Report - template

Assignment 2 - MySQL

Group: 79
Students: Nirushaan Selvaratnam

Introduction

The task is to upload data and answer some questions using queries from a modified dataset based on Geolife GPS Trajectory dataset. I have hosted a MySQL server using Docker and the Python class DbConnector which was provided with the task material. Most of the questions was answered by purely using MySQL queries, while some required some additional coding. The result of my work is provided below as pictures of code/queries and pictures of the results I got.

Results

Part 1

User table:

```
Data from table User, tabulated:
  id    has_labels
----  ------------
 000             0
 001             0
 002             0
 003             0
 004             0
 005             0
 006             0
 007             0
 008             0
 009             0
```

The first ten users in the User table after insertion.

Activity table:

```
Data from table Activity, tabulated:
  id    user_id  transportation_mode    start_date_time      end_date_time
----  ---------  ---------------------  -------------------  -------------------
   1        135                         2009-01-03 01:21:34  2009-01-03 05:40:31
   2        135                         2009-01-02 04:31:27  2009-01-02 04:41:05
   3        135                         2009-01-27 03:00:04  2009-01-27 04:50:32
   4        135                         2009-01-10 01:19:47  2009-01-10 04:42:47
   5        135                         2009-01-14 12:17:57  2009-01-14 12:30:53
   6        135                         2009-01-12 01:41:22  2009-01-12 02:14:01
   7        135                         2008-12-24 14:42:07  2008-12-24 15:26:45
   8        135                         2008-12-28 10:36:05  2008-12-28 12:19:32
   9        132                         2010-02-15 10:56:35  2010-02-15 12:22:33
  10        132                         2010-04-30 23:38:01  2010-05-01 00:35:31
```

The first ten activities in the Activity table. After insertion

TrackPoint table:

```
Data from table TrackPoint, tabulated:
  id    activity_id      lat    lon    altitude  date_time
----  -------------  -------  -----  ----------  -------------------
   1              1  39.9743  116.4         492  2009-01-03 01:21:34
   2              1  39.9743  116.4         492  2009-01-03 01:21:35
   3              1  39.9743  116.4         492  2009-01-03 01:21:36
   4              1  39.9743  116.4         492  2009-01-03 01:21:38
   5              1  39.9744  116.4         491  2009-01-03 01:21:39
   6              1  39.9744  116.4         491  2009-01-03 01:21:42
   7              1  39.9744  116.4         491  2009-01-03 01:21:46
   8              1  39.9745  116.4         491  2009-01-03 01:21:51
   9              1  39.9745  116.4         490  2009-01-03 01:21:56
  10              1  39.9745  116.4         489  2009-01-03 01:22:01
```

The first ten trackpoints in the TrackPoint table after insertion.

Part 2

1. Query:

```
query = """
SELECT
    (SELECT COUNT(*) FROM User) AS user_count,
    (SELECT COUNT(*) FROM Activity) AS activity_count,
    (SELECT COUNT(*) FROM TrackPoint) AS trackpoint_count;
    """
```

Query result:

```
  user_count    activity_count    trackpoint_count
____________  ________________  __________________
         182             16048             9681756
```

After insert I have 182 rows in the User table, 16048 rows in the Activity table, and 9681756 rows in the TrackPoint table. As part of data cleaning, I did not insert plt-files with more than 2500 trackpoints. When inserting labels, I only insert labels for exact matches on start time and end time for an activity. When making the TrackPoint table I did not include the date_days field, because I felt that only using the date_time field was sufficient for solving this exercise.

2. Query:

```
query = """
SELECT AVG(activity_count)
FROM (
    SELECT User.id, COUNT(Activity.id) AS activity_count
    FROM User
    LEFT JOIN Activity ON User.id = Activity.user_id
    GROUP BY User.id
) AS user_activity_count;
"""
```

Query result:

```
  AVG(activity_count)
______________________
              88.1758
```

The average number of activities per user is 88.1758.

3. Query:

```
query = """
SELECT 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 20
"""
```

Query result:

id	activity_count
128	2102
153	1793
025	715
163	704
062	691
144	563
041	399
085	364
004	346
140	345
167	320
068	280
017	265
003	261
014	236
126	215
030	210
112	208
011	201
039	198

These are the user.id for the top 20 users with the highest number of activities.

4. Query:

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

Query result:

id
085
078
062
098
111
128
163
080
010
058

These are all the user.id for all the people that have used taxi as a transportation mode for at least one of their activities.

5. Query:

```
query = """
    SELECT transportation_mode, COUNT(id)
    FROM Activity
    WHERE NOT transportation_mode= 'None'
    GROUP BY transportation_mode
    """
```

Query result:

transportation_mode	COUNT(id)
walk	481
bike	262
bus	199
subway	133
taxi	37
car	419
train	2
run	1
airplane	3
boat	1

These are all the registered transportation modes in the dataset and the number of activities that has the given transportation mode.

6. a) Query:

```
query = """
WITH RECURSIVE year_series AS (
    SELECT
        id,
        YEAR(start_date_time) AS activity_year,
        YEAR(end_date_time) AS end_year
    FROM Activity
    UNION ALL
    SELECT
        id,
        activity_year + 1,
        end_year
    FROM year_series
    WHERE activity_year < end_year
)
SELECT
    activity_year AS year,
    COUNT(DISTINCT id) AS activity_count
FROM year_series
GROUP BY activity_year
ORDER BY activity_count DESC;
"""
```

If the start_date_time and end_date_time span over different years, this query would take it into consideration and count it for both years. E.g. If the start_date_time is 2009-12-31 23:00:00 and end_date_time is 2010-01-01 01:00:00, the activity would count for both years.

Query result:

year	activity_count
2008	5895
2009	5880
2010	1487
2011	1204
2007	994
2012	588
2000	1

The year with the most activities is 2008

b) Query:

```python
total_hours = {}
query = """
SELECT id, start_date_time, end_date_time
FROM Activity
    """
self.cursor.execute(query)
rows = self.cursor.fetchall()

for row in rows:
    id = row[0]
    start_year = row[1].year
    end_year = row[2].year

    if start_year == end_year:
        hours = (row[2] - row[1]).total_seconds() / 3600
        total_hours[start_year] = total_hours.get(start_year, 0) + hours
    else:
        end_of_start_year = datetime(start_year, 12, 31, 23, 59, 59)
        hours_start_year = (end_of_start_year - row[1]).total_seconds() / 3600
        total_hours[start_year] = total_hours.get(start_year, 0) + hours_start_year

        start_of_end_year = datetime(end_year, 1, 1, 0, 0, 0)
        hours_end_year = (row[2] - start_of_end_year).total_seconds() / 3600
        total_hours[end_year] = total_hours.get(end_year, 0) + hours_end_year
sorted_total_hours = sorted(total_hours.items(), key=lambda x: x[1], reverse=True)
headers = ["Year", "total hours"]
print(tabulate(sorted_total_hours[:20],headers=headers, floatfmt=".4f"))
```

Query result:

Year	total hours
2009	11612.6292
2008	9200.5917
2007	2315.4186
2010	1388.7275
2011	1132.3517
2012	711.2133
2000	0.0511

From the results we can see that even though 2008 has the most recorded activities the total hour of activity is by far higher in 2009. The total hour of activity is calculated by first calculating the time passed in seconds for each activity by comparing the start_date_time and end_date_time and then dividing it by 3600 so we do not only count whole hours for activities, but also partial hours. Then we simply add the number for every activity within a year. If an activity stretches over two years, the code takes it into consideration.

7. Query and code:

```
query = """
SELECT TrackPoint.lat, TrackPoint.lon, Activity.id, TrackPoint.date_time, Activity.transportation_mode
FROM User
JOIN Activity ON User.id = Activity.user_id
JOIN TrackPoint ON Activity.id = TrackPoint.activity_id
WHERE User.id = '112'
AND YEAR(TrackPoint.date_time) = 2008
AND Activity.transportation_mode = 'walk'
ORDER BY Activity.id DESC, TrackPoint.date_time ASC;
"""
self.cursor.execute(query)
rows = self.cursor.fetchall()
distances = {}
current_activity_id = None
last_coordinates = None

for row in rows:
    lat, lon, activity_id, date_time, transportation_mode = row
    coordinates = (lat, lon)

    if activity_id not in distances:
        distances[activity_id] = 0.0
        last_coordinates = None

    if last_coordinates is not None and current_activity_id == activity_id:
        distance = haversine(last_coordinates, coordinates)
        distances[activity_id] += distance

    last_coordinates = coordinates
    current_activity_id = activity_id

print(f"Total distance walked in 2008 by user 112: {sum(distances.values())} km")
```

Query and code result:

```
Total distance walked in 2008 by user 112: 115.47465961508004 km
```

The total distance walked by user 112 in 2008 is 115.47 km. This was more challenging than the other queries because you had to make sure that your previous TrackPoint was from the same activity when calculating distance between two TrackPoints. To make sure that consecutive trackpoints were being considered within each activity I order on Activity.id first and then on TrackPoint.date_time.

8. Query:

```
query = """
SELECT User.id, Activity.id, TrackPoint.altitude, TrackPoint.date_time
FROM User
JOIN Activity ON User.id = Activity.user_id
JOIN TrackPoint ON Activity.id = TrackPoint.activity_id
WHERE NOT TrackPoint.altitude = -777
ORDER BY User.id DESC, Activity.id ASC, TrackPoint.date_time ASC;
"""
```

I only include the query because the code is long, but it can be found in geolife.py inside function: task_8().

Query and code result:

user	altitude gained
128	2135759.0000
153	1820766.0000
004	1089358.0000
041	789890.0000
003	766613.0000
085	714049.0000
163	673439.0000
062	596103.0000
144	588767.0000
030	576428.0000
039	481311.0000
084	430319.0000
000	398638.0000
002	377503.0000
167	370647.0000
025	358098.0000
037	325528.0000
140	311151.0000
126	272389.0000
017	205270.0000

These are the top 20 users with the most altitude gained throughout their registered activities. I only consider invalid altitude values to be -777, if it has any other value than that it would be considered valid.

9. Query:

```
query = """
    WITH InvalidActivities AS (
        SELECT
            Activity.user_id,
            Activity.id AS activity_id
        FROM Activity
        JOIN TrackPoint T_1 ON Activity.id = T_1.activity_id
        JOIN TrackPoint T_2 ON Activity.id = T_2.activity_id
        WHERE T_1.id = T_2.id - 1
        AND TIMESTAMPDIFF(MINUTE, T_1.date_time, T_2.date_time) >= 5
        GROUP BY Activity.id, Activity.user_id
    )
    SELECT
        InvalidActivities.user_id,
        COUNT(InvalidActivities.activity_id) AS invalid_activity_count
    FROM InvalidActivities
    GROUP BY InvalidActivities.user_id
    ORDER BY InvalidActivities.user_id DESC;
    """
```

Query result:

user_id	invalid_activity_count
181	14
180	2
179	28
176	8
175	4
174	54
173	5
172	9
171	3
170	2
169	9
168	19
167	134
166	2
165	2
164	6
163	233
162	9
161	7
159	5
158	9
157	9
155	30
154	14
153	557
152	2
151	1
150	16
147	30
146	7
145	5
144	157
142	52
141	1
140	86
139	12
138	10
136	6
135	5
134	31
133	4
132	3
131	10
130	8
129	6
128	720
127	4
126	105
125	25
124	4
123	3
122	6
121	4
119	22
118	3
117	3
115	58
114	3
113	1
112	67
111	26
111	26
110	17
109	3
108	5
107	1
106	3
105	9
104	97
103	24
102	13
101	46
100	3
099	11
098	5
097	14
096	35
095	4
094	16
093	4
092	101
091	63
090	3
089	40
088	11
087	3
086	5
085	184
084	99
083	15
082	27
081	16
080	6
079	2
078	19
077	3
076	8
075	6
074	19
073	18
072	2
071	29
070	5
069	6
068	139
067	33
066	6
065	26
064	7
063	8
062	249
061	12
060	1
059	5
058	13
057	16
056	7
055	15
054	2
053	7
052	44
051	36
050	8
048	1
047	6
046	13
045	7
045	7
044	32
043	21
042	55
041	201
040	17
039	147
038	58
037	100
036	34
035	23
034	88
033	2
032	12
031	3
030	112
029	25
028	36
027	2
026	18
025	263
024	27
023	11
022	55
021	7
020	20
019	31
018	27
017	129
016	20
015	46
014	118
013	29
012	43
011	32
010	50
009	31
008	16
007	30
006	17
005	45
004	219
003	179
002	98
001	45
000	101

These are all the users that have invalid activities, and the number of invalid activities per user. The query compares two consecutive trackpoints within the same activity and makes sure that the time between them is not 5 minutes or more.

10. Query:

```
query = """
SELECT DISTINCT User.id
FROM User
JOIN Activity ON User.id = Activity.user_id
JOIN TrackPoint ON Activity.id = TrackPoint.activity_id
WHERE TrackPoint.lat LIKE '39.916%'
AND TrackPoint.lon LIKE '116.397%';
"""
```

Query result:

id
018
019

These are the users that have Trackpoints recorded within the forbidden city of Beijing. I matched the latitude to 39.916 and longitude to 116.397. Every decimal after the first three can be whatever, which means that the integer part and the first three decimals are an exact match.

11. Query:

```
query = """
WITH TransportationCount AS (
    SELECT
        User.id,
        Activity.transportation_mode,
        COUNT(Activity.transportation_mode) AS mode_count
    FROM User
    JOIN Activity ON User.id = Activity.user_id
    WHERE User.has_labels = true AND Activity.transportation_mode IS NOT NULL
    GROUP BY User.id, Activity.transportation_mode
),
RankedTransportation AS (
    SELECT
        id,
        transportation_mode,
        mode_count,
        ROW_NUMBER() OVER (PARTITION BY id ORDER BY mode_count DESC) AS mode_rank
    FROM TransportationCount
)
SELECT
    id,
    transportation_mode AS most_used_transportation_mode
FROM RankedTransportation
WHERE mode_rank = 1
ORDER BY id ASC;
"""
```

Query result:

id	most_used_transportation_mode
010	taxi
020	bike
021	walk
052	bus
056	bike
058	walk
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	walk
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

These are all the users who have registered transportation mode and their most used transportation mode. I did not include rows where the mode is null so some users, even if they have labeled their data, I could not find any exact matches in their activities.

Discussion

I followed the recommended table and fields layout for the database which was provided in the assignment sheet. When it came to choosing between keeping the date_days or date_time field in the TrackPoint table, I chose date_time as it was using the datetime type, because it is easier to work with for MySQL queries.

A pain point from this assignment was that using “simple”, or brute-force solutions often made writing or reading from the database very time consuming. When writing to the database I chose to store certain information in data structures and used batch inserts instead of inserting single rows. Storing certain information in dictionaries or lists made it so that creating the class took longer, but significantly reduced the time it took to insert data into both the Activity and TrackPoint table.

From this assignment I learned that a lot of time is spent on data cleaning, and it is the part that is hardest. Especially in comparison to finding query syntax to answer questions from the database. I also learned that there are lots of anomalies or just bad data hidden when you have a huge dataset that cannot be found by simple scrolling through the data.

In conclusion, this assignment highlighted the importance of optimizing database operations. Batch processing and storing information in data structures proved to be effective for improving efficiency. Finally, it became evident that data cleaning is crucial and often a challenging step when working with large datasets.