NTNU

Assignment 3

TDT4225 - Very Large, Distributed Data Volumes

Deadline: Oct 31 at 14:00

Introduction

In this assignment you'll be working individually or in groups of up to 3 students to solve some database tasks using the MongoDB database **(version 7.0.1)** and coding in Python.

This exercise will be graded and counts 25% of your final grade. The grading will be finished within 3 weeks of the delivery deadline.

Groups 1-80 have access to a virtual machine at IDI's cluster, running Ubuntu. This machine will have MongoDB already installed and set up for you to use. The other groups may run MongoDB themselves, e.g. using docker. The virtual machines have limited power, so a regular PC is more powerful.

This assignment will look at an open dataset of movies, and your task is to perform exploratory data analysis (EDA), design tables, clean and insert the data, and then make queries to answer a set of questions. Your Python program will handle these steps.

Along with this assignment sheet, you have been given several files:

- Dataset - The Movies dataset.
- `DbConnector.py` - a Python class that connects to MongoDB on the virtual machine.
- `example.py` - a Python class with examples on how to create tables, insert, fetch and delete data in MongoDB.
- `requirements.txt` - a file containing some pip packages that should be used in this assignment.

It is strongly recommended that you read through and understand the whole assignment sheet before you start solving the tasks. There are also some tips at the bottom of this assignment, which could be very handy!

Dataset – The Movies Dataset

In this assignment you will work with *The Movies Dataset* from Kaggle. The dataset contains information about tens of thousands of films, including metadata such as titles, budgets, revenues, ratings, cast, crew, production companies, and keywords. It is designed to give you experience working with both structured and semi-structured data. Many of the fields are nested JSON objects, making it particularly relevant for practicing MongoDB queries.

The dataset is provided as multiple CSV files, some of which contain nested JSON fields that require parsing before use.

Files included

- **movies_metadata.csv** – Main metadata file with 45,000+ movies. Includes title, original title, overview, tagline, status, release date, runtime, budget, revenue, vote average, vote count, genres, production companies, production countries, spoken languages, and optional franchise/collection membership.

- **credits.csv** – Information on cast and crew for each movie. The *cast* field includes actor names, characters, and billing order. The *crew* field lists members with their job and department (e.g., "Director", "Producer").

- **keywords.csv** – Contains plot keywords for movies as an array of keyword objects.

- **ratings.csv** - Contains the full ratings data with 26 million ratings from 270,000 users.

- **ratings_small.csv** – Subset of 100,000 ratings from 700 users on 9,000 movies. Each record includes userId, movieId, rating, and timestamp.

- **links.csv / links_small.csv** – Files that map MovieLens IDs to TMDB and IMDb IDs, allowing joins between ratings and metadata.

Note: For initial testing or development, you may use ratings_small.csv and links_small.csv. However, your final solution and delivery should be designed to handle the full ratings.csv and links.csv datasets.

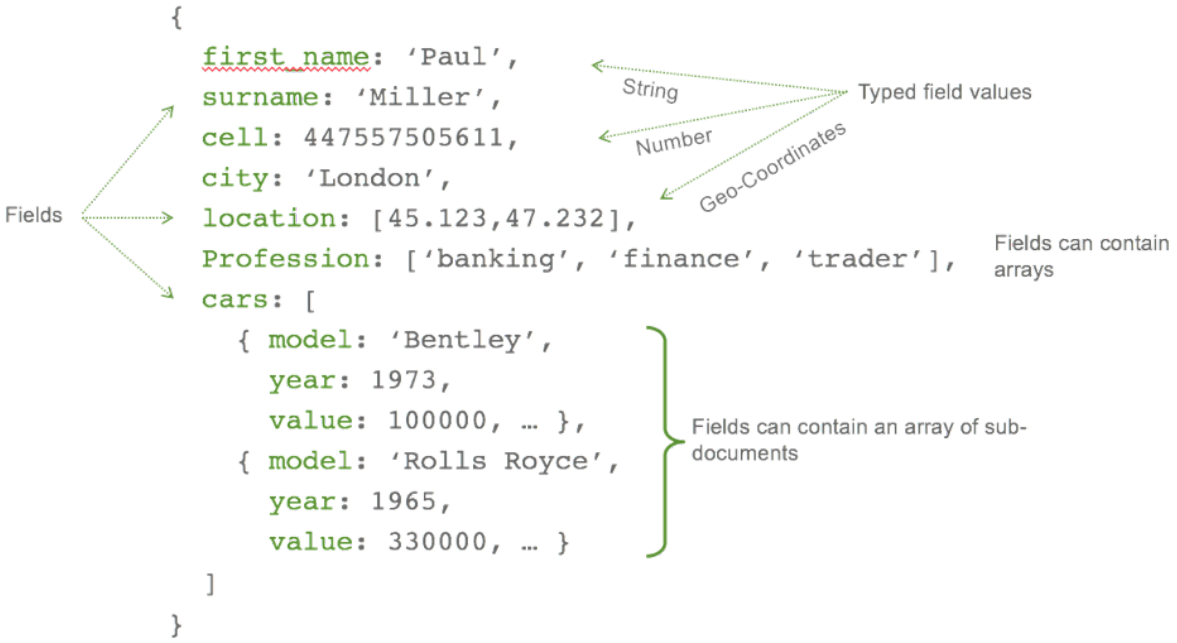
Example fields from movies_metadata.csv

- **id**: TMDB movie identifier (unique key)
- **title**: Movie title
- **original_title**: Title in the original language
- **overview**: Short description of the plot
- **release_date**: Release date of the movie
- **runtime**: Duration in minutes
- **budget**: Production budget in USD
- **revenue**: Worldwide gross revenue in USD
- **vote_average**: Average TMDB user rating (0–10)
- **vote_count**: Number of votes contributing to the rating
- **genres**: Array of objects
- **production_companies**: Array of production company objects
- **production_countries**: Array of country objects with ISO codes
- **spoken_languages**: Array of language objects with ISO codes
- **belongs_to_collection**: Object if the movie belongs to a franchise/series

NTNU

MongoDB - NoSQL

A quick note on MongoDB and NoSQL. There is some terminology that is different from the regular relational-database. MongoDB stores data in documents. Documents are JSON documents based on the JSON specification.
An example of a JSON document would be as follows:



Notice that documents are not just key-value pairs but can include arrays and subdocuments. The data itself can be different data types like geospatial, decimal, string etc. This does not have to be pre-defined, neither does two documents in the same collection have to include the same fields.

A **database** in MongoDB is the container for **collections**. A **collection** is a container for **documents**. This grouping is similar to relational databases and is pictured below:

Relational concept	MongoDB equivalent
Database	Database
Tables	Collections
Rows	Documents
Index	Index

Source: Robert Walters, *'Getting Started with Python and MongoDB'*, 2017.
Url: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

Collections

As mentioned earlier, collections in MongoDB are not as strict as in relational databases. How you decide to organize your collections and documents is up to you. Notice that foreign keys are not really a thing in MongoDB, but it is optional how you want to reference things in each document.

Because MongoDB offers a lot of ways to handle this, we encourage you to investigate what solution that might suit the purpose of this assignment best.
Here's three great articles that we recommend reading before making a decision:
MongoDB Schema Design - Part 1 (5 min reading time)
MongoDB Schema Design - Part 2 (5 min reading time)
MongoDB Schema Design - Part 3 (5 min reading time)

Notice how the id is written as "_id". In MongoDB, there will always be a "_id" as an ObjectID if the field is not specified (read more here). E.g., creating a document like this:

```
Person = {
        id: 1,
        name: 'Name Namesen'
}
```

would result in a Person-document like this when inserted to the database:

```
Person = {
        _id: ObjectID('Some12byteNumber')
        id: 1,
        name: 'Name Namesen'
}
```

You have to manually override the "_id"-field if you want to set a defined ID (e.g. '014' for Users).

Datetime should be consistent throughout your collections, e.g. in the format YYYY-MM-DD HH:MM:SS.

Setup database

These steps will guide you through how to connect to the virtual machine from your computer and create a MongoDB-database user with privileges.

1. First, you need to use NTNU's VPN to access the virtual machines. Log onto NTNU-VPN with your Feide user (check out this link if you have not done this before).

2. Open your terminal and enter the following command:
   `ssh your_username@tdt4225-xx.idi.ntnu.no` where **"your_username"** is the Feide-username and **"xx"** is your group number for this project (01, 02, 03… etc).
   (If this message pops up, enter **yes**:
   `The authenticity of host 'tdt4225-xx.idi.ntnu.no (xxx.xxx.xxx.xxx)' can't be established.`
   `ECDSA key fingerprint is …`
   `Are you sure you want to continue connecting (yes/no)? yes`)
   You will then be asked to enter your password, use the Feide-password here. If the password is correct, you have now successfully logged in!

3. On the virtual machines there would probably be a running MongoDB already. This we need to kill, to be able to start our own MongoDB.
   % ps aux | grep mongo
   % mongodb 834 …… /usr/bin/mongod –config /etc/mongod.conf
   % sudo kill -kill 834

4. We also need to create the database device on the virtual machines.
   % sudo mkdir /data
   % sudo mkdir /data/db

5. When using MongoDB, we have to have a server running in one terminal window. First, type `sudo mongod --repair` (only have to be done the first time you start your server) in your terminal window and type your Feide password. Once that is completed type `sudo mongod --bind_ip_all`. If successful, you have now a running MongoDB-server (the bind_ip_all flag opens for connections from any IP-address, but don't worry, we'll use authentication). This server must be running whenever you are working with the task. Only one of the group members has to run the server at a time.

6. Open a new terminal window and ssh into the virtual machine. Do not close the server-window you have running. To open a client-window for MongoDB type `sudo mongosh`. Now, you have your MongoDB-server running in one terminal window, and a MongoDB-client running in the other. If you check your server, the client-login should've been logged there.

7. Now we want to create an admin MongoDB-user. The MongoDB-client accepts shell commands written in JavaScript.

Start by creating an admin-user by typing the following commands:
`> use admin`, this chooses the admin database

```
> db.createUser({
      user: 'your_username',
      pwd: 'your_password',
      roles: ['root']
})
```

, this creates a user with the root-role. Change your_username and your_password with desired names. The command can be written in one line, it is spread across several lines here for readability. Here is an example where your_username=adminuser and pwd=admin123:

```
> db.createUser({
      user: 'adminuser',
      pwd: 'admin123',
      roles: ['root']
})
```

Now type `> show users;` and find your newly created admin user.

8. Now let's create the user and the database that the group will use during this assignment. To create a database you simply type `> use my_db;` where my_db is your chosen name. If the database does not exist, it will create it for you. Now do the same command as in Step 5, but with a read/write access-role:

```
> db.createUser({
      user: 'your_username',
      pwd: 'your_password',
      roles: ['readWrite']
})
```

Now, your user will have read and write access to the my_db database that you created. Type `> show users;` and find your newly created user, and check that the role is correct and attached to the correct database (my_db not admin). You can also type , but the database will not show until you have created collections and inserted data into it.

9. You can use this client window to create collections etc with the shell commands, but from now on the Python program will be our MongoDB-client. Remember to keep the MongoDB-server running.

Setup Python

We will be using a MongoDB connector to Python (PyMongo) in this assignment. If you for some reason would like to complete the assignment in another language, you are

allowed to do so. ***(NB! We will <u>not</u> provide support or documentation for other languages, so we strongly encourage you to use Python.)***

With the files provided in this assignment, you will find a `requirements.txt`, `DbConnector.py` and `example.py`, among others.

1. To set up the required pip-packages for this assignment, simply run `pip install -r requirements.txt` while you are in the directory with the requirements-file. This will install the connector, along with tabulate (used to print pretty tables in python) and haversine (used to calculate distance between two coordinates).

2. Now, open `DbConnector.py` and look at the code. There will be a TODO there to set up the database correctly with the settings from the database setup. Update the values accordingly:
   Host = tdt4225-xx.idi.ntnu.no, where xx is your group number
   Database = the name you provided for your database in step 6 of the database setup (my_db from the example)
   User = the username provided in step 6, (not the admin user)
   Password = the password provided in step 6

3. Open `example.py`, the file has a main method that creates a connection to the database from DbConnector, creates a table named "Person", inserts some names in the database, displays the data, and then drops the table.
   Try to run the code. If everything is set up correctly, you will establish a connection to the database-server and insert/delete data. You can use this file for inspiration when solving the tasks in this assignment.

Tasks

The tasks are divided into three parts. Part 1 will focus on exploratory data analysis (EDA), where you analyze and clean the Porto dataset before inserting it into the database. Part 2 will then involve writing queries to the database in order to gain insights from the data. Finally, Part 3 will focus on writing a report where you discuss your answers and reflect on your findings.

We recommend looking at the documentation for the MongoDB-Python connector before you start coding.

Part 1

In this task you will begin by performing exploratory data analysis (EDA) on the Movies dataset. The goal is to understand the structure and quality of the data, identify potential issues such as missing values, and prepare the dataset for insertion into your own MongoDB database.

1. Through EDA you should examine the main attributes to gain an overview of the data distribution, and investigate the presence and impact of missing data. The insights you gather here will help guide how you clean and organize the data for later analysis.
2. Once you have a clear understanding of the dataset, you should design and create the MongoDB database and insert the cleaned data. You are free to adapt the structure as you see fit, but do explain your reasoning in the report.

Note: This part is deliberately kept open-ended to encourage exploration. You are expected to include all findings you consider relevant in your final submission.

Part 2

Some of the tasks can be answered using MongoDB-queries only, while some might require both queries and Python code to manipulate the data correctly. Use pprint to pretty-print the data output from MongoDB.

Answer the following questions by writing a Python program using MongoDB-queries (like in `example.py`):

1. Considering only crew with job = *Director*, which 10 directors with ≥ 5 movies have the highest median revenue?
   Also report each director's movie count and mean *vote_average*.

2. List actor pairs have co-starred in ≥ 3 movies together, their number of co-appearances, and their average movie *vote_average*. Sort by number of co-appearances.

3. List the top 10 actors (with ≥ 10 credited movies) that have the widest genre breadth. Report the actor, the number of distinct genres they've appeared in, and up to 5 example genres.

4. For film collections (*belongs_to_collection.name* not null) with ≥ 3 movies, which 10 collections have the largest total revenue?
   For each, report movie count, total revenue, median *vote_average*, and the earliest → latest release date.

5. By decade (e.g., 1970s, 1980s, …) and primary genre (first element in *genres*), what is the median runtime and movie count?
   List results sorted by decade then median runtime (desc).

6. For each movie's top-billed 5 cast (by *order*), what is the proportion of female cast?
   Aggregate by decade and list decades sorted by average female proportion (desc), including movie counts used. Unknown gender may be ignored.

7. Using a text search over *overview* and *tagline*, which 20 movies matching "noir" or "neo-noir" (and *vote_count* ≥ 50) have the highest *vote_average*?
   Return title, year, *vote_average*, and *vote_count*.

8. Which 20 director–actor pairs with ≥ 3 collaborations (same movie) have the highest mean *vote_average*, considering only movies with *vote_count* ≥ 100?
   Include the pair's films count and mean revenue.

9. Among movies where *original_language* ≠ "en" but at least one production company or country is United States, which are the top 10 original languages by count?
   For each language, report the count and one example title.

10. For each user, what is their ratings count, population variance of ratings, and number of distinct genres rated?
    List the top 10 most genre-diverse users, and separately the top 10 highest-variance users (only users with ≥ 20 ratings).

Part 3

Write a short report (see `report-template.docx` in Blackboard exercise 2 where you will display and discuss your results from the tasks. Include screenshots from both part 1 (EDA) and part 2 (all the results to each task).

Hand in both the report (as PDF) and your code to BlackBoard within Oct 31, at 14:00.

Tips

- Using the MongoDB terminal on your Ubuntu machine could be useful for checking simple queries without having to use Python and the connector.
    - Be sure to have a server running, `sudo mongod --bind_ip_all`
    - Open the MongoDB client terminal by typing `sudo mongo`, log in with Feide-password and type `use name_of_db` to do this.
- Using dictionaries/hashmaps are faster for lookups than lists/arrays in your Python code
- Use some time to consider the different choices for how to store your data.
    - Embedded documents, one-way referencing or two-way referencing.
- Remember to insert data with the right type. Insert integer as int, date as some dateformat, etc..
    - MongoDB support e.g ISODate
- Comparison between SQL and MongoDB.
- Comparison between SQL-functions and MongoBD-functions.
- Aggregation and the pipeline stages for this are worth checking out.