Module 3

CRUD Operations

Basic Operations

Table of Contents

How to create a database in MongoDB?

To create a database in MongoDB we simply write "**use Database_Name",** where **Database_Name** can be any name like: mydb. This command will create a new database, if it does not exists otherwise it will return the existing database.

SYNTAX: Basic syntax to create a database in MongoDB is:

use Database_Name

If you want to create a database with name "mydb" then

use mydb

```
>
> use mydb
switched to db mydb
>
```

To check currently selected data, write

>db

```
> db
mydb
> _
```

If you want to check your database list, write

>show dbs

```
> show dbs
DATABASE_NAME   0.078GB
admin           0.078GB
local           0.078GB
test            0.078GB
> _
```

Or you can use “show databases” command to print the list of databases same as “show dbs”

How to create a collection in MongoDB?

To create a collection in MongoDB use the command “db.createCollection(name, options)”.

Where “name” specifies the name of collection which is of string type and “options” specifies the options about memory size and indexing.

“options” parameter is optional here. Which contains the following parameters:

Field	Type	Description
capped	boolean	If true, enables a Capped Collection. Capped collection is a collection that is fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	boolean	If true, automatically create index on _id fields. **By default its value is false.**
size	number	Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	Specifies the maximum number of documents allowed in the capped collection.

db.createCollection(“mycoll”)

```
> use mydb
switched to db mydb
>
>
> db.createCollection("mycoll")
{ "ok" : 1 }
> _
```

show collections

```
> show collections
mycoll
system.indexes
>
```

If you do not want to predefine the collection then a new collection can be created by using a command like:

db.mycollection.insert ({“name”: “sam”})

```
> db.mycollection.insert({"name" : "sam"})
WriteResult({ "nInserted" : 1 })
>
>
```

Now see the collection list using “show collections”.

```
> show collections
mycoll
mycollection
system.indexes
> _
```

db.createCollection(“coll” , { capped : true, autoIndexID : true, size : 6142800, max : 10000 })

```
> db.createCollection("coll", { capped : true, autoIndexID : true, size : 614280
0, max : 10000 })
{ "ok" : 1 }
> _
```

How to insert the documents in a Collection?

To insert data into a collection you need to use insert() method in mongodb.

Syntax: db.Collection_Name.insert(document)

db.mycoll.insert (

{ “name” : “mary”,

“age” : 22,

“address” :

{ “home” : “banglore”,

“office” : “delhi” }

})

```
> db.mycoll.insert({"name" : "sam" , "age" : 23 })
WriteResult({ "nInserted" : 1 })
> db.mycoll.insert({"name" : "mary" , "age" : 22, "address" : {"home" : "banglor
","office" : "delhi"}})
WriteResult({ "nInserted" : 1 })
>
```

To insert multiple documents in a single query, you can pass an array of documents in insert() command.

```
> db.mycollection.insert([ {"title" : "MongoDB Training", "by" : "easylearning.g
uru" , "courses" : ["mongoDB" ,"hadoop" ,"python"], "users": 1000}, {"title" : "
MongoDB class" , "website" : "www.easylearning.guru" , "comments": [ {"user" : "
user1", "msg" : "nice" , "like" : 100}]}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

If we want to insert embedded documents in a single query, you can use the following command for insert():

db.mycollection.insert({“title” : “A blog post” ,”heading” : “blog is about mongodb” , “comments” : [{ “name” : “joe”, “email” : “joe@example.com” , “content” : “nice blog” }, {“name” : “mary” , “email” : “mary@abc.com” , “content” : “good post” }] })

```
> db.mycollection.insert({"title" : "A blog post" , "heading" : "blog is about m
ongodb" , "comments" : [{"name" : "joe" ,"email" : "joe@example.com" ,"content"
: "nice blog" },{"name" : "mary" ,"email" :"mary@abc.com" , "content" : "good po
st" }]})
WriteResult({ "nInserted" : 1 })
>
>
>
```

How to query any document in MongoDB?

If you want to query data from a database, you need to use find() method in MongoDB. find() method will display all the documents in a non-structured way.

Syntax: db.Collection_Name.find()

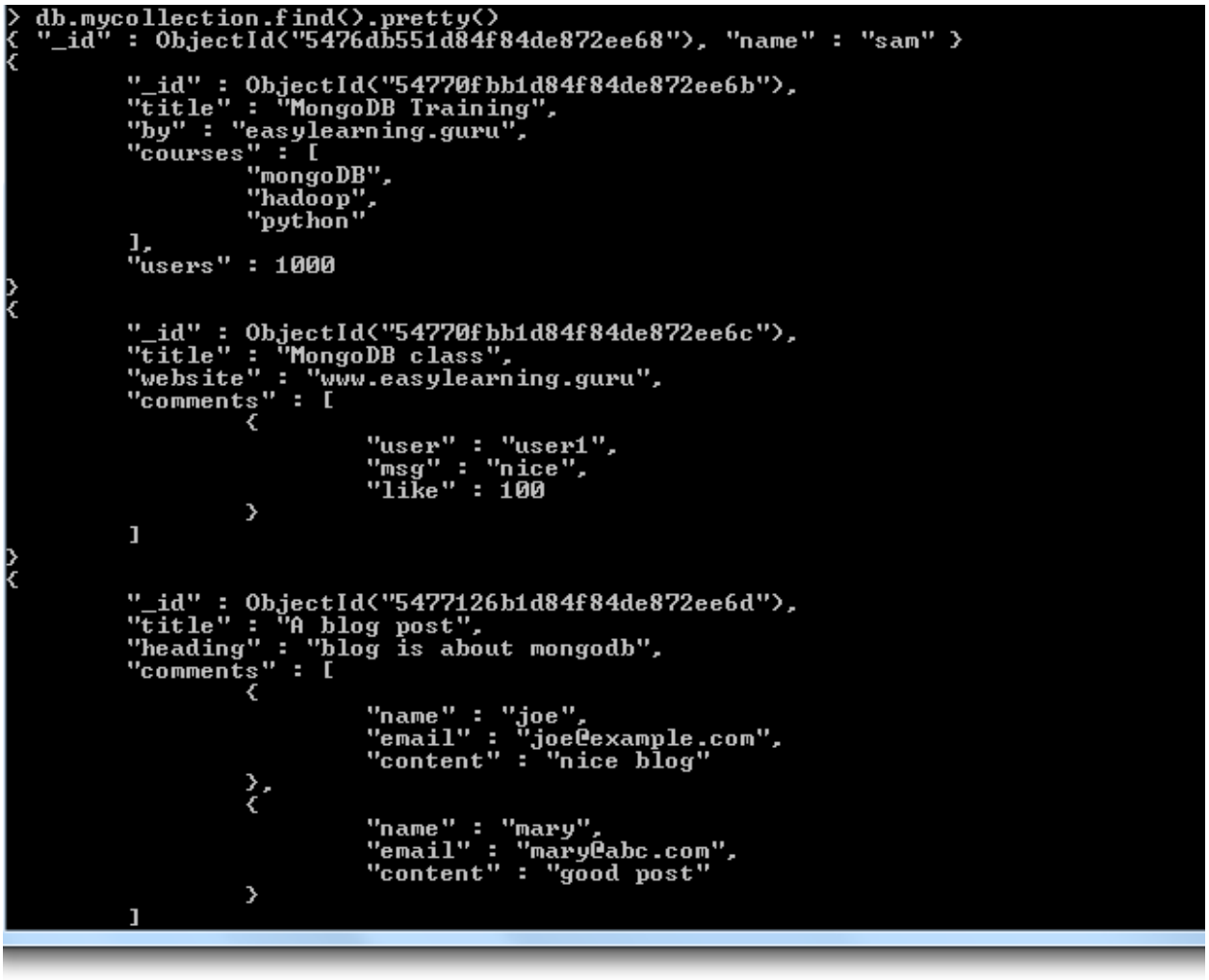
db.mycollection.find()

```
> db.mycollection.find()
{ "_id" : ObjectId("5476db551d84f84de872ee68"), "name" : "sam" }
{ "_id" : ObjectId("54770fbb1d84f84de872ee6b"), "title" : "MongoDB Training", "b
y" : "easylearning.guru", "courses" : [ "mongoDB", "hadoop", "python" ], "users"
 : 1000 }
{ "_id" : ObjectId("54770fbb1d84f84de872ee6c"), "title" : "MongoDB class", "webs
ite" : "www.easylearning.guru", "comments" : [ { "user" : "user1", "msg" : "nice
", "like" : 100 } ] }
{ "_id" : ObjectId("5477126b1d84f84de872ee6d"), "title" : "A blog post", "headin
g" : "blog is about mongodb", "comments" : [ { "name" : "joe", "email" : "joe@ex
ample.com", "content" : "nice blog" }, { "name" : "mary", "email" : "mary@abc.co
m", "content" : "good post" } ] }
> _
```

Pretty() method : is used to display the result in a formatted way.

Syntax: db.Collection_Name.find().pretty()

db.mycollection.find().pretty()



```
> db.mycollection.find().pretty()
{ "_id" : ObjectId("5476db551d84f84de872ee68"), "name" : "sam" }
{
        "_id" : ObjectId("54770fbb1d84f84de872ee6b"),
        "title" : "MongoDB Training",
        "by" : "easylearning.guru",
        "courses" : [
                "mongoDB",
                "hadoop",
                "python"
        ],
        "users" : 1000
}
{
        "_id" : ObjectId("54770fbb1d84f84de872ee6c"),
        "title" : "MongoDB class",
        "website" : "www.easylearning.guru",
        "comments" : [
                {
                        "user" : "user1",
                        "msg" : "nice",
                        "like" : 100
                }
        ]
}
{
        "_id" : ObjectId("5477126b1d84f84de872ee6d"),
        "title" : "A blog post",
        "heading" : "blog is about mongodb",
        "comments" : [
                {
                        "name" : "joe",
                        "email" : "joe@example.com",
                        "content" : "nice blog"
                },
                {
                        "name" : "mary",
                        "email" : "mary@abc.com",
                        "content" : "good post"
                }
        ]
```

findOne() method:

findOne() is the method used to find only one document. It will return one document that satisfies the specified query criteria. If multiple documents satisfy the query, this method returns the first document according to natural order which reflects the order of documents on the disk.

Syntax: db.Collection_Name.findOne(<criteria>,<projection>)

Here, Criteria and projection are optional parameters.

Parameter	Description
Criteria	Specifies query selection criteria using *query operators*.
projection	Specifies the fields to return using *projection operators*. Omit this parameter to return all fields in the matching document.

The projection parameter takes a document of the following form:

{ field1 : <Boolean> , field2 : <Boolean> …..}

<Boolean> can be: **1 or true** to include and **0 or false** to exclude

The **findOne()** method always includes the *"_id"* field even if the field is not explicitly specified in the *projection* parameter.

```
> db.mycollection.findOne()
{ "_id" : ObjectId("5476db551d84f84de872ee68"), "name" : "sam" }
>
>
```

If you want to display a document, for example whose title is "A blog post" and we don't want to display its "_id". Then use the following command.

db.mycollection.findOne({"title" : "A blog post"},{"_id" : 0})

```
> db.mycollection.findOne({"title" : "A blog post"},{"_id" :0 })
{
        "title" : "A blog post",
        "heading" : "blog is about mongodb",
        "comments" : [
                {
                        "name" : "joe",
                        "email" : "joe@example.com",
                        "content" : "nice blog"
                },
                {
                        "name" : "mary",
                        "email" : "mary@abc.com",
                        "content" : "good post"
                }
        ]
}
>
```

How to update documents in a collection?

To update the documents in a collection we use update() and save() methods in MongoDB. update() method is used to update the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

Syntax for update() method:

db.Collection_Name.update(Selection_Criteria , Updated_Data)

Update specific field:

To update the **specific fields** in a document, mongodb provides update operators, such as **$set** used to modify the values and also it will create the field if the field does not exist.

db.mycoll.update({“name” : “mary” }, { $set : {“age” : 12 } })

This statement defines that update the document with name “mary” and set the value of age as 12.

```
> db.mycoll.update({"name" : "mary"}, { $set : { "age" : 12 } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> _
```

Update an embedded field:

To update any **field within an embedded document**, use the dot notation. When using the dot notation, enclose the whole dotted field names in quotes.

Here it will update the document with name “mary” and will set the value of embedded field “home” within the “address” filed.

```
> db.mycoll.update({"name" : "mary"}, { $set : { "address.home" : "America" } } )
>
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> _
```

Update multiple documents:

To **update the multiple documents**: by default update() method updates a single document. To update the multiple documents use “multi” option in the update() method.

```
> db.mycoll.find().pretty()
{ "_id" : ObjectId("5476f96b1d84f84de872ee69"), "name" : "sam", "age" : 23 }
{
        "_id" : ObjectId("5476f9ac1d84f84de872ee6a"),
        "name" : "mary",
        "age" : 12,
        "address" : {
                "home" : "America",
                "office" : "delhi"
        }
}
{ "_id" : ObjectId("547810339479fb86c70c478e"), "name" : "sam", "age" : 27 }
{ "_id" : ObjectId("5478103e9479fb86c70c478f"), "name" : "sam", "age" : 65 }
>
```

For example to update the “age” field in a document whose name is “sam”.

Use db.mycoll.update({“name” : “sam” }, { $set : {“age” : 33}}, {multi : true})

```
> db.mycoll.update({"name" : "sam"}, {$set : {"age" : 33}},{multi : true})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
>
```

Now you can see here that age field has been updated whose name is “sam”

```
> db.mycoll.find().pretty()
{ "_id" : ObjectId("5476f96b1d84f84de872ee69"), "name" : "sam", "age" : 33 }
{
        "_id" : ObjectId("5476f9ac1d84f84de872ee6a"),
        "name" : "mary",
        "age" : 12,
        "address" : {
                "home" : "America",
                "office" : "delhi"
        }
}
{ "_id" : ObjectId("547810339479fb86c70c478e"), "name" : "sam", "age" : 33 }
{ "_id" : ObjectId("5478103e9479fb86c70c478f"), "name" : "sam", "age" : 33 }
> _
```

To replace the entire document:

To replace the entire content of a document except for the _id field, pass an entirely new document as the second argument to update().
The replacement document can have different fields from the original document. In the replacement document, you can omit the _id field since the _id field is immutable. If you do include the _id field, it must be the same value as the existing value.

```
> db.mycoll.find().pretty()
{ "_id" : ObjectId("5476f96b1d84f84de872ee69"), "name" : "sam", "age" : 33 }
{
        "_id" : ObjectId("5476f9ac1d84f84de872ee6a"),
        "name" : "mary",
        "age" : 12,
        "address" : {
                "home" : "America",
                "office" : "delhi"
        }
}
{ "_id" : ObjectId("547810339479fb86c70c478e"), "name" : "sam", "age" : 33 }
{ "_id" : ObjectId("5478103e9479fb86c70c478f"), "name" : "sam", "age" : 33 }
>
>
```

For example you want to replace the document whose field name is “mary”.

```
>
> db.mycoll.update({"name" : "mary" }, {"name" : "joe", "salary" : 50000, "count
ry" : "india", "state":"delhi"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
>
```

So, here the whole document is replaced with some new values and the fields like address and age does no longer exists in the new document. It is just the replacement of the document within the same “_id” field.

```
> db.mycoll.find().pretty()
{ "_id" : ObjectId("5476f96b1d84f84de872ee69"), "name" : "sam", "age" : 33 }
{
        "_id" : ObjectId("5476f9ac1d84f84de872ee6a"),
        "name" : "joe",
        "salary" : 50000,
        "country" : "india",
        "state" : "delhi"
}
{ "_id" : ObjectId("547810339479fb86c70c478e"), "name" : "sam", "age" : 33 }
{ "_id" : ObjectId("5478103e9479fb86c70c478f"), "name" : "sam", "age" : 33 }
>
```

To replace an existing document we have a method called save()

Save() method replaces the existing document with the new document passed in save() method.
For example we had this document:

```
{
        "_id" : ObjectId("5497ccca33c573c4679d9998"),
        "item" : "ZYT1",
        "details" : {
                "model" : "14Q1",
                "manufacturer" : "ABC Company"
        },
        "stock" : [
                {
                        "size" : "S",
                        "qty" : 5
                },
                {
                        "size" : "M",
                        "qty" : 5
                }
        ],
        "category" : "houseware"
}
```

And we want to replace it with a new document. Then pass the following query:

```
> db.sales.save({"_id" : ObjectId("5497ccca33c573c4679d9998"), "category" : "med
icine"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Now the document is being replaced with the new document.

```
{ "_id" : ObjectId("5497ccca33c573c4679d9998"), "category" : "medicine" }
{
```

How to drop a collection?

If you want to drop a collection from a database, use db.collection.drop(), where collection is the name of the collection.
Example: check the available collections in your database “database”.

```
> use database
switched to db database
> show collections
data
doc
system.indexes
>
>
```

If you want to drop the “doc” collection from the “database” database.
Use “db.doc.drop()” and you will get a true value for that.

```
> db.doc.drop()
true
>
```

Now again check the list of collections in your database and “doc” collection has been dropped.

```
> show collections
data
system.indexes
>
```

How to drop a Database?

If you want to drop a database, then **db.dropDatabase()** command is used to drop an existing database.
Example: First, check the list available databases by using the command “**show dbs**”

```
> show dbs
DATABASE_NAME   0.078GB
admin           0.078GB
database        0.078GB
local           0.078GB
mydb            0.078GB
>
```

To drop an existing database first go to that database which you want to delete by using “use” command. If you want to delete database <database>, then dropDatabase() command would be as follows:

```
> use database
switched to db database
> db.dropDatabase()
{ "dropped" : "database", "ok" : 1 }
>
```

Now, check the list of databases.

```
> show dbs
DATABASE_NAME   0.078GB
admin           0.078GB
local           0.078GB
mydb            0.078GB
>
```

Remove() method in mongoDB

If you want to remove the documents from a collection, then db.collection.remove() is used to remove the documents. You can remove all documents from a collection, remove all documents that match a condition, or limit the operation to remove just a single document.

Syntax: db.collection.remove(<query> , <justOne>)

Parameter	Type	Description
Query	Document	Specifies deletion criteria using query opeartors. (query opeartors provide ways to locate data within the database). To delete all documents in a collection, pass an empty document({}).
justOne (optional)	boolean	To limit the deletion to just one document, set to true. Omit to use the default value of false and delete all documents matching the deletion criteria.

Remove all documents:
To remove all documents from a collection, pass an empty query document {} to the remove() method.
For example, **db.mycollect.remove({ })** command will remove all the documents from the mycollect collection.
To remove all documents from a collection, it may be more efficient to use the drop() method to drop the entire collection, including the indexes, and then recreate the collection and rebuild the indexes.

Remove documents that match a condition:
To remove the documents that match a deletion criteria, call the remove() method with the <query> parameter.
Following example removes all documents from the **mycollect** collection where the **name** field equals to **joy.**
db.mycollect.remove({“name” : “joy”})

For large deletion operations, it may be more efficient to copy the documents that you want to keep to a new collection and then use drop() on the original collection.

Remove a single document that matches a condition:

To remove a single document, call the remove() method with the justOne parameter set to true or1.
Following example removes one document from mycollect collection where the name field equals mary:
db.mycollect.remove({ “name” : “mary” }, 1)

Ordered and Unordered Bulk() operations:

Bulk write operations can be either ordered or unordered. With an ordered list of operations, MongoDB executes the operations serially. If an error occurs during the processing of one of the write operations, MongoDB will return without processing any remaining write operations in the list. With an unordered list of operations, MongoDB can execute the operations in parallel. If an error occurs during the processing of one of the write operations, MongoDB will continue to process remaining write operations in the list.

To use Bulk() methods:
Initialize a list of operations using either db.collection.initializeOrderedBulkOp() or db.collection.initializeUnorderedBulkOp().

db.collection.initializeOrderedBulkOp():

Example:

```
var bulk = db.users.initializeOrderedBulkOp();
bulk.insert ( { user : “abc”, status : “A”, points : 0  } );
bulk.insert ( { user : “ijk”, status : “A”, points : 0  } );
bulk.insert ( { user : “mop”, status : “P”, points : 100  } );
bulk.find( { status : “D” }).remove();
bulk.find({ status : “P” }).update( { $set : {comment: “Pending” }});
bulk.execute();
```

Here we have initialized order bulk operations:

```
> var bulk = db.users.initializeOrderedBulkOp();
> bulk.insert( { user: "abc123", status: "A", points: 0 } );
> bulk.insert( { user: "ijk123", status: "A", points: 0 } );
> bulk.insert( { user: "mop123", status: "P", points: 0 } );
> bulk.find( { status: "D" } ).remove();
> bulk.find( { status: "P" } ).update( { $set: { comment: "Pending" } } );
> bulk.execute();
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 1,
        "nModified" : 1,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

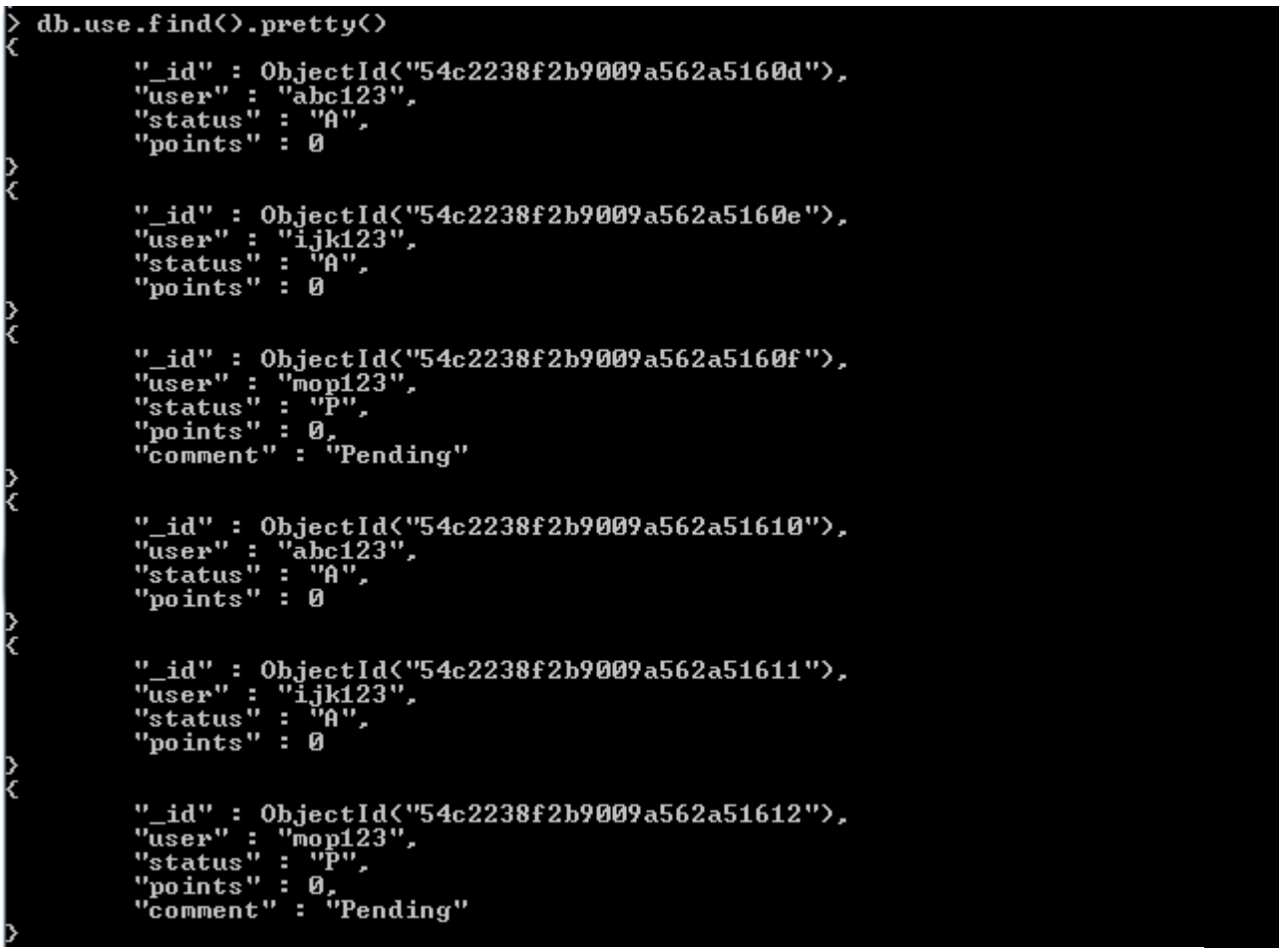
Here 3 documents are written, 1 is matched and modified, and nothing is removed.

db.collection.initializeUnorderedBulkOp():

Let’s take another example, where we initialize unordered bulk operations:

```
> var bulk = db.use.initializeUnorderedBulkOp();
> bulk.insert( { user: "abc123", status: "A", points: 0 } );
> bulk.insert( { user: "ijk123", status: "A", points: 0 } );
> bulk.insert( { user: "mop123", status: "P", points: 0 } );bulk.insert( { user:
 "abc123", status: "A", points: 0 } );
> bulk.insert( { user: "ijk123", status: "A", points: 0 } );
> bulk.insert( { user: "mop123", status: "P", points: 0 } );
> bulk.find( { status: "D" } ).remove();
> bulk.find( { status: "P" } ).update( { $set: { comment: "Pending" } } );
> bulk.execute();
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 6,
        "nUpserted" : 0,
        "nMatched" : 2,
        "nModified" : 2,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

Then the output is:



Executing an ordered list of operators on a sharded collection will generally be slower than executing unordered list since with an ordered list, each operations must wait for the previous operations to finish.