CRUD Operations

Objectives

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

- Working with Arrays
- MongoDB Primary Key
- Inserting Documents
- Finding Documents
- Updating Documents
- Deleting Documents

MongoDB Adding Array

Add MongoDB Array using insert()

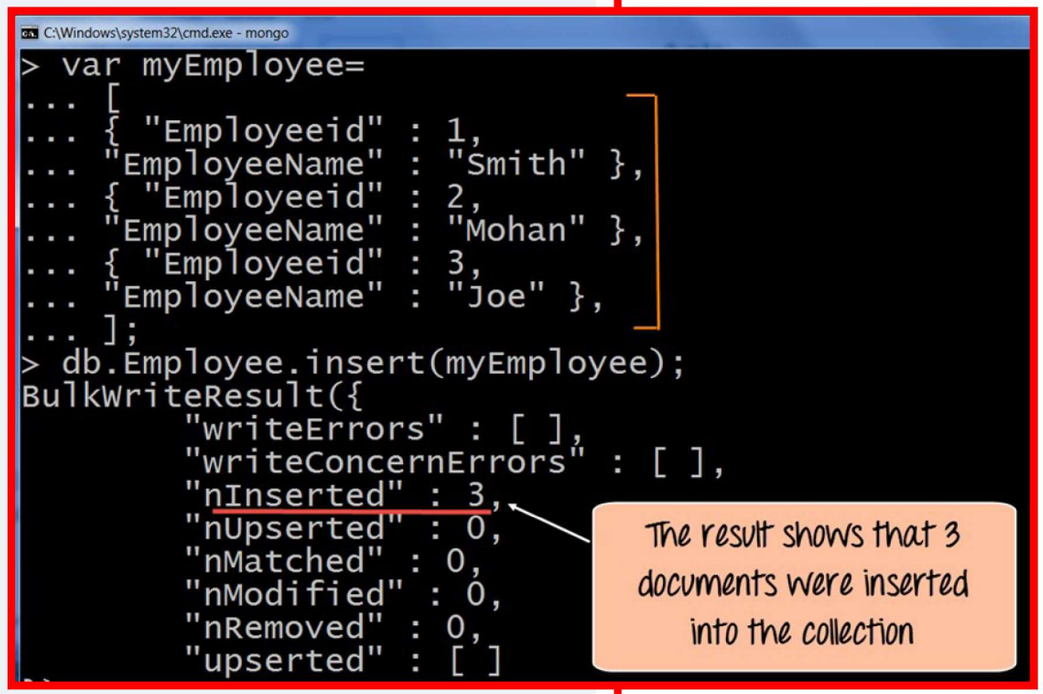
- The “insert” command can also be used to insert multiple documents into a collection at one time. The below code example can be used to insert multiple documents at a time.
  1. Create a JavaScript variable called myEmployee to hold the array of documents
  2. Add the required documents with the Field Name and values to the variable
  3. Use the insert command to insert the array of documents into the collection

```
var myEmployee=
        [

                {
                        "Employeeid" : 1,
                        "EmployeeName" : "Smith"
                },
                {
                        "Employeeid"    : 2,
                        "EmployeeName" : "Mohan"
                },
                {
                        "Employeeid"    : 3,
                        "EmployeeName" : "Joe"
                },

        ];

        db.Employee.insert(myEmployee);
```



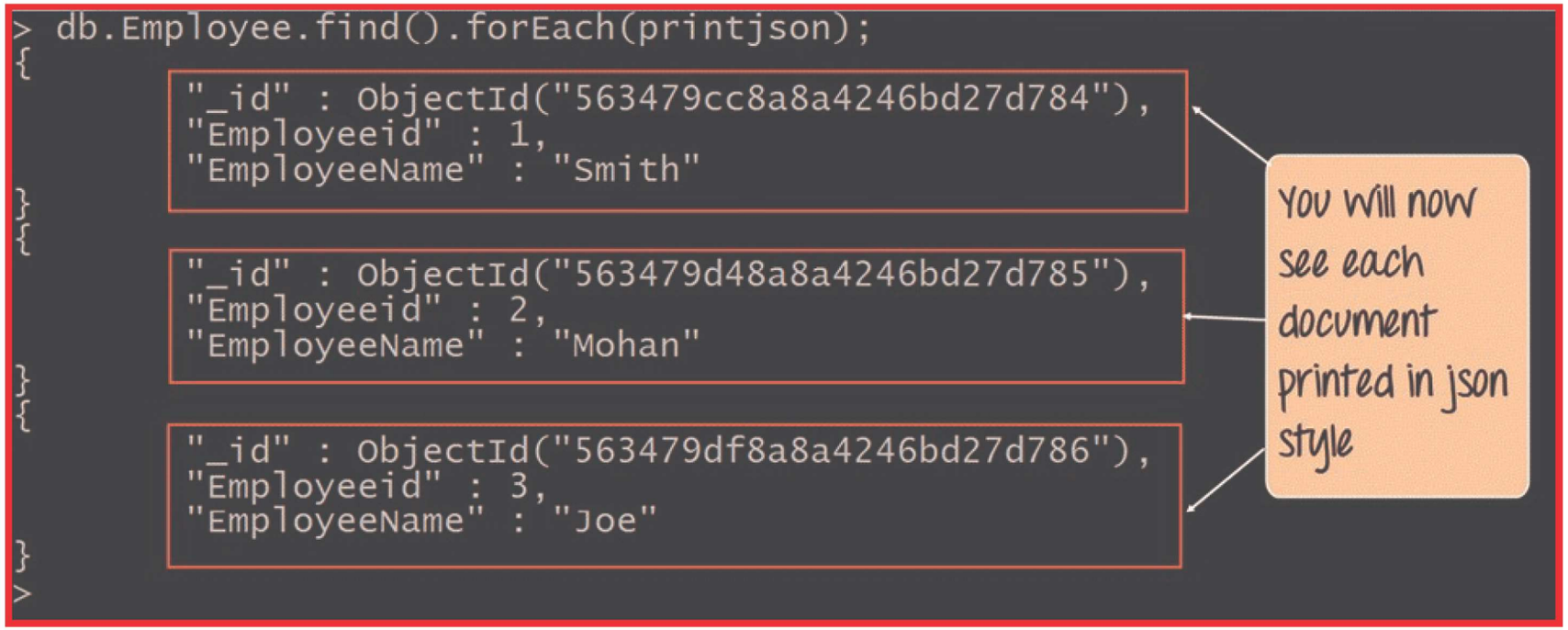
Printing in JSON format

- JSON is a format called **JavaScript Object Notation**, and is just a way to store information in an organized, easy-to-read manner. In our further examples, we are going to use the JSON print functionality to see the output in a better format.

1. The first change is to append the function called for Each() to the find() function. What this does is that it makes sure to explicitly go through each document in the collection. In this way, you have more control of what you can do with each of the documents in the collection.
2. The second change is to put the printjson command to the forEach statement. This will cause each document in the collection to be displayed in JSON format.

```
db.Employee.find().forEach(printjson)
```

Output

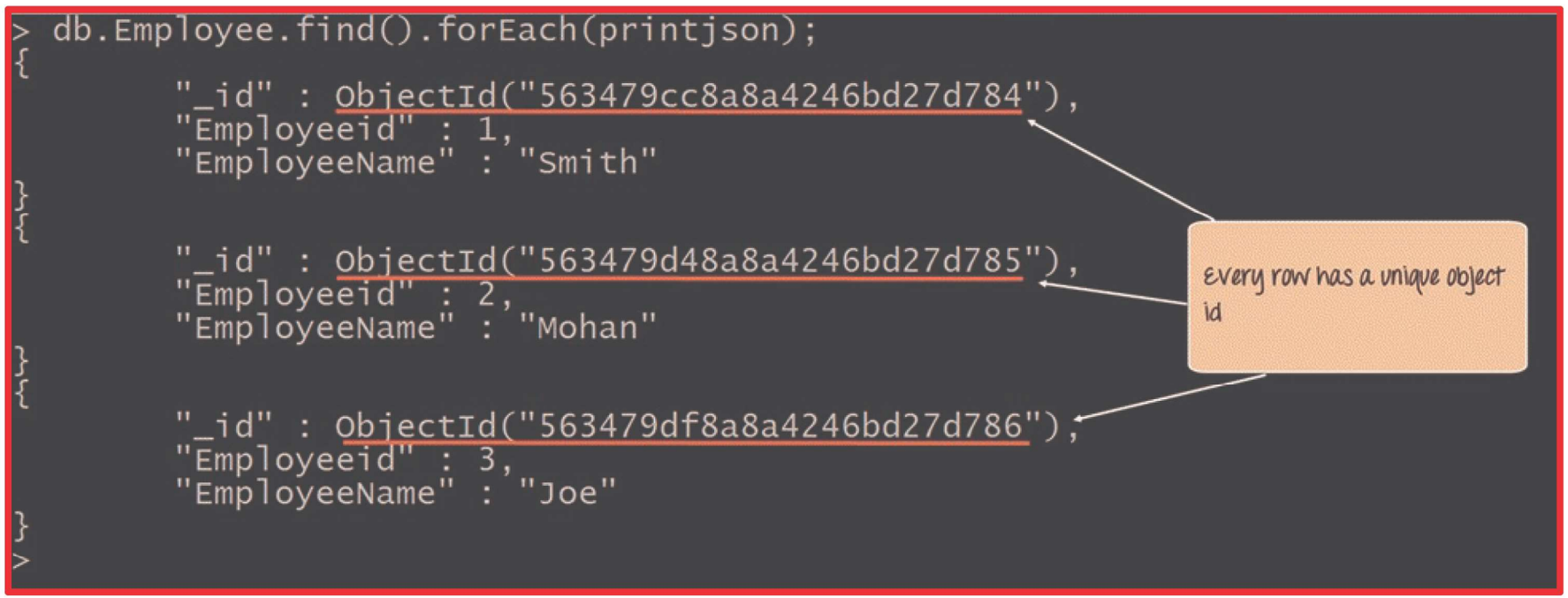

db.Employee.find().forEach(printjson);
"_id" : ObjectId("563479cc8a8a4246bd27d784"),
"Employeeid" : 1,
"EmployeeName" : "Smith"
"_id" : ObjectId("563479d48a8a4246bd27d785"),
"Employeeid" : 2,
"EmployeeName" : "Mohan"
"_id" : ObjectId("563479df8a8a4246bd27d786"),
"Employeeid" : 3,
"EmployeeName" : "Joe"
You will now see each document printed in json style

MongoDB Primary Key

Mongodb Primary Key : to set _id field with ObjectId()

What is Primary Key in MongoDB?

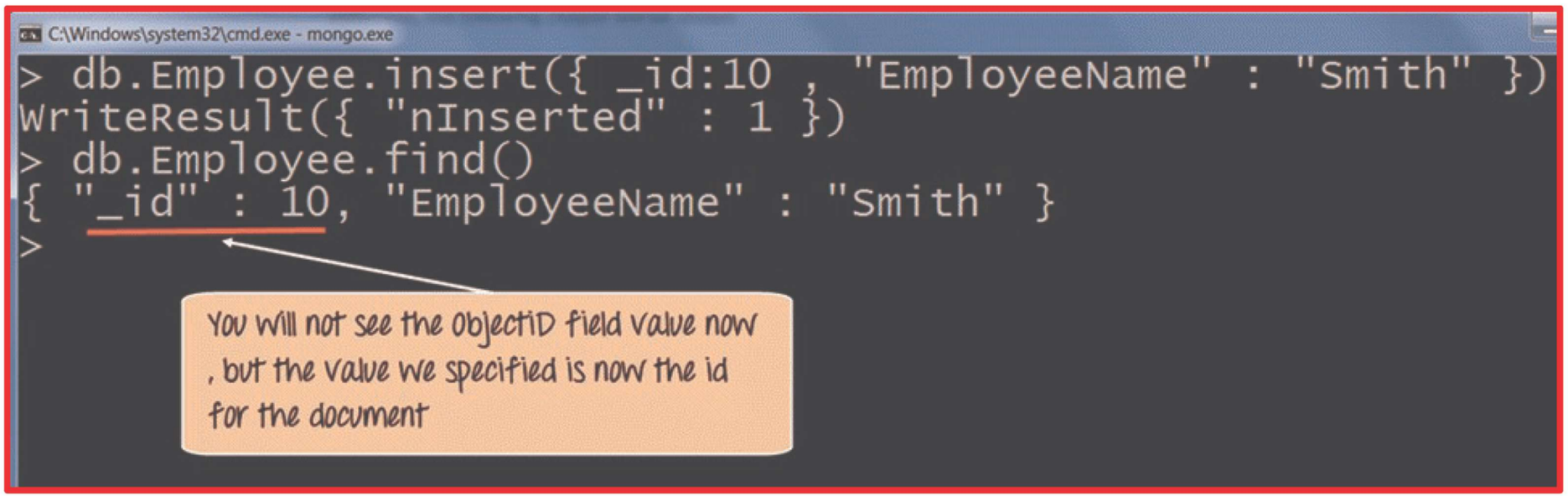
- In MongoDB, _id field as the primary key for the collection so that each document can be uniquely identified in the collection. The _id field contains a unique ObjectID value.
- By default when inserting documents in the collection, if you don't add a field name with the _id in the field name, then MongoDB will automatically add an Object id field



- When you query the documents in a collection, you can see the ObjectId for each document in the collection.
- If you want to ensure that MongoDB does not create the _id Field when the collection is created and if you want to specify your own id as the _id of the collection, then you need to explicitly define this while creating the collection.
- When explicitly creating an id field, it needs to be created with _id in its name.

```
db.Employee.insert({_id:10, "EmployeeName": "Smith"})
```

- We are assuming that we are creating the first document in the collection and hence in the above statement while creating the collection, we explicitly define the field _id and define a value for it.



MongoDB Insert Documents

- In MongoDB, the **db.collection.insert()** method is used to add or insert new documents into a collection in your database.
- **Upsert**
  - There are also two methods "db.collection.update()" method and "db.collection.save()" method used for the same purpose. These methods add new documents through an operation called upsert.
  - Upsert is an operation that performs either an update of existing document or an insert of new document if the document to modify does not exist.

Syntax and Example

- Syntax

```
db.collectionName.insert(document)
```

- Example

```
db.Courses.insert(
  {
    courseName: "Java",
    details: {
       duration: "6 months",
       Trainer: “Santhosh Ashok"
    },
    Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
    category: "Programming Language"
  }
)
```

MongoDB insert multiple documents

- If you want to insert multiple documents in a collection, you have to pass an array of documents to the db.collection.insert() method.
- **Create an array of documents**

```
db.Courses.insert(AllCourses)
```

```
var AllCourses =
   [
    {
      Course: "Java",
      details: { Duration: "6 months", Trainer: “Santhosh Ashok" },
      Batch: [ { size: "Medium", qty: 25 } ],
      category: "Programming Language"
    },
    {
      Course: ".Net",
      details: { Duration: "6 months", Trainer: “Deepa" },
      Batch: [ { size: "Small", qty: 5 }, { size: "Medium", qty: 10 }, ],
      category: “Framework"
    },
    {
      Course: "Web Designing",
      details: { Duration: "3 months", Trainer: “Laxman" },
      Batch: [ { size: "Small", qty: 5 }, { size: "Large", qty: 10 } ],
      category: “Front End"
    }
   ];
```

```
BulkWriteResult({
   "writeErrors" : [ ],
   "writeConcernErrors" : [ ],
   "nInserted" : 3,
   "nUpserted" : 0,
   "nMatched" : 0,
   "nModified" : 0,
   "nRemoved" : 0,
   "upserted" : [ ]
})
```

Insert multiple documents with Bulk

In its latest version of MongoDB (MongoDB 2.6) provides a Bulk() API that can be used to perform multiple write operations in bulk.

- **Initialize a bulk operation builder**
- First initialize a bulk operation builder for the collection **Courses**.

```
var bulk = db.Courses.initializeUnorderedBulkOp();
```

- This operation returns an unorder operations builder which maintains a list of operations to perform .

Add insert operations to the bulk object

```
bulk.insert(
  {
    course: “Java",
    details: {
      duration: "6 months",
      Trainer: “Santhosh Ashok"
    },
    Batch: [ { size: "Small", qty: 15 }, { size: "Medium", qty: 25 } ],
    category: "Programming language"
  }
);
```

Execute the bulk operation

- Call the execute() method on the bulk object to execute the operations in the list.

```
bulk.execute();
```

- After the successful insertion of the documents, this method will return a **BulkWriteResult** object with its status.

```
BulkWriteResult({
   "writeErrors" : [ ],
   "writeConcernErrors" : [ ],
   "nInserted" : 1,
   "nUpserted" : 0,
   "nMatched" : 0,
   "nModified" : 0,
   "nRemoved" : 0,
   "upserted" : [ ]
})
```

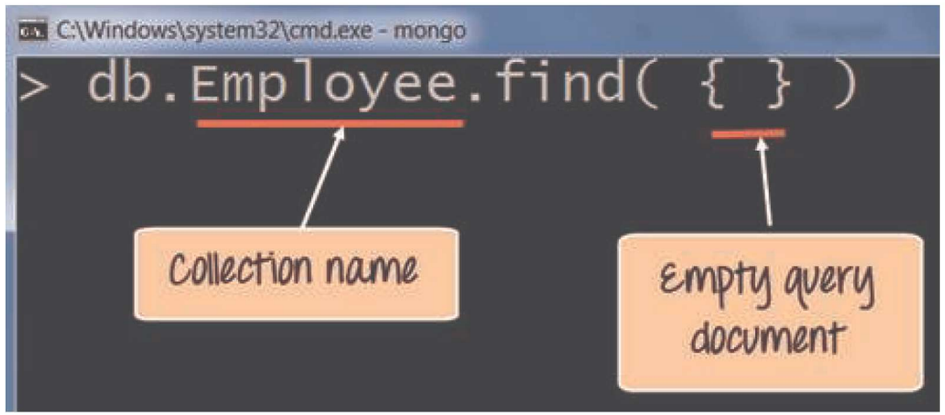
MongoDB Querying Documents

MongoDB Query Document: db.collection.find()

- The method of fetching or getting data from a MongoDB database is carried out by using MongoDB queries. While performing a query operation, one can also use criteria's or conditions which can be used to retrieve specific data from the database.
- MongoDB provides a function called **db.collection.find()** which is used for retrieval of documents from a MongoDB database.
- The db.collection.find() method reads operations in mongoDB shell and retrieves documents containing all their fields.
- **Note: You can also restrict the fields to return in the retrieved documents by using some specific queries. For example: you can use the db.collection.findOne() method to return a single document. It works same as the db.collection.find() method with a limit of 1.**

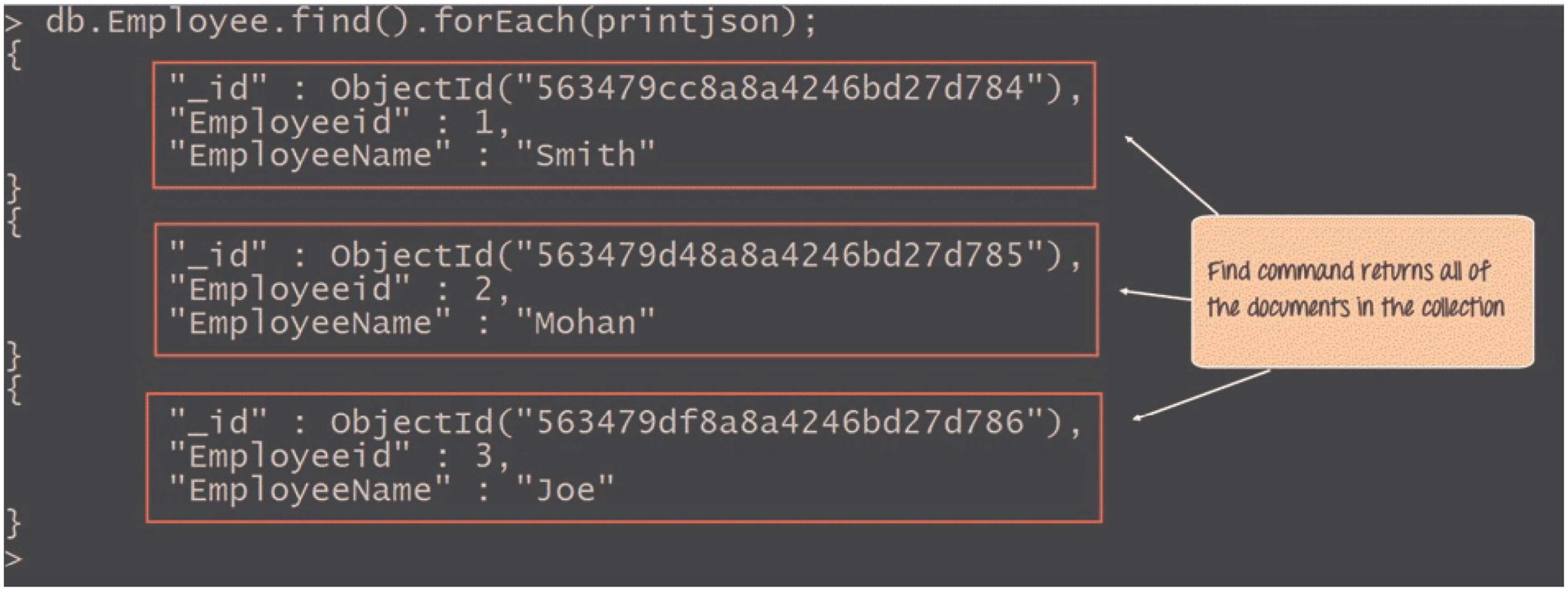
MongoDB Basic Query Operations

- The basic MongoDB query operators cover the simple operations such as getting all of the documents in a MongoDB collection.
- All of our code will be run in the MongoDB JavaScript command shell.



1. Employee is the collection name in the MongoDB database
2. The MongoDB find query is an in-built function which is used to retrieve the documents in the collection.

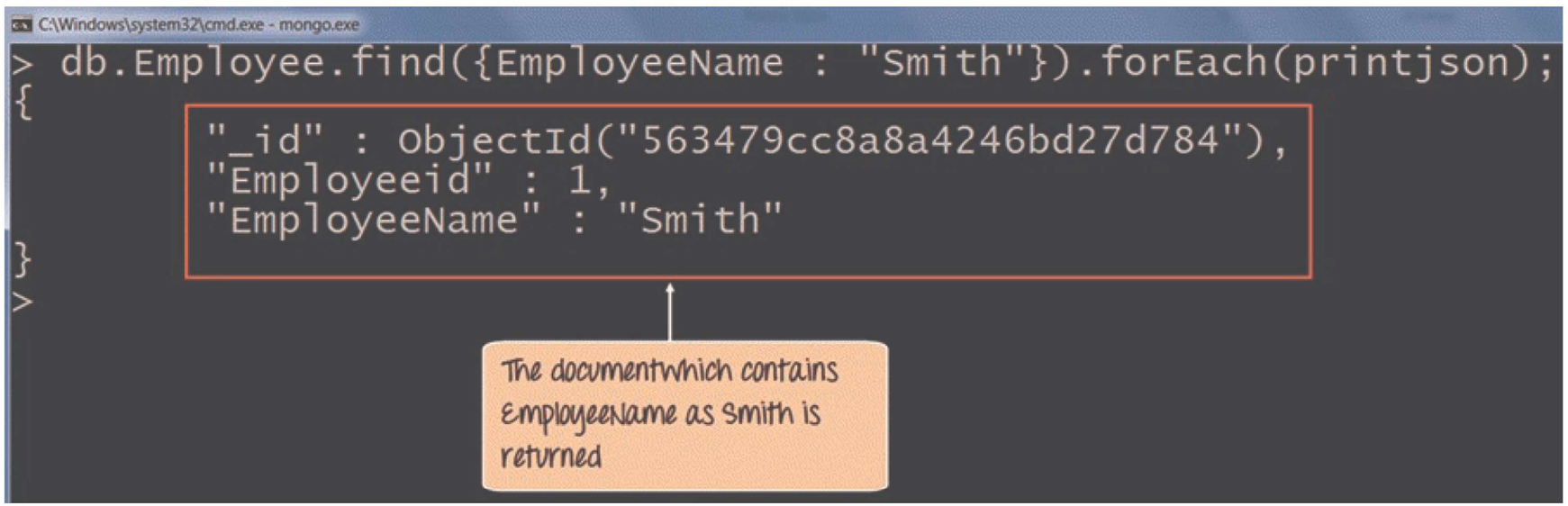
Output


> db.Employee.find().forEach(printjson);
{
"_id" : ObjectId("563479cc8a8a4246bd27d784"),
"Employeeid" : 1,
"EmployeeName" : "Smith"
}
{
"_id" : ObjectId("563479d48a8a4246bd27d785"),
"Employeeid" : 2,
"EmployeeName" : "Mohan"
}
{
"_id" : ObjectId("563479df8a8a4246bd27d786"),
"Employeeid" : 3,
"EmployeeName" : "Joe"
}
Find command returns all of the documents in the collection

MongoDB Query Example

- Here we want to find for an Employee whose name is “Smith” in the collection , hence we enter the filter criteria as EmployeeName : “Smith”

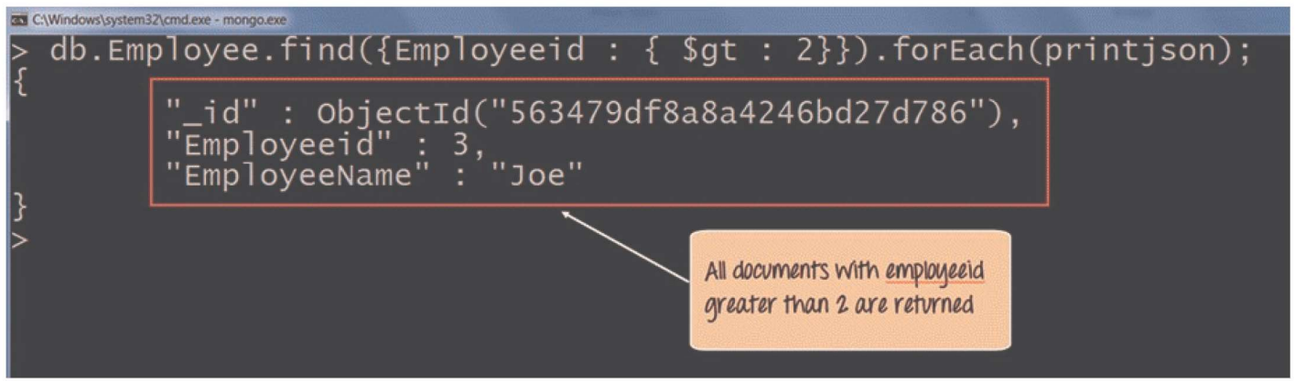
```
db.Employee.find({EmployeeName: “Smith”}).forEach(printjson);
```



MongoDB Query Example – 2

- let's take a look at another code scenario which makes use of the greater than search criteria. When this criteria is included, it actually searches those documents where the value of the field is greater than the specified value.
- Here we want to find for all Employee's whose id is greater than 2. The $gt is called a query selection operator, and what is just means is to use the greater than expression.

```
db.Employee.find({EmployeeId: {$gt:2}}).forEach(printjson);
```



The pretty() Method

- To display the results in a formatted way, you can use pretty() method.

```
db.Employee.find({EmployeeName: "Smith"}).pretty();
```

The findOne() method

- Apart from the find() method, there is **findOne()** method, that returns only one document.

```
db.Employee.findOne();
```

RDBMS Where Clause Equivalents in MongoDB

- To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eq;<value>}}	db.mycol.find({"by":“Bangalore"}).pretty()	where by = ‘Bangalore'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50

Operation	Syntax	Example	RDBMS Equivalent
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50
Values in an array	{<key>:{$in:[<value1>, <value2>,……<valueN>]}}	db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in :["Raj", "Ram", "Raghu"]
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty()	Where name values is not in the array :["Ramu", "Raghav"] or, doesn't exist at all

AND in MongoDB

- To query documents based on the AND condition, you need to use $and keyword.
- Syntax :

```
db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
```

- Example

```
db.mycol.find({$and:[{"Name":"Rahul"},
{"Location": "Bangalore"}]}).pretty()
```

OR in MongoDB

- To query documents based on the OR condition, you need to use **$or** keyword.

Syntax

```
>db.mycol.find(
   {
      $or: [
         {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

Example

```
db.mycol.find({$or:[{"Name":"Rahul"},
{"Location": "Bangalore"}]}).pretty()
```

Using AND and OR Together

- To query documents based on the OR condition, you need to use **$or** keyword.

Syntax

```
>db.mycol.find({key1: value1
   {
      $or: [
         {key2: value2}, {key3:value3}
      ]
   }
).pretty()
```

Example

```
db.mycol.find({"DepartmentId": {$gt:10}, $or: [{"DepartmentName":
"Administration"},
   {"Location": "1700"}]}).pretty()
```

NOR in MongoDB

- To query documents based on the NOT condition, you need to use $not keyword.
- **Syntax**

```
>db.COLLECTION_NAME.find(
        {
            $not: [
                    {key1: value1}, {key2:value2}
            ]
        }
)
```

Eg :

```
[
    {
        First_Name: "Radhika",
        Last_Name: "Sharma",
        Age: "26",
        e_mail: "radhika_sharma.123@gmail.com",
        phone: "9000012345"
    },
    {
        First_Name: "Rachel",
        Last_Name: "Christopher",
        Age: "27",
        e_mail: "Rachel_Christopher.123@gmail.com",
        phone: "9000054321"
    },
    {
        First_Name: "Fathima",
        Last_Name: "Sheik",
        Age: "24",
        e_mail: "Fathima_Sheik.123@gmail.com",
        phone: "9000054321"
    }
]
```

```
db.empDetails.find(
    {
        $nor:[

            {"First_Name": "Radhika"},
            {"Last_Name": "Christopher"}
        ]
    }
).pretty()
{
    "_id" : ObjectId("5dd631f270fb13eec3963bef"),
    "First_Name" : "Fathima",
    "Last_Name" : "Sheik",
    "Age" : "24",
    "e_mail" : "Fathima_Sheik.123@gmail.com",
    "phone" : "9000054321"
}
```

NOT in MongoDB

- To query documents based on the NOT condition, you need to use $not keyword

- **Syntax**

```
>db.COLLECTION_NAME.find(
          {
               $NOT: [
                      {key1: value1}, {key2:value2}
               ]
          }
).pretty()
```

Output

```
> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )
{
        "_id" : ObjectId("5dd6636870fb13eec3963bf7"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Age" : "24",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

MongoDB Updating Documents

MongoDB Update() Document

- MongoDB provides the update() command to update the documents of a collection. To update only the documents you want to update, you can add a criteria to the update statement so that only selected documents are updated.
- The basic parameters in the command is a condition for which document needs to be updated, and the next is the modification which needs to be performed.

MongoDB Update() Method

- MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.
- Syntax

```
db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

Steps to Update the Document

Step 1) Issue the update command

Step 2) Choose the condition which you want to use to decide which document needs to be updated. In our example, we want to update the document which has the Employee id 22.

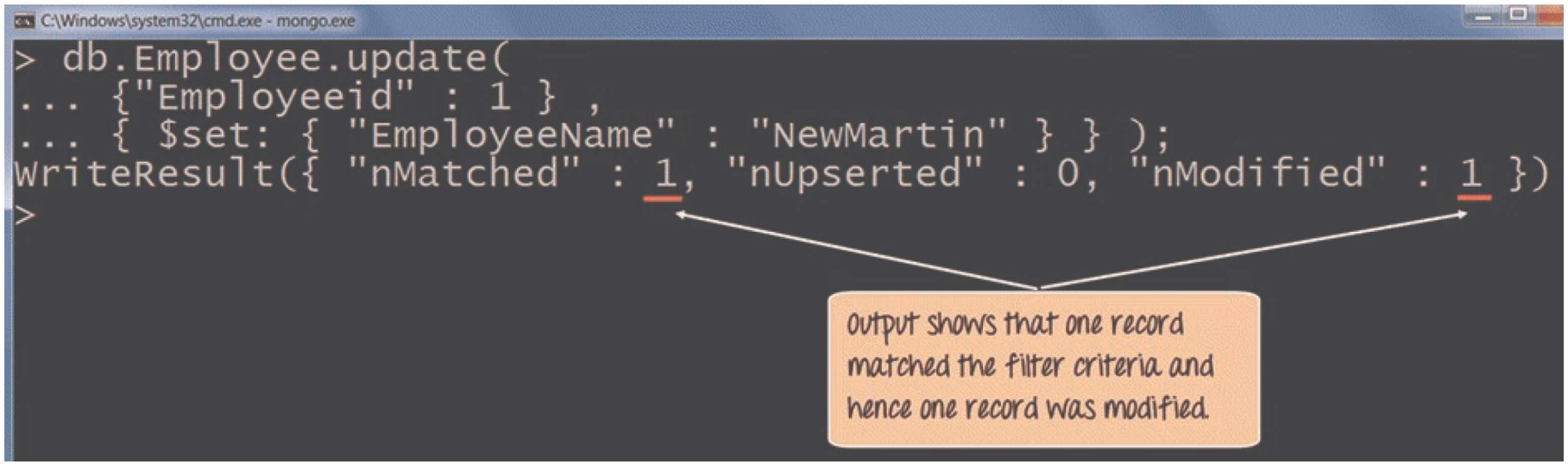
Step 3) Use the set command to modify the Field Name

Step 4) Choose which Field Name you want to modify and enter the new value accordingly.

- Command

```
db.Employee.update(
{"Employeeid" : 1},
{$set: { "EmployeeName" : "NewMartin"}});
```

- Output



Updating Multiple Values

- To ensure that multiple/bulk documents are updated at the same time in MongoDB you need to use the multi option because otherwise by default only one document is modified at a time.
- The example shows how to update many documents.
- In this example, we are going to first find the document which has the Employee id as “1” and change the Employee name from “Martin” to “NewMartin”

Steps Involved

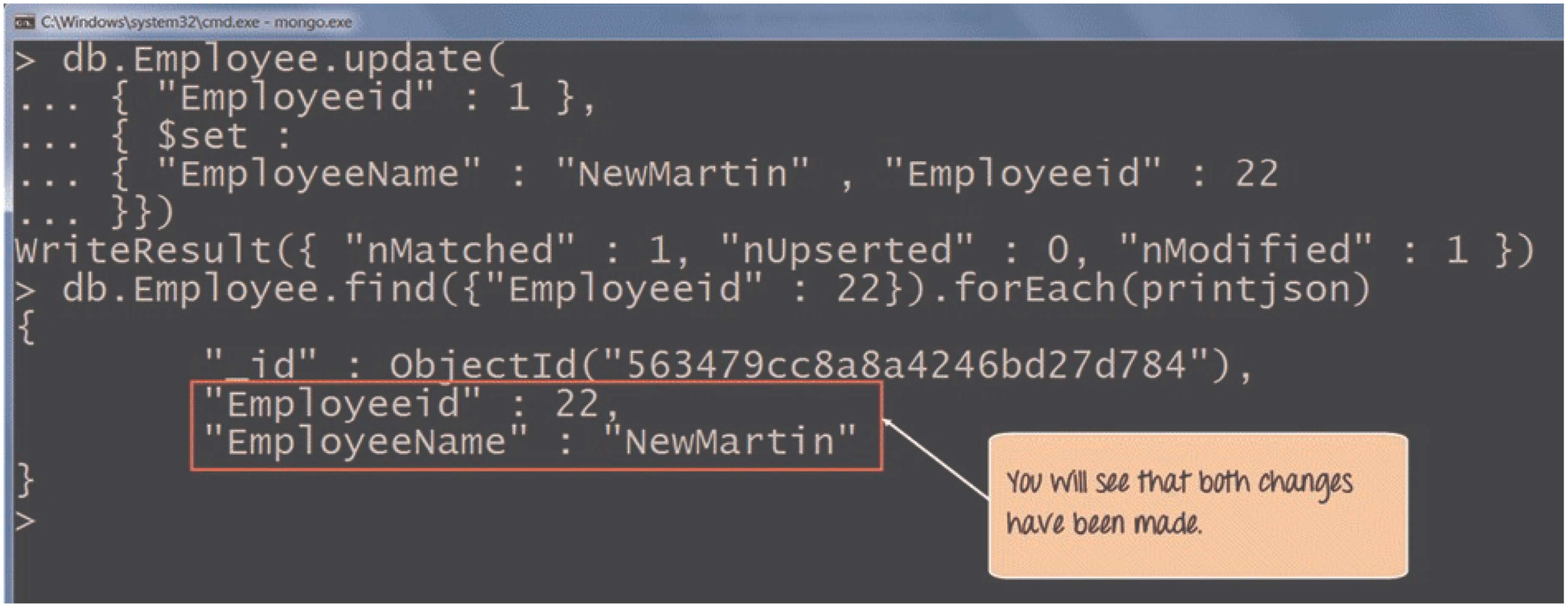
Step 1) Issue the update command

Step 2) Choose the condition which you want to use to decide which document needs to be updated. In our example, we want the document which has the Employee id of “1” to be updated.

Step 3) Choose which Field Name’s you want to modify and enter their new value accordingly.

- Command

```
db.Employee.update
(
        {
            Employeeid : 1
        },
        {
            $set :
            {
                "EmployeeName" : "NewMartin",
                "Employeeid" : 22
            }
        }
)
```


C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.update(
... { "Employeeid" : 1 },
... { $set :
... { "EmployeeName" : "NewMartin" , "Employeeid" : 22
... }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Employee.find({"Employeeid" : 22}).forEach(printjson)
{
"_id" : ObjectId("563479cc8a8a4246bd27d784"),
"Employeeid" : 22,
"EmployeeName" : "NewMartin"
}
>
You will see that both changes have been made.

- By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
db.mycol.update({'LocationID':'1700'},
   {$set:{'LocationID':'1800'}},{multi:true})
```

MongoDB Save() Method

- The **save()** method replaces the existing document with the new document passed in the save() method.
- Syntax :

```
db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

MongoDB findOneAndUpdate() method

- The **findOneAndUpdate()** method updates the values in the existing document.

Syntax

```
db.COLLECTION_NAME.findOneAndUpdate(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

Example

- Assume we have created a collection named empDetails and inserted three documents in it

```
db.empDetails.insertMany(
    [
        {
            First_Name: "Radhika",
            Last_Name: "Sharma",
            Age: "26",
            e_mail: "radhika_sharma.123@gmail.com",
            phone: "9000012345"
        },
        {
            First_Name: "Rachel",
            Last_Name: "Christopher",
            Age: "27",
            e_mail: "Rachel_Christopher.123@gmail.com",
            phone: "9000054321"
        },
        {
            First_Name: "Fathima",
            Last_Name: "Sheik",
            Age: "24",
            e_mail: "Fathima_Sheik.123@gmail.com",
            phone: "9000054321"
        }
    ]
)
```

This updates the age and email values of the document with name

```
db.empDetails.findOneAndUpdate(
    {First_Name: 'Radhika'},
    { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
)
{
    "_id" : ObjectId("5dd6636870fb13eec3963bf5"),
    "First_Name" : "Radhika",
    "Last_Name" : "Sharma",
    "Age" : "30",
    "e_mail" : "radhika_newemail@gmail.com",
    "phone" : "9000012345"
}
```

MongoDB updateOne() method

- This methods updates a single document which matches the given filter.
- **Syntax**

```
db.COLLECTION_NAME.updateOne(<filter>, <update>)
```

- **Example**

```
db.empDetails.updateOne(
        {First_Name: 'Radhika'},
        { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
)
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
```

MongoDB updateMany() method

- The updateMany() method updates all the documents that matches the given filter.
- **Syntax**

```
db.COLLECTION_NAME.updateMany(<filter>, <update>)
```

- **Example**

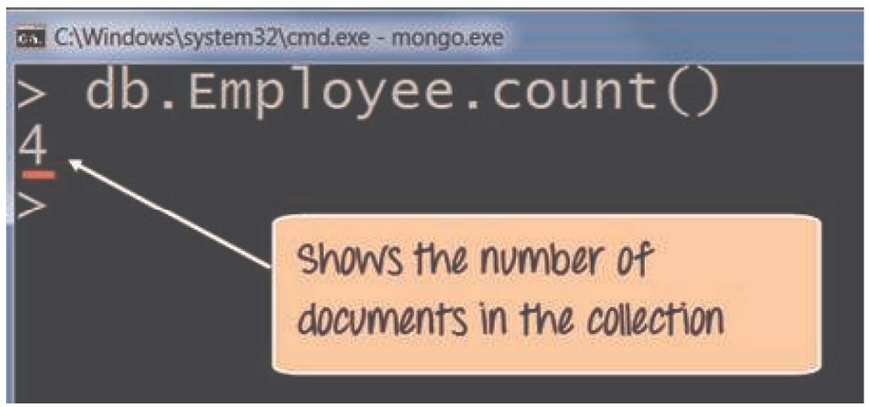
```
db.empDetails.updateMany(
        {Age:{ $gt: "25" }},
        { $set: { Age: '00'}}
)
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

MongoDB Count
Deleting Documents

MongoDB Count() & Remove() Functions

- The concept of aggregation is to carry out a computation on the results which are returned in a query.
- For example, suppose you wanted to know what is the count of documents in a collection as per the query fired, then MongoDB provides the count() function.

```
db.Employee.count();
```

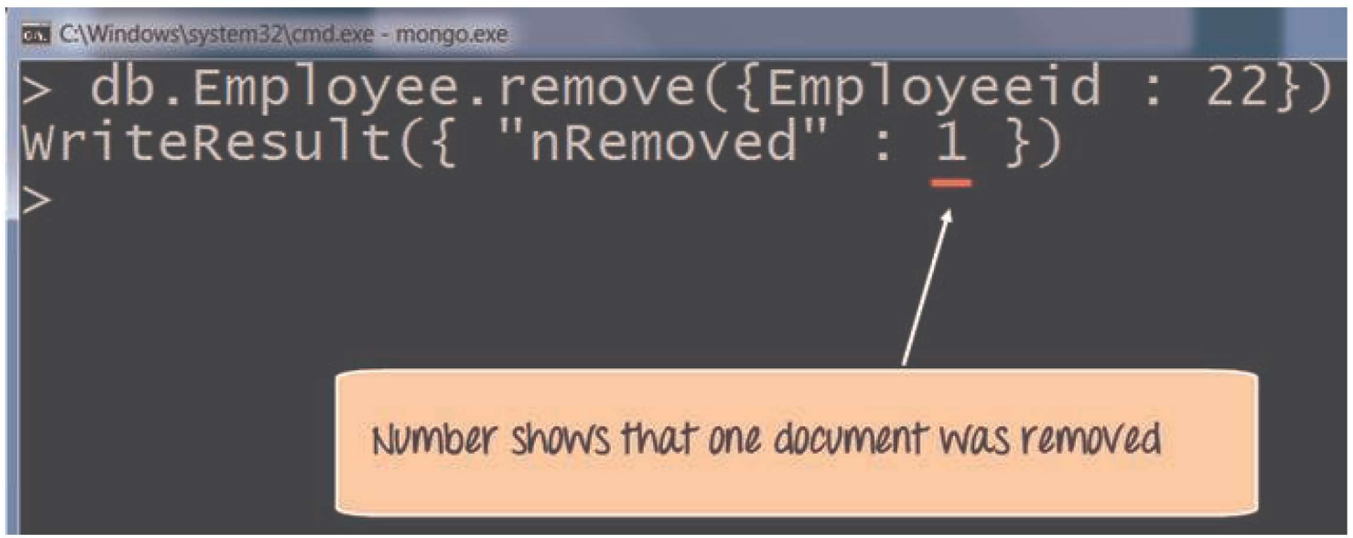


Deleting Documents

- The update operations allow one to modify existing data, and the remove operations allow the deletion of data from a collection.
- In MongoDB, the **db.collection.remove ()** method is used to remove documents from a collection. Either all of the documents can be removed from a collection or only those which matches a specific condition.
- If you just issue the remove command, all of the documents will be removed from the collection.

```
db.Employee.remove({EmployeeId: 22});
```

The above code use the remove function and specifies the criteria which in this case is to remove the documents which have the Employee id as 22.



- The remove() method works on two parameters.

1. Deletion criteria: With the use of its syntax you can remove the documents from the collection.

2. JustOne: It removes only one document when set to true or 1.

Remove all documents

- If you want to remove all documents from a collection, pass an empty query document {} to the remove() method. The remove() method does not remove the indexes.

```
db.Employee.remove({});
```

Remove a single document that match a condition

- If you want to remove a document that match a specific condition, call the remove() method with the <query> parameter.

```
db.Employee.remove({type : "programming language" }, 1});
```

Summary

In this lesson, you should have learned how to:

- Working with Arrays
- MongoDB Primary Key
- Inserting Documents
- Finding Documents
- Updating Documents
- Deleting Documents