Database Profiler ¶

On this page

The database profiler collects fine grained data about MongoDB write operations, cursors, database commands on a running `mongod` instance. You can enable profiling on a per-database or per-instance basis. The profiling level is also configurable when enabling profiling. The profiler is *off* by default.

The database profiler writes all the data it collects to the `system.profile` collection, which is a capped collection. See Database Profiler Output for overview of the data in the `system.profile` documents created by the profiler.

This document outlines a number of key administration options for the database profiler. For additional related information, consider the following resources:

- Database Profiler Output
- Profile Command
- `db.currentOp()`

Profiling Levels

The following profiling levels are available:

- `0` - the profiler is off, does not collect any data. `mongod` always writes operations longer than the `slowOpThresholdMs` threshold to its log. This is the default profiler level.
- `1` - collects profiling data for slow operations only. By default slow operations are those slower than 100 milliseconds.
  You can modify the threshold for “slow” operations with the `slowOpThresholdMs` runtime option or the `setParameter` command. See the Specify the Threshold for Slow Operations section for more information.

- 2 - collects profiling data for all database operations.

Enable Database Profiling and Set the Profiling Level

You can enable database profiling from the `mongo` shell or through a driver using the `profile` command. This section will describe how to do so from the `mongo` shell. See your driver documentation if you want to control the profiler from within your application.

When you enable profiling, you also set the profiling level. The profiler records data in the `system.profile` collection. MongoDB creates the `system.profile` collection in a database after you enable profiling for that database.

To enable profiling and set the profiling level, use the `db.setProfilingLevel()` helper in the `mongo` shell, passing the profiling level as a parameter. For example, to enable profiling for all database operations, consider the following operation in the `mongo` shell:

```
db.setProfilingLevel(2)
```

The shell returns a document showing the *previous* level of profiling. The `"ok" : 1` key-value pair indicates the operation succeeded:

```
{ "was" : 0, "slowms" : 100, "ok" : 1 }
```

To verify the new setting, see the Check Profiling Level section.

Specify the Threshold for Slow Operations

The threshold for slow operations applies to the entire `mongod` instance. When you change the threshold, you change it for all databases on the instance.

> **IMPORTANT:**
> Changing the slow operation threshold for the database profiler also affects the profiling subsystem's slow operation threshold for the entire `mongod` instance. Always set the threshold to the highest useful value.

By default the slow operation threshold is 100 milliseconds. Databases with a profiling level of `1` will log operations slower than 100 milliseconds.

To change the threshold, pass two parameters to the `db.setProfilingLevel()` helper in the `mongo` shell. The first parameter sets the profiling level for the current database, and the second sets the default slow operation threshold *for the entire* `mongod` *instance*.

For example, the following command sets the profiling level for the current database to `0`, which disables profiling, and sets the slow-operation threshold for the `mongod` instance to 20 milliseconds. Any database on the instance with a profiling level of `1` will use this threshold:

```
db.setProfilingLevel(0,20)
```

Check Profiling Level

To view the profiling level, issue the following from the `mongo` shell:

```
db.getProfilingStatus()
```

The shell returns a document similar to the following:

```
{ "was" : 0, "slowms" : 100 }
```

The `was` field indicates the current level of profiling.

The `slowms` field indicates how long an operation must exist in milliseconds for an operation to pass the "slow" threshold. MongoDB will log operations that take longer than the threshold if the profiling level is `1`. This document returns the profiling level in the `was` field. For an explanation of profiling levels, see Profiling Levels.

To return only the profiling level, use the `db.getProfilingLevel()` helper in the `mongo` as in the following:

```
db.getProfilingLevel()
```

Disable Profiling

To disable profiling, use the following helper in the `mongo` shell:

```
db.setProfilingLevel(0)
```

Enable Profiling for an Entire `mongod` Instance

For development purposes in testing environments, you can enable database profiling for an entire `mongod` instance. The profiling level applies to all databases provided by the `mongod` instance.

To enable profiling for a `mongod` instance, pass the following parameters to `mongod` at startup or within the configuration file:

```
mongod --profile 1 --slowms 15
```

This sets the profiling level to `1`, which collects profiling data for slow operations only, and defines slow operations as those that last longer than `15` milliseconds.

SEE ALSO:

`mode` and `slowOpThresholdMs`.

Database Profiling and Sharding

You *cannot* enable profiling on a `mongos` instance. To enable profiling in a sharded cluster, you must enable profiling for each `mongod` instance in the cluster.

View Profiler Data

The database profiler logs information about database operations in the `system.profile` collection.

To view profiling information, query the `system.profile` collection. You can use `$comment` to add data to the query document to make it easier to analyze data from the profiler. To view example queries, see Profiler Overhead.

For an explanation of the output data, see Database Profiler Output.

Example Profiler Data Queries

This section displays example queries to the `system.profile` collection. For an explanation of the query output, see Database Profiler Output.

To return the most recent 10 log entries in the `system.profile` collection, run a query similar to the following:

```
db.system.profile.find().limit(10).sort( { ts : -1 } ).pretty()
```

To return all operations except command operations ($cmd), run a query similar to the following:

```
db.system.profile.find( { op: { $ne : 'command' } } ).pretty()
```

To return operations for a particular collection, run a query similar to the following. This example returns operations in the `mydb` database's `test` collection:

```
db.system.profile.find( { ns : 'mydb.test' } ).pretty()
```

To return operations slower than 5 milliseconds, run a query similar to the following:

```
db.system.profile.find( { millis : { $gt : 5 } } ).pretty()
```

To return information from a certain time range, run a query similar to the following:

```
db.system.profile.find(
                       {
                        ts : {
                               $gt : new ISODate("2012-12-09T03:00:00Z") ,
                               $lt : new ISODate("2012-12-09T03:40:00Z")
                              }
                       }
                      ).pretty()
```

The following example looks at the time range, suppresses the `user` field from the output to make it easier to read, and sorts the results by how long each operation took to run:

```
db.system.profile.find(
                        {
                          ts : {
                                 $gt : new ISODate("2011-07-12T03:00:00Z") ,
                                 $lt : new ISODate("2011-07-12T03:40:00Z")
                                }
                        },
                        { user : 0 }
                       ).sort( { millis : -1 } )
```

Show the Five Most Recent Events

On a database that has profiling enabled, the `show profile` helper in the `mongo` shell displays the 5 most recent operations that took at least 1 millisecond to execute. Issue `show profile` from the `mongo` shell, as follows:

```
show profile
```

Profiler Overhead

When enabled, profiling has a minor effect on performance. The `system.profile` collection is a capped collection with a default size of 1 megabyte. A collection of this size can typically store several thousand profile documents, but some application may use more or less profiling data per operation.

Change Size of `system.profile` Collection on the Primary

To change the size of the `system.profile` collection, you must:

1. Disable profiling.
2. Drop the `system.profile` collection.
3. Create a new `system.profile` collection.
4. Re-enable profiling.

For example, to create a new `system.profile` collections that's `4000000` bytes, use the following sequence of operations in the `mongo` shell:

```
db.setProfilingLevel(0)

db.system.profile.drop()

db.createCollection( "system.profile", { capped: true, size:4000000 } )

db.setProfilingLevel(1)
```

Change Size of `system.profile` Collection on a Secondary

To change the size of the `system.profile` collection on a secondary, you must stop the secondary, run it as a standalone, and then perform the steps above. When done, restart the standalone as a member of the replica set. For more information, see Perform Maintenance on Replica Set Members.

Additional Resources

- MongoDB Performance Evaluation and Tuning Consulting Package