

## **DB Assignment 5**

Giannina Flamiano


November 22, 2024

## 1. Over how many years was the unemployment data collected?

### JS QUERY

```
db.unemployment.aggregate([
  {
    $group: { _id: "$Year" }
  },
  {
    $count: "totalYears"
  }
])
```

### SCREENSHOT



```
< {
  totalYears: 27
}
```

### EXPLANATION

This query first groups the documents by year to get the distinct values of year in the collection. Then, it counts the number of groups, passed from the previous stage, to get the distinct count of years in the dataset.


---

## 2. How many states were reported on this dataset?

### JS QUERY

```
db.unemployment.aggregate([{$group: {_id: "$State"} },{$count: "totalStates" }])
```

### SCREENSHOT



```
< {
  totalStates: 47
}
test>
```

### EXPLANATION

This query first groups the documents by state to get the distinct values of state in the collection. Then, it counts the number of groups, passed down from the previous stage, to get the distinct count of states in the dataset.

---

### 3. What does this query compute? `db.unemployment.find({Rate : {$lt: 1.0}}).count()`

#### JS QUERY

```
db.unemployment.find({Rate : {$lt: 1.0}}).count()
```

#### SCREENSHOT

A screenshot of a terminal window with a dark background. It shows a light blue prompt character followed by the number 657.

#### EXPLANATION

This query computes the number of documents in the unemployment collection that have a Rate value that is less than 1. `db.unemployment.find({Rate: {$lt: 1.0}})` retrieves all documents from the unemployment collection where the Rate field has a value less than 1. `count()` counts the total number of documents that match the query criteria.

---

### 4. Find all counties with unemployment rate higher than 10%

#### JS QUERY

```
db.unemployment.aggregate([
  {
    $match: {Rate: {$gt:10}}
  },
  {
    $group: {_id: "$County"}
  }
])
```

#### EXPLANATION

This query first filters out which documents have a rate that is greater than 10. Then, it groups the documents passed from the first stage by county. Thus, the output is a list of all counties with an unemployment rate higher than 10%.

#### SCREENSHOT

A screenshot of a MongoDB shell window. The title bar says 'MONGODB'. The output is a list of objects, each with an '\_id' field representing a county name. The counties listed are: Scioto County, Dodge County, Garland County, Bureau County, Calloway County, Pope County, Rock Island County, Bartholomew County, Yuma County, Bibb County, Wilson County, Sawyer County, Hampshire County, Dillon County, Pecos County, Somervell County, and Rosebud County. At the bottom, it says 'Type "it" for more'.

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

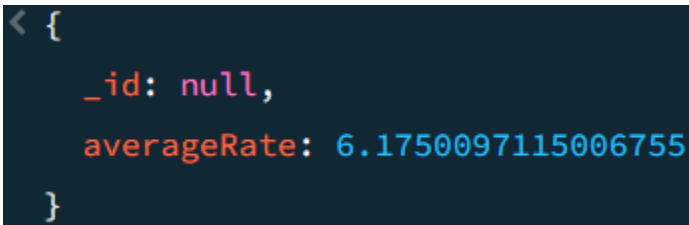
---

## 5. Calculate the average unemployment rate across all states.

### JS QUERY

```
db.unemployment.aggregate([
  {
    $project: {
      Rate: { $ifNull: ["$Rate", 0] }
    }
  },
  {
    $group: {
      _id: null,
      averageRate: { $avg: "$Rate" }
    }
  }
])
```

### SCREENSHOT



```
< {
  _id: null,
  averageRate: 6.1750097115006755
}
```

### EXPLANATION

This query first does a projection to do a data cleanup on the rate field; it changes any null rate values to 0 to ensure proper calculation in later stages. Then, the query groups all documents into one single group so that it can calculate across the entire collection. Finally, it calculates the average of the rate field to get the average unemployment rate across all states.

---

## 6. Find all counties with an unemployment rate between 5% and 8%.

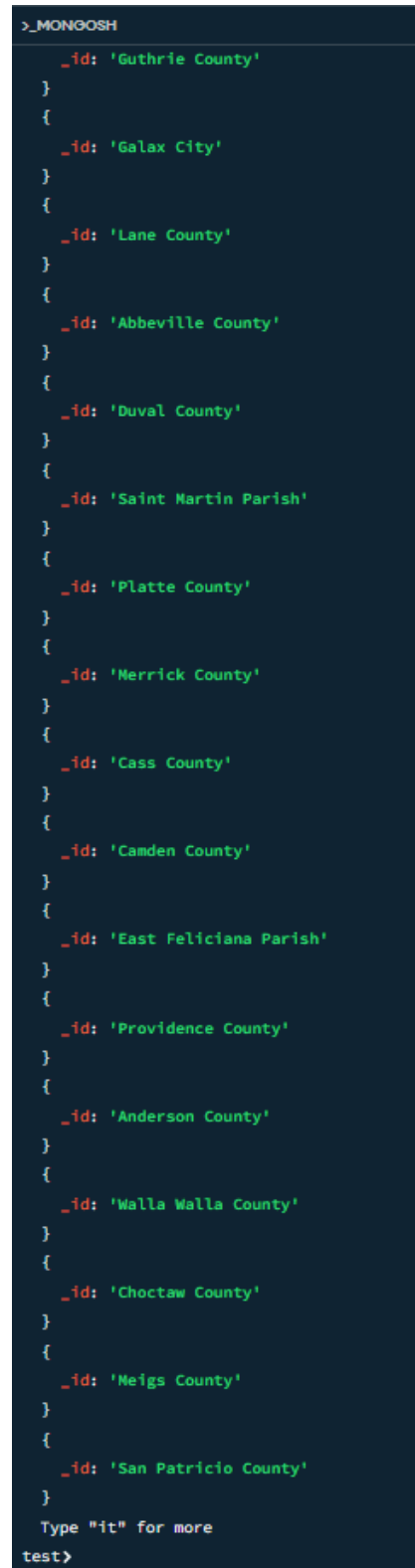
### JS QUERY

```
db.unemployment.aggregate([
  {
    $match: {Rate: {$gt:5, $lt:8}}
  },
  {
    $group: {_id: "$County"}
  }
])
```

### EXPLANATION

This query first returns all documents that have a rate that is greater than 5 and less than 8. Then, it groups the documents by county to get a list of the unique counties. Thus the output of the query will be a list of all counties with an unemployment rate between 5% and 8%.

### SCREENSHOT



```
>_MONGODB
{
  "_id": 'Guthrie County'
}
{
  "_id": 'Galax City'
}
{
  "_id": 'Lane County'
}
{
  "_id": 'Abbeville County'
}
{
  "_id": 'Duval County'
}
{
  "_id": 'Saint Martin Parish'
}
{
  "_id": 'Platte County'
}
{
  "_id": 'Merrick County'
}
{
  "_id": 'Cass County'
}
{
  "_id": 'Camden County'
}
{
  "_id": 'East Feliciana Parish'
}
{
  "_id": 'Providence County'
}
{
  "_id": 'Anderson County'
}
{
  "_id": 'Walla Walla County'
}
{
  "_id": 'Choctaw County'
}
{
  "_id": 'Weigs County'
}
{
  "_id": 'San Patricio County'
}
Type "it" for more
test>
```

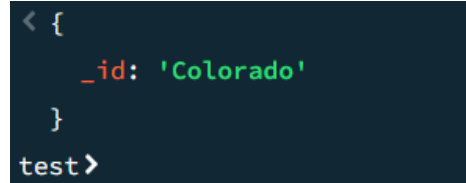
(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

## 7. Find the state with the highest unemployment rate. Hint. Use { \$limit: 1 }

### JS QUERY

```
db.unemployment.aggregate([
  {
    $sort: {Rate: -1}
  },
  {
    $limit: 1
  },
  {
    $group: { _id: "$State" }
  }
])
```

### SCREENSHOT



```
< {
  _id: 'Colorado'
}
test>
```

### EXPLANATION

This query first sorts the documents in descending order by rate. Then, it limits the results by 1 so that we get the highest unemployment rate across the entire collection. Finally, the query groups the document by state. Since there is only one record after the \$limit stage, the result extracts the state from the record as \_id, thus giving us the state with the highest unemployment rate.

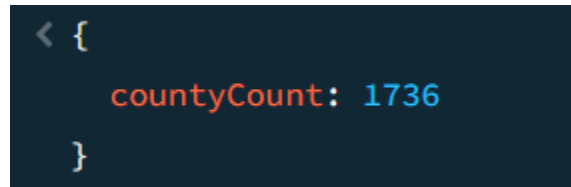
---

## 8. Count how many counties have an unemployment rate above 5%.

### JS QUERY

```
db.unemployment.aggregate([
  {
    $match: { Rate: { $gt: 5 } }
  },
  {
    $group: { _id: "$County" }
  },
  {
    $count: "countyCount"
  }
])
```

### SCREENSHOT



```
< {
  countyCount: 1736
}
```

### EXPLANATION

This query first filters out the documents with a rate greater than 5. Then, it groups the documents by county to get a list of unique states. Finally, it counts the number of unique documents returned from the previous stage to get the count of states with an unemployment rate above 5%.

---

## 9. Calculate the average unemployment rate per state by year.

### JS QUERY

```
db.unemployment.aggregate([
  {
    $group: {
      _id: {
        State: "$State",
        Year: "$Year"
      },
      avgRate: { $avg: "$Rate" }
    }
  },
  {
    $project: {
      _id: 0,
      State: "$_id.State",
      Year: "$_id.Year",
      avgRate: 1
    }
  },
  {
    $sort: { State: 1, Year: 1 }
  }
])
```

### EXPLANATION

This query first groups the documents by state and year so that each record returned is a unique combination of state and year. Then, it computes the average of the rate field. In the second stage, the query does a projection to exclude the `_id` from the results, but include state, year, and average rate. Lastly, the query sorts the documents in ascending order by state and year. Thus, the result is a list of the average unemployment per state per year.

### SCREENSHOT



```
< {
  avgRate: 8.226990049751244,
  State: 'Alabama',
  Year: 1990
}
{
  avgRate: 9.0818407960199,
  State: 'Alabama',
  Year: 1991
}
{
  avgRate: 9.296890547263681,
  State: 'Alabama',
  Year: 1992
}
{
  avgRate: 9.182462686567163,
  State: 'Alabama',
  Year: 1993
}
{
  avgRate: 7.622139303402587,
  State: 'Alabama',
  Year: 1994
}
{
  avgRate: 7.607089552238806,
  State: 'Alabama',
  Year: 1995
}
{
  avgRate: 7.235199004975125,
  State: 'Alabama',
  Year: 1996
}
{
  avgRate: 6.9175373134328355,
  State: 'Alabama',
  Year: 1997
}
{
  avgRate: 6.167039800995025,
  State: 'Alabama',
  Year: 1998
}
{
  avgRate: 6.621641791044777,
  State: 'Alabama',
  Year: 1999
}
{
  avgRate: 5.503482587064677,
```

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

## 10. (Extra Credit) For each state, calculate the total unemployment rate across all counties (sum of all county rates).

### JS QUERY (1)

```
db.unemployment.aggregate([
  {
    $group: {
      _id: "$State",
      totalUnemploymentRate: {
        $sum: { $divide: ["$Rate", 100] }
      }
    }
  },
  {
    $project: {
      _id: 0,
      State: "$_id",
      totalUnemploymentRate: 1
    }
  }
])
```

### EXPLANATION (1)

This query first groups the documents by state then adds up all the scaled rate values within each state group. We scale the rates by 100 since rate fields are stored as percentages (ex. 5.0 = 5%). The second stage of this query does a projection to reformat the output by excluding the `_id` fields, adding the state field, and keeping the `totalUnemploymentRate` in the output. We then get a list of the sum of all the county rates for each state.

### SCREENSHOT (1)

```
>_MONGODB
{
  State: 'Missouri',
  totalUnemploymentRate: 436.96,
}
{
  State: 'New Jersey',
  totalUnemploymentRate: 1250.724,
}
{
  State: 'New York',
  totalUnemploymentRate: 3398.6620000000003,
}
{
  State: 'Texas',
  totalUnemploymentRate: 324.725,
}
{
  State: 'Maine',
  totalUnemploymentRate: 1971.874,
}
{
  State: 'Michigan',
  totalUnemploymentRate: 1014.862,
}
{
  State: 'Washington',
  totalUnemploymentRate: 1526.616,
}
{
  State: 'California',
  totalUnemploymentRate: 660.793,
}
{
  State: 'North Dakota',
  totalUnemploymentRate: 341.041,
}
{
  State: 'Wyoming',
  totalUnemploymentRate: 450.745,
}
{
  State: 'Arizona',
  totalUnemploymentRate: 915.539,
}
{
  State: 'Oregon',
  totalUnemploymentRate: 140.80100000000002,
}
{
  State: 'New Hampshire',
  totalUnemploymentRate: 140.80100000000002,
}
Type "it" for more
test>|
```

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)



## JS QUERY (2)

```
db.unemployment.aggregate([
  {
    $group: {
      _id: "$State",
      totalRate: {
        $sum: {
          $divide: ["$Rate", 100]
        }
      },
      countyCount: { $sum: 1 }
    },
  },
  {
    $project: {
      _id: 0,
      State: "$_id",
      averageRate: {
        $multiply: [{
          $divide:
            ["$totalRate",
             "$countyCount"]
        }, 100] }
    }
  }
])
```

## EXPLANATION (2)

The original query 1 returns documents with rate fields greater than 100%. This data is not that meaningful, and we can write a better and more informative query by calculating the average unemployment rate for each state across all counties. This query first groups documents by state then computes totalRate. totalRate is the sum of all the rate values after scaling the rate by 100. The query then computes the number of counties per state and holds that result in countyCount. The second stage of this query does a projection to filter the output results. We exclude \_id, but include state and the averageRate per state. averageRate divides totalRate by countyCount then normalizes the results by multiplying by 100 to convert rate value back to a percentage. The output of this query is then a list of all the states and their average unemployment rate across all their counties.

## SCREENSHOT (2)

```
>_MONGOSH
  averageRate: 6.2584245840042945
}
{
  State: 'New Jersey',
  averageRate: 6.422104644326867
}
{
  State: 'New York',
  averageRate: 6.226224611708482
}
{
  State: 'Texas',
  averageRate: 5.8945194075410186
}
{
  State: 'Maine',
  averageRate: 6.263985339506172
}
{
  State: 'Washington',
  averageRate: 8.031513137068693
}
{
  State: 'Michigan',
  averageRate: 8.136136326126424
}
{
  State: 'California',
  averageRate: 9.045005332385353
}
{
  State: 'Arizona',
  averageRate: 9.274588477366255
}
{
  State: 'Wyoming',
  averageRate: 4.576502952227591
}
{
  State: 'Oregon',
  averageRate: 7.849271262002744
}
{
  State: 'New Hampshire',
  averageRate: 4.345709876543211
}
{
  State: 'North Dakota',
  averageRate: 3.848084090379688
}
Type "it" for more
test>
```

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

## 11. (Extra Credit) The same as Query 10 but for states with data from 2015 onward.

### JS QUERY (1)

```
db.unemployment.aggregate([
  {
    $match: { Year: { $gte: 2015 } }
  },
  {
    $group: {
      _id: "$State",
      totalUnemploymentRate: {
        $sum: {
          $divide: ["$Rate", 100]
        }
      }
    }
  },
  {
    $project: {
      _id: 0,
      State: "$_id",
      totalUnemploymentRate: 1
    }
  }
])
```

### EXPLANATION (1)

This query first filters out documents with a year greater than or equal to 2015. The second stage of this query then groups the documents by state then adds up all the scaled rate values within each state group. We scale the rates by 100 since rate fields are stored as percentages (ex. 5.0 = 5%). The third stage of this query does a projection to reformat the output by excluding the `_id` fields, adding the state field, and keeping the `totalUnemploymentRate` in the output. We then get a list of the sum of all the county rates for each state using data from 2015 onward.

### SCREENSHOT (1)



```
>_MONGODB
State: 'Missouri'
}
{
  totalUnemploymentRate: 28.436,
  State: 'New Jersey'
}
{
  totalUnemploymentRate: 145.575,
  State: 'North Carolina'
}
{
  totalUnemploymentRate: 78.763,
  State: 'New York'
}
{
  totalUnemploymentRate: 381.144,
  State: 'Texas'
}
{
  totalUnemploymentRate: 17.588,
  State: 'Maine'
}
{
  totalUnemploymentRate: 96.792,
  State: 'California'
}
{
  totalUnemploymentRate: 42.982,
  State: 'North Dakota'
}
{
  totalUnemploymentRate: 64.338,
  State: 'Washington'
}
{
  totalUnemploymentRate: 29.744,
  State: 'Arizona'
}
{
  totalUnemploymentRate: 24.947,
  State: 'Wyoming'
}
{
  totalUnemploymentRate: 52.789,
  State: 'Oregon'
}
{
  totalUnemploymentRate: 7.418,
  State: 'New Hampshire'
}
Type "it" for more
test>
```

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

## JS QUERY (2)

```
db.unemployment.aggregate([
  {
    $match: { Year: { $gte: 2015 } }
  },
  {
    $group: {
      _id: "$State",
      totalRate: {
        $sum: {
          $divide: ["$Rate", 100]
        }
      },
      countyCount: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      State: "$_id",
      averageRate: {
        $multiply: [{
          $divide:
            ["$totalRate",
             "$countyCount"]
        }, 100] }
    }
  }
])
```

## EXPLANATION (2)

The original query 1 returns documents with rate fields greater than 100%. This data is not that meaningful, and we can write a better and more informative query by calculating the average unemployment rate for each state across all counties. This query first filters out documents that have a year greater than or equal to 2015. In the second stage of this query, it groups documents by state then computes totalRate. totalRate is the sum of all the rate values after scaling the rate by 100. The query then computes the number of counties per state and holds that result in countyCount. The third stage of this query does a projection to filter the output results. We exclude \_id, but include state and the averageRate per state. averageRate divides totalRate by countyCount then normalizes the results by multiplying by 100 to convert rate value back to a percentage. The output of this query is then a list of all the states and their average unemployment rate across all their counties using data from 2015 onwards.

## SCREENSHOT (2)

```
>_MONGOSH
averageRate: 5.251521739130435
}
{
  State: 'New Jersey',
  averageRate: 5.642863492063492
}
{
  State: 'New York',
  averageRate: 5.293212365591398
}
{
  State: 'Texas',
  averageRate: 4.94082624671916
}
{
  State: 'Maine',
  averageRate: 4.580208333333333
}
{
  State: 'Washington',
  averageRate: 6.873717948717949
}
{
  State: 'Michigan',
  averageRate: 5.619879518072289
}
{
  State: 'California',
  averageRate: 7.25577211394303
}
{
  State: 'Arizona',
  averageRate: 8.262222222222222
}
{
  State: 'Wyoming',
  averageRate: 4.519384057971014
}
{
  State: 'Oregon',
  averageRate: 6.100578703703704
}
{
  State: 'New Hampshire',
  averageRate: 3.090833333333333
}
{
  State: 'North Dakota',
  averageRate: 3.3790880503144654
}
Type "it" for more
test>
```

(NOTE: More documents are returned, I just could not fit them all in one screenshot.)

## ANALYZE SCHEMA SCREENSHOT

localhost:27017 > test > unemployment

>\_ Open MongoDB shell

Documents 885.5K Aggregations Schema Indexes 1 Validation

🕒 ▼ Type a query: { field: 'value' } or [Generate query](#) ➦

Reset

Analyze



Options ►

This report is based on a sample of 1000 documents. [Learn more](#)

\_id

objectid



County

string



Month

string



Rate

double



## State

string



Year

int32

