Report

Assignment 3 - MongoDB

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 MongoDB database and Python. The dataset contains data for user, activities, and track points, which we 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).

Differently to MySQL, MongoDB is a document database, which led to some difference in approach, which we deemed necessary. Document databases are easy to model and store, and reads may be efficient for certain types of stored data. Relations are more complicated. This led to us storing the data slightly differently, for example with the total distance of an activity being stored in the activity itself, a task which we considered difficult in exercise 2. In this way one could limit the amount of joins on read, and instead calculate it on insert, sacrificing some storage for efficiency.

Results

Part 1

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

The major difference in the approach as compared to MySQL, was that for this exercise we only stored users with registered activities. We did this because MongoDB recommends that data that depend on each other also coexist. This means that the only users stored in the database are those who also have at least one activity. We decided to also store the user_id in each track_point-document.

The consequence of this was that only two tables were necessary; activities and track_points. This was easily done with:

```
self.db.create_collection(collection_name)
```

```python
class DatabaseUtil:
    ACTIVITIES_COLLECTION_NAME = 'activities'
    TRACK_POINTS_COLLECTION_NAME = 'track_points'

    def __init__(self):
        self.connection = DbConnector()
        self.client = self.connection.client
        self.db = self.connection.db

        self.activity_collection = self.db[self.ACTIVITIES_COLLECTION_NAME]
        self.track_point_collection = self.db[self.TRACK_POINTS_COLLECTION_NAME]

    def cleanup(self):
        self.connection.close_connection()
```

2. Inserting the data

The creation of activities and track points were described extensively in the first report. This report will therefore highlight the differences in approach. The code will still be documented with the implementation details.

As the document database does not use the predefined schema as the SQL-model, making insertion quite a bit easier programmatically, while the creation of the Activity class still requires the same data-cleaning as in exercise 2.

```python
def insert(self):
    count = 0
    users = self._get_users()
    activities = []


    for user in users:
        print(f"Start {user['_id']}, count: {count}")
        user_activities, user_track_points = self.insert_user_activities(user)
        if len(user_track_points) > 0:
            self.track_point_collection.insert_many(user_track_points)
        activities.extend(user_activities)
        count += 1
        print(f"Finished {user['_id']}, count: {count}")

    self.activity_collection.insert_many(activities)
```

Note that the insert_user_activities function creates activities slightly different, as explained in the introduction, saving the total_distance and altitude_gained at creation.

```
return Activity(
            _id = activity_id,
            transportation_mode = labels_activity.transportation_mode,
            start_date_time = string_to_datetime(labels_activity.start_date),
            end_date_time = string_to_datetime(labels_activity.end_date),
            user_id = user_id,
            total_distance = total_distance,
            altitude_gained = altitude_gained
        )
```

Part 2

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

```
Task 2.1
Table           Entries
------------  ---------
Users               173
Activities        16048
Track points    9681756
```

Answer: Due to the fact that users without activities are not stored, the number of users is slightly lower than the original: 173, 16048, 9681756

2. Find the average number of activities per user.

```
Task 2.2
Average amount of activites per user: 92.76300578034682
```

Answer: The average number of activities per user is less due to the lower number of users: 92.76.

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

Answer: Since we save the num_activties on insertion, this is easily read by sorting on num_activities.

```
Task 2.3
  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
      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: Found using distinct selection on transportation_mode taxi: [10, 58, 62, 78, 80, 85, 98, 111, 128, 163]

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

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: Using the aggregate function with a match on not None values of transportation mode, grouping by the count, we can present the transportation mode count.

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

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

Answer: The year with the most activities is 2008

```
Task 2.6 a)
  Year    Number of activities
------  ----------------------
  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 numberOfHours can be represented by the datediff of startDate, endDate in seconds, converted to hours. The top year now being 2009.

```
Task 2.6 b)
  Year    Number of hours
------  -----------------
  2009          11612.4
  2008           9200.8
  2007           2315.42
  2010           1388.73
  2011           1132.35
  2012            711.213
  2000              0.0511111
```

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: 115.47465961508004 km
```

Answer: In this task the distance is found by matching on several expressions using $eq, for example: `{"$eq": ["$user_id", "112"]},`
The total distance is then found by grouping the user_ids and taking the sum of the corresponding total distances giving us 115.47 km. This is possible because of the preprocessing done when inserting where we calculate the total distance for each eactivity.

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

```
Task 2.8
  UserId    Altitude meters gained
--------  ------------------------
     128                  650952
     153                  554961
     004                  332036
     041                  240769
     003                  233664
     085                  217643
     163                  205274
     062                  181693
     144                  179442
     030                  175680
     039                  146704
     084                  131161
     000                  121505
     002                  115063
     167                  112974
     025                  109159
     037                   99234.6
     140                   94846.3
     126                   83025.8
     017                   62581.4
```

Answer: Because of the preprocessing done when inserting we can simply sum up the total altitude converted to meters and sort them in descending order and limited top 20 results.

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
092	101
000	101
037	100
084	99
002	98
104	97
034	88
140	86
112	67
091	63
115	58
038	58
042	55
022	55
174	54
142	52
010	50
015	46
101	46
005	45
001	45
052	44
012	43
089	40
028	36
051	36
096	35
036	34
067	33
011	32
044	32
019	31
009	31
134	31
147	30
155	30
007	30

071	29
013	29
179	28
024	27
018	27
082	27
065	26
111	26
029	25
125	25
103	24
035	23
119	22
043	21
016	20
020	20
168	19
074	19
078	19
026	18
073	18
040	17
110	17
006	17
057	16
008	16
081	16
094	16
150	16
083	15
055	15
097	14
181	14
154	14
046	13
058	13
102	13
061	12
032	12
139	12
088	11
099	11
023	11
131	10
138	10
158	9
172	9
157	9
162	9
105	9
169	9

063	8
076	8
130	8
050	8
176	8
056	7
064	7
045	7
146	7
053	7
161	7
021	7
129	6
080	6
122	6
047	6
066	6
075	6
164	6
136	6
069	6
173	5
059	5
135	5
098	5
145	5
086	5
070	5
108	5
159	5
121	4
093	4
175	4
124	4
133	4
127	4
095	4
123	3
132	3
106	3
117	3
109	3
171	3
087	3
090	3
031	3
118	3
114	3
100	3
077	3

Answer: This is done by grouping all track_points on their activity_id and then adding each track point for sed activit id as an array on the result, we then use a reduce function to calculate if the activity is valid or not, and then group this result on user id while summing up the total number of invalid activities for the given user.

180	2
027	2
152	2
079	2
165	2
054	2
166	2
033	2
170	2
072	2
107	1
060	1
151	1
048	1
113	1
141	1

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

Answer: Finding the longitude and latitude within the coordinate intervals. The activity ids are matched to the user ids and listed. The unique users are 18 and 19.

```
Task 2.10
 User_id
---------
     019
     018
```

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

Answer: To find the most used transportations modes per user, we grouped the result by user_id. For the user the max value of the given number of activity for each mode is retrieved.

```
Task 2.11
 UserId  Mode
--------  ------
    010  taxi
    020  bike
    021  walk
    052  bus
    056  bike
    058  walk
    060  walk
    062  walk
    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  walk
    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

This was an instructive exercise, where we got to consider the different approaches using SQL based and document based databases. Due to the differences which will be discussed in the next task, we chose to deviate somewhat from the solution in exercise 2. Mainly in the form of adding total distance to the activity, and considering only users with activities registered. There are obviously pros and cons with the different approaches, but the increase in efficiency made it a worthwhile sacrifice for the sake of the exercise. In addition, the different approaches lead to greater outcome of learning, in our opinion.

MySQL vs MongoDB

Discuss the differences between MySQL and MongoDB (relational databases vs NoSQL databases). Which database did you prefer to solve these tasks? What are the pros and cons using one versus the other?

The main difference between the relational databases, and the NoSQL databases, is naturally the manner in which the data is stored and combined/joined. In exercise two, the definition of the schema was an essential part of the implementation, in MongoDB it is much easier and more intuitive to model and store the data together as a chunk in the database. This is done through creating documents, containing the data for the different tables. MySQL is also a more rigid database system meaning that it's less flexible when it comes to making changes to the structure of the data. MongoDB allows the user to change the structure of the data because there aren't any complex constraints that cannot be violated. Because of the unstructured nature of MongoDB it is harder to perform queries which combine relational data. MySQL is also an older system and the documentation is harder to interpret and find than it is for MongoDB. The unstructured nature of MongoDB also makes data integrity hard, because you cannot easily make sure that a relationship between documents is upheld through updates of collections, although this also makes it easy to insert documents with references without having to insert them in a certain order.

On the other hand, we considered the retrieval of data from different data that do not naturally co-exists to be a slightly more difficult task. To combat this, we did some more preprocessing of the data, both to make the operations themselves easier, but also to provide faster reads when retrieving the data. The cost of this is that more data needs to be stored, for example with total_distance being stored as a field in activity. Due to the large computational cost of joining activities on track_points and calculating geo data, this was deemed beneficial.

What we preferred to solve the task is not inherently apparent. Each approach provides its own upsides and downsides. Insertion is quite a bit easier with MongoDB, due to the less strict requirements to relations, it was also naturally much faster to write. This was a bottleneck in the first task with MySQL. While this was more efficient, joining on not co-existing data proved quite a bit more difficult, as well as slower when retrieving large amounts of data. Aggregating the data, and extending the collections to include more fields was an alternative making it easier to complete the task. We do however recognize the drawbacks of this, as described earlier. In a broader context we would prefer to use MySQL databases on large volumes of distributed data, for the ability to have constraints across tables, and the added schema-security. In addition, we did design the MongoDB-"schema" in this assignment specifically for the given tasks. The aggregate operation in MongoDB was impressive and made it easy to create a pipeline of operations to be runned, and it was easier to write compared to SQL. But when we tried to use "$lookup" in an aggregate operation to include data from a different collection, the performance dropped so much that we found it much smarter to duplicate this other data instead of using lookup. MySQL would not have had this performance-issue.

In a real world case where the tasks/operations may be unknown in advance, we think that a relational database would in most cases be the most future proof solution. This is for example because of how duplicating data is a common practice in document-databases and a change of structure can require a lot of updates that aren't necessarily straight-forward. While in relational databases, data is stored once and retrieved by relations which means that changes in structures don't require other data updates.