Leonidas Kesaris

Database Management Systems

Dr. Forouraghi

11/22/2024

Problem 1:

```
> db["unemployment"].distinct("Year").length
  //Counts the amount of unique years in the data set using distinct and length
< 27
Assigment_5>
```

The above query uses distinct to found the unique years in the data set with the array generated from the distinct(“Year”) method.

Problem 2:

```
> db["unemployment"].distinct("State").length
  // counts unique states in the data set using distinct and length
< 47
```

The above query uses distinct to find the amount of unique states in the array generated by the distinct(“State”) method.

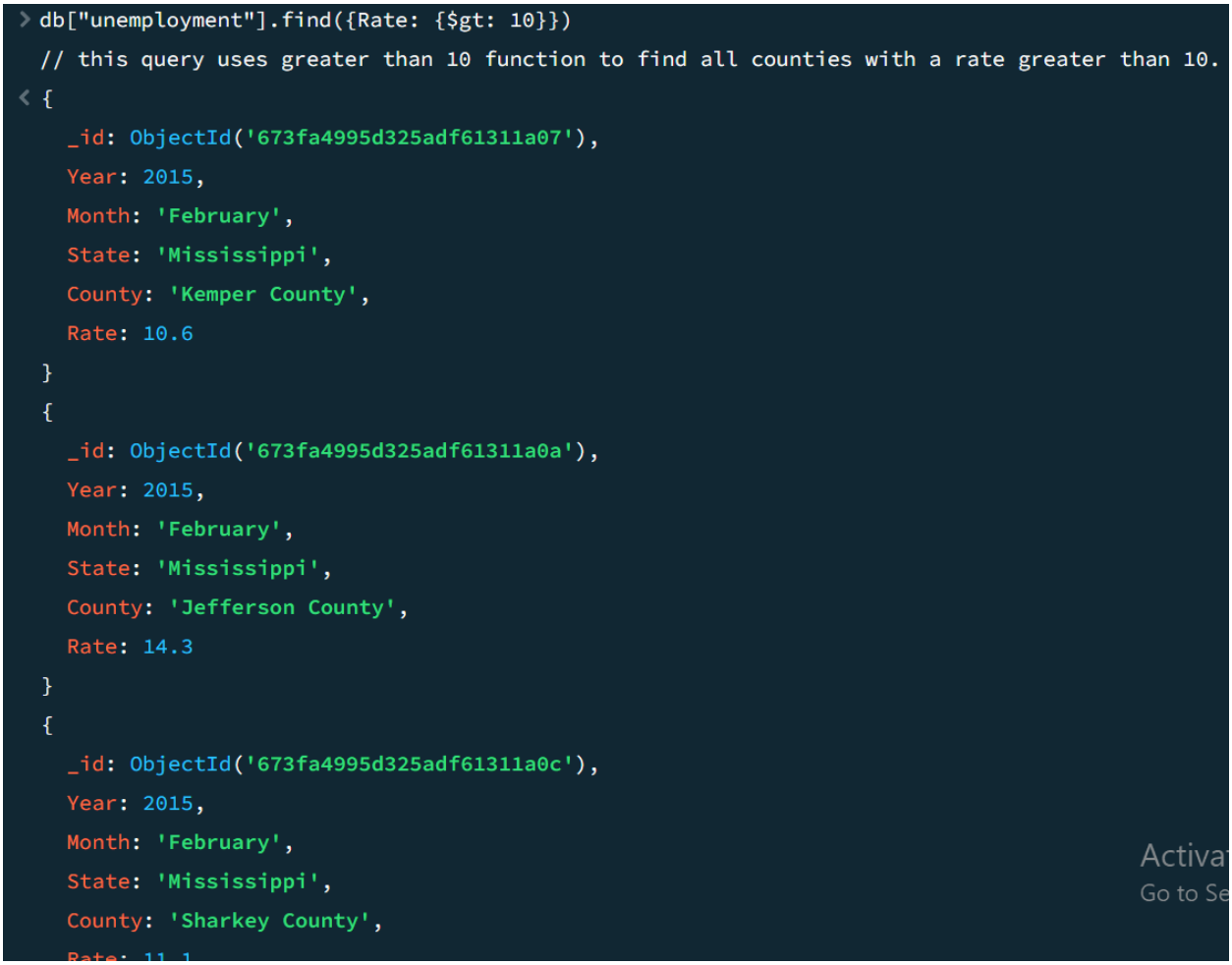
Problem 3

```
> use Assigment_5
< switched to db Assigment_5
> db.unemployment.find({Rate : {$lt: 1.0}}).count()// this function will get the count of all records
                                                   // with a rate less than 1


< 657
```

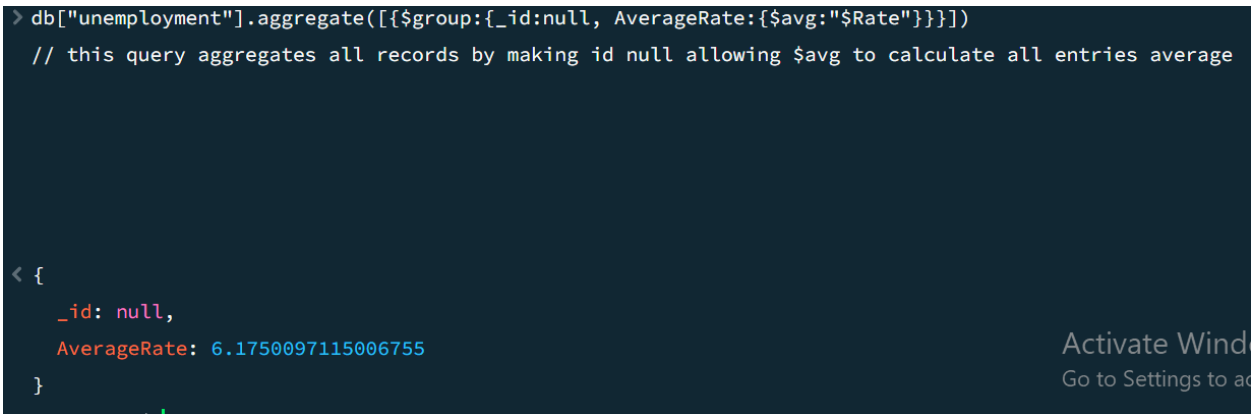
The above query finds the amount of entries in the database with a rate less than 1.0 using the count function.

Problem 4



The above query uses the greater than function which shows all entries in the database with a rate larger than 10, which also showcases each county with a rate larger than 10.

Problem 5



The above query groups all entries by making _id null, allowing for all entries to be put in the same group. Therefore the $avg function calculates the average rate of all entries in the data base.

Problem 6

```
> db["unemployment"].find({Rate: {$gte: 5, $lte: 8}})
  // Query finds all records with a rate greater than equal to five and les than or equal to 8.
< {
    _id: ObjectId('673fa4995d325adf61311a03'),
    Year: 2015,
    Month: 'February',
    State: 'Mississippi',
    County: 'Newton County',
    Rate: 6.1
  }
  {
    _id: ObjectId('673fa4995d325adf61311a05'),
    Year: 2015,
    Month: 'February',
    State: 'Mississippi',
    County: 'Monroe County',
    Rate: 7.9
  }
  {
    _id: ObjectId('673fa4995d325adf61311a06'),
    Year: 2015,
    Month: 'February',
    State: 'Mississippi',
    County: 'Hinds County',
    Rate: 6.1
```


Above query finds all entries in the database (counties) with a rate 5 or above and 8 or below using the gte and lte comparison operators.

Problem 7

```
> db["unemployment"].aggregate([{$sort: { Rate: -1}},{$limit:1}])
  // Query sorts by rate -1 to sort in descending order, then limits to show highest value.
< {
    _id: ObjectId('673fa4ac5d325adf61367602'),
    Year: 1992,
    Month: 'January',
    State: 'Colorado',
    County: 'San Juan County',
    Rate: 58.4
  }
```

The above query uses the sort operator with -1 to set the data returned to be descending, and then this is limited to 1 to show only the highest value in the data.

Problem 8

```
> db["unemployment"].distinct("County", {Rate:{$gt:5}}).length
  // returns the count of all distinct counties with a unemployment more than 5%
< 1736
Assigment_5 >
```

The query above uses distinct to find all counties and then filters these values with the $gt:5 to show only responses where Rate is larger than 5%. Length is then use to count the amount of entries which are returned by the array which filters counties with rates greater than 5.

Problem 9

```
> db["unemployment"].aggregate([{$group:{_id:{State: "$State",Year:"$Year"},
  AverageRate:{$avg:"$Rate"}}},
  {$sort:{"_id.State": 1, "_id.year": 1}}
  ])
  // the query aggreagates by state and year, with ID as ocomposite key.
  // average function returns the average of each group
< {
    _id: {
      State: 'Alabama',
      Year: 2011
    },
    AverageRate: 11.211069651741294
  }
  {
    _id: {
      State: 'Alabama',
      Year: 2009
    },
    AverageRate: 13.166169154228855
  }
  {
    _id: {
      State: 'Alabama',
      Year: 2016
    },
```

The above query groups entries in the database bu state and year using a composite key of _id: (State: “$State”, Year: “$Year”} to create distinct groups for each unique combination of state and year. Next $avg is used to get the average rate of each group. The data is then sorted alphabetically by state and then sorted numerically ascending.

Problem 10

```
> db["unemployment"].aggregate([{$group:{_id:"$State", TotalRate: {$sum: "$Rate"}}}])
  //groups the data by state and then gets the sum of all unemployment rates for each state and year
< {
    _id: 'Arizona',
    TotalRate: 45074.5
  }
  {
    _id: 'Wyoming',
    TotalRate: 34104.1
  }
  {
    _id: 'Louisiana',
    TotalRate: 50502.9
  }
  {
    _id: 'Vermont',
    TotalRate: 22427.6
  }
  {
    _id: 'Montana',
    TotalRate: 96261.5
  }
  {
    _id: 'Pennsylvania',
    TotalRate: 140577.6
```


Above query groups data in unemployment by state then total rate is created and holds the sum of the rate values for unique states with $sum operator.

Problem 11

```
> db["unemployment"].aggregate([{ $match:{ Year: {$gte: 2015}}},
                                {$group:{_id:"$State", TotalRate: {$sum: "$Rate"}}}])
  //filters data with the years greater than or equal to 2015
  //groups by state and sums for the countys in each state.

< {
    _id: 'California',
    TotalRate: 9679.2
  }
  {
    _id: 'Virginia',
    TotalRate: 15594.6
  }
  {
    _id: 'Montana',
    TotalRate: 5851.4
  }
  {
    _id: 'Pennsylvania',
    TotalRate: 9146.6
  }
  {
    _id: 'Louisiana',
    TotalRate: 7474.8
  }
```

Above query uses $match to filter data to include responses with a year greater than or equal to 2015. The data is then grouped by state and then summed to calculate unemployment rate for each state summing rates for states in the year 2015 and after.