Lab03

- Kappa -

Von Bauer, Prütz, Thomondy, Weidele

Setup

Create a docker image with a MongoDB Server:

sudo docker run -d --rm --name mymongo -p 27017-27019:27017-27019 mongo:latest

Import data with Python script:

```python
import sys
import os
from datetime import datetime, timedelta
from pymongo import MongoClient

BATCH_SIZE = 100_000


# Check args
if len(sys.argv) < 3:
    print("Eror: No filename to import provided")
    sys.exit(1)

# Get filename from sysargs
path = sys.argv[1]
movie_path = sys.argv[2]

if not os.path.exists(path) or not os.path.exists(movie_path):
    print("Error: Invalid filename")
    sys.exit(1)

rows = []
movie_rows = []

batch_counter = 1

client = MongoClient("mongodb://localhost:27017")
db = client.movies

print("!!! Clearing data in the titles collection !!!")
db.titles.delete_many({})

with open(movie_path, "r", encoding = "ISO-8859-1") as f:

    line = f.readline()

    while line:

        if not line:
            break

        line_ar = line.rstrip().split(",")

        if str(line_ar[1]) == "NULL":
            movie_rows.append(
                {"_id": int(line_ar[0]), "title": str(line_ar[2])})
        else:
            movie_rows.append({"_id": int(line_ar[0]), "year": int(
                line_ar[1]), "title": str(line_ar[2])})
        line = f.readline()

print("Inserting data innto db...")
result = db.titles.insert_many(movie_rows)
print("Inserted " + str(db.titles.count_documents({})) + " title documents")

print("!!! Clearing data in the reviews collection !!!")
db.reviews.delete_many({})

with open(path, "r") as f:
    while True:
        if batch_counter % BATCH_SIZE == 0:
            print("Processed " + str(batch_counter) + " lines")
            print("Writing to mongodb...")
            result = db.reviews.insert_many(rows)
            rows = []

        line = f.readline()
        if not line:
            break

        line = line.rstrip().split(",")
        date_of_rating = datetime.strptime(line[3], '%Y-%m-%d')

        rows.append({"movie_id": int(line[0]), "userd_id": int(
            line[1]), "rating": int(line[2]), "time": date_of_rating})
        batch_counter += 1

print("Inserted " + str(db.reviews.count_documents({})) + " documents")
print("Done")
```

Tasks

a) Compute the average rating, lowest rating, highest rating and number of ratings per movie

```
moviegroup = db.reviews.aggregate([
    { '$group':
        { '_id': "$movie_id",
            "count":
                { '$sum' : 1 },
            "average":
                { '$avg': '$rating'},
            "min":
                { '$min': '$rating'},
            "max":
                { '$max': '$rating'}
        }
    },
    {"$sort":  { "_id": 1}},
        {'$lookup': {'from': 'titles', 'localField': '_id', 'foreignField': '_id', 'as': 'title'}},
        {'$project': {'overall': 1, 'title.title': 1, 'çount': 1, 'average': 1, 'min': 1, 'max': 1,
'_id': 0}}
] )

for group in moviegroup:
    print(group)
```

b) Get the movie with the lowest overall rating. If the movies are tied for rating, take the movie with the lowest movie_id

```
lowestrating = db.reviews.aggregate(
    [
        {'$group':
         {'_id': '$movie_id',
          'overall':
          {'$avg': {'$sum': '$rating'}}
          }},
        {'$sort': {'overall': 1, '_id': 1}},
        {'$limit': 1},
        {'$lookup': {'from': 'titles', 'localField': '_id', 'foreignField': '_id', 'as': 'title'}},
        {'$project': {'overall': 1, 'title.title': 1, '_id': 0}}
    ])

for group in lowestrating:
    print(group)
```

{'overall': 1.2878787878787878, 'title': [{'title': 'Avia Vampire Hunter'}]}

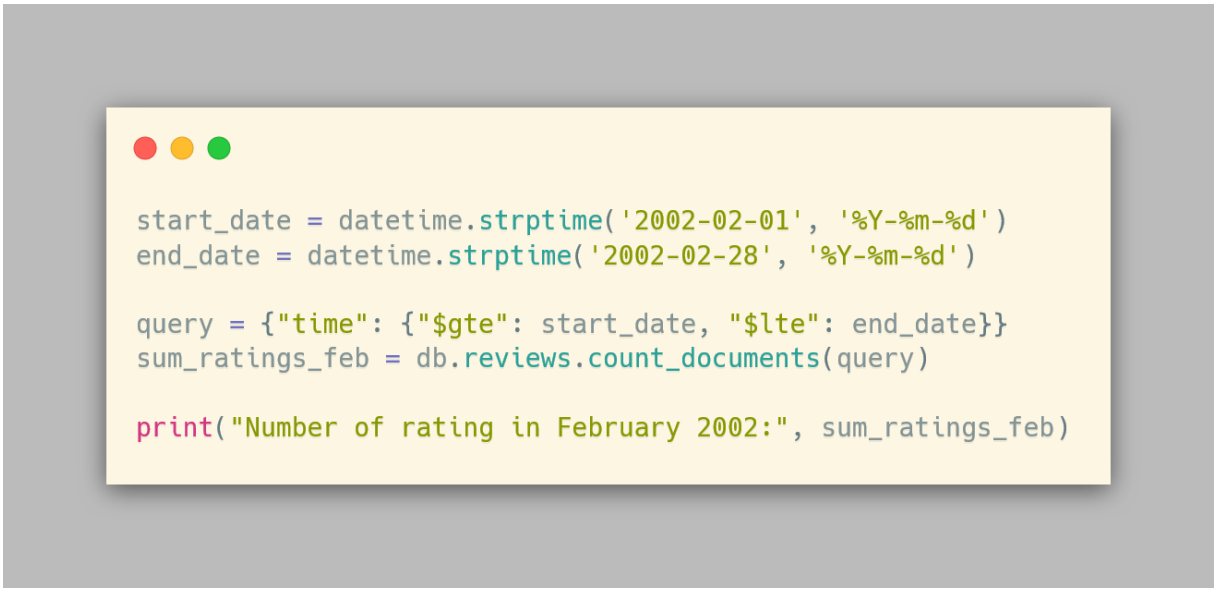
c) Get the movie with the highest overall rating. If the movies are tied for rating, take the movie with the lowest movie_id

```
highestrating = db.reviews.aggregate(
    [
        {'$group':
         {'_id': '$movie_id',
          'overall':
          {'$avg': {'$sum': '$rating'}}
          }},
        {'$sort': {'overall': -1, '_id': 1}},
        {'$limit': 1},
        {'$lookup': {'from': 'titles', 'localField': '_id', 'foreignField': '_id', 'as': 'title'}},
        {'$project': {'overall': 1, 'title.title': 1, '_id': 0}}
    ])

for group in highestrating:
    print(group)
```

{'overall': 4.723269925683507, 'title': [{'title': 'Lord of the Rings: The Return of the King: Extended Edition'}]}

d) Get the number of ratings in February 2002



```
start_date = datetime.strptime('2002-02-01', '%Y-%m-%d')
end_date = datetime.strptime('2002-02-28', '%Y-%m-%d')

query = {"time": {"$gte": start_date, "$lte": end_date}}
sum_ratings_feb = db.reviews.count_documents(query)

print("Number of rating in February 2002:", sum_ratings_feb)
```

Number of rating in February 2002: 261954

e) Get the user that creates the lowest average rating and has rated at least 5 times. If two users are tied, use the one with the lower user_id

```
lowest_user_rating = db.reviews.aggregate(
    [
        {'$group':
         {'_id': '$userd_id',
          'overall':
          {'$avg': {'$sum': '$rating'}}
          }},
        {'$sort': {'overall': 1, '_id': 1}},
        {'$limit': 1}

    ])

print("User that gives the lowest rating: ")
for group in lowest_user_rating:
    print(group)
```

{'_id': 3174, 'overall': 1.0}

f) For the movie “The Spy Who Loved Me”: Get the average rating for each year and month sorted by year and month in ascending order

```
result = db.titles.find({"title": "The Spy Who Loved Me"})

for r in result:
    spy_id = r.get("_id")

average_ratings_over_time = db.reviews.aggregate(
    [
        {'$project': {
            'month': {'$month': '$time'},
            'year': {'$year': '$time'},
            'movie_id': '$movie_id',
            'time': '$time',
            'rating': '$rating'
        }
        },
        {'$match': {
            'movie_id': spy_id}
         },
        {'$group': {
            '_id': ['$year', '$month'],
            'avg': {'$avg': '$rating'}}
         },
        {'$sort': {'_id': 1}}
    ]
)

for r in average_ratings_over_time:
    print(r)
```

{'_id': [2000, 1], 'avg': 3.3666666666666667}

{'_id': [2000, 2], 'avg': 3.5952380952380953}

{'_id': [2000, 3], 'avg': 3.3962264150943398}

{'_id': [2000, 4], 'avg': 3.6481481481481484}

{'_id': [2000, 5], 'avg': 3.2972972972972974}

{'_id': [2000, 6], 'avg': 3.4782608695652173}

{'_id': [2000, 7], 'avg': 4.5}

{'_id': [2000, 8], 'avg': 3.8}

{'_id': [2000, 9], 'avg': 3.6122448979591835}

{'_id': [2000, 10], 'avg': 3.3157894736842106}

{'_id': [2000, 11], 'avg': 3.4318181818181817}

{'_id': [2000, 12], 'avg': 3.6}

{'_id': [2001, 1], 'avg': 3.43}

{'_id': [2001, 2], 'avg': 3.4645669291338583}

{'_id': [2001, 3], 'avg': 3.596774193548387}

{'_id': [2001, 4], 'avg': 3.4285714285714284}

{'_id': [2001, 5], 'avg': 3.1333333333333333}

{'_id': [2001, 6], 'avg': 3.473684210526316}

{'_id': [2001, 7], 'avg': 3.452830188679245}

{'_id': [2001, 8], 'avg': 3.211267605633803}

{'_id': [2001, 9], 'avg': 3.357142857142857}

{'_id': [2001, 10], 'avg': 3.4035087719298245}

{'_id': [2001, 11], 'avg': 3.5}

{'_id': [2001, 12], 'avg': 3.4242424242424243}

{'_id': [2002, 1], 'avg': 3.486842105263158}

{'_id': [2002, 2], 'avg': 3.2666666666666666}

{'_id': [2002, 3], 'avg': 3.2}

{'_id': [2002, 4], 'avg': 3.857142857142857}

{'_id': [2002, 5], 'avg': 3.875}

{'_id': [2002, 6], 'avg': 3.2}

{'_id': [2002, 7], 'avg': 3.769230769230769}

{'_id': [2002, 8], 'avg': 3.0}

{'_id': [2002, 9], 'avg': 3.625}

{'_id': [2002, 10], 'avg': 3.361111111111111}

{'_id': [2002, 11], 'avg': 3.4411764705882355}

{'_id': [2002, 12], 'avg': 3.488372093023256}

{'_id': [2003, 1], 'avg': 3.5142857142857142}

{'_id': [2003, 2], 'avg': 3.4}

{'_id': [2003, 3], 'avg': 3.909090909090909}

{'_id': [2003, 4], 'avg': 3.574468085106383}

{'_id': [2003, 5], 'avg': 3.545045045045045}

{'_id': [2003, 6], 'avg': 3.62751677852349}

{'_id': [2003, 7], 'avg': 3.606779661016949}

{'_id': [2003, 8], 'avg': 3.5727272727272728}

{'_id': [2003, 9], 'avg': 3.7943548387096775}

{'_id': [2003, 10], 'avg': 3.7047970479704797}

{'_id': [2003, 11], 'avg': 3.7598684210526314}

{'_id': [2003, 12], 'avg': 3.6545454545454548}

{'_id': [2004, 1], 'avg': 3.6657824933687}

{'_id': [2004, 2], 'avg': 3.6299019607843137}

{'_id': [2004, 3], 'avg': 3.7971311475409837}

{'_id': [2004, 4], 'avg': 3.749598715890851}

{'_id': [2004, 5], 'avg': 3.8121951219512193}

{'_id': [2004, 6], 'avg': 3.8466666666666667}

{'_id': [2004, 7], 'avg': 3.8508997429305913}

{'_id': [2004, 8], 'avg': 3.839103869653768}

{'_id': [2004, 9], 'avg': 3.9307692307692306}

{'_id': [2004, 10], 'avg': 3.828402366863905}

{'_id': [2004, 11], 'avg': 3.8620102214650767}

{'_id': [2004, 12], 'avg': 3.8952879581151834}

{'_id': [2005, 1], 'avg': 3.849740932642487}

{'_id': [2005, 2], 'avg': 3.913680781758958}

{'_id': [2005, 3], 'avg': 3.9312}

{'_id': [2005, 4], 'avg': 3.9008130081300814}

{'_id': [2005, 5], 'avg': 3.8392101551480957}

{'_id': [2005, 6], 'avg': 3.863128491620112}

{'_id': [2005, 7], 'avg': 3.918918918918919}

{'_id': [2005, 8], 'avg': 3.92625}

{'_id': [2005, 9], 'avg': 3.881491344873502}

{'_id': [2005, 10], 'avg': 3.9315726290516206}

{'_id': [2005, 11], 'avg': 3.958588957055215}

{'_id': [2005, 12], 'avg': 3.822125813449024}

Conclusion:

MongoDB proved to be the ideal solution due to the flexible aggregation pipeline.