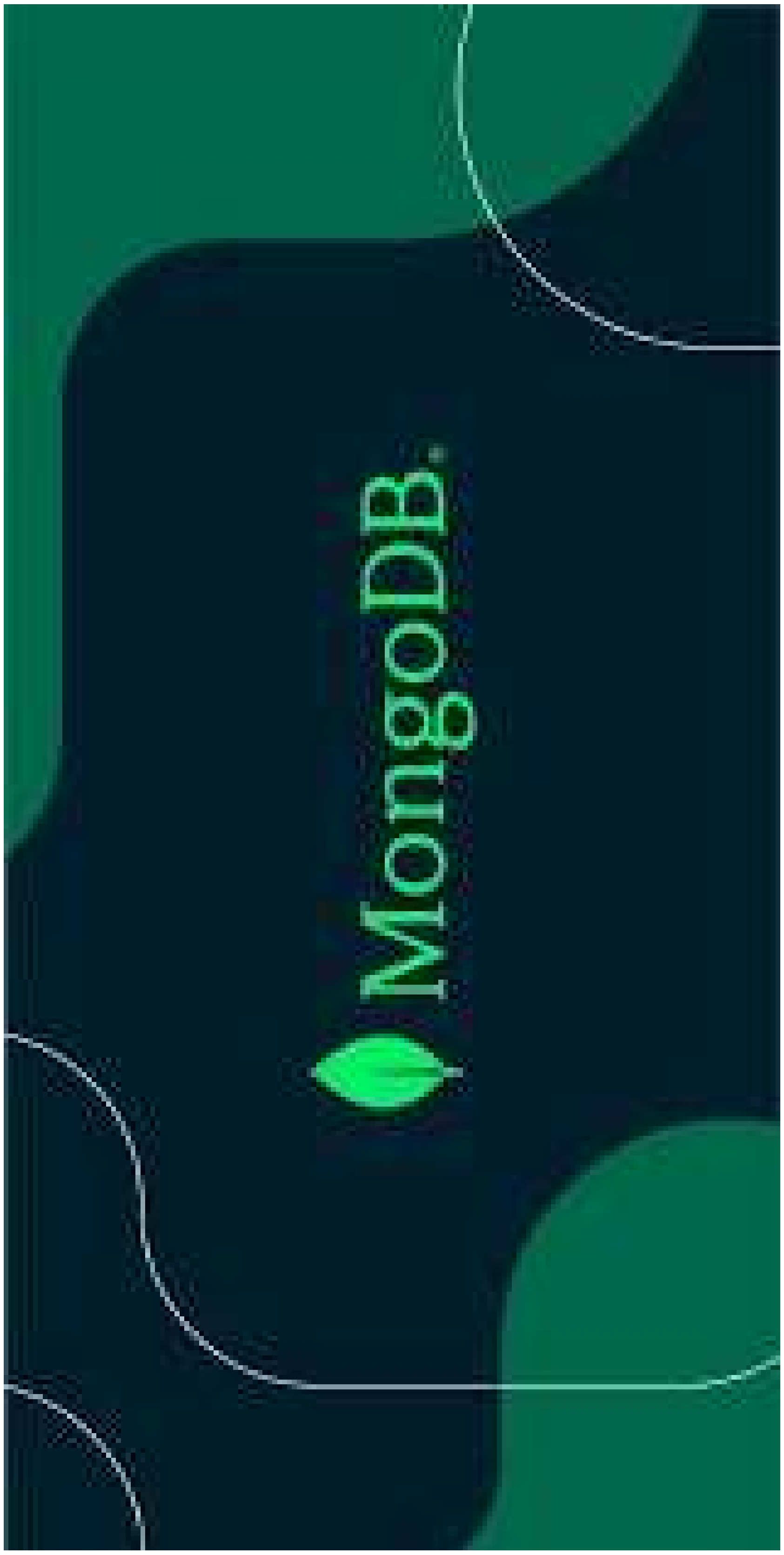

MongoDB.

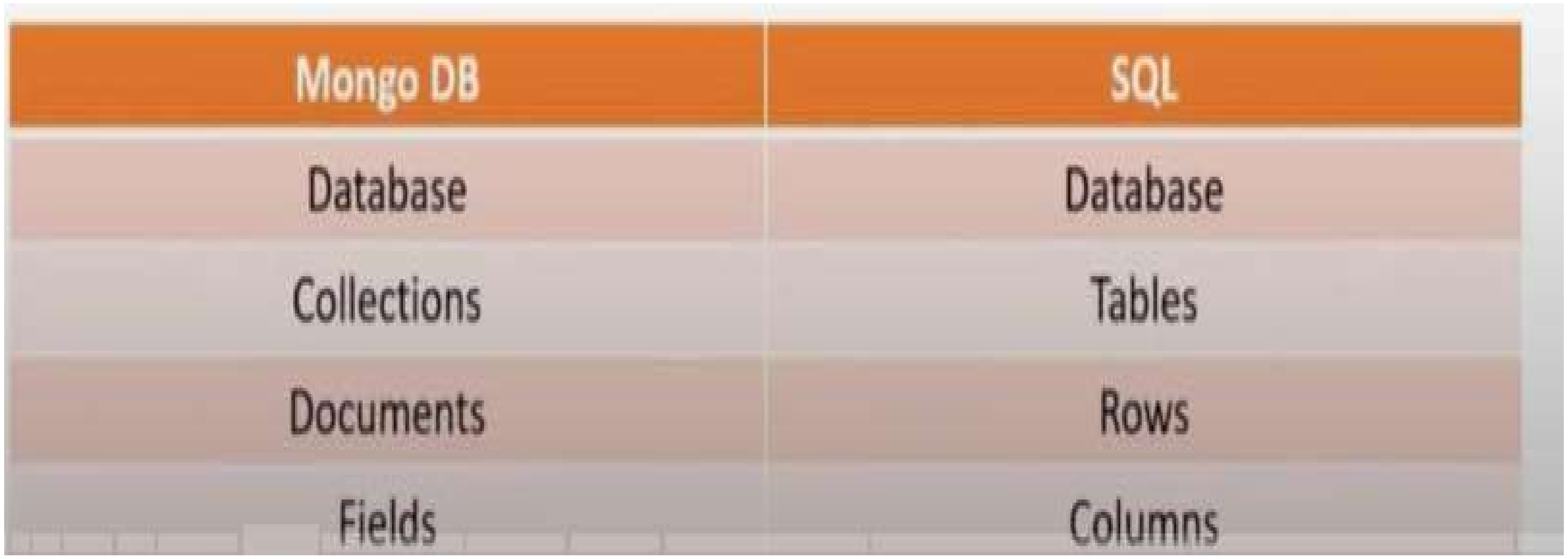
- *MongoDB is a popular Opensource NoSQL and document-based database.*
- *MongoDB’s documents are structures in JSON (JavaScript Object notation) with the proper schema.*
- *MongoDB is available as Cloud platform called MongoDB Atlas and Software as MongoDB server.*

- MongoDB is a document database designed for ease of development and scaling.
- Classified as a NoSQL database.
- Document Database

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects(BSON). The values of fields may include other documents, arrays, and arrays of documents.

```
{
product: “Smart Phone”,
brand: “Apple”,
model: “Iphone 13”,
price: 5000,
colour: [“red”, “white”, “blue”]
}
```

| Product | Brand | Model | Price | Colour |
|---|---|---|---|---|
| Smart Phone | Apple | Iphone 13 | 5000 | red |



| Mongo DB | SQL |
|---|---|
| Database | Database |
| Collections | Tables |
| Documents | Rows |
| Fields | Columns |

## MongoDB CURD Operations

**CRUD operations *create*, *read*, *update*, and *delete* documents.**

**CREATE Operation**

**Create Database**

The use Command

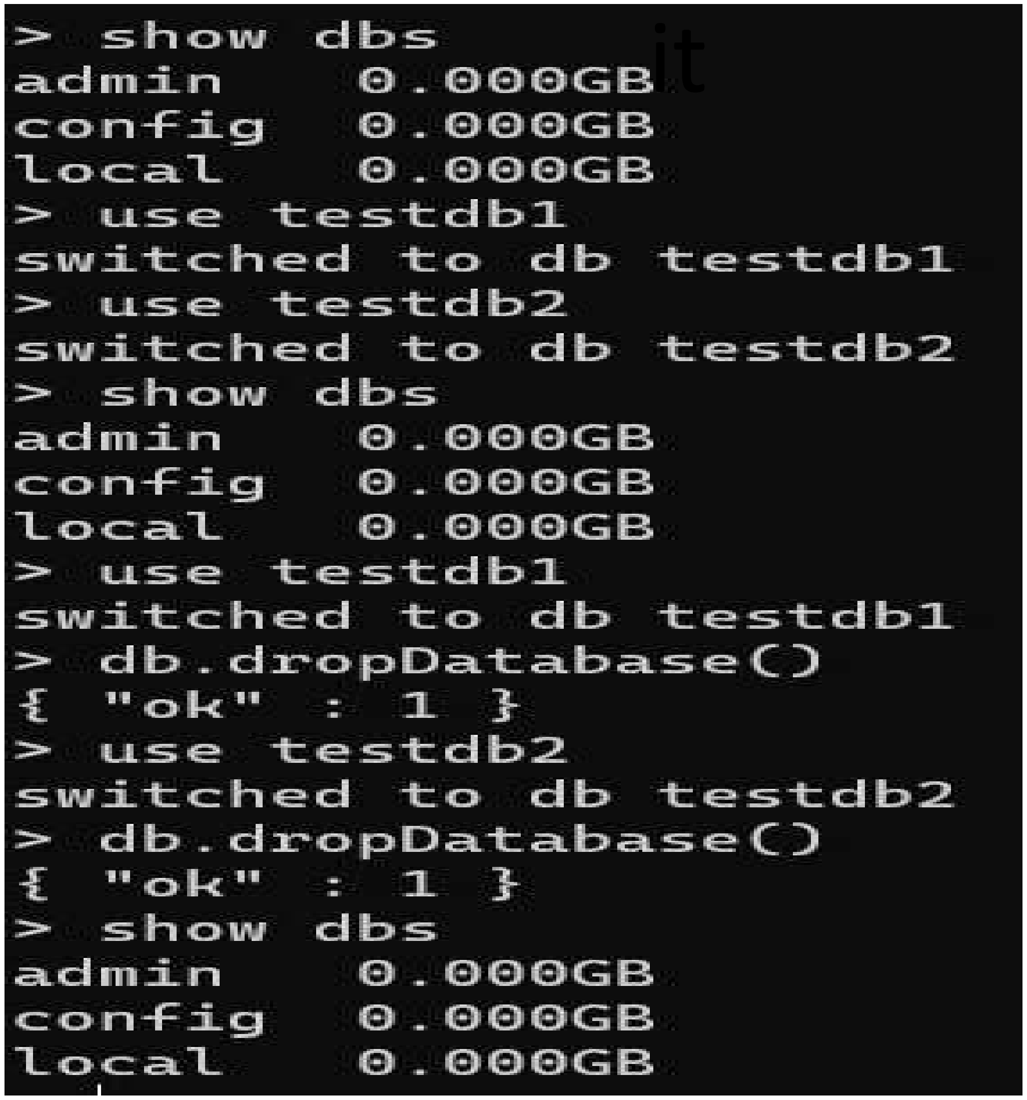
MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows −

```
use DATABASE_NAME
```

## How to create a database and to drop it

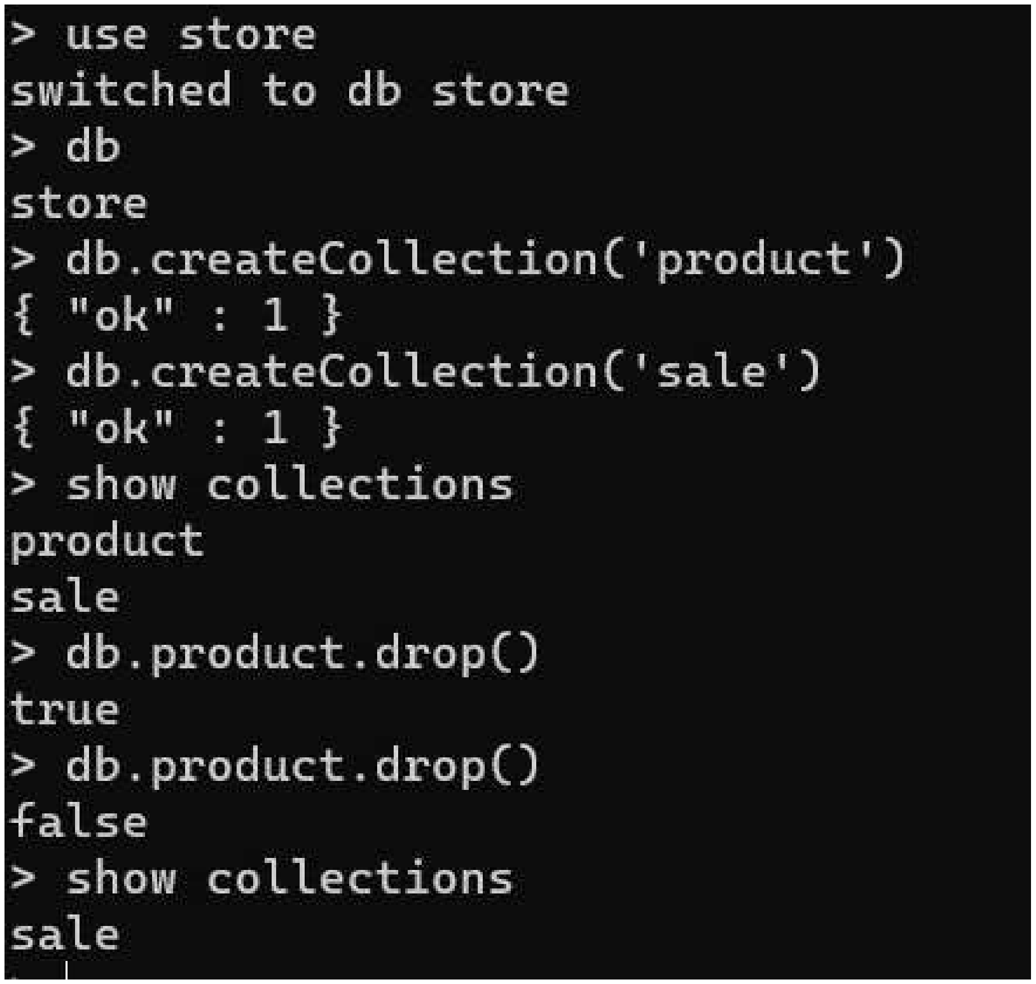


```
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
> use testdb1
switched to db testdb1
> use testdb2
switched to db testdb2
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
> use testdb1
switched to db testdb1
> db.dropDatabase()
{ "ok" : 1 }
> use testdb2
switched to db testdb2
> db.dropDatabase()
{ "ok" : 1 }
> show dbs
admin    0.000GB
config   0.000GB
local    0.000GB
```

## To create a collection

- The createCollection() Method
- Basic syntax of **createCollection()** command is as follows −
- db.createCollection(name, options)
- In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.
- Options parameter is optional, so you need to specify only the name of the collection.
- >db.createCollection("mycollection")
- >db.createCollection(‘mycl’)

## How to create a collection and drop it



```
> use store
switched to db store
> db
store
> db.createCollection('product')
{ "ok" : 1 }
> db.createCollection('sale')
{ "ok" : 1 }
> show collections
product
sale
> db.product.drop()
true
> db.product.drop()
false
> show collections
sale
```

- In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.
- **To show the collections**
- >show collections
- mycl
- **To insert one document**
- >db.movie.insert({"name":"MIB"})
- 
- 
- **//movie – collection name**
- **>show collections**
- mycl
- movie

**To drop a database using MongoDB command.**

The dropDatabase() Method

```
>db.dropDatabase()
```

```
>use mydb
switched to db mydb
>db.dropDatabase()
```

**To drop a collection**

The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax

Basic syntax of **drop()** command is as follows −

```
>db.COLLECTION_NAME.drop()
```

```
>db.mycl.drop()
True
```

## WRITE Operation

**To insert document in MongoDB collection**

insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

**Syntax**

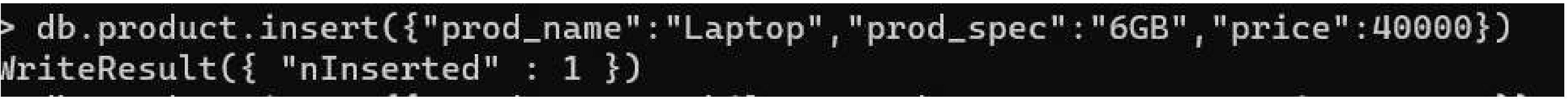
The basic syntax of **insert()** command is as follows –

>db.COLLECTION_NAME.insert(document)

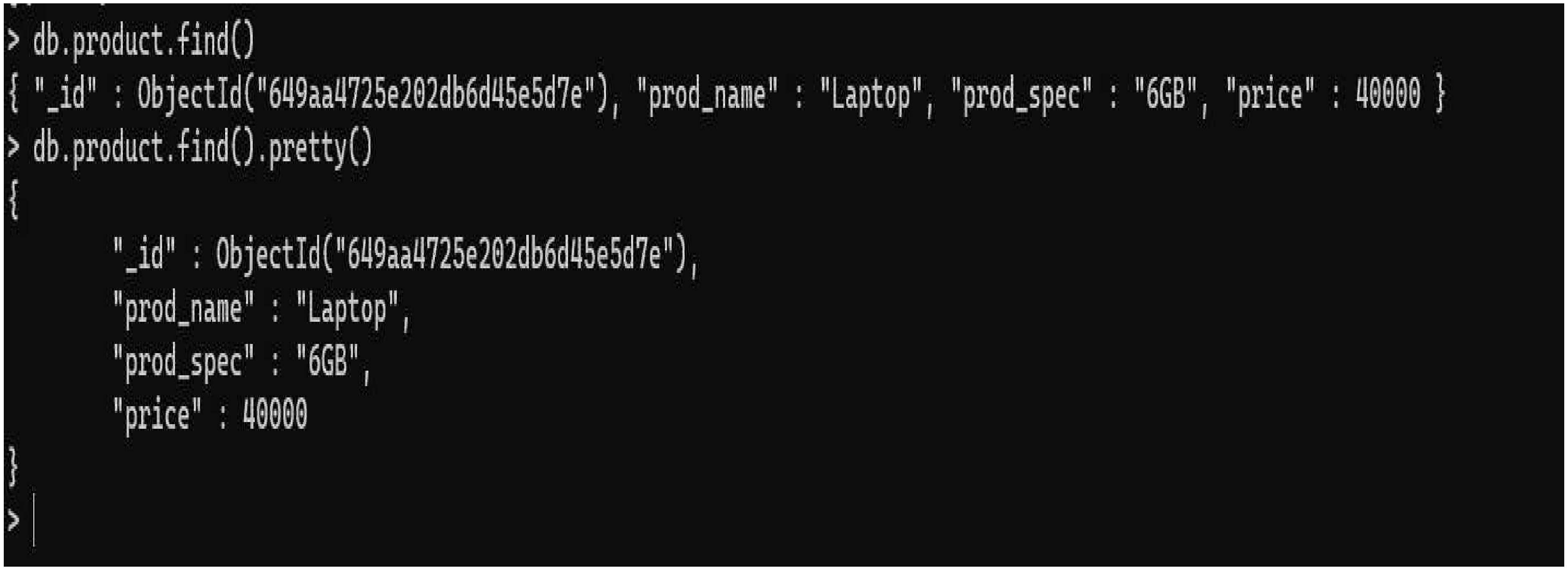
- > db.ta.insert({regno:123,name:"raji"})
- WriteResult({ "nInserted" : 1 })

- if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
- _id is 12 bytes hexadecimal number unique for every document in a collection
- `>db.ta.insert({regno:123})  //`

## How to insert in to the collections



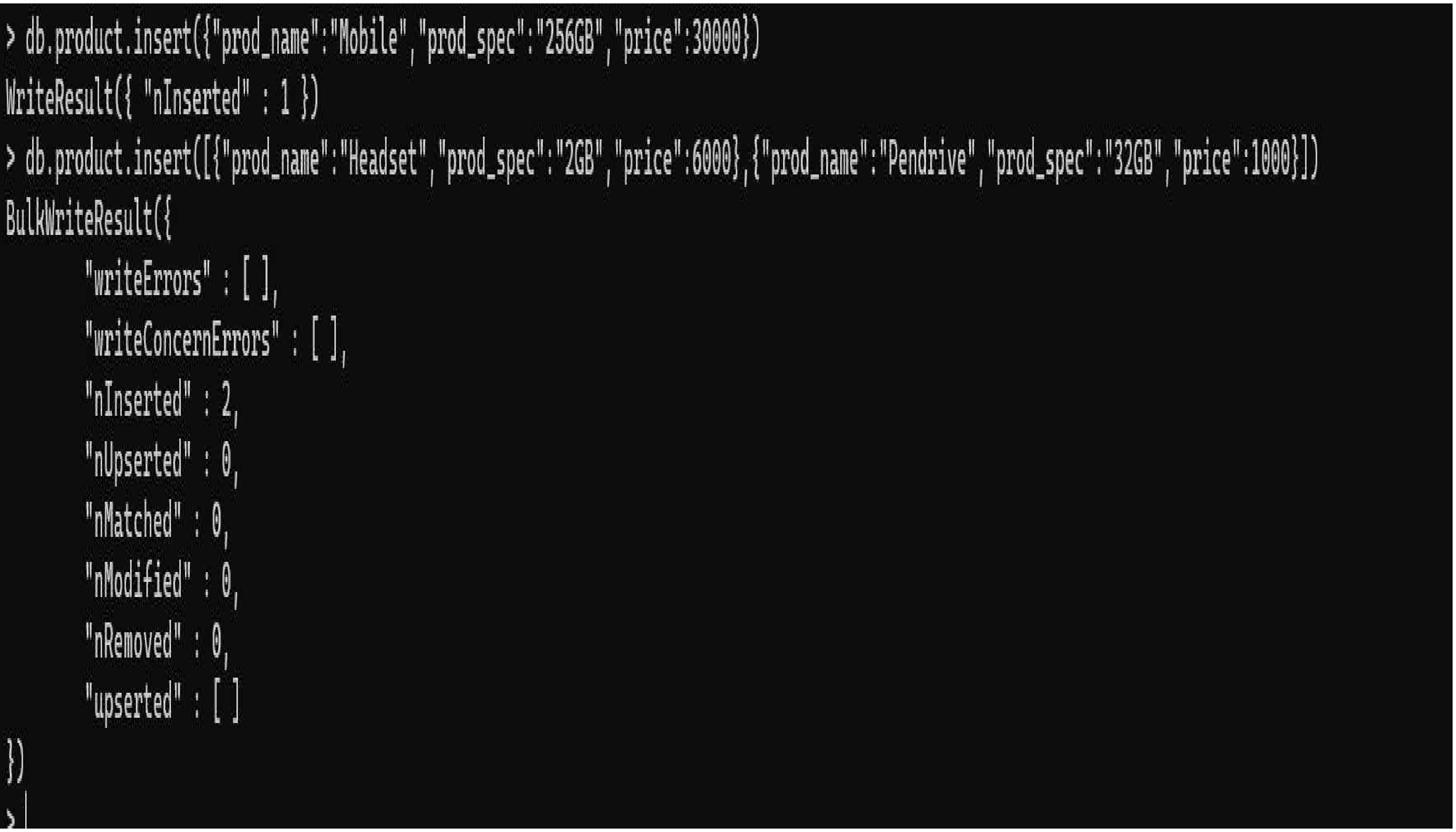
```
> db.product.insert({"prod_name":"Laptop","prod_spec":"6GB","price":40000})
WriteResult({ "nInserted" : 1 })
```



```
> db.product.find()
{ "_id" : ObjectId("649aa4725e202db6d45e5d7e"), "prod_name" : "Laptop", "prod_spec" : "6GB", "price" : 40000 }
> db.product.find().pretty()
{
        "_id" : ObjectId("649aa4725e202db6d45e5d7e"),
        "prod_name" : "Laptop",
        "prod_spec" : "6GB",
        "price" : 40000
}
>
```

## Array of documents into the insert() Method

- >
  db.bookdata.insert([{Title:"DBMS",Author:"Forozan"},{Title:"NoSQL
  ",Author:"Davis",comments:[{user:"Manu",Comment:"good",likes:1
  }]}])
- BulkWriteResult({
-         "writeErrors" : [ ],
-         "writeConcernErrors" : [ ],
-         "nInserted" : 2,
-         "nUpserted" : 0,
-         "nMatched" : 0,
-         "nModified" : 0,
-         "nRemoved" : 0,
-         "upserted" : [ ]
- })



```
> db.product.insert({"prod_name":"Mobile","prod_spec":"256GB","price":30000})
WriteResult({ "nInserted" : 1 })
> db.product.insert([{"prod_name":"Headset","prod_spec":"2GB","price":6000},{"prod_name":"Pendrive","prod_spec":"32GB","price":1000}])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
>
```

## The insertOne() command

- The basic syntax of insertOne() command is as follows –
- >db.COLLECTION_NAME.insertOne(document)
-
-
- db.bookdata.insertOne({Title:"Java",Author:"Edison"})
- {
-         "acknowledged" : true,
-         "insertedId" : ObjectId("6093c686293335ade5c65b85")
- }

## The insertMany() method

- You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.
- >db.ta.insertMany( [{},{},{}] )

## READ Operation

**To view the documents in a collection**

```
> db. COLLECTION_NAME. find()
```

// use the command db.bookdata.find().pretty() and find the result

Note

To insert the document you can use **db.post.save(document)** also

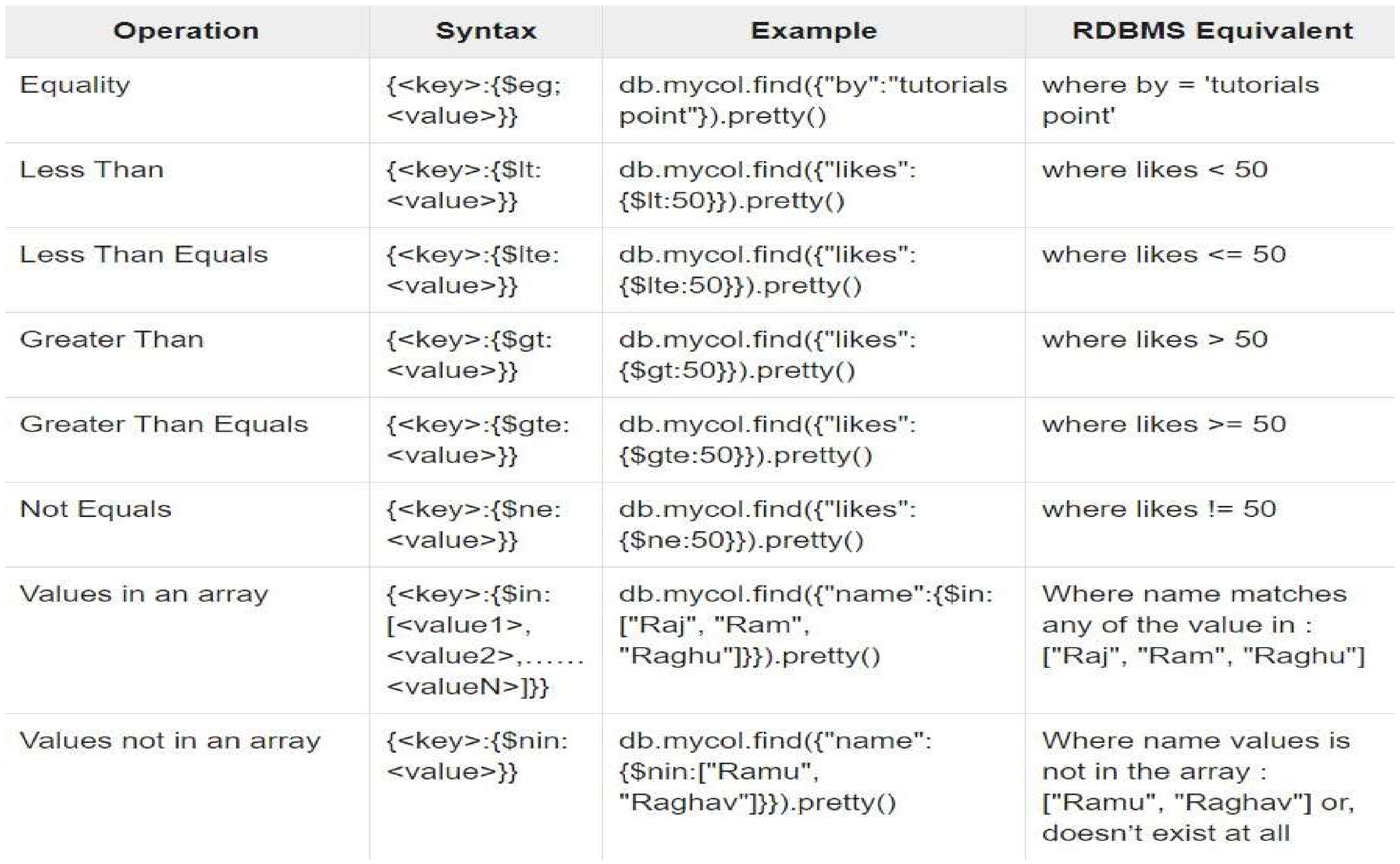
**To find the no.of  documents in a collection**

```
> db.bookdata.count()
4
```

## The findOne() method

- Apart from the find() method, there is **findOne()** method, that returns only one document.
- Syntax
- >db.COLLECTIONNAME.findOne()
- > db.ta.findOne({regno:124})

# RDBMS Where Clause Equivalents in MongoDB



| Operation | Syntax | Example | RDBMS Equivalent |
|---|---|---|---|
| Equality | {<key>:{$eg;<value>}} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$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":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$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

Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND −

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

```
>db.ta.find({$and:[{regno:{$lt:200}},{age:{$gt:18}}]})
```

## OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** –

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

## Using AND and OR Together

- The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (author='Forozan' OR title = 'MongoDB Overview')'**
- >db.mycol.find({"likes": {$gt:10}, $or: [{"author": "Forozan"},{"title": "MongoDB Overview"}]}).pretty()

## NOR in MongoDB

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

               {"First_Name": "Radhika"},
               {"Last_Name": "Christopher"}
          ]
     }
).pretty()
```

## NOT in MongoDB

Following example will retrieve the document(s) whose age is not greater than 25

```
> db.empDetails.find( { "Age": { $not: { $gt: "25" } } } )
```

**findOne() method**

returns a single document

```
db.restaurants.findOne()
```

**findOne() method with a Query Specification**

```
db.restaurants.findOne(
  {
    $or: [
          { "name" : /^G/ },
          { "address.coord": { $gt: 90 } }
         ]
  }
);
```

## UPDATE Operation

**MongoDB update**

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.

```
>db.bookdata.update({'Author':'Edison'},{$set:{'Title':'RDBMS'}})
```

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

```
>db.bookdata.update({'Author':'RamMohan'},{$set:{Title:'COA'}},{multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 0 })
```

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

```
>db.bookdata.save(
  {
     "_id" : ObjectId("507f191e810c19729de860ea"),
         "Title":"DBMS",
     "by":"Temko"
  }
)
```

Will replace the document with Title:DBMS by the given fields

## MongoDB findOneAndUpdate() method

- The **findOneAndUpdate()** method updates the values in the existing document.
- // Create a database named Employee. Create a collection named empDetails

(Q.No.1)

- Following example updates the age and email values of the document with name 'Radhika'.
- > db.empDetails.findOneAndUpdate(
-  {First_Name: 'Radhika'},
-  { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
- )

## MongoDB updateOne() method

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

```
> db.empDetails.updateOne(
    {First_Name: 'Radhika'},
    { $set: { Age: '30',e_mail:
'radhika_newemail@gmail.com'}}
)
```

## MongoDB updateMany() method

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

```
> db.empDetails.updateMany(
    {Age:{ $gt: "25" }},
    { $set: { Age: '00'}}
)
```

## Delete Operation

- db.collection.deleteMany()
- db.collection.deleteOne()
- **Delete All Documents**
- To delete all documents from a collection, pass an empty filter document {} to the db.collection.deleteMany() method

- **Delete All Documents that Match a Condition**
- db.collectionname.deleteMany({<field1>: <value1>, ... })
- db.bookdata.deleteMany({ Author : "Forozan" })
- **Delete Only One Document that Matches a Condition**
- To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter) use the db.collection.deleteOne() method.
- db.bookdata.deleteOne({ Author : "Forozan" })