



To work out why a query is slow

• .explain()

• db.collection.explain()

• cursor.explain()

Profile

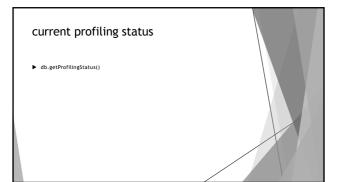
▶ db.setProfilingLevel(?)

▶ Level 0 - The profiler is off and does not collect any data. This is the default profiler level.

▶ Level 1 - The profiler collects data for operations that take longer than the value of slowms, which you can set.

▶ Level 2 - The profiler collects data for all operations.

3



What we achieve?

• when the query was run
• how fast it ran
• how many documents were examined
• the type of plan it used
• whether it was able to fully-use and index
• what sort of locking happened

5

6

8

## Sort queries by when they were recorded, showing just commands • db.system.profile.find(["command.pipeline": { Sexists: true } }, { "command.pipeline": 1 }).sort([Snatural:-1]).pretty();

Find all queries doing a COLLSCAN because there is no suitable index

be db.system.profile.find(["planSummary":{Seq:"COLLSCAN"], "op": {Seq:"query"}}).sort([millis:-1])

collscan - Collection scan

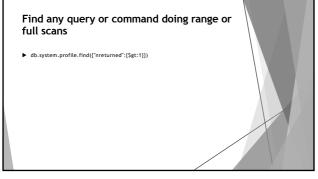
lixScan - Scan of data in index keys

FETCH - Retrieving documents

SHARD\_MERGE - Merging results from shards

SORT - Explicit sort rather than using index order

1



Find the source of the top ten slowest queries

• db.system.profile.find(["op": [Seq:'query"]], ["query": NumberInt(1), "millis": NumberInt(1) } ).sort([millis:-1], [Slimit:10]).pretty()

9

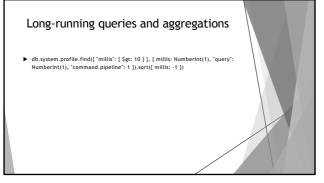
10

## Find the source of the top ten slowest aggregations • db.system.profile.find(["op": {Seq:'command"]}, {"command.pipeline": Numberint(1), "millis": Numberint(1) } ).sort([millis:-1],{Slimit:10}).pretty()

Find all queries that take more than ten milliseconds

▶ db.system.profile.find(["millis":[Sgt:10]]) .sort([millis:-1]))

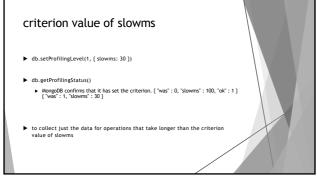
▶ displaying both queries and aggregations



Find all queries that took more than 20 ms

• db.system.profile.find(['millis':{Sgt:20]})

13 14





15 16