THE UNIVERSITY OF SYDNEY

School of Computer Science
Dr. Ying Zhou

**COMP5338: Advanced Data Models** **Sem. 2/2020**

# Week 2: MongoDB Basic Queries

**01.09.2020**

## Learning Objectives

In this week's lab, we work on a small data set downloaded from wikipedia. The data set contains one month's page revision data of two wiki pages: `Donald Trump` and `Hillary Clinton`. It also contains information of editors who made those revisions. We use the data set to practice basic MongoDB features including:

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

### Question 1: Start MongoDB engine and Robo 3T

We assume that you have installed and tested MongoDB and Robo3T on your own computer following week 1 MongDB setup guide. Start MongoDB engine following the MongoDB installation instruction. Start Robo 3T following week 1 MongoDB set up guideline and make a connection to the MongoDB engine on the same machine.

### Question 2: Import json file to MongoDB collection

Download the three json files from Canvas:

- `Donald_Trump2016-07-01.json`,
- `Hillary_Clinton2016-07-01.json`,
- `users.json`.

The content of the files can be inspected using any text editor. The first two files contain revision data of two wiki pages 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. The `users.json` contains editor data stored as a large array, with each object representing an editor.

MongoDB provides a simple tool `mongoimport` to import JSON, CSV or TSV files to mongo collection. From MongoDB 4.4, you need to install `mongoimport` separately, as part of MongoDB Database Tools package. Instructions on how to install MongoDB Database Tools can be found in the documentation. We will use it to import the revision and user data in the two newly created collections: `revisions` and `users`.

Open a shell window (command window or power shell window on windows OS) and change to the directory where the files are stored.

The following command import a file to the `revisions` collection. Note the command should be put in a single line, we break it in two lines for displaying purpose

```
mongoimport --jsonArray --db wikipedia --collection \
revisions --file <full-path-to-your-file-including-filenames>
```

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.

Run similar command to import the other two json files into the their respective collections.

### Question 3: Simple MongoDB read queries

Go to Robo 3T main screen, you will see the newly created collections: `revisions`, `users` on the left panel. Double click collection name`revisions`. 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 1. 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. We will use script to update the field type and value in next week's tutorial.

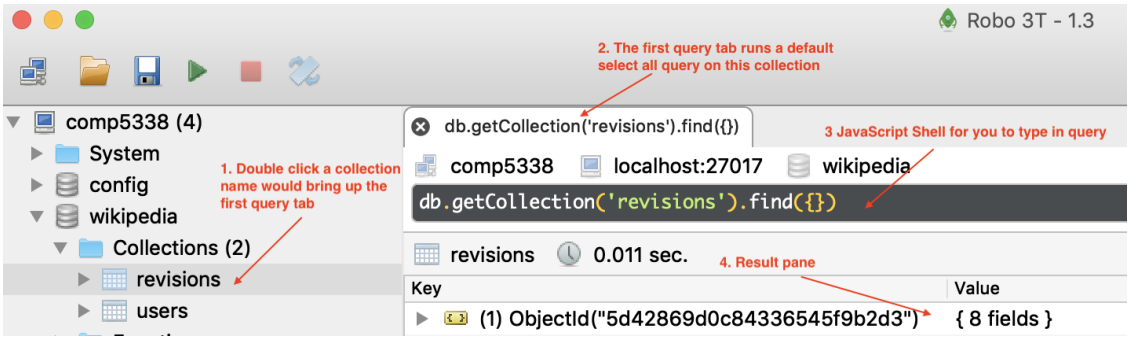


Figure 1: Robo3T first query and result

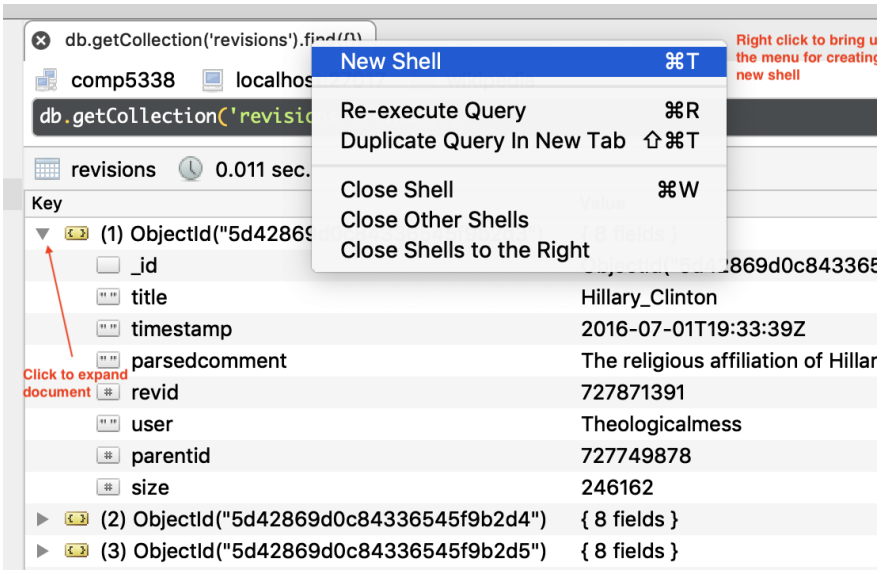


Figure 2: Robo3T New Query Tab

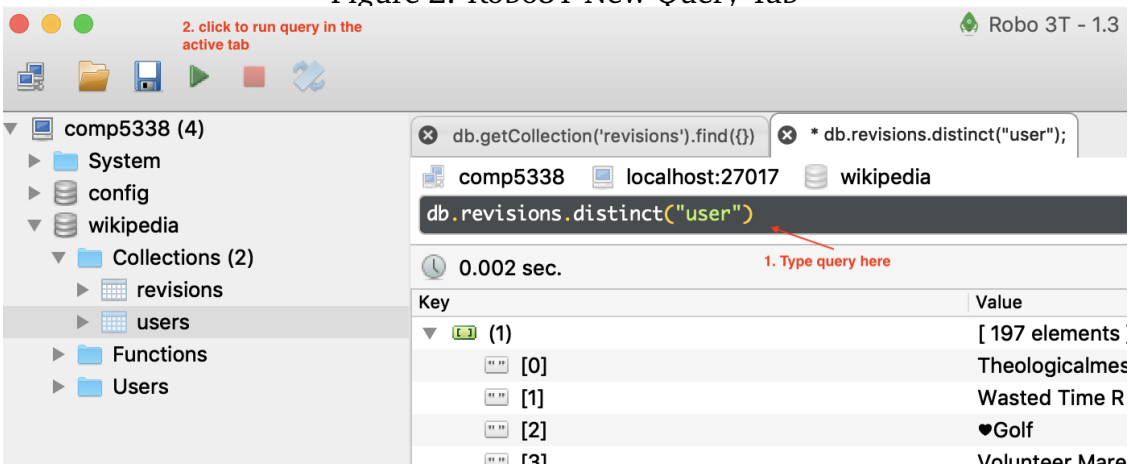


Figure 3: Robo3T Run Queries

There can be 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. Figure 2 and 3 show you how to open a new tab and run queries in it.

You may slightly modify the automatically generated query to see how many documents are in each collection. The two sample queries find out the number of documents in `revisions` and `users` collections.

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

Here are a few other queries you can try:

1. Find all female editors from the `users` collection:
   `db.users.find({gender:'female'})`

2. Find distinct users in the `revisions` collection:
   `db.revisions.distinct('user')`
3. Find the top 5 users with the highest number of edit count from `users` collection:
   `db.users.find().sort({editcount:-1}).limit(5)`
   This query starts by retrieving all documents in the `users` collection (`db.users.find()`), then sort them by the field `editcount` in descending order (`.sort({editcount:-1})`) and output only the first 5 documents in the results (`.limit(5)`).
   If you look at the details of each returned document, you will notice that the top 3 users all made very large number of edits and their name all contain the string "bot". This is a good indication that they are not human editors but are Robot designed for tedious editing task such as fixing typos or simple grammatical errors. We want to get rid of these type of editors in the next query.
4. Find the top 5 non-bot users with the highest number of edit count from `users` collections:

   ```
   db.users.find({name: {$regex: '^((?!bot).)*$', $options: 'si' }})
   .sort({editcount:-1}).limit(5)
   ```

   This query assumes that all bot users have names with the string "bot" in it. The above query uses a regular expression match to filter out all names with the sub string "bot". The `$options: 'si'` indicates that we want to be case insensitive and use the wild card character "." in the regular expression. For more information about regular expression matching, see this document page: https://docs.mongodb.com/manual/reference/operator/query/regex/. Compare the result of this query to the result of the previous query, you notice it does not contain the top 3 editors with "bot" in the name.

db.users.find({name: {$regex: '^((?!bot).)*$', $options: 'si' }}).sort({editcount:-1}).limit(5)

**Question 4: Simple MongoDB update queries**

Suppose we want to store basic user information on the `revisions` collection together with the revision that user made. An example of the original document looks like the following:

```
{
    "_id" : ObjectId("5799843ee2cbe65d76ed919b"),
    "title" : "Hillary_Clinton",
    "timestamp" : "2016-07-23T02:02:06Z",
    "revid" : 731113635,
    "user" : "BD2412",
    "parentid" : 731113573,
    "size" : 251742
}
```

We want to use an embedded document to store the basic user information

```
{
    "_id" : ObjectId("5799843ee2cbe65d76ed919b"),
    "title" : "Hillary_Clinton",
    "timestamp" : "2016-07-23T02:02:06Z",
    "revid" : 731113635,
    "user" : "BD2412",
    "userprofile": {
        "editcount" : 775871,
        "gender" : "male",
        "userid" : 196446,
        "name" : "BD2412",
        "registration" : "2005-02-20T18:14:10Z"
    }
    "parentid" : 731113573,
    "size" : 251742
}
```

Below is an sample update query to update documents in the `revisions` collection.

Updates many documents. i.e. all entries of BD2412

```
db.revisions.updateMany({user:"BD2412"},
{$set:{"userprofile": {
"editcount" : 775871,
"gender" : "male",
"userid" : 196446,
"name" : "BD2412",
"registration" : ISODate("2005-02-20T18:14:10Z")
    } }})
```

This query has two arguments: the first one finds the document meets given criteria (`user:'BD2412'`); the second one update the document(s) by set the value to a field called 'userprofile' (`$set:{'userprofile':{...}}`). Note, we convert the String date type to proper `ISODate` in the embedded document (`''registration'' : ISODate(''2005-02-20T18:14:10Z'')`).

Now write your own query to update documents belong to user 'Yobot'. The user profile can be find by querying the `users` collection. You can obtain a text version of each returned document by right click the document in the result pane and select "View Object" from the drop down menu (see figure 4). The text version of the document will be displayed in a prompt window (see figure 5).

Embedded fields can be queried. For instance, the following query find out the wiki page titles that have been edited by a male editor using data stored in `revisions` collection.

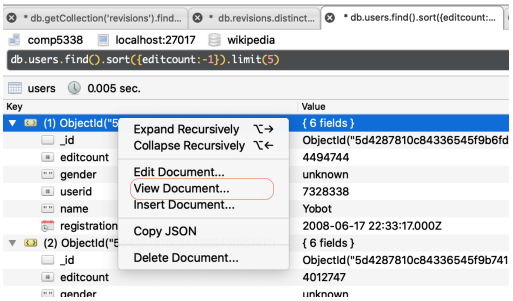


Figure 4: Robo 3T View Object Menu


Figure 5: Robo 3T Object Source View

```
db.revisions.find({"userprofile.gender": 'male'},{title:1})
```

**Question 5: Write your own query**

Write your own query to find out:

- 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 page: `Hillary Clinton`
- the non-bot users that have made over 100,000 edits

db.revisions.find({minor:{ $exists: false }}).count()

db.revisions.find({ minor:{ $exists: true }}, {title: "Hillary Clinton"}).count()

db.users.find({name: {$regex: '^((?!bot).)*$', $options: 'si' }, editcount: {"$gt": 100000} })