query for task 5

The SQL query gives the results shown in Figure 9 above. The top 10 users by amount of different transportation modes are users with ID 128, 62, 126, 153, 167, 84, 163, 68, 10 and 85 in descending order respectively. This result was produced by query 5 in Appendix F.

2.2.6 Activities that are registered multiple times

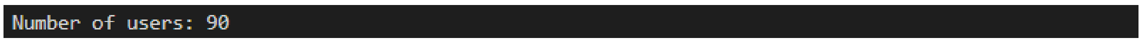


User ID	Transportation Mode	Start Date Time	End Date Time

Figure 10: Result from SQL query for task 6

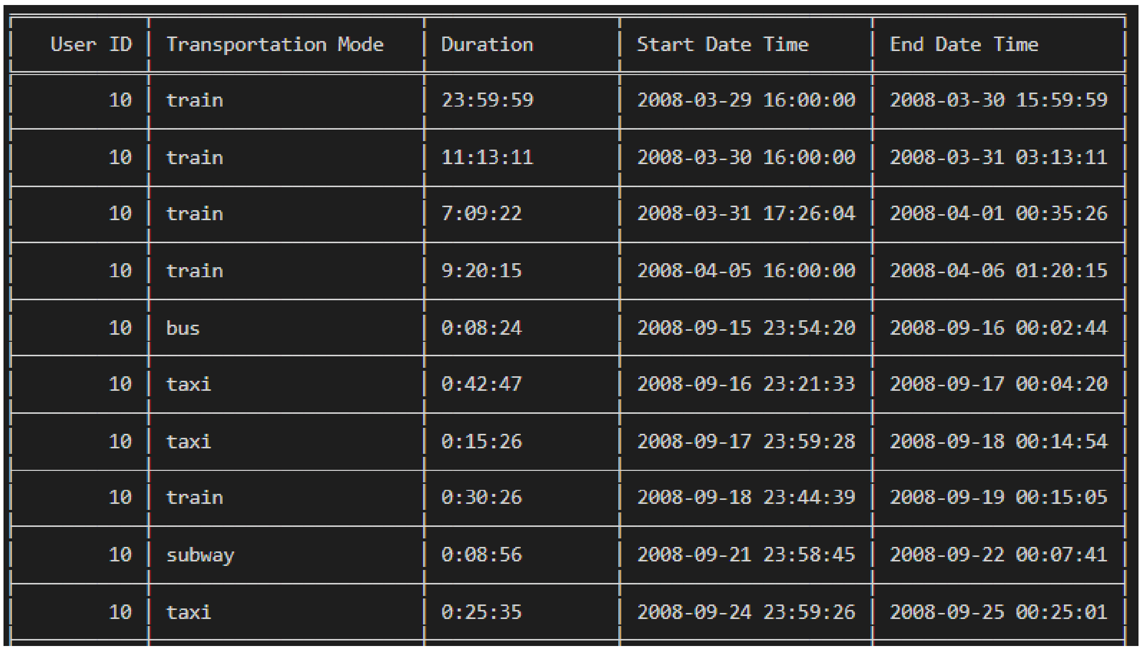
The SQL query gives the results shown in Figure 10 above. There are no activities that are registered multiple times. This result was produced by query 6 in Appendix G.

2.2.7 Users that have started an activity one day and ended it the next day



```
Number of users: 90
```

Figure 11: Result from SQL query for task 7a



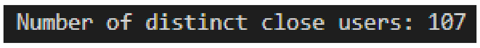
```
| User ID | Transportation Mode | Duration | Start Date Time     | End Date Time       |
|      10 | train               | 23:59:59 | 2008-03-29 16:00:00 | 2008-03-30 15:59:59 |
|      10 | train               | 11:13:11 | 2008-03-30 16:00:00 | 2008-03-31 03:13:11 |
|      10 | train               | 7:09:22  | 2008-03-31 17:26:04 | 2008-04-01 00:35:26 |
|      10 | train               | 9:20:15  | 2008-04-05 16:00:00 | 2008-04-06 01:20:15 |
|      10 | bus                 | 0:08:24  | 2008-09-15 23:54:20 | 2008-09-16 00:02:44 |
|      10 | taxi                | 0:42:47  | 2008-09-16 23:21:33 | 2008-09-17 00:04:20 |
|      10 | taxi                | 0:15:26  | 2008-09-17 23:59:28 | 2008-09-18 00:14:54 |
|      10 | train               | 0:30:26  | 2008-09-18 23:44:39 | 2008-09-19 00:15:05 |
|      10 | subway              | 0:08:56  | 2008-09-21 23:58:45 | 2008-09-22 00:07:41 |
|      10 | taxi                | 0:25:35  | 2008-09-24 23:59:26 | 2008-09-25 00:25:01 |
```

Figure 12: Result from SQL query for task 7b

The SQL query gives the results shown in Figure 11 and Figure 12 above. There are 90 different users that have started an activity in one day and ended the activity the next day (Figure 11). This result was produced by query 7a in Appendix H. The transportation mode, users id and duration for these activities are listed in Figure 12 above. This result was produced by query 7b in Appendix I.

NOTE: We only show the first 10 rows of the table in Figure 12 that the query produces. There are actually 702 entries in the table, but we found that it was not suitable to show all 702 rows. Therefore, we have not included all entries in the report. The table is sorted on user ID and show entries with transportation mode before entries without transportation mode. All rows can be found in the results folder in the code files.

2.2.8 Number of users which have been close to each other in time and space

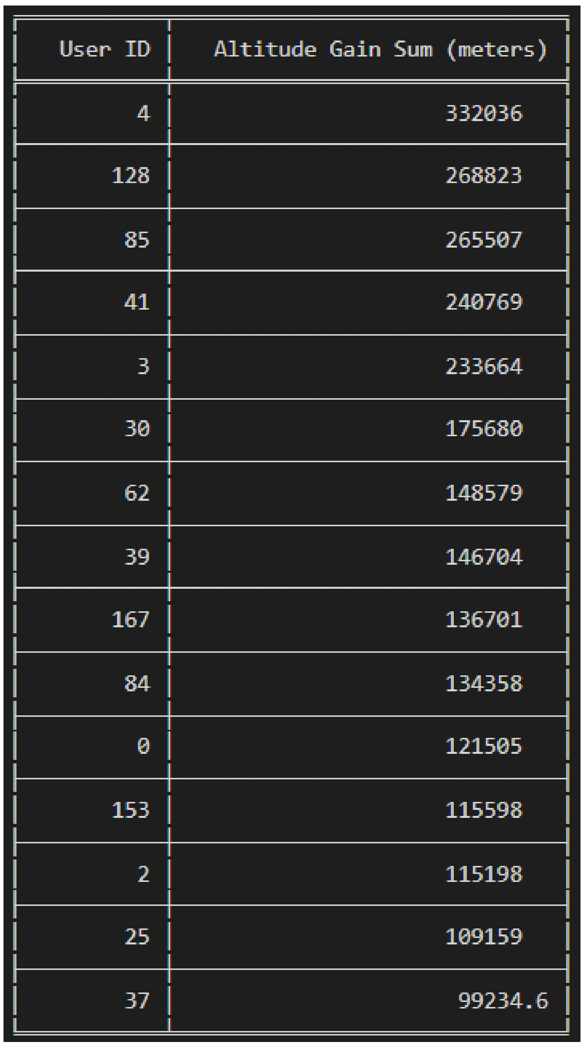


```
Number of distinct close users: 107
```

Figure 13: Result from SQL query for task 8

The SQL query gives the results shown in Figure 13 above. There are 107 users which have been close to each other in time and space. This result was produced by query 8 in Appendix J.

2.2.9 Top 15 users who have gained the most altitude metres

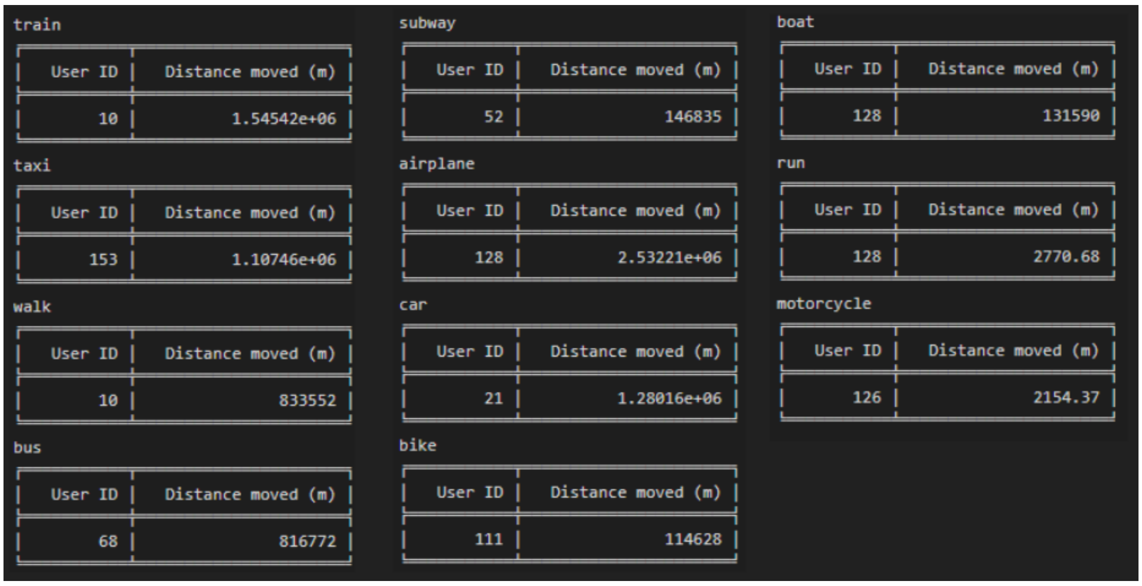


```
| User ID | Altitude Gain Sum (meters) |
|---------|----------------------------|
|       4 |                     332036 |
|     128 |                     268823 |
|      85 |                     265507 |
|      41 |                     240769 |
|       3 |                     233664 |
|      30 |                     175680 |
|      62 |                     148579 |
|      39 |                     146704 |
|     167 |                     136701 |
|      84 |                     134358 |
|       0 |                     121505 |
|     153 |                     115598 |
|       2 |                     115198 |
|      25 |                     109159 |
|      37 |                       99234.6 |
```

Figure 14: Result from SQL query for task 9

The SQL query gives the results shown in Figure 14 above. The top 15 users that have gained the most altitude are users with id 4, 128, 85, 41, 3, 30, 62, 39, 167, 84, 0, 153, 2, 25 and 37 in descending order respectively. This result was produced by query 9 in Appendix K.

2.2.10 Users that have travelled the longest total distance in one day for each transportation mode



```
train
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│        10 │           1.54542e+06 │
└───────────┴───────────────────────┘
taxi
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       153 │           1.10746e+06 │
└───────────┴───────────────────────┘
walk
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│        10 │                833552 │
└───────────┴───────────────────────┘
bus
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│        68 │                816772 │
└───────────┴───────────────────────┘
```

```
subway
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│        52 │                146835 │
└───────────┴───────────────────────┘
airplane
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       128 │           2.53221e+06 │
└───────────┴───────────────────────┘
car
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│        21 │           1.28016e+06 │
└───────────┴───────────────────────┘
bike
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       111 │                114628 │
└───────────┴───────────────────────┘
```

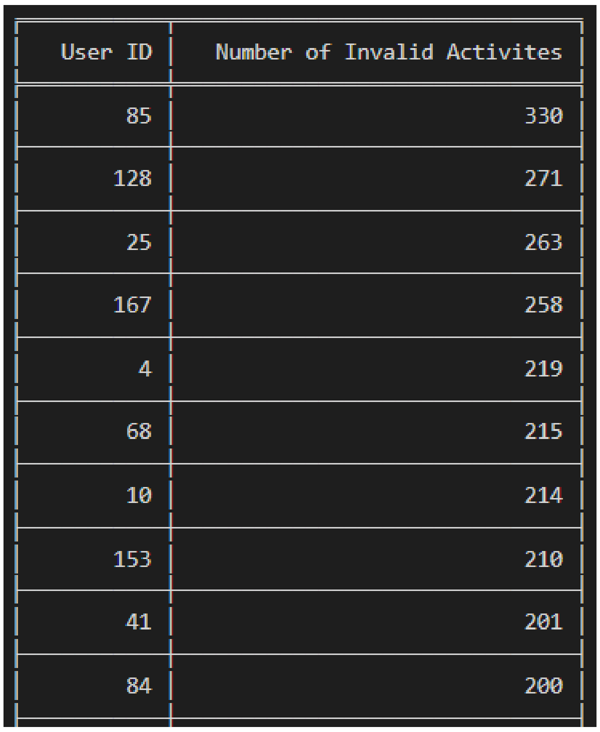
```
boat
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       128 │                131590 │
└───────────┴───────────────────────┘
run
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       128 │               2770.68 │
└───────────┴───────────────────────┘
motorcycle
┌───────────┬───────────────────────┐
│   User ID │   Distance moved (m)  │
├───────────┼───────────────────────┤
│       126 │               2154.37 │
└───────────┴───────────────────────┘
```

Figure 15: Result from SQL query for task 10

The SQL query gives the results shown in Figure 15 above. This result was produced by query 10 in Appendix L. The users that have traveled the longest distance in one day for each transportation mode is:

- Train: User with ID 10, which have moved $\sim$ 1.5 million meters
- Taxi: User with ID 153, which have moved $\sim$ 1.11 million meters
- Walk: User with ID 10, which have moved 833 552 meters
- Bus: User with ID 68, which have moved 816 772 meters
- Subway: User with ID 52, which have moved 146 853 meters
- Airplane: User with ID 128, which have moved $\sim$ 2.53 million meters
- Car: User with ID 21, which have moved $\sim$ 1.28 million meters
- Bike: User with ID 111, which have moved 114 628 meters
- Boat: User with ID 128, which have moved 131 590 meters
- Run: User with ID 128, which have moved 2770.68 meters
- Motorcycle: User with ID 126, which have moved 2154.37 meters

2.2.11 All users with invalid activities, and the number of invalid activities per user



```
| User ID | Number of Invalid Activites |
|      85 |                         330 |
|     128 |                         271 |
|      25 |                         263 |
|     167 |                         258 |
|       4 |                         219 |
|      68 |                         215 |
|      10 |                         214 |
|     153 |                         210 |
|      41 |                         201 |
|      84 |                         200 |
```

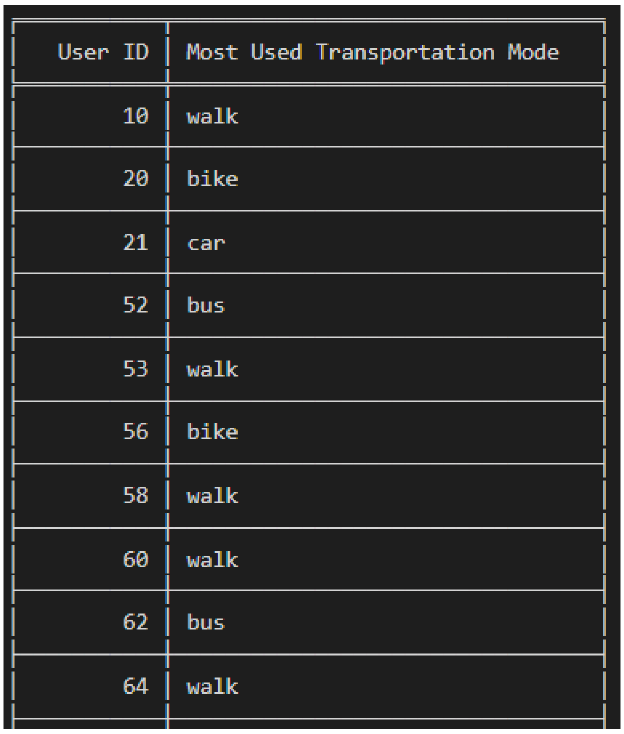
Figure 16: Result from SQL query for task 11

The SQL query gives the results shown in Figure 16 above. This result was produced by query 11 in Appendix M. The results shows the first 10 results in the table sorted by the number of invalid activities, but there are 165 users in total. The top 10 users with invalid activities and the number of invalid activity per user respectively are:

- User with ID 85, with 330 invalid activities
- User with ID 128, with 271 invalid activities
- User with ID 25, with 263 invalid activities
- User with ID 167, with 258 invalid activities
- User with ID 4, with 219 invalid activities
- User with ID 68, with 215 invalid activities
- User with ID 10, with 214 invalid activities
- User with ID 153, with 210 invalid activities
- User with ID 41, with 201 invalid activities
- User with ID 84, with 200 invalid activities

NOTE: We only show the first 10 rows of the table in Figure 16 that the query produces. There are actually 165 entries in the table, but we found that it was not suitable to show all 165 rows. Therefore, we have not included all entries in the result. The table is sorted descendingly on the number of invalid activities. All rows can be found in the results folder in the code files.

2.2.12 Users with registered transportation mode and their most used transportation mode



```
| User ID | Most Used Transportation Mode |
|      10 | walk                          |
|      20 | bike                          |
|      21 | car                           |
|      52 | bus                           |
|      53 | walk                          |
|      56 | bike                          |
|      58 | walk                          |
|      60 | walk                          |
|      62 | bus                           |
|      64 | walk                          |
```

Figure 17: Result from SQL query for task 12

The SQL query gives the results shown in Figure 17 above. This result was produced by query 12 in Appendix N. The figure shows only the first 10 results sorted by User ID, but there were a total of 64 users in the results. Top 10 Users with registered transportation mode and their most used transportation mode is:

- User with ID 10, with "walk" as the most used transportation mode
- User with ID 20, with "bike" as the most used transportation mode
- User with ID 21, with "car" as the most used transportation mode
- User with ID 52, with "bus" as the most used transportation mode
- User with ID 53, with "walk" as the most used transportation mode
- User with ID 56, with "bike" as the most used transportation mode
- User with ID 58, with "walk" as the most used transportation mode
- User with ID 60, with "walk" as the most used transportation mode
- User with ID 62, with "bus" as the most used transportation mode
- User with ID 64, with "walk" as the most used transportation mode

NOTE: We only show the first 10 rows of the table in Figure 17 that the query produces. There are actually 64 entries in the table, but we found that it was not suitable to show all 64 rows. Therefore, we have not included all entries in the result. The table is sorted on user ID. All rows can be found in the results folder in the code files.

3 Discussion

3.1 Part 1

Decision to include both activities with and without labels

As described in the assumptions part above, we assumed that for users with labeled activities one trajectory file could contain several activities, and for non-labeled users there was only one activity per file. There were several reasons why we did this. In the beginning we thought that there could only be one activity per trajectory file, as the assignment sheet stated this. However, after inspection of the data we realized that this was not the case by looking at some of the users with several labeled activities on the same date. We then decided for the labeled users that we could find more than one activity in one trajectory file. This was a good solution because it meant that we could find more matches for labeled activities, and dropping less activities than if we treated one file as an activity. It was however, a much more complex solution then to just treat every trajectory file as one activity.

Moving on, we decided to also include the users that had not labeled their activities and their trackpoints in our implementation. The reason is that we felt it was unatural to exclude these users and their activities just because they were not labeled. After all, they actually had done something on the times of their TrackPoints. The upside of this is that we got a more comprehensive database with more Users, Activities and TrackPoints. The downside of this solution is that it leads to more invalid activities in the database. Since there is no labels to these user's activities, we had to treat every trajectory file as an activity. This could mean that some of the activities we then treated as one could actually contain several activities as explained above, and contain trackpoints with huge time gaps. Anyways, we thought it was best to include all users in our solution.

Solution

Inserting the data into the database was relatively straightforward once we had decided how the datacleaning process should be, and we did not have any major issues. Our process was as follows:

1. Insert User objects based on the name of the subdirectories in the dataset.
2. Creating the Activity objects for the users that had labeled their activities based on their labels.
3. Insert TrackPoint data iterating over the *.plt*-files for each user. For users without labeled activities: create one Activity for every trajectory file, and bulk create the TrackPoints in that file and connect them to that Activity. For users with labels: find correct Activity-object in the database for every TrackPoint based on time, and ignore the TrackPoint if it does not match any Activity. We dropped Activities with more than 2500 TrackPoints.
4. Add relevant indexes based on what we were going to query and modifying some columns. For example converting altitude to meters, and replacing invalid altitudes of -777 with NaN.
5. Dropping empty activities, i.e. dropping labeled activities that could not be perfectly matched with any TrackPoints.

This was relatively performant, so we did not see any reason to optimize it further. The reason we decided to add more columns than in the suggested-database scheme will be discussed for the relevant queries in part 2.

Challenges

One challenge we faced was the need to preprocess data to ensure our queries ran quickly. As a result, we had to rebuild our database multiple times, which was time-consuming. In retrospect, if we had planned our queries before determining the database columns, we might have avoided this problem.

3.2 Part 2

For the first queries in part 2, SQL alone was sufficient to effectively get the results wanted and no modification to the suggested database scheme was needed. Specifically the solutions to queries 1-7 and 12 is done solely through SQL queries, and uses only the variables in the suggested database-scheme in the assignment text. However, when reaching exercise 8 and beyond, there was a significant increase in computation time needed to get the wanted results using SQL only. Therefore, several columns were added to the database table on insert to make computation less time-expensive, as explained above. We decided to do this instead of further dataprocessing in python after the SQL queries, as it was the most effective for time optimization. Listed below are the changes made related to each query and why.

Query 8:

To efficiently solve this query, we decided to add a new column to the database, called *euclidian_radial*, that allowed us to iterate over the TrackPoints in pages at runtime, ensuring that all other TrackPoints that potentially could be close enough in time and space to meet the requirements were in the page.

The way we defined the *euclidian_radial* column was by creating a two-dimensional Euclidian space, where time was one dimension, and distance from the centre of the world was another dimension. We scaled the dimensions, so that one unit of time corresponded to the time limit of 30 seconds given in the task, and one unit of distance corresponded to the distance limit of 50 meters. Mathematically, the calculation we used to calculate the *euclidian_radial* column was:

$$\mathit{eucledian\_radial} = \sqrt{d^2 + t^2}$$

where

- t is the time dimension as described above (units of 30 seconds, and point 0 being the same as point 0 for the date_days column)
- d is the distance from the centre of the world, measured in units of 50 meters

We found it appropriate to also account for altitude differences, as the task asked for users close in "space and time", and space is 3-dimensional. This also makes sense, since some users have tracked e.g. airplane and subway travel, so they would not have met even in the case that they were on the same longitude and latitude coordinates at the same time. Therefore, our definition of d was:

$$d = \frac{\sqrt{d_h^2 + a^2}}{50}$$

where

- d_h is the distance from the centre of the world measured in metres, calculated using the haversine formula on the longitude and latitude coordinates
- a is the altitude, measured in metres

At runtime, we could then query the TrackPoint table in chunks (pages), ordering by the *euclidian_radial* column. To ensure that this was performant, we added an index for the column.

Based on the definitions above, we knew that two TrackPoints could only meet the intersection criteria if they had *euclidian_radial* values that differed by less than or exactly $\sqrt{2}$. Thus, our query was simply

```
SELECT user_id, activity_id, lat, lon, date_days, altitude,
↪  euclidian_radial
FROM TrackPoint
LEFT JOIN Activity ON TrackPoint.activity_id = Activity.id
WHERE euclidian_radial > {min_euclidian_radial}
ORDER BY euclidian_radial
LIMIT 100000
```

Where we in each iteration updated *min_euclidian_radial* to be the maximum of the *euclidian_radial* column in the iteration, minus $\sqrt{2}$ in case some of the last points in the previous query intersected with some of the first in the next.

Using the returned data, we applied the DBSCAN-algorithm from scikit-learn to find clusters of TrackPoints that fit the intersection criteria in the task (less than 50 metres apart in a timeframe of 30 seconds). Using the results of the DBSCAN-algorithm, we looked at all the clusters with more than one user ID, and stored those user IDs in the set.

Iterating over the whole database took around one minute, and after all iterations were complete we had a set of 107 IDs for the users who at one point met another user.

Query 9:

In order to easily calculate the altitude gain of different users, we decided to add a column to the TrackPoint table. This column called *altitude_diff* was the altitude difference between two consecutive TrackPoints, and allowed us to easily calculate the altitude gain for each user by summing each user's positive altitude differences. Then we could select the 15 users with the most altitude gain. This solution addressed the pain point of having to compare trackpoints directly in SQL two at a time for each user, which was harder to do and more time expensive. Now we could iterate in linear time.

Query 10:

For query 10 we decided to do the exact same as for query 9; adding a new column *meters_moved* for each TrackPoint, being the metres moved since the last TrackPoint in that activity. Doing this on insertion instead of in the query again allowed us to iterate in linear time when doing the query, instead of having to compare two and two TrackPoints. This was done using the haversine function to calculate the meter distance between the latitude, longitude coordinates of the two consecutive TrackPoints.

Initially when we looked at the results, we thought the distances traveled was very long, maybe too long to make sense. However, we did a sanity check that confirmed that our results could make sense. For example, if we look at transportation mode "train" where user 10 have moved ~ 1.5 million meters. Inspecting the data, we found a day where user 10 traveled by train for 24 hours. That means if user 10 traveled for 24 hours and moved a total distance of ~ 1.5 million meters, the user must have traveled $1\,545\,420/(3600*24) = 17.88 m/s * 3.6 =\sim 64 km/h$. Traveling at that speed on average by train sounds reasonable.

For other activities like "walk", traveling 833 km in one day would mean traveling at a speed of $\sim 35 km/h$ for 24 hours which does not sound reasonable. On the other hand, the result could still make sense if for example the activity was wrongly marked with another transportation mode which is very easy and common to do in apps like Strava.

Query 11:

For query 11 we again decided to add a column in the TrackPoint table for simplicity when querying the database. An invalid activity is defined as an activity having consecutive TrackPoints with at least 5 minutes difference in the timestamps. In order to find these activities in the query, we would again have to compare all consecutive TrackPoints for the activities, being time consuming in the query. Therefore, we decided to add the column *minutes_diff* on insertion, being minute difference between consecutive TrackPoint timestamps. Thus in our query, we only had to find activities that had a TrackPoint where this column was bigger than 5 in order to identify an invalid activity, a query that was really simple to perform.

From the results in query 11 we can see that their are 165 users in total that has invalid activities, and that some users have quite a lot of invalid activities. This could be because of invalid activities coming from the activities generated from the users without labeled activities. However, upon further inspection we can see that 7/10 of the users with the most invalid activities are users with labeled activities. Thus, we can not conclude that including the unlabeled users on a general level raised the percentage of invalid activities. In fact, this find supports the decision to add the unlabeled users, as they don't seem to be the ones with the most invalid activities at all.

3.3 What we learned

In summary there are 2 main things that we have learned during this assignment:

1. Big queries can be slow in a large database
When having a database with so many data points, it is really important to design efficient queries. If the queries are inefficient and causes a lot of joins or loops, the computation time can be huge, something we experienced especially in query 8.

2. Understanding the purpose of an application is really important in database design
Since inefficient queries can be slow when having so much data, it is really important to design a database with the purpose of the application in mind. Although this is not always easy or even possible, understanding the use-areas of an application, what queries will be made and what information that would likely be retrieved is therefore crucial.

In our case the assignment was not an application, but as mentioned earlier it would have been helpful to design the queries before the database creation, so that we knew what information we would like to have in our scheme to effectively answer some of the queries. As redeployment of a database, database scheme changes and data insertion is time consuming, knowing exactly what you need when starting to design a database is important. With an industry perspective, we have therefore learned the importance of application-oriented database design for companies deploying live applications and systems.

4 Feedback

In general we liked the that on an overall basis there were clear problem descriptions and that the setup guide for the project was easy to follow. With regards to the exercises they were interesting with a good mix of easy and though queries, were most also was easy to interpret and understand.

With query 8, we were a bit unsure what "space" meant. It could perhaps have been specified if we should look at only latitude and longitude, or if altitude should be included. Also, we felt (maybe because of our interpretation) that task 8 was a lot tougher than the others, and definitely a notch up from number 7. Maybe it could have been the last one in the list and a bit clearer in the description?

Lastly we felt that the description and definition of the trajectory files and their relation to activities could have been a bit clearer. The assignment sheet says that we should only insert activities that have fewer than or exactly 2500 TrackPoints, and that we could do so by checking that the size of the *.plt*-files do not exceed 2500 lines. Also, in the tips section it says that start_time and end_time for Activities can be found by looking at the date and time for the first and last TrackPoint in each *.plt*-file. At first this led us to believe that the *.plt*-files only belonged to one activity, so that is what we did when initially setting up the project. However, as we later saw, there could be TrackPoints for several activities in one trajectory file for users with labeled activities. Therefore this information is contradictory, and a file might contain more than 2500 TrackPoints in total, but since those could be distributed over several activities the file should not be dropped for that reason. Although we understand that to interpret this is all part of the datacleaning process, it could perhaps have been specified more.

Appendix

A SQL query for creating database

```
# Crate tables
CREATE TABLE IF NOT EXISTS User (
    id INT NOT NULL PRIMARY KEY,
    has_labels boolean
)

CREATE TABLE IF NOT EXISTS Activity (
    id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    user_id INT NOT NULL,
    transportation_mode VARCHAR(30),
    start_date_time DATETIME NOT NULL,
    end_date_time DATETIME NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(id)
)

CREATE TABLE IF NOT EXISTS TrackPoint (
    id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    activity_id INT NOT NULL,
    lat DOUBLE NOT NULL,
    lon DOUBLE NOT NULL,
    altitude DOUBLE,
    altitude_diff DOUBLE,
    meters_moved DOUBLE,
    minutes_diff DOUBLE,
    date_days DOUBLE NOT NULL,
    date_time DATETIME NOT NULL,
    euclidian_radial DOUBLE NOT NULL,
    FOREIGN KEY (activity_id) REFERENCES Activity(id)
)



# Sets relevant indices on the database
ALTER TABLE TrackPoint ADD INDEX idx_trackpoint_activity_date (activity_id,
↪  date_days)

ALTER TABLE TrackPoint ADD INDEX idx_trackpoint_euclidian_radial
↪  (euclidian_radial)



# Bulk insert into User, Activity and TrackPoint table
INSERT INTO User (id, has_labels) VALUES (%s, %s)

INSERT INTO Activity (user_id, transportation_mode, start_date_time,
↪  end_date_time) VALUES (%s, %s, %s, %s)

INSERT INTO TrackPoint (activity_id, lat, lon, altitude, altitude_diff,
↪  meters_moved, minutes_diff, date_days, date_time, euclidian_radial)
↪  VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
```

B Query 1

```
SELECT
 (SELECT COUNT(*) FROM User) AS user_count,
 (SELECT COUNT(*) FROM Activity) AS activity_count,
 (SELECT COUNT(*) FROM TrackPoint) AS trackpoint_count
```

C Query 2

```
SELECT
    AVG(TrackPointCount) AS Average,
    MAX(TrackPointCount) AS Maximum,
    MIN(TrackPointCount) as Minimum
FROM (
    SELECT User.id, SUM(TrackPoints) AS TrackPointCount
    FROM User INNER JOIN (
        SELECT Activity.user_id, Activity.id, COUNT(TrackPoint.id) AS TrackPoints
        FROM Activity INNER JOIN TrackPoint ON Activity.id =
        ↪ TrackPoint.activity_id
        GROUP BY Activity.user_id, Activity.id
    ) AS UserActTrackSub ON User.id = UserActTrackSub.user_id
    GROUP BY User.id
) AS JoinedTable
```

D Query 3

```
SELECT User.id AS user_id, COUNT(Activity.id) AS activity_count
FROM User LEFT JOIN Activity ON User.id = Activity.user_id
GROUP BY User.id
ORDER BY activity_count DESC
LIMIT 15;
```

E Query 4

```
SELECT DISTINCT User.id
FROM User LEFT JOIN Activity ON User.id = Activity.user_id
WHERE Activity.transportation_mode = 'bus'
```

F Query 5

```
SELECT
    user_id,
    COUNT(DISTINCT Activity.transportation_mode) AS num_different_transports
FROM Activity
GROUP BY user_id
ORDER BY num_different_transports DESC
LIMIT 10
```

G Query 6

```
SELECT user_id, transportation_mode, start_date_time, end_date_time
FROM Activity
```

```
GROUP BY user_id, transportation_mode, start_date_time, end_date_time
HAVING COUNT(*) > 1;
```

H Query 7a

```
SELECT COUNT(DISTINCT user_id)
FROM Activity
WHERE DATEDIFF(end_date_time, start_date_time) = 1;
```

I Query 7b

```
SELECT user_id, transportation_mode, TIMEDIFF(end_date_time, start_date_time) AS
↪  duration, start_date_time, end_date_time
FROM Activity
WHERE DATEDIFF(end_date_time, start_date_time) = 1;
```

J Query 8

```
SELECT user_id, activity_id, lat, lon, date_days, altitude, euclidian_radial
FROM TrackPoint
LEFT JOIN Activity ON TrackPoint.activity_id = Activity.id
WHERE euclidian_radial > {min_euclidian_radial}
ORDER BY euclidian_radial
LIMIT {POINTS_PER_QUERY}
```

K Query 9

```
SELECT
    a.user_id,
    SUM(
        CASE
            WHEN t.altitude_diff > 0 THEN altitude_diff
            ELSE 0
        END
    ) as alt_gain_sum
FROM
    TrackPoint t
JOIN
    Activity a ON a.id = t.activity_id
GROUP BY
    a.user_id
ORDER BY alt_gain_sum DESC
LIMIT 15
```

L Query 10

```
SELECT Activity.user_id AS user_id, SUM(meters_moved) AS total_distance
FROM TrackPoint INNER JOIN Activity ON Activity.id = TrackPoint.activity_id
WHERE Activity.transportation_mode = '{transportation_mode}'
GROUP BY user_id, DATE(TrackPoint.date_time)
ORDER BY total_distance DESC
LIMIT 1;
```

M Query 11

```
SELECT user_id, COUNT(DISTINCT Activity.id) as num_invalid_activites
FROM Activity LEFT JOIN TrackPoint ON Activity.id = TrackPoint.activity_id
WHERE TrackPoint.minutes_diff >= 5
GROUP BY user_id
ORDER BY num_invalid_activites DESC
```

N Query 12

```
SELECT user_id,
    (
    SELECT transportation_mode
    FROM Activity AS sub_a
    WHERE transportation_mode IS NOT NULL AND a.user_id = sub_a.user_id
    GROUP BY transportation_mode
    ORDER BY COUNT(*) DESC
    LIMIT 1
    ) AS most_used_transportation_mode
FROM Activity AS a
WHERE transportation_mode IS NOT NULL
GROUP BY user_id
ORDER BY user_id
```