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

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:

```
{
 "_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"
   }
 ]
}
```

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.

```
/**
 * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
 * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
 */

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

/* Q1: 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")?
*/

db.students.find({'courses.course_status': 'In Progress'}).count()

/* Q2: Produce a grouping of the documents that contains the name of each
home city and the number of students enrolled from that home city.
*/

db.students.aggregate(
    [
        {
            "$group": {
                _id: "$home_city",
                enrolledStudents: {
                    $sum: 1
                }
            }
        }
    ]
)

// Q3: Which hobby or hobbies are the most popular?

db.students.aggregate(
    [
        {
            $unwind: "$hobbies"
        },

        {
```

```
            "$group": {
                _id: "$hobbies",
                popularity: {
                    $sum: 1
                }
            }
        },

        {
            $sort: {
                popularity: -1
            }
        },

        {
            $limit: 1
        }
    ]
)

db.students.aggregate(
    [
        {
            $unwind: "$hobbies"
        },

        {
            "$group": {
                _id: "$hobbies",
                popularity: {
                    $sum: 1
                }
            }
        },

        {
            $sort: {
                popularity: -1
            }
        },

        {
            $limit: 5
        }
    ]
)

/* Q4: What is the GPA (ignoring dropped classes and in progress classes)
of the best student?
*/

db.students.aggregate(
    [
        {
            $match: {'courses.course_status': { $nin: [ 'In Progress', 'Dropped' ] }}
        },
        {
            $unwind: "$courses"
        },
```

```
        {
            $group: {
               _id: "$_id",

               GPA: {  $avg: '$courses.grade' }
            }
        },

        {$sort: {GPA: -1}},

        {$limit: 1}

    ]
)


// Q5: Which student has the largest number of grade 10's?

db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
                _id: "$_id",
                countMaxGrade: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.grade', 10]
                        }, 1, 0]
                    }
                }
            }
        },

        {
            $sort: {
                countMaxGrade: -1
            }
        },

        {
            $limit: 1
        }

    ]
)


// Q6: Which class has the highest average GPA?

db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
```

```
                _id: "$courses.course_code",

                "course_title": {
                    "$first": "$courses.course_title"
                },

                avgGrade: {
                    $avg: '$courses.grade'
                }

            }
        },
        {
            $sort: {
                avgGrade: -1
            }
        },

        {
            $limit: 1
        }

    ]
).pretty()

// Q7: Which class has been dropped the most number of times?

db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
                _id: "$courses.course_code",
                "course_name": {
                    "$first": "$courses.course_title"
                },
                numberOfDropouts: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'Dropped']
                        }, 1, 0]
                    }
                }
            }
        },

        {
            $sort: {
                numberOfDropouts: -1
            }
        },

        {
            $limit: 1
        }

    ]
```

```
).pretty()


/* Q8: 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.
*/

db.students.aggregate(
   [
     {
         $unwind: "$courses"
     },
     {
         $group:
         {
           _id: { $substr: [ "$courses.course_code", 0, 1 ] },
           numberOfTimesCompleted: {
               $sum: {
                   $cond: [ { $eq: [ '$courses.course_status', 'Complete' ] }, 1, 0 ]
                }
            }
         }
     },

     {$sort: {numberOfTimesCompleted: -1}}

   ]
)

/* Q9: 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.
*/

db.students.aggregate(
    [{
        $project: {
            home_city: 1,
            first_name: 1,
            hobbies: 1,
            hobbyist: {
                $cond: {
                    if: {
                        $gt: [{
                            $size: "$hobbies"
                        }, 3]
                    },
                    then: true,
                    else: false
                }
            },
            favourite_os: 1,
            laptop_cost: 1,
            courses: 1
        }
    }]
)

```

```
/* Q10: 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.
*/

db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
                _id: "$_id",
                "home_city": {
                    "$first": "$home_city"
                },
                "first_name": {
                    "$first": "$first_name"
                },
                "hobbies": {
                    "$first": "$hobbies"
                },
                "hobbyist": {
                    "$first": "$hobbyist"
                },
                "favourite_os": {
                    "$first": "$favourite_os"
                },
                "laptop_cost": {
                    "$first": "$laptop_cost"
                },
                "courses": {
                    "$push": "$courses"
                },
                completed_courses: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'Complete']
                        }, 1, 0]
                    }
                }
            }
        }
    ]
).pretty()

/* Q11: Produce a transformation of the documents in the collection so that
they look like the following object.

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.

{
 "_id": "ObjectId('558d08925e083d8cdd7be831')",
 "first_name": "Eirini",
 "GPA": 8.5,
 "classesInProgress": 3,
```

```
 "droppedClasses": 0
}

*/
db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
                _id: "$_id",
                "first_name": {
                    "$first": "$first_name"
                },
                GPA: {
                    $avg: '$courses.grade'
                },
                classesInProgress: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'In Progress']
                        }, 1, 0]
                    }
                },
                droppedClasses: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'Dropped']
                        }, 1, 0]
                    }
                }
            }
        }

    ]
).pretty()

/* Q12: 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 the following object.

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.

{
 "_id": "M102",

 "course_title": "Data Mining",

 "numberOfDropouts": 34,

 "numberOfTimesCompleted": 34,

```

```
 "currentlyRegistered": ["ObjectId('558d08925e083d8cdd7be831')", "..."],

 "maxGrade": 10,

 "minGrade": 2,

 "avgGrade": 7.6
}

*/

db.students.aggregate(
    [
        {
            $unwind: "$courses"
        },
        {
            $group: {
                _id: "$courses.course_code",

                course_title: {
                    "$first": "$courses.course_title"
                },
                numberOfDropouts: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'Dropped']
                        }, 1, 0]
                    }
                },
                numberOfTimesCompleted: {
                    $sum: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'Complete']
                        }, 1, 0]
                    }
                },
                currentlyRegistered: {
                    $push: {
                        $cond: [{
                            $eq: ['$courses.course_status', 'In Progress']
                        }, "$_id", null]
                    }
                },
                maxGrade: {
                    $max: '$courses.grade'
                },
                minGrade: {
                    $min: '$courses.grade'
                },
                avgGrade: {
                    $avg: '$courses.grade'
                },

            }
        },
        {
            $addFields: {
                "currentlyRegistered": {
```

```
                    "$setDifference": ["$currentlyRegistered", [null]]
                }
            }
        },
        {
            $out: "classes"
        }
    ]
)
```

3 Part Two (Python & MongoDB)

3.1 part2_python_mongodb.py

```
# pylint: disable=invalid-name
"""
    python_mongodb.py: Implement simple operations on
        mongo database.
"""

import pprint
import pymongo
import pandas as pd
import numpy as np

__author__ = "Stratos Gounidellis, Lamprini Koutsokera"
__copyright__ = "Copyright 2017, BDSMasters"


def connect_to_mongo(db_name, collection_name):
    """Connect to mongo database and collection.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :return: A coonection to a collection and a MongoClient
        object.
    """
    try:
        client = pymongo.MongoClient()
        db = client[db_name]
        collection = db[collection_name]
    except pymongo.errors.ConnectionFailure:
        print "Unable to connect to mongo!"
        quit()
    return collection, client


def insert_one(db_name, collection_name, record):
    """Connect to mongo database and collection and insert
        a record.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param record: The records to be inserted to the mongo
        collection.
    """
    collection = connect_to_mongo(db_name, collection_name)
    try:
        collection[0].delete_many({})
```

```
    except pymongo.errors.ServerSelectionTimeoutError:
        print "Unable to connect to mongo!"
        quit()
    print '\nInserting Christiano to the collection.\n'
    collection[0].insert_one(record)
    collection[1].close()


def insert_many(db_name, collection_name, records_list):
    """Connect to mongo database and collection and insert multiple
        records.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param records_list: The records to be inserted to the
        mongo collection.
    """
    print 'Inserting Maria and Dimitris to the collection.\n'
    collection = connect_to_mongo(db_name, collection_name)
    collection[0].insert_many(records_list)
    collection[1].close()


def print_records(db_name, collection_name):
    """Connect to mongo database and collection and print its
        content.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "Printing collection's content.\n"
    collection = connect_to_mongo(db_name, collection_name)
    for record in collection[0].find():
        pprint.pprint(record)
    collection[1].close()


def update_collection(db_name, collection_name):
    """Connect to mongo database and collection and update its
        documents.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "\nUpdating Christiano's age field."
    collection = connect_to_mongo(db_name, collection_name)
    collection[0].update_one({
        'name': "Christiano"
    }, {
        '$set': {
            'age': 26
        }
    }, upsert=True)

    print "Updating Maria's name."
    collection[0].update_one({
        'name': "Maria"
    }, {
        '$set': {
            'name': "Ioanna"
        }
    }, upsert=True)
```

```
    print "Deleting Dimitris."
    collection[0].delete_one({"name": "Dimitris"})
    collection[1].close()


def print_records_field(db_name, collection_name, field):
    """Connect to mongo database and collection and print
        specific field.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param field: The name of the field to be printed.
    """
    print "\nPrinting info about " + str(field) + ".\n"
    collection = connect_to_mongo(db_name, collection_name)
    check_exists = False
    for record in collection[0].find():
        if field in record.keys():
            pprint.pprint(record[field])
            check_exists = True
    if not check_exists:
        print "No records with field '" + str(field) + "' were found!"
    collection[1].close()


def mongo_to_df(db_name, collection_name):
    """Connect to mongo database and collection and convert the collection
        to a dataframe.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :return: A dataframe containing the content of the collection.
    """
    print "\nConverting collection to dataframe.\n"
    collection = connect_to_mongo(db_name, collection_name)
    fields = []
    for record in collection[0].find():
        keys = record.keys()
        for key in keys:
            if key not in fields:
                fields.append(key)

    results_array = np.zeros(len(fields))
    for record in collection[0].find():
        temp_list = []
        for field in fields:
            if field in record.keys():
                temp_list.append(record[field])
            else:
                temp_list.append(None)
        temp_results = np.array(temp_list)
        results_array = np.vstack((temp_results, results_array))
    results_array = results_array[:-1, :]
    df_results = pd.DataFrame(data=results_array, columns=fields)
    collection[1].close()
    return df_results


def df_to_mongo(df, db_name, collection_name):
    """Connect to mongo database and collection and import data
        from a dataframe.
```

```
    :param df: The dataframe to import to the mongo collection.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "\nImporting dataframe to collection."

    collection = connect_to_mongo(db_name, collection_name)
    for _, row in df.iterrows():
        row_dict = row.to_dict()
        for key in row_dict.keys():
            if row_dict.get(key) is None:
                row_dict.pop(key, None)
            else:
                try:
                    row_dict[key] = int(row_dict.get(key))
                except ValueError:
                    pass
        collection[0].insert_one(row_dict)
    collection[1].close()


if __name__ == "__main__":
    db_name = "project"
    collection_name = "pymongo_project"
    christiano = {"language": "Portuguese", "name": "Christiano"}
    insert_one(db_name, collection_name, christiano)

    maria = {"name": "Maria", "age": 34, "language": "English"}
    dimitris = {"name": "Dimitris", "language": "Greek"}
    records_list = [maria, dimitris]
    insert_many(db_name, collection_name, records_list)

    print_records(db_name, collection_name)

    update_collection(db_name, collection_name)

    print_records_field(db_name, collection_name, "age")

    df_mongo = mongo_to_df(db_name, collection_name)
    print df_mongo

    records_array = np.zeros(3)
    giannis = ["Giannis", None, "German"]
    nikos = ["Nikos", 23, "Polish"]
    clio = ["Clio", 19, "Greek"]
    eleni = ["Eleni", 29, None]
    records = [giannis, nikos, clio, eleni]

    for record in records:
        records_array = np.vstack((record, records_array))
    records_array = records_array[:-1, :]
    df_records = pd.DataFrame(data=records_array,
                              columns=("name", "age", "language"))
    df_to_mongo(df_records, db_name, collection_name)

    df_mongo = mongo_to_df(db_name, collection_name)
    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.)

```
/**
 * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
 * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
 */

var mapWordCount = function() {
    // Declare a string with the stop words
    var stopWords = "a, of, and, to, in, for, the";
    // Iterate over the courses in each document
    for (var idx = 0; idx < this.courses.length; idx++) {
        var course_title = this.courses[idx].course_title;
        // Covert to lowercase in order to avoid duplicates
        course_title = course_title.toLowerCase().split(" ");
        for (var i = course_title.length - 1; i >= 0; i--) {
            var regex = new RegExp("\\b" + course_title[i] + "\\b", "i");
            // Check whether the word is a stop word or not
            if (stopWords.search(regex) < 0) {
                if (course_title[i]) {
                    emit(course_title[i], 1);
                }
            }
        }
    }
};

var reduceWordCount = function(key, values) {
    var count = 0;
    // Sum the occureces of a word
    values.forEach(function(value) {
        count += value;
    });
    return count;
};

db.students.mapReduce(mapWordCount,
    reduceWordCount, {
        // Save the results at a collection
        out: "count_courseTitle"
    }
)

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.).

```
/**
 * @author Stratos Gounidellis <stratos.gounidellis@gmail.com>
 * @author Lamprini Koutsokera <lkoutsokera@gmail.com>
 */

var mapAvgGrade = function() {
    // Iterate over the courses in each document
    for (var idx = 0; idx < this.courses.length; idx++) {
        var course_status = this.courses[idx].course_status;
        var course_grade = this.courses[idx].grade;
        // Check that the course status is complete
        if (course_status === "Complete") {
            var course_title = this.courses[idx].course_code;
            // Set as key the home city and the course type
            var key = {
                home_city: this.home_city,
                course_type: course_title[0]
            };
            var value = {
                count: 1,
                sum: course_grade
            };

            emit(key, value);
        }

    }
};

var reduceAvgGrade = function(key, values) {
    var reducedVal = {
        count: 0,
        sum: 0
    };

    values.forEach(function(value) {
        reducedVal.count += value.count;
        reducedVal.sum += value.sum;
    });

    return reducedVal;
};

var finalizeAvgGrade = function(key, reducedVal) {
    // Calculate the average grade
    reducedVal.avg = (reducedVal.sum / reducedVal.count).toFixed(4);

    return reducedVal.avg;

};

db.students.mapReduce(mapAvgGrade,
    reduceAvgGrade, {
        // Save the results at a collection
        out: {
            merge: "avgGrade_city"
        },
        finalize: finalizeAvgGrade
    }
```

```
)

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*. ttps://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*].