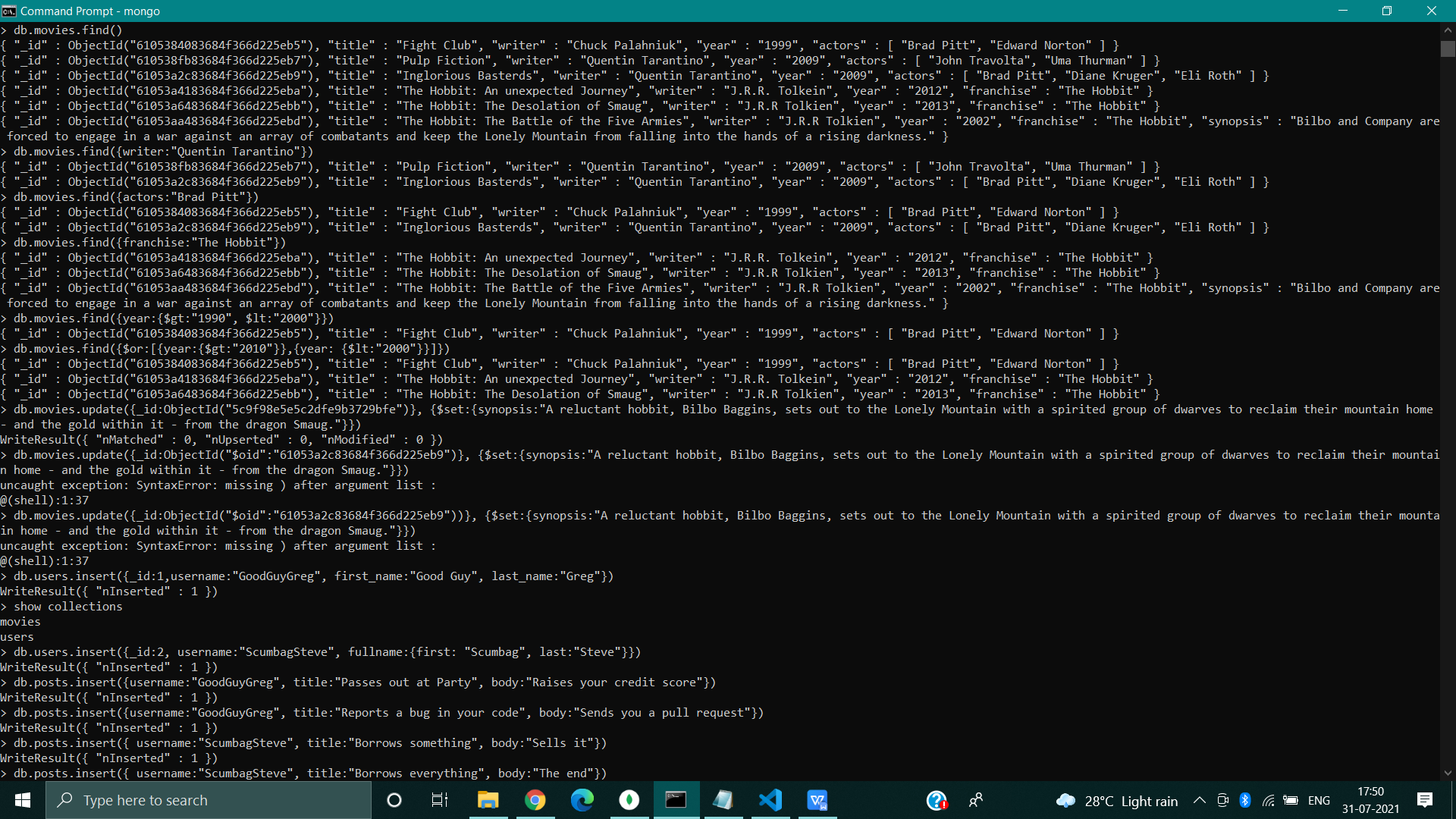
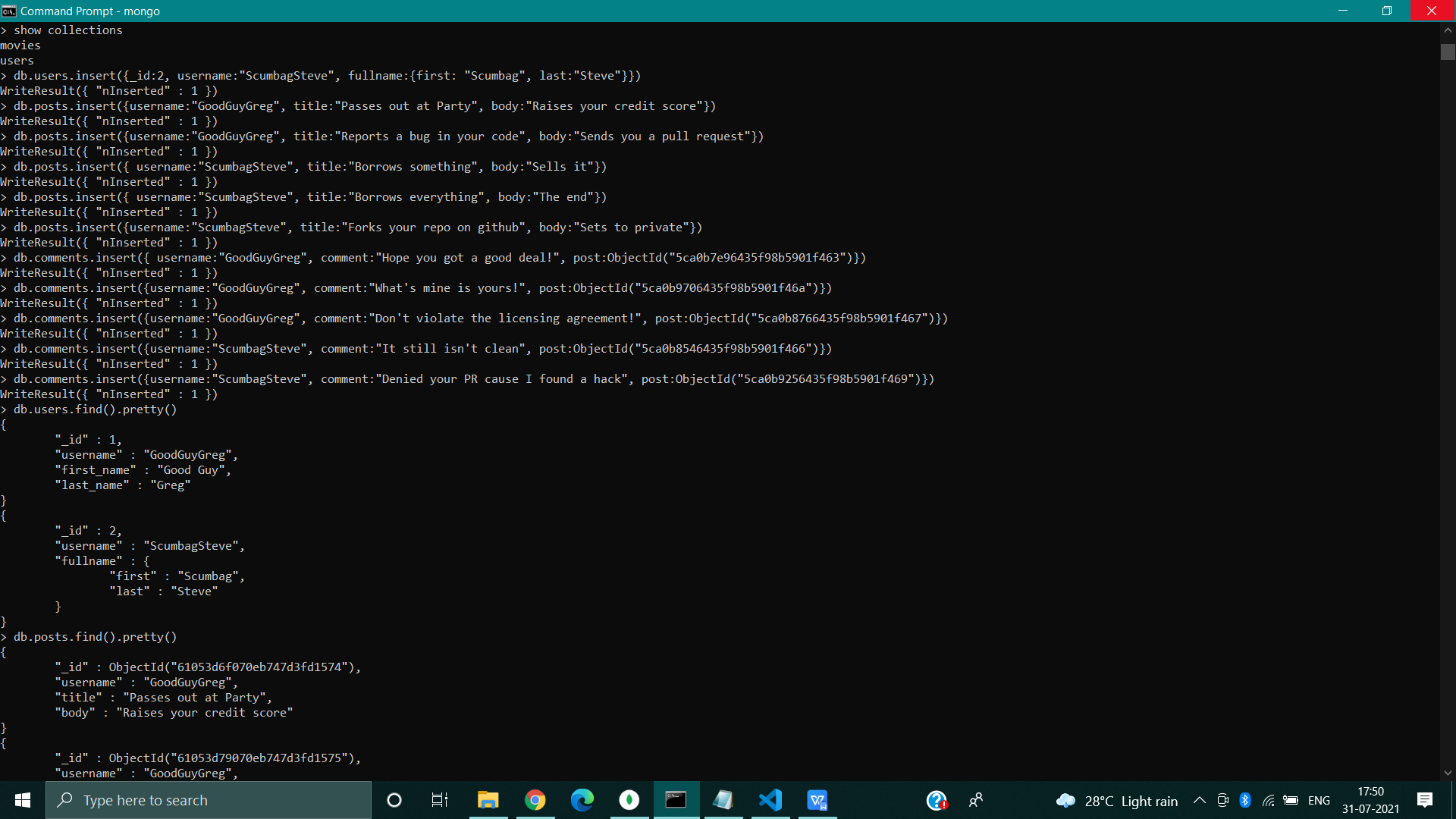
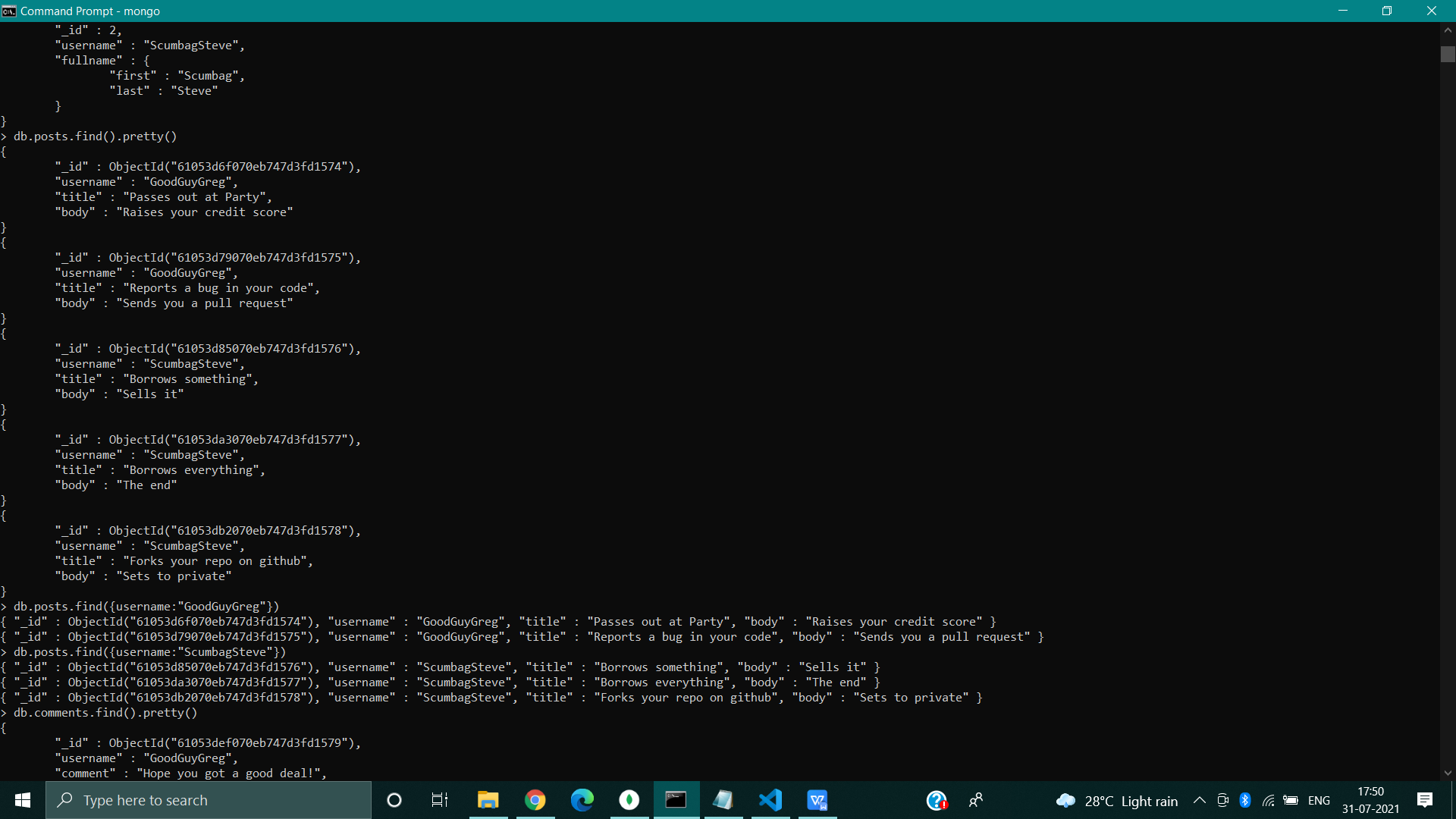
## Mongodb assignments 1

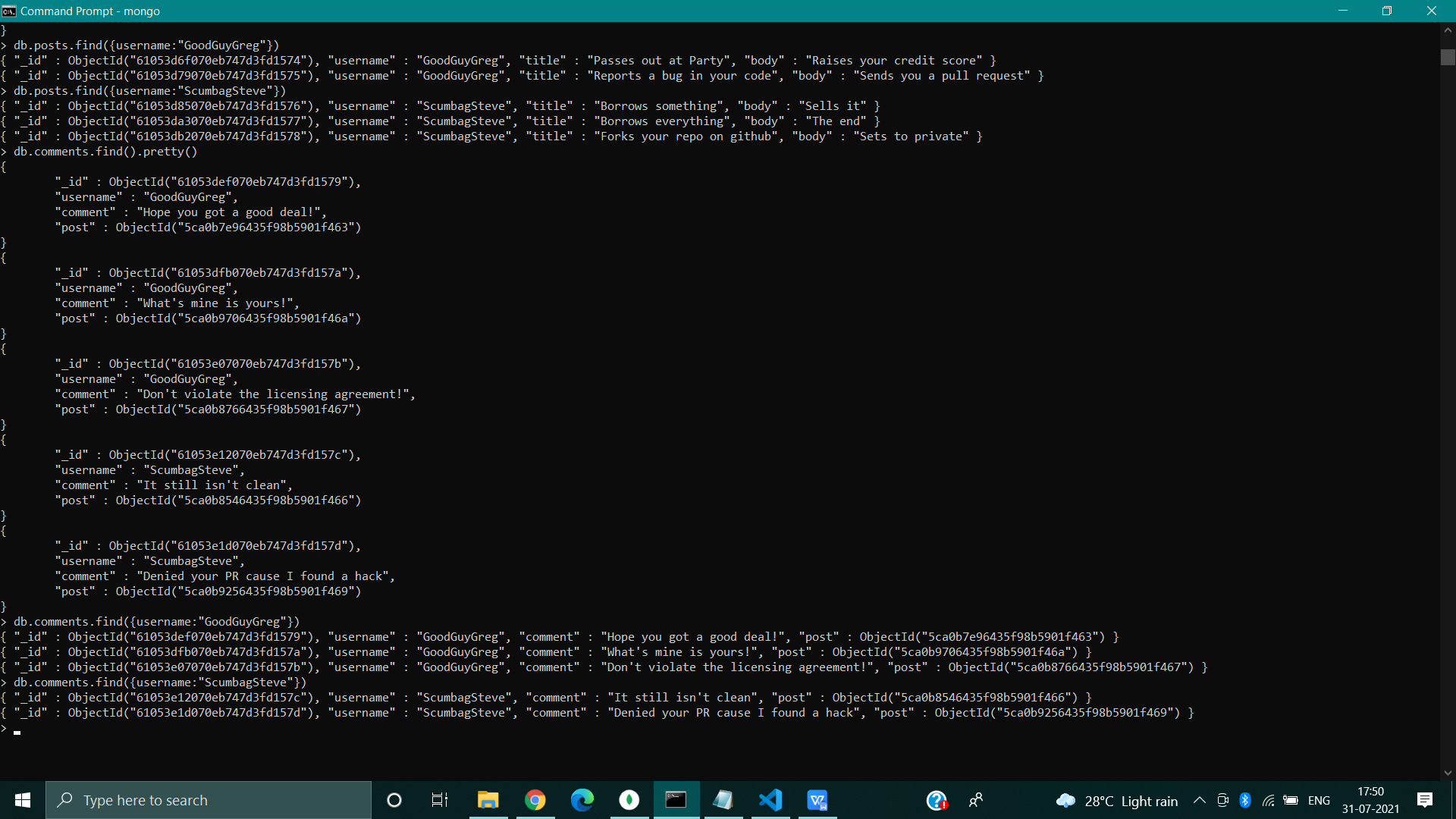
**Tanaya Jadhav**

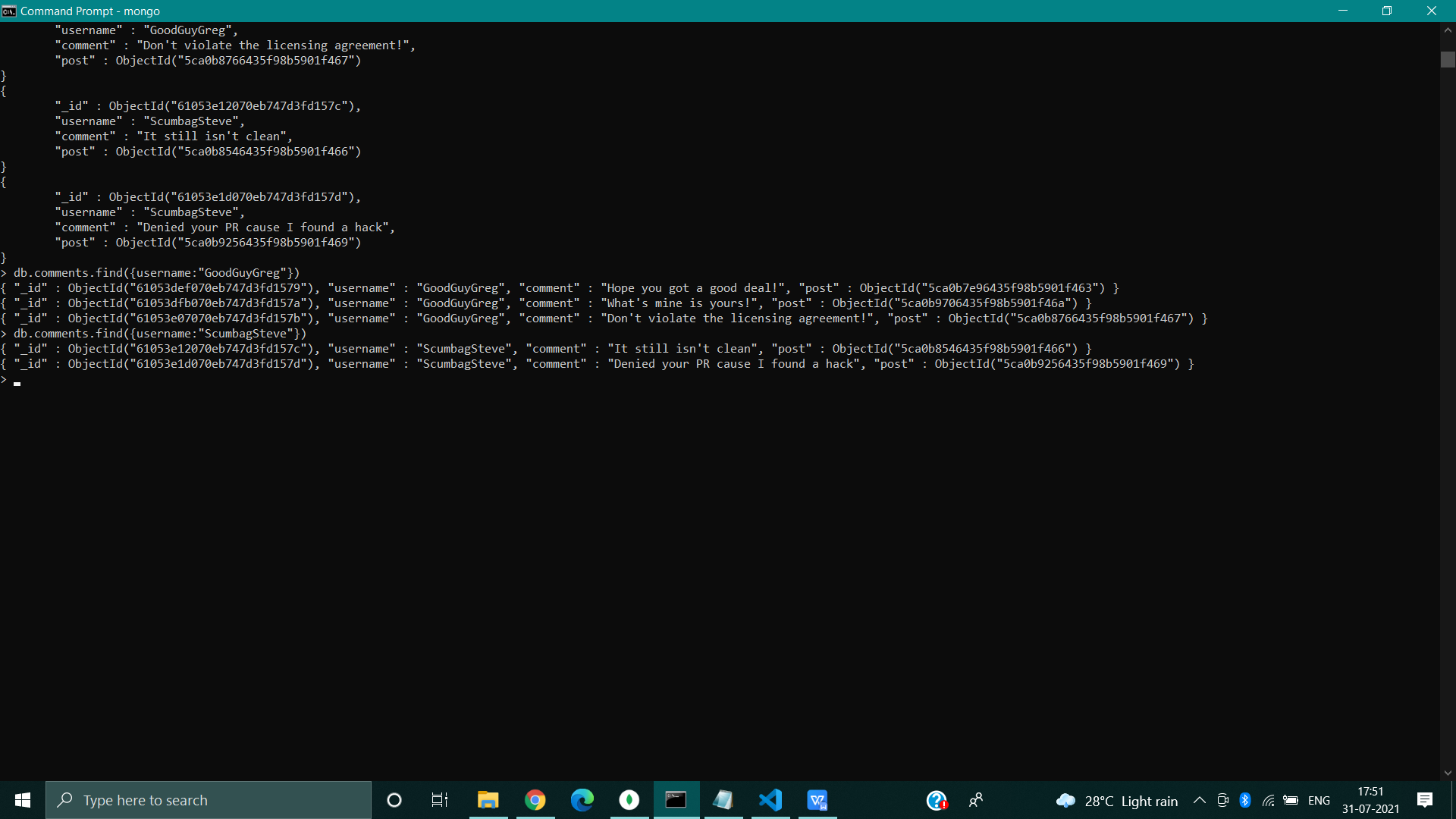
(c) Microsoft Corporation. All rights reserved.











C:\Users\Tanaya>mongo

MongoDB shell version v4.4.7

connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb

Implicit session: session { "id" : UUID("ac4f54f3-e8d6-45df-ba09-56d435d46953") }

MongoDB server version: 4.4.7

---

The server generated these startup warnings when booting:

2021-07-30T18:25:59.485+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted

---

---

Enable MongoDB's free cloud-based monitoring service, which will then receive and display

metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you

and anyone you share the URL with. MongoDB may use this information to make product

improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()

To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

---

> show dbs

admin 0.000GB

config 0.000GB

local 0.000GB

mongo\_practice 0.000GB

test 0.000GB

> use mongo\_practice

switched to db mongo\_practice

> show collection

uncaught exception: Error: don't know how to show [collection] :

shellHelper.show@src/mongo/shell/utils.js:1191:11

shellHelper@src/mongo/shell/utils.js:819:15

@(shellhelp2):1:1

> show collections

movies

> db.movies.find()

{ "\_id" : ObjectId("6105384083684f366d225eb5"), "title" : "Fight Club", "writer" : "Chuck Palahniuk", "year" : "1999", "actors" : [ "Brad Pitt", "Edward Norton" ] }

{ "\_id" : ObjectId("610538fb83684f366d225eb7"), "title" : "Pulp Fiction", "writer" : "Quentin Tarantino", "year" : "2009", "actors" : [ "John Travolta", "Uma Thurman" ] }

{ "\_id" : ObjectId("61053a2c83684f366d225eb9"), "title" : "Inglorious Basterds", "writer" : "Quentin Tarantino", "year" : "2009", "actors" : [ "Brad Pitt", "Diane Kruger", "Eli Roth" ] }

{ "\_id" : ObjectId("61053a4183684f366d225eba"), "title" : "The Hobbit: An unexpected Journey", "writer" : "J.R.R. Tolkein", "year" : "2012", "franchise" : "The Hobbit" }

{ "\_id" : ObjectId("61053a6483684f366d225ebb"), "title" : "The Hobbit: The Desolation of Smaug", "writer" : "J.R.R Tolkien", "year" : "2013", "franchise" : "The Hobbit" }

{ "\_id" : ObjectId("61053aa483684f366d225ebd"), "title" : "The Hobbit: The Battle of the Five Armies", "writer" : "J.R.R Tolkien", "year" : "2002", "franchise" : "The Hobbit", "synopsis" : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness." }

> db.movies.find({writer:"Quentin Tarantino"})

{ "\_id" : ObjectId("610538fb83684f366d225eb7"), "title" : "Pulp Fiction", "writer" : "Quentin Tarantino", "year" : "2009", "actors" : [ "John Travolta", "Uma Thurman" ] }

{ "\_id" : ObjectId("61053a2c83684f366d225eb9"), "title" : "Inglorious Basterds", "writer" : "Quentin Tarantino", "year" : "2009", "actors" : [ "Brad Pitt", "Diane Kruger", "Eli Roth" ] }

> db.movies.find({actors:"Brad Pitt"})

{ "\_id" : ObjectId("6105384083684f366d225eb5"), "title" : "Fight Club", "writer" : "Chuck Palahniuk", "year" : "1999", "actors" : [ "Brad Pitt", "Edward Norton" ] }

{ "\_id" : ObjectId("61053a2c83684f366d225eb9"), "title" : "Inglorious Basterds", "writer" : "Quentin Tarantino", "year" : "2009", "actors" : [ "Brad Pitt", "Diane Kruger", "Eli Roth" ] }

> db.movies.find({franchise:"The Hobbit"})

{ "\_id" : ObjectId("61053a4183684f366d225eba"), "title" : "The Hobbit: An unexpected Journey", "writer" : "J.R.R. Tolkein", "year" : "2012", "franchise" : "The Hobbit" }

{ "\_id" : ObjectId("61053a6483684f366d225ebb"), "title" : "The Hobbit: The Desolation of Smaug", "writer" : "J.R.R Tolkien", "year" : "2013", "franchise" : "The Hobbit" }

{ "\_id" : ObjectId("61053aa483684f366d225ebd"), "title" : "The Hobbit: The Battle of the Five Armies", "writer" : "J.R.R Tolkien", "year" : "2002", "franchise" : "The Hobbit", "synopsis" : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness." }

> db.movies.find({year:{$gt:"1990", $lt:"2000"}})

{ "\_id" : ObjectId("6105384083684f366d225eb5"), "title" : "Fight Club", "writer" : "Chuck Palahniuk", "year" : "1999", "actors" : [ "Brad Pitt", "Edward Norton" ] }

> db.movies.find({$or:[{year:{$gt:"2010"}},{year: {$lt:"2000"}}]})

{ "\_id" : ObjectId("6105384083684f366d225eb5"), "title" : "Fight Club", "writer" : "Chuck Palahniuk", "year" : "1999", "actors" : [ "Brad Pitt", "Edward Norton" ] }

{ "\_id" : ObjectId("61053a4183684f366d225eba"), "title" : "The Hobbit: An unexpected Journey", "writer" : "J.R.R. Tolkein", "year" : "2012", "franchise" : "The Hobbit" }

{ "\_id" : ObjectId("61053a6483684f366d225ebb"), "title" : "The Hobbit: The Desolation of Smaug", "writer" : "J.R.R Tolkien", "year" : "2013", "franchise" : "The Hobbit" }

> db.movies.update({\_id:ObjectId("5c9f98e5e5c2dfe9b3729bfe")}, {$set:{synopsis:"A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})

WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })

> db.movies.update({\_id:ObjectId("$oid":"61053a2c83684f366d225eb9")}, {$set:{synopsis:"A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})

uncaught exception: SyntaxError: missing ) after argument list :

@(shell):1:37

> db.movies.update({\_id:ObjectId("$oid":"61053a2c83684f366d225eb9"))}, {$set:{synopsis:"A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."}})

uncaught exception: SyntaxError: missing ) after argument list :

@(shell):1:37

> db.users.insert({\_id:1,username:"GoodGuyGreg", first\_name:"Good Guy", last\_name:"Greg"})

WriteResult({ "nInserted" : 1 })

> show collections

movies

users

> db.users.insert({\_id:2, username:"ScumbagSteve", fullname:{first: "Scumbag", last:"Steve"}})

WriteResult({ "nInserted" : 1 })

> db.posts.insert({username:"GoodGuyGreg", title:"Passes out at Party", body:"Raises your credit score"})

WriteResult({ "nInserted" : 1 })

> db.posts.insert({username:"GoodGuyGreg", title:"Reports a bug in your code", body:"Sends you a pull request"})

WriteResult({ "nInserted" : 1 })

> db.posts.insert({ username:"ScumbagSteve", title:"Borrows something", body:"Sells it"})

WriteResult({ "nInserted" : 1 })

> db.posts.insert({ username:"ScumbagSteve", title:"Borrows everything", body:"The end"})

WriteResult({ "nInserted" : 1 })

> db.posts.insert({username:"ScumbagSteve", title:"Forks your repo on github", body:"Sets to private"})

WriteResult({ "nInserted" : 1 })

> db.comments.insert({ username:"GoodGuyGreg", comment:"Hope you got a good deal!", post:ObjectId("5ca0b7e96435f98b5901f463")})

WriteResult({ "nInserted" : 1 })

> db.comments.insert({username:"GoodGuyGreg", comment:"What's mine is yours!", post:ObjectId("5ca0b9706435f98b5901f46a")})

WriteResult({ "nInserted" : 1 })

> db.comments.insert({username:"GoodGuyGreg", comment:"Don't violate the licensing agreement!", post:ObjectId("5ca0b8766435f98b5901f467")})

WriteResult({ "nInserted" : 1 })

> db.comments.insert({username:"ScumbagSteve", comment:"It still isn't clean", post:ObjectId("5ca0b8546435f98b5901f466")})

WriteResult({ "nInserted" : 1 })

> db.comments.insert({username:"ScumbagSteve", comment:"Denied your PR cause I found a hack", post:ObjectId("5ca0b9256435f98b5901f469")})

WriteResult({ "nInserted" : 1 })

> db.users.find().pretty()

{

"\_id" : 1,

"username" : "GoodGuyGreg",

"first\_name" : "Good Guy",

"last\_name" : "Greg"

}

{

"\_id" : 2,

"username" : "ScumbagSteve",

"fullname" : {

"first" : "Scumbag",

"last" : "Steve"

}

}

> db.posts.find().pretty()

{

"\_id" : ObjectId("61053d6f070eb747d3fd1574"),

"username" : "GoodGuyGreg",

"title" : "Passes out at Party",

"body" : "Raises your credit score"

}

{

"\_id" : ObjectId("61053d79070eb747d3fd1575"),

"username" : "GoodGuyGreg",

"title" : "Reports a bug in your code",

"body" : "Sends you a pull request"

}

{

"\_id" : ObjectId("61053d85070eb747d3fd1576"),

"username" : "ScumbagSteve",

"title" : "Borrows something",

"body" : "Sells it"

}

{

"\_id" : ObjectId("61053da3070eb747d3fd1577"),

"username" : "ScumbagSteve",

"title" : "Borrows everything",

"body" : "The end"

}

{

"\_id" : ObjectId("61053db2070eb747d3fd1578"),

"username" : "ScumbagSteve",

"title" : "Forks your repo on github",

"body" : "Sets to private"

}

> db.posts.find({username:"GoodGuyGreg"})

{ "\_id" : ObjectId("61053d6f070eb747d3fd1574"), "username" : "GoodGuyGreg", "title" : "Passes out at Party", "body" : "Raises your credit score" }

{ "\_id" : ObjectId("61053d79070eb747d3fd1575"), "username" : "GoodGuyGreg", "title" : "Reports a bug in your code", "body" : "Sends you a pull request" }

> db.posts.find({username:"ScumbagSteve"})

{ "\_id" : ObjectId("61053d85070eb747d3fd1576"), "username" : "ScumbagSteve", "title" : "Borrows something", "body" : "Sells it" }

{ "\_id" : ObjectId("61053da3070eb747d3fd1577"), "username" : "ScumbagSteve", "title" : "Borrows everything", "body" : "The end" }

{ "\_id" : ObjectId("61053db2070eb747d3fd1578"), "username" : "ScumbagSteve", "title" : "Forks your repo on github", "body" : "Sets to private" }

> db.comments.find().pretty()

{

"\_id" : ObjectId("61053def070eb747d3fd1579"),

"username" : "GoodGuyGreg",

"comment" : "Hope you got a good deal!",

"post" : ObjectId("5ca0b7e96435f98b5901f463")

}

{

"\_id" : ObjectId("61053dfb070eb747d3fd157a"),

"username" : "GoodGuyGreg",

"comment" : "What's mine is yours!",

"post" : ObjectId("5ca0b9706435f98b5901f46a")

}

{

"\_id" : ObjectId("61053e07070eb747d3fd157b"),

"username" : "GoodGuyGreg",

"comment" : "Don't violate the licensing agreement!",

"post" : ObjectId("5ca0b8766435f98b5901f467")

}

{

"\_id" : ObjectId("61053e12070eb747d3fd157c"),

"username" : "ScumbagSteve",

"comment" : "It still isn't clean",

"post" : ObjectId("5ca0b8546435f98b5901f466")

}

{

"\_id" : ObjectId("61053e1d070eb747d3fd157d"),

"username" : "ScumbagSteve",

"comment" : "Denied your PR cause I found a hack",

"post" : ObjectId("5ca0b9256435f98b5901f469")

}

> db.comments.find({username:"GoodGuyGreg"})

{ "\_id" : ObjectId("61053def070eb747d3fd1579"), "username" : "GoodGuyGreg", "comment" : "Hope you got a good deal!", "post" : ObjectId("5ca0b7e96435f98b5901f463") }

{ "\_id" : ObjectId("61053dfb070eb747d3fd157a"), "username" : "GoodGuyGreg", "comment" : "What's mine is yours!", "post" : ObjectId("5ca0b9706435f98b5901f46a") }

{ "\_id" : ObjectId("61053e07070eb747d3fd157b"), "username" : "GoodGuyGreg", "comment" : "Don't violate the licensing agreement!", "post" : ObjectId("5ca0b8766435f98b5901f467") }

> db.comments.find({username:"ScumbagSteve"})

{ "\_id" : ObjectId("61053e12070eb747d3fd157c"), "username" : "ScumbagSteve", "comment" : "It still isn't clean", "post" : ObjectId("5ca0b8546435f98b5901f466") }

{ "\_id" : ObjectId("61053e1d070eb747d3fd157d"), "username" : "ScumbagSteve", "comment" : "Denied your PR cause I found a hack", "post" : ObjectId("5ca0b9256435f98b5901f469") }

>DB methods:

db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]

db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor

db.auth(username, password)

db.cloneDatabase(fromhost) - will only function with MongoDB 4.0 and below

db.commandHelp(name) returns the help for the command

db.copyDatabase(fromdb, todb, fromhost) - will only function with MongoDB 4.0 and below

db.createCollection(name, {size: ..., capped: ..., max: ...})

db.createUser(userDocument)

db.createView(name, viewOn, [{$operator: {...}}, ...], {viewOptions})

db.currentOp() displays currently executing operations in the db

db.dropDatabase(writeConcern)

db.dropUser(username)

db.eval() - deprecated

db.fsyncLock() flush data to disk and lock server for backups

db.fsyncUnlock() unlocks server following a db.fsyncLock()

db.getCollection(cname) same as db['cname'] or db.cname

db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections

db.getCollectionNames()

db.getLastError() - just returns the err msg string

db.getLastErrorObj() - return full status object

db.getLogComponents()

db.getMongo() get the server connection object

db.getMongo().setSecondaryOk() allow queries on a replication secondary server

db.getName()

db.getProfilingLevel() - deprecated

db.getProfilingStatus() - returns if profiling is on and slow threshold

db.getReplicationInfo()

db.getSiblingDB(name) get the db at the same server as this one

db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set

db.hostInfo() get details about the server's host

db.isMaster() check replica primary status

db.hello() check replica primary status

db.killOp(opid) kills the current operation in the db

db.listCommands() lists all the db commands

db.loadServerScripts() loads all the scripts in db.system.js

db.logout()

db.printCollectionStats()

db.printReplicationInfo()

db.printShardingStatus()

db.printSecondaryReplicationInfo()

db.resetError()

db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into {cmdObj: 1}

db.serverStatus()

db.setLogLevel(level,<component>)

db.setProfilingLevel(level,slowms) 0=off 1=slow 2=all

db.setVerboseShell(flag) display extra information in shell output

db.setWriteConcern(<write concern doc>) - sets the write concern for writes to the db

db.shutdownServer()

db.stats()

db.unsetWriteConcern(<write concern doc>) - unsets the write concern for writes to the db

db.version() current version of the server

db.watch() - opens a change stream cursor for a database to report on all changes to its non-system collections.