CSC 369: Distributed Computing
Spring 2020
Alex Dekhtyar
Lab 4: MongoDB Application

Test Cases
Part 1: Test Cases 01 -- 05

This document is a companion to the test suite that is released to support your work on **Lab 4.** It provides descriptions of each individual configuration file released.

Note: The JSON objects were generated in plain text editor. I did my best to match parentheses, but I will not be surprised if I failed to do this everywhere. If you find a syntax issue with any of the test cases, please report to me via Slack or email, and I will fix the test case for everyone.

This document describes the first five test cases. Additional test cases will be released later.

Test case 01: config01.json

```
{refresh: true,
 collection: "covid",
 aggregation: "state",
 time: {start:20200401, end:20200415},
 target: "CA",
 analysis: {[{task: {track: "positive"},
              output: {graph:{ type: "line",
                               legend:"off",
                               combo:"combine"}
                       }
             }
             ]
           }
}
```

This configuration file contains one information request:

"Report the cumulative numbers of positive COVID-19 cases in the state of California for the time period from April 1 to April 15 as a line graph"

This request should generate the following mongoDB aggregation pipeline:

```
db.covid.aggregate({$match: {state: "CA"}},
                   {$match: {date: {$gte: 20200401, $lte: 20200415}}},
                   {$project: {_id:0, positive:1, date:1}},
                   {$sort: {date:1}}
                  )
```

How the pipeline is generated.

1. The `collection` parameter value sets the `db.covid.aggregate(...)` call.
2. The `target` parameter value sets the first `$match` stage.
3. The `time` parameter sets the shape of the second `$match` stage
4. The `task.track` value in the only task in the configuration document tells us to track the cumulative number of positive cases, represented in the `covid` collection in the `"positive"` field. Our independent variable is time, so we track `"date"` field as well.
5. The $sort stage is a convenience stage to make sure that the data is returned to your application in a meaningfully ordered way. This stage should be used any time a time series is returned as a result of your query (this is controlled by the value of the `time` parameter).
6. The aggregation level is obviated by the fact that data for only one state is requested. Therefore the output would be the same for any allowable aggregation level. See additional test cases to see the effects of the aggregation level on the query generated.

Expected Result. Only a simple graph has been requested. The graph can be generated using `myplotlib.pyplot.plot()` from the data retrieved.

Note. This is not the only possible pipeline that does the trick. If you decide that you prefer a single object with arrays of values for number of cases and dates, you can generate the following pipeline:

```
db.covid.aggregate({$match: {state: "CA"}},
                   {$match: {date: {$gte: 20200401, $lte: 20200415}}},
                   {$project: {_id:0, positive:1, date:1}},
                   {$sort: {date:1}},
                   {$group: {_id:"CA",
                             positiveArray:{$push:"$positive"},
                             dateArray: {$push: "$date"}
                            }
                   }
                  )
```

In discussions of additional test cases, I will use the query style that ends with the `$sort` stage. If you prefer to finish your query with this grouping stage, or with some other reformatting of the output (or with adding additional data), adjust your query generation accordingly.

Test Case 02: config02.json

```
{refresh: true,
 collection: "covid",
 aggregation: "state",
 target: "CA",
 analysis: {[{task: {track: "death"},
              output: {table:{ row: "state",
                               column:"time",
                               title:"COVID deaths in California over time”}
                       }
              }
             ]
           }
}
```

Queries. The file contains one information request:

“Report the daily cumulative death counts for the state of California for all days in the dataset in a form of a table with each row representing a single day”

This request should generate the following aggregation pipeline:

```
db.covid.aggregate({$match: {state: "CA”}},
                   {$project: {_id:0, death:1, date:1}},
                   {$sort: {date:1}}
                  )
```

How the pipeline is generated.

1. The **`collection`** parameter value sets the **`db.covid.aggregate(...)`** call.
2. The **`target`** parameter value sets the first **`$match`** stage.
3. The **`time`** parameter is missing, so no filter on the date range is created.
4. The **`task.track`** value in the only task in the configuration document tells us to track the cumulative number of deaths, represented in the **`covid`** collection in the **`"death"`** field. Our independent variable is time, so we track **`"date"`** field as well.
5. The **`$sort`** stage is a convenience stage to make sure that the data is returned to your application in a meaningfully ordered way.

6. The aggregation level is obviated by the fact that data for only one state is requested. Therefore the output would be the same for any allowable aggregation level.

Expected Result. Only a table is requested. The table shall be prefaced in the HTML output by the requested title. The table itself has two columns. First column is for date values, second -- for the number of deaths. A possible HTML rendering of this table is

```
<table>
<tr><th>Date</th><th>Cumulative Deaths</th></tr>
<tr><td> March 3, 2020</td> <td> 0 </td></tr>
….
<tr><td>April 5, 2020</td><td>319</td></tr>
</table>
```

Note that you are allowed to decorate your HTML output in any way you like. The HTML code below is only a suggested bare-bones variant.

Test Case 03: config03.json

```
{refresh: true,
 collection: "covid",
 aggregation: "usa",
 time: "month",
 analysis: {[{task: {track: "positive"},
              output: {graph:{ type: "line",
                               legend:"off",
                               combo:"combine"},
                       table:{row: "state",
                              column: "time",
                              title: "COVID cases in the USA this month"
                              }
                       }
              }
             ]
           }
}
```

Queries. The file contains one information request:

"Report daily numbers of positive COVID-19 cases from the beginning of the current month through today for the entire US (including territories and possessions)"

This request should generate the following aggregation pipeline if run on April 29, 2020:

```
db.covid.aggregate({$match: {date: {$gte: 20200401, $lte: 20200429}}}
                   {$project: {_id:0, positive:1, date:1}},
                   {$group: {_id:"$date",
                             positive: {sum: "$positive"}}
                   }
                   {$sort: {date:1}}
                  )
```

How the pipeline is generated.

1. The **`collection`** parameter value sets the **`db.covid.aggregate(...)`** call.
2. The **`target`** parameter is missing, which means that no filter on states is needed.
3. The **`time`** parameter sets a check for current date, and yields a **`$match`** stage selecting the dates from the first of the month to today.
4. The **`task.track`** value in the only task in the configuration document tells us to track the cumulative number of positive cases (see Test Case 01). The **`$project`** stage here is optional, you can decide if your code can be optimized to generate only the **`$group`** stage that follows.
5. The **`$group`** stage is the result of the aggregation level being set to **`"usa".`** The covid collection reports all data on a state-by-state basis. USA-wide numbers must be constructed by adding up the totals for each day from all the states, territories and possessions. Note, that the **`$group`** stage essentially obviates the need for the $project stage in this query.
6. The **`$sort`** stage is a convenience stage to make sure that the data is returned to your application in a meaningfully ordered way.

Expected Result. This configuration file requests both a line graph for the result computed, and a table. The table shall look the same way as in Test Case 02, the graph - the same way as in Test Case 01.

Test Case 04.

```
{refresh: true,
 collection: "covid",
 aggregation: "state",
 target: ["CA","NY","WA"],
 time: "week",
 analysis: {[{task: {ratio: {numerator: "death",
                             denominator: "positive"}},
              output: {graph:{ type: "line",
                               legend:"off",
                               combo:"combine",
                               title: "Death to Positive ratios this week"},
                       table:{row: "state",
                              column: "time",
                              }
                       }
              },
             [{task: {stats: ["death", "positive"]}},
               output:{table: {row: "stats",
                               column:"state"}
                       }
               }
             ]
           }
}
```

Queries. The file contains two information requests:

1. "Report daily numbers for the ratio of cumulative deaths to cumulative positive cases in the states of California, New York, and Washington, on a state-by-state basis for the last seven days.

2. "Report the mean and population standard deviation values for cumulative number of deaths and cumulative number of positive cases for the states of California, New York and Washington."

You can approach query generation for multiple tasks in a variety of ways. Below I show two independent queries that can be generated. These queries can also be merged using a simple faceted search stage after the common stages. There are additional ways to write these queries

as a single aggregation pipeline, but you do not need to pursue them. Generating two queries is sufficient.

Aggregation Pipeline 1 (run on April 29, 2020):

```
db.covid.aggregate({$match: {state {$in: ["CA","WA","NY"]}}},
                   {$match: {date: {$gte: 202023, $lte: 20200429}}},
                   {$project: {_id:0, date:1,
                               ratio:{$divide: ["$death", "$positive]}
                              } },
                   {$sort: {state:1, date:1}}
                  )
```

The pipeline above returns the data in a minimally acceptable form. If you want to separate individual state data, you can replace the **`$sort`** stage above with

```
{$sort:{date:1}),
{$group: {_id:"state",
          dateArray: {$push:"$date"},
          ratioArray: {$push: "$ratio"}
         }
 }
```

Aggregation Pipeline 2 (run on April 29, 2020):

```
db.covid.aggregate({$match: {state: {$in: ["CA","WA","NY"]}}},
                   {$match: {date: {$gte: 20200423, $lte: 20200429}}},
                   {$group:{_id:"$state",
                            avgPositive: {$avg: "$positive"},
                            stdPositive: {$stdDevPop:"$positive"},
                            avgDeath:{$avg:"$death"},
                            stdDeath:{$stdDevPop:"$death"}
                           }
                   },
                   {$sort: {state:1}}
                  )
```

How the pipeline is generated.

1. The **`collection`** parameter value sets the **`db.covid.aggregate(...)`** call.
2. The **`target`** parameter sets up the first selection/filter stage in both pipelines: we need to keep only California, Washington, and New York data.

3. The **`time`** parameter sets a check for current date, and yields a **`$match`** stage selecting the dates for the last seven days, ending on today (April 23 through April 29). This stage is present in both pipelines.
4. The first pipeline then proceeds to compute the ratio between the number of deaths and the number of positive cases (on a day-by-day basis) as requested by the first element of the **`analysis`** array. The aggregation level is set to **`"state":`** this means that no additional aggregation of results is needed.
5. The output of the first pipeline is sorted by state and date to create a convenient order for the Python program to process.
6. The second analytical task asks for "**`stats",`** which to us means computing means and population standard deviations for a given set of variables. The list of variables indicated that we need to aggregate the cumulative number of positive cases and the cumulative number of deaths. This sets up the **`$group`** stage. The data is grouped by state, due to the aggregation level being set to **`"state" -`** which means data is reported on a state-by-state basis.

Expected Result. The first query asks for the output to be rendered as both a graph and a table. There are three "lines' to be drawn on the graph - one per state. The combination strategy (value of the **`combo`** field) of **`"combine"`** asks you to draw a single **`matplotlib.pyplot.plot()`** plot and place the three line graphs for the ratios for Washington, California, and New York on it., (Note: the test case calls for no legend, so it won't be possible to determine which line is which, but that's ok).

Additionally, the same data needs to be reported in the form of an HTML table. Here, the time dimension is "vertical" - each new row is a new date, while the state dimension is horizontal - each new column is a new state… Please make certain you parse the instructions properly - they may be confusing. The shape of the HTML table is (all numbers are fake):

```
<table>
<tr>
  <th>Date</th>
  <th>California</th>
  <th>New York</th>
  <th>Washington</th>
</tr>
<tr>
 <td>April 23, 2020</td>
 <td> 0.12</td>
 <td> 0.06</td>
 <td> 0.11</td>
</tr>
…
<tr>
 <td>April 29, 2020</td>
```

```
 <td> 0.08</td>
 <td> 0.02</td>
 <td> 0.01</td>
</tr>
</table>
```

The second query returns an HTML table in which the horizontal dimension is different statistics, and the vertical dimension is different states. The table shall look as follows (specific labels are left up to you to form).

```
<table>
<tr>
  <th>State</th>
  <th>Mean Positive</th>
  <th>Positive St. Dev.</th>
  <th>Mean Death</th>
  <th>Death St. Dev.</th>
</tr>
<tr>
  <td>CA</td>
  <td> …</td><td>...</td><td>...</td><td>...</td>
</tr>
<tr>
  <td>NY</td>
  <td> …</td><td>...</td><td>...</td><td>...</td>
</tr>
<tr>
  <td>WA</td>
  <td> …</td><td>...</td><td>...</td><td>...</td>
</tr>
</table>
```

Test Case 05:

```
{refresh: false,
 collection: "collection",
 aggregation: "state",
 target: "IA",
 counties: ["Polk", "Scott"],
 time: "month",
 analysis: {[{task: {track: "positive"}},
             {output:{graph: {type: "scatter",
                              legend:"on",
                              combo: "split",
                              title: "Covid Cases this month"
                             }
                      }
              }]
            }
}
```

Queries. The file contains one information request:

"Report daily cumulative numbers of positive COVID-19 cases from the beginning of the current month through today for Polk and Scott counties in Iowa"

This request should generate the following aggregation pipeline if run on April 29, 2020. The query below assumes that the dates are represented in the same format as in the **`covid`** collection, and that the names of the attributes are consistent with those in the **`covid`** collection:

```
db.states.aggregate({$match: {state:"IA"}},
                    {$match: {county: {$in: ["Polk", "Scott"]}},
                    {$match: {date: {$gte: 20200401, $lte: 20200429}}}
                    {$project: {_id:0, county:1, positive:1, date:1}},
                   {$sort: {county:1, date:1}}
                  )
```

How the pipeline is generated.

1. The **`collection`** parameter value sets the **`db.states.aggregate(...)`** call.
2. The **`target`** parameter specifies the state of Iowa and sets up the first **`$match`** stage.
3. The **`counties`** parameter is present, and sets the second **`$match`** stage in which only information about the two specified counties is kept.
4. The value of the **`time`** parameter (**`"month"`**) sets a check for current date, and yields the third **`$match`** stage selecting the dates from the first of the month to today.

5. The **`task.track`** value in the only task in the configuration document tells us to track the cumulative number of positive cases (see Test Case 01). The $project stage keeps the number of positive cases in the result, removes everything else. The level of aggregation is set to **`"county".`** This means no additional aggregation needs to be performed.
6. The **`$sort`** stage is a convenience stage to make sure that the data is returned to your application in a meaningfully ordered way.

You can also follow the notes in Test Case 04 and set up the output to contain an array of dates and an array of positive case values for each county.

Expected Result. The results have to be reported as graphs. The combination strategy is set to **`"split",`** which instructs your program to create two graphs: one per county (this is determined by the level of aggregation). The legend is turned on, so the name of the county has to be present in the legend in each graph. Both graphs shall be placed into the HTML document under the same title.