MongoDB Database Administration

MIHALIS TSOUKALOS

Mihalis Tsoukalos is a UNIX System Administrator, a programmer (UNIX and iOS), a DBA, a mathematician, and a technical writer. You can reach him at http://www.mtsoukalos.eu/ and @mactsouk. mactsouk@gmail.com

MongoDB [1, 2] is a document-oriented NoSQL database that has become quite popular. In this article, I will show you how to perform various administrative tasks, after setting up a dummy collection that will be used as an example. You will learn how to create and drop collections, use indexes, and convert a MongoDB database from using the MMAPv1 storage engine to using WiredTiger. I will also talk about mtools, which is a convenient set of Python scripts for processing MongoDB log files.

Table 1 introduces you to the MongoDB terminology compared to the well-known Relational Database terminology.

SQL Term	MongoDB Term
Database	Database
Table	Collection
Index	Index
Row	BSON document
Column	BSON field
Primary key	_id field
Group	Aggregation
Join	Embedding and linking

Table 1: MongoDB and RDBMS terminology

For this article, I used MongoDB version 3.2.1 [3] running on Mac OS X; however, most of the presented commands will also work on the older 2.6 version. MongoDB was installed on Mac OS X using the Homebrew [5] package. You will most certainly find a ready-to-install package for your operating system, but you can also get precompiled MongoDB binaries from [4].

A bit of warning before continuing with the rest of the article. MongoDB neither monitors disk space nor displays any warning messages related to disk space, so it is up to the system administrator to deal with such issues. The only occasion where you will hear MongoDB complaining about disk space is when there is no disk space left!

Basic DBA Commands

Most of the tasks presented in this article will be performed from the Mongo shell, which starts by executing the `mongo` command. The name of the MongoDB server process is `mongod`. First, you should run the following JavaScript code from the MongoDB shell in order to add some data on your database and have something to experiment with:

```
> use login
switched to db login
> for (var i=3D0; i<100000; i++)
{ db.myData.insert({x:i, y:2*i}); }
WriteResult({ "nInserted" : 1 })
> db.myData.count();
100000
```

The first command switches to the "login" database—if the database does not exist, it will be automatically created. It then uses a JavaScript for loop so that it can insert 100,000 documents to the myData collection, which will also be created if needed. The last command shows how you can find out the total number of records that exist in a collection.

As you can see, you do not have to specifically create a collection or its fields (keys). If you try to insert a document to a collection that does not exist, MongoDB will automatically create the collection. Additionally, if you try to insert a document that has a different set of keys to an existing collection, MongoDB will create it without any complaints. This means that small typographical errors cannot be detected very easily.

If you wish to delete the entire "myData" collection and start with an empty one, you should use the drop() method:

```
> db.myData.drop();
true
```

As saving data on a database takes disk space, it is good to know how to delete an entire database. The following command deletes the entire "login" database, including its data files:

```
> use login
switched to db login
> db.runCommand( { dropDatabase: 1 } )
{ "dropped" : "login", "ok" : 1 }
```

Should you wish to view the list of the available databases on the MongoDB server you are connected to, you can execute the following command from the MongoDB shell:

```
> show databases
LXF    0.031GB
local  0.078GB
login  0.063GB
test   0.031GB
```

After you select a database, you can see its available collections as follows:

```
> show tables
myData
system.indexes
```

The system.indexes collection contains information about the indexes of a database. However, it should not be accessed directly as if it was a regular collection but with the help of the getIndexes() function.

You can manually start a MongoDB server process from the command line as follows:

```
$ mongod --fork --logpath a.log --smallfiles --oplogSize 50 --port
  27101 --dbpath w1 --replSet w --logappend
```

The --port parameter defines the port number that the MongoDB server will listen to, the --dbpath value defines the folder that will contain the database files, the value of the --logpath parameter shows the file that will hold the log messages, and the --fork parameter tells the operating system that the process will run in the background without a terminal. The --replset parameter defines the name of the replica set and should only be included when you want to define a replica set. The --logappend parameter tells the MongoDB process to append to the log file instead of overwriting it.

By default, MongoDB does not require users to log in to connect from the local machine, which means that anyone who has access to a machine can do whatever she wants with the entire MongoDB server. In order to enable authorization you must use the --auth when running mongod or use the security.authorization setting in the configuration file of MongoDB.

Converting a Database from MMAPv1 to WiredTiger

MongoDB currently supports two Storage Engines: MMAPv1 and WiredTiger [6]; the good thing is that all commands related to database administration are the same regardless of the storage engine used.

WiredTiger is an open source project that was built separately from MongoDB and is also used by other databases. Apart from the performance gains, its main advantage is that it supports document-level locking, allowing you to lock only the document you are currently processing instead of locking the entire collection your document belongs to. Starting with MongoDB version 3.2, WiredTiger is the default storage engine whereas previous MongoDB versions used MMAPv1. This section will show you how to convert a MongoDB database from the MMAPv1 to the WiredTiger storage engine.

The data directory of a MongoDB database that uses WiredTiger looks like the following:

```
$ ll data/
total 272
-rw-r--r--  1 mtsouk  staff    49 Feb 13 14:12 WiredTiger
-rw-r--r--  1 mtsouk  staff    21 Feb 13 14:12 WiredTiger.lock
-rw-r--r--  1 mtsouk  staff   918 Feb 13 14:14 WiredTiger.turtle
-rw-r--r--  1 mtsouk  staff 40960 Feb 13 14:14 WiredTiger.wt
-rw-r--r--  1 mtsouk  staff  4096 Feb 13 14:12 WiredTigerLAS.wt
-rw-r--r--  1 mtsouk  staff 16384 Feb 13 14:13 _mdb_catalog.wt
-rw-r--r--  1 mtsouk  staff 16384 Feb 13 14:13
  collection-0-7818407182795123090.wt
-rw-r--r--  1 mtsouk  staff  4096 Feb 13 14:21
  collection-2-7818407182795123090.wt
drwxr-xr-x  4 mtsouk  staff   136 Feb 13 14:21 diagnostic.data
-rw-r--r--  1 mtsouk  staff 16384 Feb 13 14:13 index-1
-7818407182795123090.wt
-rw-r--r--  1 mtsouk  staff  4096 Feb 13 14:21 index-3
-7818407182795123090.wt
drwxr-xr-x  5 mtsouk  staff   170 Feb 13 14:12 journal
-rw-r--r--  1 mtsouk  staff     5 Feb 13 14:12 mongod.lock
-rw-r--r--  1 mtsouk  staff 16384 Feb 13 14:14 sizeStorer.wt
-rw-r--r--  1 mtsouk  staff    95 Feb 13 14:12 storage.bson
```

The filenames in the data directory show whether you are using WiredTiger or not. However, you can find the storage engine of your MongoDB server from the shell by executing the following command:

```
> db.serverStatus().storageEngine
{ "name" : "mmapv1", "supportsCommittedReads" : false }
```

Alternatively, you can execute the following command, which gives the same information in a different format:

```
> db.serverStatus().storageEngine.name
mmapv1
```

So the previous database uses MMAPv1. Using the same commands on a database that uses WiredTiger produces the following output:

```
> db.serverStatus().storageEngine
{ "name" : "wiredTiger", "supportsCommittedReads" : true }
> db.serverStatus().storageEngine.name
wiredTiger
```

Executing the "`db.serverStatus().wiredTiger`" command on a database that uses WiredTiger produces a large amount of output with information about various WiredTiger parameters and useful statistics, including buffer sizes, number of update, insert, remove, search calls, cache data, connection data, etc.

Converting a MongoDB 3.0.x or newer database that uses MMAPv1 to one using WiredTiger is a relatively easy process. You will first need to back up your data, delete existing data files, change the configuration file of MongoDB in order to make it use WiredTiger, and then import your backup data to MongoDB. Starting MongoDB with an empty data directory makes MongoDB generate all necessary files, which makes our job much easier. For Mac OS X, the required steps and commands are the following:

```
$ mongo
MongoDB shell version: 3.2.1
connecting to: test
> db.serverStatus().storageEngine.name
mmapv1
> use login
switched to db login
> db.myData.count()
100000
<Control-C>
$ mongodump -d login -c myData
2016-02-13T18:40:00.114+0200 writing login.myData to
2016-02-13T18:40:00.367+0200 done dumping login.myData
      (100000 documents)
$ launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.
      mongodb.plist
$ rm -rf /usr/local/var/mongodb
$ mkdir /usr/local/var/mongodb
$ cp /usr/local/etc/mongod.conf{,.old}
$ vi /usr/local/etc/mongod.conf
$ diff /usr/local/etc/mongod.conf /usr/local/etc/mongod.conf.old
7d6
<   engine: "wiredTiger"
$ launchctl load ~/Library/LaunchAgents/homebrew.mxcl.
      mongodb.plist
$ mongorestore
$ mongo
MongoDB shell version: 3.2.1
connecting to: test
> db.serverStatus().storageEngine.name
wiredTiger
> use login
switched to db login
> db.myData.count()
100000
```

The two `count()` commands verify that all documents have been successfully imported. As you can understand from the output of the `diff` command, you just need to add a single line in the MongoDB configuration file to make MongoDB use WiredTiger. The `mongodump` command creates a directory named "`dump`" inside the current directory. If you execute the `mongorestore` command from the same directory you ran `mongodump`, then `mongorestore` will automatically find and use the "`dump`" directory.

About Log Files

The default location of the log files of a HomeBrew [5] MongoDB installation on a Mac OS X system as defined in the `/usr/local/etc/mongod.conf` file is `/usr/local/var/log/mongodb/mongo.log`. Data files are kept inside `/usr/local/var/mongodb`. If you choose not to use `mongod.conf`, you will have to define all required parameters from the command line. On a usual Linux installation, you can find `mongodb.conf` inside `/etc`, and `mongodb.log` inside `/var/log/mongodb`, whereas the data directory is usually located at `/var/lib/mongodb/`.

If you try to start a MongoDB server without a proper log file directory, you are going to get the following error message:

```
2015-11-29T12:01:54.349+0200 F CONTROL  Failed global
initialization:
FileNotOpen Failed to open "/Users/mtsouk/Downloads/./aPath
/a.log"
```

You are also going to a get a similar error message if the data directory is missing:

```
2015-11-29T12:02:42.547+0200 I STORAGE  [initandlisten]
exception in
initAndListen: 29 Data directory ./myData not found.,
terminating
2015-11-29T12:02:42.548+0200 I CONTROL  [initandlisten]
dbexit:  rc: 100
```

Dropping an entire database and deleting its data files produces the following kind of log messages:

```
2016-02-13T11:04:29.593+0200 I COMMAND  [conn25]
dropDatabase
      green starting
2016-02-13T11:04:29.714+0200 I JOURNAL  [conn25]
journalCleanup...
2016-02-13T11:04:29.714+0200 I JOURNAL  [conn25]
removeJournalFiles
2016-02-13T11:04:29.718+0200 I JOURNAL  [conn25]
journalCleanup...
2016-02-13T11:04:29.718+0200 I JOURNAL  [conn25]
removeJournalFiles
2016-02-13T11:04:29.720+0200 I COMMAND  [conn25]
dropDatabase
      green finished
2016-02-13T11:04:29.720+0200 I COMMAND  [conn25] command
green
command: dropDatabase { dropDatabase: 1.0 } keyUpdates:0
writeConflicts:0 numYields:0 reslen:41 locks:{ Global:
{ acquireCount:
{ r: 2, w: 1, W: 1 } }, MMAPV1Journal: { acquireCount: { w: 4 } },
Database: { acquireCount: { W: 1 } } } protocol:op_command 126ms
```

Therefore, a command like the following would show the databases that were dropped from your MongoDB installation:

```
$ grep -w dropDatabase /usr/local/var/log/mongodb/mongo.log |
grep -w finished | awk {'print $6'}
```

Looking at log files is an important task of a DBA, so the next section presents mtools, a set of Python scripts that deal with MongoDB log files.

The mtools Set of Scripts

mtools [7] is a set of Python scripts that can help you parse, filter, and visualize MongoDB log files. The mtools set includes mloginfo, mlogfilter, mplotqueries, mlogvis, mgenerate, and mlaunch. Please note that at the moment, mtools is not compatible with Python 3. The mloginfo script displays information about the data of a log file in a format similar to the following:

```
$ mloginfo /usr/local/var/log/mongodb/mongo.log
      source: /usr/local/var/log/mongodb/mongo.log
        host: iMac.local:27017
       start: 2015 Sep 12 19:21:04.358
         end: 2016 Feb 14 23:10:16.192
date format: iso8601-local
      length: 5118
      binary: mongod
     version: 3.0.6 -> 3.0.7 -> 3.2.0 -> 3.2.1
     storage: wiredTiger
```

As you can see, mloginfo shows the MongoDB versions that generated the log messages as well as the storage engine and the time period of the log entries.

The mlogfilter script is a log file parser that can be used for extracting information out of busy MongoDB log files. Think of it as an advanced `grep` utility for MongoDB log files. The mlaunch script lets you create MongoDB test environments on your local machine quickly. The mplotqueries script is used for visualizing MongoDB log files. The mlogvis script is similar to the mplotqueries script, but instead of creating a graphics file, its output is an interactive HTML page on a Web browser. Last, the mgenerate script produces pseudo-random data for populating MongoDB databases with sample data.

The first thing you should learn is about the installation of the mtools package, which can be done as follows:

```
$ pip install mtools
...
Successfully built mtools psutil
Installing collected packages: psutil, mtools
Successfully installed mtools-1.1.9 psutil-3.4.2
```

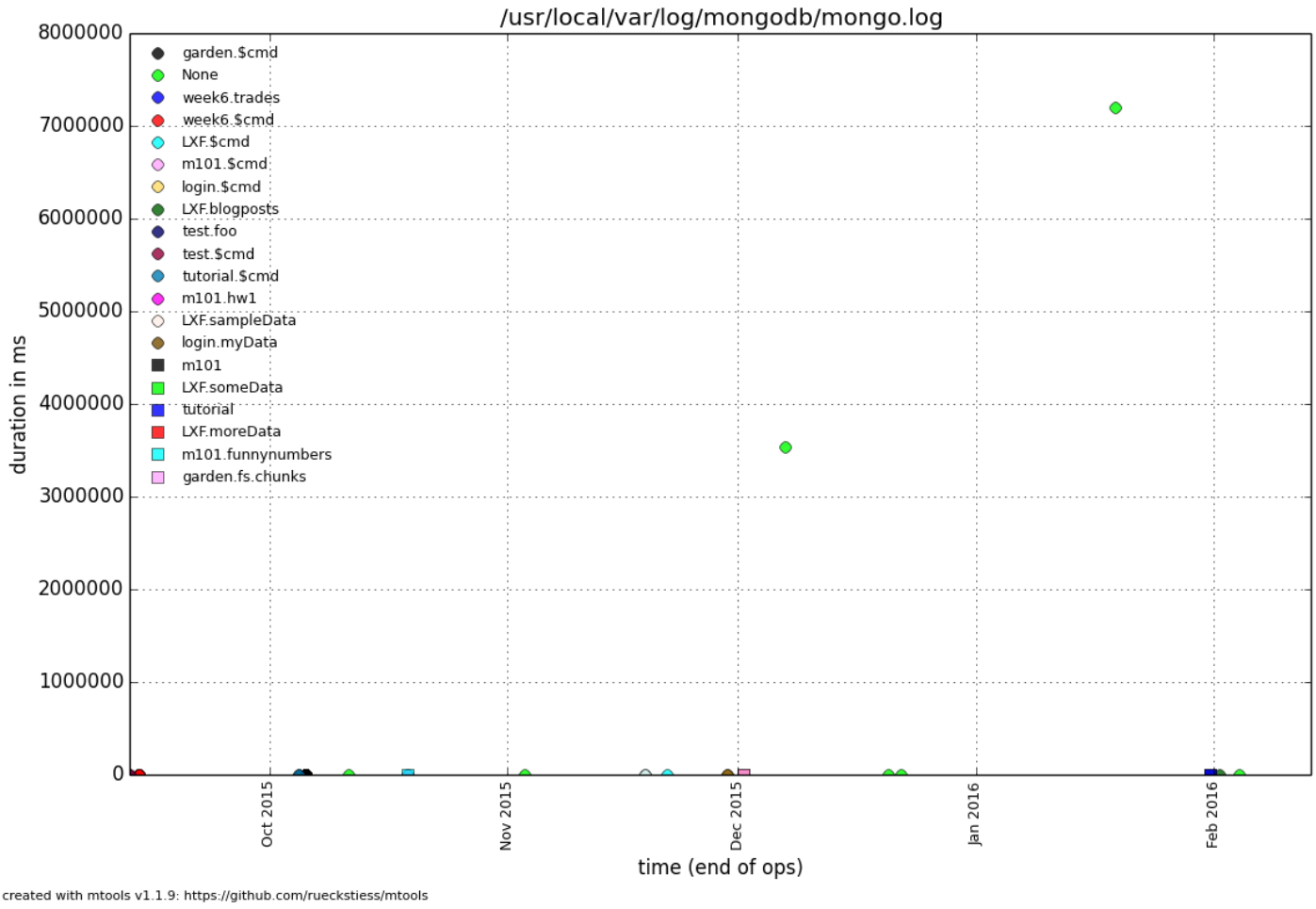

Figure 1: The output of the mplotqueries Python script on a log file from my local MongoDB server

For ease of installation, it is recommended that you use the `pip` utility to install mtools when possible; otherwise visit the mtools installation page [8] for more information.

Now that you have installed mtools, it is time to use mplotqueries to get information from a MongoDB log file and plot it on screen. All you have to do is execute the following command:

```
$ mplotqueries /usr/local/var/log/mongodb/mongo.log
      --output-file login.png
```

The mplotqueries tool reads the specified log file and summarizes the information based on the name of the collection used. It then prints the duration of each operation on a timeline. Figure 1 shows the output mplotqueries produced using the MongoDB log file found on my Mac. As you can see, most operations ran almost instantly. This is a very handy way of overseeing your MongoDB data that can also run as a cron job. If you do not use the `--output-file` option, an interactive output is automatically going to be displayed on your screen.

Figure 2 shows the interactive output generated by mlogvis on the same log file as before.

Generally speaking, both mlogvis and mplotqueries are very handy for detecting outliers. If you think certain operations on a collection are running slow for some reason, you can use mlogfilter to look into them:

```
$ mlogfilter /usr/local/var/log/mongodb/mongo.log --word login
```

The previous command returns log entries that contain the "login" keyword. One of them is the following:

```
015-11-29T15:33:11.226+0200 I COMMAND [conn46] command
login.$cmd
command: insert { insert: "myData", documents: [ { _id:
ObjectId('565afe9781f59f422de5bd05'), x: 0.0, y: 0.0 } ], ordered:
true } keyUpdates:0 writeConflicts:0 numYields:0 reslen:40 locks:{
Global: { acquireCount: { r: 2, w: 2 } }, MMAPV1Journal: {
```

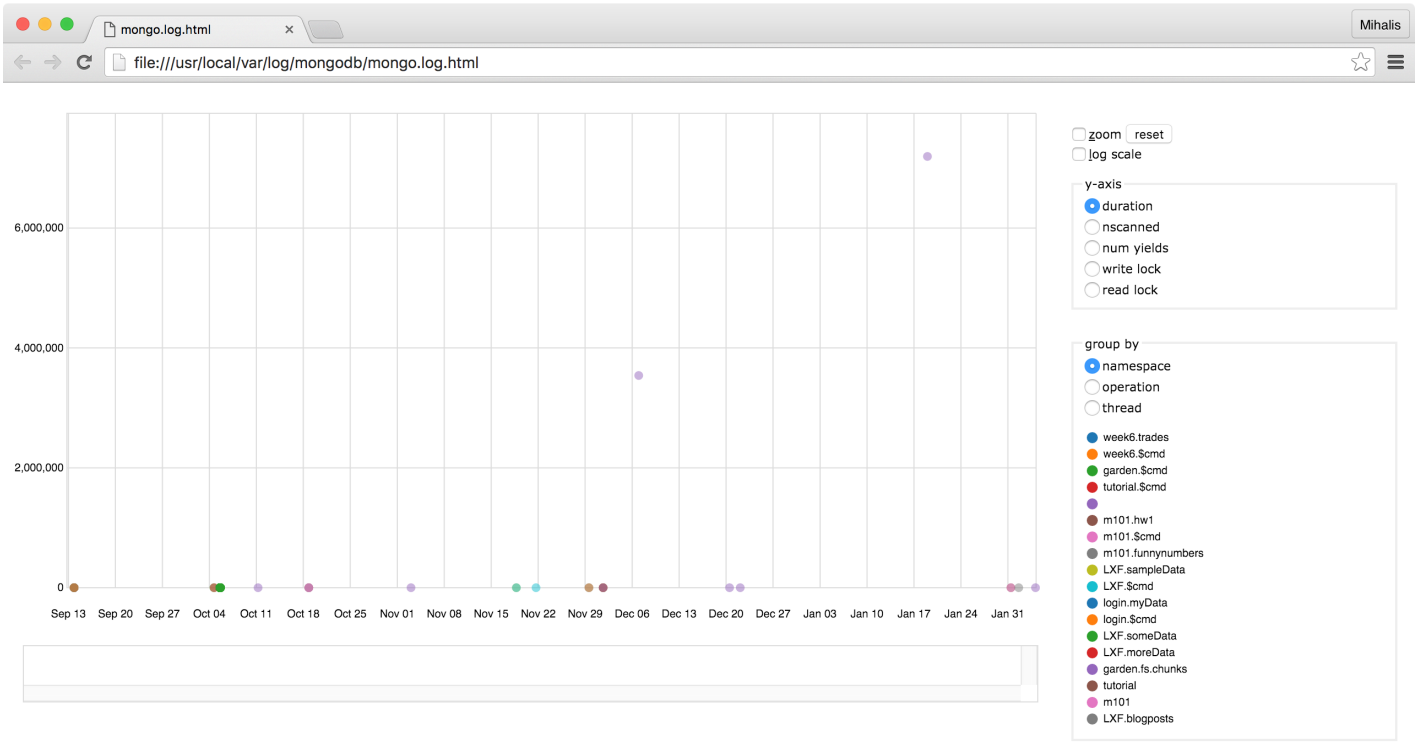

Figure 2: The output of the mlogvis Python script on a log file from my local MongoDB server

```
acquireCount: { w: 8 }, acquireWaitCount: { w: 1 },
timeAcquiringMicros: { w: 2058 } },
Database: { acquireCount:
{ w: 1, W: 1 } }, Collection: { acquireCount:
{ W: 1 } },
Metadata: { acquireCount: { W: 4 } } }
199ms
```

What the previous log entry says is that it took MongoDB 199 ms to execute an insert operation in the "myData" collection of the "login" database. If you want to speed up an insert operation, you might need to upgrade your hardware; however, if such operations do not happen very often, you should not be concerned.

If you want to find out all possible options for each tool, you can execute it with the `--help` option. I think that the mtools set of Python scripts is a useful tool to add to your arsenal.

Indexes

Analyzing a query is a very good way to find out why a query runs slow as well as how your query is executed. This can happen with the help of the `explain()` method. The interesting part from the `explain("executionStats")` command that displays how a query is executed is the following:

```
> db.myData.find({ "x": { $gt: 99990}
}).explain("executionStats")
...

"executionStats" : {
"executionSuccess" : true,
"nReturned" : 9,
"executionTimeMillis" : 46,
"totalKeysExamined" : 0,
"totalDocsExamined" : 100000,
"executionStages" : {
"stage" : "COLLSCAN",
"filter" : {
"x" : {
"$gt" : 99990
}
},
"nReturned" : 9,
"executionTimeMillisEstimate" : 0,
...
"docsExamined" : 100000
...
```

The explain command shows that the execution plan chosen does a full collection scan (COLLSCAN)—in other words, it searches all documents in the requested collection, which is not very efficient. As you can see from the values of both "totalDocsExamined" and "docsExamined," 100,000 documents were accessed in order to return nine documents, as indicated by the value of "nReturned".

This time I will define an index and then execute the previous query and show a part of its execution plan. Please note that as of MongoDB 3.0, the ensureIndex() command that used to create an index is deprecated; you should use createIndex() instead. The index for the "x" key will be created as follows:

```
> db.myData.createIndex({"x":1})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

You can verify that the index you just created is there with the help of the getIndexes() function that reads the system.indexes collection:

```
> db.myData.getIndexes()
[
{
"v" : 1,
"key" : {
"_id" : 1
},
"name" : "_id_",
"ns" : "login.myData"
},
{
"v" : 1,
"key" : {
"x" : 1
},
"name" : "x_1",
"ns" : "login.myData"
}
]
```

As you can see, the index for the "x" key is named "x_1". Please note that MongoDB automatically creates an index for the _id field for every collection.

By executing exactly the same explain() command, you can verify the usefulness of the index. The interesting part of the output is the following:

```
…
"executionStats" : {
"executionSuccess" : true,
"nReturned" : 9,
"executionTimeMillis" : 1,
"totalKeysExamined" : 9,
"totalDocsExamined" : 9,
…
"inputStage" : {
"stage" : "IXSCAN",
"nReturned" : 9,
"executionTimeMillisEstimate" : 0,
…
```

This query returned the results by scanning the index keys (IXSCAN); therefore, it is significantly faster than the previous query. As you can also see, the select query accessed only nine documents this time, as indicated by the value of "totalDocs Examined" in order to return nine documents, which is perfect!

Summary

This article is far from complete as no single article can cover all aspects of MongoDB administration. For example, Replication and Sharding were not covered at all. However, the commands and knowledge presented will help you start working effectively with a MongoDB database and perform many administrative tasks.

References

[1] MongoDB site: https://www.mongodb.org/.

[2] Kristina Chodorow, *MongoDB: The Definitive Guide*, 2nd edition (O'Reilly Media, 2013).

[3] Kyle Banker, Peter Bakkum, Shaun Verch, Douglas Garrett and Tim Hawkins, *MongoDB in Action*, 2nd edition (Manning Publications, 2016).

[4] Download MongoDB: https://www.mongodb.org/downloads.

[5] HomeBrew: http://brew.sh/.

[6] WiredTiger: http://www.wiredtiger.com/.

[7] mtools: https://github.com/rueckstiess/mtools.

[8] mtools installation: https://github.com/rueckstiess/mtools/blob/master/INSTALL.md.

Save the Date!

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

OSDI '16

12th USENIX Symposium on Operating Systems Design and Implementation

November 2–4, 2016 • Savannah, GA

Join us in Savannah, GA, November 2–4, 2016, for the **12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)**. The Symposium brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software.

Co-located with OSDI '16 on Tuesday, November 1:

- **Diversity '16:** 2016 Workshop on Supporting Diversity in Systems Research
- **INFLOW '16:** 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads

The full program and registration will be available in August.

www.usenix.org/osdi16