School of Computer Science

COMP47470

Lab 4
NoSQL Systems (MongoDB)

Teaching Assistant:	Leandro Batista de Almeida
Coordinator:	Dr Anthony Ventresque
Date:	Thursday 14th February, 2019
Total Number of Pages:	4

General Requirements/Resources

For this lab session you can either use the account created for you on `csserver.ucd.ie` (recommended) or you can install a MongoDB server on your own laptop.

1 Command-line client (mongo shell)

The mongo command-line tool is a simple command shell, that functions like a JavaScript interface to MongoDB, and is a component of the MongoDB distribution. You can use mongo shell to query and update data as well as perform administrative operations, and also to run JavaScript batchs of commands. The command shell has also an auto-completing feature, using the Tab key. All tasks of this section are performed in a command shell. You must install the example database Restaurants from the MongoDB documentation website

- Get help for the mongo command-line tool:

  ```
  $> mongo -help
  ```

- Connecting to the local server (localhost):

  ```
  $> mongo
  ```

2 Working with mongo shell

In the mongo command-line tool, you can display the databases available for your user, the collections in it, and the data inside the documents. After the command, you can type ';', but it is not mandatory, unless you want to send more than one command in just one line.

- Display the help page and the help page for the databases:

  ```
  >help
  >db.help()
  ```

- Show the databases available for your user:

  ```
  >show databases
  ```

- Select the test database:

  ```
  >use test
  ```

- Show the collections in the database in use, like showed in Figure 7:

  ```
  >show collections
  ```

- Display the documents from a collection, using the alias db for the current database:

  ```
  >db.restaurants.find()
  ```

3 Basic DML (Data Manipulation Language), insert, update, delete

- Learn about the commands used in collections:

```
>db.mycoll.help()
```

- Show one document (the first, usually), all the documents, and all the documents in a pretty (hierarchical view) way:

```
>db.restaurants.findOne()
>db.restaurants.find()
>db.restaurants.find().pretty()
```

- Create a new database using your Student ID (cs99999999):

```
>use cs9999999
```

- In this new database, insert new documents in a new collection, like showed in Figure 8:

```
>db.student.insert({ name: J a m e s Nicholas G r a y , dept_name: C o m p u t e r S c i e n c e ,
tot_cred:10, age:63 })
>db.student.insert({ name: A l a n Mathison T u r i n g , dept_name: M a t h e m a t i c s ,
tot_cred:10 , age:41 })
>db.student.insert({ name: C l a u d e Elwood S h a n n o n , dept_name: E l e c t r i c a l
E n g i n e e r i n g , tot_cred:10, age:84 })
>db.student.insert({ name: G r a c e Brewster Murray H o p p e r , dept_name: C o m p u t e r
S c i e n c e , tot_cred:10, age:85 })
```

- Display the students of the Computer Science department:

```
>db.student.find({dept_name:"Computer Science"})
```

- Display the name of students under the age of 50:

```
>db.student.find({age:{$lt:50}}, {name:1})
```

- Update the department name of student 3 (Alan Turing) to Computer Science:

```
>db.student.update({name:"Alan Mathison Turing"}, {$set: {dept_name:"Computers
Science"}})
>db.student.find({name:"Alan Mathison Turing"})
```

- Delete the students of Electrical Engineering department:

```
>db.student.remove( {dept_name:"Electrical Engineering"} )
>db.student.find()
```

4 Basic DML (Data Manipulation Language), retrieval of documents

The find() function is not so powerful as the sql select command, but has several options. If find() is not enough, there is an aggregation framework and mapreduce functions to extend it.

- Retrieve only the fields name and age from all documents in the collection students:

  ```
  >db.student.find({},{name:1, age:1})
  ```

- Order the documents by a field or a set of fields, in forward or reverse order:

  ```
  >db.student.find().sort({name:1})
  >db.student.find().sort({dept_name:1, name:1})
  >db.student.find().sort({age:-1})
  ```

- Order the documents by a field inside another field (member of):

  ```
  >db.restaurants.find().sort({"borough": 1, "address.zipcode": 1})
  ```

- Filter the documents using a condition based on values of the fields:

  ```
  >db.student.find({age:{$gte:60}}, {name:1, age:1})
  >db.student.find({age:{$gte:60,$lte:70}})
  ```

- Filter documents using parts of strings, with a penalty on performance:

  ```
  >db.student.find({name: /^Alan./})
  >db.student.find({name: /Nicholas/})
  ```

- Count how many documents are in a query:

  ```
  >db.student.find().count()
  ```

Optional: Read about the Agregation Framework in MongoDB Manual and solve the following queries:

- Retrieve the average age from students:
- Retrieve the sum of credits of all students, and from students of the Computer Science department
- Retrieve the sum of the students credits, grouping by department

5 Bash Scripts for MongoDB

In the same way as MySQL, MongoDB allows you to send commands to the client from a bash script, submitting one line at a time, using the option -eval.

Some examples are:

```
mongo -eval "db.hostinfo()"
mongo world_database -eval "printjson(db.getCollectionNames())"
```

You can find more information at:

- MongoDB Command Line Client Manual
- How to execute MongoDB command from the Linux shell - Blog

6 Exercise

Get the titanic database from this url: http://jse.amstat.org/jse data archive.htm. (this is the dataset we used last week). Create and populate a database (1 collection is enough - e.g, use a Bash script that would read the csv file and create Mongo queries to insert values in the database). The name of the database must contain your Student ID, to avoid confusion with other students (i.e., titanic¡csstudentnumber¿).

Now anaswer the follwing questions:

- How many passengers were there on the Titanic? (use a Mongo query)
- How many passengers survived? (use a Mongo query)
- What percentage of passengers survived? (use a (nested) Mongo query)