
5
Mongo Shell

Objectives

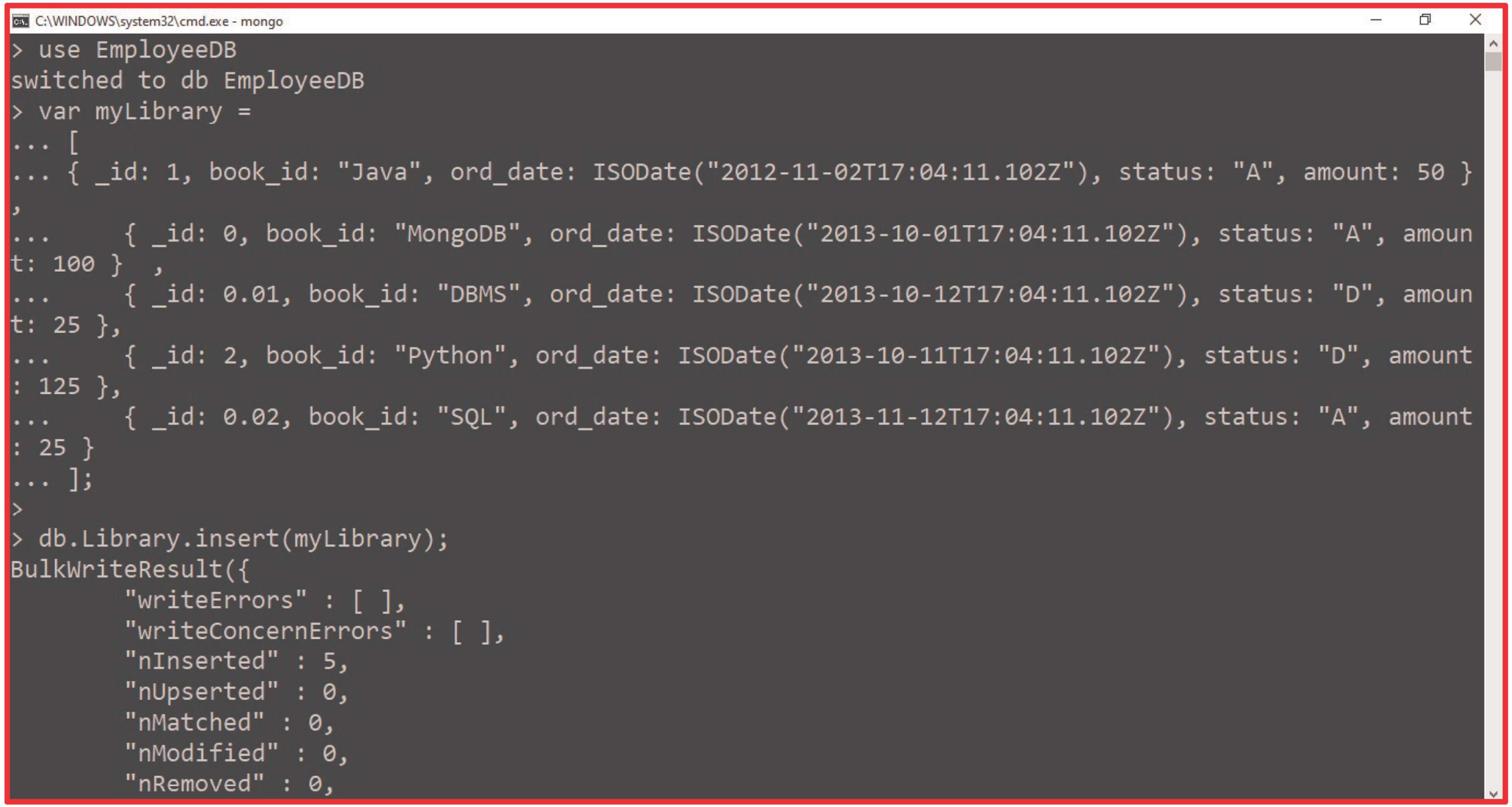
After completing this lesson, you should be able to do the following:

- Shell Collection Methods
- Cursor Methods
- MongoDB Database Commands
- User Management Methods
- Role Management Methods
- MongoDB Replication Methods
- Bulk Operation Methods

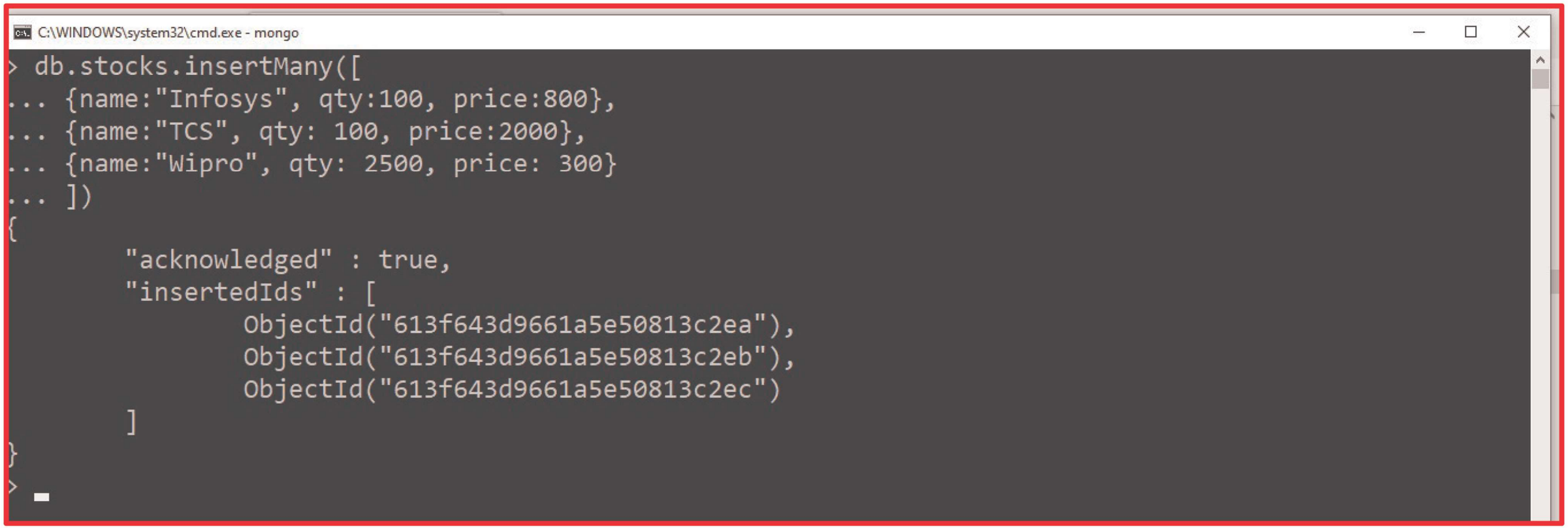
MongoDB Shell Collection Methods

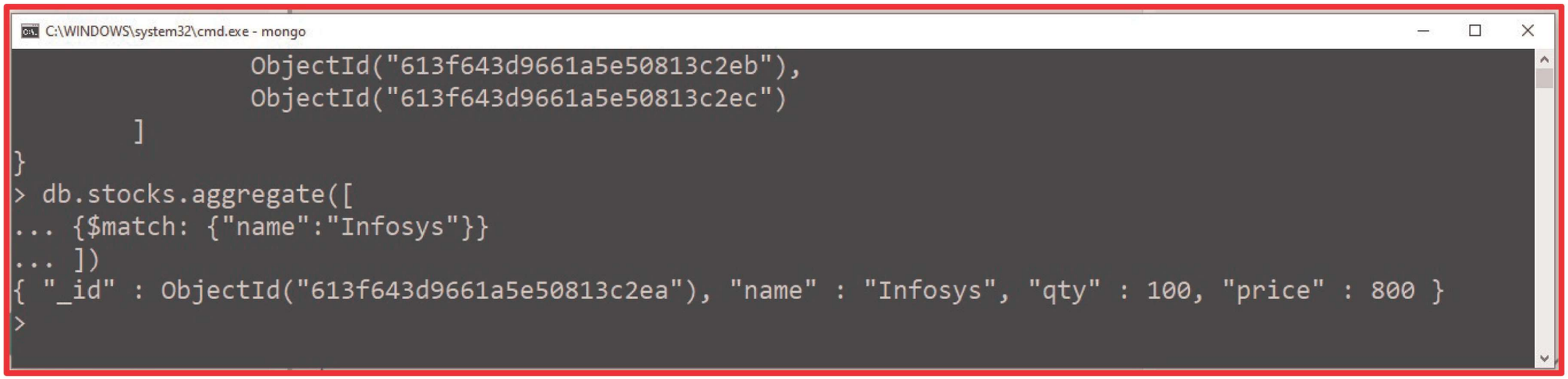
1: db.collection.aggregate(pipeline, option)

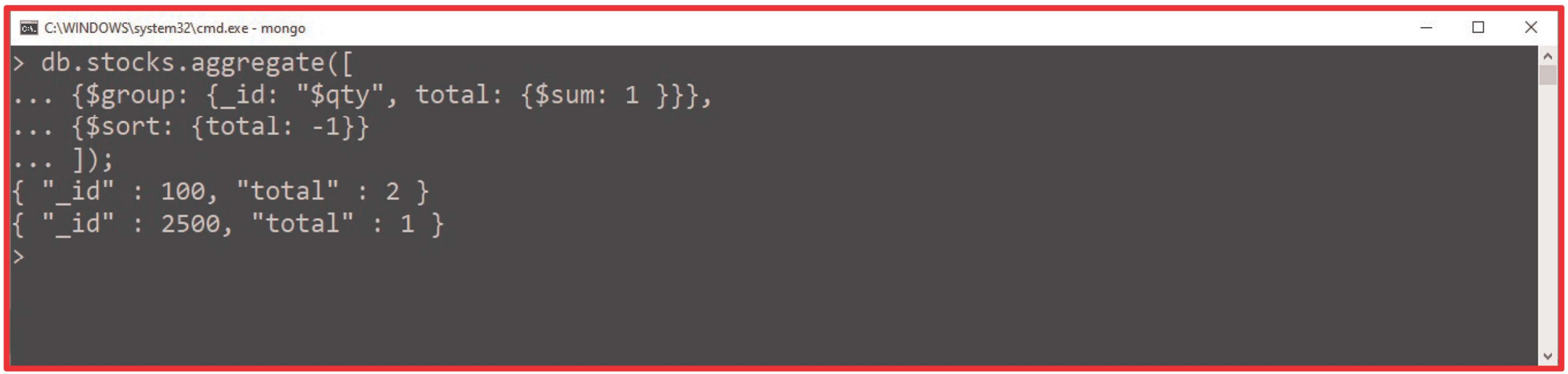
- The aggregate method calculates mass values for the data in a collection/table or in a view.
  - **Pipeline:** It is an array of mass data operations or stages. It can accept the pipeline as a separate argument, not as an element in an array. If the pipeline is not specified as an array, then the second parameter will not be specified.
  - **Option:** A document that passes the aggregate command. It will be available only when you specify the pipeline as an array.


C:\WINDOWS\system32\cmd.exe - mongo
> use EmployeeDB
switched to db EmployeeDB
> var myLibrary =
... [
... { _id: 1, book_id: "Java", ord_date: ISODate("2012-11-02T17:04:11.102Z"), status: "A", amount: 50 }
,
...   { _id: 0, book_id: "MongoDB", ord_date: ISODate("2013-10-01T17:04:11.102Z"), status: "A", amoun
t: 100 } ,
...   { _id: 0.01, book_id: "DBMS", ord_date: ISODate("2013-10-12T17:04:11.102Z"), status: "D", amoun
t: 25 },
...   { _id: 2, book_id: "Python", ord_date: ISODate("2013-10-11T17:04:11.102Z"), status: "D", amount
: 125 },
...   { _id: 0.02, book_id: "SQL", ord_date: ISODate("2013-11-12T17:04:11.102Z"), status: "A", amount
: 25 }
...];
>
> db.Library.insert(myLibrary);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 5,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,

Calculating the sum



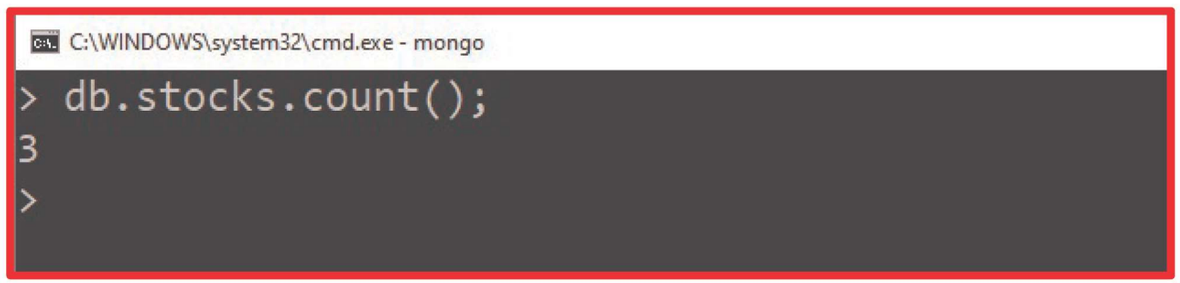

C:\WINDOWS\system32\cmd.exe - mongo
ObjectId("613f643d9661a5e50813c2eb"),
ObjectId("613f643d9661a5e50813c2ec")
]
}
> db.stocks.aggregate([
... {$match: {"name":"Infosys"}}
...])
{ "_id" : ObjectId("613f643d9661a5e50813c2ea"), "name" : "Infosys", "qty" : 100, "price" : 800 }
>


C:\WINDOWS\system32\cmd.exe - mongo
> db.stocks.aggregate([
... {$group: {_id: "$qty", total: {$sum: 1 }}},
... {$sort: {total: -1}}
...]);
{ "_id" : 100, "total" : 2 }
{ "_id" : 2500, "total" : 1 }
>

2. db.collection.count()

- The count() method return the number of documents that would match a find method query for the collection or view.
- Syntax

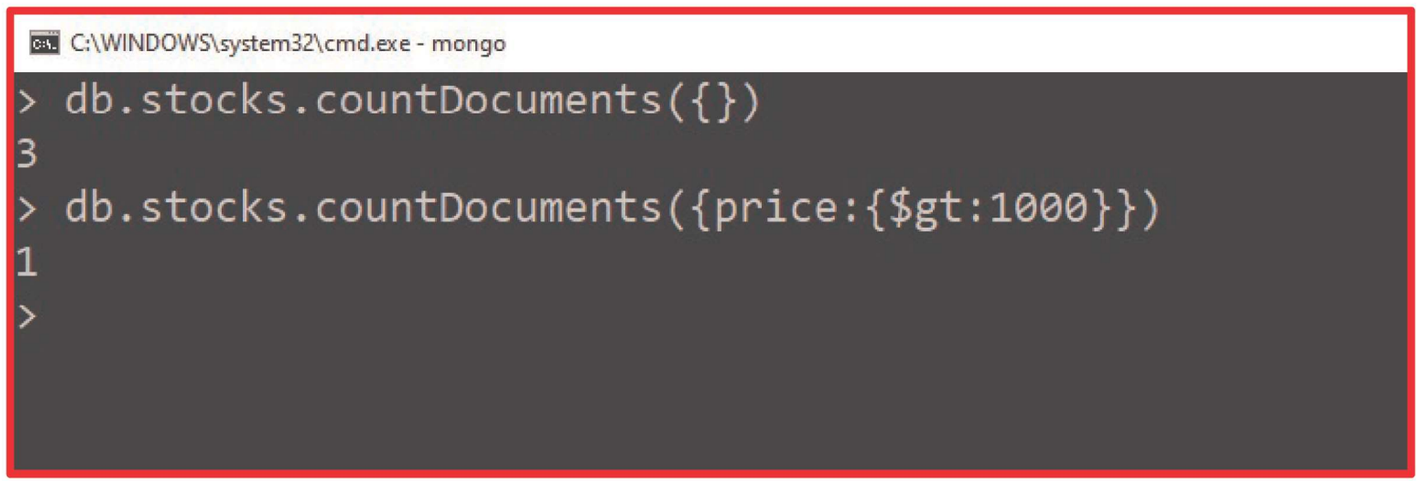
```
db.stocks.count()
```



3. Db.collection.countDocuments(query, options)

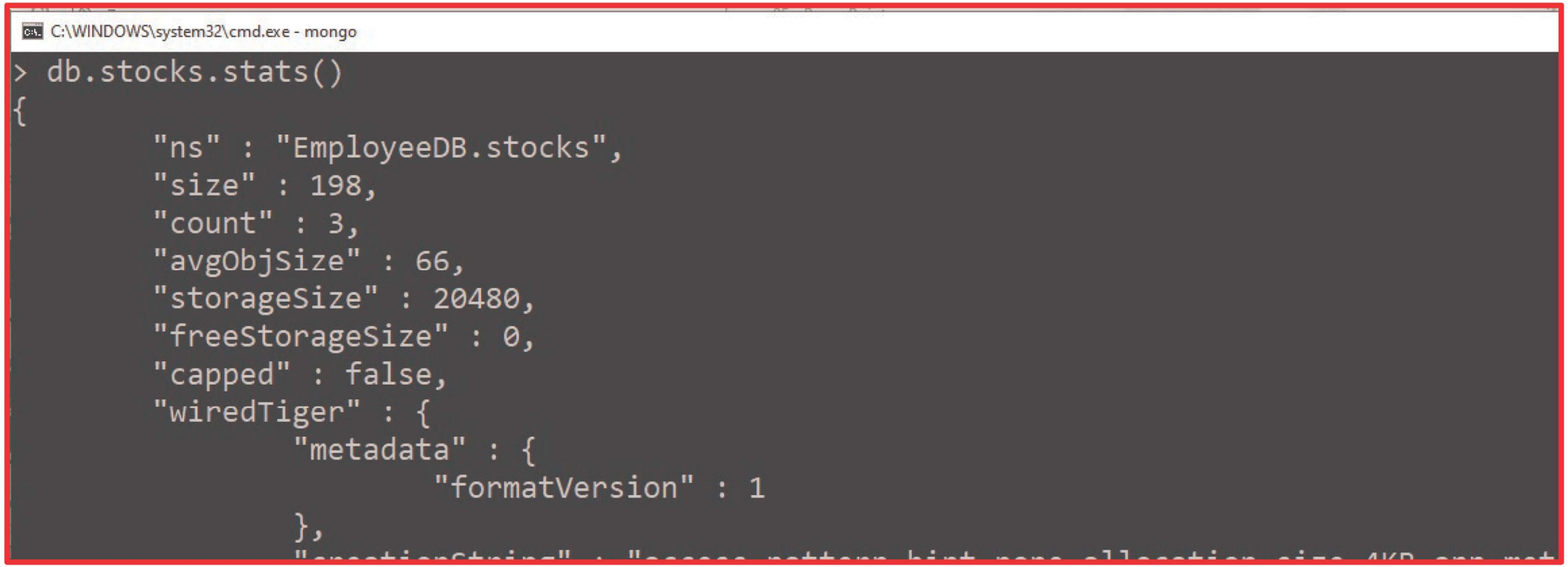
- The countDocument() method return the number of documents that match the query for a collection or view. it does not use the metadata to return the count.
- Syntax :

```
db.collection.countDocuments( <query>, <options> )
```



4. db.collection.dataSize()

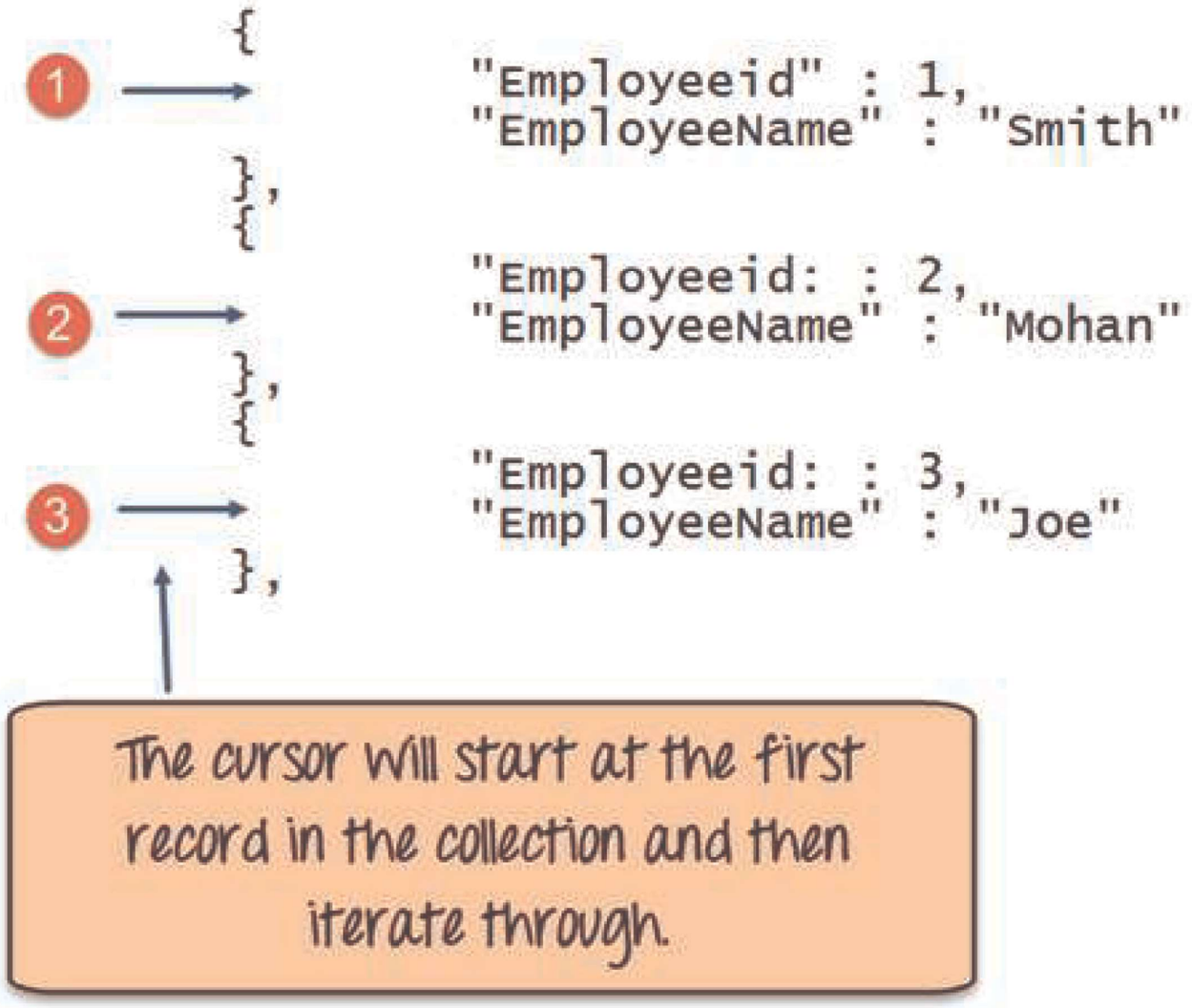
- The data size method have a cover around the output of the collStats (i.e. db.collection.stats()) command.



MongoDB Cursors

What is Cursor in MongoDB?

- When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.
- By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one
- The cursor object will point to the first document and then iterate through all of the documents of the collection.

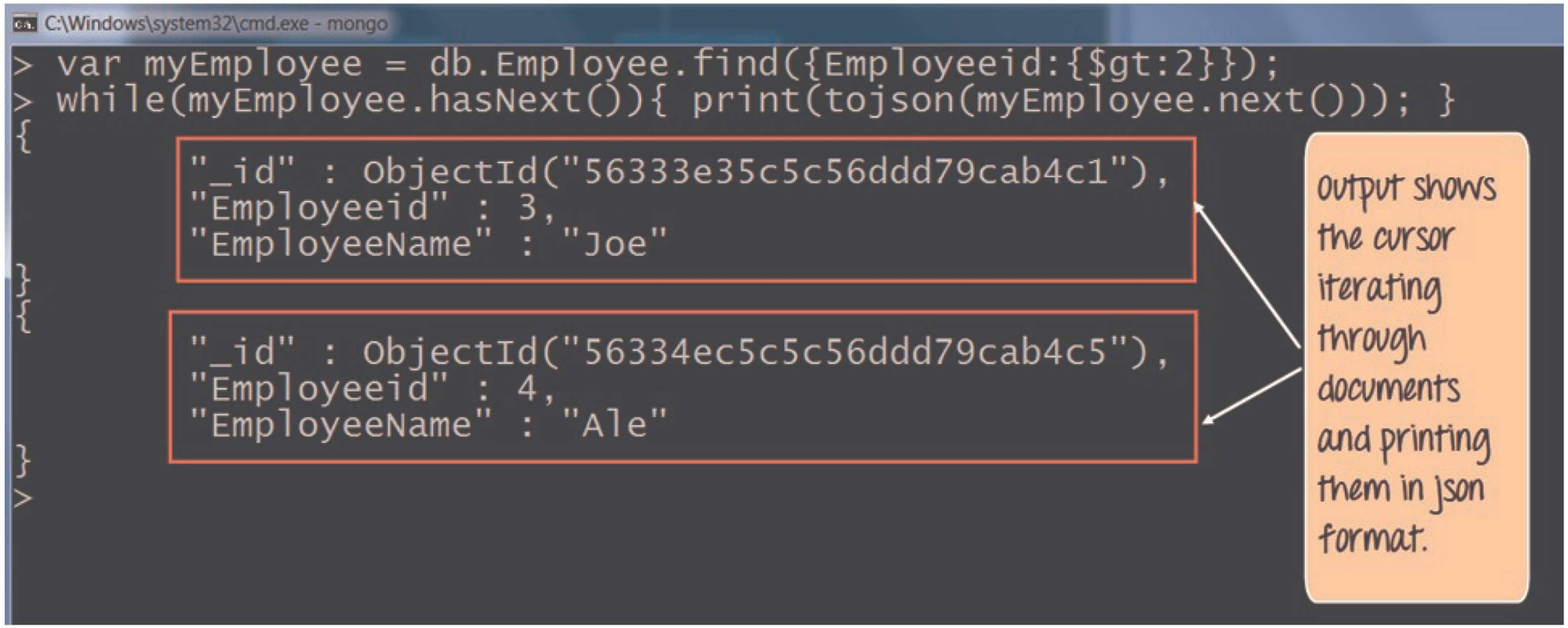

{
1
"Employeeid" : 1,
"EmployeeName" : "Smith"
},
{
"Employeeid: : 2,
2
"EmployeeName" : "Mohan"
},
{
"Employeeid: : 3,
3
"EmployeeName" : "Joe"
},
The cursor will start at the first record in the collection and then iterate through.

Sample

```
var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});

        while(myEmployee.hasNext())

        {

                print(tojson(myEmployee.next()));

        }
```

- First it will take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the JavaScript variable 'myEmployee'
- Next we use the while loop to iterate through all of the documents which are returned as part of the query.
- Finally for each document, we print the details of that document in JSON readable format.

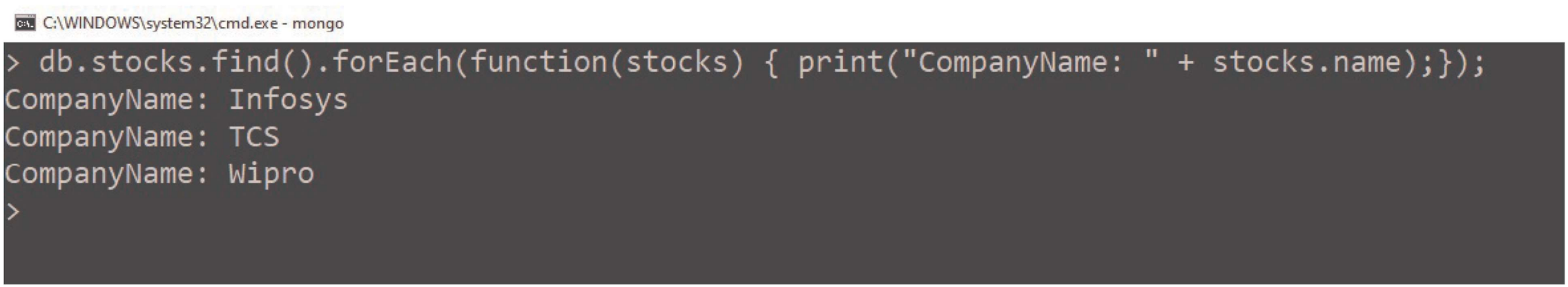
Output


C:\Windows\system32\cmd.exe - mongo
> var myEmployee = db.Employee.find({Employeeid:{$gt:2}});
> while(myEmployee.hasNext()){ print(tojson(myEmployee.next())); }
{
"_id" : ObjectId("56333e35c5c56ddd79cab4c1"),
"Employeeid" : 3,
"EmployeeName" : "Joe"
}
{
"_id" : ObjectId("56334ec5c5c56ddd79cab4c5"),
"Employeeid" : 4,
"EmployeeName" : "Ale"
}
output shows the cursor iterating through documents and printing them in json format.

1. **cursor.forEach(function)**

   JavaScript function will be applied to all the documents by the cursor using the forEach method.

```
db.collection.find().forEach(<function>)
```



2. cursor.limit()

This method is used to specify the maximum number of documents returned by the cursor. It will be used within the cursor and comparable to the LIMIT statement in a SQL database.

```
db.collection.find(<query>).limit(<number>)
```

3. cursor.max() and cursor.min()

The max method is used to restrict the results of find().max() method. MongoDB specifies the exclusive upper bound for a specific index that provides a way to specify an upper bound for the compound key indexes.

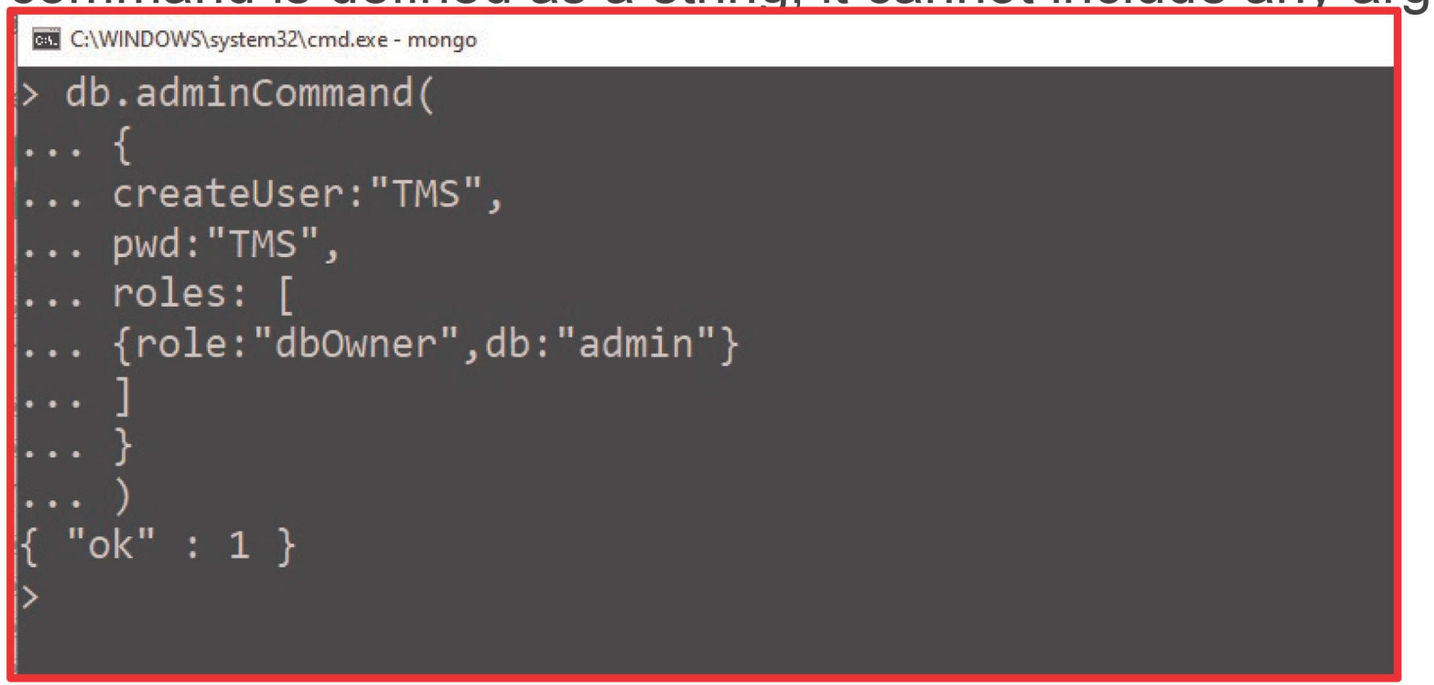
To constrain the results of find(). min() MongoDB specifies the lower bound for a specific index in order . This method provides a way to define the lower bounds on compound key indexes.

MongoDB Database Commands

The MongoDB database commands are used to create, modify, and update the database.

1. db.adminCommand(cmd)

- The admin command method runs against the admin database to run specified database commands by providing a helper.
- Command: Either the argument is specified in the document form or a string form. If the command is defined as a string, it cannot include any argument.



2. db.commandHelp(command)

We have the help option for the specified database command using the commandHelp method. The command parameter contains the name of a database command.

```
> help
    db.help()                    help on db methods
    db.mycoll.help()             help on collection methods
    sh.help()                    sharding helpers
    rs.help()                    replica set helpers
    help admin                   administrative help
    help connect                 connecting to a db help
    help keys                    key shortcuts
    help misc                    misc things to know
    help mr                      mapreduce

    show dbs                     show database names
    show collections             show collections in current database
    show users                   show users in current database
    show profile                 show most recent system.profile
                                 entries with time >= 1ms

    show logs                    show the accessible logger names
    show log [name]              prints out the last segment of log
                                 in memory, 'global' is default

    use <db_name>                set current database
    db.foo.find()                list objects in collection foo
    db.foo.find( { a : 1 } )     list objects in foo where a == 1
    it                           result of the last line evaluated;
                                 use to further iterate
```

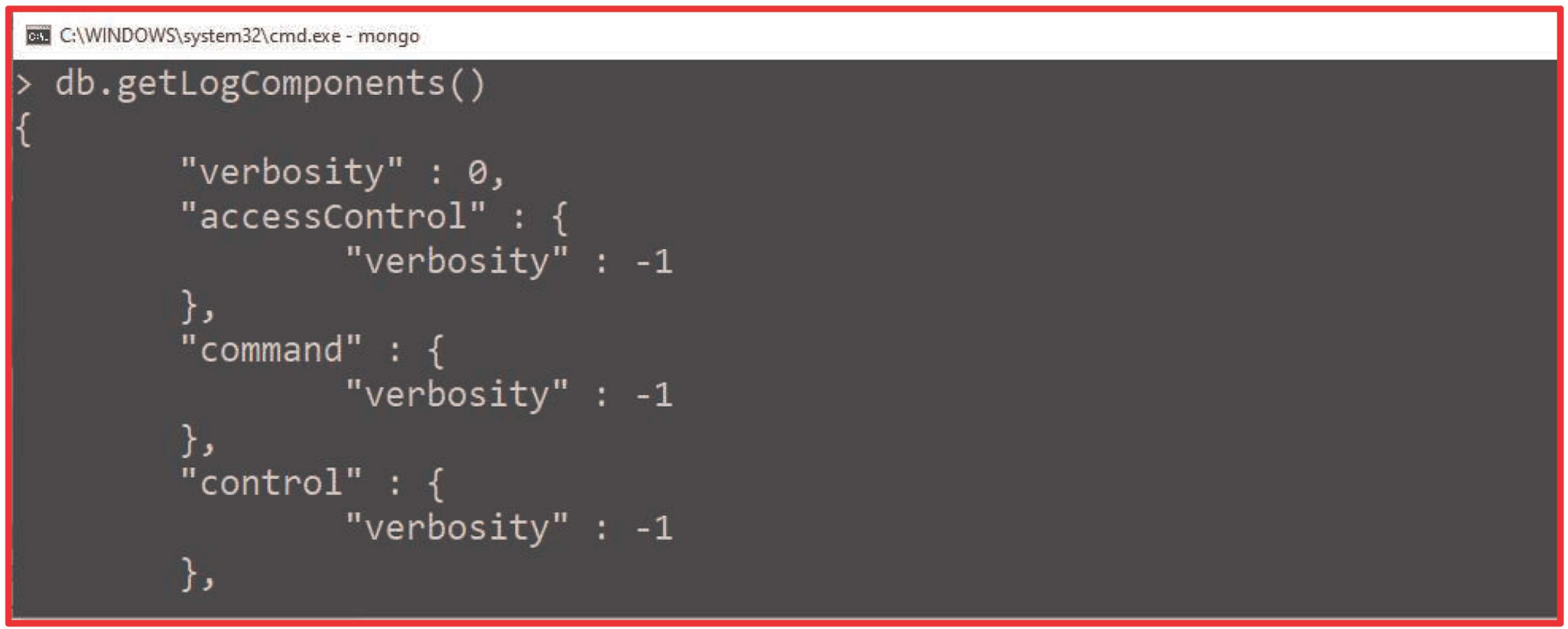
3. db.createCollection(name, options)

A new collection or view will be created using this method. The createCollection method is used primarily for creating new collections that use specific options when the collection is first referenced in a command.

```
db.createCollection( "student", {
  validator: { $jsonSchema: {
    bsonType: "object",
    required: [ "phone" ],
    properties: {
      phone: {
        bsonType: "string",
        description: "must be a string and is required"
      },
      email: {
        bsonType : "string",
        pattern: "@mongodb\.com$",
        description: "must be a string and match the regular expression pattern"
      },
      status: {
        enum: [ "Unknown", "Incomplete" ],
        description: "can only be one of the enum values"
      }
    }
  } }
} )
```

4. db.getLogComponents()

The getLog method returns the current stiltedly settings. The method determines the amount of Log Messages produced by MongoDB for each log message component.



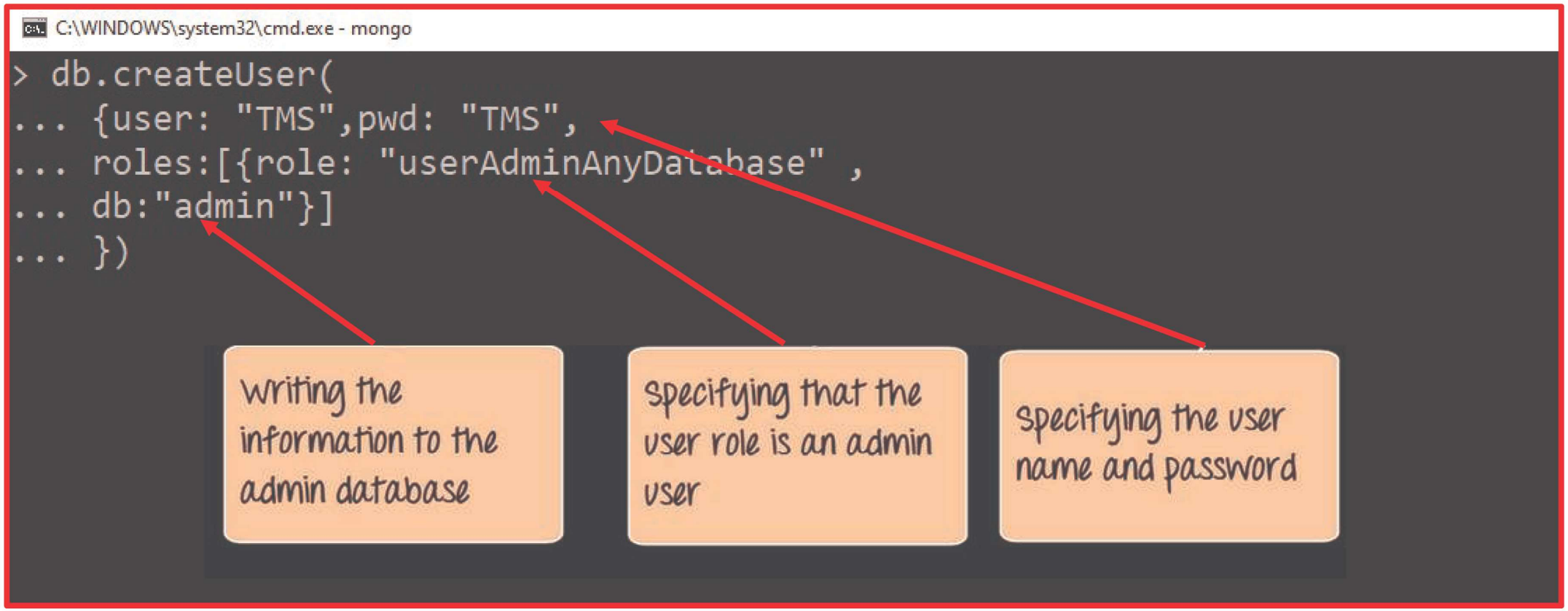
User & Role Management Commands

MongoDB Create Administrator User

- Creating a user administrator in MongoDB is done by using the createUser method. The following example shows how this can be done.

```
db.createUser(
{           user: “TMS",
            pwd: “TMS",

            roles:[{role: "userAdminAnyDatabase" , db:"admin"}]})
```

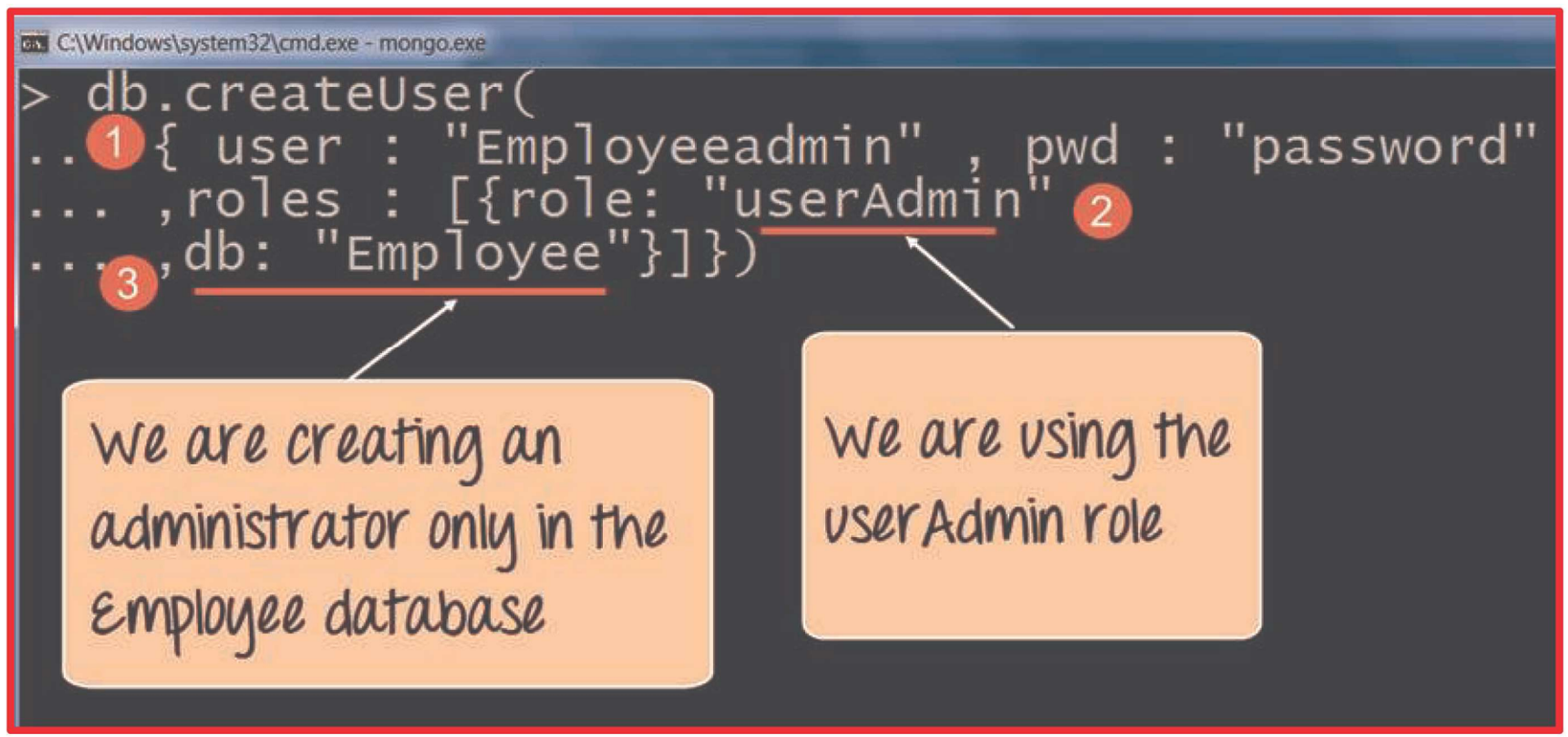
1. The first step is to specify the “username” and “password” which needs to be created.
2. The second step is to assign a role for the user. Since it needs to be a database administrator in which case we have assigned to the “userAdminAnyDatabase” role. This role allows the user to have administrative privileges to all databases in MongoDB.
3. The db parameter specifies the admin database which is a special Meta database within MongoDB which holds the information for this user.


C:\WINDOWS\system32\cmd.exe - mongo
> db.createUser(
... {user: "TMS",pwd: "TMS",
... roles:[{role: "userAdminAnyDatabase" ,
... db:"admin"}]
... })
writing the information to the admin database
specifying that the user role is an admin user
specifying the user name and password

Command Executed Successfully

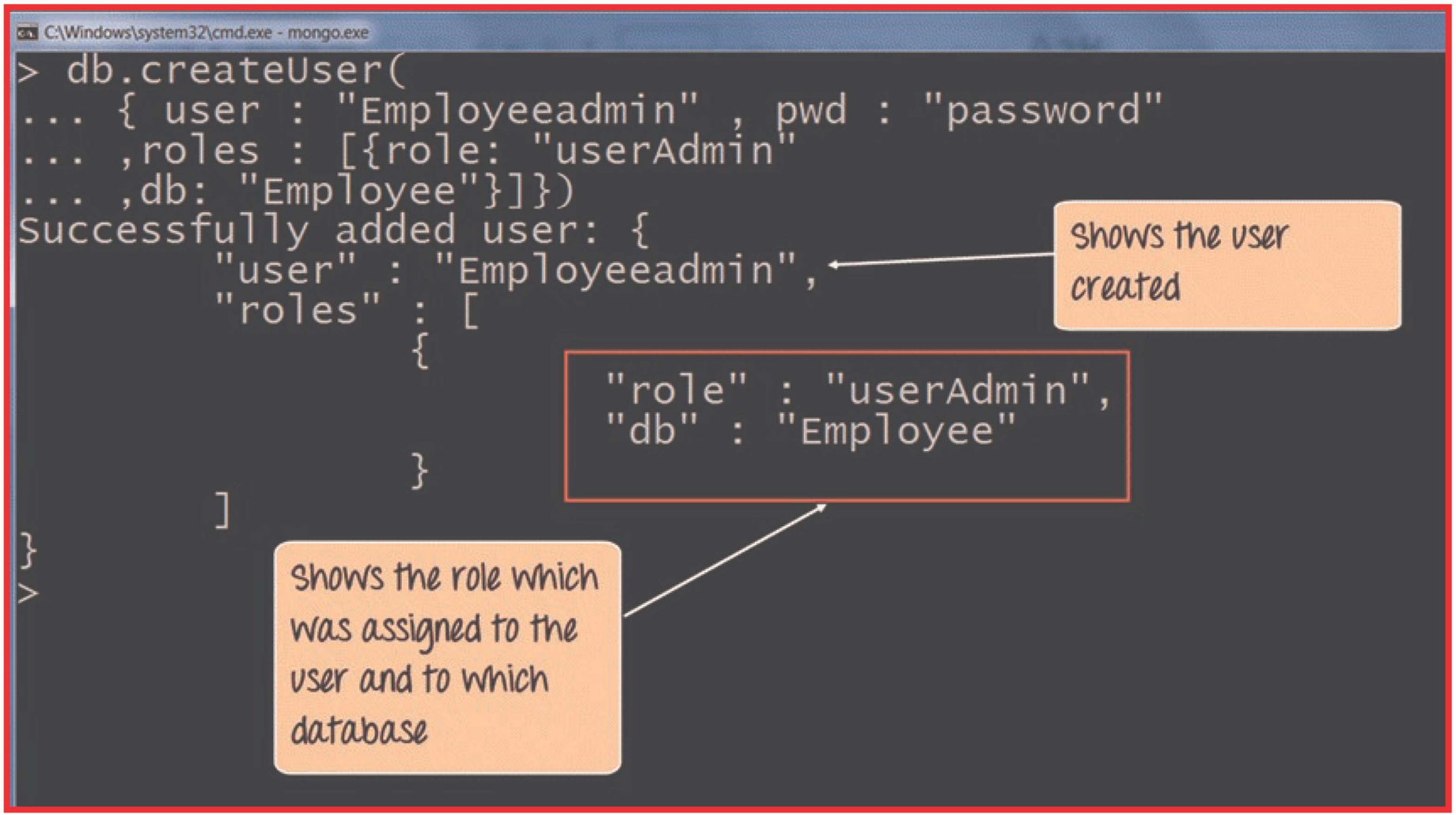
MongoDB Create User for Single Database

- To create a user who will manage a single database, we can use the same command as mentioned above but we need to use the “userAdmin” option only.



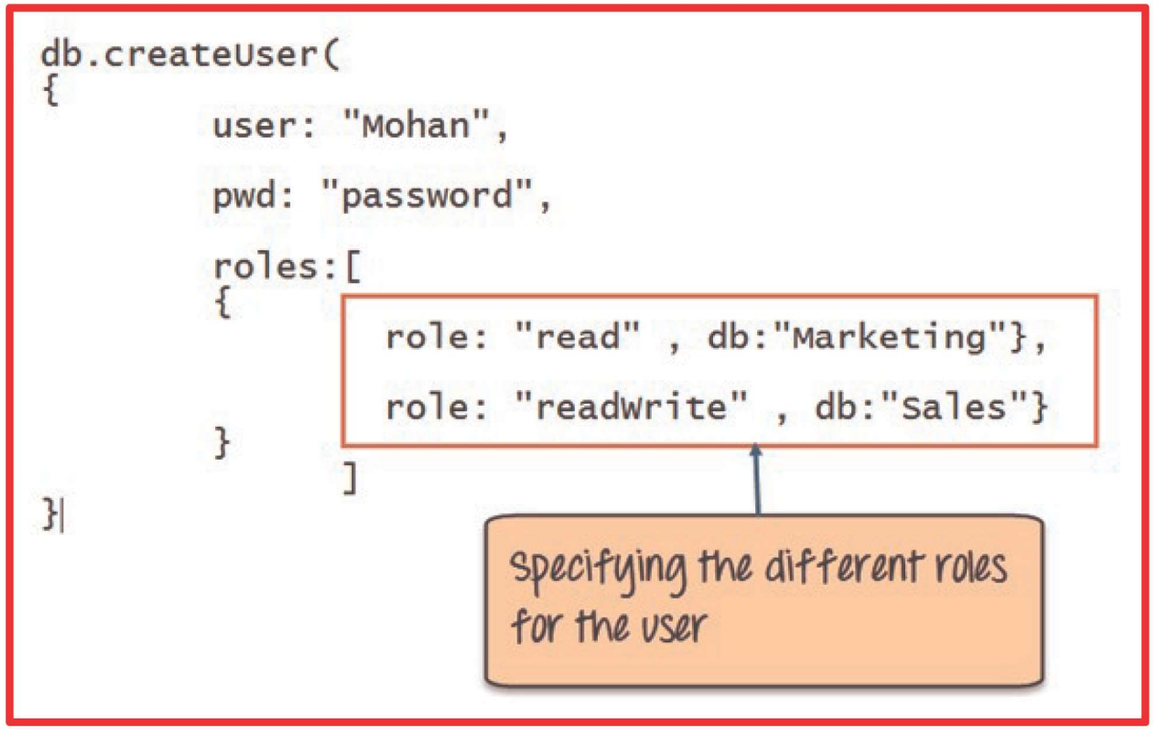
1. The first step is to specify the “username” and “password” which needs to be created.
2. The second step is to assign a role for the user which in this case since it needs to be a database administrator is assigned to the “userAdmin” role. This role allows the user to have administrative privileges only to the database specified in the db option.
3. The db parameter specifies the database to which the user should have administrative privileges on.

Output


C:\Windows\system32\cmd.exe - mongo.exe
> db.createUser(
... { user : "Employeeadmin" , pwd : "password"
... ,roles : [{role: "userAdmin"
... ,db: "Employee"}]})
Successfully added user: {
"user" : "Employeeadmin",
"roles" : [
{
"role" : "userAdmin",
"db" : "Employee"
}
]
}
>
Shows the user created
Shows the role which was assigned to the user and to which database

Managing users

- First understand the roles which you need to define.
- There is a whole list of role available in MongoDB.
- For example, there is a the “read role” which only allows read only access to databases and then there is the “readwrite” role which provides read and write access to the database , which means that the user can issue the insert, delete and update commands on collections in that database.



- The above code snippet shows that a user called Mohan is created, and he is assigned multiple roles in multiple databases. In the above example, he is given read only permission to the “Marketing” database and readWrite permission to the “Sales” database.

MongoDB
Replication Methods

Methods [Already Discussed]

1. rs.add()
2. rs.conf()
3. rs.initiate(configuration)
4. rs.reconfig(configuration, force)

MongoDB
Bulk Operation Methods

MongoDB Bulk Operation Methods

The MongoDB bulk methods are used to perform bulk operation like bulk write and bulk remove.

1. db.collection.initializeOrderBulkOp()

- The initializeOrderBulkOp and gives a new Bulk() operations builder for the collection. It constructs an ordered list of write operations that MongoDB runs in bulk.
- he following initializes a Bulk() operations builder on the users collection, adds a series of write operations, and executes the operations:

C:\WINDOWS\system32\cmd.exe - mongo

```
>     var bulk = db.users.initializeOrderedBulkOp();
>     bulk.insert( { user: "John", status: "A", points: 0 } );
>     bulk.insert( { user: "Peter", status: "A", points: 0 } );
>     bulk.insert( { user: "Clint", status: "B", points: 0 } );
>     bulk.find( { status: "D" } ).remove();
>     bulk.find( { status: "B" } ).update( { $set: { comment: "Awaited" } } );
>     bulk.execute();
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 1,
        "nModified" : 1,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

2. Bulk() method

- Bulk method can be used to create a list of write operations to perform in bulk for one collection.

Bulk.insert()	It can be used to add an insert operation inside the list of operations.
Bulk.find()	It specifies the query condition for a remove or an update operation.
Bulk.find.removeOne()	It adds a remove operation to a list of operations for a single document.
Bulk.find.remove()	It adds remove operation to a list of operations for multiple document.
Bulk.find.replaceOne()	It adds a document replacement operation to a numbers of operations.
Bulk.find.updateOne()	It adds a single document update operation to an array of operations.
Bulk.find.update()	It adds a multi update operation to an array of operations.
Bulk.find.upsert()	It can be used to specify upsert that is true for an update operation.
Bulk.execute()	It executes an array of operations in bulk.
Bulk.getOperations()	It gives an array of write operations executed in the Bulk() operations object.
Bulk.tojson()	It returns a JSON document that have the number of operations and batches in the Bulk() operations object.
Bulk.toString()	It gives the Bulk.tojson() results as a string.

3. Bulk.execute() method

   It runs the list of operations that is built by the Bulk() methods builder.

4. **Bulk.find.remove() method**

   It adds a remove operation to a bulk operations list. The Bulk.find() method can be used to specify the condition, which determines the documents to be remove. The Bulk.find.remove() removes all matching documents from the collection.

5. Bulk.toString()

It can be used to return the output as a string a JSON document that contains the number of operations and batches inside the Bulk() object.

```
var bulk = db.items.initializeOrderedBulkOp();
bulk.insert( { item: "abc123", status: "A", defaultQty: 500, points: 5 } );
bulk.insert( { item: "ijk123", status: "A", defaultQty: 100, points: 10 } );
bulk.find( { status: "D" } ).removeOne();
bulk.toString();
```

Output

```
> var bulk = db.items.initializeOrderedBulkOp();
> bulk.insert( { item: "abc123", status: "A", defaultQty: 500, points: 5 } );
> bulk.insert( { item: "ijk123", status: "A", defaultQty: 100, points: 10 } );
> bulk.find( { status: "D" } ).removeOne();
> bulk.toString();
{ "nInsertOps" : 2, "nUpdateOps" : 0, "nRemoveOps" : 1, "nBatches" : 2 }
>
```

Summary

In this lesson, you should have learned how to:

- Shell Collection Methods
- Cursor Methods
- MongoDB Database Commands
- User Management Methods
- Role Management Methods
- MongoDB Replication Methods
- Bulk Operation Methods