Document for MongoDB Coursework

Part 1

Pre-defined Queries and Results

Shanchuan Wu

Sw9n14@soton.ac.uk

Query 1

Code and Explanation

Code

```
33      'mapReduce': function () {
34          db.getCollection(this.collectionName).mapReduce(
35              function () {
36                  emit(this.id_member, {
37                      'tweetsCnt': 1,
38                      'tweetsLen': this.text.length
39                  });
40              }, //map
41              function (key, values) {
42                  var total = 0;
43                  var tweetsLen = 0;
44                  for (var i = 0; i < values.length; i++) {
45                      total += values[i].tweetsCnt;
46                      tweetsLen += values[i].tweetsLen;
47
48                  }
49                  return {
50                      'tweetsCnt': total,
51                      'tweetsLen': tweetsLen
52                  }
53              }, //reduce s
54              {
55                  out: 'mrout'
56
57              }
58          )
59      },
```

```
115         if ((num == db.getCollection(this.collectionName).count()) && (count != 0)) {
116             this.numOfTweets = num;
117
118             //Q1
119             this.uniqueUserCnt = db.getCollection(this.mrcollectionName).count();
120             return true;
121         } else {
122             this.numOfTweets = 0;
123             this.uniqueUserCnt = 0;
124             return false;
125         }
```

```
269          print("-------------------------------------------------------");
270          print("Q1:uniqueUserCnt:" + this.uniqueUserCnt);
271          print("-------------------------------------------------------");
```

Explanation

When the data are cleaned, function *mapReduce* is defined to group the data according to *id_member* and a new collection named *mrout* is created. The structure of *mrout* is shown as below. As we can see, each document in *mrout* contains every user's total number of tweets and total length of tweets. By using the query above, the value of *uniqueUserCnt* can be returned, which stands for the quantity of unique users.

```
/* 1 */
{
    "_id" : 855.0000000000000000,
    "value" : {
        "tweetsCnt" : 3.0000000000000000,
        "tweetsLen" : 323.0000000000000000
    }
}

/* 2 */
{
    "_id" : 2644.0000000000000000,
    "value" : {
        "tweetsCnt" : 5.0000000000000000,
        "tweetsLen" : 437.0000000000000000
    }
}
```

Result

```
------------------------------------------------------------
Q1:uniqueUserCnt:117858
------------------------------------------------------------
```

Query 2

Code and Explanation

Code

```
176             //Q2
177             //print("Following is the amount of the tweetsCnt of the top10 users");
178             var tmp = [];
179             db.getCollection(this.mrcollectionName).find().sort({
180                 "value.tweetsCnt": -1
181             }).limit(10).forEach(function (it) {
182                 var rec = {
183                     'id': it._id,
184                     'tweetsCnt': it.value.tweetsCnt
185                 };
186                 tmp.push(rec);
187 
188             });
189             this.tweetsCnt_top10 = tmp;
```

```
271         print("------------------------------------------------------------");
272         print("Q2:top10 users");
273         for (var i = 0; i < this.tweetsCnt_top10.length; i++) {
274             var rec = this.tweetsCnt_top10[i];
275             print("\ttop" + (i + 1) + ":id:" + rec.id + "\t\t\ttweetsCnt:" + rec.tweetsCnt);
276         };
277         print("------------------------------------------------------------");
```

Explanation

Firstly, get a list of top 10 users in a descending order from collection *mrout* and allocate each value of these users to the array *tweetsCnt_top10*, then we use a loop to print the array.

Result

```
-----------------------------------------------------
Q2:top10 users
        top1:id:1484740038                      tweetsCnt:9594
        top2:id:497145453                       tweetsCnt:4792
        top3:id:1266803563                      tweetsCnt:4667
        top4:id:37402072                        tweetsCnt:2715
        top5:id:1544159024                      tweetsCnt:2321
        top6:id:418909674                       tweetsCnt:2010
        top7:id:229045023                       tweetsCnt:1810
        top8:id:229940852                       tweetsCnt:1640
        top9:id:29035604                        tweetsCnt:1437
        top10:id:14525315                       tweetsCnt:1339
-----------------------------------------------------
```

Query 3

Code and Explanation

Code

```
191             //Q3
192             this.timeStamp_e = db.getCollection(this.collectionName).find().sort({
193                 "timestamp": 1
194             }).limit(1)[0].timestamp;
195             this.timeStamp_l = db.getCollection(this.collectionName).find().sort({
196                 "timestamp": -1
197             }).limit(1)[0].timestamp;
```

```
277         print("-----------------------------------------------------");
278         print("Q3:timeStamp_e:" + this.timeStamp_e);
279         print("Q3:timeStamp_l:" + this.timeStamp_l);
280         print("-----------------------------------------------------");
```

Explanation

In order to get the earliest timestamp, it is necessary to sort all timestamps in an ascending order, then the first timestamp is the earliest timestamp. Likewise, get the latest timestamp by sorting time in a descending order.

Result

```
-----------------------------------------------------
Q3:timeStamp_e:2014-06-22 23:00:00
Q3:timeStamp_l:2014-06-30 21:59:59
-----------------------------------------------------
```

Query 4

Code and Explanation

Code

```
199             //Q4
200             var timeStampCount = 0;
201             var earlisetTime = this.timeStamp_e
202             db.getCollection(this.collectionName).distinct('timestamp').forEach(function (it) {
203                 timeDelta += ISODate(it).getTime() - ISODate(earlisetTime).getTime();
204                 timeStampCount++;
205             });
206             this.meanTimeDelta = timeDelta / timeStampCount;
207
```

```
280         print("-----------------------------------------------------");
281         print("Q4:meanTimeDelta:" + this.meanTimeDelta + "ms");
282         print("-----------------------------------------------------");
```

Explanation

Function *getTime()* can return the number of milliseconds since 1970/01/01, so it can be used to calculate the time delta between the earliest timestamp and 1970/01/01. Likewise, it is easy to know the time deltas between every timestamp and 1970/01/01. *Distinct()* is used to return different timestamps and *ISODate()* is used to convert the format of time into ISO date format. Finally, the mean time delta equals *timeDelta* / *timeStampCount*.

Result

```
-----------------------------------------------------
Q4:meanTimeDelta:349664293.02711195ms
-----------------------------------------------------
```

Query 5

Code and Explanation

Code

```
208             //Q5
209             db.getCollection(this.mrcollectionName).find().forEach(function (it) {
210                 averLen += it.value.tweetsLen / numOfTweets;
211             });
212             this.averLen = averLen;
```

```
282         print("-----------------------------------------------------");
283         print("Q5:averLen:" + this.averLen);
284         print("-----------------------------------------------------");
```

Explanation

Because each document in the collection *mrout* has two values, *tweetsCnt* and *tweetsLen*, it is easy to get the total number and string length of all messages. Then, the average length of messages can be obtained by calculating *averLen* += *it.value.tweetsLen* / *numOfTweets*.

Result

```
-----------------------------------------------------
Q5:averLen:71.27809248085795
-----------------------------------------------------
```

Query 6

Code and Explanation

Code

```
    'mapReduce2': function () {
        db.getCollection(this.collectionName).mapReduce(
            function () {
                var text = this.text;
                if (text) {
                    // quick lowercase to normalize per your requirements
                    text = text.toString().toLowerCase().split(" ");
                    for (var i = text.length - 1; i >= 0; i--) {
                        // might want to remove punctuation, etc. here
                        if (text[i]) { // make sure there's something
                            emit(text[i], {
                                'type': 0,
                                'count': 1
                            }); // store a 1 for each word
                        }
                    };
                    for (var i = 0; i < text.length - 1; i++) {
                        if (text[i]) {
                            emit(text[i] + ' ' + text[i + 1], {
                                'type': 1,
                                'count': 1
                            })
                        }
                    };
                }
            }, //map
            function (key, values) {
                var cnt = 0;
                for (var i = 0; i < values.length - 1; i++) {
                    cnt++;
                }
                return {
                    'type': values[0].type,
                    'count': cnt
                } //reduce
            }, {
                out: 'textout'

            }
        )
    },
```

```
            //Q6
            var txtTmp = [];
            db.getCollection(this.txtcollectionName).find({
                "value.type": 0
            }).sort({
                "value.count": -1
            }).limit(10).forEach(function (it) {
                var unigram = {
                    'id': it._id,
                    'count': it.value.count
                };
                txtTmp.push(unigram);

            });
            this.unigramCnt = txtTmp;

            var txtTmp2 = [];
            db.getCollection(this.txtcollectionName).find({
                "value.type": 1
            }).sort({
                "value.count": -1
            }).limit(10).forEach(function (it) {
                var bigram = {
                    'id': it._id,
                    'count': it.value.count
                };
                txtTmp2.push(bigram);

            });
            this.bigramCnt = txtTmp2;

```

```
        print("-----------------------------------------------------");
        print("Q6:top10 unigram strings");
        for (var i = 0; i < this.unigramCnt.length; i++) {
            var unigram = this.unigramCnt[i];
            print("\ttop" + (i + 1) + ":id:" + unigram.id + "\t\t\ttxtCnt:" + unigram.count);
        };
        print("Q6:top10 bigram strings");
        for (var i = 0; i < this.bigramCnt.length; i++) {
            var bigram = this.bigramCnt[i];
            print("\ttop" + (i + 1) + ":id:" + bigram.id + "\t\t\ttxtCnt:" + bigram.count);
        };
        print("-----------------------------------------------------");
```

Explanation

First of all, function *mapReduce2* is defined to group the messages according to different unigrams or bigrams and create a new collection *textout*. In *mapReduce2*, every text is converted into low case and divided into different unigrams and bigrams. Type 0 stands for unigrams and type 1 stands for bigrams. The structure of collection *textout* is shown as below. Finally, the 10 most common unigrams and bigrams are printed by using a loop.

```
/* 1 */
{
    "_id" : "!the very",
    "value" : {
        "type" : 1.0000000000000000,
        "count" : 1.0000000000000000
    }
}

/* 2 */
{
    "_id" : "!they",
    "value" : {
        "type" : 0.0000000000000000,
        "count" : 1.0000000000000000
    }
}
```

Result

```
----------------------------------------------------
Q6:top10 unigram strings
        top1:id:stuff                   txtCnt:1853
        top2:id:round                   txtCnt:1840
        top3:id:enjoy                   txtCnt:1838
        top4:id:worth                   txtCnt:1828
        top5:id:leave                   txtCnt:1820
        top6:id:mind                    txtCnt:1817
        top7:id:used                    txtCnt:1813
        top8:id:once                    txtCnt:1812
        top9:id:you!                    txtCnt:1810
        top10:id:tv                     txtCnt:1807
Q6:top10 bigram strings
        top1:id:cant believe                    txtCnt:1877
        top2:id:that is                 txtCnt:1850
        top3:id:my life                 txtCnt:1831
        top4:id:i really                        txtCnt:1824
        top5:id:at least                        txtCnt:1823
        top6:id:need a                  txtCnt:1821
        top7:id:the only                        txtCnt:1818
        top8:id:to watch                        txtCnt:1818
        top9:id:to make                 txtCnt:1817
        top10:id:makes me                       txtCnt:1816
----------------------------------------------------
```

Query 7

Code and Explanation

Code

```
244                 //Q7
245                 db.getCollection(this.collectionName).find({
246                     "text": /#/
247                 }).forEach(function (it) {
248                     hashTagCnt += it.text.match(/#/g).length;
249                 });
250                 this.averHashTagCnt = hashTagCnt / numOfTweets;
```

```
295         print("-----------------------------------------------------");
296         print("Q7:averHashTagCnt:" + this.averHashTagCnt);
297         print("-----------------------------------------------------");
```

Explanation

First of all, it is necessary to find messages that contain hashtags. So, *match(/#/g).length* is used to help find all hashtags in a message and return the number of hashtags in this message. Next step is to add up all numbers of hashtags in every message and get the total number of hashtags. Finally, the average number of hashtags can be obtained by calculating *hashTagCnt / numOfTweets*.

Result