Report TDT4225

Assignment 2 - MySQL

Group: 51
Students: Tore Apeland Fossland

Introduction

Assignment 2 in TDT4225 consists of solving some database tasks using a MySQL database and coding in python. The open source Geolife GPS Trajectory dataset is my data source, which is a recording of a broad range of user's outdoor activities. The first part of the assignment was setting up the database itself using docker a container. Then I defined and created suitable tables and relations, and lastly I cleaned and inserted the data. Secondly I made SQL queries to answer a set of given questions. Some code was also provided to serve as a starting point, which I ended up not using.

GitHub repo: https://github.com/ToreFossland/TDT4225

Results

data. First ten entries in the 3 tables users, activities and trackpoints.

users | Enter a SQL expression to filter results (use Ctrl+Space)

	id	has_labels
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	1

activities | Enter a SQL expression to filter results (use Ctrl+Space)

	id	user_id	transportation_mode	start_date_time	end_date_time
1	1	0	[NULL]	2009-05-10 02:02:06	2009-05-10 06:20:11
2	2	0	[NULL]	2008-12-01 11:18:27	2008-12-01 11:35:15
3	3	0	[NULL]	2009-06-21 10:14:07	2009-06-21 14:47:12
4	4	0	[NULL]	2008-12-11 12:14:32	2008-12-11 12:24:32
5	5	0	[NULL]	2009-06-29 01:16:30	2009-06-29 11:13:12
6	6	0	[NULL]	2009-05-09 18:16:36	2009-05-09 18:30:36
7	7	0	[NULL]	2009-07-03 00:28:00	2009-07-03 08:37:23
8	8	0	[NULL]	2009-06-07 07:52:55	2009-06-07 09:46:03
9	9	0	[NULL]	2009-06-11 22:12:04	2009-06-11 22:34:54
10	10	0	[NULL]	2009-04-26 17:55:13	2009-04-26 18:08:13

trackpoints | Enter a SQL expression to filter results (use Ctrl+Space)

	id	activity_id	lat	lon	altitude	date_days	date_time
1	1	1	40	116.327	80	39,915.301	2009-04-12 07:33:09
2	2	1	40	116.327	99	39,915.301	2009-04-12 07:33:14
3	3	1	40	116.327	109	39,915.301	2009-04-12 07:33:19
4	4	1	40	116.327	111	39,915.301	2009-04-12 07:33:24
5	5	1	40	116.327	114	39,915.301	2009-04-12 07:33:29
6	6	1	40	116.327	120	39,915.301	2009-04-12 07:33:34
7	7	1	40	116.327	125	39,915.301	2009-04-12 07:33:39
8	8	1	40	116.327	126	39,915.301	2009-04-12 07:33:44
9	9	1	40	116.327	127	39,915.301	2009-04-12 07:33:49
10	10	1	39.999	116.327	134	39,915.301	2009-04-12 07:33:54

Counts of users, activities and trackpoints in the dataset after filtering.

```
Fetch with plain query: 0.13 seconds.
Fetch with plain query: 0.04 seconds.
Fetch with plain query: 1.46 seconds.
User count: 181, activity count: 16047, trackpoint count: 9681755.
(sqlalchemy) torefossland@Tores-MacBook-Pro src %
```

Average number of activities per user.

```
Fetch with plain query: 0.09 seconds.
Fetch with plain query: 0.05 seconds.
Average number of activities per user: 88.
(sqlalchemy) torefossland@Tores-MacBook-Pro src %
```

The top 20 users with the highest number of activities by user id.

```
Fetch with plain query: 0.09 seconds.
User ID:  128  Activity Count:  2102
User ID:  153  Activity Count:  1793
User ID:  025  Activity Count:  715
User ID:  163  Activity Count:  704
User ID:  062  Activity Count:  691
User ID:  144  Activity Count:  563
User ID:  041  Activity Count:  399
User ID:  085  Activity Count:  364
User ID:  004  Activity Count:  346
User ID:  140  Activity Count:  345
User ID:  167  Activity Count:  320
User ID:  068  Activity Count:  280
User ID:  017  Activity Count:  265
User ID:  003  Activity Count:  261
User ID:  014  Activity Count:  236
User ID:  126  Activity Count:  215
User ID:  030  Activity Count:  210
User ID:  112  Activity Count:  208
User ID:  011  Activity Count:  201
User ID:  039  Activity Count:  198
```

the highest number of activities by user id.

```
Fetch with plain query: 0.06 seconds.
walk :  481
car :  419
bike :  262
bus :  199
subway :  133
taxi :  37
airplane :  3
train :  2
run :  1
boat :  1
```

All users who have taken a taxi.

```
Fetch with plain query: 0.05 seconds.
User ID:  010
User ID:  058
User ID:  062
User ID:  078
User ID:  080
User ID:  085
User ID:  098
User ID:  111
User ID:  128
User ID:  163
```

We first have the year with the most activities, and then the year with the most recorded hours. As you can see from the screenshot they are not the same.

```
Fetch with plain query: 0.06 seconds.
2008 :  5895
Fetch with plain query: 0.02 seconds.
2009 :  9156
```

Total distance walked in kilometers by user 112 in 2008.

```
Fetch with plain query: 2.53 seconds.
148.1599642945354 km
```

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

```
Fetch with plain query: 125.95 seconds.
128 :  790004.4144
153 :  680517.5112
004 :  357654.7584
041 :  290691.4176
062 :  273058.7376
144 :  271847.4624
003 :  249292.8720
163 :  248622.0072
085 :  240774.6264
030 :  184685.9400
039 :  161060.5872
025 :  157826.9640
084 :  145663.0056
140 :  138547.4496
167 :  131601.0576
000 :  129109.3176
002 :  128310.1320
037 :  117696.3864
126 :  105427.8816
034 :  102428.9544
```

All users who have invalid activities by descending count, only showing the top users because of space considerations.

```
Fetch with plain query: 123.03 seconds.
User id:  128  Invalid Activities:  719
User id:  153  Invalid Activities:  558
User id:  025  Invalid Activities:  264
User id:  062  Invalid Activities:  249
User id:  163  Invalid Activities:  234
User id:  004  Invalid Activities:  219
User id:  041  Invalid Activities:  201
User id:  085  Invalid Activities:  184
User id:  003  Invalid Activities:  179
User id:  144  Invalid Activities:  156
User id:  039  Invalid Activities:  147
User id:  068  Invalid Activities:  139
User id:  167  Invalid Activities:  133
User id:  017  Invalid Activities:  128
User id:  014  Invalid Activities:  117
User id:  030  Invalid Activities:  112
User id:  126  Invalid Activities:  105
User id:  037  Invalid Activities:  101
User id:  092  Invalid Activities:  101
User id:  000  Invalid Activities:  100
User id:  084  Invalid Activities:  99
User id:  002  Invalid Activities:  98
User id:  104  Invalid Activities:  96
User id:  034  Invalid Activities:  89
User id:  140  Invalid Activities:  86
User id:  112  Invalid Activities:  66
User id:  091  Invalid Activities:  62
```

The users who have tracked an activity in the Forbidden City of Beijing.

```
Fetch with plain query: 7.14 seconds.
User id:  004
User id:  018
User id:  019
User id:  131
```

All users who have registered transportation_mode and their most used transportation_mode.

```
Fetch with plain query: 0.12 seconds.
010 :  taxi
020 :  bike
021 :  walk
052 :  bus
056 :  bike
058 :  taxi
060 :  walk
062 :  bus
064 :  bike
065 :  bike
067 :  walk
069 :  bike
073 :  walk
075 :  walk
076 :  car
078 :  walk
080 :  taxi
081 :  bike
082 :  walk
084 :  walk
085 :  walk
086 :  car
087 :  walk
089 :  car
091 :  bus
092 :  bus
097 :  bike
098 :  taxi
101 :  car
102 :  bike
107 :  walk
108 :  walk
111 :  taxi
112 :  walk
115 :  car
117 :  walk
125 :  bike
126 :  bike
128 :  car
136 :  walk
138 :  bike
139 :  bike
144 :  walk
153 :  walk
161 :  walk
163 :  bike
167 :  bike
175 :  bus
```

Discussion

The main thing that was done differently was the use of SQLAlchemy. SQLAlchemy is an object relational mapper (ORM for short) which provides a database abstraction layer. An ORM is layered between the developer and the actual database engine. This lets you skip the low-level approach of setting up a relational DBMS locally yourself. With SQLAlchemy you can define Python objects and methods which are automatically translated into SQL database instructions. This gave me a better overview of my overall database structure and reduced the time required to setup the system.

A pain point during the initial part of the assignment was that I struggled with knowing which data the databased contained at any given time. For example, I would insert to one table to test and then forget when doing another full insertion, resulting in duplicate values. Therefore I decided to start using DBeaver to see the content of my database tables in real time. This gave me a full overview of which data was inserted into the database and made troubleshooting much easier.

I did not end up using a lot of git since I completed the assignment by myself. I also did not have much use for issues etc. since the scope of the assignment was limited.

When looking into how to perform bulk inserts with SQLAlchemy there was several options available. Since I wanted to use plain SQL queries and not SQLAlchemy's Core API however, the Connection.exec_driver_sql() seemed to be the best method to perform bulk inserts,. This method utilizes the underlying DBAPI directly and should have the similar performance as using the Core API. I do not have a reference to compare the insertion speed with, so I am not sure if it is an improvement compared to a sequential insert.

Feedback

The database schema provided in the assignment text worked well. It was too much work for 25% doing it by myself, but this was my own choice. I really enjoy these types of practical exercises and believe this is very valuable experience for work and projects in the future.