Project1 Task1

Nathaniel Leake
424003778
Time to complete Task: **3 hrs**

Part A:

I support the AggieFit team in its decision to use MongoDB over a relational database like MySQL. Because of the dynamic development of the system and high level of variance between different employee accounts, storing individual-pertinent data in JSON-like objects will be more space efficient than having a big SQL table with lots of fields that are blank for most employees, and faster and less complicated than several SQL tables for different user attributes that are linked by id. Having all the data for a user in a single spot while yet being able to potentially assign an arbitrarily large number of fields to any given user is real benefit of MongoDB, and is the primary reason to use it in this case. This is only strengthened by the fact that AggieFit is in its early stages of database development and there is a decent probability of changing the structure in the future, which is much easier to do in MongoDB than in a table-structure database.

Part B:

WQ1:

I ran this one directly in the terminal to import the dummy file into the MongoDB

```
mongoimport --host 34.233.78.56 --db fitness_424003778 --collection fitness_424003778 --file ~/Downloads/dummy-fitness.json —jsonArray
```

WQ2:

To import the fitness data from the user-1001 JSON file, I simply ran the following in the mongo shell:

```
db.fitness_424003778.insert({
  "uid": 1001,"height": "5ft10in",
  "weight": "180lbs",
  "tags": ["ambitious"]
});
```

There was one issue with this, and that was that for some reason the “uid” field ended up being saved as ‘1001.0’ instead of ‘1001’. However, this didn’t seem to break any of the functionality later on.

RQ1:

This one was super-simple. I just did the following in queries.py:

```
pprint.pprint(collection.find().count())
```

RQ2:

This was the first ‘find’ query with any special conditions

```
active_employees = collection.find({"tags": "active"})
```

RQ3:
This one is where it got tricky. Since the 'activityGoal' values are strings and can contain additional text, I decided the best way to find values that are strictly greater than 60 was to build a regex to match for them using MongoDB's $regex query term:

```
gt_60_regex = "([7-9]\d)|(6[1-9])|\d{3,}"
goal_dur_gt_60 = collection.find({"goal.activityGoal": {"$regex": gt_60_regex}})
```

RQ4:
This one was initially challenging, but once I read up on MongoDB aggregates it turned out pretty simple.

```
collection.aggregate([
   { '$unwind': '$activityDuration' },
   { '$group': {
      '_id': '$uid',
      'total_dur': { '$sum': '$activityDuration' }
   }}
])
```

Part C:

What could go wrong if only one server is used? From my viewpoint, the primary concern should be that if anything should go wrong on the server, such as data corruption, crashes, accidental overwrites or queries, then the data could be damaged irreparably. The other concern is that the server could run out of space and be unable to accept new data, have overflow issues, or significantly reduced performance. Spreading out the data over several servers to create a distributed network is a good idea if AggieFit has the necessary funding, but there are also database services online with reliable connectivity, built in redundancy, and scalable storage that in a real scenario would be a better option. Speaking of which, if AggieFit does continue to build its own server network, both of the questions they raised should be answered with "yes". The data should be spread across several servers, but it would also be good to keep 2-3 copies of the data if possible, or to run periodic backups to a separate set of servers.