Big Data Systems:
MongoDB Homework Assignment

Instructions

The objective of this assignment is to hone your skills with MongoDB. This is an individual assignment; you are not allowed to work in groups. The assignment consists of three parts. In the first part you will focus on building up your technique with writing queries in MongoDB, and on using the aggregate pipeline in particular. In the second part of the assignment you will leverage your knowledge of R to setup communication between R and MongoDB. In the third and final part you will learn how to write simple map-reduce queries in MongoDB.

Part One (Queries and the Aggregation Pipeline)

In this section we are going to focus on learning how to write complex queries in MongoDB. To do this, we are going to create the student database by loading the JavaScript file that we saw in class, which is provided with this assignment and with the notes in a file called prep.js.

1. Using the load() function inside the mongo shell, load the prep.js file

This will create a *students* collection in whatever database you are currently using. We will use only this collection for the remainder of this first part. Be advised that this script contains a random component so that you will each get a slightly different (but same structure) version of the students collection.

Here is an example of an object that could be from the students collection:

```
db.students.findOne()
{
        "_id" : ObjectId("558d08925e083d8cdd7be831"),
        "home_city" : "Kalamata",
        "first_name" : "Eirini",
        "hobbies" : [
                "skydiving",
                "guitar",
                "AD&D"
        ],
        "favourite_os" : "OS X",
        "laptop_cost" : 1506,
        "courses" : [
                {
                        "course_code" : "P102",
                        "course_title" : "Introduction to R",
                        "course_status" : "Complete",
                        "grade" : 10
                },
                {
                        "course_code" : "S102",
                        "course_title" : "Mathematical Statistics",
                        "course_status" : "In Progress"
                },
                {
                        "course_code" : "P201",
                        "course_title" : "Advanced R",
                        "course_status" : "In Progress"
                },
                {
                        "course_code" : "S202",
                        "course_title" : "Graph Theory",
                        "course_status" : "Complete",
```

```
                "grade" : 7
        },
        {
                "course_code" : "M102",
                "course_title" : "Data Mining",
                "course_status" : "In Progress"
        }
]
```

As you can see, this collection as we discussed in class is a collection that tracks the performance of different students enrolled in classes

2. For EACH of the following we want you to write a SINGLE command that will produce the desired result. We want you to either use db.students.find() function for the simpler questions, or to use db.students.aggregate() function (i.e. the aggregation pipeline). Consult the slides for these but also make sure you also consult the excellent online documentation, in particular for the aggregation pipeline (http://docs.mongodb.org/manual/core/aggregation-pipeline/).

- **How many students in your database are currently taking at least 1 class (i.e. have a class with a *course_status* of "In Progress")?**
- **Produce a grouping of the documents that contains the name of each home city and the number of students enrolled from that home city.**
- **Which hobby or hobbies are the most popular?**
- **What is the GPA (ignoring dropped classes and in progress classes) of the best student?**
- **Which student has the largest number of grade 10's?**
- **Which class has the highest average GPA?**
- **Which class has been dropped the most number of times?**
- **Produce of a count of classes that have been COMPLETED by class type. The class type is found by taking the first letter of the course code so that M102 has type M. So I basically want how many courses have been completed in type M, how many of type S, how many of type P etc… (HINT: check out the $substr function here: http://docs.mongodb.org/manual/reference/operator/aggregation/substr/)**
- **Produce a transformation of the documents so that the documents now have an additional boolean field called "hobbyist" that is true when the student has more than 3 hobbies and false otherwise.**
- **Produce a transformation of the documents so that the documents now have an additional field that contains the number of classes that the student has completed**
- **Produce a transformation of the documents in the collection so that they look like this:**

  {
  "_id" : ObjectId("558d08925e083d8cdd7be831"),
  "first_name" : "Eirini",
  "GPA" : 8.5

"classesInProgress" : 3
"droppedClasses" : 0
}
The GPA is the average grade of all the completed classes. The other two computed fields are the number of classes currently in progress and the number of classes dropped. No other fields should be in there. No other fields should be present.

- **Produce a NEW collection (HINT: Use $out in the aggregation pipeline) so that the new documents in this correspond to the classes on offer. The structure of the documents should be like this:**
  {
  "_id" : "M102"
  "course_title" : "Data Mining",
  "numberOfDropouts: 34
  "numberOfTimesCompleted: 34
  "currentlyRegistered": [ObjectId("558d08925e083d8cdd7be831"), …]
  "maxGrade" : 10
  "minGrade" : 2
  "avgGrade" : 7.6
  }

  The _id field should be the course code. The course_title is what it was before. The numberOfDropouts is the number of students who dropped out. The numberOfTimesCompleted is the number of students that completed this class. The currentlyRegistered array is an array of ObjectID's corresponding to the students who are currently taking the class. Finally, for the students that completed the class, the maxGrade, minGrade and avgGrade are the summary statistics for that class.

Please read each question carefully and make sure your answer for each of this produces only what is being asked. For example, look at the first question. This is clearly asking for a single number ("How many students…"). If I run the command you submit in your assignment for this question it should therefore just give me back a number not, say, a list of students with the number of classes they are taking. The third question asks "Which hobby or hobbies..," Try to give me back a list of documents that only has the names of the hobbies in question i.e. use appropriate projection. The same applies to the remainder of the questions. This exercise is excellent practice for building up expertise in writing queries and studying your data which I assure you is an integral part of doing data analysis in the workplace.

Part Two (R and MongoDB)

In this part we are going to show you how to communicate with MongoDB from R. This section is going to be relatively straightforward. Just follow the instructions in each paragraph and you will do fine.

In order to talk with Mongo we need to use a special package called rmongodb. There is an excellent cheatsheat for this here: http://cran.r-project.org/web/packages/rmongodb/vignettes/rmongodb_cheat_sheet.pdf We'll go through some of the more basic functionalities in this section and leave you to explore some of the more advanced features on your own.

1. Install and load the package rmongodb

Mongo should be running on port 27017 by default. In R, and also with other programming languages, we connect to a Mongo instance by creating and storing an object that represents the connection itself. We do this using the function mongo.create(). If we specify no arguments, all the default values are used and we should be able to connect to our local database immediately.

2. Open a connection to mongo and store the object in an object called mongo

At any time, you can check if the connection is still active by running the function mongo.is.connected() and passing it a mongo connection object that was previously created.

3. Check that you are connected to your mongo installation

Once you are connected let's first create a variable that will contain the name of our namespace as we will need this often. A namespace is the combination of database and collection name in the format <database name>.<collection name>

4. Create a string namespace variable to represent a mongo collection "lab2" in a database "r"

In Mongo, database objects are Binary JSON objects, also known as BSON objects. If we want to communicate with Mongo, we need a way to create and read BSON objects and this is exactly the functionality that the package provides. In the Mongo shell we can just write regular JSON objects and they are automatically converted to BSON objects. We can do something similar here. The way to create a JSON string object is to use single quotes, followed by curly brackets followed by a comma separated list of key-value pairs where the key is separated from the value using a colon.

We definitely need an example here to refresh our memory. Suppose we want a JSON object that has a city's name and population. Here it is (try it in R!):

```
JSON_string<-'{"name":"London", "population":"10.5 million"}'
```

The reason why we use single quotes is that we need regular quotes for the fields themselves and we need a way to tell R that the whole thing is a string object and this string object itself has quotes (another way is to use a backslash in front of every nested quote symbol)

5. Create a JSON string variable to represent a person whose name is Cristiano and whose language is Portuguese.

We can insert this person into R using the command mongo.insert(). This takes in a connection object, a string with the name of the namespace, and a BSON object. To create a BSON object from our JSON string just pass that string into the function mongo.bson.from.JSON() first.

6. Insert Cristiano into your MongoDB database and save the result of the call into a variable called ok

We can check whether the result was successful in R, by checking that the value of the ok variable is TRUE.

Let's now open a mongo shell using the mongo command and see if we really did something. Type the following to commands and make sure you see a single document with Cristiano.

```
use r
db.lab2.find()
```

The first command switches to the r database and the second is a find all command in the namespace lab2.

Another way to create a BSON object is from a list using the mongo.bson.from.list() command and passing it a list of key value pairs. For example, a valid list to represent the London document we worked before would be:

```
l<- list(name="London", population="10.5 million")
```

7. Create two new BSON objects to represent Ioanna, whose language is English and her age is 34, and Dimitris, whose language is Greek and his age is 29

Now we have more than one object that we want to put inside Mongo and this is a typical case. We can use the command mongo.insert.batch() to add multiple objects. This has the same syntax as mongo.insert() except that the final argument must be a list of BSON objects

8. Insert Ioanna and Dimitris in the database and use both the result of the call as well as the mongo shell to make sure you were successful

Let's say we want to update documents now. Remember that in Mongo, we must provide a document that describes all the documents we want to update and then provide a new document, which will replace these documents. We can issue a mongo update from R using the mongo.update() command. This takes a mongo connection object, the namespace, and can take two strings in JSON format. The first will represent the query and the second will be the updated document

9. Update Cristiano so that he now has an age of 26. Once again check your results both in R and in Mongo

We can also remove documents from R using mongo.remove(). This has the exact same syntax as mongo.update() except that we don't have the final argument which is a string in JSON format to replace the documents found with the query JSON.

10. Remove Dimitris from the database. Once again check your results both in R and in Mongo

We can run queries in mongo as well. We do this with the mongo.find() function. If we pass in just a connection and a namespace we will retrieve all the documents from the namespace. If, instead we also pass in a third argument with a JSON string describing the documents we want, we will retrieve documents that match that query. The trick is, that we receive an object that is a cursor. This is an object that we must repeatedly call a next() function on in order to get the next result. So, if we want to process a range of rows we will need a while loop as follows (shown here for a find all operation)

```
cursor <- mongo.find(mongo, namespace)
current_row_number <- 0

while(mongo.cursor.next(cursor)) {

	current_row_number<- current_row_number+1
	current_row<-cval = mongo.cursor.value(cursor)

	#Some code to do something with the row

}
```

Once we have the current_row object, we can extract a particular field from the document using the function mongo.bson.value(). For example, if we have an "age" field we get it by calling mongo.bson.value(current_row,"age")

11. Add some more people into the collection. Then, extract all the people from the collection using the code just given, and store them into a data frame.

Actually there is a function for doing this called mongo.cursor.to.data.frame but we want you to practice your coding skills and not use that function. Now, we have a way of getting data from mongo and processing it with R. Neat! We can also export data to a collection.

12. Write a function to store the contents of the heart data frame into a heart collection. You know everything you need to do this.

You've now seen the basics of working with MongoDB. As a final command, close your connection to Mongo by issuing the command mongo.destroy() and passing it your mongo connection object.

13. Close your MongoDB connection.

Part Three (MapReduce)

In this final part we want you to learn how to write map reduce queries in MongoDB. Please read the inline documentation at http://docs.mongodb.org/manual/core/map-reduce/ and also check out some examples at http://docs.mongodb.org/manual/tutorial/map-reduce-examples/. In general you will see that the syntax is simple and intuitive. To learn this functionality, we will ask you to write the following map reduce jobs:

1. Write a map reduce job on the students collection similar to the classic word count example. Specifically we want you to do a word count using the course title field as the text. In addition we want you to exclude stop words from this list and we will leave you to find/write your own list of stop words. Remember that stop words are the common words in the English language like "a", "in", "to" "the" etc…

2. Now write a map reduce job on the students collection whose goal is to compute average GPA scores for completed courses by home city and by course type (M, B, P, etc… please refer to question 2 of Part One)