

Technical Debt

a perspective on agile development, and other things, from jon kern.

- [Home](#)
- [About](#)
- [Crib Sheets](#)
 - [Git is It!](#)
 - [HAML](#)
 - [Map-Reduce with MongoMapper \(Part 1\)](#)
- [Getting Started with Ruby](#)
 - [Installing Basics](#)
 - [MongoDB](#)
 - [RVM Quick Start](#)

« [Configuring MongoMapper Indexes in Rails App](#)

[MongoDB Index Performance](#) »

Nov
26

A Walk Through the Valley of Indexing in MongoDB

Categories:

- [Database](#),
- [MongoDB](#),
- [mongomapper](#)

by [jon](#)

As you walk through the valley of MongoDB performance, you will undoubtedly find yourself wanting to optimize your indexes at some point or other.

How to Watch Your Queries

Run your database with profiling on. I have an alias for starting up mongo in profile mode ('p' stands for profile):

```
1 | alias mongop="<mongodb-install>/bin/mongod
2 | --smallfiles --noprealloc --profile=1 --dbpath <mongodb-install>/data/db"
```

This will default to considering queries > 100ms being deemed “slow.”

Add a logger (if you are using MongoMapper) and tail the log file to see the queries.

```
1 | ##### MONGODB SETTINGS #####
2 | # You can use :logger => Rails.logger, to get output of the mongo queries, or create
3 | logger = Logger.new('mongo-development.log')
4 | MongoMapper.connection = Mongo::Connection.new('localhost', 27017, {:pool_size => 5,
5 | MongoMapper.database = "mdalert-development"
```

Run your query/exercise the app.

Examine the mongo log (trimmed for legibility), and look for the primary collection you were querying (**bolded** below).

```
1 | ['$cmd'].find("#settings", "query"=>{:identifier=>"ItemsPerPage"}, "fields"=>nil}>).limit(-1)
2 | ['$cmd'].find("#accounts", "query"=>{:state=>"active"}, "fields"=>nil}>).limit(-1)
3 | <strong>['accounts'].find({:state=>"active"}).limit(15).sort(email1)</strong>
4 | ['settings'].find({:identifier=>"AutoEmail"}).limit(-1)
```

Open up the mongo shell (<mongodb-install>/bin/mongo) and enter the query that you want explained. Hint, you can take much of it from the query in the log.

Without any indexes, you can see the query is scanning the entire table basically. A bad thing! Another tip is the cursor type is “BasicCursor.”

```
1 > db.accounts.find({state: "active"}).limit(15).sort({email: 1}).explain();
2 {
3   "cursor" : "BasicCursor",
4   "nscanned" : 11002,
5   "nscannedObjects" : 11002,
6   "n" : 15,
7   "scanAndOrder" : true,
8   "millis" : 44,
9   "nYields" : 0,
10  "nChunkSkips" : 0,
11  "isMultiKey" : false,
12  "indexOnly" : false,
13  "indexBounds" : {
14  }
15 }
```

Since I was doing a find on state, and a sort on email (or last_name), I added a compound index using MongoMapper (you could have just easily done it at the mongo console).

```
1 Account.ensure_index([[:state,1],[email,1]])
2 Account.ensure_index([[:state,1],[last_name,1]])
```

Re-running the explain, you can see

- the cursor type is now BtreeCursor (*i.e.*, using an index)
- the entire table is not scanned.
- The retrieval went from 44 millis down to 2 millis
- Success!!

```
1 > db.accounts.find({state: "active"}).limit(15).sort({email: 1}).explain();
2 {
3   "cursor" : "BtreeCursor state_1_email_1",
4   "nscanned" : 15,
5   "nscannedObjects" : 15,
6   "n" : 15,
7   "millis" : 2,
8   "nYields" : 0,
9   "nChunkSkips" : 0,
10  "isMultiKey" : false,
11  "indexOnly" : false,
12  "indexBounds" : {
13    "state" : [ [ "active", "active" ] ],
14    "email" : [ [ {"$minElement" : 1}, {"$maxElement" : 1} ] ]
15  }
16 }
```

Fiddling a Bit More – Using the Profiler

You can drop into the mongo console and see more specifics using the mongo profiler.

```
1 > db.setProfilingLevel(1,15)
2 { "was" : 1, "slowms" : 100, "ok" : 1 }
```

For this example, I cleared the indexes on accounts. and I ran the following query, and examined its profile data.

Note: the timing can vary over successive runs, but it generally is fairly consistent — and it is close to the “millis” value you see in explain output.

```

3   { "ts" : ISODate("2011-11-27T21:09:04.237Z"),
4     "info" : "query mdalert-development.accounts
5     ntoreturn:15 scanAndOrder
6     reslen:8690
7     nscanned:11002
8     query: { query: { state: \"active\" },
9       orderby: { email: 1.0 } }
10    nreturned:15 163ms", "millis" : 43 }

```

Now let's add back the indexes... one at a time. First up, let's add "state."

```

1  > db.accounts.ensureIndex({state:1})
2  >db.accounts.find({state: "active"}).limit(15).sort({email: 1})
3  db.system.profile.find({info: /.accounts/})
4  { "ts" : ISODate("2011-11-27T21:26:29.801Z"),
5    "info" : "query mdalert-development.accounts
6    ntoreturn:15 scanAndOrder
7    reslen:546
8    nscanned:9747
9    query: { query: { state: \"active\" },
10      orderby: { email: 1.0 }, $explain: true }
11    nreturned:1 81ms", "millis" : 81 }

```

Hmmm. Not so good! Let's add in the compound index that we know we need, and run an explain:

```

1  >db.accounts.ensureIndex({state:1,email:1})
2  > db.accounts.find({state: "active"}).limit(15).sort({email: 1}).explain()
3  {
4    "cursor" : "BtreeCursor state_1_email_1",
5    "nscanned" : 15,
6    "nscannedObjects" : 15,
7    "n" : 15,
8    "millis" : 0,

```

And sure enough, we get good performance. The millis is so small, that this query will not show up in the profiler.

If you want to clear the profile stats, you'll soon find out you can't remove the documents. The only way I saw how to do it was as follows:

- restart mongod in non-profiling mode
- reopen the mongo console and type:

```

1  db.system.profile.drop()

```

You should now see the profile being empty:

```

1  > show profile
2  db.system.profile is empty

```

Now you can restart mongod in profiling mode and see your latest profiling data without all the ancient history.

Some Gotchas

Regex searches cannot be indexed

If your query is a regex, then the index can't help. With regex, retrieval is 501 ms (not bad, given 317K records):

```

3     "cursor" : BtreeCursor patient_name_1 multi ,
4     "nscanned" : 317265,
5     "nscannedObjects" : 27,
6     "n" : 27,
7     "millis" : 501,
8     "nYields" : 0,
9     "nChunkSkips" : 0,
10    "isMultiKey" : false,
11    "indexOnly" : false,
12    "indexBounds" : {
13      "patient_name" : [ [ "", { } ], [ /ben franklin/, /ben franklin/ ] ]
14    }
15  }

```

Without regex, it is essentially instantaneous:

```

1  > db.message_logs.find({patient_name:'Ben Franklin'}).sort({created_at:-1}).explain()
2  {
3    "cursor" : "BtreeCursor patient_name_1",
4    "nscanned" : 27,
5    "nscannedObjects" : 27,
6    "n" : 27,
7    "scanAndOrder" : true,
8    "millis" : 0,
9    "nYields" : 0,
10   "nChunkSkips" : 0,
11   "isMultiKey" : false,
12   "indexOnly" : false,
13   "indexBounds" : {
14     "patient_name" : [ [ "Ben Franklin", "Ben Franklin" ] ]
15   }
16 }

```

Tips for examining profiler output

- Look at the most recent offenders:
show profile
- Look at a single collection:
db.system.profile.find({info: /message_logs/})
- Look at a slow queries:
db.system.profile.find({millis : {\$gt : 500}})
- Look at a single collection with a specific query param, and a response >100ms:
db.system.profile.find({info: /message_logs/, info: /patient_name/, millis : {\$gt : 100}})

Summary

So here you have an example of how to see indexes in action, how to create them, and how to measure their effects.

References:

- Kyle Banker has a great post that puts [MongoDB indexes](#) into perspective: “[The Joy of Indexing](#).”
- And Richard Kreuter has a nice presentation on “[Indexing and Query Optimizer](#).”

Share this:



Tags: [indexing](#), [profiling](#)

1 ping

1. [MongoDB Index Performance » Technical Debt](#) says:

Comments have been disabled.

Recent Posts

- [Jira vs Post-It Notes](#)
- [Use Agile Wisely](#)
- [Uncle Bob Challenges The Architecture of a Rails App](#)
- [MongoDB Group Map-Reduce Performance](#)
- [Hiring a Team Member](#)


Categories

Select Category

Categories

[agile](#) [agile techniques](#) [Architecture](#) [certification](#) [cool tools](#) [cucumber](#) [Database](#) [Development](#) [humor](#) [kanban](#) [metaprogramming](#) [modeling](#) [MongoDB](#) [mongomapper](#) [off topic](#) [OOAD](#) [people](#) [Philosophy](#) [planning](#) [pow](#) [process](#) [quality](#) [rails](#) [refactoring](#) [rspec](#) [ruby](#) [RVM](#) [scrum](#) [survey](#) [talks](#) [testing](#) [tools](#) [Uncategorized](#)

Latest tweets

-  Loading tweets...

[Follow me on Twitter](#)

Search

Search

Copyright

© 2012 Technical Debt.

- [Return to top](#)

Powered by [WordPress](#) and the [Graphene Theme](#).

