NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4225 - Very Large, Distributed Data Volumes

Assignment 3 - MongoDB

Group:
6

Students:
Tord Johan Espe
Sivert Eggen
Kristoffer Grude

27.10.2023

1 Introduction

1.1 Intro to task and dataset

In this project, we leveraged the modified Geolife GPS Trajectory dataset from Microsoft, encompassing 182 users and a total of 18,669 tracked activities. Each activity contained multiple GPS trackpoints. Our tasks for this NoSQL-based data analysis using MongoDB were:

1. Cleaning the dataset according to provided guidelines and inserting data into the appropriate MongoDB collections. We followed the recommended schema and introduced a few pre-processed properties calculated on insertion to boost performance and simplify queries.
2. Crafting queries using MongoDB and Python to extract insights from the data, followed by visual representation of the query results.

1.2 Assumptions

1. Users without activity labels were included in our MongoDB collections, meaning we analyzed both labeled and unlabeled user data.
2. We observed that some trajectory files for labeled users, contained trackpoints spanning multiple activities dated the same day. This was inferred from overlapping start and end times. For these users, only activities aligning perfectly with their labeled activities were included, but there could be several activities within each *.plt* file.
3. For unlabeled users we assumed each *.plt*-file represented a singular activity. Therefore, our database houses activities from both labeled and unlabeled users.
4. For the query that finds users that have tracked an activity in the Forbidden City, we counted a user as having been in the Forbidden City if they have trackpoints with coordinates that begin with the decimals provided in the assignment, i.e. $39.916 \leq \text{latitude} < 39.917$ and $116.397 \leq \text{longitude} < 116.398$.

1.3 Method

For this assignment, our approach centered around collaborative group work and pair programming, ensuring continuous alignment among team members. While tasks were somewhat distributed, it was essential that each member participated in every phase, as part 1 and 2 depended on each other.

We initiated the project by adhering to the provided setup guide, and created a Git repository for which the link is provided below.

1.4 Link to github

https://github.com/tjespe/bigdata/tree/master/assignment3

2 Results

2.1 Part 1

2.1.1 Database and collection structure

Table 1 & 2 shows the collection structure and object structure in the database and its document fields.

Figure 1: users and activities collection stucture

users	activities
_id	_id
has_labels	transportation_mode
	start_date_time
	end_date_time
	trackpoints: {array of trackpoint objects}

Figure 2: trackpoint object structure

trackpoint
lat
lon
altitude
date_days
date_time
altitude_diff
minutes_diff
meters_moved

2.1.2 Screenshots of database collections and documents

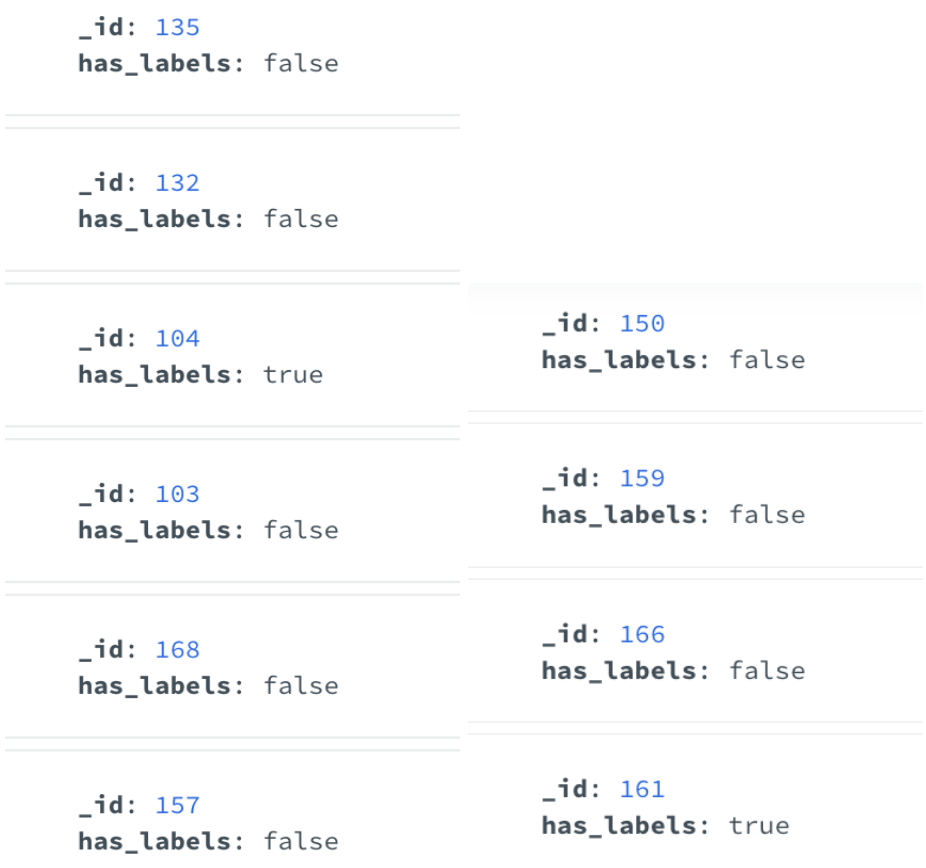

Figure 3: Top 10 users in the User collection

Figure 4: Top 10 activities in the Activity collection

Figure 5: One Activity document with the first 10 associated TrackPoints

2.2 Part 2

The following section provides the results outputted for each of the 11 queries in the assignment. All queries and code used to produce this result can be found in the code files submitted alongside this report. Each query is also included in the Appendix.

2.2.1 Number of users, activities and trackpoints in the dataset

```
Number of users: 182
Number of activities: 15641
Number of trackpoints: 8471126
```

Figure 6: Result from query for task 1

2.2.2 Average number of activities per user

```
The average number of activities per user:  85.93956043956044
```

Figure 7: Result from query for task 2

2.2.3 Top 20 users with the highest number of activities

Top 20 users with the most activities:

User ID	Number of activities
85	1086
153	975
68	920
128	875
167	808
25	715
62	539
126	418
84	411
10	402
41	399
163	354
4	346
140	345
179	309
52	282
17	265
3	261
14	236
30	210

Figure 8: Result from query for task 3

2.2.4 All users who have taken a taxi

User ID			
			84
85	175	75	80
126	62	58	52
161	111	153	105
163	78	82	114
65	179	21	128
102	10	154	56
167	98	68	139

Figure 9: Result from query for task 4

2.2.5 All types of transportation modes and how many activities that are tagged with that transportation mode

Number of activities per transportation mode:

Transportation mode	Count
walk	3924
bus	1822
bike	1515
car	755
subway	613
taxi	512
train	134
airplane	14
boat	7
run	4
motorcycle	2

Figure 10: Result from query for task 5

2.2.6 Year with the most activities and year with the most recorded hours

```
The year with the most activities is:  2008  with  7867  activities!
The year with the most activityhours is:  2009  with  9788.809722222222  total hours!
```

Figure 11: Result from query for task 6

2.2.7 Total distance walked in 2008 by user with id=112

```
Total distance walked by user 112 in 2008: 224.10167184132897 km
```

Figure 12: Result from query for task 7

2.2.8 Top 20 users that have gained the most altitude

Top 20 users with the most altitude meters:

Rank	User id	Total altitude meters
1	4	332036
2	128	266856
3	85	259145
4	41	240769
5	3	233664
6	30	175680
7	39	146704
8	62	144913
9	84	132445
10	167	130201
11	0	121505
12	2	115198
13	153	112922
14	25	109159
15	37	99234.6
16	140	94846.3
17	52	81623
18	17	62581.4
19	34	61430.5
20	42	61332.1

Figure 13: Result from query for task 8

2.2.9 All users that have invalid activities and number of invalid activities per user

164 users have invalid activities

User ID	Number of Invalid Activites
128	263
25	263
85	259
4	219
41	201
62	185
3	179
84	176
68	174
153	160

Figure 14: Result from query for task 9

Only the top 10 results are shown in Figure 14, but in total there were 164 users with invalid activities.

2.2.10 Users that have tracked an activity in the Forbidden City of Beijing

Users who have tracked an activity in the Forbidden City:

User ID
19
18

Figure 15: Result from query for task 10

2.2.11 All users with registered transportation_mode and their most used transportation_mode

User ID	Most used transportation mode
10	walk
20	bike
21	car
52	bus
53	walk
56	bike
58	walk
60	walk
62	bus
64	walk

Figure 16: Result from query for task 11

Only the first 10 results are shown in Figure 16, but in total there are 64 results.

3 Discussion

3.1 Part 1

When defining a structure for the database, the specific use cases were considered. In most of the use cases, both activity data and trackpoint data is used, and the queries are usually focused on the activities. Thus, the database is centered around activity documents, with trackpoints as subdocuments. This enables simple querying of activities and analysis of the associated trackpoints.

The only differences we made to setup part was to add some additional fields to the trackpoint objects under insertion:

- **altitude**: Altitude converted from feet to meters
- **altitude_diff**: The difference between current altitude and previous altitude for consecutive TrackPoints
- **meters_moved**: Haversine distance moved between (lat, lon) and previous (lat, lon) for consecutive TrackPoints in meters
- **minutes_diff**: Time difference since last TrackPoint in minutes

These fields where added to make some of the queries easier to perform.

3.2 Part 2

For all tasks except query 7, 8 and 9, we used only basic NoSQL queries. All queries can be found in the appendix. As we did in assignment 2, we did some calculations in advance, before creating the database and added it as fields to the collection schema. It was easier to do the complex computations in advance in python instead of in NoSQL, in addition to making the queries much faster. We did the following for queries 7, 8 and 9:

- **Query 7**: For this query we introduced a "meters_moved" field for each TrackPoint that indicates the distance moved since the last TrackPoint in the activity. The reason for this was that by calculating this using python code during insertion, we could achieve linear time queries using NoSQL, by eliminating the need for comparing all trackpoints pairwise.
- **Query 8**: To easily calculate the altitude gain for different users, we decided to include an "altitude_diff" field in the TrackPoint objects. This field is the the altitude difference between consecutive TrackPoints which made it possible to easily calculate total altitude gain by summing the positive differences for each user. This approach made it possible to not compare trackpoints directly for each user in NoSQL, making the process more efficient and the queries simpler.
- **Query 9**: Again, for this query we decided to add a field before insertion called "minutes_diff" to trackpoints to improve query speed end simplify the NoSQL queries. The "minutes_diff" field represents the minute difference between consecutive TrackPoint timestamps and made it possible to identify invalid activities by just checking if "minutes_diff" is greater than 5 in our queries, avoiding the need for the time consuming process of pairwise comparisons.

3.3 What we learned

In this assignment we learned first hand the significant difference in flexibility in schema changes offered by the document-model compared to the relational model. MongoDB's schema-less approach allows for easy modification to the structure without impacting the existing data. More about this is in section 4.

4 MySQL vs. MongoDB

In this section, we delve into the inherent differences between MySQL and MongoDB. While both databases serve specific purposes, understanding their nuances, strengths, and limitations can better inform database selection for various tasks.

4.1 Differences between MySQL and MongoDB

Data Storage: One of the primary distinctions between MySQL and MongoDB is how they store data. MongoDB employs a document-based approach, using JSON-like documents to store data. In contrast, MySQL, being a relational database system, organizes data into structured tables with rows and columns.

Schema Flexibility: MongoDB offers more flexibility when it comes to data storage, as it's not as stringent as relational databases like MySQL. In MySQL the scheme has to be predefined. This means that in MongoDB developers can easily incorporate varied data formats without having to redefine the schema, but in MySQL the schema has to be redefined each time.

Syntax: The syntax between the two databases varies significantly, with MongoDB using a more JSON-oriented syntax, while SQL has its own query language.

Data Relations: In MySQL, relationships between data sets often require joining tables, which can sometimes be complex. On the other hand, MongoDB, due to its document-oriented nature, often encapsulates related data within a single document, eliminating the need for joins in many cases.

4.1.1 Pros and cons using one versus the other

Setup Speed: Establishing a database in MongoDB is notably faster, taking roughly 5 minutes. In contrast, setting up in SQL can take upwards of 30 minutes.

Schema Strictness: As previously highlighted, MongoDB is more lenient in its data storage, which can be seen as both a pro and a con. It provides flexibility, but this same flexibility can sometimes lead to inconsistent data representations. The MySQL strictness can be an advantage when rules are easily predefined, while the flexibility in MongoDB is advantage when regular schema changes can occur.

Data Relations: An advantage of MongoDB is the elimination of the often cumbersome process of joining tables, as is frequently required in SQL. However, when having many-to-many relationships, the document model-not having specific support for joins makes these relations harder to find and query. In this exercise, there was no many-to-many relationships to account for, but rather an hierarchical and nested structure, for which MongoDB and the document-model works fine.

4.1.2 Our preferred database to solve these tasks

Upon weighing the strengths and limitations of both databases, our team leans towards MongoDB, primarily due to the schema flexibility and efficiency this provides. In the previous assignment working with MySQL, we did several schema changes, which was a longsome process due to the strictness to specify a scheme and having to reinsert data. For MongoDB this was easier, and for example when changing structure to the activity collection or trackpoints objects, we did not have to do anything about the user collection and the data could stay in the database. Although many-to-many relations is harder to handle in the document-model, they were not existent for the assignment and the hierarchical structure of our data suits the document-model well.

When it comes to query language we prefer SQL to the syntax of MongoDB, as the syntax was better known and used by our team members.

5 Feedback

We enjoyed the exercise of trying a different database model on the same exercises and tasks. This gave hands-on understanding of how changes in data model impact our work. Also in this assignment, it was a good mix of easy and challenging exercises. It kept the learning process engaging. As we wrote in assignment 2, instruction on how to set up the database was very good and clear. The query questions was in this assignment also more clear and easy to understand without a lot of interpretation. However, as for assignment 2 the descriptions of how trackpoints and activities are connected and how they should be created was a bit unclear and not that easy to interpret.

Appendix

A Query 1

```
pipeline = [
        {"$project": {"trackpoint_count": {"$size": "$trackpoints"}}},
        {"$group": {"_id": None, "trackpoint_count": {"$sum":
        ↪  "$trackpoint_count"}}},
    ]
trackpoint_count =
↪  db["activities"].aggregate(pipeline).next()["trackpoint_count"]
```

B Query 2

```
user_count = db["users"].count_documents({})
activity_count = db["activities"].count_documents({})

print("The average number of activities per user: ", activity_count / user_count)
```

C Query 3

```
query = db.activities.aggregate(
        [
            {"$group": {"_id": "$user_id", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            {"$limit": 20},
        ]
    )
```

D Query 4

```
pipeline = [
        {"$match": {"transportation_mode": "taxi"}},
        {"$group": {"_id": "$user_id"}},
    ]

output = db["activities"].aggregate(pipeline)
```

E Query 5

```
query = db.activities.aggregate(
        [
            {"$match": {"transportation_mode": {"$ne": None}}},
            {"$group": {"_id": "$transportation_mode", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
        ]
    )
```

F Query 6

```
query = db.activities.aggregate(
        [
            {
                "$addFields": {
                    "duration": {
                        "$divide": [
                            {"$subtract": ["$end_time", "$start_time"]},
                            3600000,
                        ]
                    }
                }
            },
            {
                "$group": {
                    "_id": {"$year": "$start_time"},
                    "hours": {"$sum": "$duration"},
                }
            },
            {"$sort": {"hours": -1}},
            {"$limit": 1},
        ]
    )
```

G Query 7

```
pipeline = [
        {
            "$match": {
                "transportation_mode": "walk",
                "start_time": {
                    "$gte": datetime.datetime(2008, 1, 1),
                    "$lt": datetime.datetime(2009, 1, 1),
                },
                "user_id": user_id,
            }
        },
        {"$unwind": "$trackpoints"},
        {"$match": {"trackpoints.meters_moved": {"$gt": 0}}},
        {
            "$group": {
                "_id": None,
                "total_distance_2008": {"$sum": "$trackpoints.meters_moved"},
            }
        },
    ]

output = db.activities.aggregate(pipeline)
```

H Query 8

```
query = db.activities.aggregate(
        [
```

```
            {"$unwind": "$trackpoints"},
            {"$match": {"trackpoints.altitude_diff": {"$gt": 0}}},
            {
                "$group": {
                    "_id": "$user_id",
                    "total_alt": {"$sum": "$trackpoints.altitude_diff"},
                }
            },
            {"$sort": {"total_alt": -1}},
            {"$limit": 20},
        ]
    )
```

I Query 9

```
pipeline = [
        {"$unwind": "$trackpoints"},
        {"$match": {"trackpoints.minutes_diff": {"$gte": 5}}},
        {"$group": {"_id": "$user_id", "num_invalid_activities": {"$sum": 1}}},
        {"$sort": {"num_invalid_activities": -1}},
    ]

output = db["activities"].aggregate(pipeline)
```

J Query 10

```
pipeline = [
        {"$unwind": "$trackpoints"},
        {
            "$match": {
                "trackpoints.lat": {
                    "$gte": coordinates_of_the_forbidden_city["lat"],
                    "$lte": coordinates_of_the_forbidden_city["lat"] + 0.001,
                },
                "trackpoints.lon": {
                    "$gte": coordinates_of_the_forbidden_city["lon"],
                    "$lte": coordinates_of_the_forbidden_city["lon"] + 0.001,
                },
            }
        },
        {"$group": {"_id": "$user_id"}},
    ]

output = db.activities.aggregate(pipeline)
```

K Query 11

```
cursor = db.activities.aggregate(
        [
            {"$match": {"transportation_mode": {"$ne": None}}},
            {
                "$group": {
                    "_id": {
                        "user_id": "$user_id",
```

```
                    "transportation_mode": "$transportation_mode",
                },
                "count": {"$sum": 1},
            }
        },
        {"$sort": {"count": -1}},
        {
            "$group": {
                "_id": "$_id.user_id",
                "transportation_mode": {"$first":
                ↪  "$_id.transportation_mode"},
            }
        },
    ]
)
```