TDT4225 - Assignment 2

Group 26
Ole Jonas Liahagen
Martin Johannes Nilsen

8th October 2021

Introduction

In this assignment we were to insert and process data from the Geolife GPS Trajectory dataset. The data given to us consisted of 182 unique users, each with their own registered activites. These activities each consists of a list of tracking points containing GPS coordinates and timestamps. Activities could either be labeled with a transportation mode or not at all. To solve the assigment, we have used mainly database queries to get the data desired and on some occasions structured and filtered these data further using python if necessary.

First we started off creating a GitHub[1] repository, and making a template for the report. Then we proceeded to filter and insert data into the database with pair programming ensuring that both of us would agree on the implementation. After we had built the database structure and inserted data, we split up and solved the assignments separately, with the possibility to ask each other questions and share some thoughts to check that we agreed on the solutions. Lastly, the report was written using Overleaf to collaboratively write the LaTeX report.

[1]https://github.com/MartinJohannesNilsen/tdt4225-assignment2.git

Results

Part 1

id	has_labels
000	0
001	0
002	0
003	0
004	0
005	0
006	0
007	0
008	0
009	0
010	1

Top rows of Users inserted to db.

id	user_id	transportation_mode	start_date_time	end_date_time
0	000	NULL	2008-10-23 02:53:04	2008-10-23 11:11:12
1	000	NULL	2008-10-24 02:09:59	2008-10-24 02:47:06
2	000	NULL	2008-10-26 13:44:07	2008-10-26 15:04:07
3	000	NULL	2008-10-27 11:54:49	2008-10-27 12:05:54
4	000	NULL	2008-10-28 00:38:26	2008-10-28 05:03:42
5	000	NULL	2008-10-29 09:21:38	2008-10-29 09:30:28
6	000	NULL	2008-10-29 09:30:38	2008-10-29 09:46:43
7	000	NULL	2008-11-03 10:13:36	2008-11-03 10:16:01
8	000	NULL	2008-11-03 23:21:53	2008-11-04 03:31:08
9	000	NULL	2008-11-10 01:36:37	2008-11-10 03:46:12
10	000	NULL	2008-11-11 00:17:04	2008-11-11 02:35:54

Top rows of Activities inserted to db.

id	activity_id	latitude	longitude	altitude	date_days	date_time
1	0	39.984702	116.318417	492	39744.1201851852	2008-10-23 02:53:04
2	0	39.984683	116.31845	492	39744.1202546296	2008-10-23 02:53:10
3	0	39.984686	116.318417	492	39744.1203125	2008-10-23 02:53:15
4	0	39.984688	116.318385	492	39744.1203703704	2008-10-23 02:53:20
5	0	39.984655	116.318263	492	39744.1204282407	2008-10-23 02:53:25
6	0	39.984611	116.318026	493	39744.1204861111	2008-10-23 02:53:30
7	0	39.984608	116.317761	493	39744.1205439815	2008-10-23 02:53:35
8	0	39.984563	116.317517	496	39744.1206018519	2008-10-23 02:53:40
9	0	39.984539	116.317294	500	39744.1206597222	2008-10-23 02:53:45
10	0	39.984606	116.317065	505	39744.1207175926	2008-10-23 02:53:50

Top rows of TrackPoints inserted to db.

Part 2

Task 1

The amount of users, activities and trackpoints in the dataset after inserted to the database:

Task 1

n_users	n_activities	n_trackpoints
182	16048	9681756

Results for task 1.

Task 2

The *average*, *minimum* and *maximum* number of activities per user. It is important to notice that some users have tracked activities, but because we exclude all activities with more than 2500 trackpoints, some users end up having zero activities.

average	maximum	minimum
88.1758	2102	0

Results for task 2.

Task 3

Top 10 users with the highest number of activities:

uid	n_activities
128	2102
153	1793
025	715
163	704
062	691
144	563
041	399
085	364
004	346
140	345

Results for task 3.

Task 4

The number of users that have started the activity in one day and ended the activity the next day:

```
Task 4
  Users with multi-day activities
---------------------------------
                               98
```

Results for task 4.

Task 5

Activities that are registered multiple times, i.e. the same start- and end-time, but different ids. The dataset includes zero duplicated activities.

```
Task 5
user    id_1    id_2
------  ------  ------
```

Results for task 5.

Task 6

This task asked for a sort of proximity tracking of the users of the application. We were tasked with finding all users that had been within 100 meters of each other in a time window of 60 seconds. This makes it necessary to do a lot of comparisons which then lead to a very long execution time. We have tried different methods of comparing the different trackpoints, but they all took too long. We will represent and explain these methods in the discussion part of the report.

Task 7

We were in this task requested to find all users that have never taken a taxi. The results includes column *has_index* as this indicates which of the users who have labeled their data but not taken a taxi. The result has 172 rows, which indicates that only 10 users have taken a taxi and labeled it. Out of the 172, 60 users have labeled their data (the rest we can't know if have taken a taxi or not).

```
Task 7
172
  id    has_labels
----  ------------
 000             0
 001             0
 002             0
 003             0
 004             0
 005             0
 006             0
 007             0
 008             0
 009             0
 011             0
 012             0
 013             0
 014             0
 015             0
 016             0
 017             0
 018             0
 019             0
 020             1
 021             1
 022             0
 023             0
 024             0
 025             0
 026             0
 027             0
 028             0
 029             0
 030             0
 031             0
 032             0
 033             0
 034             0
 035             0
 036             0
 037             0
 038             0
 039             0
 040             0
 041             0
 042             0
 043             0
 044             0
 045             0
 046             0
 047             0
 048             0
 049             0
 050             0
 051             0
 052             1
 053             1
```

Figure 1: Results for task 7.

054	0
055	0
056	1
057	0
059	1
060	1
061	0
063	0
064	1
065	1
066	0
067	1
068	1
069	1
070	0
071	0
072	0
073	1
074	0
075	1
076	1
077	0
079	0
081	1
082	1
083	0
084	1
086	1
087	1
088	1
089	1
090	0
091	1
092	1
093	0
094	0
095	0
096	1
097	1
099	0
100	1
101	1
102	1
103	0
104	1
105	1
106	1
107	1
108	1
109	0
110	1
112	1
113	0
114	1
115	1
116	1
117	1
118	1

Results for task 7.

118	1
119	0
120	0
121	0
122	0
123	0
124	1
125	1
126	1
127	0
129	1
130	0
131	0
132	0
133	0
134	0
135	0
136	1
137	0
138	1
139	1
140	0
141	1
142	0
143	0
144	1
145	0
146	0
147	1
148	0
149	0
150	0
151	0
152	0
153	1
154	1
155	0
156	0
157	0
158	0
159	0
160	0
161	1
162	0
164	0
165	0
166	0
167	1
168	0
169	0
170	1
171	0
172	0
173	0
174	1
175	1
176	0
177	0
178	0
179	1
180	0
181	0

Results for task 7.

Task 8

All types of transportation modes and how many distinct users that have used them:

Task 8

transportation_mode	unique_uids
airplane	1
bike	19
boat	1
bus	12
car	8
run	1
subway	4
taxi	10
train	2
walk	31

Results for task 8.

Task 9

a) The year and month with the most activities:

Task 9a

Year with most activities	Month with most activities in this year
2008	November

Results for task 9a.

b) The two users with the most activities this year and month, and how many recorded hours they have. The user with second most activities has indeed more hours than the user with the most activities.

Task 9b

uid	n_activities	hours
062	130	47.3136
128	75	68.2211

Results for task 9b.

Task 10

The total distance (in km) walked in 2008, by user with id=112. This distance seems small, but we think it is correct after going over our code and queries. Looking over the trackpoints associated with user 112, it seems they have not moved much around by foot this year. This might be due to our method of labeling the data, so that certain walks were not accepted by our filtering. However, this is only speculation.

```
Task 10
User_id: 112
Distance in 2008: 1.3497848643311852
```

Results for task 10.

Task 11

The top 20 users who have gained the most altitude meters:

```
Task 11
  uid    m gained
-----  ----------
  128     2135455
  153     1820766
    4     1089358
   41      789890
    3      766613
   85      714049
  163      673439
   62      596103
  144      588771
   30      576377
   39      481311
   84      430319
    0      398638
    2      377503
  167      370647
   25      358098
   37      325528
  140      311151
  126      272389
   17      205270
```

Results for task 11.

Task 12

In the figures below you will find the users who have invalid activities, and the number of invalid activities per user. An invalid activity is when the distance in time between two trackpoints is 5 minutes or more.

uid	Invalid activities
0	445
1	116
2	296
3	849
4	1173
5	127
6	49
7	136
8	33
9	114
10	154
11	38
12	163
13	69
14	328
15	138
16	61
17	493
18	97
19	65
20	27
21	72
22	191
23	30
24	75
25	426
26	51
27	14
28	91
29	37
30	511
31	7
32	37
33	3
34	241
35	124
36	126
37	279
38	266
39	427
40	52
41	389
42	118
43	73
44	83
45	19
46	26
47	19
48	5
49	0
50	17
51	146
52	138
53	39
54	8
55	28

Results for task 12.

56	11
57	60
58	25
59	8
60	1
61	41
62	397
63	27
64	10
65	46
66	40
67	83
68	327
69	65
70	16
71	64
72	6
73	30
74	19
75	8
76	33
77	11
78	26
79	3
80	8
81	42
82	65
83	56
84	230
85	384
86	19
87	3
88	13
89	116
90	15
91	110
92	315
93	4
94	60
95	20
96	67
97	27
98	25
99	113
100	5
101	166
102	16
103	44
104	351
105	35
106	66
107	2
108	28
109	9
110	48
111	251
112	140
113	1

Results for task 12.

113	1
114	4
115	183
116	0
117	3
118	31
119	79
120	0
121	15
122	21
123	21
124	12
125	69
126	242
127	8
128	1288
129	41
130	18
131	59
132	3
133	18
134	83
135	21
136	34
137	0
138	27
139	50
140	195
141	1
142	200
143	0
144	551
145	17
146	28
147	64
148	0
149	0
150	60
151	4
152	4
153	1154
154	28
155	70
156	0
157	65
158	28
159	7
160	0
161	7
162	27
163	559
164	20
165	4
166	4
167	334
168	36
169	12
170	2
171	14
172	21
173	11
174	188
175	20
176	24
177	0
178	0
179	42
180	2
181	49

Results for task 12.

Discussion

During this assignment, we attempted to do most of the tasks doing as much of the work with MySQL as possible, however, some of the tasks were made a lot easier by the use of some extra sorting and filtering after the data was retrieved. This strategy worked fine for all the tasks except one.

As stated in the results portion of the report, task 6 wanted us to do some sort of proximity tracking between all users. Seeing as we had inserted ca. 9 million trackpoints into the database, comparing all these could take quite some time. And so it did. Our first attempt used the cartesian product between the trackpoints table and itself where the timestamps did not differ by more than 60 seconds. This query in itself is very naive, due to the fact that the table is very big on its own. Executing the query would yield around 80 trillion rows which, understandably, is a very large amount of data to sift through. After some more testing our best attempt tried to avoid the cartesian product, and instead do comparisons on batches of data. To do this we read data from the database and inserted each row into hashmaps/dictionaries based on the day and time of day they were recorded. Doing this would drastically reduce the amount of comparisons needed since each trackpoint would now only need to be compared with their respective batch of trackpoints in the same list of trackpoints. From running the program, it seems the amount of points in a given array is around the order of 10000, meaning we go from doing some 9 million comparisons per point to around 10000 give or take. Unfortunately this approach requires a lot of memory, and our program did not finish due to this hindrance. This means there could be more points per row than mentioned, but these results are what we have gathered from the data we could extract. A possible solution here would be to do requests on batches of data from the database, so that we did not need to keep everything in memory at once. Another further improvement could be the implementation of multiprocessing and parallellization to speed up the data processing. The way we are computing the results seem highly parallellisable, leading to this suggestion. We unfortunately did not have the time to implement these improvements, but believe that the task would be solved much faster with the use of them.

In task 9, we were tasked to find the year and month with the most activities. The question here is whether or not the month should be dependent on the year with the most activities, or if it is the month with more activities overall. At first, we implemented a query for getting the independent most active year and month, but for task 9b we assumed that it would make more sense to have the month based on the most active year. The month with the most activities is May, but in year 2008 it is November. In the code there is possibilites for both, with the first choice commented out. The second important element we came accross in task 9, is how we count difference in time in SQL. With the use of `TIMEDIFF` in hours, the query takes the floor of all whole hours, making the result quite off. The solution was to use `TIMESTAMPDIFF` in seconds, dividing by 3600 and summing all the rows together, giving us a more accurate answer.

Overall the assignment went relatively pain free, with the exception of task 6. Here we learned a valuable lesson, and that is to filter the data as much as possible before doing computations on them. One could also resort to batch processing of the data and make use of multiprocessing. If not, you could end up with massive amounts of data that will take unfeasible amounts of time to iterate through with our available resources.