TDT4225 Assignment 2

Group: 78 Students: Simen Omholt-Jensen

September 30, 2021

Introduction

In this assignment, the Geolife project dataset was parsed from files and inserted into a MySQL database. The relational diagram can be found in figure 1 in the appendix. The subsequent descriptions of the MySQL tables can be found in tables 1, 2, and 3 in the appendix. A public repository containing the source code and a description of how to reproduce the results can be found at Github.

A printout of the ten first lines of the resulting tables in the local MySQL can be found in tables 4, 5 and 6 in the appendix.

After insertion, a combination of MySQL queries and data manipulation in Python was utilized to answer the given questions. This report summarizes the results, and describes some of the methods used for data cleaning before insertion into the database. These data cleaning steps were performed based on the group's interpretation of the given assignment text.

As the group consisted of a single member, there was no need for work coordination. In hindsight, the single group member was content with this decision as the amount of work was manageable. This decision was based on previous experience with working in a team in similar courses.

Results

The tables containing the answers to the questions can be found in the appendix.

Discussion

Part 1

- First, the `main.py` script queries the user for its local MySQL server login information.
- Then, the `main.py` script calls the `setup_database` function in the `database.py` module to connect to a local MySQL server by using the `mysql-connector-python` package. The function then proceeds to drop any database with the name `TDT4225ProjectGroup78` if it exists, and then creates a new database with the same name. The schema definition can be found in the `tables.py` module.
- After the instantiation of the database, the `main.py` script calls the `insert_data` function in the `database.py` module which again calls the `parse_data` function to read the data from the `.plt` files into Pandas DataFrames.
- The `parse_data` function finds the user ids from the folder names and stores them as strings as some of the user ids are prefixed with zeros (e.g. '007'). Information about which users that have labels is then read from the `labeled_ids.txt` file and stored as a boolean value together with the user ids in the `user_df` Pandas DataFrame.
- The `parse_data` function then loads each `.plt` file, stores an activity record as a Pandas Series object, and stores the trackpoint information as a Pandas DataFrame. These are then appended to their respective lists.
- Special care had to be taken to ensure that no trackpoint with more than 2500 entries were read. Further, the user `'020'` had labeled an activity twice, with labels walk and bike. To ensure that all the data was reflected in the database. The group decided to create separate activities for both labels, as this ensures that the database reflects the recorded data.
- The lists containing information about each activity and corresponding trackpoint information is then concatenated into Pandas DataFrames. One for the activities, and one for the trackpoints.
- Lastly, the altitude value of `-777` was replaced by `NULL` values.
- Once the data is parsed into Pandas DataFrames, the `insert_data` establishes a connection to the local MySQL server by using the Python packages `sqlalchemy` and `pymysql`. This allows the Pandas DataFrames to be inserted directly into the MySQL database, and improved performance over the `mysql-connector-python` package.

Part 2

- On completion of the data insertion, the `main.py` script calls the `query_database` function in the `database.py` module. It establishes a connection to the local MySQL server, and calls the relevant query functions found in the `queries.py` module to answer the given questions.

- The query functions for the different questions behave in a similar manner, but some includes additional data manipulation using python packages. Most commonly, the functions query the local MySQL server with some declarative query of which the result is stored in a Pandas DataFrame. The Pandas DataFrame is then printed to the console with the `tabulate` Python package.
- Questions 1 through 5, 7 through 8, and 11 through 12 are answered with MySQL queries only.
- To answer question 6, the local MySQL server is queried by joining the `Activity` and `TrackPoint` tables. Next, the `scikit-learn` python package's implementation of the `DBSCAN` algorithm is used to find clusters of users that are close in time (trackpoints records within 60 seconds). Once these clusters are found, the `DBSCAN` algorithm is run again on each of the time-clusters to find users that are close in space and time (trackpoint records within 100 meters). The haversine distance is used to approximate the earth as a sphere with a radius of 6371.0008 km [1]. Lastly, the number of users is counted as the total number of users that are close in time and space. This means that some users are double counted as they are found in multiple time-space-clusters. Due to the size of the dataset, the group ran into performance issues when trying to query the database directly. The advantage of the use of the `scikit-learn`'s implementation of the `DBSCAN` algorithm is that it uses KDTrees or BallTrees to avoid having to compute the full distance matrices. However, the implementation was quite memory intensive. For example, when trying to cluster on space before time, the computer used to run the code ran out of its 32 GB of memory.
- Question 9 is answered by querying the local MySQL server for all the data in the `Activity` table. The result is then further manipulated in Pandas DataFrames to find the most common year-month. Next, the two users with the most recorded activities in this month are found. To ensure that the number of recorded activities is correct, it is asserted that these two users did not start and activity in the most common year-month and end it in the subsequent.
- Question 10 is answered by querying the local MySQL server for the relevant information from a joining of the `Activity` and `TrackPoint` tables. Then, the total number of kilometers is found by using the `haversine` python package on the resulting Pandas DataFrame.

Considerations, problems, and learning outcomes

The implantation used for this project followed what was suggested and described in the assignment text closely. The tables schemas used were the same as the proposed ones. As MySQL does not include a distinct datatype for boolean values, the TINYINT(1) datatype was used as is the recommended practice. Due to the way the assignment text explained how to match labels to trackpoints, on exact matches on start time and end time, there was a significant number of activities that ended up without a transportation mode. As mentioned above, the implementation had to be adjusted for users that had different labels for the same activity. Further, the Python packages `sqlalchemy` and `pymysql` were used to utilize the Pandas DataFrame interface to MySQL. Most of the queries could be answered directly by querying the local MySQL server, however some answers required additional data manipulation in Pandas DataFrames.

Overall, working on the assignment did not pose too many problems. The group encountered some memory exhaustion issues when trying to use the `scikit-learn`'s implementation of the `DBSCAN` algorithm to cluster on space before time. This was solved by first clustering on time, and then on space.

The group found this assignment interesting as it presented an opportunity to work with a large dataset. Consequently, the runtime of the implementation had to be considered during development and debugging. The parsing and insertion into the database took about 7 minutes, answering question 6 took about 25 minutes, and answering questions 11 and 12 took about 10 minutes. These final durations were significantly lowered from initial attempts by using the `sqlalchemy` and `pymysql` packages to leverage Pandas DataFrame's interface to MySQL. Further, the `scikit-learn`'s implementation of `DBSCAN` had to be utilized in order to answer question 6. The assignment also provided an opportunity to consider memory exhaustion issues, and the group had to think about ways to avoid this problem. As this is a problem not frequently encountered in an academic setting, it proved to be an interesting challenge similar to what student will encounter in a real-life work setting.

References

[1] "Earth radius - Wikipedia." https://en.wikipedia.org/wiki/Earth_radius#Arithmetic_mean_radius.

Appendix

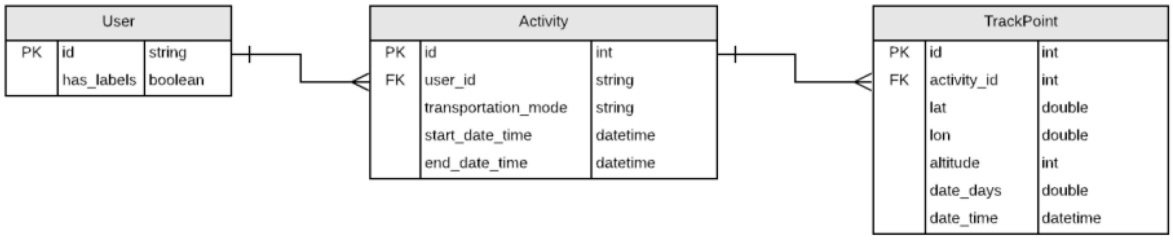

Figure 1: Relational diagram of the Geolife dataset.

Tables

Field	Type	Null	Key	Default	Extra
id	varchar(3)	NO	PRI	NULL	
has_labels	tinyint(1)	NO		NULL	

Table 1: MySQL description of the User table.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
user_id	varchar(3)	NO	MUL	NULL	
transportation_mode	varchar(20)	YES		NULL	
start_date_time	datetime	NO		NULL	
end_date_time	datetime	NO		NULL	

Table 2: MySQL description of the Activity table.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
activity_id	int	NO	MUL	NULL	
lat	double	NO		NULL	
lon	double	NO		NULL	
altitude	int	YES		NULL	
date_days	double	NO		NULL	
date_time	datetime	NO		NULL	

Table 3: MySQL description of the Trackpoint table.

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

Table 4: First ten rown in the MySQL User table.

id	user_id	transportation_mode	start_date_time	end_date_time
0	000	NULL	2009-04-12 07:33:03	2009-04-12 17:24:59
1	000	NULL	2009-05-10 02:02:06	2009-05-10 06:20:11
2	000	NULL	2008-12-01 11:18:27	2008-12-01 11:35:15
3	000	NULL	2009-06-21 10:14:07	2009-06-21 14:47:12
4	000	NULL	2008-12-11 12:14:32	2008-12-11 12:24:32
5	000	NULL	2009-06-29 01:16:30	2009-06-29 11:13:12
6	000	NULL	2009-05-09 18:16:36	2009-05-09 18:30:36
7	000	NULL	2009-07-03 00:28:00	2009-07-03 08:37:23
8	000	NULL	2009-06-07 07:52:55	2009-06-07 09:46:03
9	000	NULL	2009-06-11 22:12:04	2009-06-11 22:34:54

Table 5: First ten rown in the MySQL Activity table.

id	activity_id	lat	lon	altitude	date_days	date_time
0	0	40.000017	116.327479	105	39915.3146180556	2009-04-12 07:33:03
1	0	40.000168	116.327474	80	39915.3146875	2009-04-12 07:33:09
2	0	40.000055	116.327454	99	39915.3147453704	2009-04-12 07:33:14
3	0	40.000021	116.327407	109	39915.3148032407	2009-04-12 07:33:19
4	0	40.000035	116.327281	111	39915.3148611111	2009-04-12 07:33:24
5	0	39.999983	116.327285	114	39915.3149189815	2009-04-12 07:33:29
6	0	39.999853	116.327267	120	39915.3149768518	2009-04-12 07:33:34
7	0	39.999745	116.327165	125	39915.3150347222	2009-04-12 07:33:39
8	0	39.999661	116.326997	126	39915.3150925926	2009-04-12 07:33:44
9	0	39.999528	116.326873	127	39915.315150463	2009-04-12 07:33:49

Table 6: First ten rown in the MySQL TrackPoint table.

Question 1:

Users	Activities	TrackPoints
182	16049	9.682e+06

Table 7: Number of users, activities, and trackpoints in the database after insertion.

Question 2:

Average	Minimum	Maximum
88.1813	0	2102

Table 8: Average, minimum, and maximum number of activities per user.

Question 3:

user_id	Number of Activities
128	2102
153	1793
025	715
163	704
062	691
144	563
041	399
085	364
004	346
140	345

Table 9: The top 10 users with the highest number of activities.

Question 4

Number of Users
98

Table 10: Number of users that have started an activity during one day and ended the activity during the next day.

Question 5:

user_id	start_date_time	end_date_time	count
020	2011-11-02 02:04:29	2011-11-02 02:09:31	2

Table 11: Duplicate activities.

Question 6:

```
Number of close users: 776
```

Question 7:

user_id	user_id	user_id	user_id
000	044	092	138
001	045	093	139
002	046	094	140
003	047	095	141
004	048	096	142
005	049	097	143
006	050	099	144
007	051	100	145
008	052	101	146
009	053	102	147
011	054	103	148
012	055	104	149
013	056	105	150
014	057	106	151
015	059	107	152
016	060	108	153
017	061	109	154
018	063	110	155
019	064	112	156
020	065	113	157
021	066	114	158
022	067	115	159
023	068	116	160
024	069	117	161
025	070	118	162
026	071	119	164
027	072	120	165
028	073	121	166
029	074	122	167
030	075	123	168
031	076	124	169
032	077	125	170
033	079	126	171
034	081	127	172
035	082	129	173
036	083	130	174
037	084	131	175
038	086	132	176
039	087	133	177
040	088	134	178
041	089	135	179
042	090	136	180
043	091	137	181

Table 12: All users that have never taken a taxi.

Question 8:

transportation_mode	Number of Distinct Users
airplane	1
bike	19
boat	1
bus	13
car	8
run	1
subway	4
taxi	10
train	2
walk	34

Table 13: Number of distinct users that have used the different transportation modes.

Question 9:

user_id	recorded_hours	number_of_activities	year_month
062	47.3136	130	2008-November
128	68.2211	75	2008-November

Table 14: The top two users with the most number of recorded activities for the year-month with the most number of recorded activities.

Question 10:

```
Total distance walked: 115.475 km
```

Question 11:

user_id	total_elevation_gain
128	2.13573e+06
153	1.82075e+06
004	1.08936e+06
041	789867
003	766613
085	714064
163	673435
062	596106
144	588730
030	576377
039	481311
084	430319
000	398638
002	377503
167	370647
025	358136
037	325550
140	311129
126	272397
017	205314

Table 15: The top 20 users with the most amount of altitude meters gained.

Question 12:

user_id	number_of_invalid_activities
128	1457
153	1150
025	454
163	446
062	376
085	308
041	302
144	280
004	250
068	218
167	212
003	204
017	181
039	173
140	166
014	150
126	144
092	142
084	140
030	139
115	137
034	122
002	121
000	116
112	114
037	110
104	110
011	90
091	87
013	84
142	80
042	69
022	68
147	64
038	63
168	63
010	61
067	61
082	61
174	60
101	59
096	58
020	57
019	53
134	53
005	52
015	52
089	52
001	51
028	51
071	51
074	50
052	49
051	48
012	47
044	46
073	38
179	38
036	37

user_id	number_of_invalid_activities
103	37
029	36
111	36
065	35
009	34
078	34
125	34
024	33
155	33
007	31
018	31
046	28
088	28
113	28
043	27
119	27
016	26
035	24
169	24
110	23
008	21
026	21
057	21
094	21
150	21
006	20
081	20
040	18
061	18
083	18
097	17
102	17
055	16
058	16
154	16
079	15
181	15
095	14
032	13
138	13
165	13
023	12
064	12
130	12
139	12
157	12
172	12
063	11
099	11
114	11
131	11
158	11
162	11
050	10
056	10
076	10
105	10
161	10
146	9
053	8
093	8

user_id	number_of_invalid_activities
136	8
176	8
021	7
045	7
047	7
066	7
080	7
135	7
159	7
069	6
075	6
086	6
090	6
098	6
122	6
129	6
164	6
173	6
059	5
070	5
100	5
108	5
117	5
121	5
127	5
145	5
109	4
118	4
124	4
133	4
170	4
175	4
027	3
031	3
033	3
077	3
087	3
106	3
107	3
123	3
132	3
166	3
171	3
054	2
072	2
152	2
180	2
048	1
060	1
141	1
151	1

Table 16: The number of invalid activities per user.