NodeJS Training: Module 4

MongoDB Aggregate Assignment

1. Write the aggregation query that will find the number of products by category of a collection that has the form:

```
{
 "_id" : ObjectId("50b1aa983b3d0043b51b2c52"),
 "name" : "Nexus 7",
 "category" : "Tablets",
 "manufacturer" : "Google",
 "price" : 199
}
```

```
db.collection.aggregate( [
  {
    $group : {
      category : ‘$category’,
       noOfProducts : { $sum : 1 }
    }
  }
] )
```

2. If you have the following collection of stuff:

```
# > db.stuff.find()
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }
```

# and you perform the following aggregation:

```
db.stuff.aggregate([{$group:{_id:'$c'}}])
```

# How many documents will be in the result set from aggregate?

3 { ‘_id’ : 1 }
  { ‘_id’ : 2 }
  { ‘_id’ : 3 }

3. Given the following collection:

```
# > db.stuff.find()
{ "_id" : ObjectId("50b26f9d80a78af03b5163c8"), "a" : 1, "b" : 1, "c" : 1 }
{ "_id" : ObjectId("50b26fb480a78af03b5163c9"), "a" : 2, "b" : 2, "c" : 1 }
{ "_id" : ObjectId("50b26fbf80a78af03b5163ca"), "a" : 3, "b" : 3, "c" : 1 }
{ "_id" : ObjectId("50b26fcd80a78af03b5163cb"), "a" : 3, "b" : 3, "c" : 2 }
{ "_id" : ObjectId("50b26fd380a78af03b5163cc"), "a" : 3, "b" : 5, "c" : 3 }
{ "_id" : ObjectId("50b27f7080a78af03b5163cd"), "a" : 3, "b" : 3, "c" : 2 }
```

# And the following aggregation query:

```
db.stuff.aggregate([{$group:
                 {_id:
                  {'moe':'$a',
                   'larry':'$b',
                   'curly':'$c'
                  }
                 }
                }])
```

# How many documents will be in the result set?

5 { '_id' : { 'moe' : 1, 'larry' : 1, 'curly' : 1 } }
{ '_id' : { 'moe' : 2, 'larry' : 2, 'curly' : 1 } }
{ '_id' : { 'moe' : 3, 'larry' : 3, 'curly' : 2 } }
{ '_id' : { 'moe' : 3, 'larry' : 3, 'curly' : 1 } }
{ '_id' : { 'moe' : 3, 'larry' : 5, 'curly' : 3 } }

6. Which of the following aggregation expressions must be used in conjunction with a sort to make any sense?
$first
$last
Both because they fetch first and last element respectively in an ordered data.

Note: This problem, and some after it, use the zips collection from **zips.json**. You don't need to download it, but you can if you want, allowing you to test your queries within MongoDB. You can import, once downloaded, using mongoimport.
Suppose we have a collection of populations by postal code. The postal codes in are in the _id field, and are therefore unique. Documents look like this:

```
{
      "city" : "CLANTON",
      "loc" : [
            -86.642472,
            32.835532
      ],
      "pop" : 13990,
      "state" : "AL",
      "_id" : "35045"
}
```

Note: For students outside the United States, there are 50 non-overlapping states in the US with two letter abbreviations such as NY and CA. In addition, the capital of Washington is within an area designated the District of Columbia, and carries the abbreviation DC. For purposes of the mail, the postal service considers DC to be a "state." So in this dataset, there are 51 states. We call postal codes "zip codes." A city may overlap several zip codes.

7. Write an aggregation query to sum up the population (pop) by state and put the result in a field called population. The collection name is zips. so something along the lines of db.zips.aggregate...

```
db.zips.aggregate( [
  {
    $group : {
      state : ‘$state’,
      population : { $sum : ‘$pop’}
    }
  }
] )
```

8. Given population data by zip code (postal code), write an aggregation expression to calculate the average population of a zip code (postal code) by state.

```
{ "NY": 9705.34, "NJ": 16949.9, "CT": 13226.48, "CA": 19067.72 }
```

```
db.zips.aggregate( [
  {
    $group : {
      _id : { state : ‘$state’, postal_code : ‘$_id’ },
     population : { $sum : ‘$pop’ }
    },
    $group : {
     _id : ‘$_id.state’,
     avgPopulation : { $avg : ‘$pop’ }
    }
  }
] )
```

9. Suppose we population by zip code (postal code) data that looks like this (putting in a query for the zip codes in Palo Alto)

```
# > db.zips.find({state:"CA",city:"PALO ALTO"})
{ "city" : "PALO ALTO", "loc" : [ -122.149685, 37.444324 ], "pop" : 15965,
"state" : "CA", "_id" : "94301" }
{ "city" : "PALO ALTO", "loc" : [ -122.184234, 37.433424 ], "pop" : 1835,
"state" : "CA", "_id" : "94304" }
{ "city" : "PALO ALTO", "loc" : [ -122.127375, 37.418009 ], "pop" : 24309,
"state" : "CA", "_id" : "94306" }
```
# Write an aggregation query that will return the postal codes that cover each city. The results should look like this:
```
{
      "_id" : "CENTREVILLE",
      "postal_codes" : [
            "22020",
            "49032",
            "39631",
            "21617",
            "35042"
      ]
}
```

```
db.zips.aggregate( [
 {
   $group : {
     _id : '$city', postal_codes : {$push : '$id' }
   }
 }
] )
```

10. Given the zipcode dataset that has documents that look like this:
> db.zips.findOne()
```
{
      "city" : "ACMAR",
      "loc" : [
            -86.51557,
            33.584132
      ],
      "pop" : 6055,
      "state" : "AL",
      "_id" : "35004"
}
```

Would you expect the following two queries to produce the same result or different results?
```
db.zips.aggregate([{"$group":{"_id":"$city", "postal_codes":
{"$push":"$_id"}}}])
db.zips.aggregate([{"$group":{"_id":"$city", "postal_codes":
{"$addToSet":"$_id"}}}])
```

Different because $addToSet doesnot add item to given field if it already contains it while $push will add object to field whether it exists or not.

11. Again thinking about the zip code database, write an aggregation query that will return the population of the postal code in each state with the highest population. It should return output

that looks like this:
```
{
 "_id" : "WI",
 "pop" : 57187
},
{
 "_id" : "WV",
 "pop" : 70185
},
# ..and so on
```
# Once again, the collection is named zips.

```
db.zips.aggregate( [
 {
   $group :
     { _id : '$state' ,
       pop : { $max : '$pop' }
     }
 }
] )
```

12. Write an aggregation query with a single projection stage that will transform the documents in the zips collection from this:
```
{
 "city" : "ACMAR",
 "loc" : [
 -86.51557,
 33.584132
 ],
 "pop" : 6055,
 "state" : "AL",
 "_id" : "35004"
}
```
# to documents in the result set that look like this:
```
{
 "city" : "acmar",
 "pop" : 6055,
 "state" : "AL",
 "zip" : "35004"
}
db.zips.aggregate( [
  {
    $project : {
      city : 1, pop : 1, state : 1, zip : '$_id', _id : 0
    }
  }
] )
```

13. Again, thinking about the zipcode collection, write an aggregation query with a single match phase that filters for zipcodes with greater than 100,000 people.
# Assume the collection is called zips.

```
db.zips.aggregate( [
 {
   $group : { zipcodes : ‘$_id’ , pop : { $sum : ‘$pop’ } }
 },
 {
   $match : { pop : { $gt : 100*1000 } }
 }
] )
```

14. Again, considering the zipcode collection, which has documents that look like this,

```
{
 "city" : "ACMAR",
 "loc" : [
 -86.51557,
 33.584132
 ],
 "pop" : 6055,
 "state" : "AL",
 "_id" : "35004"
}
```

# Write an aggregation query with just a sort stage to sort by (state, city), both ascending. Assume the collection is called zips.

```
db.zips.aggregate( [
 {
   $sort : { state : 1, city : 1 }
 }
] )
```

15. Given the following collection:
# > db.fun.find()

```
{ "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }
{ "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }
{ "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }
{ "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }
{ "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }
{ "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }
{ "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }
{ "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```

# What would be the value of c in the result from this aggregation query? (Evaluate without actually querying it. Read the docs to understand what this query would return)

```
db.fun.aggregate([
    {$match:{a:0}},
    {$sort:{c:-1}},
    {$group:{_id:"$a", c:{$first:"$c"}}}
])
```
c = 54

16. Given the following collection:

# > db.fun.find()
```
{ "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }
{ "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }
{ "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }
{ "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }
{ "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }
{ "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }
{ "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }
{ "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```
# And the following aggregation query
```
db.fun.aggregate([{$group:{_id:{a:"$a", b:"$b"}, c:{$max:"$c"}}}, {$group:
{_id:"$_id.a", c:{$min:"$c"}}}])
```
# What values are returned?

{ ‘_id’ : 0, ‘c’ : 97 },
{ ‘_id’ : 1, ‘c’ : 97 }

17. Suppose you have the following collection:

db.people.find()
```
{ "_id" : "Barack Obama", "likes" : [ "social justice", "health care", "taxes"
] }
{ "_id" : "Mitt Romney", "likes" : [ "a balanced budget", "corporations",
"binders full of women" ] }
```
# And you unwind the "likes" array of each document. How many documents will you wind up with?

6

{ ‘_id’ : ‘Barack Obama’ , ‘likes’ : ‘social justice’ }
{ ‘_id’ : ‘Barack Obama’ , ‘likes’ : ‘health care’ }
{ ‘_id’ : ‘Barack Obama’ , ‘likes’ : ‘taxes’ }
{ ‘_id’ : ‘Mitt Romney’ , ‘likes’ : ‘a balanced budget’ }
{ ‘_id’ : ‘Mitt Romney’ , ‘likes’ : ‘corporations’ }
{ ‘_id’ : ‘Mitt Romney’ , ‘likes’ : ‘binders full of women’ }