MongoDB – Update documents

- Updating the database is another essential operation that every database system must have.
- If you have put some data in your database and want to alter the values of any particular document, you need to use the update operation.
- We learn about how to update data within MongoDB database.
- The update operation in a database is used to change or update values in a document. MongoDB makes use of the **update()** method for updating the documents within a MongoDB collection.

MongoDB – Update documents

- For updating any specific documents, a new criterion can be incorporated with the update statement, which will only update the selected documents.
- You have to put some specific condition in the form of the parameter to update the document in MongoDB. Here is a stepwise representation of how this can be performed:
-

  Make use of the update() method.

MongoDB – Update documents

- Prefer the circumstance that you wish to implement for deciding which document needs an update in their database.
- Let us assume an example where you want to update your document which is having an id 4.
- Then make use of the set command for modifying the Field Name.
- Select which Field Name you wish for modifying and go into the new value consequently.

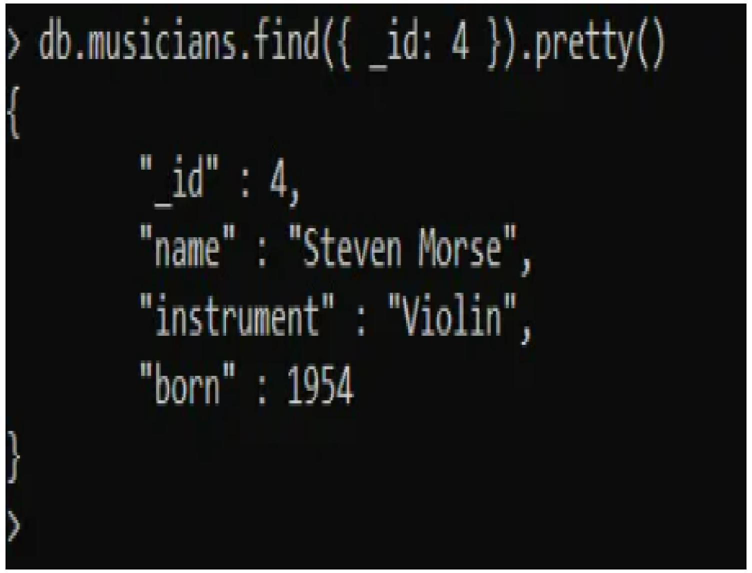
MongoDB – Update documents

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>,
     collation: <document>,
     arrayFilters: [ <filterdocument1>, ... ]
   }
```

Example:

```
db.musicians.find({ _id: 4 }).pretty()
```

Output:



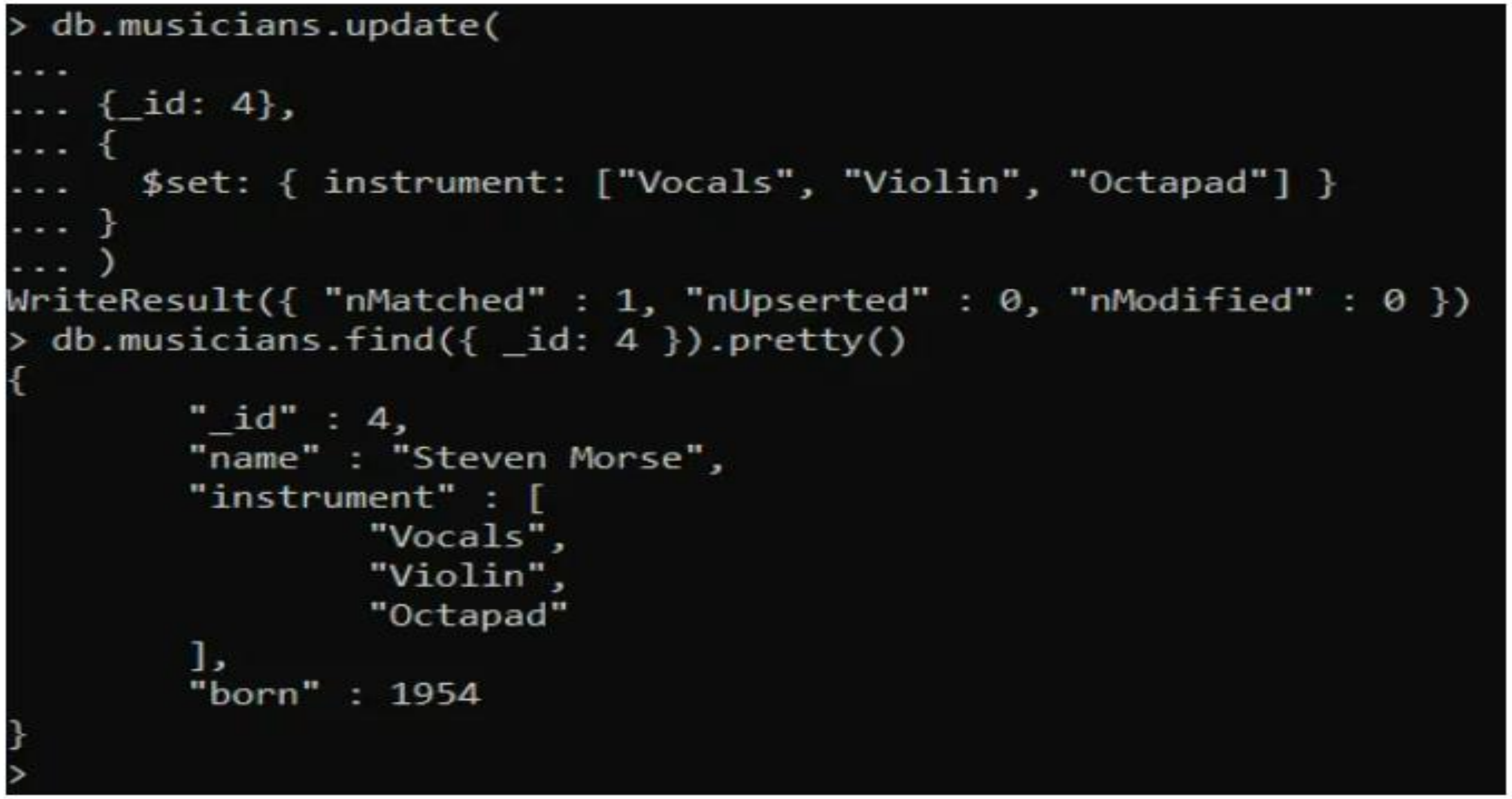
```
> db.musicians.find({ _id: 4 }).pretty()
{
        "_id" : 4,
        "name" : "Steven Morse",
        "instrument" : "Violin",
        "born" : 1954
}
>
```

So, now let us update the list of instrument played by this person, by making use of the **$set** operator for updating a single field.

Example:

```
db.musicians.update(


{_id: 4},
{
  $set: { instrument: ["Vocals", "Violin", "Octapad"] }
}
)
```

Output:



```
> db.musicians.update(
...
... {_id: 4},
... {
...   $set: { instrument: ["Vocals", "Violin", "Octapad"] }
... }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.musicians.find({ _id: 4 }).pretty()
{
        "_id" : 4,
        "name" : "Steven Morse",
        "instrument" : [
                "Vocals",
                "Violin",
                "Octapad"
        ],
        "born" : 1954
}
>
```

MongoDB – Characteristics of Update

- In case your field does not subsist in the current document, the **$set** operator will insert a new field with the specified value, until and unless it violates the type constraint.
- MongoDB users can also make use of { multi: true } for updating multiple documents which will meet the query criteria.
- Making use of { upsert: true } for creating a new document is also possible as no document goes with the query.

MongoDB – Characteristics of Update

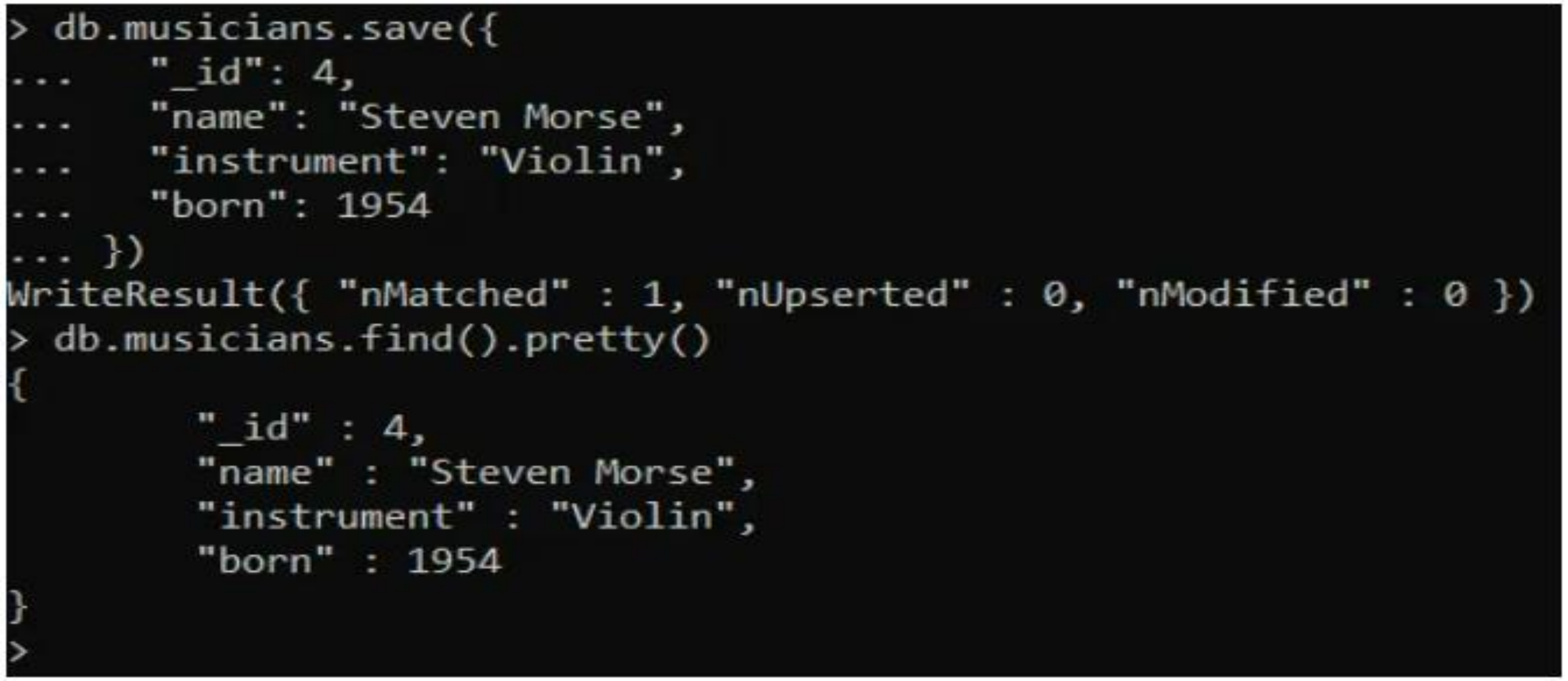
As you have encountered so far the update() method which helps in updating the Mongo database values in any existing document; on the other hand, the **save()** method is used to replace a document with another document conceded in the form of a parameter.

- **In other words, it can be said that the save() is a blend of both update() as well as insert().**
- **As the save() method is used, the document that exists will get updated. Otherwise, when it does not exist, it will create one.**
- When an _id field is not specified, MongoDB automatically creates a document with this **_id** containing an ObjectId value (as conducted by the insert() method).

Example:

```
db.musicians.save({
  "_id": 4,
  "name": "Steven Morse",
  "instrument": "Violin",
  "born": 1954
})
```

Output:



```
> db.musicians.save({
...    "_id": 4,
...    "name": "Steven Morse",
...    "instrument": "Violin",
...    "born": 1954
... })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.musicians.find().pretty()
{
        "_id" : 4,
        "name" : "Steven Morse",
        "instrument" : "Violin",
        "born" : 1954
}
>
```

MongoDB – Delete documents

- After creating an updating the document, a situation might occur where you want to delete any document or a set of documents.
- MongoDB also allows you to delete any particular document or multiple collections of documents.
- we learn about how to delete documents from a MongoDB database.
- **Deleting Documents in MongoDB**
- MongoDB allows you to delete a document or documents collectively using its one of the three methods. Three methods provided by MongoDB for deleting documents are:
- db.collection.deleteOne()
- db.collection.remove()
- db.collection.deleteMany()

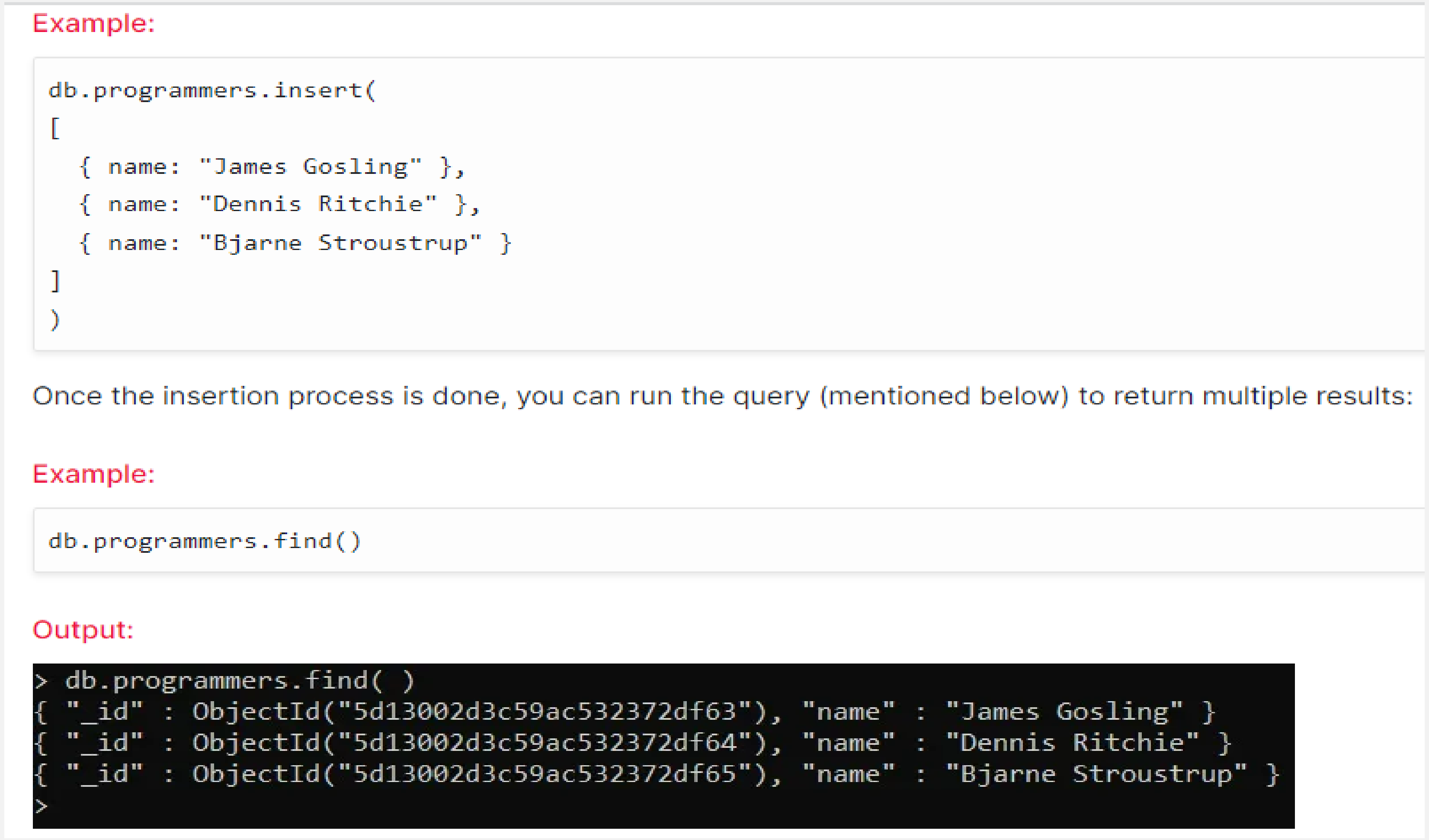
MongoDB – db.collection.deleteOne() Method

- This method is used to delete only a single document, even when more than one document matches with the criteria.

- Here is an example of using this db.collection.deleteOne() method for deleting the single document.

- To perform this process here, we have created a database and saved all the data separately.

-



Example:

```
db.programmers.insert(
[
  { name: "James Gosling" },
  { name: "Dennis Ritchie" },
  { name: "Bjarne Stroustrup" }
]
)
```

Once the insertion process is done, you can run the query (mentioned below) to return multiple results:

Example:

```
db.programmers.find()
```

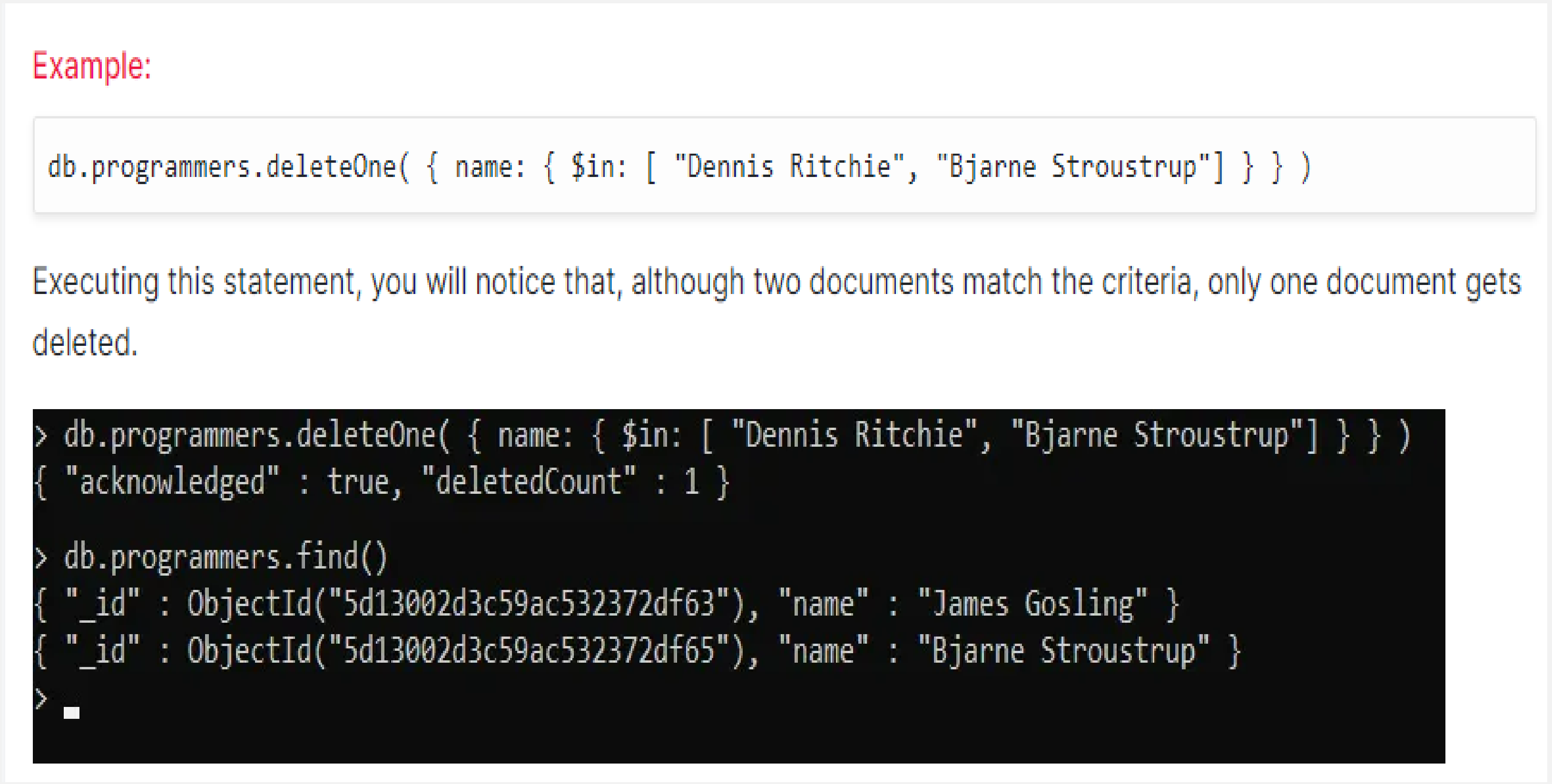
Output:

```
> db.programmers.find( )
{ "_id" : ObjectId("5d13002d3c59ac532372df63"), "name" : "James Gosling" }
{ "_id" : ObjectId("5d13002d3c59ac532372df64"), "name" : "Dennis Ritchie" }
{ "_id" : ObjectId("5d13002d3c59ac532372df65"), "name" : "Bjarne Stroustrup" }
>
```

- Once you execute the above line, you will find that some documents match your query criteria and will get displayed as output.

MongoDB – db.collection.deleteOne() Method

- Now, you can make use of the following criteria to delete the documents’



Example:

```
db.programmers.deleteOne( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup"] } } )
```

Executing this statement, you will notice that, although two documents match the criteria, only one document gets deleted.

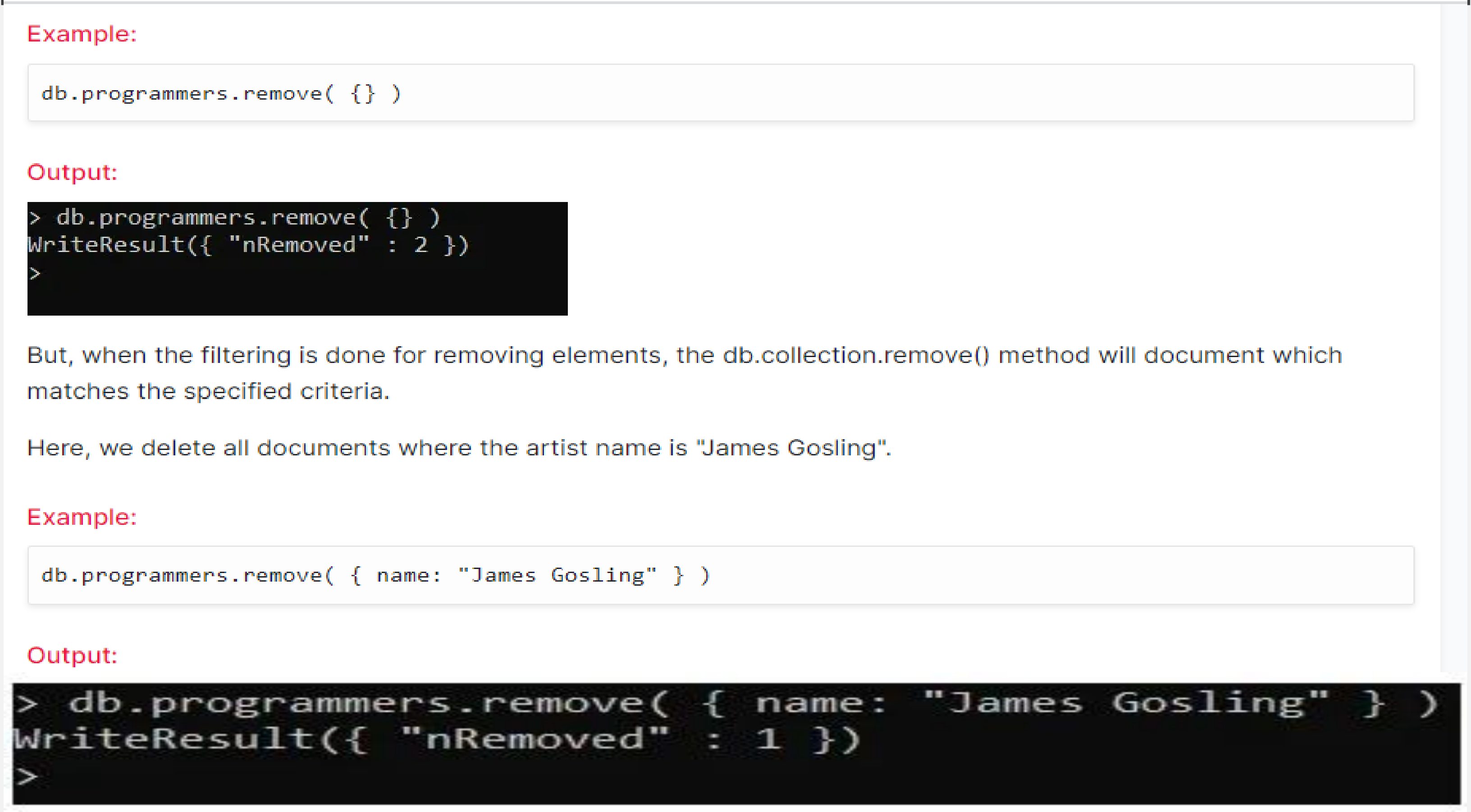
```
> db.programmers.deleteOne( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup"] } } )
{ "acknowledged" : true, "deletedCount" : 1 }

> db.programmers.find()
{ "_id" : ObjectId("5d13002d3c59ac532372df63"), "name" : "James Gosling" }
{ "_id" : ObjectId("5d13002d3c59ac532372df65"), "name" : "Bjarne Stroustrup" }
> 
```

MongoDB – Deletes each document

- It is possible for you to delete all your existing documents in a collection by simply excluding the filter criteria that is mentioned in the parenthesis () and specifying the document names within it.

- Now, to delete all documents from the programmers' collection, you have to write the query like this:

MongoDB – Deletes each document



Example:

```
db.programmers.remove( {} )
```

Output:

```
> db.programmers.remove( {} )
WriteResult({ "nRemoved" : 2 })
>
```

But, when the filtering is done for removing elements, the db.collection.remove() method will document which matches the specified criteria.

Here, we delete all documents where the artist name is "James Gosling".

Example:

```
db.programmers.remove( { name: "James Gosling" } )
```

Output:

```
> db.programmers.remove( { name:  "James Gosling" } )
WriteResult({ "nRemoved" : 1 })
>
```

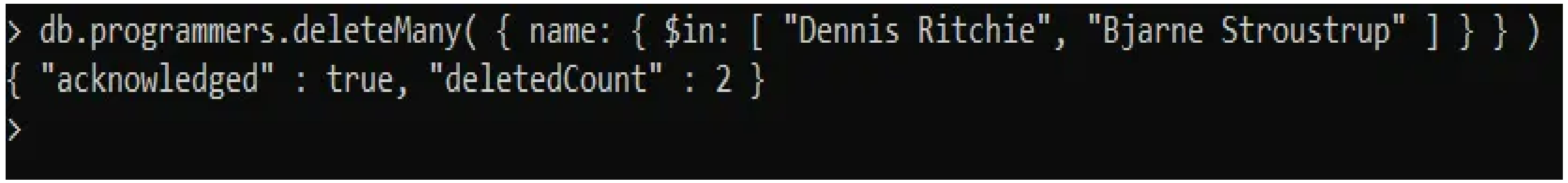
db.collection.deleteMany() Method

MongoDB allows you to delete multiple documents using the db.collection.deleteMany() method. This method deletes all your documents whichever match its criteria mentioned in the parameter. To check its implementation of db.collection.deleteMany() method, you can use the method the same way as done previously:

Example:

```
db.programmers.deleteMany( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup" ] } } )
```

Output:



```
> db.programmers.deleteMany( { name: { $in: [ "Dennis Ritchie", "Bjarne Stroustrup" ] } } )
{ "acknowledged" : true, "deletedCount" : 2 }
>
```

In this way, continuing from the previous example implementation, now all the records were also deleted, and the deleteMany() method was used this time. Now, if you try to find the documents using find() method, it won't show any result of your query or search.

MongoDB – Concept of aggregation

- When MongoDB users want to gather metrics from a MongoDB database, aggregation of MongoDB is the best tool for this.
- Bundling the data from numerous record sources which are then operated in various ways on a pool of data for returning a combined result is what MongoDB allows its users.
- In this chapter, you will learn about the concept of aggregation that s supported by MongoDB.

MongoDB – Concept of aggregation

What is Aggregation?

- In MongoDB, aggregation can be defined as the operation that is used for processing various types of data in the collection, which returns a calculated result.

- The concept of aggregation mainly clusters out your data from multiple different documents which are then used and operates in lots of ways (on these clustered data) to return a combined result which can bring new information to the existing database.

- You can relate aggregation to that of the count(*) along with the 'group by' used in SQL since both are equivalent in terms of the working.

MongoDB – Concept of aggregation

- MongoDB offers three different ways of performing aggregation:
- The aggregation pipeline.
- The map-reduce function.
- Single purpose aggregation methods.

- **aggregate() Method in MongoDB**
- MongoDB's aggregate function will cluster out the records in the form of a collection which can be then employed for providing operations like total number(sum), mean, minimum and maximum, etc. from the aggregated group of data extracted.

MongoDB – Concept of aggregation

- For performing such an aggregate function, the aggregate() method is used.
- The syntax of this method looks something like this:
- db.collection_name.aggregate(aggregate_operation)

Now, let us see how the aggregate() method works:

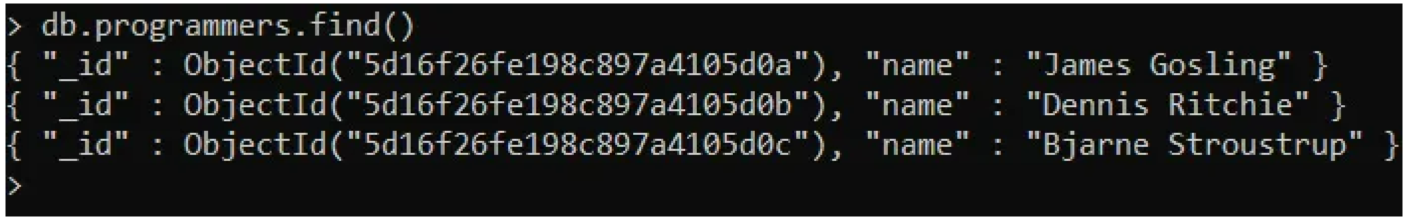
Implementation of aggregate() Method

Consider a collection named **programmers**, which has the following data. You have used the **find()** method to take a look at all the different data available within it:

Example:

```
db.programmers.find()
```

Output:

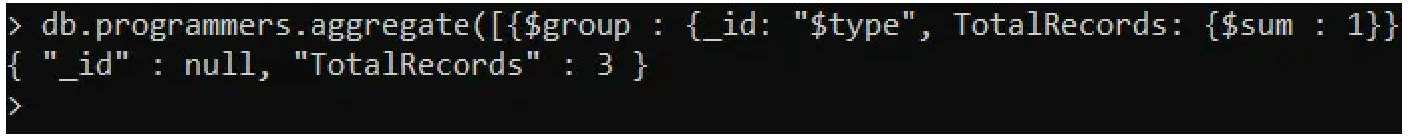


```
> db.programmers.find()
{ "_id" : ObjectId("5d16f26fe198c897a4105d0a"), "name" : "James Gosling" }
{ "_id" : ObjectId("5d16f26fe198c897a4105d0b"), "name" : "Dennis Ritchie" }
{ "_id" : ObjectId("5d16f26fe198c897a4105d0c"), "name" : "Bjarne Stroustrup" }
>
```

Example:

```
db.programmers.aggregate([{$group : {_id: "$type", TotalRecords: {$sum : 1}}}])
```

Output:

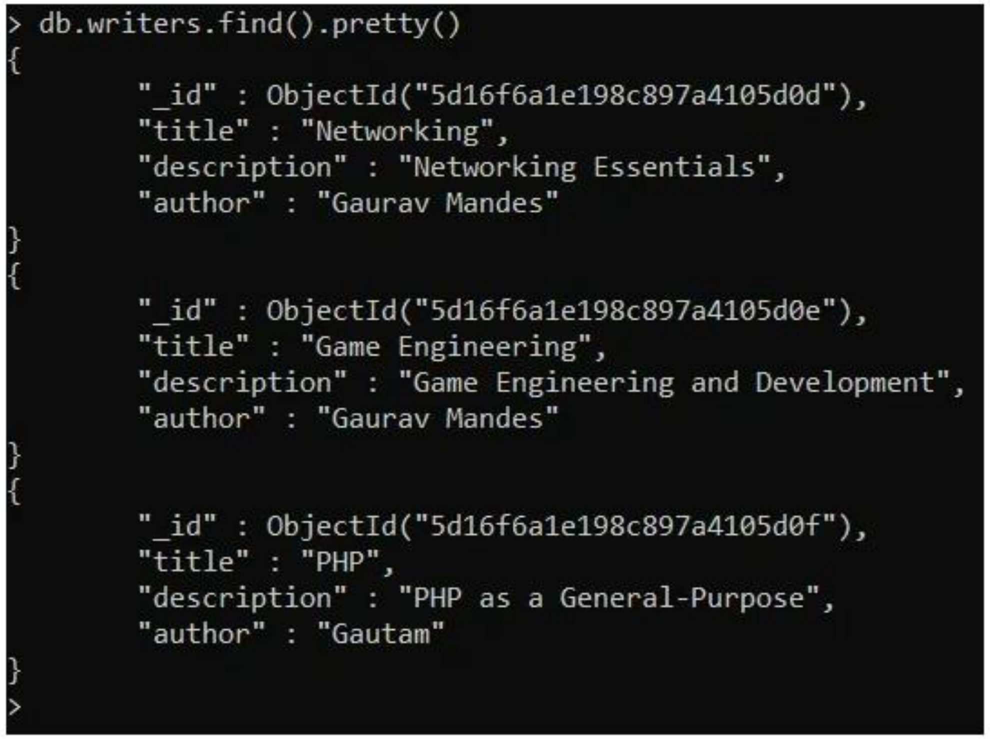


```
> db.programmers.aggregate([{$group : {_id: "$type", TotalRecords: {$sum : 1}}
{ "_id" : null, "TotalRecords" : 3 }
>
```

The above-executed aggregate() method will give the result shown. It says that there are three **records** which do not have any specific type and are available within the collection "programmers" for aggregating. Hence, the above aggregation method has clustered the collection's data in its best possible way.

Another example where we have a collection named **writers**, which has the following data:

Example:



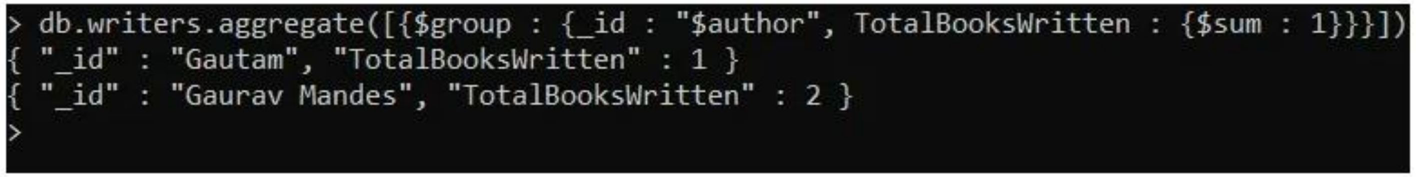
```
> db.writers.find().pretty()
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0d"),
        "title" : "Networking",
        "description" : "Networking Essentials",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0e"),
        "title" : "Game Engineering",
        "description" : "Game Engineering and Development",
        "author" : "Gaurav Mandes"
}
{
        "_id" : ObjectId("5d16f6a1e198c897a4105d0f"),
        "title" : "PHP",
        "description" : "PHP as a General-Purpose",
        "author" : "Gautam"
}
>
```

Now, execute the aggregate() method:

Example:

```
db.writers.aggregate([{$group : {_id : "$author", TotalBooksWritten : {$sum : 1}}}])
```

Output:



```
> db.writers.aggregate([{$group : {_id : "$author", TotalBooksWritten : {$sum : 1}}}])
{ "_id" : "Gautam", "TotalBooksWritten" : 1 }
{ "_id" : "Gaurav Mandes", "TotalBooksWritten" : 2 }
>
```

Different Expressions Used by Aggregate Function

Expression	Description
$sum	adds up the definite values of every document of a collection.
$avg	computes the average values of every document of a collection.
$min	finds and returns the minimum of all values from within a collection.
$max	finds and returns the maximum of all values from within a collection.
$push	feeds in the values to an array in the associated document.
$first	fetches out the first document.
$last	fetches out the last document.
$addToSet	feeds in the values to an array without duplication.

MongoDB – Sorting

- Sorting is one of the necessary database operations. It helps to simplify readability and sort the data as per the requirement.
- We will learn about sorting concept and how it can be implemented in the MongoDB database.
- What is sorting?
- Sorting can be defined as the ordering of data in the increase as well as decreasing manner based on a number of linear relationships among the various data items residing in the database.
- Sorting can be performed on entities like information (which includes names and facts), numbers as well as records.
- Sorting also increases the readability and uniformity to the displayed data from the database.

sort() Method of MongoDB

For sorting your MongoDB documents, you need to make use of the **sort()** method. This method will accept a document that has a list of fields and the order for sorting. For indicating the sorting order, you have to set the value **1 or -1** with the specific entity based on which the ordering will be set and displayed. One indicates organizing data in ascending order while -1 indicates organizing in descending order.

Syntax:

The basic syntax for the sort() method is:

```
db.collection_name.find().sort({FieldName1: sort order 1 or -1, FieldName2: sort order})
```

Example:

Consider a collection that is having the following data:

```
{ "_id" : ObjectId(5983548781331abf45ec5), "topic":"Cyber Security"}
{ "_id" : ObjectId(5565548781331aef45ec6), " topic ":"Digital Privacy"}
{ "_id" : ObjectId(5983549391331abf45ec7), " topic ":"Application Security Engineering"}
```

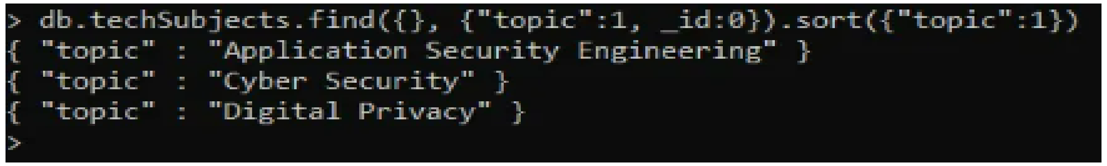
Suppose I want to get data only from the topic field from all the documents in **ascending order**, then it will be executed like this:

Example:

```
db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":1})
```

MongoDB **projection** is used to display only selected fields of the document.
The above statement will result in:

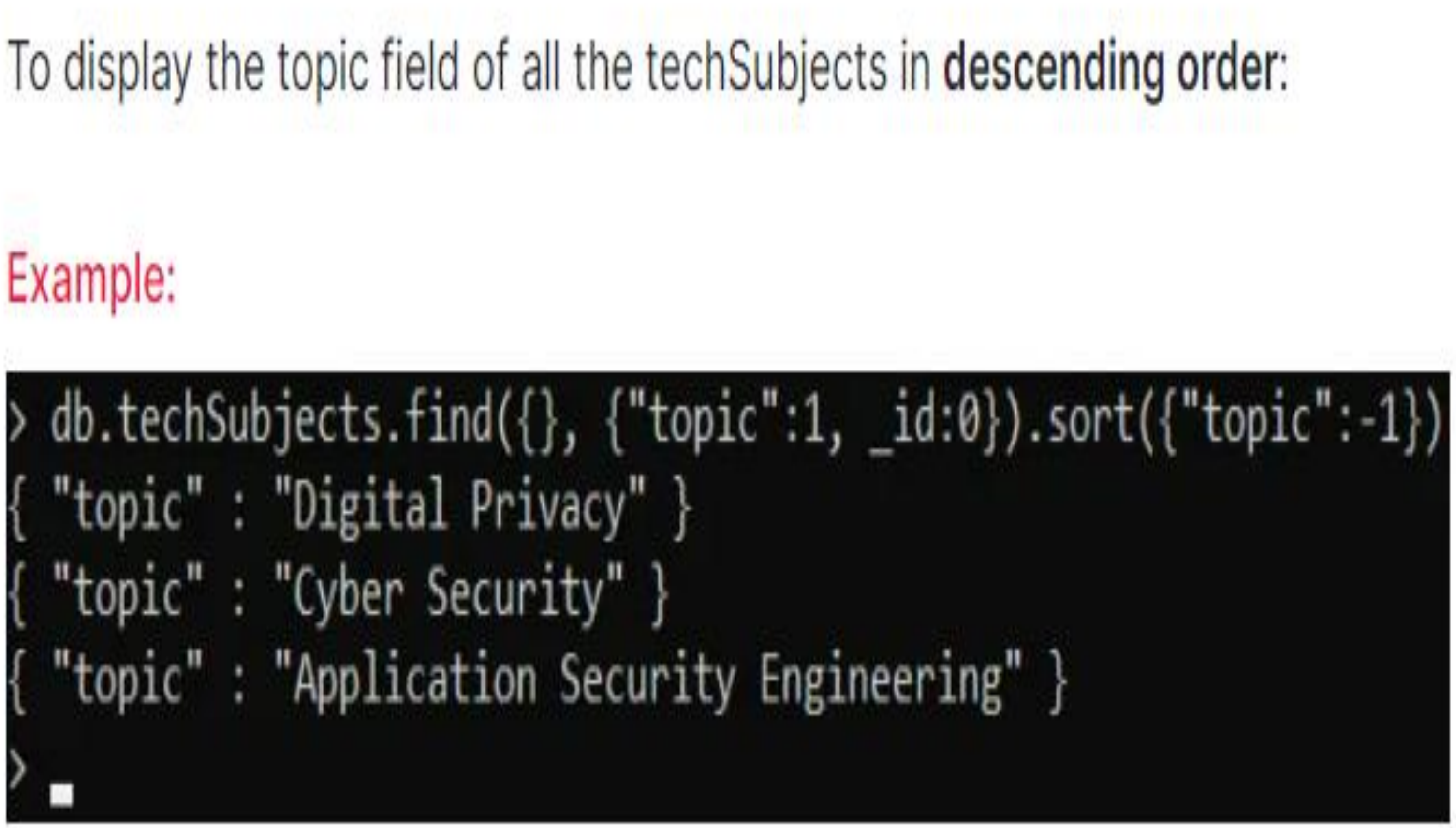
Output:



```
> db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":1})
{ "topic" : "Application Security Engineering" }
{ "topic" : "Cyber Security" }
{ "topic" : "Digital Privacy" }
>
```

It is to be noted that in case you do not state or explicitly define the sorting preference **(1 or -1)**, then, the sort() method will exhibit your documents in ascending order by default.

MongoDB – Sorting



To display the topic field of all the techSubjects in **descending order**:

Example:

```
> db.techSubjects.find({}, {"topic":1, _id:0}).sort({"topic":-1})
{ "topic" : "Digital Privacy" }
{ "topic" : "Cyber Security" }
{ "topic" : "Application Security Engineering" }
> ▪
```

Metadata Sort Technique

You can also specify the sorting of parameters through a new field name for the calculated metadata which you can specify under the **$meta** expression as a value. Here is a sample document that specifies a descending sort using the metadata "textCount":

Example:

```
{ count: { $meta: "textCount" } }
```

Here this specified metadata will determine the sort order.

Here is the syntax:

Syntax:

```
db.collection_name.find(
<MongoDB query statement>,
{ value: { $meta: <metadataKeyword> } }
)
```

Output:

```
db.techWriter.find({},
{ count: { $meta: "textCount" } }
).sort( { count: { $meta: "textCount" } } )
```

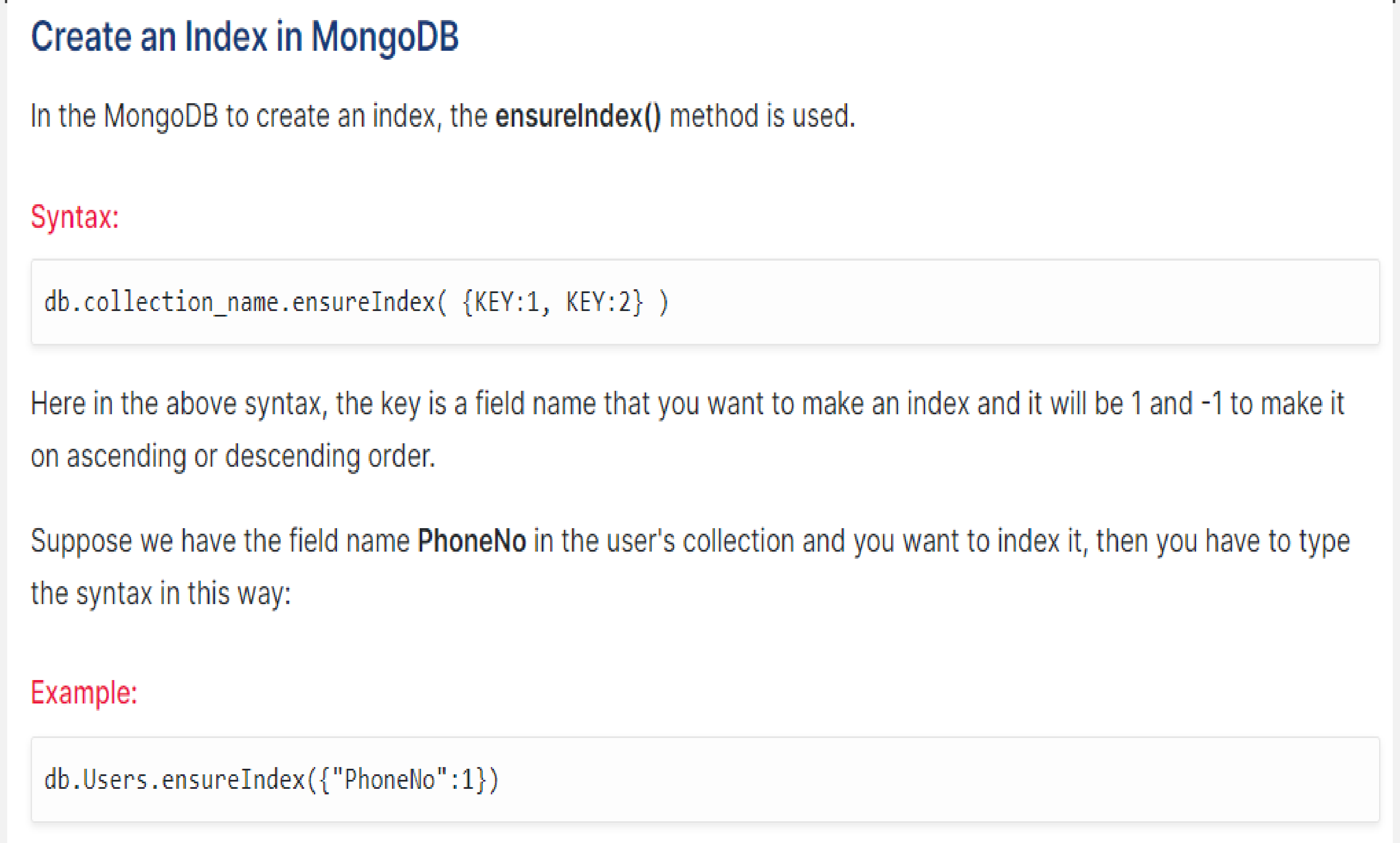
MongoDB – Indexing

- Indexing is a necessary operation in MongoDB which brings efficiency in various execution of statements.
- In this chapter, you will learn about the concept of indexing and how it can be implemented in the MongoDB database.
- What is indexing in MongoDB?
- The concept of indexing is crucial for any database, and so for MongoDB also.

MongoDB – Indexing

- Databases having indexes make queries performance more efficient.
- When you have a collection with thousands of documents and no indexing is done, your query will keep on finding certain documents sequentially.
- This would require more time to find the documents.
- But if your documents have indexes, MongoDB would restrict make it specific the number of documents to be searched within your collection.

MongoDB – Indexing



Create an Index in MongoDB

In the MongoDB to create an index, the **ensureIndex()** method is used.

Syntax:

```
db.collection_name.ensureIndex( {KEY:1, KEY:2} )
```

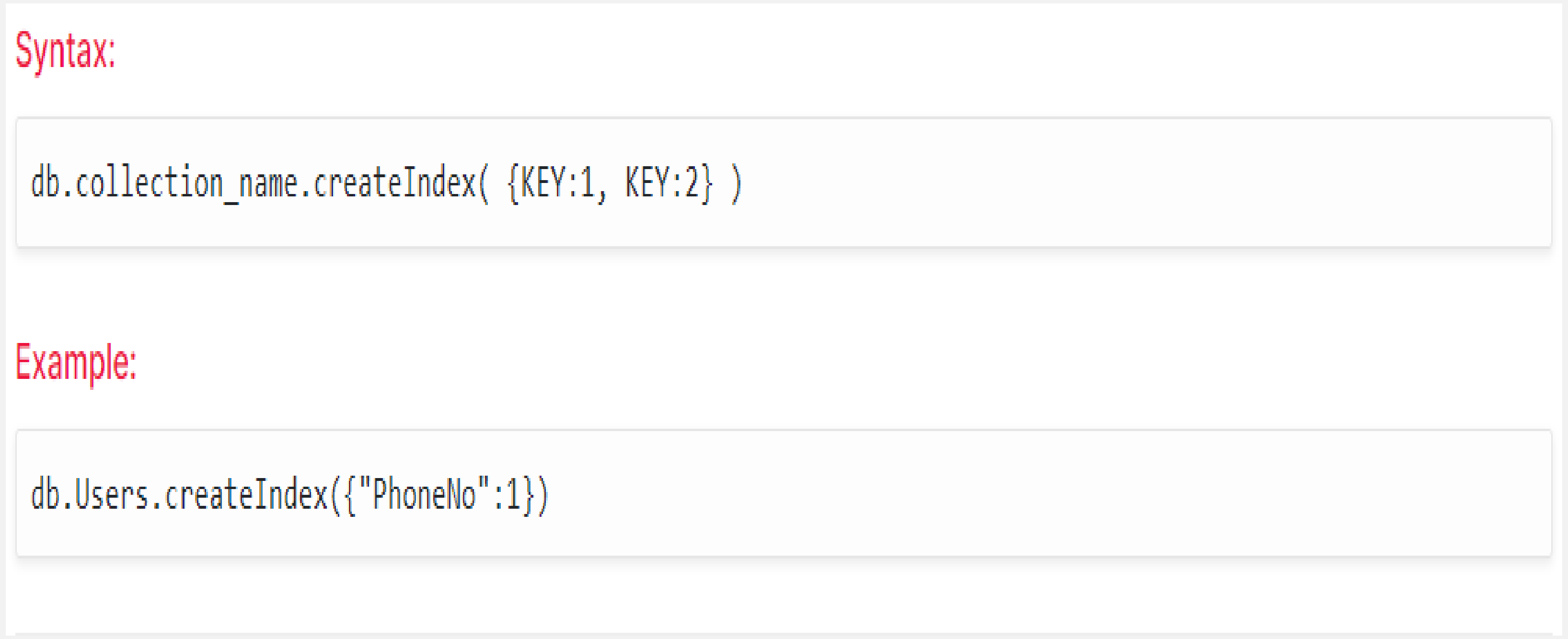
Here in the above syntax, the key is a field name that you want to make an index and it will be 1 and -1 to make it on ascending or descending order.

Suppose we have the field name **PhoneNo** in the user's collection and you want to index it, then you have to type the syntax in this way:

Example:

```
db.Users.ensureIndex({"PhoneNo":1})
```

MongoDB – Indexing



Syntax:

```
db.collection_name.createIndex( {KEY:1, KEY:2} )
```

Example:

```
db.Users.createIndex({"PhoneNo":1})
```

MongoDB – Advanced Indexing

Indexes in any database are essential in improving the performance of their data retrieval speed, and the same is true for MongoDB.

By using indexes, performance in MongoDB queries becomes more efficient.

We will learn about the different indexing techniques and their limitations.

MongoDB – Advanced Indexing

Indexing Array Fields

Let us consider a situation where you wish to look for user documents depending on the user's tags.

For this, you have to create an index on your tag array within your collection.

For building an index based array, you have to create separate index entries for each of its fields.
Let us see how it can be implemented

If you consider the below-mentioned document of users collection:

Example:

```
{
  "location": {
    "city": "Calgary, Alberta",
    "country": " Canada"
  },
  "tags": [
    "techwriter",
    "developer",
    "programmer"
  ],
  "name": "James Gosling"
}
```

For creating the index on tag array, you have to implement, i.e. execute the following code:

```
db.users.ensureIndex ( {"tags": 1} )
```