Lab 3: MongoDB aggregation pipeline

Due date: Friday, April 24, 11:59pm. **Note:** Grace period will extend over the weekend, **but** Lab 4 assignment will be handed out on Friday, April 24, and you may benefit from switching your attention to it.

Lab Assignment

Assignment Preparation

This is an individual lab.

This is a two-part lab. The first part, released to you on Friday, April 17 as a list of 10 queries over the `covid` dataset, is designed to familiarize you with the MongoDB aggregation pipeline and its operations. For the sake of completeness, this part is also featured in this document.

Part 1

This part was assigned on April 17. For the first part of the assignment, you are working with the daily.json file of COVID-19 statistics in the US we introduced in Lab 1. You are asked to create a collection `covid` in your database (please make sure you name it properly - this will make my life easier when I run your queries), and implement all 10 information requests as MongoDB's aggregation pipelines.

See below for information about the format of the files you would need to submit.

The information requests are:

Question 1. Report the cumulative number of positive cases, cumulative number of deaths, and the number of hospitalizations (use hospitalized key) for the state of Illinois. Report the data in chronological order. Easy route: report all nulls. Hard route: if a value of the field is null, remove the field from that object.

Question 2. Compute the percentage of all tests that tested positive (cumulatively) for each day of April in California. For each day, report state id, number of cumulative positive cases, number of cumulative total tests, and the percentage you computed. Report results in chronological order

Question 3. Report the average number of deaths per day during the month of March (for the days available to you) for each of the states. In each document include the code of the state, and the average number of deaths. Report in descending order of the average number of deaths. For this exercise use aggregation on the `deathIncrease` key.

Question 4. Report the number of states that on April 1 had over 5,000 reported positive cases. Report in the format:

```
    {nStates: <number>}
```

Question 5. Find the state with the largest number of tested people on March 15, 2020. Report the state code, and the number of tests. (It looks like you can use `totalTestResults` field for it).

Question 6. Taking Question 5 further, for each date between March 15 and March 31 (inclusive) return the state with the largest number of performed tests. The output documents shall contain the state Id, the date, and the number of performed tests. The documents shall be returned in the chronological order.

Question 7. For each day from March 20 to April 5 (inclusive), find the ratio of new deaths to total deaths for the states of California and New York. Report the two ratios in the same document for each date. The format shall be as follows:

```
{date: <date>,
 ratioCA: <california ratio>,
        ratioNY: <new york ratio>
      }
```

Report results in chronological order.

Question 8. For each day starting March 20 and through April 5 inclusive, report the list of states with 100 or more deaths. Each day shall be represented as a single object, with state codes and numbers of deaths (cumulative) embedded inside an array. The format shall be as follows:

```
  { date: <date>,
    stateList: [ {state:<stateCode>,
                  deaths: <nDeaths>},
                  ...,
                  {state:<stateCode>,
                   deaths: <nDeaths>}
  ]
   }
```

Question 9. For each state classify how well they were doing on April 4. If the total number of deaths is greater than 5% of the total number of positive cases, the state is doing poorly, otherwise the state is doing ok. The output shall have the format:

```
  {state: <stateCode>,
   status: <ok, or poorly>
  }
```

Report results in alphabetical order by state code.

Question 10. This is a bit tricky, but you can do it. For states reported as doing poorly in Query 9, find for how many days they are doing poorly (i.e., how many days in the dataset for that state, the total number of deaths is greater than the 5% total number of positive cases). Report results in the form:

```
{state: <stateCode>,
 nDays: <number of days>
}
```

Report results in alphabetical order by state, and remember - only for the states they were doing poorly on April 4.

Part 2

Part 2 of the Lab contains a number of tasks that need to be performed using the `deb.collection.aggregate()` commands. However, these tasks a more involved, and sometimes include multiple steps.

All tasks shall start with the `covid` collection. For some of the tasks, you will be asked to create additional collections and place the results of certain information needs/requests into those collections.

Task 1. For this task, we want to report the list of all the states where on April 5 there were more positive COVID-19 cases than in the state of New York on March 20. For each state, report its two-letter code, and the number of positive COVID-19 cases on April 5. The output shall have the format

```
{state: <stateCode>,
 positive: <numberPositiveCases>
}
```

Hint. This is essentially a continuation of the types of queries you saw in the first part of the lab. This query shall be executed as a single aggregation pipeline.

Task 2. Now, we want to expand the analysis we did for **Task 1**. For this task, we want to report - for each day from March 25 to April 5 (inclusive) how many states had higher number of positive COVID-19 cases, than the state of NY on March 20. The output shall be presented in the format:

```
{date: <date>,
 nStates: <numberOfStates>,
 states: [<state1>, ...,<stateK>]
}
```

Here, `<numberOfStates>` is the number of states that had on the reported date more positive cases than New York on March 20.

Use a single aggregation pipeline to express this query. Place the results of the query into a collection named `"caseGrowth"`.

Task 3. Let us take a look at the USA-wide daily numbers of positive cases, deaths, and total number of tests administered. Write an aggregation pipeline that produces these three values for each date available in the data we have. The output shall have the format:

```
{date: <date>,
 positive: <totalPositiveCases>,
 deaths: <totalDeaths>,
 tested: <totalTested>
}
```

Note: use cumulative numbers.

The output of this pipeline shall be stored in a collection `covidUsa` in the same database as your `covid` collection.

Task 4. Task 3 was pretty straightforward, but it allows us to work with a simplified dataset that spans the entire USA in subsequent analysis. Let us put this to work. Write an aggregation pipeline that produces the following output for each day:

```
{date: <date>,
 positive: {tomorrow: <nCasesTomorrow>,
            today: <nCasesToday>,
            yesterday: <nCasesYesterday>
            },
 deaths: {tomorrow: <nDeathsTomorrow>,
            today: <nDeathsToday>,
            yesterday: <nDeathsYesterday>
            }
}
```

That is, for each day, include the number of positive cases on the day itself as well as the previous day and the next day. Do the same for the number of deaths. Store the output of this query in a collection called `threeDayWindow`.

Hint. There are three basic ways to combine information from multiple existing documents into a single document: grouping, faceted search, and joins. When using joins, one must remember that MongoDB restricts the join condition to be a single equality of values between a pair of keys (attributes) in the two collections. If one wants a more sophisticated join condition, some advance preparation may be warranted.

Task 5. For this task, you will learn how to simulate a join with an arbitrary join condition. In some cases, this may require additional transformations on the "forein" collection (this is the key difference between how joins are handled in relational algebra and how MongoDB handles them). The key to computing a join under any condition (an not only the equality attributes) is to remember that in relational algebra, join is a compound operation that consists of a cartesian product, followed be a selection, followed (if necessary/desired) by a projection.

Consider the following task: from the `covid` collection extract pairs of states and dates such that, the following conditions are satisfied:

- The two states are different.
- The number of positive cases in one state is within 20 cases of the number of positive cases in another state.

In other words we want to output a collection of documents, each matching a sentence "On date X state Y was within 20 cumulative positive cases of state Z on date W". (this, of course, is somewhat artificial, but not without reason. If we are looking for what will be happening to a specific "late-breaking" state, it is useful to go back in time and find other states that had similar number of cases in the past, and see where their case numbers went.... but the primary purpose of this exercise is for you to learn how to perform an arbitrary join in MongoDB).

Write a single aggregation pipeline that produces the results, and dump the results into a collection called `covidPairs`. Note - you *may need to make some changes to the `covid` collection* in order for this to be feasible. You **may make these changes** in a separate operation. After you complete this task, run a command that reverses those changes.

Each individual document in the new collection shall have the format:

```
{state1: {state: <stateId>,
          date: <date>,
          positive: <positiveCases>
         },
 state2: {state: <stateId>,
          date: <date>,
          positive: <positiveCases>
          }
}
```

To manage this a little bit better, let us postulate that `state1.date` should be the later of the two dates (if the two dates are different). This will resolve most of the duplicates. Any duplicates not resolved by this should be ignored (i.e., don't worry about programmatically eliminating them, although you could always break the ties by ordering the states lexicographically).

Then, write an aggregation pipeline over the `covidPairs` collection that for each state reports how many times it shows up as `state1` in the collection. Report the results in descending order by the count in the format:

```
{state: <stateId>,
 nAppearances: <numberOfAppearances>
}
```

Hint. Figure out how to do cartesian product using `$lookup`. As mentioned above, this may require an extra command (an aggregation pipeline) before the main pipeline is executed.

Task 6. Write an aggregation pipeline that computes the following three results at the end:

- For each state - the total number of days on which the number of deaths exceeded 50. (it is ok to omit states where this number of days is 0).
- The difference between the number of positive cases on March 31 and on March 15 for the states of California, New York, Washington, and Kentucky.
- For each date between March 15 and March 31 (inclusive) - the number of states where the number of new positive cases exceeded 100.

All this needs to be done in a single aggregation pipeline. Unlike other tasks, for this task you are give full freedom on how to format the output, however, please make certain that the output (when pretty-printed) is human-readable. Primarily this means creating appropriate labels (keys) for the output object/objects.

Task 7. Compute two histograms: one for March 15 and one for April 4. Each histogram reports the total number of states with the cumulative number of cases in each of the following ranges: $[0, 100)$, $[100, 500)$, $[500, 1000)$,

$[1000, 5000)$, $[5000, 10000)$, $[1000, 50000]$ and $[50000, 1000000]$. Output both histograms in a way that allows for convenient comparison.

Preparing aggregation pipeline queries for submission

You will create two files: `lab3-queries.mongo` and `lab3-tasks.mongo`. The first file will contain Lab 3-1 queries, the second file - Lab 3 tasks.

Each query and/or task shall be written in plain formatted text and shall be preceded by a Javascript style comment identifying it - at the very least the comment shall state

```
// Query N
```

or

```
// Task N
```

The beginning of each file shall contain a Javascript style comment with your name and email address.

Submission

Submit the following artefacts:

- `README` file with your name and any comments regarding your submission.
- `lab3-queries.mongo` and `lab3-tasks.mongo`

Use `handin` to submit as follows:

```
$ handin dekhtyar lab3 <FILES>
```

(Note: for now assume you are submitting on the CSL `unix` servers).

Good Luck!