Engineering Big-Data Systems
Assignment 1:

Name: Abhilash Kosaraju **NEUID :**001205393

1. READING ASSIGNMENT

Application of NoSQL Database in Web Crawling.
Web crawling is one of the most important applications of the Internet; the selection of database storage directly affects the performance of search engines
Compared to relational database, MongoDB supports schema-free, has great query performance with huge amount of data and provides easy horizontal scalability with low cost of hardware. It is more suitable for data storage in Web crawling.

Web crawling requests great capacity of storage space and low hardware costs. Its requirements of consistency and integrity can be reduced, but need high availability, scalability and performance. The relational databases store data in the form of a two-dimensional table with strictly row and column format, and emphasize the consistency and integrity of data, thus the performance in the distributed data management has low efficient, resulting in poor horizontal scalability and high hardware costs when increasing the data storage

Web crawling is a process that automatically obtains information from the Web pages through the link relationships between them and expands to the entire Web. This process is mainly done by the Web Crawler which usually consists of spider, controller and original page library

MongoDB is one of the most popular NoSQL database[12], whose main objective is to bridge the gap between key-value stores with high performance and scalability and traditional RDBMS with rich management, and take the advantages of both in one[

MongoDB uses BSON (Binary JSON) with loose data structure as the data format, and document-oriented storage. It provides auto-sharding to achieve mass data storage, and supports full indexes. With the powerful query language syntax similar to the object-oriented language, MongoDB can achieve the most function of single-table query in relational database. It also supports atomic in-place update and two replication mechanisms of Master/Slave and replica set [9] [11]. MongoDB both has the high performance and scalability of key-value data store and rich data processing functions of traditional relational database

Data Aggregation System - a system for information retrieval on demand over relational and non-relational distributed data sources.

The Data Aggregation System, a system for information retrieval and aggregation from heterogenous sources of relational and non-relational data.
these metadata aspects are stored into a specialised system, using a variety of different technologies.
in the data-taking era users regularly need to perform queries across multiple data services (for example, to look up the recorded luminosity, machine conditions and software configuration for a particular dataset)
DAS simplifies data look-up
The architecture of DAS was designed as a series of independent components

The web server handles user sessions, whether they originate from a browser or automated scripts
The cache server consists of a pool of worker threads which handle the DAS queries received from the web front-end.
The query is then analyzed to identify the set of data service APIs which are relevant. The cache is checked for items of data matching the query, or data that would form a superset of the query
The cache consists of one or more MongoDB shards a document store which natively stores the JSON.
The choice of a document store instead of a traditional relational database was driven by the need to be able to store and subsequently query individual members of the deeply-nested data structures that DAS handles, the structure of which are not known until run-time
The analytics server provides a facility for scheduling regular tasks. The rest of DAS operates only when triggered by user input, and although some clean-up is performed at run-time (such as discarded expired records) it is also necessary to have a facility for running asynchronous operations.
Analytics consists of a task scheduler and a pool of workers which execute the tasks.
DAS assumes that the data in the cache can be recreated from the original data sources at any time, and thus it can function with a limited amount of space by deleting old data, nor does it require backup of this space.
DAS queries are made in a custom text-based language.
Filters are commands that either eliminate complete data records or prune data members within a record.
Aggregators are functions which run over all the selected (and filtered, if applicable) records, summarizing their contents.
Almost any source of information can be incorporated in DAS, such as an SQL database cursor, a command called in a sub shell or an HTML page scraper, but in practice most data services used by DAS are APIs accessed over HTTP and returning data in JSON or XML format.
Data in DAS is principally stored in two separate collections; the raw and merged caches. Two types of data are stored in the raw cache; the documents returned by the respective APIs and documents describing the current status of queries. The latter serves as the central record for a

query, holding the original query, description of the APIs called, status of the processing and a common key with all the results documents.
The DAS analytics system is a daemon that schedules and executes small tasks that access the DAS document store. This can be used to perform prosaic maintenance tasks, such as cleaning out expired data or pruning still-valid data if space becomes limited, but the main function is to analyze the queries received by DAS to provide information to developers and to perform automated cache optimization.
To benchmark DAS performance, we used a 64-bit linux node with 8 cores (each 2.33GhZ) and 16GB of RAM, which we would expect to be typical of the hardware DAS would use in production. All DAS systems and MongoDB share this node.

Comparing NoSQL MongoDB to an SQL DB

There are not many studies that compare the performance of processing a modest amount of structured data in a NoSQL database with a traditional relational database. In this paper, we compare one of the NoSQL solutions, MongoDB, to the standard SQL relational database, SQL Server. We compare the performance, in terms of runtime, of these two databases for a modest-sized structured database.

NoSQL can help deal with data that is not structured. Data can be semi-structured, such that similar data objects can be grouped together, but the objects may have different characteristics. Schema information may also be mixed in with data values in semi-structured data, such as found in XML data. Unstructured data can be of any type and may have no format. This data cannot be represented by any type of schema, such as web pages in HTML.

SQL Server Express on a machine running an Intel i7 quad core 3.4GHz processor with 8GB of DDR3 Ram. Both databases were installed and stored their data on a SSD (Solid State Drive) for the fastest possible reads and writes. We wrote both test applications in C# using Visual Studio 2012 under the .Net 4.0 framework. tests consisted of four separate tests with 100 runs for each test. Figure 1 shows the object schema used for the tests. As shown in Figure 1, the data consisted of 3 tables/collections representing department, project and user. There are relationships between these data objects, as a user has a particular department and an array of projects. Similarly, a project is associated with a particular department, a manager that is a user and also an array of users of the project.

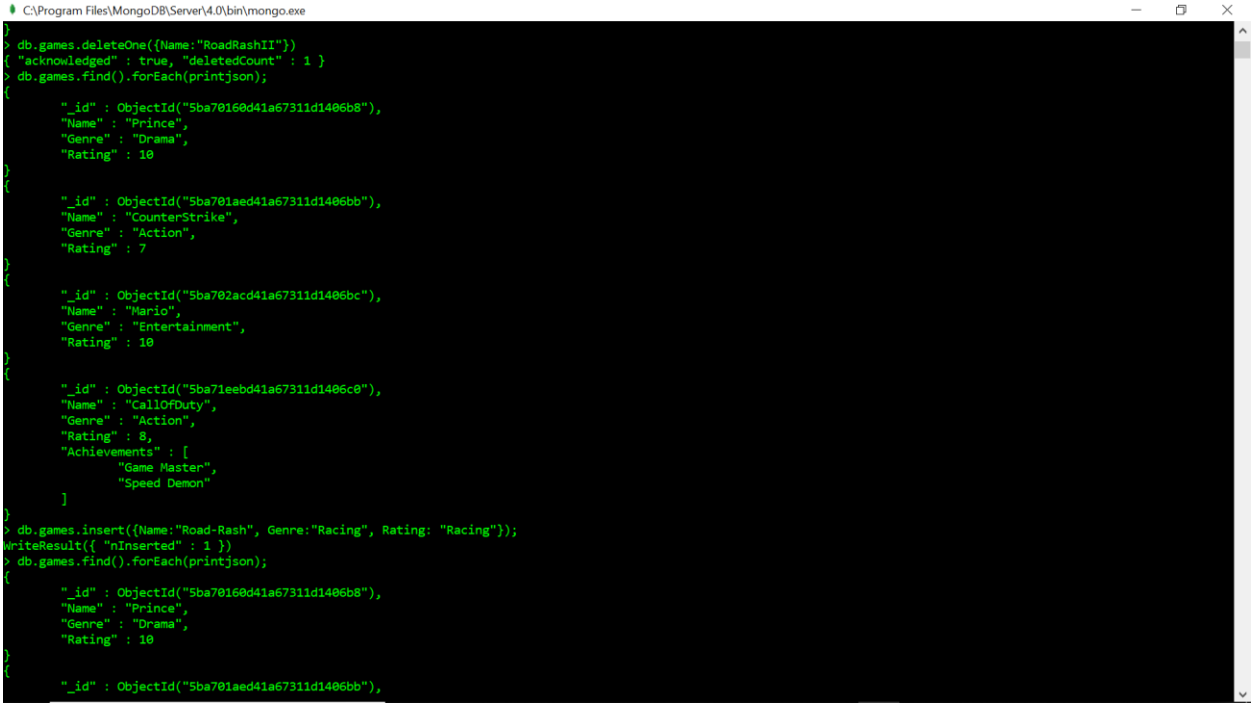
The four different simple selects we generated for the tests are as follows. The first select retrieved a department by its primary key. The second retrieved a department by a randomly chosen name. All departments had unique names, and the name was guaranteed to match one result in the department's collection. The third select retrieved a user by its primary key. The fourth select retrieved all users whose first name matched a randomly chosen first name; this randomly chosen name may not match any users in the collection.

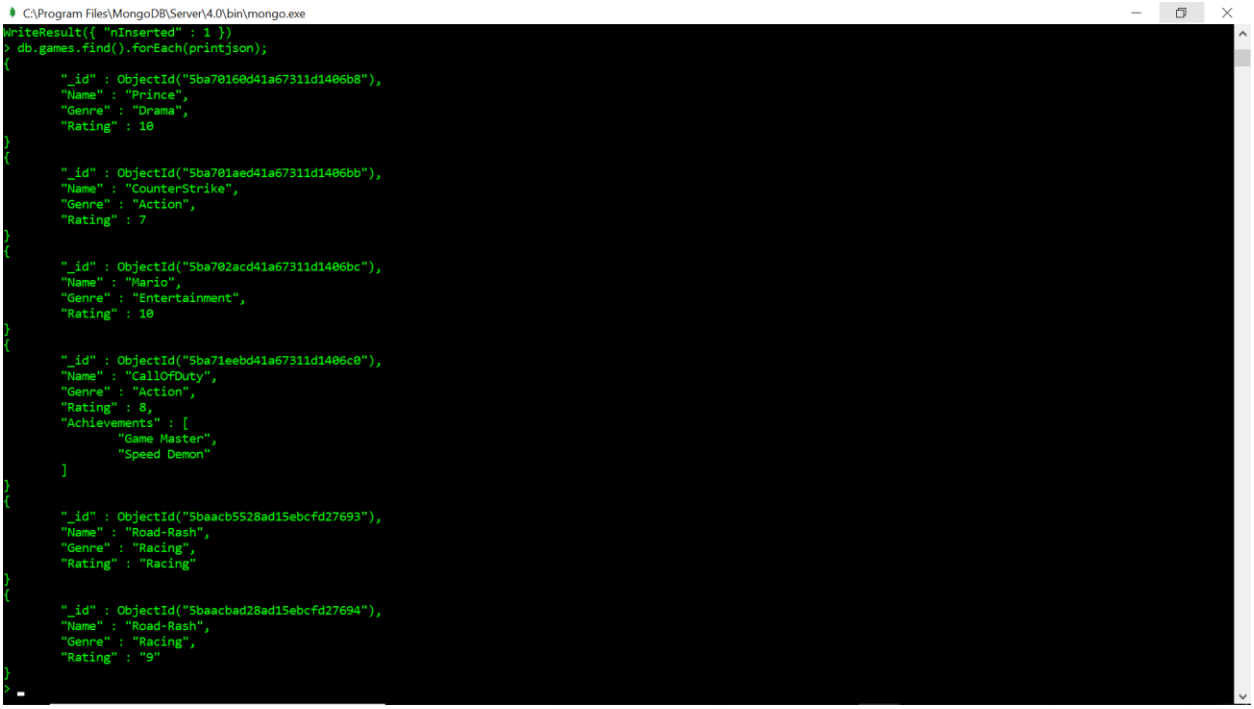
The three different complex selects used in the tests involved a join operation and are as follows. The first complex select retrieved all departments that contain one or more users that

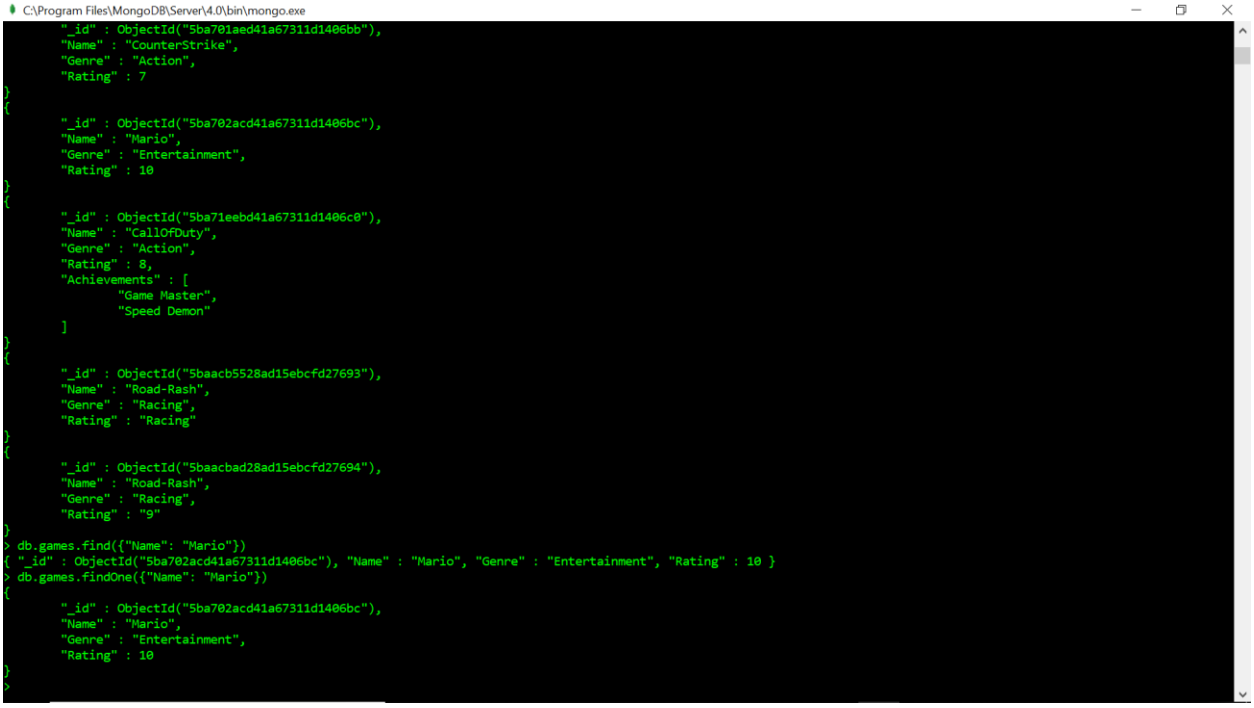
have a randomly chosen first name. Since first names were randomly assigned this may or may not return any matches. The second complex select retrieved all users who work on any of the three randomly chosen projects. Since users are randomly assigned to projects this query may also not return any valid matches. The final complex select returned the average age of the users who work on any of the three randomly chosen projects, and obviously, contains the aggregate function average(). This final query may return zero if no users were assigned to the projects.

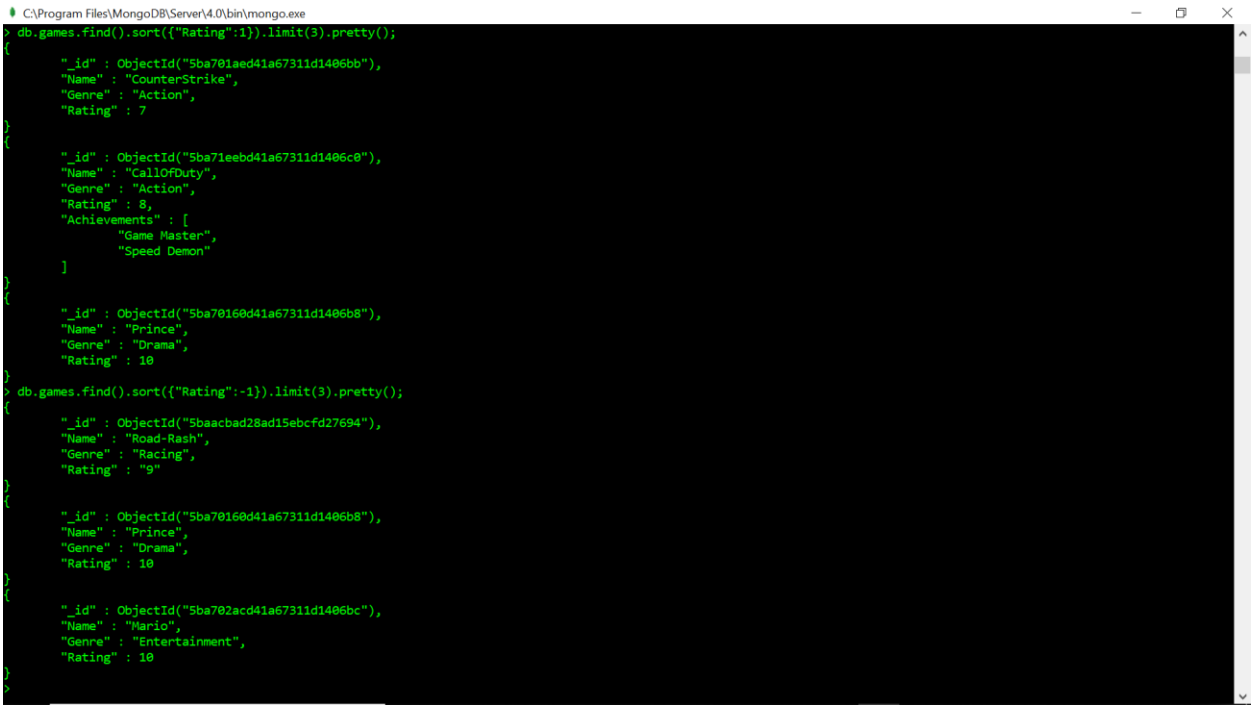
Overall, when comparing SQL to MongoDB, MongoDB has better runtime performance for inserts, updates and simple queries. SQL performed better when updating and querying non-key attributes, as well as for aggregate queries.
MongoDB could be a good solution for larger data sets in which the schema is constantly changing or in the case that queries performed will be less complex. Since MongoDB has no true schema defined and SQL requires a rigid schema definition.
For those users that have a strict schema defined and a modest amount of structured data we also found MongoDB to perform better than SQL in general

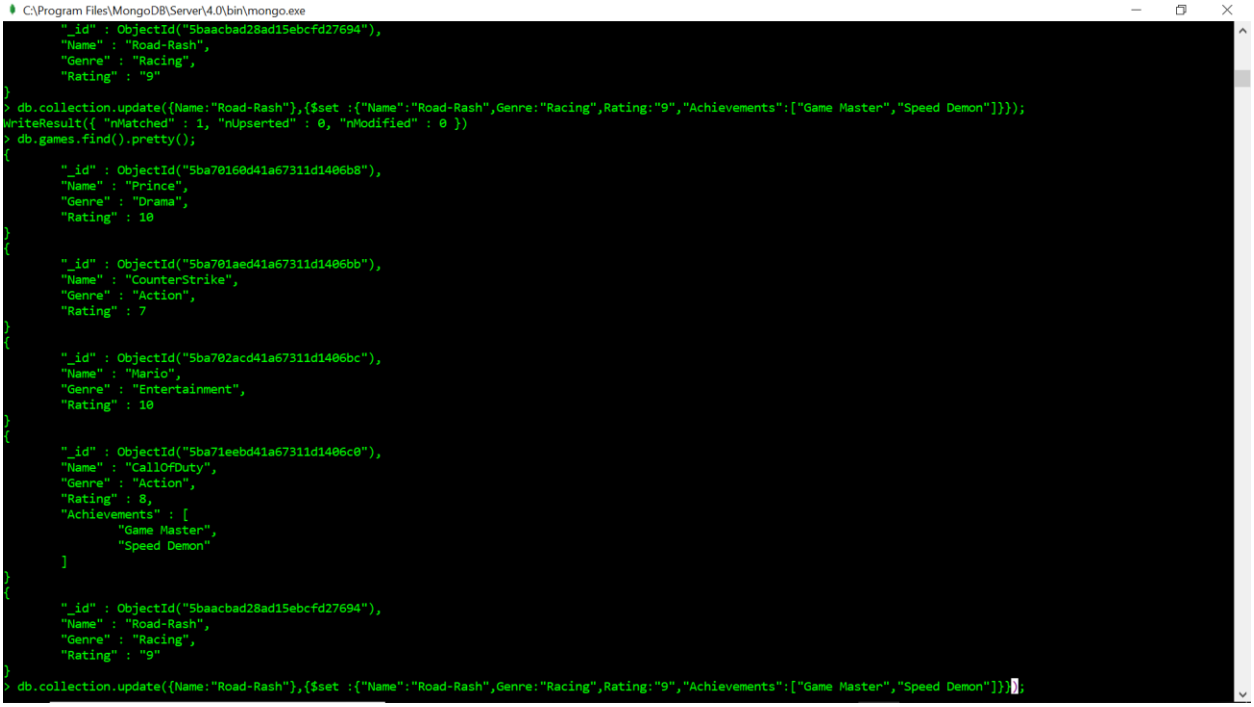
3- PROGRAMMING ASSIGNMENT

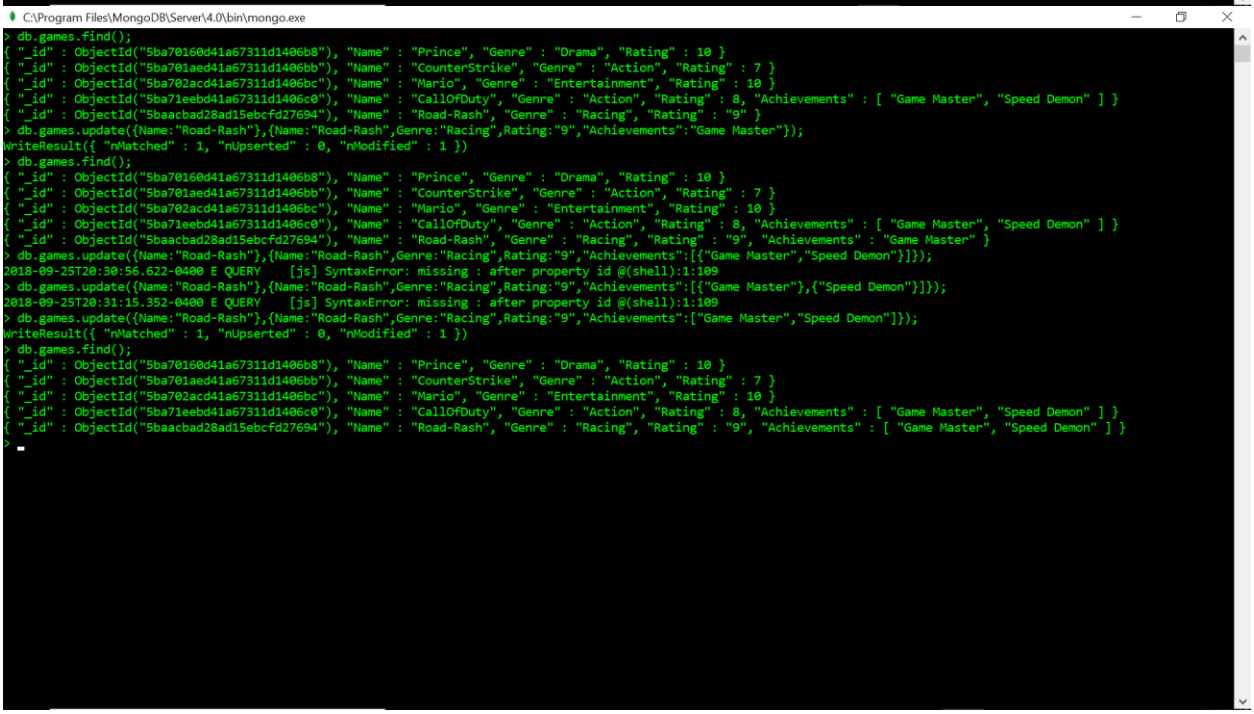

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
}
> db.games.deleteOne({Name:"RoadRashII"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.games.find().forEach(printjson);
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
{
"_id" : ObjectId("5ba701aed41a67311d1406bb"),
"Name" : "CounterStrike",
"Genre" : "Action",
"Rating" : 7
}
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
{
"_id" : ObjectId("5ba71eebd41a67311d1406c0"),
"Name" : "CallOfDuty",
"Genre" : "Action",
"Rating" : 8,
"Achievements" : [
"Game Master",
"Speed Demon"
]
}
> db.games.insert({Name:"Road-Rash", Genre:"Racing", Rating: "Racing"});
WriteResult({ "nInserted" : 1 })
> db.games.find().forEach(printjson);
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
{
"_id" : ObjectId("5ba701aed41a67311d1406bb"),

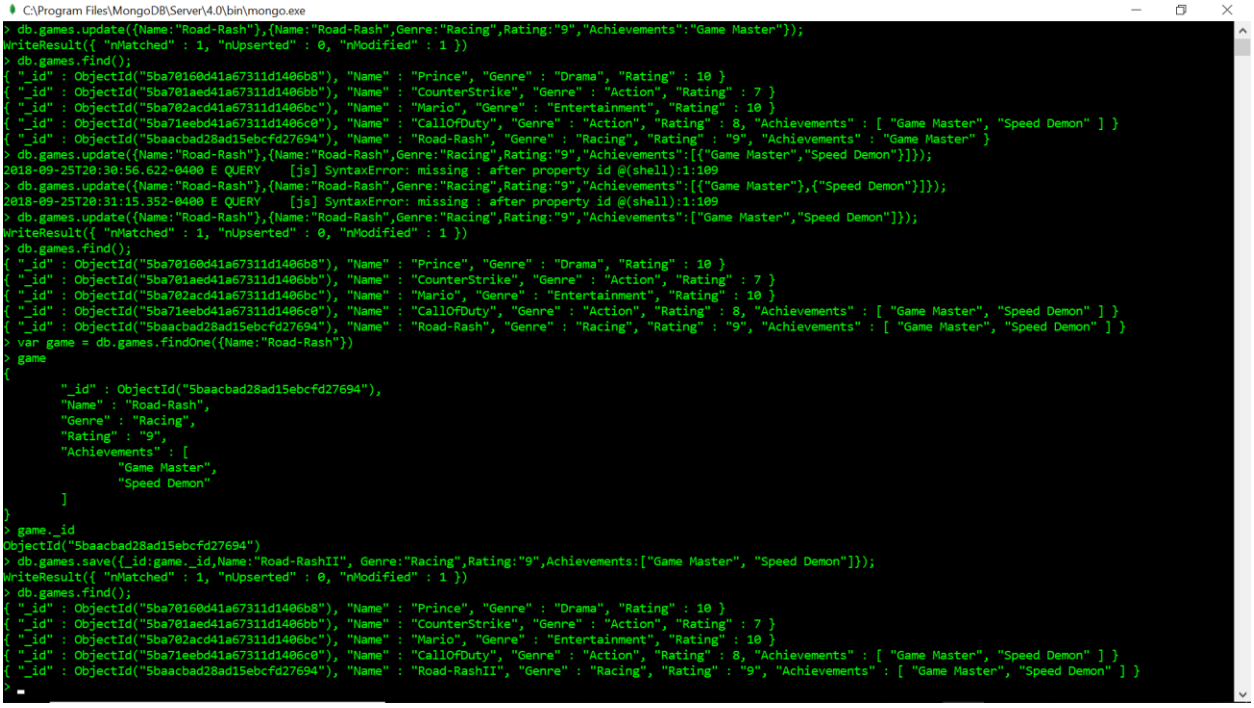

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
WriteResult({ "nInserted" : 1 })
> db.games.find().forEach(printjson);
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
{
"_id" : ObjectId("5ba701aed41a67311d1406bb"),
"Name" : "CounterStrike",
"Genre" : "Action",
"Rating" : 7
}
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
{
"_id" : ObjectId("5ba71eebd41a67311d1406c0"),
"Name" : "CallOfDuty",
"Genre" : "Action",
"Rating" : 8,
"Achievements" : [
"Game Master",
"Speed Demon"
]
}
{
"_id" : ObjectId("5baacb5528ad15ebcfd27693"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "Racing"
}
{
"_id" : ObjectId("5baacbad28ad15ebcfd27694"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "9"
}
>


C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
"_id" : ObjectId("5ba701aed41a67311d1406bb"),
"Name" : "CounterStrike",
"Genre" : "Action",
"Rating" : 7
}
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
{
"_id" : ObjectId("5ba71eebd41a67311d1406c0"),
"Name" : "CallOfDuty",
"Genre" : "Action",
"Rating" : 8,
"Achievements" : [
"Game Master",
"Speed Demon"
]
}
{
"_id" : ObjectId("5baacb5528ad15ebcfd27693"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "Racing"
}
{
"_id" : ObjectId("5baacbad28ad15ebcfd27694"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "9"
}
> db.games.find({"Name": "Mario"})
{ "_id" : ObjectId("5ba702acd41a67311d1406bc"), "Name" : "Mario", "Genre" : "Entertainment", "Rating" : 10 }
> db.games.findOne({"Name": "Mario"})
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
>

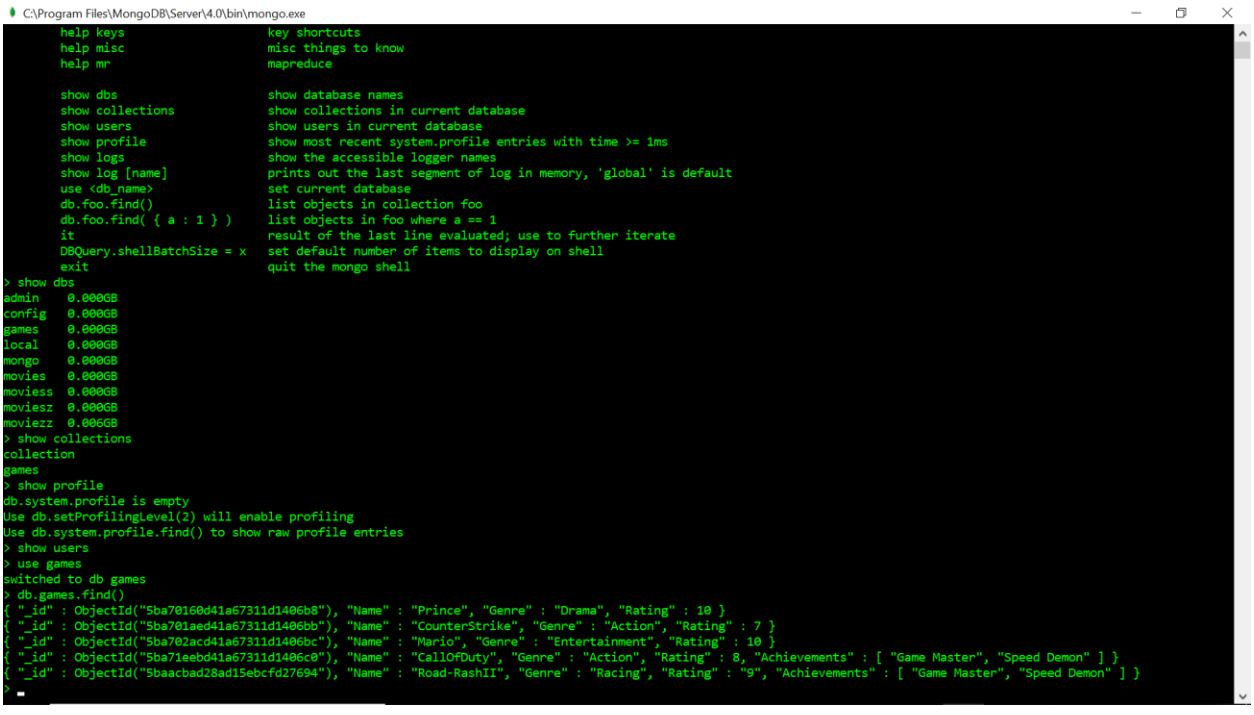

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.games.find().sort({"Rating":1}).limit(3).pretty();
{
"_id" : ObjectId("5ba701aed41a67311d1406bb"),
"Name" : "CounterStrike",
"Genre" : "Action",
"Rating" : 7
}
{
"_id" : ObjectId("5ba71eebd41a67311d1406c0"),
"Name" : "CallOfDuty",
"Genre" : "Action",
"Rating" : 8,
"Achievements" : [
"Game Master",
"Speed Demon"
]
}
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
> db.games.find().sort({"Rating":-1}).limit(3).pretty();
{
"_id" : ObjectId("5baacbad28ad15ebcfd27694"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "9"
}
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
>


C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
"_id" : ObjectId("5baacbad28ad15ebcfd27694"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "9"
}
> db.collection.update({Name:"Road-Rash"},{$set :{"Name":"Road-Rash",Genre:"Racing",Rating:"9","Achievements":["Game Master","Speed Demon"]}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.games.find().pretty();
{
"_id" : ObjectId("5ba70160d41a67311d1406b8"),
"Name" : "Prince",
"Genre" : "Drama",
"Rating" : 10
}
{
"_id" : ObjectId("5ba701aed41a67311d1406bb"),
"Name" : "CounterStrike",
"Genre" : "Action",
"Rating" : 7
}
{
"_id" : ObjectId("5ba702acd41a67311d1406bc"),
"Name" : "Mario",
"Genre" : "Entertainment",
"Rating" : 10
}
{
"_id" : ObjectId("5ba71eebd41a67311d1406c0"),
"Name" : "CallOfDuty",
"Genre" : "Action",
"Rating" : 8,
"Achievements" : [
"Game Master",
"Speed Demon"
]
}
{
"_id" : ObjectId("5baacbad28ad15ebcfd27694"),
"Name" : "Road-Rash",
"Genre" : "Racing",
"Rating" : "9"
}
> db.collection.update({Name:"Road-Rash"},{$set :{"Name":"Road-Rash",Genre:"Racing",Rating:"9","Achievements":["Game Master","Speed Demon"]}});


C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
> db.games.find();
{ "_id" : ObjectId("5ba70160d41a67311d1406b8"), "Name" : "Prince", "Genre" : "Drama", "Rating" : 10 }
{ "_id" : ObjectId("5ba701aed41a67311d1406bb"), "Name" : "CounterStrike", "Genre" : "Action", "Rating" : 7 }
{ "_id" : ObjectId("5ba702acd41a67311d1406bc"), "Name" : "Mario", "Genre" : "Entertainment", "Rating" : 10 }
{ "_id" : ObjectId("5ba71eebd41a67311d1406c0"), "Name" : "CallOfDuty", "Genre" : "Action", "Rating" : 8, "Achievements" : ["Game Master", "Speed Demon"] }
{ "_id" : ObjectId("5baacbad28ad15ebcfd27694"), "Name" : "Road-Rash", "Genre" : "Racing", "Rating" : "9" }
> db.games.update({Name:"Road-Rash"},{Name:"Road-Rash",Genre:"Racing",Rating:"9","Achievements":"Game Master"});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.games.find();
{ "_id" : ObjectId("5ba70160d41a67311d1406b8"), "Name" : "Prince", "Genre" : "Drama", "Rating" : 10 }
{ "_id" : ObjectId("5ba701aed41a67311d1406bb"), "Name" : "CounterStrike", "Genre" : "Action", "Rating" : 7 }
{ "_id" : ObjectId("5ba702acd41a67311d1406bc"), "Name" : "Mario", "Genre" : "Entertainment", "Rating" : 10 }
{ "_id" : ObjectId("5ba71eebd41a67311d1406c0"), "Name" : "CallOfDuty", "Genre" : "Action", "Rating" : 8, "Achievements" : ["Game Master", "Speed Demon"] }
{ "_id" : ObjectId("5baacbad28ad15ebcfd27694"), "Name" : "Road-Rash", "Genre" : "Racing", "Rating" : "9", "Achievements" : "Game Master" }
> db.games.update({Name:"Road-Rash"},{Name:"Road-Rash",Genre:"Racing",Rating:"9","Achievements":[{"Game Master","Speed Demon"}]});
2018-09-25T20:30:56.622-0400 E QUERY [js] SyntaxError: missing : after property id @(shell):1:109
> db.games.update({Name:"Road-Rash"},{Name:"Road-Rash",Genre:"Racing",Rating:"9","Achievements":[{"Game Master"},{"Speed Demon"}]});
2018-09-25T20:31:15.352-0400 E QUERY [js] SyntaxError: missing : after property id @(shell):1:109
> db.games.update({Name:"Road-Rash"},{Name:"Road-Rash",Genre:"Racing",Rating:"9","Achievements":["Game Master","Speed Demon"]});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.games.find();
{ "_id" : ObjectId("5ba70160d41a67311d1406b8"), "Name" : "Prince", "Genre" : "Drama", "Rating" : 10 }
{ "_id" : ObjectId("5ba701aed41a67311d1406bb"), "Name" : "CounterStrike", "Genre" : "Action", "Rating" : 7 }
{ "_id" : ObjectId("5ba702acd41a67311d1406bc"), "Name" : "Mario", "Genre" : "Entertainment", "Rating" : 10 }
{ "_id" : ObjectId("5ba71eebd41a67311d1406c0"), "Name" : "CallOfDuty", "Genre" : "Action", "Rating" : 8, "Achievements" : ["Game Master", "Speed Demon"] }
{ "_id" : ObjectId("5baacbad28ad15ebcfd27694"), "Name" : "Road-Rash", "Genre" : "Racing", "Rating" : "9", "Achievements" : ["Game Master", "Speed Demon"] }
>



6- PROGRAMMING ASSIGNMENT

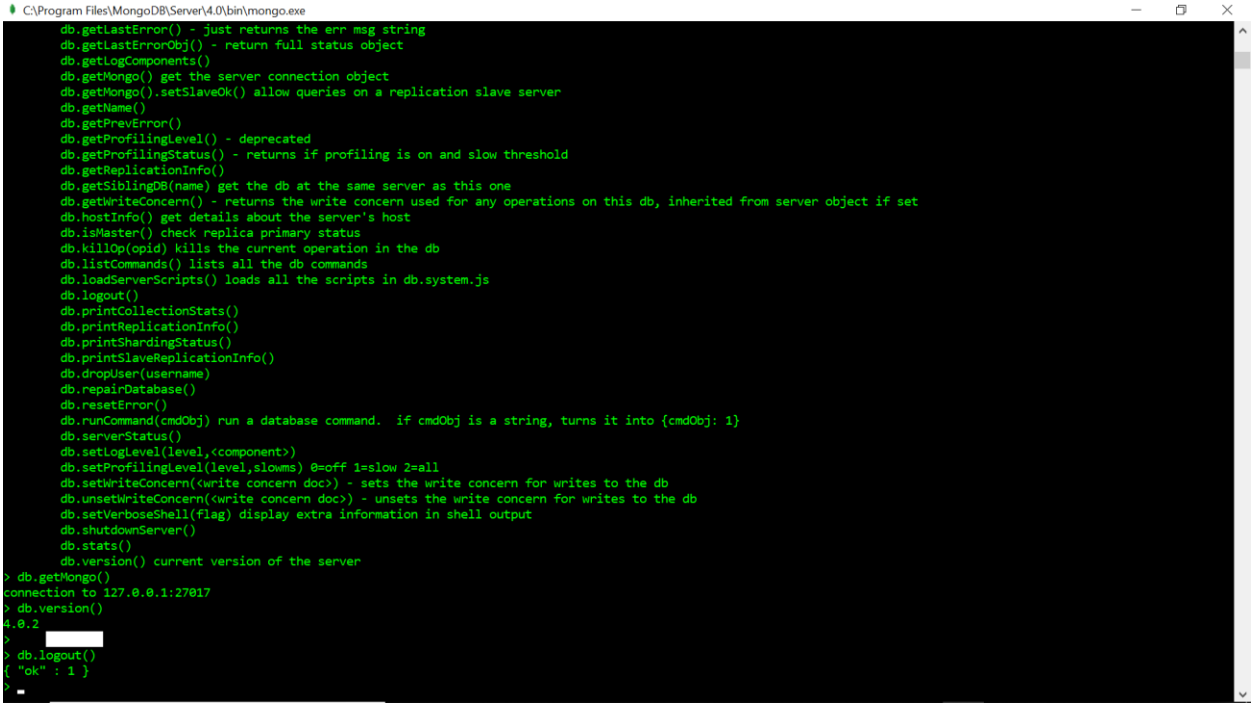


C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

```
        db.printShardingStatus()
        db.printSlaveReplicationInfo()
        db.dropUser(username)
        db.repairDatabase()
        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.setWriteConcern(<write concern doc>) - sets the write concern for writes to the db
        db.unsetWriteConcern(<write concern doc>) - unsets the write concern for writes to the db
        db.setVerboseShell(flag) display extra information in shell output
        db.shutdownServer()
        db.stats()
        db.version() current version of the server
> db.dropDatabase()
{ "dropped" : "moviezz", "ok" : 1 }
> show dbs
admin    0.000GB
config   0.000GB
games    0.000GB
local    0.000GB
mongo    0.000GB
movies   0.000GB
moviess  0.000GB
moviesz  0.000GB
> use moviesz
switched to db moviesz
> show collections
moviez
> db.moviez.count()
9125
> db.dropDatabase()
{ "dropped" : "moviesz", "ok" : 1 }
> show dbs
admin    0.000GB
config   0.000GB
games    0.000GB
local    0.000GB
mongo    0.000GB
movies   0.000GB
moviess  0.000GB
> _
```

C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe

```
        db.movies.validate( <full> ) - SLOW
        db.movies.getShardVersion() - only for use with sharding
        db.movies.getShardDistribution() - prints statistics about data distribution in the cluster
        db.movies.getSplitKeysForChunks( <maxChunkSize> ) - calculates split points over all chunks and returns splitter function
        db.movies.getWriteConcern() - returns the write concern used for any operations on this collection, inherited from server/db if set
        db.movies.setWriteConcern( <write concern doc> ) - sets the write concern for writes to the collection
        db.movies.unsetWriteConcern( <write concern doc> ) - unsets the write concern for writes to the collection
        db.movies.latencyStats() - display operation latency histograms for this collection
> db.movies.getDB();
movies
> db.movies.find().count();
9125
> db.movies.find().skip(10);
{ "_id" : ObjectId("5ba73b173dee0b665cfe309a"), "movieId" : 14, "title" : "Nixon (1995)", "genres" : "Drama" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe309b"), "movieId" : 15, "title" : "Cutthroat Island (1995)", "genres" : "Action|Adventure|Romance" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe309c"), "movieId" : 16, "title" : "Casino (1995)", "genres" : "Crime|Drama" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe309d"), "movieId" : 17, "title" : "Sense and Sensibility (1995)", "genres" : "Drama|Romance" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe309e"), "movieId" : 18, "title" : "Four Rooms (1995)", "genres" : "Comedy" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe309f"), "movieId" : 19, "title" : "Ace Ventura: When Nature Calls (1995)", "genres" : "Comedy" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a0"), "movieId" : 20, "title" : "Money Train (1995)", "genres" : "Action|Comedy|Crime|Drama|Thriller" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a1"), "movieId" : 21, "title" : "Get Shorty (1995)", "genres" : "Comedy|Crime|Thriller" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a2"), "movieId" : 1, "title" : "Toy Story (1995)", "genres" : "Adventure|Animation|Children|Comedy|Fantasy" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a3"), "movieId" : 2, "title" : "Jumanji (1995)", "genres" : "Adventure|Children|Fantasy" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a4"), "movieId" : 3, "title" : "Grumpier Old Men (1995)", "genres" : "Comedy|Romance" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a5"), "movieId" : 22, "title" : "Copycat (1995)", "genres" : "Crime|Drama|Horror|Mystery|Thriller" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a6"), "movieId" : 23, "title" : "Assassins (1995)", "genres" : "Action|Crime|Thriller" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a7"), "movieId" : 24, "title" : "Powder (1995)", "genres" : "Drama|Sci-Fi" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a8"), "movieId" : 25, "title" : "Leaving Las Vegas (1995)", "genres" : "Drama|Romance" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30a9"), "movieId" : 26, "title" : "Othello (1995)", "genres" : "Drama" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30aa"), "movieId" : 30, "title" : "Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)", "genres" : "Crime|Drama" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30ab"), "movieId" : 31, "title" : "Dangerous Minds (1995)", "genres" : "Drama" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30ac"), "movieId" : 32, "title" : "Twelve Monkeys (a.k.a. 12 Monkeys) (1995)", "genres" : "Mystery|Sci-Fi|Thriller" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe30ad"), "movieId" : 34, "title" : "Babe (1995)", "genres" : "Children|Drama" }
Type "it" for more
> db.movies.find().limit(2);
{ "_id" : ObjectId("5ba73b173dee0b665cfe3090"), "movieId" : 4, "title" : "Waiting to Exhale (1995)", "genres" : "Comedy|Drama|Romance" }
{ "_id" : ObjectId("5ba73b173dee0b665cfe3091"), "movieId" : 5, "title" : "Father of the Bride Part II (1995)", "genres" : "Comedy" }
> db.movies.find().totalSize();
2018-09-25T20:53:19.257-0400 E QUERY    [js] TypeError: db.movies.find(...).totalSize is not a function :
@(shell):1:1
> db.movies.totalSize();
536576
> _
```


C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
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().setSlaveOk() allow queries on a replication slave server
db.getName()
db.getPrevError()
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.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.printSlaveReplicationInfo()
db.dropUser(username)
db.repairDatabase()
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.setWriteConcern(<write concern doc>) - sets the write concern for writes to the db
db.unsetWriteConcern(<write concern doc>) - unsets the write concern for writes to the db
db.setVerboseShell(flag) display extra information in shell output
db.shutdownServer()
db.stats()
db.version() current version of the server
> db.getMongo()
connection to 127.0.0.1:27017
> db.version()
4.0.2
>
> db.logout()
{ "ok" : 1 }
>