MongoDB Lab Assignments

MongoDB Exercise in mongo shell

Connect to a running mongo instance,use a database named **mongo_practice.**

Document all your queries in a javaScrpit file to use as a reference.

Insert Documents

1)Command to connect to server ->mongod

2)Command to connect to shell->mongo

3)To create database->use **mongo_practise**

4)Command to insert movies->

```
> use mongo_practise
switched to db mongo_practise
> db.createCollection("movies")
{ "ok" : 1 }
> db.movies.insert({"title":"Fight Club","Writer":"Chuck Palahniuko","year":1999,actors:["Brad","Pitt","Edward","Norton"]})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"Pulp Fiction","Writer":"Auentin Tarantino","year":1994,actors:["John","Travolta","uma","Thurman"]})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"Inglorious Basterds","Writer":"Auentin Tarantino","year":2009,actors:["Brad","Pitt","Diane","Kruger","Eli","Roth"]})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"The Hobbit:An Unexpected Journey","Writer":"J.R.R. Tolkein","year":2012,"franchise":"The Hobbit"})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"The Hobbit:The Destination of Smaug","Writer":"J.R.R. Tolkein","year":2013,"franchise":"The Hobbit"})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"The Hobbit":"The Battle of the Five Armies","Writer":"J.R.R. Tolkein","year":2012,"franchise":"The Hobbit","synopsis":"Bilbo and company are forced to engage in a war against an array of com
bants and keep the lonely Mountain from falling into the hands of a rising darkness"})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"Pee Wee Herman's Big Adventure"})
WriteResult({ "nInserted" : 1 })
> db.movies.insert({"title":"Avatar"})
WriteResult({ "nInserted" : 1 })
```

Query/Find Documents

Query the **movies** collection to

1.get all documents

2.get all documents with writer set to “Questin Tarantino”

3. get all documents where actors include “Brad Pitt”

4. get all documents with franchise set to “The Hobbit”

5. get all movies released in the 90s

6.get all movies released before the year 2000 or after 2010

```
{ "_id" : ObjectId("62874bc8c77c970ae85c502c"), "title" : "Avatar" }
> db.movies.find().pretty()
{
        "_id" : ObjectId("62874868c77c970ae85c5025"),
        "title" : "Fight Club",
        "Writer" : "Chuck Palahniuko",
        "year" : 1999,
        "actors" : [
                "Brad",
                "Pitt",
                "Edward",
                "Norton"
        ]
}
{
        "_id" : ObjectId("6287494ec77c970ae85c5026"),
        "title" : "Pulp Fiction",
        "Writer" : "Auentin Tarantino",
        "year" : 1994,
        "actors" : [
                "John",
                "Travolta",
                "uma",
                "Thurman"
        ]
}
{
        "_id" : ObjectId("628749c4c77c970ae85c5027"),
        "title" : "Inglorious Basterds",
        "Writer" : "Auentin Tarantino",
        "year" : 2009,
        "actors" : [
                "Brad",
                "Pitt",
                "Diane",
                "Kruger",
                "Eli",
                "Roth"
        ]
}
{
        "_id" : ObjectId("62874a7bc77c970ae85c5028"),
        "title" : "The Hobbit:An Unexpected Journey",
        "Writer" : "J.R.R. Tolkein",
        "year" : 2012,
        "franchise" : "The Hobbit"
}
{
        "_id" : ObjectId("62874adbc77c970ae85c5029"),
```

```
}
{
        "_id" : ObjectId("62874a7bc77c970ae85c5028"),
        "title" : "The Hobbit:An Unexpected Journey",
        "Writer" : "J.R.R. Tolkein",
        "year" : 2012,
        "franchise" : "The Hobbit"
}
{
        "_id" : ObjectId("62874adbc77c970ae85c5029"),
        "title" : "The Hobbit:The Destination of Smaug",
        "Writer" : "J.R.R. Tolkein",
        "year" : 2013,
        "franchise" : "The Hobbit"
}
{
        "_id" : ObjectId("62874b7ac77c970ae85c502a"),
        "The Hobbit" : "The Battle of the Five Armies",
        "Writer" : "J.R.R. Tolkein",
        "year" : 2012,
        "franchise" : "The Hobbit",
        "synopsis" : "Bilbo and company are forced to engage in a war against an array of combants and keep the lonely Mountain from falling into the hands of a rising darkness"
}
{
        "_id" : ObjectId("62874badc77c970ae85c502b"),
        "title" : "Pee Wee Herman's Big Adventure"
}
{ "_id" : ObjectId("62874bc8c77c970ae85c502c"), "title" : "Avatar" }
>
```

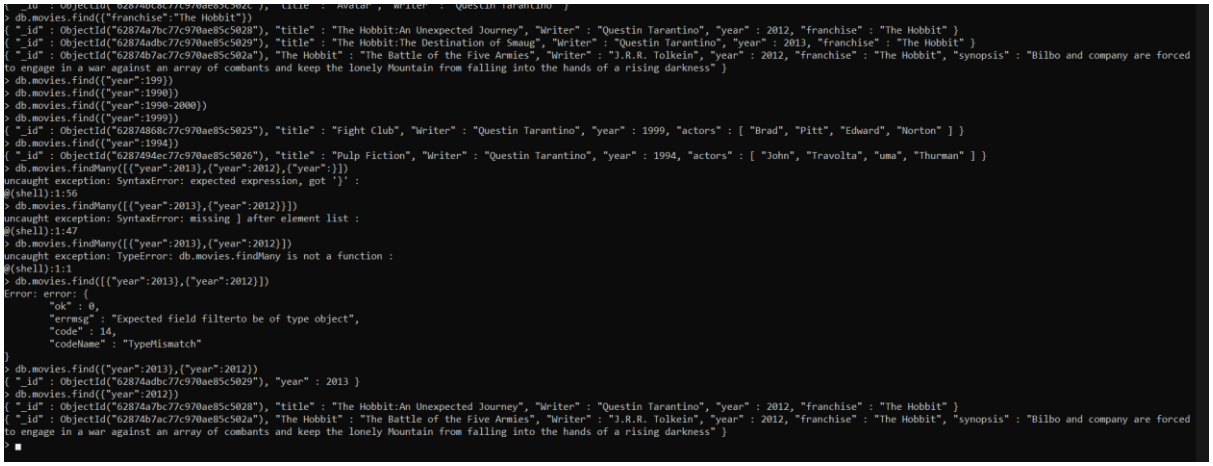
2.

```
> db.movies.find()
> use mongo_practise
switched to db mongo_practise
> db.movies.find()
{ "_id" : ObjectId("62874868c77c970ae85c5025"), "title" : "Fight Club", "Writer" : "Chuck Palahniuko", "year" : 1999, "actors" : [ "Brad", "Pitt", "Edward", "Norton" ] }
{ "_id" : ObjectId("6287494ec77c970ae85c5026"), "title" : "Pulp Fiction", "Writer" : "Auentin Tarantino", "year" : 1994, "actors" : [ "John", "Travolta", "uma", "Thurman" ] }
{ "_id" : ObjectId("628749c4c77c970ae85c5027"), "title" : "Inglorious Basterds", "Writer" : "Auentin Tarantino", "year" : 2009, "actors" : [ "Brad", "Pitt", "Diane", "Kruger", "Eli", "Roth" ] }
{ "_id" : ObjectId("62874a7bc77c970ae85c5028"), "title" : "The Hobbit:An Unexpected Journey", "Writer" : "J.R.R. Tolkein", "year" : 2012, "franchise" : "The Hobbit" }
{ "_id" : ObjectId("62874adbc77c970ae85c5029"), "title" : "The Hobbit:The Destination of Smaug", "Writer" : "J.R.R. Tolkein", "year" : 2013, "franchise" : "The Hobbit" }
{ "_id" : ObjectId("62874b7ac77c970ae85c502a"), "The Hobbit" : "The Battle of the Five Armies", "Writer" : "J.R.R. Tolkein", "year" : 2012, "franchise" : "The Hobbit", "synopsis" : "Bilbo and company are forced
to engage in a war against an array of combants and keep the lonely Mountain from falling into the hands of a rising darkness" }
{ "_id" : ObjectId("62874badc77c970ae85c502b"), "title" : "Pee Wee Herman's Big Adventure" }
{ "_id" : ObjectId("62874bc8c77c970ae85c502c"), "title" : "Avatar" }
> db.movies.update({"title":"Fight Club"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"Pulp Fiction"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"Inglorious Basterds"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"The Hobbit:An Unexpected Journey"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"The Hobbit:The Destination of Smaug"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"The Battle of the Five Armies"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.movies.update({"title":"Pee Wee Herman's Big Adventure"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"Avatar"},{$set:{"Writer":"Questin Tarantino"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.find()
{ "_id" : ObjectId("62874868c77c970ae85c5025"), "title" : "Fight Club", "Writer" : "Questin Tarantino", "year" : 1999, "actors" : [ "Brad", "Pitt", "Edward", "Norton" ] }
{ "_id" : ObjectId("6287494ec77c970ae85c5026"), "title" : "Pulp Fiction", "Writer" : "Questin Tarantino", "year" : 1994, "actors" : [ "John", "Travolta", "uma", "Thurman" ] }
{ "_id" : ObjectId("628749c4c77c970ae85c5027"), "title" : "Inglorious Basterds", "Writer" : "Questin Tarantino", "year" : 2009, "actors" : [ "Brad", "Pitt", "Diane", "Kruger", "Eli", "Roth" ] }
{ "_id" : ObjectId("62874a7bc77c970ae85c5028"), "title" : "The Hobbit:An Unexpected Journey", "Writer" : "Questin Tarantino", "year" : 2012, "franchise" : "The Hobbit" }
{ "_id" : ObjectId("62874adbc77c970ae85c5029"), "title" : "The Hobbit:The Destination of Smaug", "Writer" : "Questin Tarantino", "year" : 2013, "franchise" : "The Hobbit" }
{ "_id" : ObjectId("62874b7ac77c970ae85c502a"), "The Hobbit" : "The Battle of the Five Armies", "Writer" : "J.R.R. Tolkein", "year" : 2012, "franchise" : "The Hobbit", "synopsis" : "Bilbo and company are forced
to engage in a war against an array of combants and keep the lonely Mountain from falling into the hands of a rising darkness" }
{ "_id" : ObjectId("62874badc77c970ae85c502b"), "title" : "Pee Wee Herman's Big Adventure", "Writer" : "Questin Tarantino" }
{ "_id" : ObjectId("62874bc8c77c970ae85c502c"), "title" : "Avatar", "Writer" : "Questin Tarantino" }
>
```

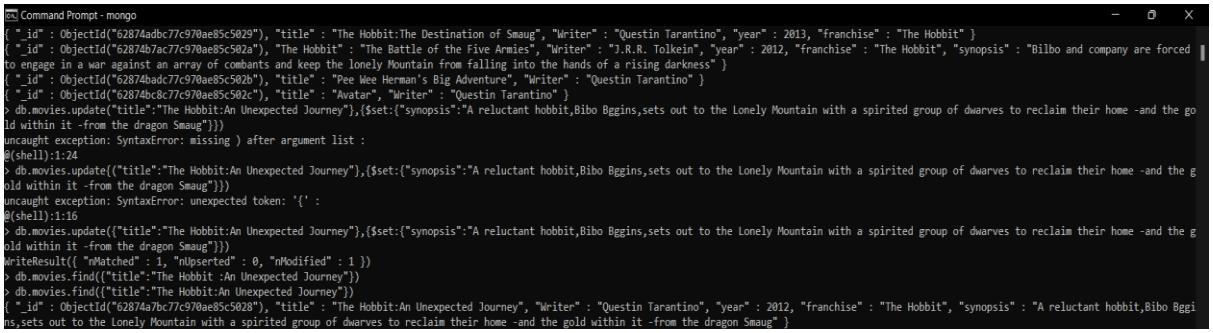
3.

```
> db.movies.find({"actors":"Brad"})
{ "_id" : ObjectId("62874868c77c970ae85c5025"), "title" : "Fight Club", "Writer" : "Questin Tarantino", "year" : 1999, "actors" : [ "Brad", "Pitt", "Edward", "Norton" ] }
{ "_id" : ObjectId("628749c4c77c970ae85c5027"), "title" : "Inglorious Basterds", "Writer" : "Questin Tarantino", "year" : 2009, "actors" : [ "Brad", "Pitt", "Diane", "Kruger", "Eli", "Roth" ] }
> db.movies.find({"actors":"Brad","Pitt"})
uncaught exception: SyntaxError: missing : after property id :
@(shell):1:38
> db.movies.find({"actors":"Pitt"})
{ "_id" : ObjectId("62874868c77c970ae85c5025"), "title" : "Fight Club", "Writer" : "Questin Tarantino", "year" : 1999, "actors" : [ "Brad", "Pitt", "Edward", "Norton" ] }
{ "_id" : ObjectId("628749c4c77c970ae85c5027"), "title" : "Inglorious Basterds", "Writer" : "Questin Tarantino", "year" : 2009, "actors" : [ "Brad", "Pitt", "Diane", "Kruger", "Eli", "Roth" ] }
>
```

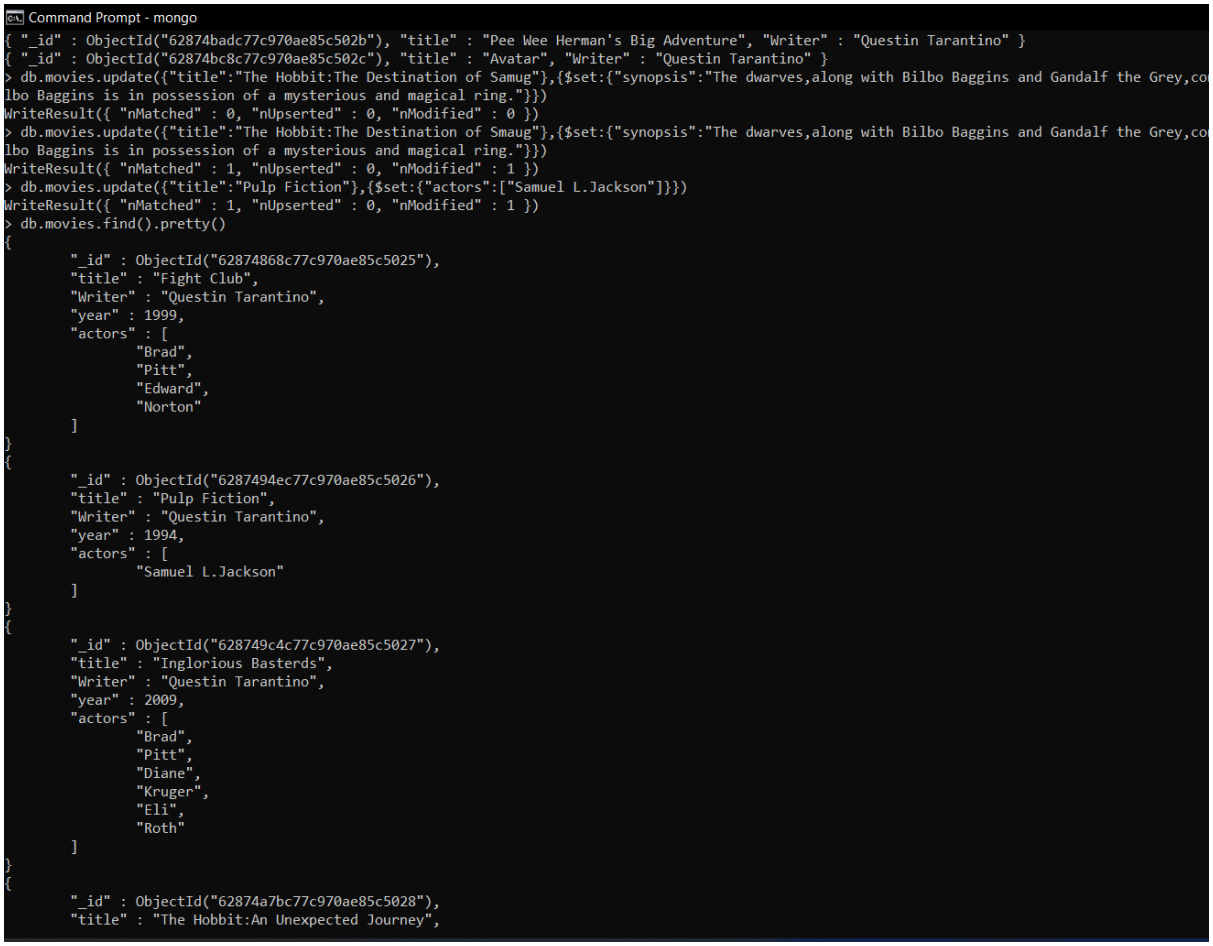
4,5,6



Update Documents



2


Command Prompt - mongo
{ "_id" : ObjectId("62874badc77c970ae85c502b"), "title" : "Pee Wee Herman's Big Adventure", "Writer" : "Questin Tarantino" }
{ "_id" : ObjectId("62874bc8c77c970ae85c502c"), "title" : "Avatar", "Writer" : "Questin Tarantino" }
> db.movies.update({"title":"The Hobbit:The Destination of Samug"},{$set:{"synopsis":"The dwarves,along with Bilbo Baggins and Gandalf the Grey,co
lbo Baggins is in possession of a mysterious and magical ring."}})
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
> db.movies.update({"title":"The Hobbit:The Destination of Smaug"},{$set:{"synopsis":"The dwarves,along with Bilbo Baggins and Gandalf the Grey,co
lbo Baggins is in possession of a mysterious and magical ring."}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.update({"title":"Pulp Fiction"},{$set:{"actors":["Samuel L.Jackson"]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.movies.find().pretty()
{
"_id" : ObjectId("62874868c77c970ae85c5025"),
"title" : "Fight Club",
"Writer" : "Questin Tarantino",
"year" : 1999,
"actors" : [
"Brad",
"Pitt",
"Edward",
"Norton"
]
}
{
"_id" : ObjectId("6287494ec77c970ae85c5026"),
"title" : "Pulp Fiction",
"Writer" : "Questin Tarantino",
"year" : 1994,
"actors" : [
"Samuel L.Jackson"
]
}
{
"_id" : ObjectId("628749c4c77c970ae85c5027"),
"title" : "Inglorious Basterds",
"Writer" : "Questin Tarantino",
"year" : 2009,
"actors" : [
"Brad",
"Pitt",
"Diane",
"Kruger",
"Eli",
"Roth"
]
}
{
"_id" : ObjectId("62874a7bc77c970ae85c5028"),
"title" : "The Hobbit:An Unexpected Journey",

3

Text Search

1.Find all movies that have a synopsis that contains the word “Bilbo”

2.Find all movies that have a synopsis that contains the word “Gandalf”

3.Find all movies that have a synopsis that contains the word “Bilbo” and not the Word “Gandalf”

4.Find all movies that have a synopsis that contains the word “dwarves” or “hobbit”

5.Find all movies that have a synopsis that contains the word “gold” and “dragon”

Relationships

Insert the following documents into a **users** collection

use Relationships

switched to db Relationships

> db.users.insert({username:"GoodGuyGreg",first_name:"Good Guy",last_name:"Greg",username:"ScumbagSteve",full_name:{first:"Scumbag",last:"Steve"}})

WriteResult({ "nInserted" : 1 })

```
db.posts.insert([{username:"GoodGuyGreg",title:"Passes out at party",body:"Wakes up early and cleans house"},{username:"GoodGuyGreg",title:"Steals your identity",body:"Raises your credit score"},{username:"GoodGuyGreg",title:"Rports a bug in your code",body:"Sends you a pull request"},{username:"ScumbagSteve",title:"Borrows something",body:"Sells it"},{username:"ScumbagSteve",title:"Borrows everything",body:"The end "},{username:"ScumbagSteve",title:"Forks your repo on gitHub",body:"Sets on private"}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 6,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

```
db.comments.insertMany([{username:"GoodGuyGreg",comment:"Hope yoi got a good deal!",post:ObjectId('628b6bb6a334ba17d563e739')},{username:"GoodGuyGreg",comment:"What's mine is yours",post:ObjectId('628b6bb6a334ba17d563e73a')},{username:"GoodGuyGreg",comment:"Don't violate the licensing agreement!",post:ObjectId('628b6bb6a334ba17d563e73b')},{username:"ScumbagSteve",comment:"it still isn't clean",post:ObjectId('628b6bb6a334ba17d563e736')},{username:"ScumbagSteve",comment:"Denied your PR cause I found a hack",post:ObjectId('628b6bb6a334ba17d563e738')}])
```

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("628b702ba334ba17d563e741"),
        ObjectId("628b702ba334ba17d563e742"),
        ObjectId("628b702ba334ba17d563e743"),
        ObjectId("628b702ba334ba17d563e744"),
        ObjectId("628b702ba334ba17d563e745")
    ]
}
```

Query related collections

1.find all users

2.find all posts

3.find all posts that was authored by “GoodGuyGreg”

4.find all posts that was authored by “ScumbagSteve”

5.find all comments

6.find all comments that was authored by “GoodGuyGreg”

7.find all comments that was authored by “ScumbagSteve”

8.find all comments belonging to the post “Rports a bug in your code”

1.

```
db.users.find().pretty()
{
	"_id" : ObjectId("628b6955a334ba17d563e734"),
	"username" : "ScumbagSteve",
	"first_name" : "Good Guy",
	"last_name" : "Greg",
	"full_name" : {
		"first" : "Scumbag",
		"last" : "Steve"
	}
}
```

2.

```
db.posts.find().pretty()
{
	"_id" : ObjectId("628b6bb6a334ba17d563e736"),
	"username" : "GoodGuyGreg",
	"title" : "Passes out at party",
	"body" : "Wakes up early and cleans house"
}
{
	"_id" : ObjectId("628b6bb6a334ba17d563e737"),
	"username" : "GoodGuyGreg",
	"title" : "Steals your identity",
	"body" : "Raises your credit score"
}
{
	"_id" : ObjectId("628b6bb6a334ba17d563e738"),
	"username" : "GoodGuyGreg",
	"title" : "Rports a bug in your code",
	"body" : "Sends you a pull request"
```

```
}
{
        "_id" : ObjectId("628b6bb6a334ba17d563e739"),
        "username" : "ScumbagSteve",
        "title" : "Borrows something",
        "body" : "Sells it"
}
{
        "_id" : ObjectId("628b6bb6a334ba17d563e73a"),
        "username" : "ScumbagSteve",
        "title" : "Borrows everything",
        "body" : "The end "
}
{
        "_id" : ObjectId("628b6bb6a334ba17d563e73b"),
        "username" : "ScumbagSteve",
        "title" : "Forks your repo on gitHub",
        "body" : "Sets on private"
}
```

3.

```
db.posts.find({username:"GoodGuyGreg"}).pretty()
{
        "_id" : ObjectId("628b6bb6a334ba17d563e736"),
        "username" : "GoodGuyGreg",
        "title" : "Passes out at party",
        "body" : "Wakes up early and cleans house"
}
{
        "_id" : ObjectId("628b6bb6a334ba17d563e737"),
        "username" : "GoodGuyGreg",
        "title" : "Steals your identity",
```

"body" : "Raises your credit score"

}

{

"_id" : ObjectId("628b6bb6a334ba17d563e738"),

"username" : "GoodGuyGreg",

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

"body" : "Sends you a pull request"

}

4.

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

{

"_id" : ObjectId("628b6bb6a334ba17d563e739"),

"username" : "ScumbagSteve",

"title" : "Borrows something",

"body" : "Sells it"

}

{

"_id" : ObjectId("628b6bb6a334ba17d563e73a"),

"username" : "ScumbagSteve",

"title" : "Borrows everything",

"body" : "The end "

}

{

"_id" : ObjectId("628b6bb6a334ba17d563e73b"),

"username" : "ScumbagSteve",

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

"body" : "Sets on private"

}

5.

```
db.comments.find().pretty()
{     "_id" : ObjectId("628b702ba334ba17d563e741"),
      "username" : "GoodGuyGreg",
      "comment" : "Hope yoi got a good deal!",
      "post" : ObjectId("628b6bb6a334ba17d563e739")
}
{
      "_id" : ObjectId("628b702ba334ba17d563e742"),
      "username" : "GoodGuyGreg",
      "comment" : "What's mine is yours",
      "post" : ObjectId("628b6bb6a334ba17d563e73a")
}
{
      "_id" : ObjectId("628b702ba334ba17d563e743"),
      "username" : "GoodGuyGreg",
      "comment" : "Don't violate the licensing agreement!",
      "post" : ObjectId("628b6bb6a334ba17d563e73b")
}
{
      "_id" : ObjectId("628b702ba334ba17d563e744"),
      "username" : "ScumbagSteve",
      "comment" : "it still isn't clean",
      "post" : ObjectId("628b6bb6a334ba17d563e736")
}
{
      "_id" : ObjectId("628b702ba334ba17d563e745"),
      "username" : "ScumbagSteve",
      "comment" : "Denied your PR cause I found a hack",
      "post" : ObjectId("628b6bb6a334ba17d563e738")
}
```

6.

```
db.comments.find({username:"GoodGuyGreg"}).pretty()
{
	"_id" : ObjectId("628b702ba334ba17d563e741"),
	"username" : "GoodGuyGreg",
	"comment" : "Hope yoi got a good deal!",
	"post" : ObjectId("628b6bb6a334ba17d563e739")
}
{
	"_id" : ObjectId("628b702ba334ba17d563e742"),
	"username" : "GoodGuyGreg",
	"comment" : "What's mine is yours",
	"post" : ObjectId("628b6bb6a334ba17d563e73a")
}
{
	"_id" : ObjectId("628b702ba334ba17d563e743"),
	"username" : "GoodGuyGreg",
	"comment" : "Don't violate the licensing agreement!",
	"post" : ObjectId("628b6bb6a334ba17d563e73b")
}
```

7.

```
db.comments.find({username:"ScumbagSteve"}).pretty()
{
        "_id" : ObjectId("628b702ba334ba17d563e744"),
        "username" : "ScumbagSteve",
        "comment" : "it still isn't clean",
        "post" : ObjectId("628b6bb6a334ba17d563e736")
}
{
        "_id" : ObjectId("628b702ba334ba17d563e745"),
        "username" : "ScumbagSteve",
        "comment" : "Denied your PR cause I found a hack",
        "post" : ObjectId("628b6bb6a334ba17d563e738")
}
```

8.

```
db.comments.find({post:ObjectId("628b6bb6a334ba17d563e738")}).pretty()
{
        "_id" : ObjectId("628b702ba334ba17d563e745"),
        "username" : "ScumbagSteve",
        "comment" : "Denied your PR cause I found a hack",
        "post" : ObjectId("628b6bb6a334ba17d563e738")
}
```

Delete Documents

1.Delete the movie “Pee Wee Hermans’s Big Adventure”

Command->**db.movies.deleteOne({"title":"Pee Wee Herman's Big Adventure"})**

2.Delete the movie “Avatar”

Command-> **db.movies.deleteOne({"title":"Avatar"})**

```
> db.movies.delete({"title":"Pee Herman's Big Adventure"})
uncaught exception: TypeError: db.movies.delete is not a function :
@(shell):1:1
> db.movies.deleteOne({"title":"Pee Herman's Big Adventure"})
{ "acknowledged" : true, "deletedCount" : 0 }
> db.movies.deleteOne({"title":"Pee Wee Herman's Big Adventure"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.movies.deleteOne({"title":"Avatar"})
{ "acknowledged" : true, "deletedCount" : 1 }
>
```

MongoDB -Aggregation Exercises

Import the zips.json file into your MongoDB. Database name is "population" and collection name is "zipcodes".

mongoimport --db population --collection zipcodes --file zips.json

Atlanta Population

1.use db.zipcodes.find() to filter results to only the results where city is ATLANTA and state is GA

db.zipcodes.find({$and:[{"city":"ATLANTA"},{"state":"GA"}]})

{ "_id" : "30303", "city" : "ATLANTA", "loc" : [-84.388846, 33.752504], "pop" : 1845, "state" : "GA" }

{ "_id" : "30305", "city" : "ATLANTA", "loc" : [-84.385145, 33.831963], "pop" : 19122, "state" : "GA" }

{ "_id" : "30306", "city" : "ATLANTA", "loc" : [-84.351418, 33.786027], "pop" : 20081, "state" : "GA" }

{ "_id" : "30307", "city" : "ATLANTA", "loc" : [-84.335957, 33.769138], "pop" : 16330, "state" : "GA" }

{ "_id" : "30308", "city" : "ATLANTA", "loc" : [-84.375744, 33.771839], "pop" : 8549, "state" : "GA" }

{ "_id" : "30309", "city" : "ATLANTA", "loc" : [-84.388338, 33.798407], "pop" : 14766, "state" : "GA" }

{ "_id" : "30310", "city" : "ATLANTA", "loc" : [-84.423173, 33.727849], "pop" : 34017, "state" : "GA" }

{ "_id" : "30311", "city" : "ATLANTA", "loc" : [-84.470219, 33.722957], "pop" : 34880, "state" : "GA" }

{ "_id" : "30312", "city" : "ATLANTA", "loc" : [-84.378125, 33.746749], "pop" : 17683, "state" : "GA" }

{ "_id" : "30313", "city" : "ATLANTA", "loc" : [-84.39352, 33.76825], "pop" : 8038, "state" : "GA" }

{ "_id" : "30314", "city" : "ATLANTA", "loc" : [-84.425546, 33.756103], "pop" : 26649, "state" : "GA" }

{ "_id" : "30315", "city" : "ATLANTA", "loc" : [-84.380771, 33.705062], "pop" : 41061, "state" : "GA" }

{ "_id" : "30316", "city" : "ATLANTA", "loc" : [-84.333913, 33.721686], "pop" : 34668, "state" : "GA" }

{ "_id" : "30317", "city" : "ATLANTA", "loc" : [-84.31685, 33.749788], "pop" : 16395, "state" : "GA" }

{ "_id" : "30318", "city" : "ATLANTA", "loc" : [-84.445432, 33.786454], "pop" : 53894, "state" : "GA" }

{ "_id" : "30319", "city" : "ATLANTA", "loc" : [-84.335091, 33.868728], "pop" : 32138, "state" : "GA" }

{ "_id" : "30324", "city" : "ATLANTA", "loc" : [-84.354867, 33.820609], "pop" : 15044, "state" : "GA" }

{ "_id" : "30326", "city" : "ATLANTA", "loc" : [-84.358232, 33.848168], "pop" : 125, "state" : "GA" }

{ "_id" : "30327", "city" : "ATLANTA", "loc" : [-84.419966, 33.862723], "pop" : 18467, "state" : "GA" }

{ "_id" : "30329", "city" : "ATLANTA", "loc" : [-84.321402, 33.823555], "pop" : 17013, "state" : "GA" }

Type "it" for more

db.zipcodes.find({$and:[{"city":"ATLANTA"},{"state":"GA"}]}).count()

31

2.use db.zipcodes.aggregate with $match to do the same as above.

> db.zipcodes.aggregate([{$match:{$and:[{"city":"ATLANTA"},{"state":"GA"}]}}])

{ "_id" : "30303", "city" : "ATLANTA", "loc" : [-84.388846, 33.752504], "pop" : 1845, "state" : "GA" }

{ "_id" : "30305", "city" : "ATLANTA", "loc" : [-84.385145, 33.831963], "pop" : 19122, "state" : "GA" }

{ "_id" : "30306", "city" : "ATLANTA", "loc" : [-84.351418, 33.786027], "pop" : 20081, "state" : "GA" }

{ "_id" : "30307", "city" : "ATLANTA", "loc" : [-84.335957, 33.769138], "pop" : 16330, "state" : "GA" }

{ "_id" : "30308", "city" : "ATLANTA", "loc" : [-84.375744, 33.771839], "pop" : 8549, "state" : "GA" }

{ "_id" : "30309", "city" : "ATLANTA", "loc" : [-84.388338, 33.798407], "pop" : 14766, "state" : "GA" }

{ "_id" : "30310", "city" : "ATLANTA", "loc" : [-84.423173, 33.727849], "pop" : 34017, "state" : "GA" }

{ "_id" : "30311", "city" : "ATLANTA", "loc" : [-84.470219, 33.722957], "pop" : 34880, "state" : "GA" }

{ "_id" : "30312", "city" : "ATLANTA", "loc" : [-84.378125, 33.746749], "pop" : 17683, "state" : "GA" }

{ "_id" : "30313", "city" : "ATLANTA", "loc" : [-84.39352, 33.76825], "pop" : 8038, "state" : "GA" }

{ "_id" : "30314", "city" : "ATLANTA", "loc" : [-84.425546, 33.756103], "pop" : 26649, "state" : "GA" }

{ "_id" : "30315", "city" : "ATLANTA", "loc" : [-84.380771, 33.705062], "pop" : 41061, "state" : "GA" }

{ "_id" : "30316", "city" : "ATLANTA", "loc" : [-84.333913, 33.721686], "pop" : 34668, "state" : "GA" }

{ "_id" : "30317", "city" : "ATLANTA", "loc" : [-84.31685, 33.749788], "pop" : 16395, "state" : "GA" }

{ "_id" : "30318", "city" : "ATLANTA", "loc" : [-84.445432, 33.786454], "pop" : 53894, "state" : "GA" }

{ "_id" : "30319", "city" : "ATLANTA", "loc" : [-84.335091, 33.868728], "pop" : 32138, "state" : "GA" }

{ "_id" : "30324", "city" : "ATLANTA", "loc" : [-84.354867, 33.820609], "pop" : 15044, "state" : "GA" }

{ "_id" : "30326", "city" : "ATLANTA", "loc" : [-84.358232, 33.848168], "pop" : 125, "state" : "GA" }

{ "_id" : "30327", "city" : "ATLANTA", "loc" : [-84.419966, 33.862723], "pop" : 18467, "state" : "GA" }

{ "_id" : "30329", "city" : "ATLANTA", "loc" : [-84.321402, 33.823555], "pop" : 17013, "state" : "GA" }

Type "it" for more

3.use $group to count the number of zip codes in Atlanta.

db.zipcodes.aggregate([{$match:{"city":"ATLANTA"}}, {$group:{_id:"$city",count:{$count:{}}}}])

{ "_id" : "ATLANTA", "count" : 41 }

4.use $group to find the total population in Atlanta.

db.zipcodes.aggregate([{$match:{"city":"ATLANTA"}}, {$group:{_id:"$city",total:{$sum:"$pop"}}}])

{ "_id" : "ATLANTA", "total" : 630046 }

>

==========

Populations By State

1.use aggregate to calculate the total population for each state

db.zipcodes.aggregate([{ $group: { _id: "$state", totalPop: { $sum: "$pop" } }}])

{ "_id" : "MI", "totalPop" : 9295297 }

{ "_id" : "TN", "totalPop" : 4876457 }

{ "_id" : "IN", "totalPop" : 5544136 }

{ "_id" : "MS", "totalPop" : 2573216 }

{ "_id" : "SD", "totalPop" : 695397 }

{ "_id" : "NM", "totalPop" : 1515069 }

{ "_id" : "UT", "totalPop" : 1722850 }

{ "_id" : "AZ", "totalPop" : 3665228 }

{ "_id" : "NV", "totalPop" : 1201833 }

{ "_id" : "DC", "totalPop" : 606900 }

{ "_id" : "WI", "totalPop" : 4891769 }

{ "_id" : "OK", "totalPop" : 3145585 }

{ "_id" : "AL", "totalPop" : 4040587 }

{ "_id" : "FL", "totalPop" : 12686644 }

{ "_id" : "RI", "totalPop" : 1003218 }

{ "_id" : "GA", "totalPop" : 6478216 }

{ "_id" : "AR", "totalPop" : 2350725 }

{ "_id" : "MO", "totalPop" : 5110648 }

{ "_id" : "CO", "totalPop" : 3293755 }

{ "_id" : "ND", "totalPop" : 638272 }

2.sort the results by population, highest first

db.zipcodes.aggregate([{ $group: { _id: "$state", totalPop: { $sum: "$pop" } }}, { $sort: { totalPop: -1 } }])

{ "_id" : "CA", "totalPop" : 29754890 }

{ "_id" : "NY", "totalPop" : 17990402 }

{ "_id" : "TX", "totalPop" : 16984601 }

{ "_id" : "FL", "totalPop" : 12686644 }

{ "_id" : "PA", "totalPop" : 11881643 }

{ "_id" : "IL", "totalPop" : 11427576 }

{ "_id" : "OH", "totalPop" : 10846517 }

{ "_id" : "MI", "totalPop" : 9295297 }

{ "_id" : "NJ", "totalPop" : 7730188 }

{ "_id" : "NC", "totalPop" : 6628637 }

{ "_id" : "GA", "totalPop" : 6478216 }

{ "_id" : "VA", "totalPop" : 6181479 }

{ "_id" : "MA", "totalPop" : 6016425 }

{ "_id" : "IN", "totalPop" : 5544136 }

{ "_id" : "MO", "totalPop" : 5110648 }

{ "_id" : "WI", "totalPop" : 4891769 }

{ "_id" : "TN", "totalPop" : 4876457 }

{ "_id" : "WA", "totalPop" : 4866692 }

{ "_id" : "MD", "totalPop" : 4781379 }

{ "_id" : "MN", "totalPop" : 4372982 }

Type "it" for more

>

3.limit the results to just the first 3 results. What are the top 3 states in population?

db.zipcodes.aggregate([{ $group: { _id: "$state", totalPop: { $sum: "$pop" } }}, { $sort: { totalPop: -1 } },{$limit:3}])

{ "_id" : "CA", "totalPop" : 29754890 }

{ "_id" : "NY", "totalPop" : 17990402 }

{ "_id" : "TX", "totalPop" : 16984601 }

>

==========

Populations by City

1.use aggregate to calculate the totalpopulation for each city (you have to use city/state combination). You can use a combination for the _id of the $group: { city: '$city', state: '$state' }

db.zipcodes.aggregate([{ $group: { _id: { city:"$city",state: "$state" }, pop: { $sum: "$pop" } } }])

{ "_id" : { "city" : "HURON", "state" : "SD" }, "pop" : 15277 }

{ "_id" : { "city" : "CHESTNUT", "state" : "IL" }, "pop" : 436 }

{ "_id" : { "city" : "PORTLAND", "state" : "AR" }, "pop" : 773 }

{ "_id" : { "city" : "LAWRENCEVILLE", "state" : "NJ" }, "pop" : 25497 }

{ "_id" : { "city" : "CREAL SPRINGS", "state" : "IL" }, "pop" : 2743 }

{ "_id" : { "city" : "BRISTOL", "state" : "VA" }, "pop" : 23166 }

{ "_id" : { "city" : "NORTH POMFRET", "state" : "VT" }, "pop" : 254 }

{ "_id" : { "city" : "EMMET", "state" : "NE" }, "pop" : 190 }

{ "_id" : { "city" : "MOUNT HOLLY", "state" : "AR" }, "pop" : 514 }

{ "_id" : { "city" : "PHILLIPS RANCH", "state" : "CA" }, "pop" : 64056 }

{ "_id" : { "city" : "BROWNFIELD", "state" : "ME" }, "pop" : 1148 }

{ "_id" : { "city" : "FOLSOM", "state" : "WV" }, "pop" : 359 }

{ "_id" : { "city" : "TAPPAHANNOCK", "state" : "VA" }, "pop" : 4270 }

{ "_id" : { "city" : "BERWICK", "state" : "IL" }, "pop" : 461 }

{ "_id" : { "city" : "EAST MOLINE", "state" : "IL" }, "pop" : 24023 }

{ "_id" : { "city" : "PITTSFIELD", "state" : "VT" }, "pop" : 450 }

{ "_id" : { "city" : "HENRY", "state" : "NE" }, "pop" : 208 }

{ "_id" : { "city" : "KELSO", "state" : "ND" }, "pop" : 2398 }

{ "_id" : { "city" : "LUGOFF", "state" : "SC" }, "pop" : 8991 }

{ "_id" : { "city" : "ALLENTON", "state" : "WI" }, "pop" : 1449 }

Type "it" for more

2.sort the results by population, highest first

db.zipcodes.aggregate([{ $group: { _id: { city:"$city",state: "$state" }, pop: { $sum: "$pop" } } },{ $sort: { pop: -1 } }])

{ "_id" : { "city" : "CHICAGO", "state" : "IL" }, "pop" : 2452177 }

{ "_id" : { "city" : "BROOKLYN", "state" : "NY" }, "pop" : 2300504 }

{ "_id" : { "city" : "LOS ANGELES", "state" : "CA" }, "pop" : 2102295 }

{ "_id" : { "city" : "HOUSTON", "state" : "TX" }, "pop" : 2095918 }

{ "_id" : { "city" : "PHILADELPHIA", "state" : "PA" }, "pop" : 1610956 }

{ "_id" : { "city" : "NEW YORK", "state" : "NY" }, "pop" : 1476790 }

{ "_id" : { "city" : "BRONX", "state" : "NY" }, "pop" : 1209548 }

{ "_id" : { "city" : "SAN DIEGO", "state" : "CA" }, "pop" : 1049298 }

{ "_id" : { "city" : "DETROIT", "state" : "MI" }, "pop" : 963243 }

{ "_id" : { "city" : "DALLAS", "state" : "TX" }, "pop" : 940191 }

{ "_id" : { "city" : "PHOENIX", "state" : "AZ" }, "pop" : 890853 }

{ "_id" : { "city" : "MIAMI", "state" : "FL" }, "pop" : 825232 }

{ "_id" : { "city" : "SAN JOSE", "state" : "CA" }, "pop" : 816653 }

{ "_id" : { "city" : "SAN ANTONIO", "state" : "TX" }, "pop" : 811792 }

{ "_id" : { "city" : "BALTIMORE", "state" : "MD" }, "pop" : 733081 }

{ "_id" : { "city" : "SAN FRANCISCO", "state" : "CA" }, "pop" : 723993 }

{ "_id" : { "city" : "MEMPHIS", "state" : "TN" }, "pop" : 632837 }

{ "_id" : { "city" : "SACRAMENTO", "state" : "CA" }, "pop" : 628279 }

{ "_id" : { "city" : "JACKSONVILLE", "state" : "FL" }, "pop" : 610160 }

{ "_id" : { "city" : "ATLANTA", "state" : "GA" }, "pop" : 609591 }

Type "it" for more

3.limit the results to just the first 3 results. What are the top 3 cities in population?

db.zipcodes.aggregate([{ $group: { _id: { city:"$city",state: "$state" }, pop: { $sum: "$pop" } } },{ $sort: { pop: -1 } },{$limit:3}])

{ "_id" : { "city" : "CHICAGO", "state" : "IL" }, "pop" : 2452177 }

{ "_id" : { "city" : "BROOKLYN", "state" : "NY" }, "pop" : 2300504 }

{ "_id" : { "city" : "LOS ANGELES", "state" : "CA" }, "pop" : 2102295 }

>

4.What are the top 3 cities in population in Texas?

Bonus

1.Write a query to get the average city population for each state.

db.zipcodes.aggregate([

... { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },

... { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }

...])

{ "_id" : "TN", "avgCityPop" : 9656.350495049504 }

{ "_id" : "MS", "avgCityPop" : 7524.023391812865 }

{ "_id" : "NM", "avgCityPop" : 5872.360465116279 }

{ "_id" : "ID", "avgCityPop" : 4320.811158798283 }

{ "_id" : "TX", "avgCityPop" : 13775.02108678021 }

{ "_id" : "VT", "avgCityPop" : 2315.8765432098767 }

{ "_id" : "IL", "avgCityPop" : 9954.334494773519 }

{ "_id" : "IA", "avgCityPop" : 3123.0821147356583 }

{ "_id" : "LA", "avgCityPop" : 10465.496277915632 }

{ "_id" : "DE", "avgCityPop" : 14481.91304347826 }

{ "_id" : "KY", "avgCityPop" : 4767.164721141375 }

{ "_id" : "MD", "avgCityPop" : 12615.775725593667 }

{ "_id" : "NC", "avgCityPop" : 10622.815705128205 }

{ "_id" : "PA", "avgCityPop" : 8679.067202337472 }

{ "_id" : "NH", "avgCityPop" : 5232.320754716981 }

{ "_id" : "VA", "avgCityPop" : 8526.177931034483 }

{ "_id" : "NE", "avgCityPop" : 3034.882692307692 }

{ "_id" : "WV", "avgCityPop" : 2771.4775888717154 }

{ "_id" : "ME", "avgCityPop" : 3006.4901960784314 }

{ "_id" : "OH", "avgCityPop" : 12700.839578454332 }

Type "it" for more

>

2.What are the top 3 states in terms of average city population?

```
db.zipcodes.aggregate( [
    { $group: { _id: { state: "$state", city: "$city" },pop: { $sum: "$pop" } } },
     { $group: { _id: "$_id.state",avgCityPop: { $avg: "$pop" } } },
       {$sort:{avgCityPop:-1}},
        {$limit:3}
] )
```

{ "_id" : "DC", "avgCityPop" : 303450 }

{ "_id" : "CA", "avgCityPop" : 27756.42723880597 }

{ "_id" : "FL", "avgCityPop" : 27400.958963282937 }

Assignment-3

MongoDB-Complex Queries

MongoDB Exercies-with the restaurants Data Set

1.Download the reastaurants.zip file

2.Unzip the file,you will see restaruarants.json file

3.Run the mongod server

4.Run the following command to import the json provided.it will load the json file into the mongodb with database name-restaurants,collections name-address

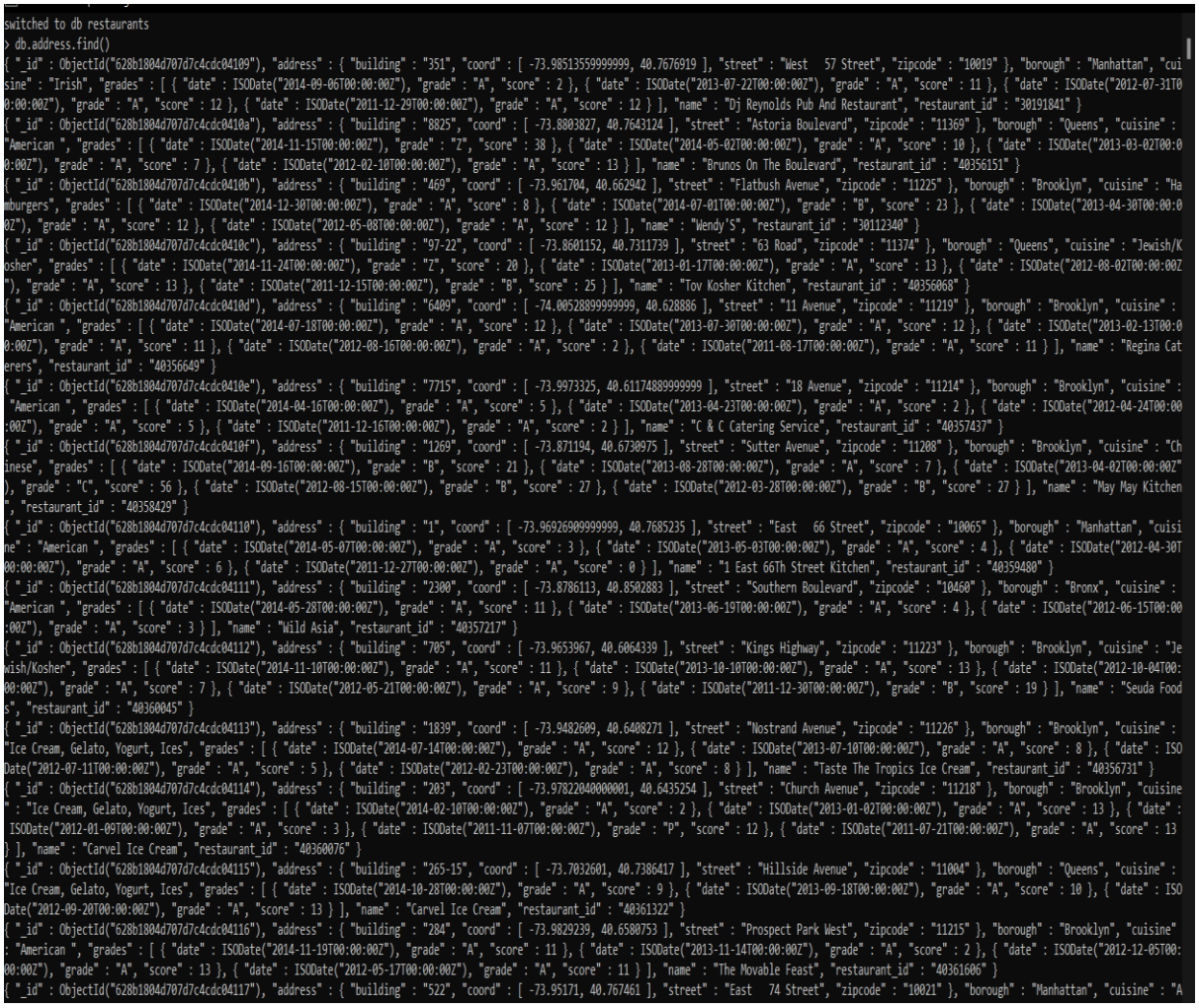
Mongoimport –db restaurants –collection address –file restaurants.json

C:\mongoDB>mongoimport --db restaurants --collection address --file restaurants.json

2022-05-23T10:43:40.722+0530 connected to: mongodb://localhost/

2022-05-23T10:43:40.988+0530 3772 document(s) imported successfully. 0 document(s) failed to import.

5,6,7,8,9.



Exercise Questions

1. Write a MongoDB query to display all the documents in the collection restaurant->

```
db.restaurants.find()
```

2. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant

```
db.restaurants.find({},{"restaurant_id" :
1,"name":1,"borough":1,"cuisine" :1})
```

3.Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant.

```
db.restaurants.find({},{"restaurant_id" :
1,"name":1,"borough":1,"cuisine" :1,"_id":0})
```

4. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.

```
db.restaurants.find({},{"restaurant_id" :
1,"name":1,"borough":1,"address.zipcode" :1,"_id":0})
```

5. Write a MongoDB query to display all the restaurant which is in the borough Bronx.

```
db.restaurants.find({"borough": "Bronx"})
```

6. Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

```
db.restaurants.find({"borough": "Bronx"}).limit(5)
```

7.Write a MongoDB query to display the next 5 restaurants after skipping first

```
db.restaurants.find({"borough": "Bronx"}).skip(5).limit(5)
```

8. Write a MongoDB query to find the restaurants who achieved a score more than 90.

```
db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 90}}}})
```

9. Write a MongoDB query to find the restaurants that achieved a score, more **than 80 but less than 100.**

```
db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt
:100}}}})
```

10. Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.

```
db.restaurants.find({"address.coord" : {$lt : -95.754168}})
```

11. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

```
db.restaurants.find(
                {$and:
```

```
                    [
                       {"cuisine" : {$ne :"American "}},
                       {"grades.score" : {$gt : 70}},
                       {"address.coord" : {$lt : -65.754168}}
                    ]
                }
                    )
```

12. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.

```
db.restaurants.find(
                          {
                            "cuisine" : {$ne : "American "},
                            "grades.score" :{$gt: 70},
                            "address.coord" : {$lt : -65.754168}
                           }
                    )
```

13. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belongs to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.

```
db.restaurants.find( {
                            "cuisine" : {$ne : "American "},
                            "grades.grade" :"A",
                            "borough": {$ne : "Brooklyn"}
                       }
                   ).sort({"cuisine":-1})
```

14. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.

```
db.restaurants.find(
{name: /^Wil/},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
}
)
```

15. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as last three letters for its name.

```
db.restaurants.find(
{name: /ces$/},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
}
)
```

16. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Reg' as three letters somewhere in its name.

```
db.restaurants.find(
{"name": /.*Reg.*/},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
}
)
```

17. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.

```
db.restaurants.find(
{
"borough": "Bronx" ,
$or : [
{ "cuisine" : "American " },
{ "cuisine" : "Chinese" }
]
}
)
```

18. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which belong to the borough Staten Island or Queens or Bronxor Brooklyn.

```
db.restaurants.find(
{"borough" :{$in :["Staten Island","Queens","Bronx","Brooklyn"]}},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
```

```
}
)
```

19. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronxor Brooklyn.

```
db.restaurants.find(
{"borough" :{$nin :["Staten Island","Queens","Bronx","Brooklyn"]}},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
}
)
```

20. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10.

```
db.restaurants.find(
{"grades.score" :
{ $not:
{$gt : 10}
}
},
{
"restaurant_id" : 1,
"name":1,"borough":1,
"cuisine" :1
}
)
```

21. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.

```
db.restaurants.find(
{$or: [
  {name: /^Wil/},
  {"$and": [
        {"cuisine" : {$ne :"American "}},
        {"cuisine" : {$ne :"Chinees"}}
```

```
   ]}
]}
,{"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1}
)
```

22. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates..

```
db.restaurants.find(
                {
                 "grades.date": ISODate("2014-08-11T00:00:00Z"),
                 "grades.grade":"A" ,
                 "grades.score" : 11
                },
                {"restaurant_id" : 1,"name":1,"grades":1}
            )
```

23. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

```
db.restaurants.find(
                    { "grades.1.date": ISODate("2014-08-
11T00:00:00Z"),
                      "grades.1.grade":"A" ,
                      "grades.1.score" : 9
                    },
                     {"restaurant_id" : 1,"name":1,"grades":1}
                )
```

24. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..

```
db.restaurants.find(
                    {
                      "address.coord.1": {$gt : 42, $lte : 52}
                    },
                      {"restaurant_id" :
1,"name":1,"address":1,"coord":1}
                )
```

25. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

```
db.restaurants.find().sort({"name":1})
```

26. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

```
db.restaurants.find().sort(
                          {"name":-1}
                          )
```

27. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
db.restaurants.find().sort(
                           {"cuisine":1,"borough" : -1,}
                          )
```

28. Write a MongoDB query to know whether all the addresses contains the street or not.

```
db.restaurants.find(
                     {"address.street" :
                         { $exists : true }
                      }
                   )
```

29. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

```
db.restaurants.find(

                     {"address.coord" :

                        {$type : 1}

                     }

                   )
```

30. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

```
db.restaurants.find(
```

```
                    {"grades.score" :
                       {$mod : [7,0]}
                    },
                       {"restaurant_id" : 1,"name":1,"grades":1}
                )
```

31. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.

```
db.restaurants.find(
                    { name :
                      { $regex : "mon.*", $options: "i" }
                    },
                        {
                          "name":1,
                          "borough":1,
                          "address.coord":1,
                          "cuisine" :1
                        }
                )
```

32. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.

```
db.restaurants.find(

                    { name :
                      { $regex : /^Mad/i, }
                    },
                        {
                          "name":1,
                          "borough":1,
                          "address.coord":1,
                          "cuisine" :1
                        }
                )
```

mongo