M101J Final Exam Answers

Hi readers,

Here are answers for MongoDb course session October 2015, Final Exam.

Final: Question 1

Please download the Enron email dataset enron.zip, unzip it and then restore it using mongorestore. It should restore to a collection called "messages" in a database called "enron". Note that this is an abbreviated version of the full corpus. There should be 120,477 documents after restore.

Inspect a few of the documents to get a basic understanding of the structure. Enron was an American corporation that engaged in a widespread accounting fraud and subsequently failed.

In this dataset, each document is an email message. Like all Email messages, there is one sender but there can be multiple recipients.

Construct a query to calculate the number of messages sent by Andrew Fastow, CFO, to Jeff Skilling, the president. Andrew Fastow's email addess was andrew.fastow@enron.com. Jeff Skilling's email was jeff.skilling@enron.com.

For reference, the number of email messages from Andrew Fastow to John Lavorato (john.lavorato@enron.com) was 1.

Answer :- 3

Steps :-

```
mongoimport -d enron -c messages > enron.json
```

```
use enron
db.messages.find({"headers.To":"andrew.fastow@enron.com","headers.From":"jeff.skilling@enron.com"}).count();
```

Final: Question 2

Please use the Enron dataset you imported for the previous problem. For this question you will use the aggregation framework to figure out pairs of people that tend to communicate a lot. To do this, you will need to unwind the To list for each message.

This problem is a little tricky because a recipient may appear more than once in the To list for a message. You will need to fix that in a stage of the aggregation before doing your grouping and counting of (sender, recipient) pairs.

Which pair of people have the greatest number of messages in the dataset?

susan.mara@enron.com to jeff.dasovich@enron.com
susan.mara@enron.com to richard.shapiro@enron.com
soblander@carrfut.com to soblander@carrfut.com
susan.mara@enron.com to james.steffes@enron.com
evelyn.metoyer@enron.com to kate.symes@enron.com
susan.mara@enron.com to alan.comnes@enron.com

Answer :-

susan.mara@enron.com to jeff.dasovich@enron.com

Steps:-

Run query on mongo shell

```
db.messages.aggregate([
{
 $project: {
 from: "$headers.From",
 to: "$headers.To"
}
},
{
 $unwind: "$to"
},
{
 $group : { _id : { _id: "$_id", from: "$from", to: "$to" }
}
},
{
 $group : { _id : { from: "$_id.from", to: "$_id.to" }, count:
{$sum :1}
}
},
{
 $sort : {count:-1}
},
{
 $limit: 2
}
])
```

Final: Question 3

This is a Hands On problem. In this problem, the database will begin in an initial state, you will manipulate it, and we will verify that the database is in the correct final state when you click 'submit'. If you need to start over at any point, you can click 'reset' to re-initialize the database, but this will not change your answer if you have already clicked 'submit'. If you wish to change your answer, get the database into the correct state, and then click 'submit'. If you leave the question and come back, the database will re-initialize. If you have clicked the 'submit' button at least once, you will see the word "Submitted" below the shell.

In this problem you will update a document in the messages collection to illustrate your mastery of updating documents from the shell. In fact, we've created a collection with a very similar schema to the Enron dataset, but filled instead with randomly generated data.

Please add the email address "mrpotatohead@10gen.com" to the list of addresses in the "headers.To" array for the document with "headers.Message-ID" of "<8147308.1075851042335.JavaMail.evans@thyme>"

Answer :- 897h6723ghf25gd87gh28

Steps:-

Run query on embedded mongo shell

```
db.messages.update({"headers.Message-ID":"<8147308.1075851042335.JavaMail.evans@thyme>"},
{$addToSet:{"headers.To":"mrpotatohead@10gen.com"}})
```

Final: Question 4

Enhancing the Blog to support viewers liking certain comments
In this problem, you will be enhancing the blog project to support

users liking certain comments and the like counts showing up the in the permalink page.

Start by downloading the code in final-problem4.zip and posts.json files from the Download Handout link. Load up the blog dataset posts.json. The user interface has already been implemented for you. It's not fancy. The /post URL shows the like counts next to each comment and displays a Like button that you can click on. That Like button POSTS to the /like URL on the blog, makes the necessary changes to the database state (you are implementing this), and then redirects the browser back to the permalink page.

This full round trip and redisplay of the entire web page is not how you would implement liking in a modern web app, but it makes it easier for us to reason about, so we will go with it.

Your job is to search the code for the string "XXX work here" and make the necessary changes. You can choose whatever schema you want, but you should note that the entry_template makes some assumptions about the how the like value will be encoded and if you go with a different convention than it assumes, you will need to make some adjustments.

MongoProc will fetch your blog, go to the first post's permalink page and attempt to increment the vote count. Remember that the blog needs to be running as well as MongoDB.

Answer :- 983nf93ncafjn20fn10f

Steps:-

You need to make changes to BlogPostDAO.java file.

Look for the method saying "XXX work here" and write

```
postsCollection.update(new BasicDBObject("permalink",
permalink), new BasicDBObject("$inc", new
BasicDBObject("comments." + ordinal + ".num_likes", 1)));
```

On mongoproc run test. if all works fine then it will tell you that you have solved this question and ask you to submit.

Final: Question 5

Suppose your have a collection stuff which has the _id index,

```
 {
   "v" : 1,
   "key" : {
     "_id" : 1
   },
   "ns" : "test.stuff",
   "name" : "_id_"
 }
```

and one or more of the following indexes as well:

```
 {
   "v" : 1,
   "key" : {
     "a" : 1,
     "b" : 1
   },
   "ns" : "test.stuff",
   "name" : "a_1_b_1"
 }
 {
   "v" : 1,
   "key" : {
     "a" : 1,
     "c" : 1
   },
   "ns" : "test.stuff",
   "name" : "a_1_c_1"
 }
 {
   "v" : 1,
   "key" : {
     "c" : 1
   },
   "ns" : "test.stuff",
   "name" : "c_1"
 }
 {
   "v" : 1,
   "key" : {
     "a" : 1,
     "b" : 1,
     "c" : -1
   },
   "ns" : "test.stuff",
   "name" : "a_1_b_1_c_-1"
```

}
Now suppose you want to run the following query against the collection.

db.stuff.find({'a':{'$lt':10000}, 'b':{'$gt': 5000}}, {'a':1, 'c':1}).sort({'c':-1})
Which of the indexes could be used by MongoDB to assist in answering the query? Check all that apply.

- c_1
- a_1_b_1
- a_1_c_1
- a_1_b_1_c_-1
- _id_

Answer :-

Correct answers are

- c_1
- a_1_b_1
- a_1_c_1
- a_1_b_1_c_-1

Final: Question 6

Suppose you have a collection of students of the following form:
{
 "_id" : ObjectId("50c598f582094fb5f92efb96"),
 "first_name" : "John",
 "last_name" : "Doe",
 "date_of_admission" : ISODate("2010-02-21T05:00:00Z"),
 "residence_hall" : "Fairweather",
 "has_car" : true,

```
 "student_id" : "2348023902",
 "current_classes" : [
  "His343",
  "Math234",
  "Phy123",
  "Art232"
 ]
}
```

Now suppose that basic inserts into the collection, which only include the last name, first name and student_id, are too slow (we can't do enough of them per second from our program). What could potentially improve the speed of inserts. Check all that apply.

- Add an index on last_name, first_name if one does not already exist.
- Remove all indexes from the collection, leaving only the index on _id in place
- Provide a hint to MongoDB that it should not use an index for the inserts
- Set w=0, j=0 on writes
- Build a replica set and insert data into the secondary nodes to free up the primary nodes.

Answer :-

Correct answers are

- Remove all indexes from the collection, leaving only the index on _id in place
- Set w=0, j=0 on writes

Final: Question 7

You have been tasked to cleanup a photo-sharing database. The database consists of two collections, albums, and images. Every image is supposed to be in an album, but there are orphan images that appear in no album.

Here are some example documents (not from the collections you will be downloading).

```
> db.albums.findOne()
{
 "_id" : 67
 "images" : [
  4745,
  7651,
  15247,
  17517,
  17853,
  20529,
  22640,
  27299,
  27997,
  32930,
  35591,
  48969,
  52901,
  57320,
  96342,
  99705
 ]
}
```

```
> db.images.findOne()
{ "_id" : 99705, "height" : 480, "width" : 640, "tags" : [ "dogs",
"kittens", "work" ] }
```

From the above, you can conclude that the image with _id = 99705 is in album 67. It is not an orphan.

Your task is to write a program to remove every image from the images collection that appears in no album. Or put another way, if an image does not appear in at least one album, it's an orphan and should be removed from the images collection.

Download final7.zip from Download Handout link and use mongoimport to import the collections in albums.json and images.json.

When you are done removing the orphan images from the collection, there should be 89,737 documents in the images collection.

Answer :- 44,787

Final: Question 8

Supposed we executed the following Java code. How many animals will be inserted into the "animals" collection?

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;

public class InsertTest {
        public static void main(String[] args) {
            MongoClient c =  new MongoClient();
            MongoDatabase db = c.getDatabase("test");
            MongoCollection<Document> animals =
db.getCollection("animals");

           Document animal = new Document("animal", "monkey");

            animals.insertOne(animal);
            animal.remove("animal");
            animal.append("animal", "cat");
            animals.insertOne(animal);
            animal.remove("animal");
            animal.append("animal", "lion");
            animals.insertOne(animal);
        }
}
```

Answer :- 1

Final: Question 9

Imagine an electronic medical record database designed to hold the medical records of every individual in the United States. Because each person has more than 16MB of medical history and records, it's not feasible to have a single document for every patient. Instead, there is a patient collection that contains basic information on each person and maps the person to a patient_id, and a record collection that contains one document for each test or procedure. One patient may have dozens or even hundreds of documents in the record collection.

We need to decide on a shard key to shard the record collection. What's the best shard key for the record collection, provided that we are willing to run inefficient scatter-gather operations to do infrequent research and run studies on various diseases and cohorts? That is, think mostly about the operational aspects of such a system. And by operational, we mean, think about what the most common operations that this systems needs to perform day in and day out.

- patient_id
- _id
- Primary care physician (your principal doctor that handles everyday problems)
- Date and time when medical record was created
- Patient first name
- Patient last name

Answer :- patient_id

Final: Question 10

Understanding the output of explain

We perform the following query on the enron dataset:

var exp = db.messages.explain('executionStats')

exp.find({ 'headers.Date' : { '$gt' : new Date(2001,3,1) } }, { 'headers.From' : 1, '_id' : 0 }).sort({ 'headers.From' : 1 })
and get the following explain output.

```
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "enron.messages",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "headers.Date" : {
        "$gt" : ISODate("2001-04-01T05:00:00Z")
      }
    },
    "winningPlan" : {
      "stage" : "PROJECTION",
      "transformBy" : {
        "headers.From" : 1,
        "_id" : 0
      },
      "inputStage" : {
        "stage" : "FETCH",
        "filter" : {
          "headers.Date" : {
            "$gt" : ISODate("2001-04-01T05:00:00Z")
          }
        },
        "inputStage" : {
          "stage" : "IXSCAN",
          "keyPattern" : {
            "headers.From" : 1
          },
          "indexName" : "headers.From_1",
          "isMultiKey" : false,
          "direction" : "forward",
          "indexBounds" : {
            "headers.From" : [
              "[MinKey, MaxKey]"
            ]
          }
        }
      }
    },
    "rejectedPlans" : [ ]
  },
  "executionStats" : {
    "executionSuccess" : true,
    "nReturned" : 83057,
    "executionTimeMillis" : 726,
    "totalKeysExamined" : 120477,
    "totalDocsExamined" : 120477,
    "executionStages" : {
      "stage" : "PROJECTION",
      "nReturned" : 83057,
      "executionTimeMillisEstimate" : 690,
      "works" : 120478,
      "advanced" : 83057,
      "needTime" : 37420,
      "needFetch" : 0,
      "saveState" : 941,
      "restoreState" : 941,
      "isEOF" : 1,
      "invalidates" : 0,
      "transformBy" : {
        "headers.From" : 1,
        "_id" : 0
      },
      "inputStage" : {
        "stage" : "FETCH",
        "filter" : {
          "headers.Date" : {
            "$gt" : ISODate("2001-04-01T05:00:00Z")
          }
        },
        "nReturned" : 83057,
        "executionTimeMillisEstimate" : 350,
        "works" : 120478,
        "advanced" : 83057,
        "needTime" : 37420,
        "needFetch" : 0,
        "saveState" : 941,
        "restoreState" : 941,
        "isEOF" : 1,
        "invalidates" : 0,
        "docsExamined" : 120477,
        "alreadyHasObj" : 0,
        "inputStage" : {
          "stage" : "IXSCAN",
          "nReturned" : 120477,
          "executionTimeMillisEstimate" : 60,
          "works" : 120477,
          "advanced" : 120477,
          "needTime" : 0,
          "needFetch" : 0,
          "saveState" : 941,
          "restoreState" : 941,
          "isEOF" : 1,
          "invalidates" : 0,
          "keyPattern" : {
            "headers.From" : 1
          },
          "indexName" : "headers.From_1",
          "isMultiKey" : false,
          "direction" : "forward",
          "indexBounds" : {
            "headers.From" : [
              "[MinKey, MaxKey]"
            ]
          },
        "keysExamined" : 120477,
        "dupsTested" : 0,
        "dupsDropped" : 0,
        "seenInvalidated" : 0,
        "matchTested" : 0
       }
      }
     }
   },
   "serverInfo" : {
     "host" : "dpercy-mac-air.local",
     "port" : 27017,
     "version" : "3.0.1",
     "gitVersion" :
"534b5a3f9d10f00cd27737fbcd951032248b5952"
   },
   "ok" : 1
}
```

Check below all the statements that are true about the way MongoDB handled this query.

- The query scanned every document in the collection.
- The query returned 120,477 documents.
- The query avoided sorting the documents because it was able to use an index's ordering.
- The query used an index to figure out which documents match the find criteria.

Answer :-

- The query did not utilize an index to figure out which documents match the find criteria.
- The query used an index for the sorting phase.
- The query performed a full collection scan

Posted by Kunal Saxena at 12/01/2015 01:25:00 PM
Labels: MongoDB

M101J: MongoDB for Java Developers : Homework : Week 6

Hi readers,

Here are answers for MongoDb course session October 2015, Week 6.

Homework 6.1

Which of the following statements are true about replication in MongoDB? Check all that apply.

- The minimum sensible number of voting nodes to a replica set is three.
- MongoDB replication is synchronous.
- By default, using the new MongoClient connection class, w=1 and j=1.
- The oplog utilizes a capped collection.

Answer :-

Correct options are

- The minimum sensible number of voting nodes to a replica set is three.
- The oplog utilizes a capped collection.

Homework 6.2

Let's suppose you have a five member replica set and want to assure that writes are committed to the journal and are acknowledged by at least 3 nodes before you proceed forward. What would be the appropriate settings for w and j?

- w=1, j=1

- w="majority", j=1
- w=3, j=0
- w=5, j=1
- w=1,j=3

Answer :-

Correct option

w='majority', j=1

Homework 6.3

Which of the following statements are true about choosing and using a shard key?

- You can change the shard key on a collection if you desire.
- MongoDB can not enforce unique indexes on a sharded collection other than the shard key itself, or indexes prefixed by the shard key.
- Any update that does not contain the shard key will be sent to all shards.
- There must be a index on the collection that starts with the shard key.
- The shard key must be unique

Answer :-

Correct options are

- There must be a index on the collection that starts with the shard key.
-
- Mongo can not enforce unique indexes on a sharded collection other than the shard key itself.

-
- Any update that does not contain the shard key will be sent to all shards.

Homework 6.4

You have a sharded system with three shards and have sharded the collections "students" in the "school" database across those shards. The output of sh.status() when connected to mongos looks like this:

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5531512ac723271f602db407")
}
  shards:
        {  "_id" : "s0",  "host" :
"s0/localhost:37017,localhost:37018,localhost:37019" }
        {  "_id" : "s1",  "host" :
"s1/localhost:47017,localhost:47018,localhost:47019" }
        {  "_id" : "s2",  "host" :
"s2/localhost:57017,localhost:57018,localhost:57019" }
  balancer:
        Currently enabled:  yes
        Currently running:  yes
                Balancer lock taken at Fri Apr 17 2015 14:32:02 GMT-
0400 (EDT) by education-iMac-
2.local:27017:1429295401:16807:Balancer:1622650073
        Collections with active migrations:
                school.students started at Fri Apr 17 2015 14:32:03
GMT-0400 (EDT)
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                2 : Success
                1 : Failed with error 'migration already in progress',
from s0 to s1
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config"
}
        {  "_id" : "school",  "partitioned" : true,  "primary" : "s0" }
                school.students
                        shard key: { "student_id" : 1 }
                        chunks:
```

```
                s0    1
                s1    3
                s2    1
            { "student_id" : { "$minKey" : 1 } } -->> {
"student_id" : 0 } on : s2 Timestamp(3, 0)
            { "student_id" : 0 } -->> { "student_id" : 2 } on :
s0 Timestamp(3, 1)
            { "student_id" : 2 } -->> { "student_id" : 3497 } on
: s1 Timestamp(3, 2)
            { "student_id" : 3497 } -->> { "student_id" : 7778
} on : s1 Timestamp(3, 3)
            { "student_id" : 7778 } -->> { "student_id" : {
"$maxKey" : 1 } } on : s1 Timestamp(3, 4)
```

If you ran the query
use school
db.students.find({'student_id':2000})
Which shards would be involved in answering the query?

- s0, s1, and s2
- s0
- s1
- s2

Answer :-

Correct option

s1

Homework 6.5 (MongoProc)

Preface to Homework 6.5 (MongoProc)

In this homework you will build a small replica set on your own computer. We will check that it works with MongoProc.

Create three directories for the three mongod processes. On unix, this could be done as follows:

```
mkdir -p /data/rs1 /data/rs2 /data/rs3
```

or on Windows:

```
mkdir \data\rs1 \data\rs2 \data\rs3
```

Now start three mongo instances as follows. Note that are three commands. The browser is probably wrapping them visually.

Linux and Mac users:

```
mongod --replSet m101 --logpath "1.log" --dbpath /data/rs1 --port 27017 --smallfiles --oplogSize 64 --fork

mongod --replSet m101 --logpath "2.log" --dbpath /data/rs2 --port 27018 --smallfiles --oplogSize 64 --fork

mongod --replSet m101 --logpath "3.log" --dbpath /data/rs3 --port 27019 --smallfiles --oplogSize 64 --fork
```

Windows users:

```
start mongod --replSet m101 --logpath 1.log --dbpath \data\rs1 --port 27017 --smallfiles --oplogSize 64
start mongod --replSet m101 --logpath 2.log --dbpath \data\rs2 --port 27018 --smallfiles --oplogSize 64
start mongod --replSet m101 --logpath 3.log --dbpath \data\rs3 --port 27019 --smallfiles --oplogSize 64
```

Now connect to a mongo shell and make sure it comes up.

```
mongo --port 27017
```

Now you will create the replica set. Type the following commands into the mongo shell:

```
config = { _id: "m101", members:[
          { _id : 0, host : "localhost:27017"},
          { _id : 1, host : "localhost:27018"},
          { _id : 2, host : "localhost:27019"} ]
};
rs.initiate(config);
```

At this point, the replica set should be coming up. You can type

```
rs.status()
```

to see the state of replication.

Answer :-

All you need to do in this assignment is to follow steps given in question as per your platform type Windows/Mac?linux.

After following all the steps, run mongoproc and click "test" button. If everything worked fine then it will show you message like

Everything looks fine. Three instances of mongodb running.
On Mongoproc

Posted by Kunal Saxena at 11/18/2015 04:57:00 PM
Labels: MongoDB

G+1 +2 Recommend this on Google

M101J: MongoDB for Java Developers : Homework : Week 5

Hi readers,

Here are answers for MongoDb course session October 2015, Week 5.

Homework 5.1 (Hands On)

Finding the most frequent author of comments on your blog

In this assignment you will use the aggregation framework to find the most frequent author of comments on your blog. We will be using the same basic dataset as last week, with posts and comments shortened considerably, and with many fewer documents in the collection in order to streamline the operations of the Hands On web shell.

Use the aggregation framework in the web shell to calculate the author with the greatest number of comments.

Just to clarify, the data set for this week is not available for download.

To help you verify your work before submitting, the author with the fewest comments is Cody Strouth and he commented 68 times.

Once you've found the correct answer with your query, please choose your answer below for the most prolific comment author.

Answer :-

Run the given command on website embedded shell

```
db.posts.aggregate([
    { $unwind: "$comments" },
    { $group: { _id: "$comments.author", count: { $sum: 1 } } },
    { $sort: { count: -1 } },
    { $limit: 1 }
])
```

So answer is Gisela Levin.

Homework 5.2 (Hands On)

Crunching the Zipcode dataset
Please calculate the average population of cities in California (abbreviation CA) and New York (NY) (taken together) with populations over 25,000.

For this problem, assume that a city name that appears in more than one state represents two separate cities.

Please round the answer to a whole number.
Hint: The answer for CT and NJ (using this data set) is 38177.

Please note:
Different states might have the same city name.
A city might have multiple zip codes.

For purposes of keeping the Hands On shell quick, we have used a subset of the data you previously used in zips.json, not the full set. This is why there are only 200 documents (and 200 zip codes), and all of them are in New York, Connecticut, New Jersey, and California.

If you prefer, you may download the handout and perform your analysis on your machine with

```
> mongoimport -d test -c zips --drop small_zips.json
```

Once you've generated your aggregation query and found your answer, select it from the choices below.

Answer :-

Run the given command on website embedded shell

```
db.zips.aggregate([
    { $match: {$or: [ {state: "CA"}, {state: "NY"} ] } },
    { $group: { _id: { city: "$city" }, pop: { $sum: "$pop" } } },
    { $match: { "pop": { $gt: 25000 } } },
    { $group: { _id: null, avg_pop_of_city: { $avg: "$pop" } } }
])
```

So correct Answer is 44805

Homework 5.3 (Hands On)

Who's the easiest grader on campus?
A set of grades are loaded into the grades collection.

The documents look like this:

```
{
 "_id" : ObjectId("50b59cd75bed76f46522c392"),
 "student_id" : 10,
 "class_id" : 5,
 "scores" : [
  {
   "type" : "exam",
   "score" : 69.17634380939022
  },
  {
   "type" : "quiz",
   "score" : 61.20182926719762
  },
  {
   "type" : "homework",
   "score" : 73.3293624199466
  },
  {
   "type" : "homework",
   "score" : 15.206314042622903
  },
  {
   "type" : "homework",
   "score" : 36.75297723087603
  },
  {
   "type" : "homework",
   "score" : 64.42913107330241
  }
 ]
}
```

There are documents for each student (student_id) across a variety of classes (class_id). Note that not all students in the same class have the same exact number of assessments. Some students have three homework assignments, etc.

Your task is to calculate the class with the best average student performance. This involves calculating an average for each student in each class of all non-quiz assessments and then averaging those numbers to get a class average. To be clear, each student's average includes only exams and homework grades. Don't include their quiz scores in the calculation.

What is the class_id which has the highest average student performance?

Hint/Strategy: You need to group twice to solve this problem. You must figure out the GPA that each student has achieved in a class and then average those numbers to get a class average. After that, you just need to sort. The class with the lowest average is the class with class_id=2. Those students achieved a class average of 37.6

If you prefer, you may download the handout and perform your analysis on your machine with
> mongoimport -d test -c grades --drop grades.json

Answer :-

Run the given command on website embedded shell

```
db.grades.aggregate([
   { $unwind: "$scores" },
   { $match: { $or: [ {"scores.type": "homework"},
{"scores.type":"exam"} ] } },
   { $group: { _id: { 'student_id': "$student_id", 'class_id':
"$class_id" }, avg: { $avg: "$scores.score" } } },
   { $group: { _id: "$_id.class_id", class_avg: { $avg: "$avg" } }
},
   { $sort: { 'class_avg': -1 } }
])
```

So correct answer is 1.

Homework 5.4

Removing Rural Residents

In this problem you will calculate the number of people who live in a zip code in the US where the city starts with a digit. We will take that to mean they don't really live in a city. Once again, you will be using the zip code collection, which you will find in the 'handouts' link in this page. Import it into your mongod using the following command from the command line:

```
> mongoimport -d test -c zips --drop zips.json
```

If you imported it correctly, you can go to the test database in the mongo shell and conform that

```
> db.zips.count()
```

yields 29,467 documents.

The project operator can extract the first digit from any field. For example, to extract the first digit from the city field, you could write this query:

```
db.zips.aggregate([
    {$project:
     {
 first_char: {$substr : ["$city",0,1]},
     }
   }
])
```

Using the aggregation framework, calculate the sum total of people who are living in a zip code where the city starts with a digit. Choose the answer below.

Answer :-

It is time to run your own local mongo server and import database as given in above steps. Follow other steps as well.

```
db.zips.aggregate([
    { $project: { _id: 0, city: 1, pop: 1 } },
    { $match: { city: /^\d.*/ } },
    { $group: { _id: null, pop: { $sum: "$pop" } } },
    { $sort: { city: 1} }
])
```

So answer you will get is 298015

Posted by Kunal Saxena at 11/16/2015 10:18:00 PM

Labels: MongoDB

G+1 +2 Recommend this on Google

M101J: MongoDB for Java Developers : Homework : Week 4

Hi readers,

Here are answers for MongoDb course session October 2015, Week 4.

Homework 4.1

Suppose you have a collection with the following indexes:

```
> db.products.getIndexes()
[
 {
  "v" : 1,
  "key" : {
   "_id" : 1
  },
  "ns" : "store.products",
  "name" : "_id_"
 },
 {
  "v" : 1,
  "key" : {
   "sku" : 1
  },
              "unique" : true,
  "ns" : "store.products",
  "name" : "sku_1"
 },
 {
  "v" : 1,
  "key" : {
   "price" : -1
  },
  "ns" : "store.products",
  "name" : "price_-1"
 },
 {
  "v" : 1,
  "key" : {
   "description" : 1
  },
  "ns" : "store.products",
  "name" : "description_1"
 },
 {
  "v" : 1,
  "key" : {
   "category" : 1,
   "brand" : 1
  },
  "ns" : "store.products",
  "name" : "category_1_brand_1"
 },
 {
  "v" : 1,
  "key" : {
   "reviews.author" : 1
  },
  "ns" : "store.products",
  "name" : "reviews.author_1"
}
```

Which of the following queries can utilize at least one index to find all matching documents or to sort? Check all that apply.

Correct Answers are :-

- db.products.find({ 'brand' : "GE" }).sort({ price : 1 })
- db.products.find({ $and : [{ price : { $gt : 30 } }, { price : { $lt : 50 } }] }).sort({ brand : 1 })

Homework 4.2

Suppose you have a collection called *tweets* whose documents contain information about the*created_at* time of the tweet and the user's *followers_count* at the time they issued the tweet. What can you infer from the following *explain* output?

Correct Answers are :-

- The query examines 251120 documents.
- The query uses an index to determine the order in which to return result documents.

Homework 4.3

Making the Blog fast

Please download hw4-3.zip from the Download Handout link to get started. This assignment requires MongoDB 2.2 or above.

In this homework assignment you will be adding some indexes to the post collection to make the blog fast.

We have provided the full code for the blog application and you don't need to make any changes, or even run the blog. But you can, for fun.

We are also providing a patriotic (if you are an American) data set for the blog. There are 1000 entries with lots of comments and tags. You must load this dataset to complete the problem.

```
# from the mongodb shell
use blog
db.posts.drop()
# from the terminal window
mongoimport -d blog -c posts < posts.json
```

The blog has been enhanced so that it can also display the top 10 most recent posts by tag. There are hyperlinks from the post tags to the page that displays the 10 most recent blog entries for that tag. (run the blog and it will be obvious)

Your assignment is to make the following blog pages fast:

- The blog home page
- The page that displays blog posts by tag (http://localhost:8082/tag/whatever)
- The page that displays a blog entry by permalink (http://localhost:8082/post/permalink)

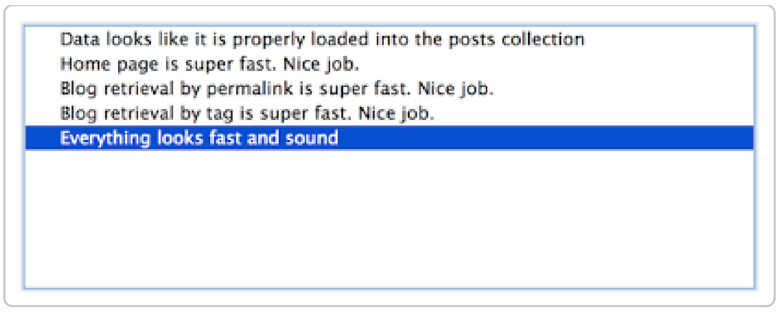
By fast, we mean that indexes should be in place to satisfy these queries such that we only need to scan the number of documents we are going to return.

Answer:-

I assume that you have imported the posts.json file using the steps given in problem itself. Now you need to open mongo shell and fire some queries like

- db.posts.ensureIndex({ date: -1})
- db.posts.ensureIndex({ permalink: 1}, {unique: true})
- db.posts.ensureIndex({ tags: 1})

You do not need to make any other changes to blog code. Above mentioned lines will create indexes and the you can check your mongoproc for solution.
Browse week 4.3 and hit test in your mongoproc. It will show you text like



```
Data looks like it is properly loaded into the posts collection
Home page is super fast. Nice job.
Blog retrieval by permalink is super fast. Nice job.
Blog retrieval by tag is super fast. Nice job.
Everything looks fast and sound
```

Submit your answer and Done.

Homework 4.4

In this problem you will analyze a profile log taken from a mongoDB instance. To start, please download sysprofile.json from Download Handout link and import it with the following command:

```
mongoimport -d m101 -c profile < sysprofile.json
```

Now query the profile data, looking for all queries to the *students* collection in the database*school2*, sorted in order of decreasing latency. What is the latency of the longest running operation to the collection, in milliseconds?

Correct Answer is :-

15820

Please subscribe to this blog for further updates. You can also like Techiekunal Facebook Page

Posted by Kunal Saxena at 11/08/2015 10:32:00 AM

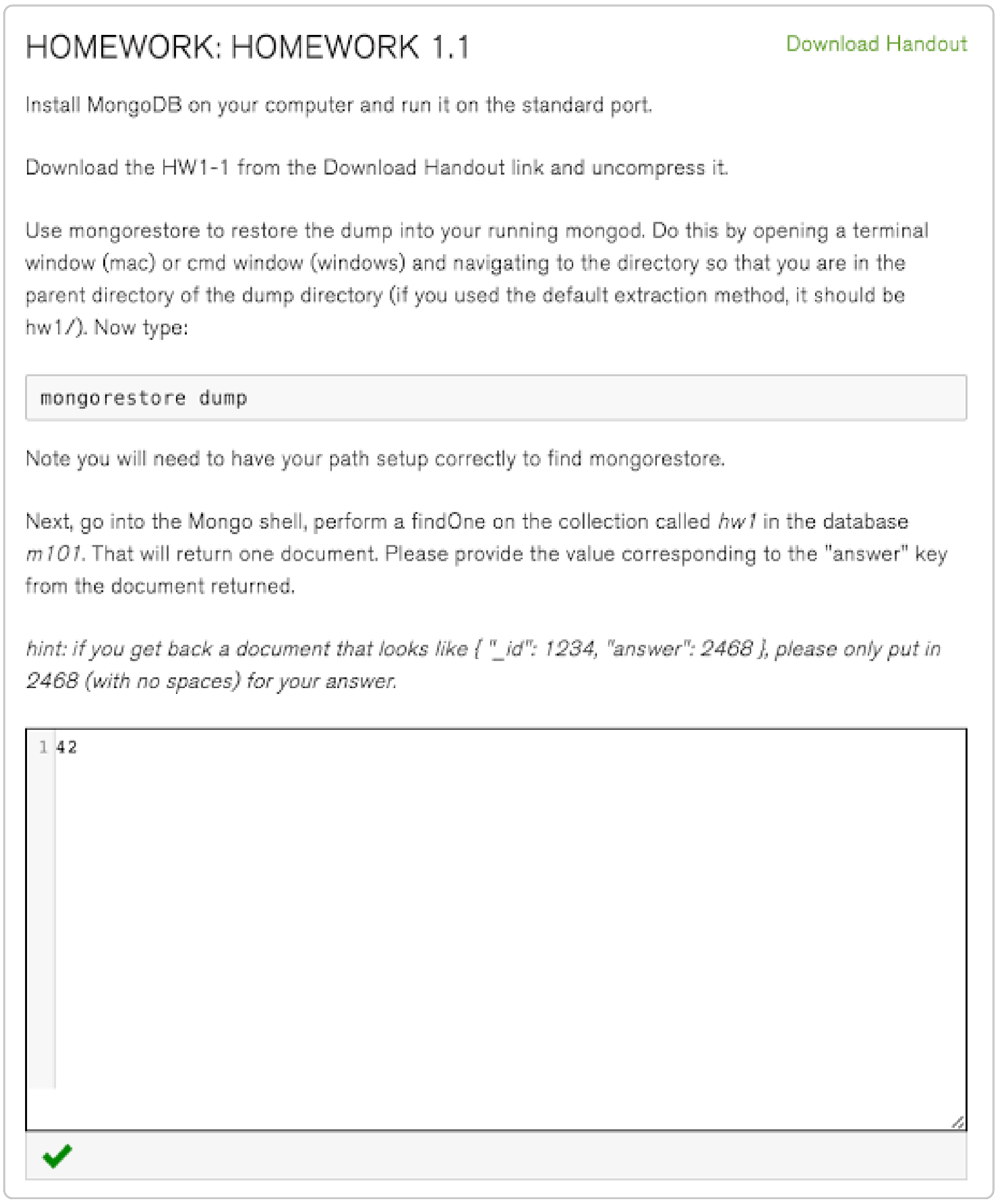
Labels: MongoDB

M101J: MongoDB for Java Developers : Homework and Answers

M101J: MongoDB for Java Developers : Homework and Answers

Class started Oct 12, 2015

Homework 1.1



HOMEWORK: HOMEWORK 1.1

Install MongoDB on your computer and run it on the standard port.

Download the HW1-1 from the Download Handout link and uncompress it.

Use mongorestore to restore the dump into your running mongod. Do this by opening a terminal window (mac) or cmd window (windows) and navigating to the directory so that you are in the parent directory of the dump directory (if you used the default extraction method, it should be hw1/). Now type:

```
mongorestore dump
```

Note you will need to have your path setup correctly to find mongorestore.

Next, go into the Mongo shell, perform a findOne on the collection called *hw1* in the database *m101*. That will return one document. Please provide the value corresponding to the "answer" key from the document returned.

hint: if you get back a document that looks like { "_id": 1234, "answer": 2468 }, please only put in 2468 (with no spaces) for your answer.

```
42
```

✓

Steps to find answers:-

- Start mongo service using "mongod" command
- go to path homework_1_1/hw1-1 and run "mongorestore". You will see messages like

```
finished restoring m101.hw1 (1 document)
finished restoring m101.funnynumbers (100 documents)
done
```

- Now run "mongo" and then run these commands one by one

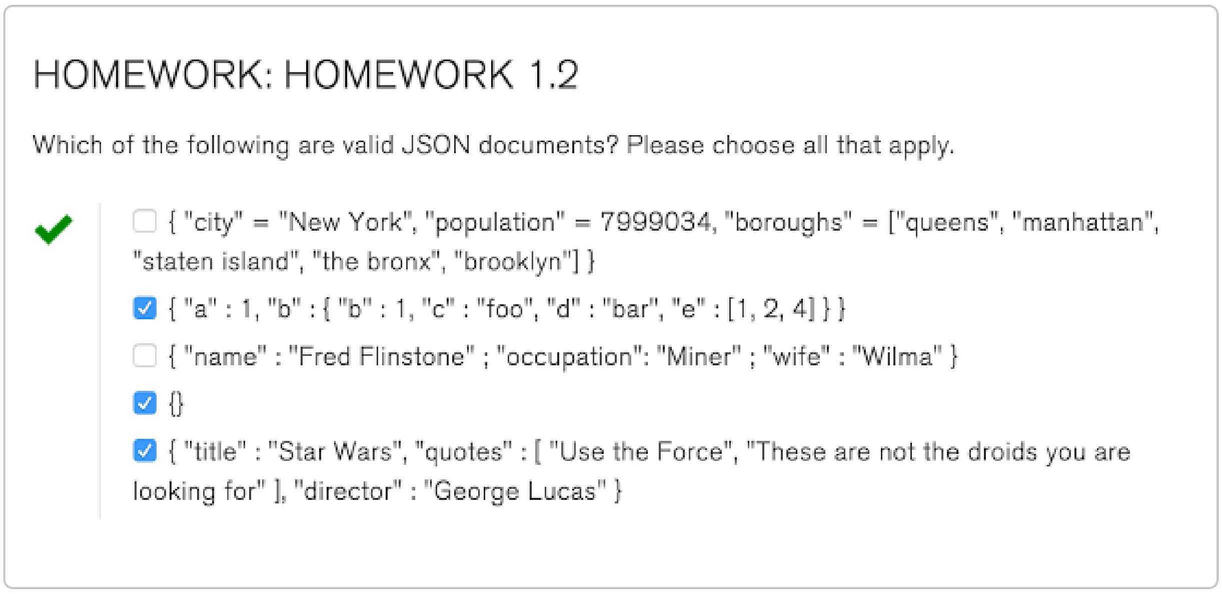
```
use m101
db.hw1.findOne()
```

In answer you will get

```
{
 "_id" : ObjectId("50773061bf44c220307d8514"),
 "answer" : 42,
 "question" : "The Ultimate Question of Life, The Universe and Everything"
}
```

Answer : 42

Homework 1.2



HOMEWORK: HOMEWORK 1.2

Which of the following are valid JSON documents? Please choose all that apply.

✓

- [ ] { "city" = "New York", "population" = 7999034, "boroughs" = ["queens", "manhattan", "staten island", "the bronx", "brooklyn"] }
- [✓] { "a" : 1, "b" : { "b" : 1, "c" : "foo", "d" : "bar", "e" : [1, 2, 4] } }
- [ ] { "name" : "Fred Flinstone" ; "occupation": "Miner" ; "wife" : "Wilma" }
- [✓] {}
- [✓] { "title" : "Star Wars", "quotes" : ["Use the Force", "These are not the droids you are looking for"], "director" : "George Lucas" }

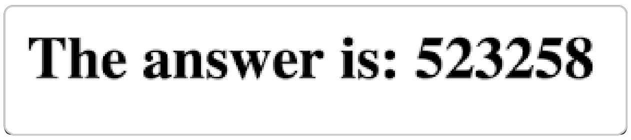
Answers are :-

{ "a" : 1, "b" : { "b" : 1, "c" : "foo", "d" : "bar", "e" : [1, 2, 4] } }

{}

{ "title" : "Star Wars", "quotes" : ["Use the Force", "These are not the droids you are looking for"], "director" : "George Lucas" }

Homework 1.3



The answer is: 523258

- Install and set up maven according to the mongo university video tutorial.
- Run command mvn compile exec:java -Dexec.mainClass=com.mongodb.SparkHomework

You will have output like

```
== Spark has ignited ...
>> Listening on 0.0.0.0:4567
167 [Thread-1] INFO org.eclipse.jetty.server.Server - jetty-9.0.2.v20130417
263 [Thread-1] INFO org.eclipse.jetty.server.ServerConnector - Started ServerConnector@12d6bb71{HTTP/1.1}{0.0.0.0:4567}
```

- Now go to address 0.0.0.0:4567 in your browser

You will find the answer

Answer is : 523258

Please like, share and put your queries in comments.

Posted by Kunal Saxena at 10/15/2015 12:58:00 AM

Labels: MongoDB

G+1 +2 Recommend this on Google