Report

Assignment 2 - MySQL

Group: 50
Students: Hermann Owren Elton, Stian Fjæran Mogen and Olaf Rosendahl

Introduction

The assignment was to insert and interact with the large Geolife GPS Trajectory dataset using a MySQL database and Python. The dataset contains data for user, activities, and track points, which the group inserted into separate tables. The user can have many activities (many to one), and an activity may have many track points representing location data (many to one).

The group chose to run the inserts locally using a MySQL-container in Docker. We did additionally use a PHPMyAdmin-container to simplify viewing the tables and testing SQL-queries. Development was done with the Visual Studio Code feature "Live Share", giving the opportunity for all three group members to work concurrently on the code. After each work session the progress was committed to the Github repository found here: https://github.com/Her0elt/TDT4225-Very-Large-Distributed-Data-Volumes

For the tasks in part two, the group's approach was to try to do as much as possible with SQL-queries due to the performance/speed compared to Python. Even though efficiency was not a criteria, it should be considered when working with large amounts of data. It is also a fun challenge and good practice. For certain more complicated tasks, Python operations were needed to process the returned data in order to answer properly.

Results

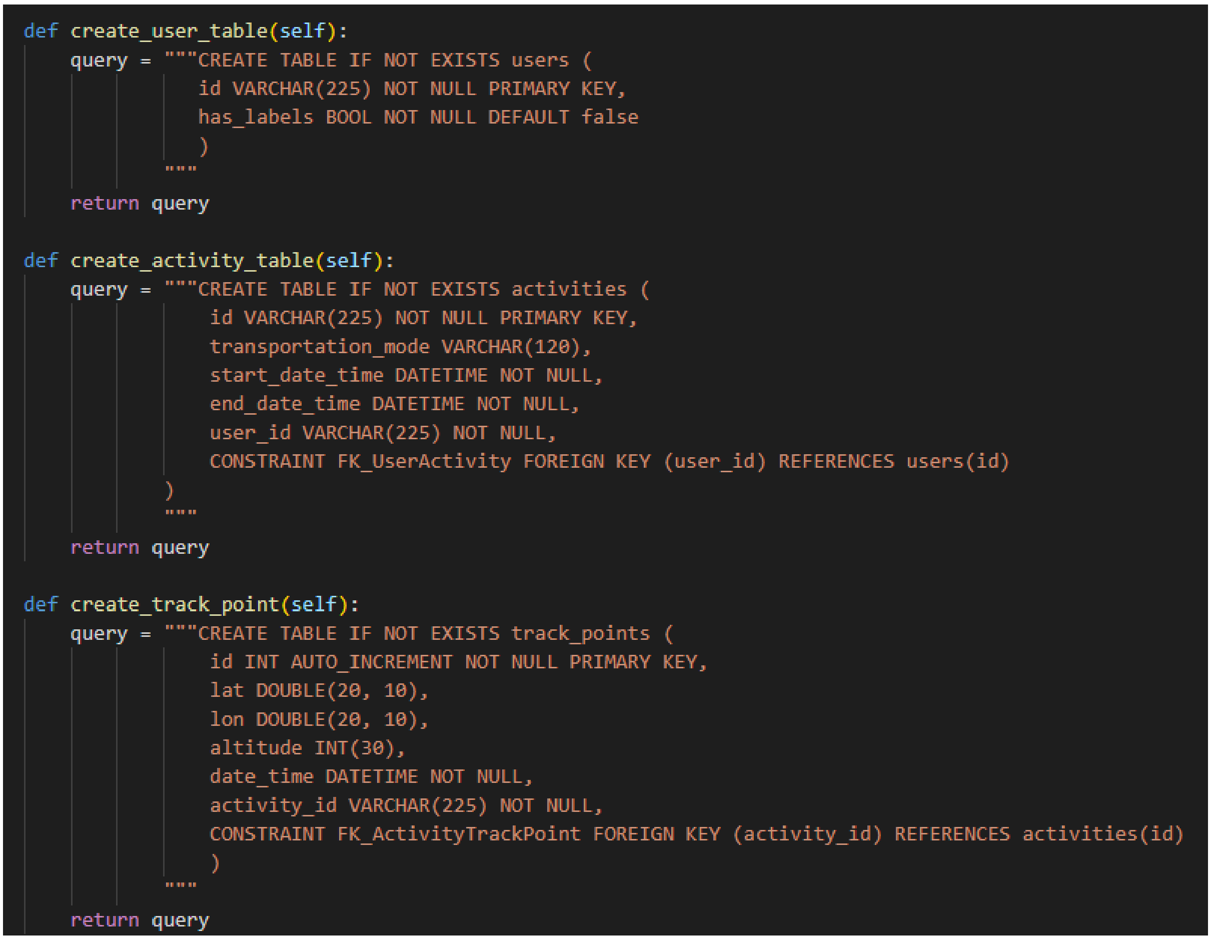
Part 1

1. Connects to the MySQL server on your Ubuntu virtual machine.

Since we chose to do the task locally, rather than connecting to the VM, we connected to the MySQL-container in Docker. Docker Compose was used to easily run MySQL and PHPMyAdmin together. The Makefile defines the commands used for setup and development.

2. Creates and defines the tables User, Activity and TrackPoint

The types are as defined in the text document. Has_labels is defaulted to false as it cannot be null. The activity-id was created by combining the trajectory-filename and user_id. The integer id in track point is auto incremented. For the track_points, we decided to ignore the days-since-column and combine date and time into a datetime-column as this was enough to complete the tasks.



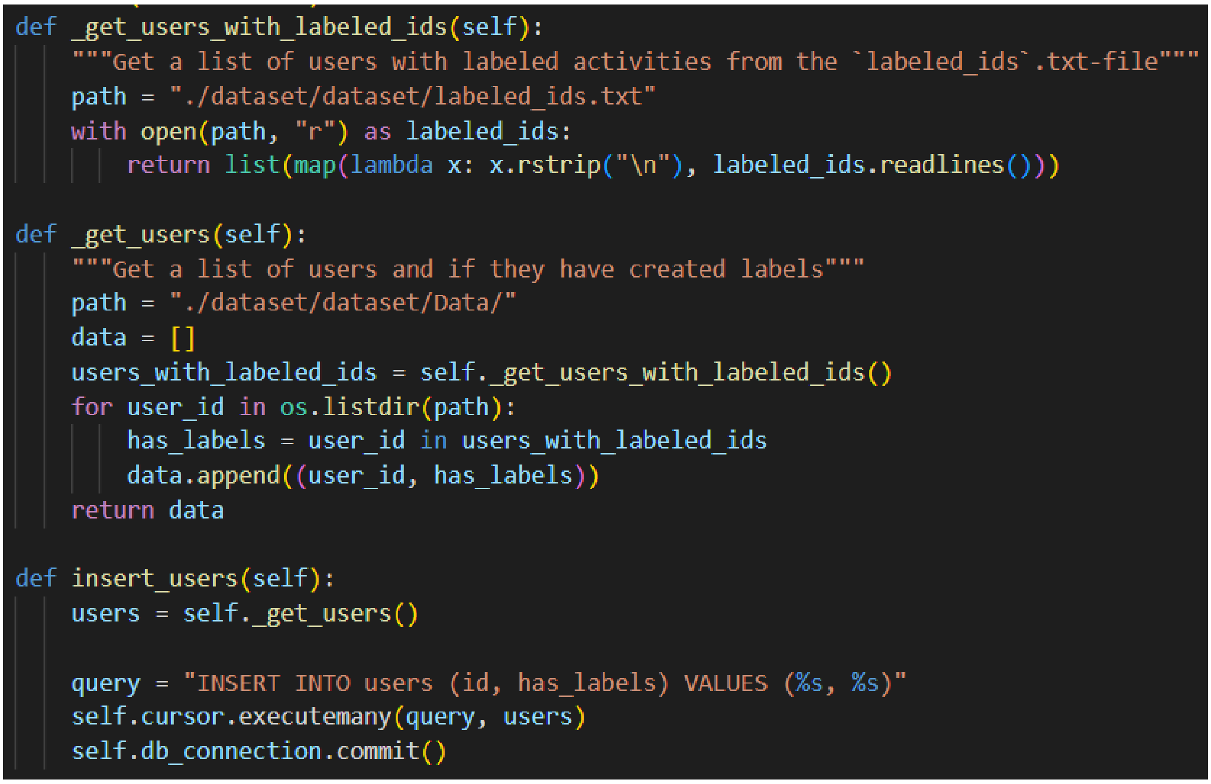
```
def create_user_table(self):
    query = """CREATE TABLE IF NOT EXISTS users (
               id VARCHAR(225) NOT NULL PRIMARY KEY,
               has_labels BOOL NOT NULL DEFAULT false
               )
            """
    return query

def create_activity_table(self):
    query = """CREATE TABLE IF NOT EXISTS activities (
                id VARCHAR(225) NOT NULL PRIMARY KEY,
                transportation_mode VARCHAR(120),
                start_date_time DATETIME NOT NULL,
                end_date_time DATETIME NOT NULL,
                user_id VARCHAR(225) NOT NULL,
                CONSTRAINT FK_UserActivity FOREIGN KEY (user_id) REFERENCES users(id)
            )
            """
    return query

def create_track_point(self):
    query = """CREATE TABLE IF NOT EXISTS track_points (
                id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
                lat DOUBLE(20, 10),
                lon DOUBLE(20, 10),
                altitude INT(30),
                date_time DATETIME NOT NULL,
                activity_id VARCHAR(225) NOT NULL,
                CONSTRAINT FK_ActivityTrackPoint FOREIGN KEY (activity_id) REFERENCES activities(id)
                )
            """
    return query
```

3. Inserts the data from the Geolife dataset into your MySQL database

For the insertion of the data, we chose to firstly insert the users, as these exist independent of any other table. To insert users, we first need to find the users to insert, and their has_labels field. As there are not too much data to insert yet, we can do this in one executemany operation:



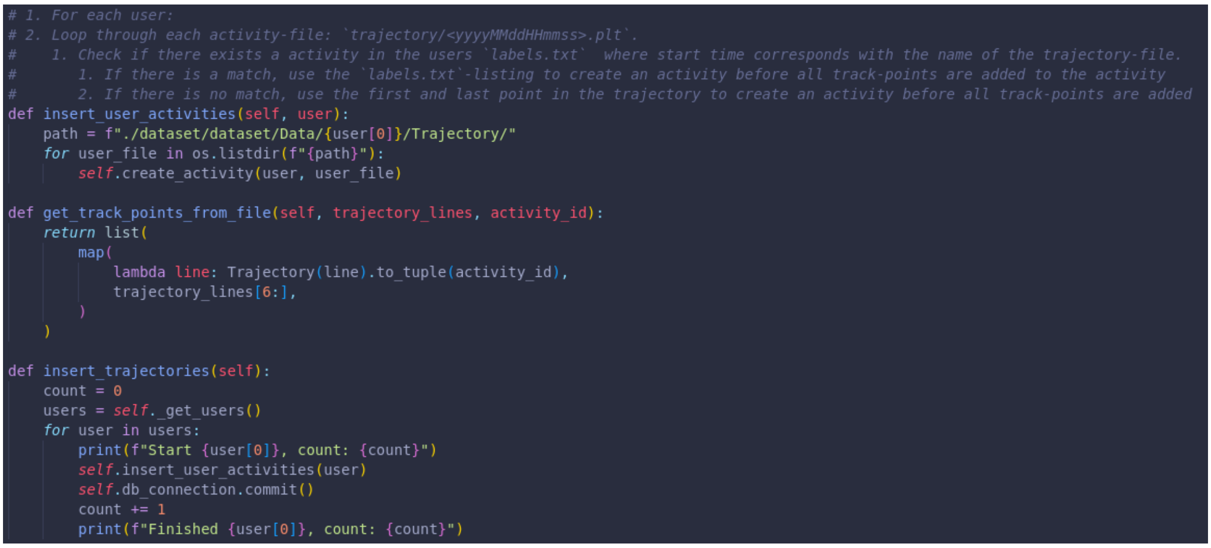
```python
def _get_users_with_labeled_ids(self):
    """Get a list of users with labeled activities from the `labeled_ids`.txt-file"""
    path = "./dataset/dataset/labeled_ids.txt"
    with open(path, "r") as labeled_ids:
        return list(map(lambda x: x.rstrip("\n"), labeled_ids.readlines()))

def _get_users(self):
    """Get a list of users and if they have created labels"""
    path = "./dataset/dataset/Data/"
    data = []
    users_with_labeled_ids = self._get_users_with_labeled_ids()
    for user_id in os.listdir(path):
        has_labels = user_id in users_with_labeled_ids
        data.append((user_id, has_labels))
    return data

def insert_users(self):
    users = self._get_users()

    query = "INSERT INTO users (id, has_labels) VALUES (%s, %s)"
    self.cursor.executemany(query, users)
    self.db_connection.commit()
```

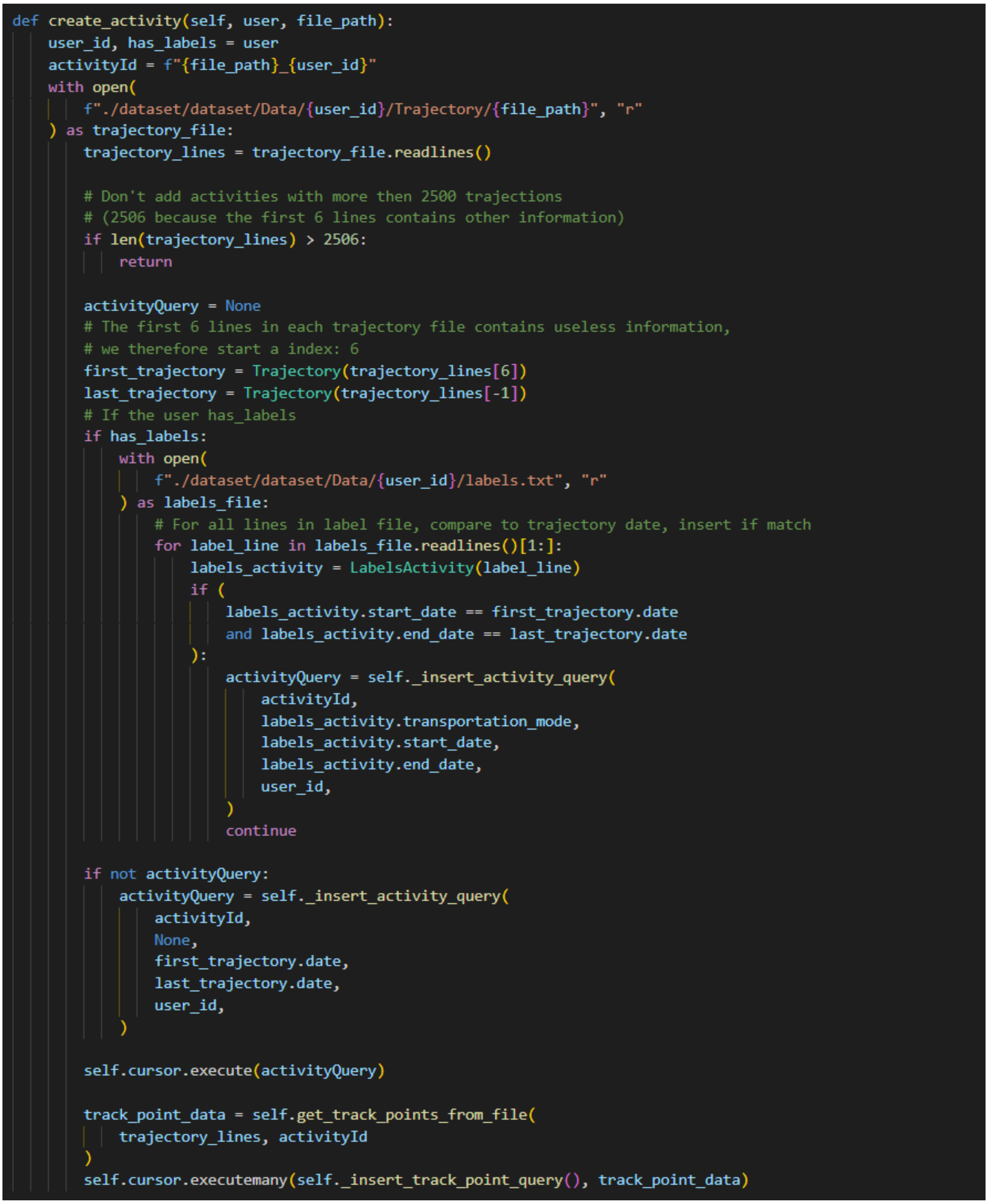
Activities and their corresponding trajectories / track points were inserted together, and executed in batches. As these are large operations with a lot of data, this was considered to be the most efficient option. To limit memory usage, the execution is limited to one activity at a time. Meaning that the program first inserts an activity for a user and executes, then finds and inserts all the corresponding track points with executemany. This process is repeated for all user activities, before it moves on to the next user.



```python
# 1. For each user:
# 2. Loop through each activity-file: `trajectory/<yyyyMMddHHmmss>.plt`.
#    1. Check if there exists a activity in the users `labels.txt`  where start time corresponds with the name of the trajectory-file.
#       1. If there is a match, use the `labels.txt`-listing to create an activity before all track-points are added to the activity
#       2. If there is no match, use the first and last point in the trajectory to create an activity before all track-points are added
def insert_user_activities(self, user):
    path = f"./dataset/dataset/Data/{user[0]}/Trajectory/"
    for user_file in os.listdir(f"{path}"):
        self.create_activity(user, user_file)

def get_track_points_from_file(self, trajectory_lines, activity_id):
    return list(
        map(
            lambda line: Trajectory(line).to_tuple(activity_id),
            trajectory_lines[6:],
        )
    )

def insert_trajectories(self):
    count = 0
    users = self._get_users()
    for user in users:
        print(f"Start {user[0]}, count: {count}")
        self.insert_user_activities(user)
        self.db_connection.commit()
        count += 1
        print(f"Finished {user[0]}, count: {count}")
```



```python
def create_activity(self, user, file_path):
    user_id, has_labels = user
    activityId = f"{file_path}_{user_id}"
    with open(
        f"./dataset/dataset/Data/{user_id}/Trajectory/{file_path}", "r"
    ) as trajectory_file:
        trajectory_lines = trajectory_file.readlines()

        # Don't add activities with more then 2500 trajections
        # (2506 because the first 6 lines contains other information)
        if len(trajectory_lines) > 2506:
            return

        activityQuery = None
        # The first 6 lines in each trajectory file contains useless information,
        # we therefore start a index: 6
        first_trajectory = Trajectory(trajectory_lines[6])
        last_trajectory = Trajectory(trajectory_lines[-1])
        # If the user has_labels
        if has_labels:
            with open(
                f"./dataset/dataset/Data/{user_id}/labels.txt", "r"
            ) as labels_file:
                # For all lines in label file, compare to trajectory date, insert if match
                for label_line in labels_file.readlines()[1:]:
                    labels_activity = LabelsActivity(label_line)
                    if (
                        labels_activity.start_date == first_trajectory.date
                        and labels_activity.end_date == last_trajectory.date
                    ):
                        activityQuery = self._insert_activity_query(
                            activityId,
                            labels_activity.transportation_mode,
                            labels_activity.start_date,
                            labels_activity.end_date,
                            user_id,
                        )
                        continue

        if not activityQuery:
            activityQuery = self._insert_activity_query(
                activityId,
                None,
                first_trajectory.date,
                last_trajectory.date,
                user_id,
            )

        self.cursor.execute(activityQuery)

        track_point_data = self.get_track_points_from_file(
            trajectory_lines, activityId
        )
        self.cursor.executemany(self._insert_track_point_query(), track_point_data)
```

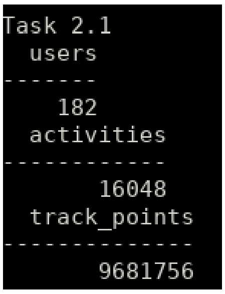
It is **important to note** that for our specific implementation, we chose to interpret the task to ask for an exact match for activity and track points on **either the start and end time of the transportation mode activity.** We are aware that one way to interpret the task is to consider all datetimes of the track points in the file. This will result in more relations between activities and track points, however we chose the more strict way of handling the relation based on our understanding of the task.

Part 2

For these tasks, most were done solely with SQL-queries, others were done partly in Python. Some are pretty straight forward, while for some tasks a short explanation is necessary.

1. How many users, activities and trackpoints are there in the dataset (after it is inserted into the database).

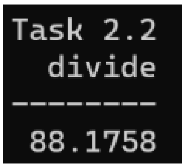
Answer: 182 users, 16048 activities, 9681756 trackpoints



```
Task 2.1
  users
-------
    182
  activities
------------
       16048
  track_points
--------------
       9681756
```

2. Find the average number of activities per user.

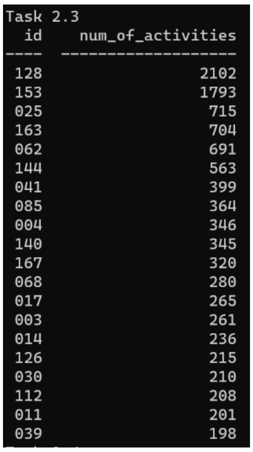
Answer: 88.1758 activities on average per user



```
Task 2.2
  divide
--------
 88.1758
```

3. Find the top 20 users with the highest number of activities.

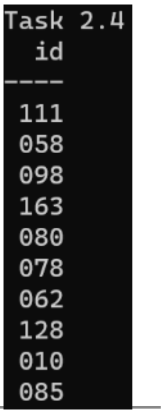
Answer: To solve this, the count of entries is selected as num_of_activites. We group the users by their id, and order by the count. The table represent the top 20 most active users.



```
Task 2.3
  id    num_of_activities
----  -------------------
 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
```

4. Find all users who have taken a taxi.

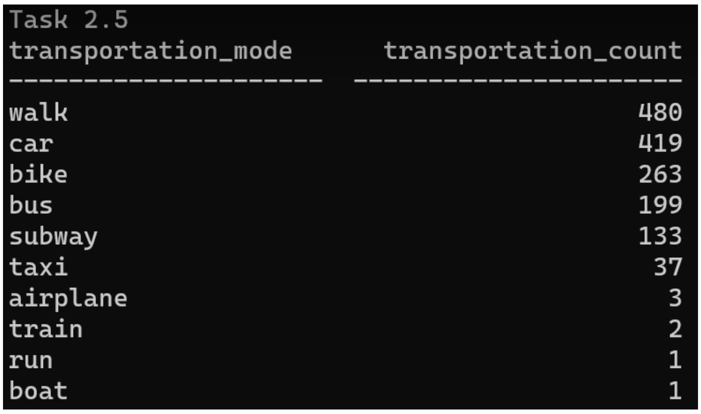
Answer; By selecting distinct users, we find that the following users have taken a taxi: { 111, 058, 098, 163, 080, 078, 062, 128, 010, 085 }.



```
Task 2.4
  id
----
 111
 058
 098
 163
 080
 078
 062
 128
 010
 085
```

5. Find all types of transportation modes and count how many activities that are tagged with these transportation mode labels. Do not count the rows where the mode is null.

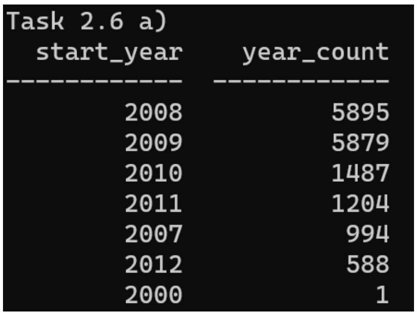
Answer: The table represents the most used transportation modes. Note that we check if we've inserted 'None' when no transportation mode is registered. This is not optimal, and we should have used a Null value instead, and would change this if we were to do the assignment again.



```
Task 2.5
transportation_mode      transportation_count
---------------------  ----------------------
walk                                      480
car                                       419
bike                                      263
bus                                       199
subway                                    133
taxi                                       37
airplane                                    3
train                                       2
run                                         1
boat                                        1
```

6. Task 6
   a. Find the year with the most activities.

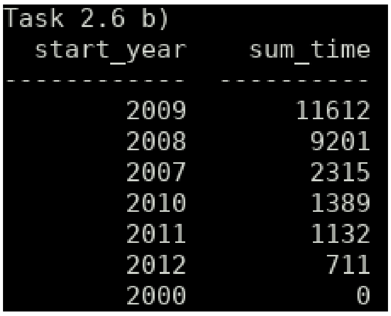
Answer: To find the year with the most activities, we group by start year and order by year count (we could limit 1, but out of interest we wanted to see all years). The most activities were registered in 2008.
Note: A task for a year is based on the start_date. This means that a task started in 2008 and ended in 2009, would be registered as an activity from 2008. This is to prevent cases where the sum of activities from each year exceeds the total amount of activities due to overlap.



```
Task 2.6 a)
  start_year    year_count
------------  ------------
        2008          5895
        2009          5879
        2010          1487
        2011          1204
        2007           994
        2012           588
        2000             1
```

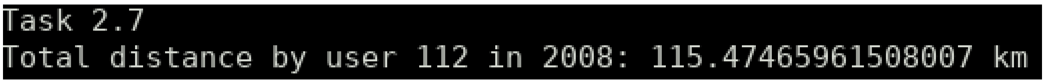
   b. Is this also the year with the most recorded hours?

Answer: The group understands the task asks for the total time spent on activities in total (mort recorded hours). By finding the sum of time difference in seconds and dividing by 3600, we can see that the year with the most recorded hours was in fact 2009, with 2008 moving down to second place.



```
Task 2.6 b)
  start_year    sum_time
------------  ----------
        2009       11612
        2008        9201
        2007        2315
        2010        1389
        2011        1132
        2012         711
        2000           0
```

7. Find the total distance (in km) walked in 2008, by user with id=112

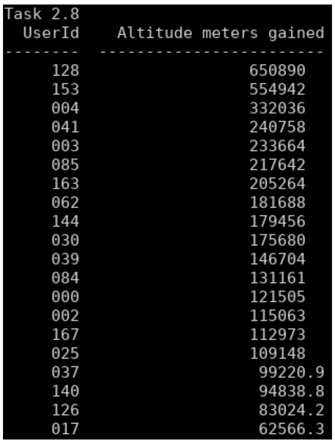


```
Task 2.7
Total distance by user 112 in 2008:  115.47465961508007 km
```

Answer: We retrieve all track_points in activities that belong to the user: 112, has transportation_mode: 'walk' and is in 2008. Then we group the track_points into their activity and calculate the total distance walked in each activity with a function that finds the distance in kilometers for a list of geo-coordinates. Finally the distance for each activity is summed up.

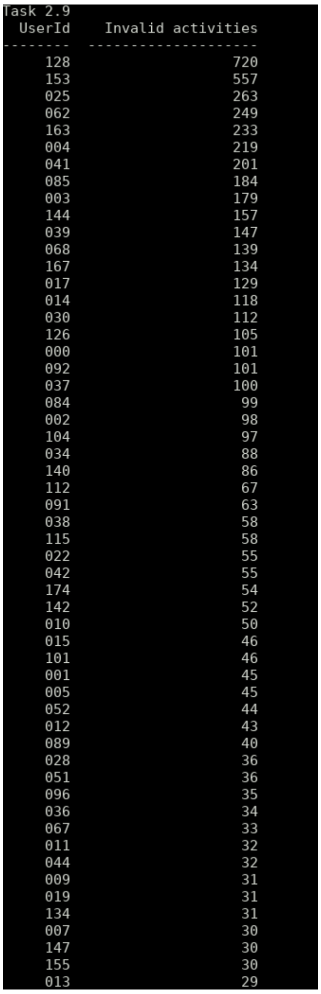
8. Find the top 20 users who have gained the most altitude meters.

Answer: Loops through all trackpoints and creates a count of altitude gained for each user. Only increases it if the compared trackpoint is from the same user ***and*** *activity. Finally the users are sorted by altitude gained and limited to top 20.*

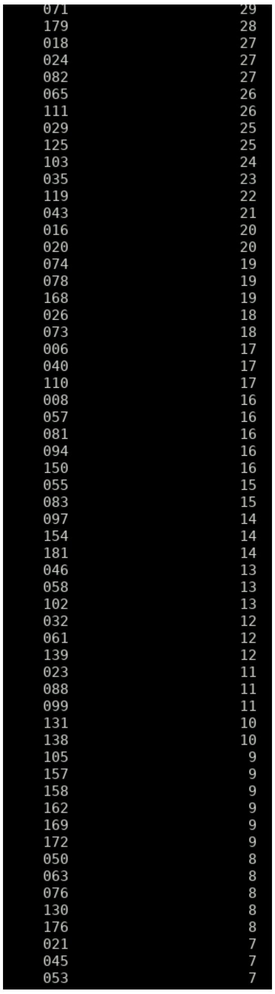


```
Task 2.8
  UserId     Altitude meters gained
--------  ------------------------
     128                    650890
     153                    554942
     004                    332036
     041                    240758
     003                    233664
     085                    217642
     163                    205264
     062                    181688
     144                    179456
     030                    175680
     039                    146704
     084                    131161
     000                    121505
     002                    115063
     167                    112973
     025                    109148
     037                     99220.9
     140                     94838.8
     126                     83024.2
     017                     62566.3
```

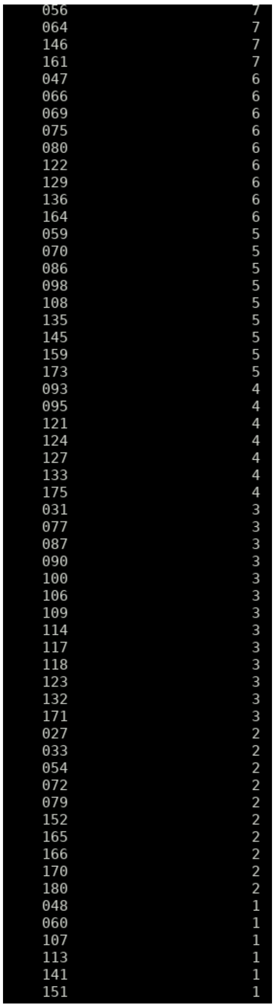
9. Find all users who have invalid activities, and the number of invalid activities per user



```
Task 2.9
  UserId     Invalid activities
--------  --------------------
     128                   720
     153                   557
     025                   263
     062                   249
     163                   233
     004                   219
     041                   201
     085                   184
     003                   179
     144                   157
     039                   147
     068                   139
     167                   134
     017                   129
     014                   118
     030                   112
     126                   105
     000                   101
     092                   101
     037                   100
     084                    99
     002                    98
     104                    97
     034                    88
     140                    86
     112                    67
     091                    63
     038                    58
     115                    58
     022                    55
     042                    55
     174                    54
     142                    52
     010                    50
     015                    46
     101                    46
     001                    45
     005                    45
     052                    44
     012                    43
     089                    40
     028                    36
     051                    36
     096                    35
     036                    34
     067                    33
     011                    32
     044                    32
     009                    31
     019                    31
     134                    31
     007                    30
     147                    30
     155                    30
     013                    29
```



```
     071                    29
     179                    28
     018                    27
     024                    27
     082                    27
     065                    26
     111                    26
     029                    25
     125                    25
     103                    24
     035                    23
     119                    22
     043                    21
     016                    20
     020                    20
     074                    19
     078                    19
     168                    19
     026                    18
     073                    18
     006                    17
     040                    17
     110                    17
     008                    16
     057                    16
     081                    16
     094                    16
     150                    16
     055                    15
     083                    15
     097                    14
     154                    14
     181                    14
     046                    13
     058                    13
     102                    13
     032                    12
     061                    12
     139                    12
     023                    11
     088                    11
     099                    11
     131                    10
     138                    10
     105                     9
     157                     9
     158                     9
     162                     9
     169                     9
     172                     9
     050                     8
     063                     8
     076                     8
     130                     8
     176                     8
     021                     7
     045                     7
     053                     7
```

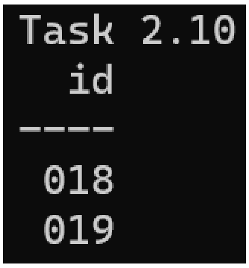


```
     056                     7
     064                     7
     146                     7
     161                     7
     047                     6
     066                     6
     069                     6
     075                     6
     080                     6
     122                     6
     129                     6
     136                     6
     164                     6
     059                     5
     070                     5
     086                     5
     098                     5
     108                     5
     135                     5
     145                     5
     159                     5
     173                     5
     093                     4
     095                     4
     121                     4
     124                     4
     127                     4
     133                     4
     175                     4
     031                     3
     077                     3
     087                     3
     090                     3
     100                     3
     106                     3
     109                     3
     114                     3
     117                     3
     118                     3
     123                     3
     132                     3
     171                     3
     027                     2
     033                     2
     054                     2
     072                     2
     079                     2
     152                     2
     165                     2
     166                     2
     170                     2
     180                     2
     048                     1
     060                     1
     107                     1
     113                     1
     141                     1
     151                     1
```

Answer: Loops through all trackpoints and creates a count of invalid activities per user. Only increases it if the compared trackpoint is more than 5 minutes apart and the activity hasn't been marked as invalid already. Finally the users are sorted by the amount of invalid activities and limited to top 20.

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

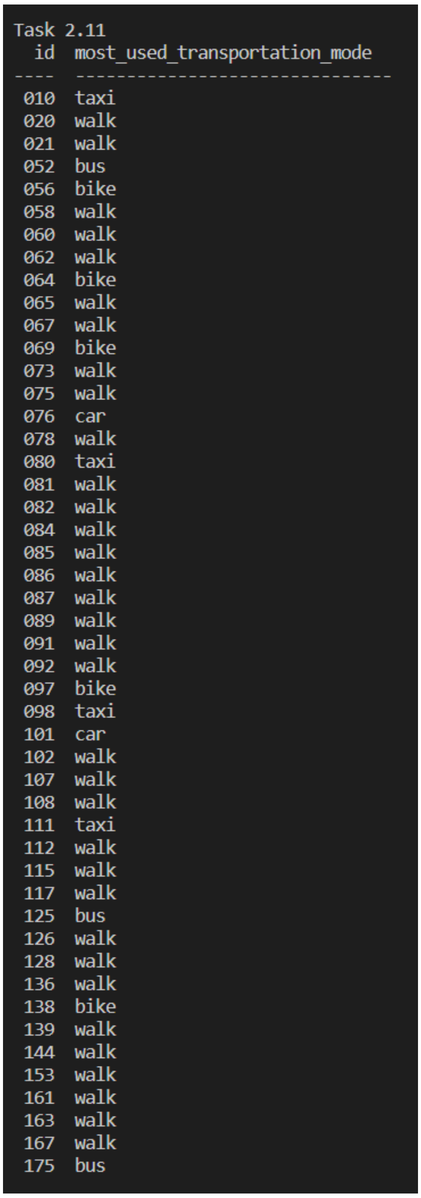
Answer: We use the BETWEEN method to find the latitude and longitude values that are within the definition of the task (since the track point data is more precise and has more decimals we cannot do an exact match). The distinct select gives us user 018 and 019.



```
Task 2.10
  id
----
 018
 019
```

11. Find all users who have registered transportation_mode and their most used transportation_mode.

Answer: Using a sub-query which returns transportation_mode and user_id grouped, it is easy to find the most used transportation_mode for each user with the MAX-function. We had to use "transportation_mode != 'None'" here as well since activities without a transportation_mode was registered with None and not NULL. Only 59 of the 69 users in labeled_ids.txt do have activities which are labeled. The 10 users without labeled activities are not included in this list.



```
Task 2.11
  id  most_used_transportation_mode
----  -----------------------------
 010  taxi
 020  walk
 021  walk
 052  bus
 056  bike
 058  walk
 060  walk
 062  walk
 064  bike
 065  walk
 067  walk
 069  bike
 073  walk
 075  walk
 076  car
 078  walk
 080  taxi
 081  walk
 082  walk
 084  walk
 085  walk
 086  walk
 087  walk
 089  walk
 091  walk
 092  walk
 097  bike
 098  taxi
 101  car
 102  walk
 107  walk
 108  walk
 111  taxi
 112  walk
 115  walk
 117  walk
 125  bus
 126  walk
 128  walk
 136  walk
 138  bike
 139  walk
 144  walk
 153  walk
 161  walk
 163  walk
 167  walk
 175  bus
```

Discussion

We found the overall assignment to be instructive and fun. It was a little challenging to navigate the file to properly insert the track_points to the activities, but conceptually it was fairly straight forward. Some parts of the task were either a bit hard to grasp, or left to interpretation. For example whether to cut down track points to 2500, or leave them out entirely if more than 2500 was registered (this was the solution choice based on an answer in piazza). In these cases, the answers from the assistants or lectures were quite helpful.

Since we decided to only match activities with transportation mode, on trackpoints with exact matches on start and end date only, it is possible that some tasks gave fewer results than the alternative, regardless we feel it still answered the task description. Regardless of the specifics in the data cleaning, the queries and methods designed for task 2 should be scalable regardless of how one chooses to insert the data.

Some of the things we have learned through this assignment is that it's important to read the tasks and assignment-description carefully. If we'd done that better from the start, we would probably have used less time on the assignment, not having to edit the insert script and run the inserts multiple times, something that took quite a long time on a couple of our machines. Additionally we've also become even more familiar with MySQL and it's built-in functions and query-options. Executing queries in MySQL really is much quicker than to manually do them in Python.