

---

# mongoDB Project: Relational databases & Document-Oriented databases

---

Efstratios Gounidellis  
stratos.gounidellis [at] gmail.com

Lamprini Koutsokera  
lkoutsokera [at] gmail.com

Course: "Big Data Management Systems"  
Professor: Damianos Chatziantoniou

Department of Management Science & Technology

School of Business  
Athens University of Economics & Business

May 4, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Part One (Queries and the Aggregation Pipeline)</b>	<b>3</b>
2.1	part1_queries.js . . . . .	3
<b>3</b>	<b>Part Two (Python &amp; MongoDB)</b>	<b>12</b>
3.1	part2_python_mongodb.py . . . . .	12
<b>4</b>	<b>Part Three (MapReduce)</b>	<b>16</b>
4.1	part3_word_count.js . . . . .	16
4.2	part3_avg_grade.js . . . . .	16
	<b>References</b>	<b>18</b>

# 1 Introduction

This assignment is a part of a project implemented in the context of the course "Big Data Management Systems" taught by Prof. Chatziantoniou in the Department of Management Science and Technology (AUEB). The aim of the project is to familiarize the students with big data management systems such as Hadoop, Redis, MongoDB and Neo4j.

In the context of this assignment on Mongo, queries will be designed and executed on a mongo collection, simple operations on mongo will be executed with python while mapreduce jobs will also be designed and executed on a mongo collection.

## 2 Part One (Queries and the Aggregation Pipeline)

### 2.1 part1\_queries.js

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. The structure of an example object is like the following:

```
1 {
2   "_id": "ObjectId('558d08925e083d8cdd7be831')",
3   "home_city": "Kalamata",
4   "first_name": "Eirini",
5   "hobbies": [
6     "skydiving",
7     "guitar",
8     "AD&D"
9   ],
10  "favourite_os": "OS X",
11  "laptop_cost": 1506,
12  "courses": [{
13    "course_code": "P102",
14    "course_title": "Introduction to R",
15    "course_status": "Complete",
16    "grade": 10
17  },
18  {
19    "course_code": "S102",
20    "course_title": "Mathematical Statistics",
21    "course_status": "In Progress"
22  },
23  {
24    "course_code": "P201",
25    "course_title": "Advanced R",
26    "course_status": "In Progress"
27  },
28  {
29    "course_code": "S202",
```

```

30   "course_title": "Graph Theory",
31   "course_status": "Complete",
32   "grade": 7
33 },
34 {
35   "course_code": "M102",
36   "course_title": "Data Mining",
37   "course_status": "In Progress"
38 }
39 ]
40 }

```

The following queries are designed and expressed in mongo query language. The execution code for the queries can be found in the file queries.js.

```

1  /**
2   * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
3   * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
4   */
5
6  // Using the load() function inside the mongo shell, load the prep.js file
7  load("prep.js")
8
9  /* Q1: How many students in your database are currently taking at least 1
10 class (i.e. have a class with a course_status of "In Progress")?
11 */
12
13 db.students.find({'courses.course_status': 'In Progress'}).count()
14
15 /* Q2: Produce a grouping of the documents that contains the name of each
16 home city and the number of students enrolled from that home city.
17 */
18
19 db.students.aggregate(
20   [
21     {
22       "$group": {
23         _id: "$home_city",
24         enrolledStudents: {
25           $sum: 1
26         }
27       }
28     }
29   ]
30 )
31
32 // Q3: Which hobby or hobbies are the most popular?
33
34 db.students.aggregate(
35   [
36     {
37       $unwind: "$hobbies"
38     },
39     {

```

```

41         "$group": {
42             _id: "$hobbies",
43             popularity: {
44                 $sum: 1
45             }
46         }
47     },
48     {
49         $sort: {
50             popularity: -1
51         }
52     },
53     {
54         $limit: 1
55     }
56 ]
57 )
58
59 db.students.aggregate(
60 [
61     {
62         $unwind: "$hobbies"
63     },
64     {
65         "$group": {
66             _id: "$hobbies",
67             popularity: {
68                 $sum: 1
69             }
70         }
71     },
72     {
73         $sort: {
74             popularity: -1
75         }
76     },
77     {
78         $limit: 5
79     }
80 ]
81 )
82
83 /* Q4: What is the GPA (ignoring dropped classes and in progress classes)
84 of the best student?
85 */
86
87 db.students.aggregate(
88 [
89     {
90         $match: {'courses.course_status': { $nin: [ 'In Progress', 'Dropped' ] }}
91     },
92     {
93         $unwind: "$courses"
94     },

```

```

100     {
101         $group: {
102             _id: "$_id",
103
104             GPA: { $avg: '$courses.grade' }
105         }
106     },
107
108     {$sort: {GPA: -1}},
109
110     {$limit: 1}
111 ]
112 )
113
114
115
116 // Q5: Which student has the largest number of grade 10's?
117
118 db.students.aggregate(
119     [
120         {
121             $unwind: "$courses"
122         },
123         {
124             $group: {
125                 _id: "$_id",
126                 countMaxGrade: {
127                     $sum: {
128                         $cond: [{
129                             $eq: ['$courses.grade', 10]
130                         }, 1, 0]
131                     }
132                 }
133             }
134         },
135         {
136             $sort: {
137                 countMaxGrade: -1
138             }
139         },
140     ],
141     {
142         $limit: 1
143     }
144 ]
145 )
146
147
148
149
150 // Q6: Which class has the highest average GPA?
151
152 db.students.aggregate(
153     [
154         {
155             $unwind: "$courses"
156         },
157         {
158             $group: {

```

```

159         _id: "$courses.course_code",
160
161         "course_title": {
162             "$first": "$courses.course_title"
163         },
164
165         avgGrade: {
166             $avg: '$courses.grade'
167         }
168     }
169 },
170 {
171     $sort: {
172         avgGrade: -1
173     }
174 },
175 {
176     $limit: 1
177 }
178 ]
179 ).pretty()
180
181 // Q7: Which class has been dropped the most number of times?
182
183 db.students.aggregate(
184 [
185     {
186         $unwind: "$courses"
187     },
188     {
189         $group: {
190             _id: "$courses.course_code",
191             "course_name": {
192                 "$first": "$courses.course_title"
193             },
194             numberOfDropouts: {
195                 $sum: {
196                     $cond: [{
197                         $eq: ['$courses.course_status', 'Dropped']
198                     }, 1, 0]
199                 }
200             }
201         }
202     },
203     {
204         $sort: {
205             numberOfDropouts: -1
206         }
207     },
208     {
209         $limit: 1
210     }
211 ]
212 )
213 ]
214
215
216
217

```

```

218 ).pretty()
219
220
221 /* Q8: Produce of a count of classes that have been COMPLETED by class
222 type. The class type is found by taking the first letter of the course
223 code so that M102 has type M.
224 */
225
226 db.students.aggregate(
227   [
228     {
229       $unwind: "$courses"
230     },
231     {
232       $group:
233       {
234         _id: { $substr: [ "$courses.course_code", 0, 1 ] },
235         numberOfTimesCompleted: {
236           $sum: {
237             $cond: [ { $eq: [ '$courses.course_status', 'Complete' ] }, 1, 0 ]
238           }
239         }
240       }
241     },
242     { $sort: { numberOfTimesCompleted: -1 } }
243   ]
244 )
245
246
247
248 /* Q9: Produce a transformation of the documents so that the documents
249 now have an additional boolean field called "hobbyist" that is true
250 when the student has more than 3 hobbies and false otherwise.
251 */
252
253 db.students.aggregate(
254   [{
255     $project: {
256       home_city: 1,
257       first_name: 1,
258       hobbies: 1,
259       hobbyist: {
260         $cond: {
261           if: {
262             $gt: [{
263               $size: "$hobbies"
264             }, 3]
265           },
266           then: true,
267           else: false
268         }
269       },
270       favourite_os: 1,
271       laptop_cost: 1,
272       courses: 1
273     }
274   }]
275 )
276

```



```

277 /* Q10: Produce a transformation of the documents so that the documents
278 now have an additional field that contains the number of classes that
279 the student has completed.
280 */
281
282 db.students.aggregate(
283   [
284     {
285       $unwind: "$courses"
286     },
287     {
288       $group: {
289         _id: "$_id",
290         "home_city": {
291           "$first": "$home_city"
292         },
293         "first_name": {
294           "$first": "$first_name"
295         },
296         "hobbies": {
297           "$first": "$hobbies"
298         },
299         "hobbyist": {
300           "$first": "$hobbyist"
301         },
302         "favourite_os": {
303           "$first": "$favourite_os"
304         },
305         "laptop_cost": {
306           "$first": "$laptop_cost"
307         },
308         "courses": {
309           "$push": "$courses"
310         },
311         completed_courses: {
312           $sum: {
313             $cond: [{
314               $eq: ['$courses.course_status', 'Complete']
315             }, 1, 0]
316           }
317         }
318       }
319     ]
320   ).pretty()
321
322
323 /* Q11: Produce a transformation of the documents in the collection so that
324 they look like the following object.
325
326 The GPA is the average grade of all the completed classes. The other two
327 computed fields are the number of classes currently in progress and the
328 number of classes dropped. No other fields should be in there. No other
329 fields should be present.
330
331 {
332   "_id": "ObjectId('558d08925e083d8cdd7be831')",
333   "first_name": "Eirini",
334   "GPA": 8.5,
335   "classesInProgress": 3,

```

```

336   "droppedClasses": 0
337 }
338
339 */
340 db.students.aggregate(
341   [
342     {
343       $unwind: "$courses"
344     },
345     {
346       $group: {
347         _id: "$_id",
348         "first_name": {
349           "$first": "$first_name"
350         },
351         GPA: {
352           $avg: '$courses.grade'
353         },
354         classesInProgress: {
355           $sum: {
356             $cond: [{
357               $eq: ['$courses.course_status', 'In Progress']
358             }, 1, 0]
359           }
360         },
361         droppedClasses: {
362           $sum: {
363             $cond: [{
364               $eq: ['$courses.course_status', 'Dropped']
365             }, 1, 0]
366           }
367         }
368       }
369     ]
370   )
371   ].pretty()
372
373
374 /* Q12: Produce a NEW collection (HINT: Use $out in the aggregation pipeline)
375 so that the new documents in this correspond to the classes on offer. The
376 structure of the documents should be like the following object.
377
378 The _id field should be the course code. The course_title is what it was before.
379 The numberOfDropouts is the number of students who dropped out. The
380 numberOfTimesCompleted is the number of students that completed this class.
381 The currentlyRegistered array is an array of ObjectID's corresponding to the
382 students who are currently taking the class. Finally, for the students that
383 completed the class, the maxGrade, minGrade and avgGrade are the summary
384 statistics for that class.
385
386 {
387   "_id": "M102",
388   "course_title": "Data Mining",
389   "numberOfDropouts": 34,
390   "numberOfTimesCompleted": 34,
391   "currentlyRegistered": [
392     "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100"
393   ]
394 }

```

```

395 "currentlyRegistered": ["ObjectId('558d08925e083d8cdd7be831')", "..."],
396
397 "maxGrade": 10,
398
399 "minGrade": 2,
400
401 "avgGrade": 7.6
402 }
403
404 */
405
406 db.students.aggregate(
407 [
408   {
409     $unwind: "$courses"
410   },
411   {
412     $group: {
413       _id: "$courses.course_code",
414
415       course_title: {
416         "$first": "$courses.course_title"
417       },
418       numberOfDropouts: {
419         $sum: {
420           $cond: [{
421             $eq: ['$courses.course_status', 'Dropped']
422           }, 1, 0]
423         }
424       },
425       numberOfTimesCompleted: {
426         $sum: {
427           $cond: [{
428             $eq: ['$courses.course_status', 'Complete']
429           }, 1, 0]
430         }
431       },
432       currentlyRegistered: {
433         $push: {
434           $cond: [{
435             $eq: ['$courses.course_status', 'In Progress']
436           }, "$_id", null]
437         }
438       },
439       maxGrade: {
440         $max: '$courses.grade'
441       },
442       minGrade: {
443         $min: '$courses.grade'
444       },
445       avgGrade: {
446         $avg: '$courses.grade'
447       },
448     }
449   },
450   {
451     {
452       $addFields: {
453         "currentlyRegistered": {

```

```

454         "$setDifference": ["$currentlyRegistered", [null]]
455     }
456 }
457 },
458 {
459     $out: "classes"
460 }
461 ]
462 )

```

## 3 Part Two (Python & MongoDB)

### 3.1 part2\_python\_mongodb.py

```

1 # pylint: disable=invalid-name
2 """
3     python_mongodb.py: Implement simple operations on
4     mongo database.
5 """
6
7 import pprint
8 import pymongo
9 import pandas as pd
10 import numpy as np
11
12 __author__ = "Stratos Gounidellis, Lamprini Koutsokera"
13 __copyright__ = "Copyright 2017, BDSMasters"
14
15
16 def connect_to_mongo(db_name, collection_name):
17     """Connect to mongo database and collection.
18     :param db_name: The name of the mongo database.
19     :param collection_name: The name of the mongo collection.
20     :return: A connection to a collection and a MongoClient
21             object.
22     """
23     try:
24         client = pymongo.MongoClient()
25         db = client[db_name]
26         collection = db[collection_name]
27     except pymongo.errors.ConnectionFailure:
28         print "Unable to connect to mongo!"
29         quit()
30     return collection, client
31
32
33 def insert_one(db_name, collection_name, record):
34     """Connect to mongo database and collection and insert
35     a record.
36     :param db_name: The name of the mongo database.
37     :param collection_name: The name of the mongo collection.
38     :param record: The records to be inserted to the mongo
39     collection.
40     """
41     collection = connect_to_mongo(db_name, collection_name)
42     try:
43         collection[0].delete_many({})

```

```

44     except pymongo.errors.ServerSelectionTimeoutError:
45         print "Unable to connect to mongo!"
46         quit()
47     print '\nInserting Christiano to the collection.\n'
48     collection[0].insert_one(record)
49     collection[1].close()
50
51
52 def insert_many(db_name, collection_name, records_list):
53     """Connect to mongo database and collection and insert multiple
54     records.
55     :param db_name: The name of the mongo database.
56     :param collection_name: The name of the mongo collection.
57     :param records_list: The records to be inserted to the
58     mongo collection.
59     """
60     print 'Inserting Maria and Dimitris to the collection.\n'
61     collection = connect_to_mongo(db_name, collection_name)
62     collection[0].insert_many(records_list)
63     collection[1].close()
64
65
66 def print_records(db_name, collection_name):
67     """Connect to mongo database and collection and print its
68     content.
69     :param db_name: The name of the mongo database.
70     :param collection_name: The name of the mongo collection.
71     """
72     print "Printing collection's content.\n"
73     collection = connect_to_mongo(db_name, collection_name)
74     for record in collection[0].find():
75         pprint.pprint(record)
76     collection[1].close()
77
78
79 def update_collection(db_name, collection_name):
80     """Connect to mongo database and collection and update its
81     documents.
82     :param db_name: The name of the mongo database.
83     :param collection_name: The name of the mongo collection.
84     """
85     print "\nUpdating Christiano's age field."
86     collection = connect_to_mongo(db_name, collection_name)
87     collection[0].update_one({
88         'name': "Christiano"
89     }, {
90         '$set': {
91             'age': 26
92         }
93     }, upsert=True)
94
95     print "Updating Maria's name."
96     collection[0].update_one({
97         'name': "Maria"
98     }, {
99         '$set': {
100             'name': "Ioanna"
101         }
102     }, upsert=True)

```

```

103 print "Deleting Dimitris."
104 collection[0].delete_one({"name": "Dimitris"})
105 collection[1].close()
106
107
108 def print_records_field(db_name, collection_name, field):
109     """Connect to mongo database and collection and print
110         specific field.
111     :param db_name: The name of the mongo database.
112     :param collection_name: The name of the mongo collection.
113     :param field: The name of the field to be printed.
114     """
115     print "\nPrinting info about " + str(field) + ".\n"
116     collection = connect_to_mongo(db_name, collection_name)
117     check_exists = False
118     for record in collection[0].find():
119         if field in record.keys():
120             pprint.pprint(record[field])
121             check_exists = True
122     if not check_exists:
123         print "No records with field '" + str(field) + "' were found!"
124     collection[1].close()
125
126
127 def mongo_to_df(db_name, collection_name):
128     """Connect to mongo database and collection and convert the collection
129         to a dataframe.
130     :param db_name: The name of the mongo database.
131     :param collection_name: The name of the mongo collection.
132     :return: A dataframe containing the content of the collection.
133     """
134     print "\nConverting collection to dataframe.\n"
135     collection = connect_to_mongo(db_name, collection_name)
136     fields = []
137     for record in collection[0].find():
138         keys = record.keys()
139         for key in keys:
140             if key not in fields:
141                 fields.append(key)
142
143     results_array = np.zeros(len(fields))
144     for record in collection[0].find():
145         temp_list = []
146         for field in fields:
147             if field in record.keys():
148                 temp_list.append(record[field])
149             else:
150                 temp_list.append(None)
151         temp_results = np.array(temp_list)
152         results_array = np.vstack((temp_results, results_array))
153     results_array = results_array[:-1, :]
154     df_results = pd.DataFrame(data=results_array, columns=fields)
155     collection[1].close()
156     return df_results
157
158
159 def df_to_mongo(df, db_name, collection_name):
160     """Connect to mongo database and collection and import data
161         from a dataframe.

```

```

162 :param df: The dataframe to import to the mongo collection.
163 :param db_name: The name of the mongo database.
164 :param collection_name: The name of the mongo collection.
165 """
166 print "\nImporting dataframe to collection."
167
168 collection = connect_to_mongo(db_name, collection_name)
169 for _, row in df.iterrows():
170     row_dict = row.to_dict()
171     for key in row_dict.keys():
172         if row_dict.get(key) is None:
173             row_dict.pop(key, None)
174         else:
175             try:
176                 row_dict[key] = int(row_dict.get(key))
177             except ValueError:
178                 pass
179     collection[0].insert_one(row_dict)
180 collection[1].close()
181
182
183 if __name__ == "__main__":
184     db_name = "project"
185     collection_name = "pymongo_project"
186     christiano = {"language": "Portuguese", "name": "Christiano"}
187     insert_one(db_name, collection_name, christiano)
188
189     maria = {"name": "Maria", "age": 34, "language": "English"}
190     dimitris = {"name": "Dimitris", "language": "Greek"}
191     records_list = [maria, dimitris]
192     insert_many(db_name, collection_name, records_list)
193
194     print_records(db_name, collection_name)
195
196     update_collection(db_name, collection_name)
197
198     print_records_field(db_name, collection_name, "age")
199
200     df_mongo = mongo_to_df(db_name, collection_name)
201     print df_mongo
202
203     records_array = np.zeros(3)
204     giannis = ["Giannis", None, "German"]
205     nikos = ["Nikos", 23, "Polish"]
206     clio = ["Clio", 19, "Greek"]
207     eleni = ["Eleni", 29, None]
208     records = [giannis, nikos, clio, eleni]
209
210     for record in records:
211         records_array = np.vstack((record, records_array))
212     records_array = records_array[:-1, :]
213     df_records = pd.DataFrame(data=records_array,
214                               columns=("name", "age", "language"))
215     df_to_mongo(df_records, db_name, collection_name)
216
217     df_mongo = mongo_to_df(db_name, collection_name)
218     print df_mongo

```

## 4 Part Three (MapReduce)

### 4.1 part3\_word\_count.js

Write a map reduce job on the students collection similar to the classic word count example. More specifically, implement a word count using the course title field as the text. In addition, exclude stop words from this list. You should find/write your own list of stop words. (Stop words are the common words in the English language like "a", "in", "to", "the", etc.)

```
1  /**
2   * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
3   * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
4   */
5
6  var mapWordCount = function() {
7    // Declare a string with the stop words
8    var stopWords = "a, of, and, to, in, for, the";
9    // Iterate over the courses in each document
10   for (var idx = 0; idx < this.courses.length; idx++) {
11     var course_title = this.courses[idx].course_title;
12     // Covert to lowercase in order to avoid duplicates
13     course_title = course_title.toLowerCase().split(" ");
14     for (var i = course_title.length - 1; i >= 0; i--) {
15       var regex = new RegExp("\\b" + course_title[i] + "\\b", "i");
16       // Check whether the word is a stop word or not
17       if (stopWords.search(regex) < 0) {
18         if (course_title[i]) {
19           emit(course_title[i], 1);
20         }
21       }
22     }
23   }
24 };
25
26 var reduceWordCount = function(key, values) {
27   var count = 0;
28   // Sum the occureces of a word
29   values.forEach(function(value) {
30     count += value;
31   });
32   return count;
33 };
34
35 db.students.mapReduce(mapWordCount,
36   reduceWordCount, {
37     // Save the results at a collection
38     out: "count_courseTitle"
39   }
40 )
41
42 db.count_courseTitle.find().sort({"value": -1})
```

### 4.2 part3\_avg\_grade.js

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.).



```

1  /**
2   * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
3   * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
4   */
5
6  var mapAvgGrade = function() {
7      // Iterate over the courses in each document
8      for (var idx = 0; idx < this.courses.length; idx++) {
9          var course_status = this.courses[idx].course_status;
10         var course_grade = this.courses[idx].grade;
11         // Check that the course status is complete
12         if (course_status === "Complete") {
13             var course_title = this.courses[idx].course_code;
14             // Set as key the home city and the course type
15             var key = {
16                 home_city: this.home_city,
17                 course_type: course_title[0]
18             };
19             var value = {
20                 count: 1,
21                 sum: course_grade
22             };
23
24             emit(key, value);
25         }
26     }
27 };
28
29
30 var reduceAvgGrade = function(key, values) {
31     var reducedVal = {
32         count: 0,
33         sum: 0
34     };
35
36     values.forEach(function(value) {
37         reducedVal.count += value.count;
38         reducedVal.sum += value.sum;
39     });
40
41     return reducedVal;
42 };
43
44 var finalizeAvgGrade = function(key, reducedVal) {
45     // Calculate the average grade
46     reducedVal.avg = (reducedVal.sum / reducedVal.count).toFixed(4);
47
48     return reducedVal.avg;
49 };
50 };
51
52 db.students.mapReduce(mapAvgGrade,
53     reduceAvgGrade, {
54         // Save the results at a collection
55         out: {
56             merge: "avgGrade_city"
57         },
58         finalize: finalizeAvgGrade
59     }

```

```
60 )  
61  
62 db.avgGrade_city.find().sort({"value": -1})
```

## References

- [1] Docs.mongodb.com. *Install MongoDB Community Edition on Windows — MongoDB Manual 3.4.* <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/> [Accessed 2 May 2017].
- [2] Docs.mongodb.com. *Aggregation Pipeline — MongoDB Manual 3.4.* <https://docs.mongodb.com/manual/core/aggregation-pipeline/> [Accessed 2 May 2017].
- [3] Docs.mongodb.com. *Map-Reduce Examples — MongoDB Manual 3.4.* <https://docs.mongodb.com/manual/tutorial/map-reduce-examples/> [Accessed 2 May 2017].
- [4] Docs.mongodb.com. *Map-Reduce — MongoDB Manual 3.4.* <https://docs.mongodb.com/manual/core/map-reduce/> [Accessed 2 May 2017].