THE UNIVERSITY OF SYDNEY

School of Information Technologies
Dr. Ying Zhou

COMP5347: Web Application Development **1.Sem./2017**

Week 9 Part One: MongoDB Basic Queries

09.05.2017

Learning Objectives

Practice basic MongoDB features with given JSON document:

- import data into mongodb from JSON file
- basic read query
- basic aggregation query
- indexing
- get familiar with MongoDB query reference document

Question 1: Start MongoDB server on lab machine

MongoDB is installed on all Lab PCs under the directory `C:/Program Files/MongoDB/`. It requires a data directory to store all data. It is recommended that you create this data directory on `U:` drive. This way you can access your data from any lab machine. `mongod` is the primary demon process of the MongoDB database engine. It can be started by a regular user with the command `mongod.exe`.

Below are the basic steps for starting MongoDB database engine the first time:

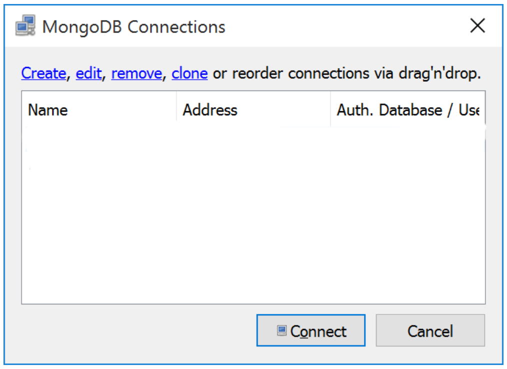
- Create a directory `comp5347/mongodb` on your U drive.
- Open a command window or power shell window and change to directory `C:/Program Files/MongoDB/Server/3.4/bin`
- run the following command

```
mongod.exe --dbpath U:/comp5347/mongodb --smallfiles
```

If the database engine starts correctly, you will see a few lines of initialization message and the last line should be “waiting for connections on port 27017”. Leave this window open unless you are ready to shut down the server.

Question 2: Start Robomongo as client shell GUI

The JavaScript interactive shell can be invoked directly from another command window. There are a lot of third party client side tools making it much easier to type in command and to view the results. All lab machines have `Robomongo` installed. You can find `Robomongo` from the query window and start it. Once started, you will see a prompt window similar to figure 1:

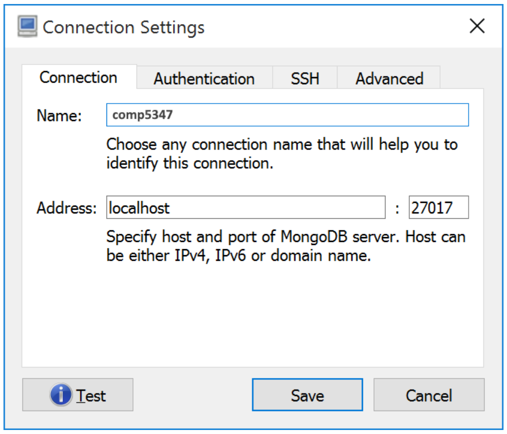

MongoDB Connections

Create, edit, remove, clone or reorder connections via drag'n'drop.

Name	Address	Auth. Database / Use

Figure 1: Robomongo start screen

Click `Create` to create a new connection and the default address would be `localhost:27017`. You can keep it and give a new name such as `comp5347`, see figure 2. Save the connection and select it to make the connection.


Connection Settings

Name: comp5347

Choose any connection name that will help you to identify this connection.

Address: localhost : 27017

Specify host and port of MongoDB server. Host can be either IPv4, IPv6 or domain name.

Figure 2: Robomongo create connection

Once connected, you will see the main screen of Robomongo. It consists of a top menu bar, left and right panels. The left panel shows all databases and tables in the current system. The right panel is used for query command and result display.

Any MongoDB server contains a `System` database. This database is used by the database engine to store system wide data such as users and global functions. You are not recommended to directly change the content of this database.

You should create and work on your own database. To create a database, right click the connection name and select `create database`. You can call the new database `wikipedia`. see figure 3.

After the database is created, expend it and you will see three items: `Collections`, `Functions` and `Users`. Right click `Collections` to create a collections: `revisions`. We will use this collection to store the wiki revision and user data.

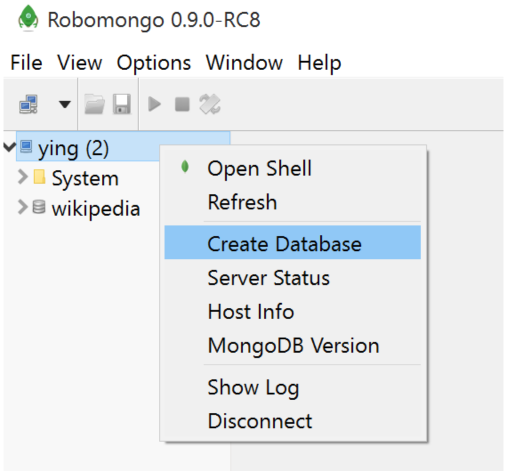


Figure 3: Robomongo create database

Question 3: Import json file to MongoDB collection

Download two json files from ELearning or Piazza: `BBC.json` and `CNN.json`. You can inspect the content of the files using Notepad++. They contain revision data of the two wikipedia articles: BBC and CNN respectively. Data about each revision is stored as an object with a number of properties: `title`, `timestamp`, `user` and so on. The collection of revisions is stored as a large array.

MongoDB provides a simple tool `mongoimport` to import JSON, CSV or TSV files to mongo collection. We will use it to import the revision data in the newly created collection `revisions`.

Open another command window or power shell window and change to directory `C:/Program Files/MongoDB/Server/3.2/bin`.

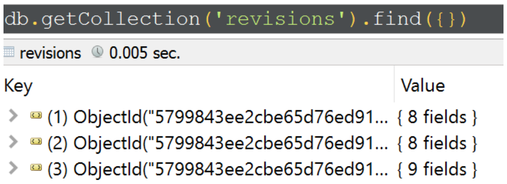
Run the following command to import a file to `revisions` collection.

```
mongoimport --jsonArray --db wikipedia --collection
revisions --file <full-path-to-your-downloaded-revision-file>
```

If the command executes successfully, you will see information such as how many documents are imported. Each object in the json file will be imported as a document in the specified collection.

Question 4: Simple MongoDB read queries

Go back to the robomongo main screen and double click the collection `revisions` on the left panel. This will invoke a select-all query on the collection. Both the query command and the results will be displayed on the right panel in a tab, see figure 4. You may notice that an `_id` field is automatically created for each document with type `ObjectId`. This is the unique identifier of each document. The `import` command is able to infer simple data type, such as string and integer, for each field. You will notice that all fields in the imported documents are either assigned a `String` or `Int32` type. This includes the `timestamp` field.



```
db.getCollection('revisions').find({})
revisions  0.005 sec.
Key                                        Value
> (1) ObjectId("5799843ee2cbe65d76ed91...  { 8 fields }
> (2) ObjectId("5799843ee2cbe65d76ed91...  { 8 fields }
> (3) ObjectId("5799843ee2cbe65d76ed91...  { 9 fields }
```

Figure 4: Robomongo query and result

You can have many tabs on the right panel, each with a text box at the top for queries and a view pane to display the corresponding query results at the bottom. You can slightly modify the automatically generated query to see how many documents are in each collection. The following command find out the number of documents in `revisions` collection.

```
db.revisions.find({}).count()
```

Here are a few other queries you can try:

1. Find distinct users in the `revisions` collection:
   `db.getCollection('revisions').distinct('user')`
2. Find distinct user from article 'BBC':
   `db.getCollection('revisions').distinct('user', {title:'CNN'})`
3. Find the first revision of article CNN
   `db.getCollection('revisions').find({title:'CNN'}).sort({timestamp:1}).limit(1)`

This query starts by retrieving all documents with title 'CNN' in the `revisions` collection (`db.getCollection('revisions').find({title:'CNN'})`), then sort them by the field 'timestamp' in ascending order (`.sort({timestamp:1})`) and output only the first document in the results (`.limit(1)`). The 'timestamp' field currently is of type `String` with format 'YYYY-MM-DDTHH:MM:SS', sorting by string results would be similar to sorting by Date result.

Question 5: Aggregation

Aggregation framework allows us to run more complex queries on the collection by grouping documents based on certain criteria and summarizing the results in different ways.

Run the following aggregat to find out which five editors made the most revisions on CNN page:

```
db.getCollection("revisions").aggregate([
{$match:{title:"CNN"}},
{$group:{_id:"$user", numOfEdits: {$sum:1}}},
{$sort:{numOfEdits:-1}},
{$limit:5}
])
```

The aggregation has four stages: the first stage ($match) finds all revision documents belong to article `‘CNN’`; the second stage groups ($group) the documents based on the “user” field value, and create a new field called “numOfEdits”, the value of which is set to the number of document with that particular “user” value; the third stage ($sort) sorts the resulting documents by the filed “numOfEdits” in descending order; the last stage ($limit) limits the output to the first five documents.

Question 6: JavaScript shell scripting

The mongodb shell is written in JavaScript and simple JavaScript statements/functions can be used to query or manipulate data. In particular, a read query `db.collection.find()` always returns a `cursor` object to the results. The interactive JavaScript shell by default iterates through the cursor for up to 50 items by calling the `next()` method and prints the matching documents out. In addition to the `next()` method, a lot of other methods can be used to traverse or manipulate the result set. The reference document for all `cursor` methods can be found from `https://docs.mongodb.com/manual/reference/method/#cursor`

In the following example, we use simple script to update the data type of existing documents in `revisions`.

```
db.revisions.find().forEach(function(doc){
  doc.timestamp = new ISODate(doc.timestamp);
  db.revisions.save(doc)
});
```

We start by a read query `db.revisions.find()` to find all documents in the `revisions` collection. This returns a cursor object. We use the `forEach()` method of the cursor to apply a JavaScript function to each document returned (`.forEach(function(doc){...})`).

The function is an anonymous function defined right at the spot with two statements: the first statement `doc.timestamp = new ISODate(doc.timestamp);` converts the `timestamp` field to type `ISODate`; the second statement `db.revisions.save(doc)` saves the updated document back in the collection. Because both documents have the same ID, the new one will override the old one. (https://docs.mongodb.com/manual/reference/method/db.collection.save/)

JavaScript statements also allows you run a queries with subsequent ones built on results from previous ones. The following example uses two queries to find out set of registered users who have made revisions on both articles. The first query finds out distinct regular users in article "CNN" and save the results as in variable `users`. The second query uses the variable to finds out the users also appear in article "BBC".

```
users = db.revisions.distinct("user",{title:"CNN", anon:{$exists:false}})
common_users = db.revisions.distinct("user",{title:"BBC", user:{$in:users}})
```

It is possible to save script as a file and use the shell command `mongo` to run it (https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/).

Question 7: Setting up indexes

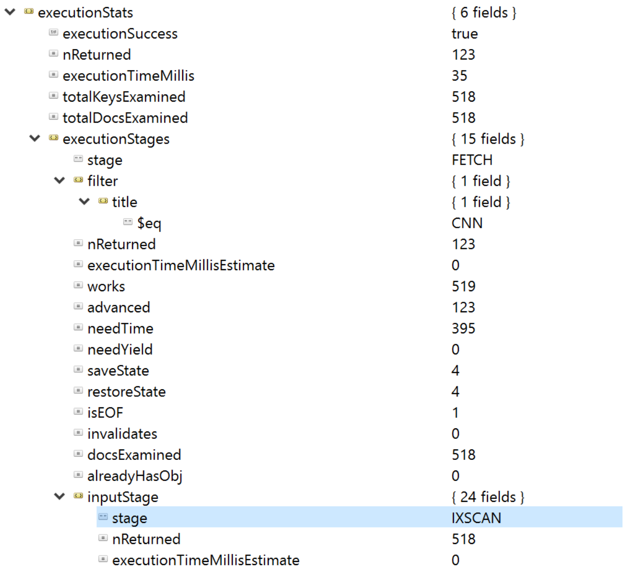
Run the following command to set up an index on `revisions` collection.

```
db.revisions.createIndex({timestamp:1})
```

The `explain` method on `cursor` object can be used to see how those indexes are used in query execution. Run the following command to see how index is used.

```
db.getCollection("revisions").find(
{title:"CNN", timestamp:{$gte: new Date("2016-01-01")}})
.explain("executionStats");
```

The query tries to find the revisions of article "CNN" made in 2016. The query condition contains two fields: `title` and `timestamp`. There is an index on the `timestamp` field. The `executionStats` shows that the `nReturned` is 123, and this is the result of examining 518 documents. Figure 5 shows a sample output. The query runs in two stages: an input stage and a filter stage. The `timpstamp` index is used in the input stage and returns 518 documents satisfy the timestamp condition. The filter stages examines the 518 documents by comparing their titles with "CNN", which returns 123 Documents as the query results.



```
executionStats                          { 6 fields }
  executionSuccess                      true
  nReturned                             123
  executionTimeMillis                   35
  totalKeysExamined                     518
  totalDocsExamined                     518
  executionStages                       { 15 fields }
    stage                               FETCH
    filter                              { 1 field }
      title                             { 1 field }
        $eq                             CNN
    nReturned                           123
    executionTimeMillisEstimate         0
    works                               519
    advanced                            123
    needTime                            395
    needYield                           0
    saveState                           4
    restoreState                        4
    isEOF                               1
    invalidates                         0
    docsExamined                        518
    alreadyHasObj                       0
    inputStage                          { 24 fields }
      stage                             IXSCAN
      nReturned                         518
      executionTimeMillisEstimate       0
```

Figure 5: Mongodb Query Execution Statistics

Question 8: Write your own query/aggregation

Write your own query or aggregation to find out:

- All revisions after 2016 in article 'BBC'.
- The number of minor revisions in the `revisions` collection. A minor revision has an `minor` field with no value. you can use the `$exist` operator to check if certain field exists (`https://docs.mongodb.com/manual/reference/operator/query/exists/`)
- The number of minor revisions belonging to each article: 'BBC' and 'CNN'