MongoDB

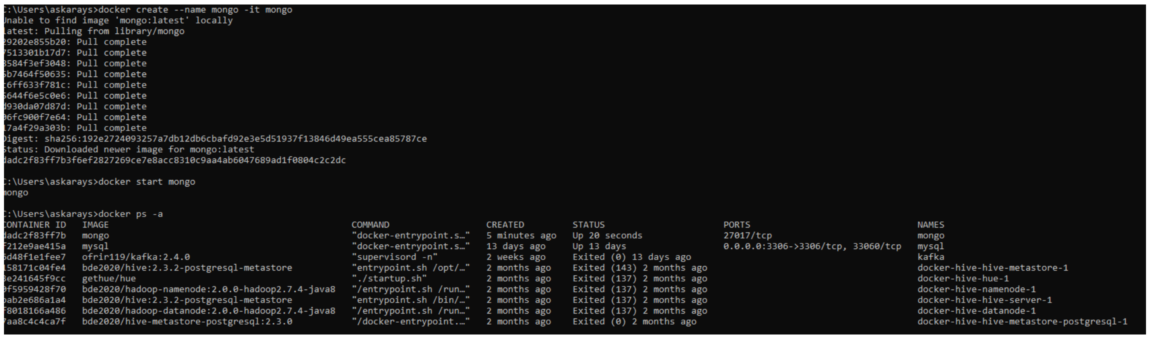
Askarova Aisha

1. NOTES REGARDING THE PRACTICE

```
docker create --name mongo -it mongo
docker start mongo
docker cp C:\Users\askarays\Downloads\products.json Mongo:/da
docker exec -it mongo /bin/bash
mongoimport --host localhost --port 27017 -d epam -c products
mongosh
db.help()  # Опции базы данных
db.products.help()  # Опции коллекции "products"
db.products.find().help()  # Опции find в коллекции "products
show dbs
use epam
show collections
db.products.find()
db.products.count()
db.products.insert({
  "Product id": "ac9",
  "Product name": "AC9 Phone",
  "Product brand": "ACME",
  "Product type": "phone",
  "Product price": 333,
  "Product Warranty (in years)": 0.25,
  "Product availability": true
})

# Query 1
db.products.find().skip(2).limit(10).pretty()
```

```
# Query 2
db.products.find({}, {name: 1, brand: 1, _id: 0}).pretty()

# Query 3
var results = db.products.find({}, {id: 1, limits: 1}).limit

# Query 4
db.products.find({"Product price": { $gte: 200 }}, {id: 1, na

# Query 5
db.products.find({}, {id: 1, name: 1, "Product price": 1}).so

# Query 6
db.products.find({"Product type": "service"}).count()

#Upd1
db.products.update({ "Product id": "ac3" }, { $set: { "compar
#Upd2
db.products.updateMany({ "Product name": /ac3/i }, { $set: {
#Delete
db.products.deleteMany({ "Product type": "service" })

# Creating an index for the "price" field
> db.products.createIndex({"Product price": 1})

# Creating a composite index for the "type" and "subtype" fie
> db.products.createIndex({"Product type": 1, "subtype": 1})

# Creating a text index for the "name" field
> db.products.createIndex({"Product name": "text"})
> db.isMaster()
> db.serverStatus()
> db.currentOp()
> rs.status()
```

Firstly I create the mongodb docker container



```
:\Users\askarays>docker create --name mongo -it mongo
nable to find image 'mongo:latest' locally
atest: Pulling from library/mongo
9202e855b20: Pull complete
513301b17d7: Pull complete
584f3ef3048: Pull complete
b7464f50635: Pull complete
6ff633f781c: Pull complete
644f6e5c0e6: Pull complete
930da07d87d: Pull complete
6fc900f7e64: Pull complete
7a4f29a303b: Pull complete
igest: sha256:192e2724093257a7db12db6cbafd92e3e5d51937f13846d49ea555cea85787ce
tatus: Downloaded newer image for mongo:latest
adc2f83ff7b3f6ef2827269ce7e8acc8310c9aa4ab6047689ad1f0804c2c2dc

:\Users\askarays>docker start mongo
ongo

:\Users\askarays>docker ps -a
ONTAINER ID   IMAGE                                             COMMAND                  CREATED          STATUS                       PORTS                               NAMES
adc2f83ff7b   mongo                                             "docker-entrypoint.s…"   5 minutes ago    Up 20 seconds                27017/tcp                           mongo
212e9ae415a   mysql                                             "docker-entrypoint.s…"   13 days ago      Up 13 days                   0.0.0.0:3306->3306/tcp, 33060/tcp   mysql
d48f1e1fee7   ofrir119/kafka:2.4.0                              "supervisord -n"         2 weeks ago      Exited (0) 13 days ago                                           kafka
58171c04fe4   bde2020/hive:2.3.2-postgresql-metastore           "entrypoint.sh /opt/…"   2 months ago     Exited (143) 2 months ago                                        docker-hive-hive-metastore-1
e241645f9cc   gethue/hue                                        "./startup.sh"           2 months ago     Exited (137) 2 months ago                                        docker-hive-hue-1
f5959428f70   bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-namenode-1
ab2e686a1a4   bde2020/hive:2.3.2-postgresql-metastore           "entrypoint.sh /bin/…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-hive-server-1
8018166a486   bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-datanode-1
aa8c4c4ca7f   bde2020/hive-metastore-postgresql:2.3.0           "/docker-entrypoint.…"   2 months ago     Exited (0) 2 months ago                                          docker-hive-hive-metastore-postgresql-1
```

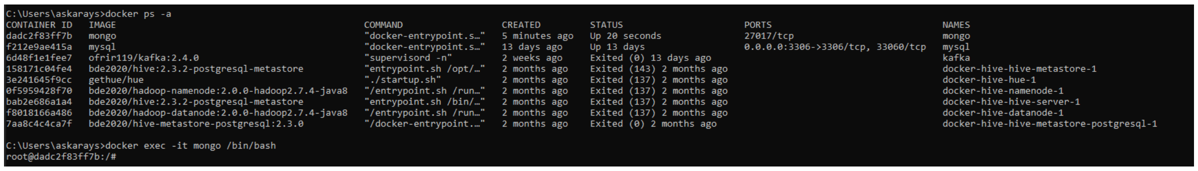
2. CONNECT TO THE THE MONGODB ENVIRONMENT

   o Verify the “MongoDB” environment is up and running:

   docker ps -a

   o Open a BASH session to the practice environment

   docker exec -it mongo /bin/bash

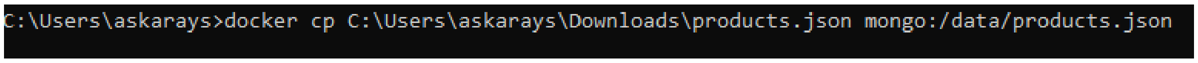


```
C:\Users\askarays>docker ps -a
CONTAINER ID   IMAGE                                             COMMAND                  CREATED          STATUS                       PORTS                               NAMES
dadc2f83ff7b   mongo                                             "docker-entrypoint.s…"   5 minutes ago    Up 20 seconds                27017/tcp                           mongo
f212e9ae415a   mysql                                             "docker-entrypoint.s…"   13 days ago      Up 13 days                   0.0.0.0:3306->3306/tcp, 33060/tcp   mysql
6d48f1e1fee7   ofrir119/kafka:2.4.0                              "supervisord -n"         2 weeks ago      Exited (0) 13 days ago                                           kafka
158171c04fe4   bde2020/hive:2.3.2-postgresql-metastore           "entrypoint.sh /opt/…"   2 months ago     Exited (143) 2 months ago                                        docker-hive-hive-metastore-1
3e241645f9cc   gethue/hue                                        "./startup.sh"           2 months ago     Exited (137) 2 months ago                                        docker-hive-hue-1
0f5959428f70   bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-namenode-1
bab2e686a1a4   bde2020/hive:2.3.2-postgresql-metastore           "entrypoint.sh /bin/…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-hive-server-1
f8018166a486   bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   2 months ago     Exited (137) 2 months ago                                        docker-hive-datanode-1
7aa8c4c4ca7f   bde2020/hive-metastore-postgresql:2.3.0           "/docker-entrypoint.…"   2 months ago     Exited (0) 2 months ago                                          docker-hive-hive-metastore-postgresql-1

C:\Users\askarays>docker exec -it mongo /bin/bash
root@dadc2f83ff7b:/#
```

3. GENERAL DETAILS AND PRACTICE PREPARATION (5)

   o Download the “products.json” file to your computer (for example, to “c:\temp”) and copy it to “/data/products.json” in the
   Docker container.

   ▪ See the Guidelines documents if you require assistance on this.



```
C:\Users\askarays>docker cp C:\Users\askarays\Downloads\products.json mongo:/data/products.json
```

4. IMPORT PRODUCTS DATA INTO MONGODB (15)

   o Import the products information from the JSON file you have loaded into MongoDB.

   ▪ Import into a collection named “products” and a database name “epam”

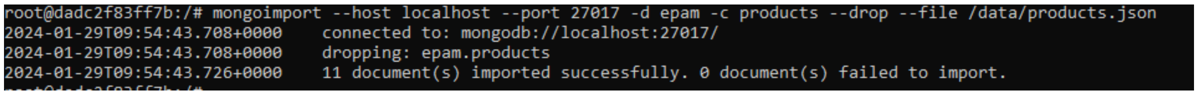
```
mongoimport --host localhost --port 27017 -d epam -c products --drop --file
/data/products.json
```

   ▪ Specify the default MongoDB port in the relevant parameter

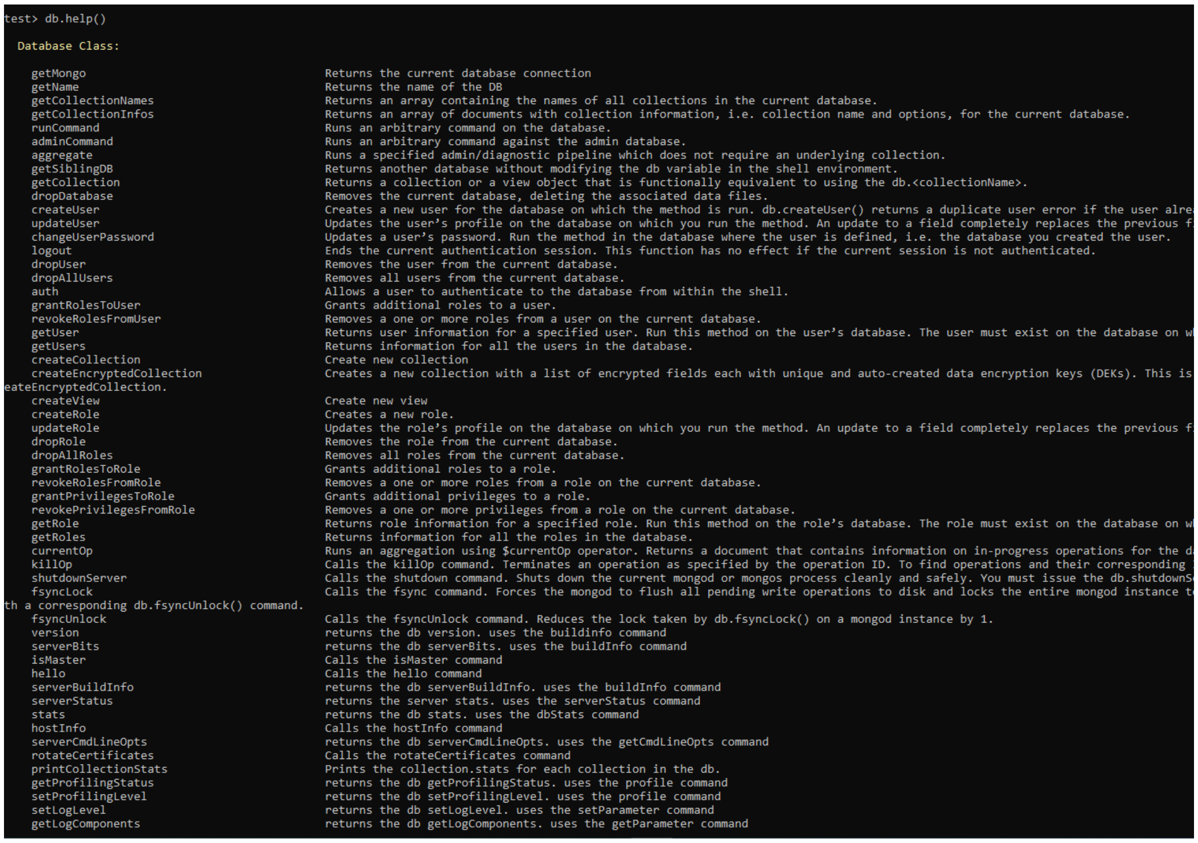
   ▪ Specify an option so that the collection will be dropped if it exists before loading the new data

• View the relevant command options using “--help" to find the relevant option

▪ See the Guidelines documents if you require assistance on this.



```
root@dadc2f83ff7b:/# mongoimport --host localhost --port 27017 -d epam -c products --drop --file /data/products.json
2024-01-29T09:54:43.708+0000    connected to: mongodb://localhost:27017/
2024-01-29T09:54:43.708+0000    dropping: epam.products
2024-01-29T09:54:43.726+0000    11 document(s) imported successfully. 0 document(s) failed to import.
```



```
test> db.help()

  Database Class:

    getMongo                                   Returns the current database connection
    getName                                    Returns the name of the DB
    getCollectionNames                         Returns an array containing the names of all collections in the current database.
    getCollectionInfos                         Returns an array of documents with collection information, i.e. collection name and options, for the current database.
    runCommand                                 Runs an arbitrary command on the database.
    adminCommand                               Runs an arbitrary command against the admin database.
    aggregate                                  Runs a specified admin/diagnostic pipeline which does not require an underlying collection.
    getSiblingDB                               Returns another database without modifying the db variable in the shell environment.
    getCollection                              Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
    dropDatabase                               Removes the current database, deleting the associated data files.
    createUser                                 Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user alre
    updateUser                                 Updates the user’s profile on the database on which you run the method. An update to a field completely replaces the previous f
    changeUserPassword                         Updates a user’s password. Run the method in the database where the user is defined, i.e. the database you created the user.
    logout                                     Ends the current authentication session. This function has no effect if the current session is not authenticated.
    dropUser                                   Removes the user from the current database.
    dropAllUsers                               Removes all users from the current database.
    auth                                       Allows a user to authenticate to the database from within the shell.
    grantRolesToUser                           Grants additional roles to a user.
    revokeRolesFromUser                        Removes a one or more roles from a user on the current database.
    getUser                                    Returns user information for a specified user. Run this method on the user’s database. The user must exist on the database on w
    getUsers                                   Returns information for all the users in the database.
    createCollection                           Create new collection
    createEncryptedCollection                  Creates a new collection with a list of encrypted fields each with unique and auto-created data encryption keys (DEKs). This is
eateEncryptedCollection.
    createView                                 Create new view
    createRole                                 Creates a new role.
    updateRole                                 Updates the role’s profile on the database on which you run the method. An update to a field completely replaces the previous f
    dropRole                                   Removes the role from the current database.
    dropAllRoles                               Removes all roles from the current database.
    grantRolesToRole                           Grants additional roles to a role.
    revokeRolesFromRole                        Removes a one or more roles from a role on the current database.
    grantPrivilegesToRole                      Grants additional privileges to a role.
    revokePrivilegesFromRole                   Removes a one or more privileges from a role on the current database.
    getRole                                    Returns role information for a specified role. Run this method on the role’s database. The role must exist on the database on w
    getRoles                                   Returns information for all the roles in the database.
    currentOp                                  Runs an aggregation using $currentOp operator. Returns a document that contains information on in-progress operations for the d
    killOp                                     Calls the killOp command. Terminates an operation as specified by the operation ID. To find operations and their corresponding
    shutdownServer                             Calls the shutdown command. Shuts down the current mongod or mongos process cleanly and safely. You must issue the db.shutdownS
    fsyncLock                                  Calls the fsync command. Forces the mongod to flush all pending write operations to disk and locks the entire mongod instance t
th a corresponding db.fsyncUnlock() command.
    fsyncUnlock                                Calls the fsyncUnlock command. Reduces the lock taken by db.fsyncLock() on a mongod instance by 1.
    version                                    returns the db version. uses the buildinfo command
    serverBits                                 returns the db serverBits. uses the buildInfo command
    isMaster                                   Calls the isMaster command
    hello                                      Calls the hello command
    serverBuildInfo                            returns the db serverBuildInfo. uses the buildInfo command
    serverStatus                               returns the server stats. uses the serverStatus command
    stats                                      returns the db stats. uses the dbStats command
    hostInfo                                   Calls the hostInfo command
    serverCmdLineOpts                          returns the db serverCmdLineOpts. uses the getCmdLineOpts command
    rotateCertificates                         Calls the rotateCertificates command
    printCollectionStats                       Prints the collection.stats for each collection in the db.
    getProfilingStatus                         returns the db getProfilingStatus. uses the profile command
    setProfilingLevel                          returns the db setProfilingLevel. uses the profile command
    setLogLevel                                returns the db setLogLevel. uses the setParameter command
    getLogComponents                           returns the db getLogComponents. uses the getParameter command
```

5. VERIFY THE LOADED DATA IN MONGODB (20)

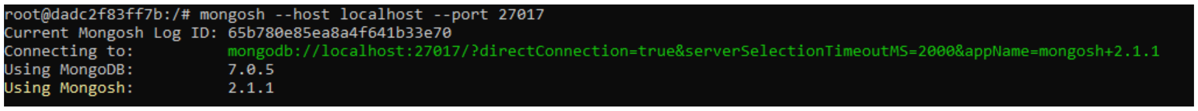
o Login to MongoDB

▪ Do we have to specify the hostname and port number? Why?

LocalHost and default port for MongoDB is 27017

▪ What is the MongoDB version?

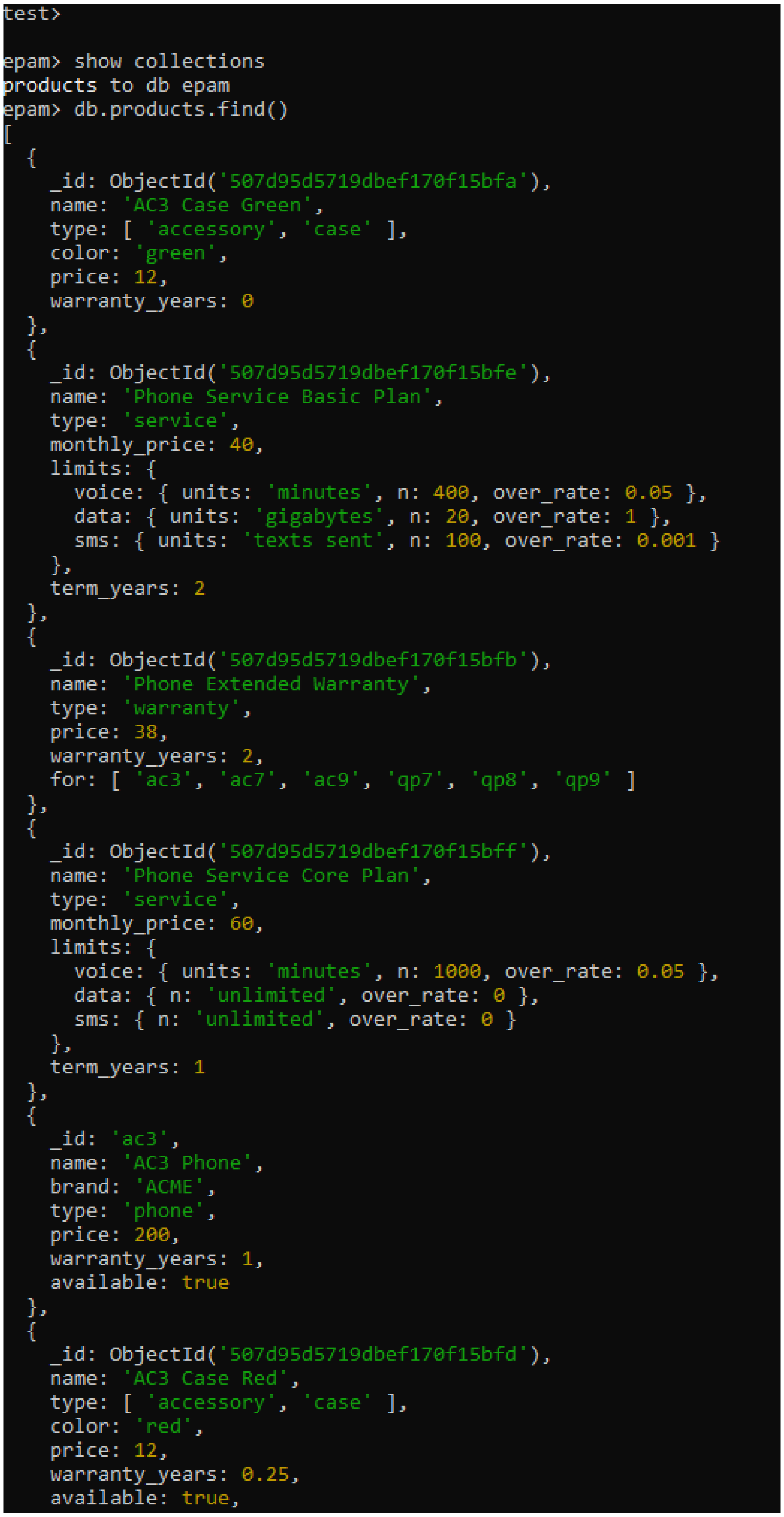
`Using MongoDB: 7.0.5`



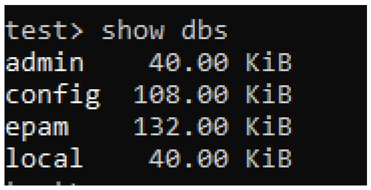
```
root@dadc2f83ff7b:/# mongosh --host localhost --port 27017
Current Mongosh Log ID: 65b780e85ea8a4f641b33e70
Connecting to:          mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.1
Using MongoDB:          7.0.5
Using Mongosh:          2.1.1
```

o Check – what options are available in MongoDB for the following:
▪ Databases
▪ Collection
▪ Find options in collections
▪ See the Guidelines documents if you require assistance on this.

```
use epam  // switch to "epam" databases
show dbs
show collections  //display collections in the current dat
abase
db.products.find()  // display all documents from the "pro
ducts" collection
db.products.count()  // display the document quantity
```


```
test>

epam> show collections
products to db epam
epam> db.products.find()
[
  {
    _id: ObjectId('507d95d5719dbef170f15bfa'),
    name: 'AC3 Case Green',
    type: [ 'accessory', 'case' ],
    color: 'green',
    price: 12,
    warranty_years: 0
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bfe'),
    name: 'Phone Service Basic Plan',
    type: 'service',
    monthly_price: 40,
    limits: {
      voice: { units: 'minutes', n: 400, over_rate: 0.05 },
      data: { units: 'gigabytes', n: 20, over_rate: 1 },
      sms: { units: 'texts sent', n: 100, over_rate: 0.001 }
    },
    term_years: 2
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bfb'),
    name: 'Phone Extended Warranty',
    type: 'warranty',
    price: 38,
    warranty_years: 2,
    for: [ 'ac3', 'ac7', 'ac9', 'qp7', 'qp8', 'qp9' ]
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bff'),
    name: 'Phone Service Core Plan',
    type: 'service',
    monthly_price: 60,
    limits: {
      voice: { units: 'minutes', n: 1000, over_rate: 0.05 },
      data: { n: 'unlimited', over_rate: 0 },
      sms: { n: 'unlimited', over_rate: 0 }
    },
    term_years: 1
  },
  {
    _id: 'ac3',
    name: 'AC3 Phone',
    brand: 'ACME',
    type: 'phone',
    price: 200,
    warranty_years: 1,
    available: true
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bfd'),
    name: 'AC3 Case Red',
    type: [ 'accessory', 'case' ],
    color: 'red',
    price: 12,
    warranty_years: 0.25,
    available: true,
```

o Check – Which databases currently exist in this MongoDB instance?



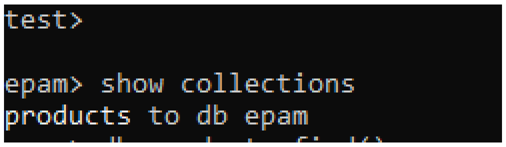
```
test> show dbs
admin     40.00 KiB
config   108.00 KiB
epam     132.00 KiB
local     40.00 KiB
```

o Switch to use the database named “epam”

`use epam`

o Check – Which collections currently exist in the database “epam”?

`show collections`



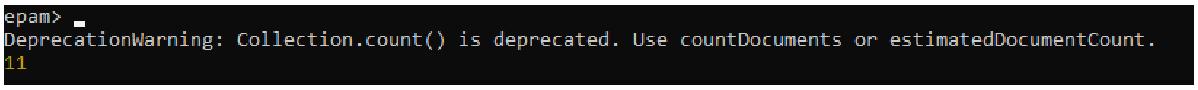
```
test>

epam> show collections
products to db epam
```

o List all data in the collection “products”

```
epam> db.products.find()
[
{
_id: ObjectId('507d95d5719dbef170f15bfa'),
name: 'AC3 Case Green',
type: [ 'accessory', 'case' ],
color: 'green',
price: 12,
warranty_years: 0
},
{
_id: ObjectId('507d95d5719dbef170f15bfe'),
name: 'Phone Service Basic Plan',
type: 'service',
monthly_price: 40,
limits: {
voice: { units: 'minutes', n: 400, over_rate: 0.05 },
data: { units: 'gigabytes', n: 20, over_rate: 1 },
sms: { units: 'texts sent', n: 100, over_rate: 0.001 }
},
term_years: 2
},
{
_id: ObjectId('507d95d5719dbef170f15bfb'),
name: 'Phone Extended Warranty',
type: 'warranty',
price: 38,
warranty_years: 2,
for: [ 'ac3', 'ac7', 'ac9', 'qp7', 'qp8', 'qp9' ]
},
{
_id: ObjectId('507d95d5719dbef170f15bff'),
name: 'Phone Service Core Plan',
type: 'service',
monthly_price: 60,
limits: {
voice: { units: 'minutes', n: 1000, over_rate: 0.05 },
data: { n: 'unlimited', over_rate: 0 },
sms: { n: 'unlimited', over_rate: 0 }
},
term_years: 1
},
{
_id: 'ac3',
name: 'AC3 Phone',
brand: 'ACME',
type: 'phone',
price: 200,
warranty_years: 1,
available: true
},
{
_id: ObjectId('507d95d5719dbef170f15bfd'),
name: 'AC3 Case Red',
type: [ 'accessory', 'case' ],
color: 'red',
price: 12,
warranty_years: 0.25,
available: true,
for: 'ac3'
},
{
_id: ObjectId('507d95d5719dbef170f15c01'),
name: 'Cable TV Basic Service Package',
type: 'tv',
monthly_price: 50,
term_years: 2,
cancel_penalty: 25,
sales_tax: true,
additional_tarriffs: [
{ kind: 'federal tarriff', amount: { percent_of_service: 0.06 } },
{ kind: 'misc tarriff', amount: 2.25 }
]
},
{
_id: 'ac7',
name: 'AC7 Phone',
brand: 'ACME',
type: 'phone',
price: 320,
warranty_years: 1,
available: false
},
{
_id: ObjectId('507d95d5719dbef170f15bfc'),
name: 'AC3 Case Black',
type: [ 'accessory', 'case' ],
color: 'black',
price: 12.5,
warranty_years: 0.25,
available: false,
for: 'ac3'
},
{
_id: ObjectId('507d95d5719dbef170f15bf9'),
name: 'AC3 Series Charger',
type: [ 'accessory', 'charger' ],
price: 19,
warranty_years: 0.25,
for: [ 'ac3', 'ac7', 'ac9' ]
},
{
_id: ObjectId('507d95d5719dbef170f15c00'),
name: 'Phone Service Family Plan',
type: 'service',
monthly_price: 90,
limits: {
voice: { units: 'minutes', n: 1200, over_rate: 0.05 },
data: { n: 'unlimited', over_rate: 0 },
sms: { n: 'unlimited', over_rate: 0 }
},
sales_tax: true,
term_years: 2
}
]
```

o Check – How many documents currently exist in this collection?

11



```
epam>
DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
11
```

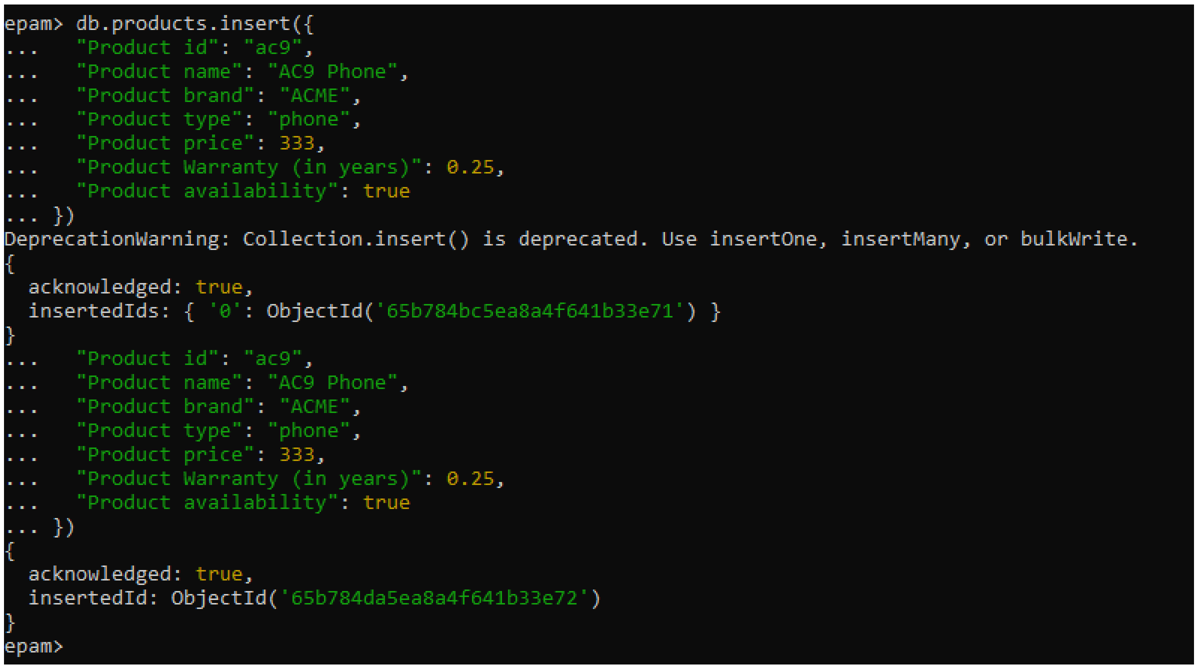
6. CRUD OPERATIONS IN MONGODB COLLECTIONS (40)

o Insert the following new document to the “products” collection with the following attributes:

▪ Product id: “ac9”

▪ Product name: “AC9 Phone”
▪ Product brand: “ACME”
▪ Product type: “phone”
▪ Product price: 333
▪ Product Warranty (in years): 0.25
▪ Product availability: true

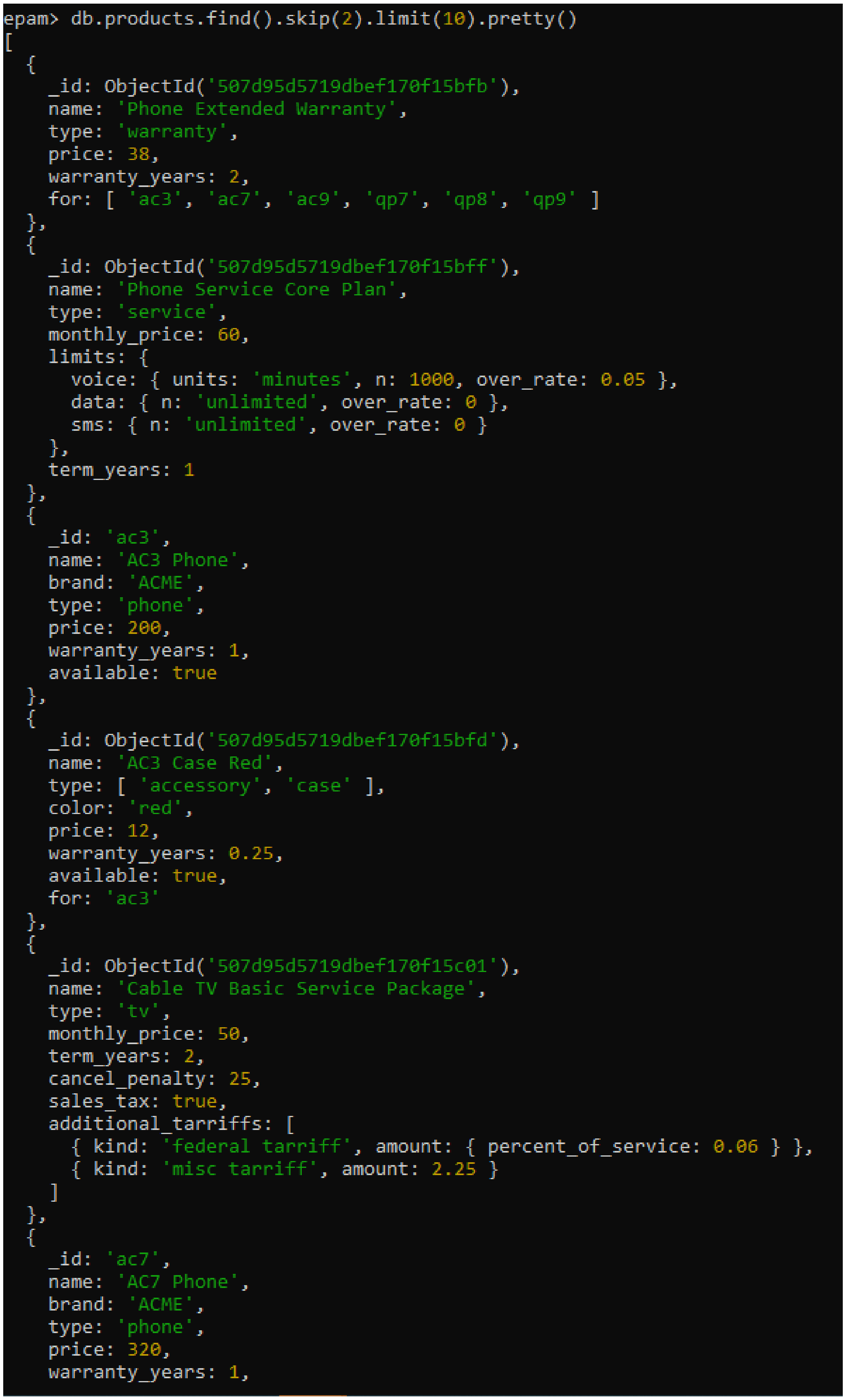
```
db.products.insertOne({
"Product id": "ac9",
"Product name": "AC9 Phone",
"Product brand": "ACME",
"Product type": "phone",
"Product price": 333,
"Product Warranty (in years)": 0.25,
"Product availability": true
})
```



```
epam> db.products.insert({
...   "Product id": "ac9",
...   "Product name": "AC9 Phone",
...   "Product brand": "ACME",
...   "Product type": "phone",
...   "Product price": 333,
...   "Product Warranty (in years)": 0.25,
...   "Product availability": true
... })
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('65b784bc5ea8a4f641b33e71') }
}
...   "Product id": "ac9",
...   "Product name": "AC9 Phone",
...   "Product brand": "ACME",
...   "Product type": "phone",
...   "Product price": 333,
...   "Product Warranty (in years)": 0.25,
...   "Product availability": true
... })
{
  acknowledged: true,
  insertedId: ObjectId('65b784da5ea8a4f641b33e72')
}
epam>
```

o Perform queries to display products according to the following requirements:
▪ Query 1:
• Skip the first 2 products and display the next 10 products in the collection.

• Make the output in an easy to read JSON format. (Each field and its value should appear in a separate row)

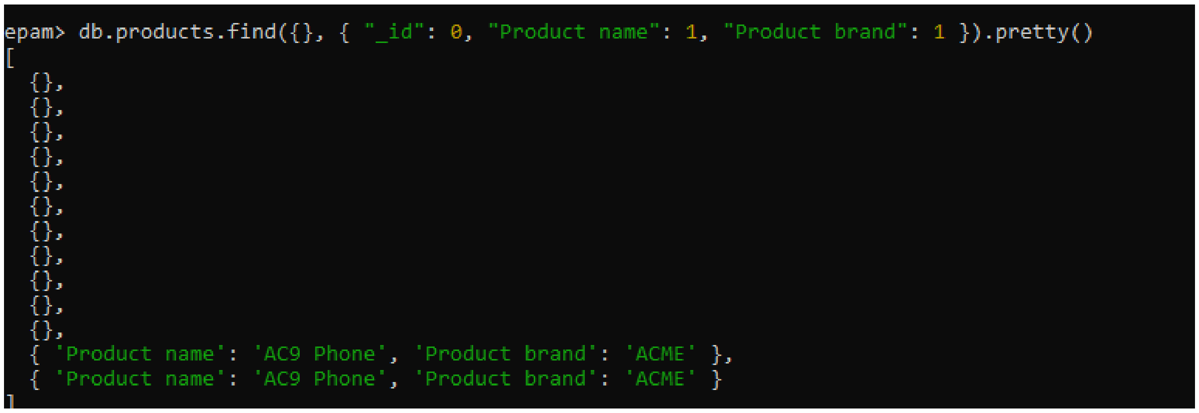


```
epam> db.products.find().skip(2).limit(10).pretty()
[
  {
    _id: ObjectId('507d95d5719dbef170f15bfb'),
    name: 'Phone Extended Warranty',
    type: 'warranty',
    price: 38,
    warranty_years: 2,
    for: [ 'ac3', 'ac7', 'ac9', 'qp7', 'qp8', 'qp9' ]
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bff'),
    name: 'Phone Service Core Plan',
    type: 'service',
    monthly_price: 60,
    limits: {
      voice: { units: 'minutes', n: 1000, over_rate: 0.05 },
      data: { n: 'unlimited', over_rate: 0 },
      sms: { n: 'unlimited', over_rate: 0 }
    },
    term_years: 1
  },
  {
    _id: 'ac3',
    name: 'AC3 Phone',
    brand: 'ACME',
    type: 'phone',
    price: 200,
    warranty_years: 1,
    available: true
  },
  {
    _id: ObjectId('507d95d5719dbef170f15bfd'),
    name: 'AC3 Case Red',
    type: [ 'accessory', 'case' ],
    color: 'red',
    price: 12,
    warranty_years: 0.25,
    available: true,
    for: 'ac3'
  },
  {
    _id: ObjectId('507d95d5719dbef170f15c01'),
    name: 'Cable TV Basic Service Package',
    type: 'tv',
    monthly_price: 50,
    term_years: 2,
    cancel_penalty: 25,
    sales_tax: true,
    additional_tarriffs: [
      { kind: 'federal tarriff', amount: { percent_of_service: 0.06 } },
      { kind: 'misc tarriff', amount: 2.25 }
    ]
  },
  {
    _id: 'ac7',
    name: 'AC7 Phone',
    brand: 'ACME',
    type: 'phone',
    price: 320,
    warranty_years: 1,
```

▪ Query 2:

• Display only the “name” and “brand” fields for each product.

```
db.products.find({}, { "_id": 0, "Product name": 1, "Product brand": 1 }).pretty()
```



```
epam> db.products.find({}, { "_id": 0, "Product name": 1, "Product brand": 1 }).pretty()
[
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  { 'Product name': 'AC9 Phone', 'Product brand': 'ACME' },
  { 'Product name': 'AC9 Phone', 'Product brand': 'ACME' }
]
```

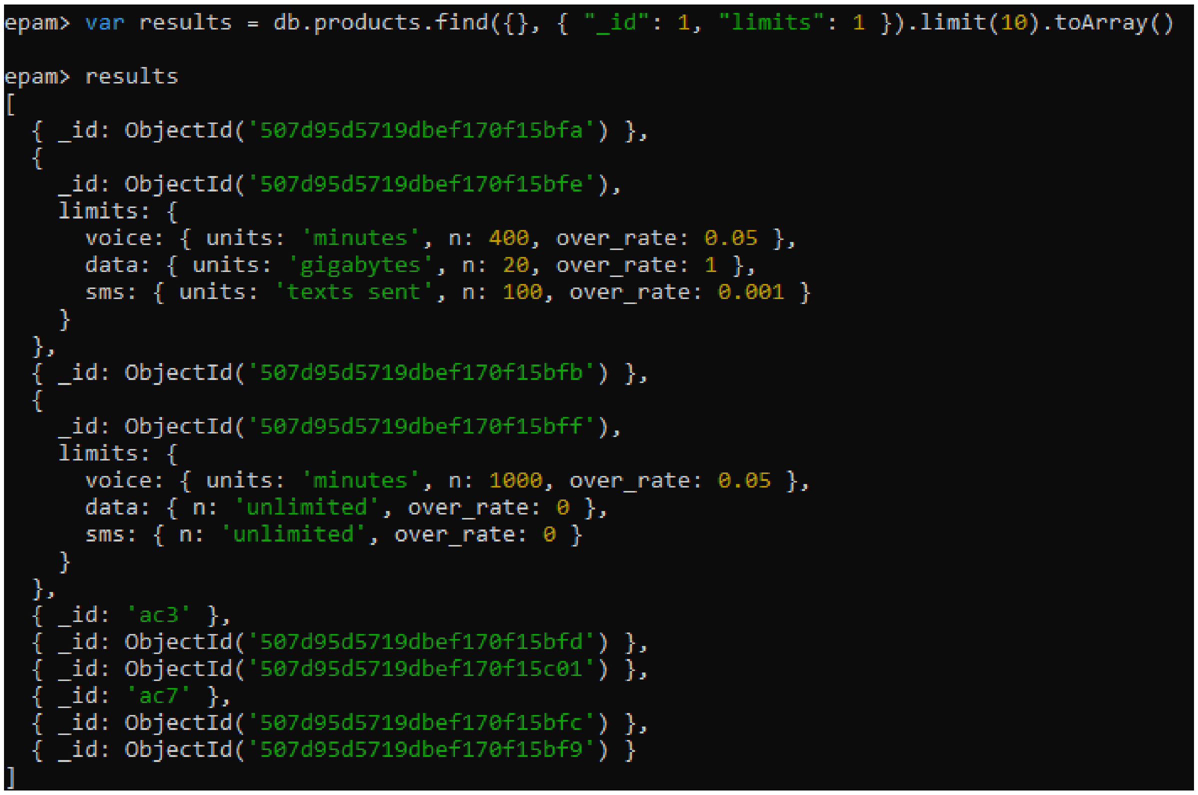
▪ Query 3:

• Display only the “id” and “limits” fields for the first 10 products

• Collect the results into a single array, in which each element is both “id” and “limits” of a specific product.

• Examine the result you have received:

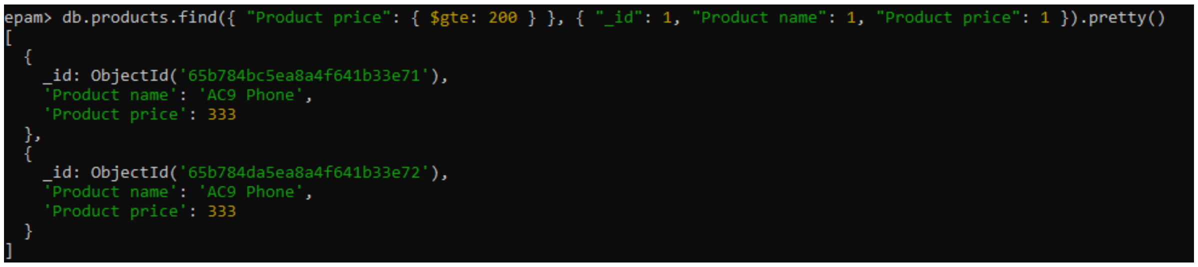
Did all “id” values had a matching “limits” value? Why so?



```
epam> var results = db.products.find({}, { "_id": 1, "limits": 1 }).limit(10).toArray()

epam> results
[
  { _id: ObjectId('507d95d5719dbef170f15bfa') },
  {
    _id: ObjectId('507d95d5719dbef170f15bfe'),
    limits: {
      voice: { units: 'minutes', n: 400, over_rate: 0.05 },
      data: { units: 'gigabytes', n: 20, over_rate: 1 },
      sms: { units: 'texts sent', n: 100, over_rate: 0.001 }
    }
  },
  { _id: ObjectId('507d95d5719dbef170f15bfb') },
  {
    _id: ObjectId('507d95d5719dbef170f15bff'),
    limits: {
      voice: { units: 'minutes', n: 1000, over_rate: 0.05 },
      data: { n: 'unlimited', over_rate: 0 },
      sms: { n: 'unlimited', over_rate: 0 }
    }
  },
  { _id: 'ac3' },
  { _id: ObjectId('507d95d5719dbef170f15bfd') },
  { _id: ObjectId('507d95d5719dbef170f15c01') },
  { _id: 'ac7' },
  { _id: ObjectId('507d95d5719dbef170f15bfc') },
  { _id: ObjectId('507d95d5719dbef170f15bf9') }
]
```

▪ Query 4:

• Display the IDs, names and prices of all products of which prices are greater or equal to 200.

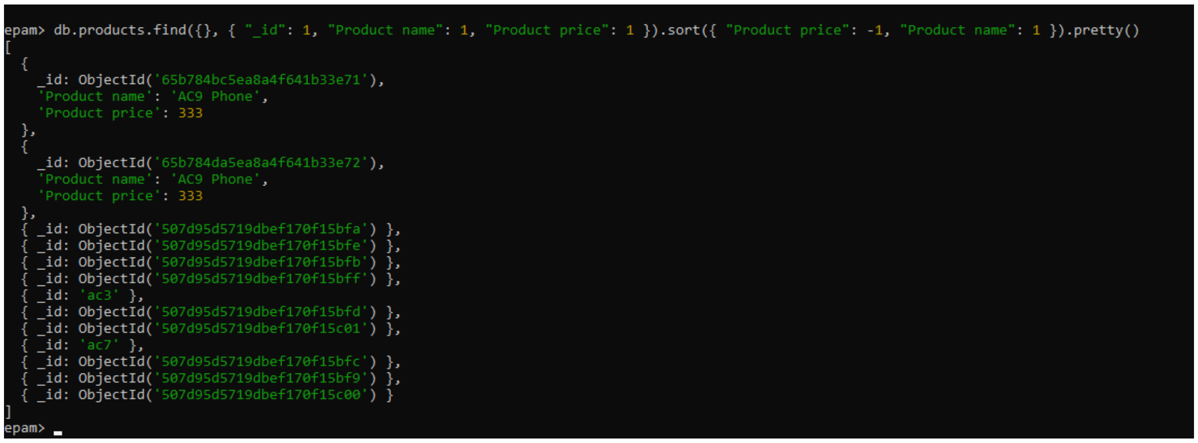


```
epam> db.products.find({ "Product price": { $gte: 200 } }, { "_id": 1, "Product name": 1, "Product price": 1 }).pretty()
[
  {
    _id: ObjectId('65b784bc5ea8a4f641b33e71'),
    'Product name': 'AC9 Phone',
    'Product price': 333
  },
  {
    _id: ObjectId('65b784da5ea8a4f641b33e72'),
    'Product name': 'AC9 Phone',
    'Product price': 333
  }
]
```

▪ Query 5:

• Display the IDs, names and prices of all products.

• Sort the result according to price in descending order and name in ascending order (secondary sort)



```
epam> db.products.find({}, { "_id": 1, "Product name": 1, "Product price": 1 }).sort({ "Product price": -1, "Product name": 1 }).pretty()
[
  {
    _id: ObjectId('65b784bc5ea8a4f641b33e71'),
    'Product name': 'AC9 Phone',
    'Product price': 333
  },
  {
    _id: ObjectId('65b784da5ea8a4f641b33e72'),
    'Product name': 'AC9 Phone',
    'Product price': 333
  },
  { _id: ObjectId('507d95d5719dbef170f15bfa') },
  { _id: ObjectId('507d95d5719dbef170f15bfe') },
  { _id: ObjectId('507d95d5719dbef170f15bfb') },
  { _id: ObjectId('507d95d5719dbef170f15bff') },
  { _id: 'ac3' },
  { _id: ObjectId('507d95d5719dbef170f15bfd') },
  { _id: ObjectId('507d95d5719dbef170f15c01') },
  { _id: 'ac7' },
  { _id: ObjectId('507d95d5719dbef170f15bfc') },
  { _id: ObjectId('507d95d5719dbef170f15bf9') },
  { _id: ObjectId('507d95d5719dbef170f15c00') }
]
epam> _
```

▪ Query 6:

• Write a query that displays how many products we have of type “service”. (Check the field which is named “type”)

o Updating records

▪ General questions

• Can we update the “_id” field? Why so?

In MongoDB, you cannot update the value of the _id field in an existing document after it has been inserted. _id is unique for each document and is used to uniquely identify it. Changing _id may break uniqueness and cause unexpected problems.

• When should we use the “set” keyword? What happens if we omit it?

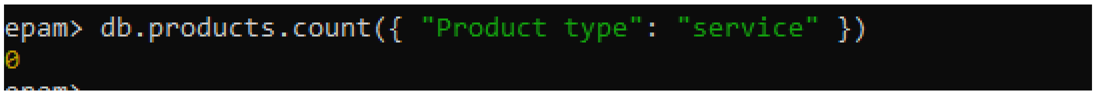
The "set" keyword is used in MongoDB update operations to specify the fields and their values that wanted to set or change in the document.

If we omit "set", the entire document will be replaced with the new values provided. This means that any existing fields not included in the update will be removed, and only the fields specified in the update will be present in the modified document.

• When should use the “multi” keyword?

- The "multi" keyword is used to update multiple documents that match the specified criteria. If "multi" is not specified, only the first document that matches the criteria will be updated.
- Omitting "multi" will update only the first document found that matches the query criteria.

  ▪ Please perform a query after each of the following updates to verify you have updates the documents as expected.



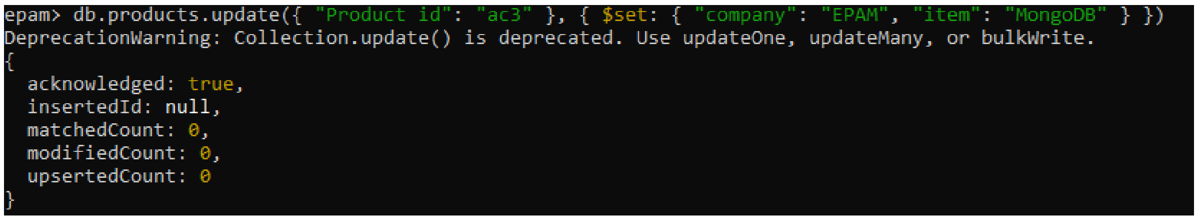
```
epam> db.products.count({ "Product type": "service" })
0
```

▪ Update 1:

• Update product with ID “ac3”, so that he will now have only the following field values:

• company: “EPAM”

• item: “MongoDB”



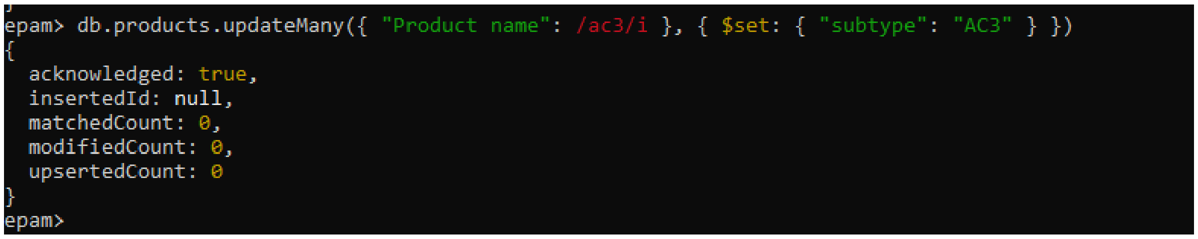
```
epam> db.products.update({ "Product id": "ac3" }, { $set: { "company": "EPAM", "item": "MongoDB" } })
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
```

▪ Update 2:

• Update all products which have “ac3” somewhere in their name, and add a new field to their document –

“subtype” with the value “AC3”.

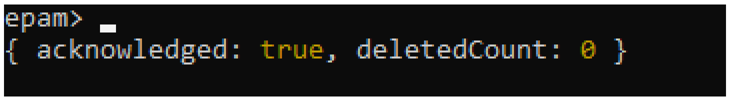
Note that the “ac3” string in the name can be either lower or upper case.



```
epam> db.products.updateMany({ "Product name": /ac3/i }, { $set: { "subtype": "AC3" } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
epam>
```

o Deleting records

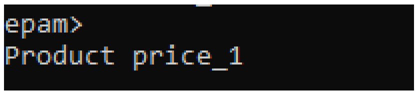
▪ Remove all records of type “service”.



```
epam>
{ acknowledged: true, deletedCount: 0 }
```

7. USING INDEXES (5)

o Create an index for the “price” field

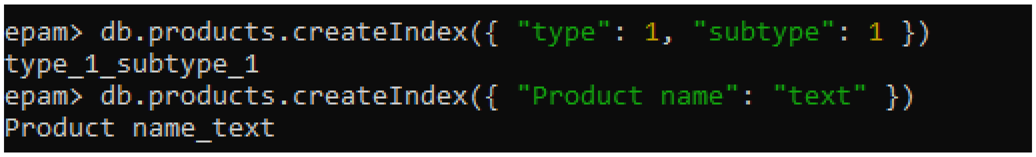


```
epam>
Product price_1
```

o Create a compound index for “type” and “subtype” fields

o Create a text index for the “name” field.

▪ What is the benefit of a text index over a regular index?



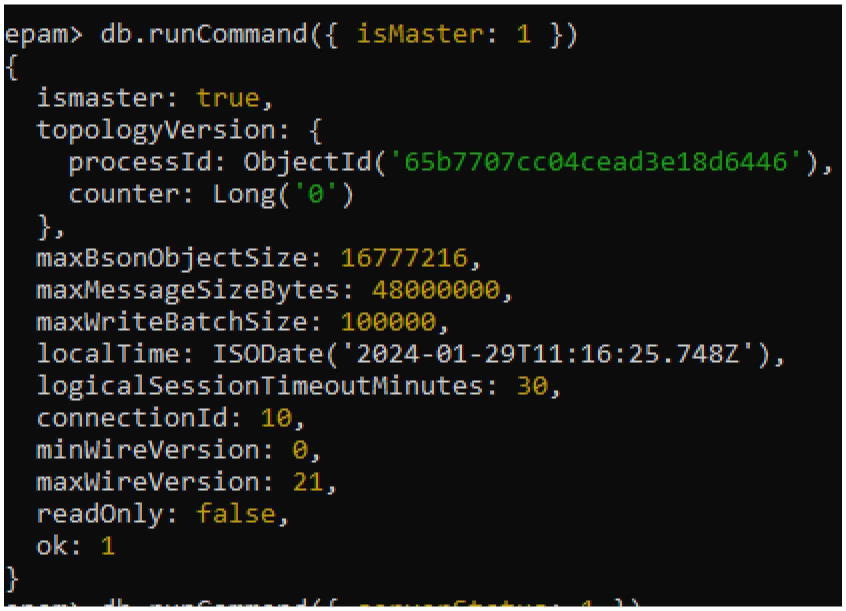
```
epam> db.products.createIndex({ "type": 1, "subtype": 1 })
type_1_subtype_1
epam> db.products.createIndex({ "Product name": "text" })
Product name_text
```

8. ARCHITECTURE AND MONITORING (15)

o Consult the guidelines document if required for assistance on the following requirements.

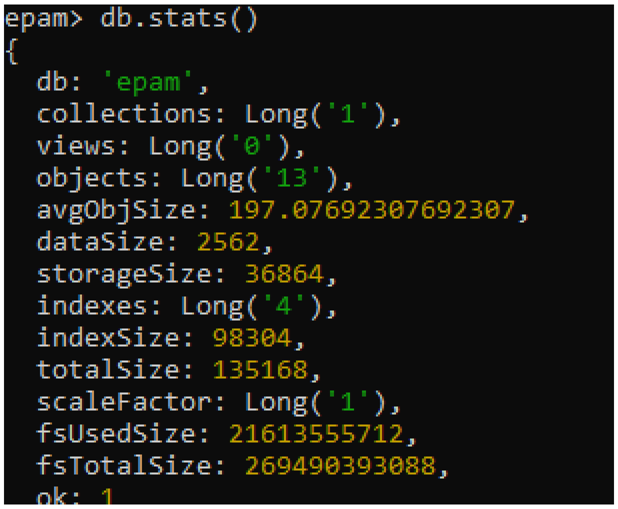
o Run a command which describes the current MongoDB node.

▪ Change the command to display only the local time of the current instance.

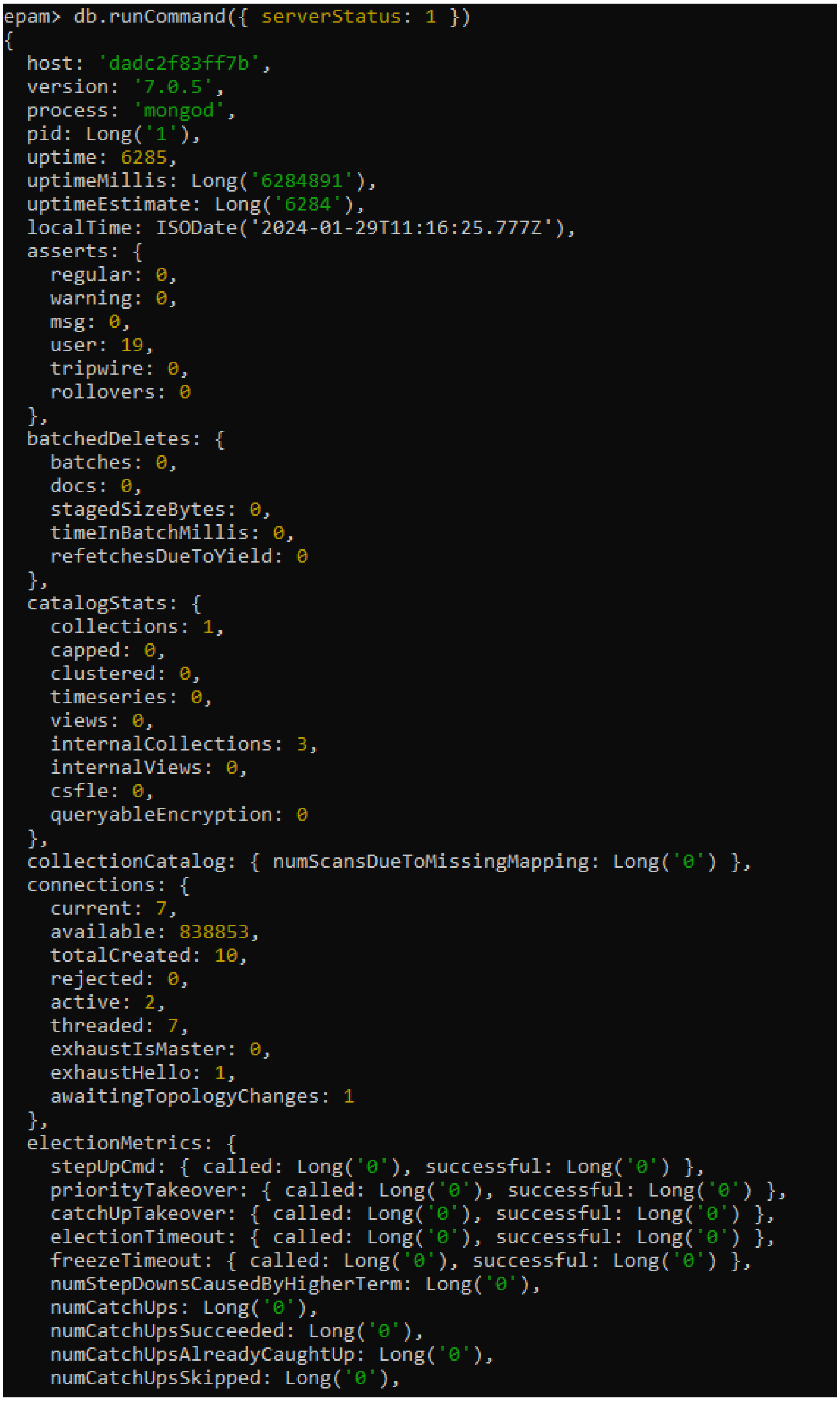


```
epam> db.runCommand({ isMaster: 1 })
{
  ismaster: true,
  topologyVersion: {
    processId: ObjectId('65b7707cc04cead3e18d6446'),
    counter: Long('0')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2024-01-29T11:16:25.748Z'),
  logicalSessionTimeoutMinutes: 30,
  connectionId: 10,
  minWireVersion: 0,
  maxWireVersion: 21,
  readOnly: false,
  ok: 1
}
```

o Run a command which describes the current state of the database, with all its metrics and stats.

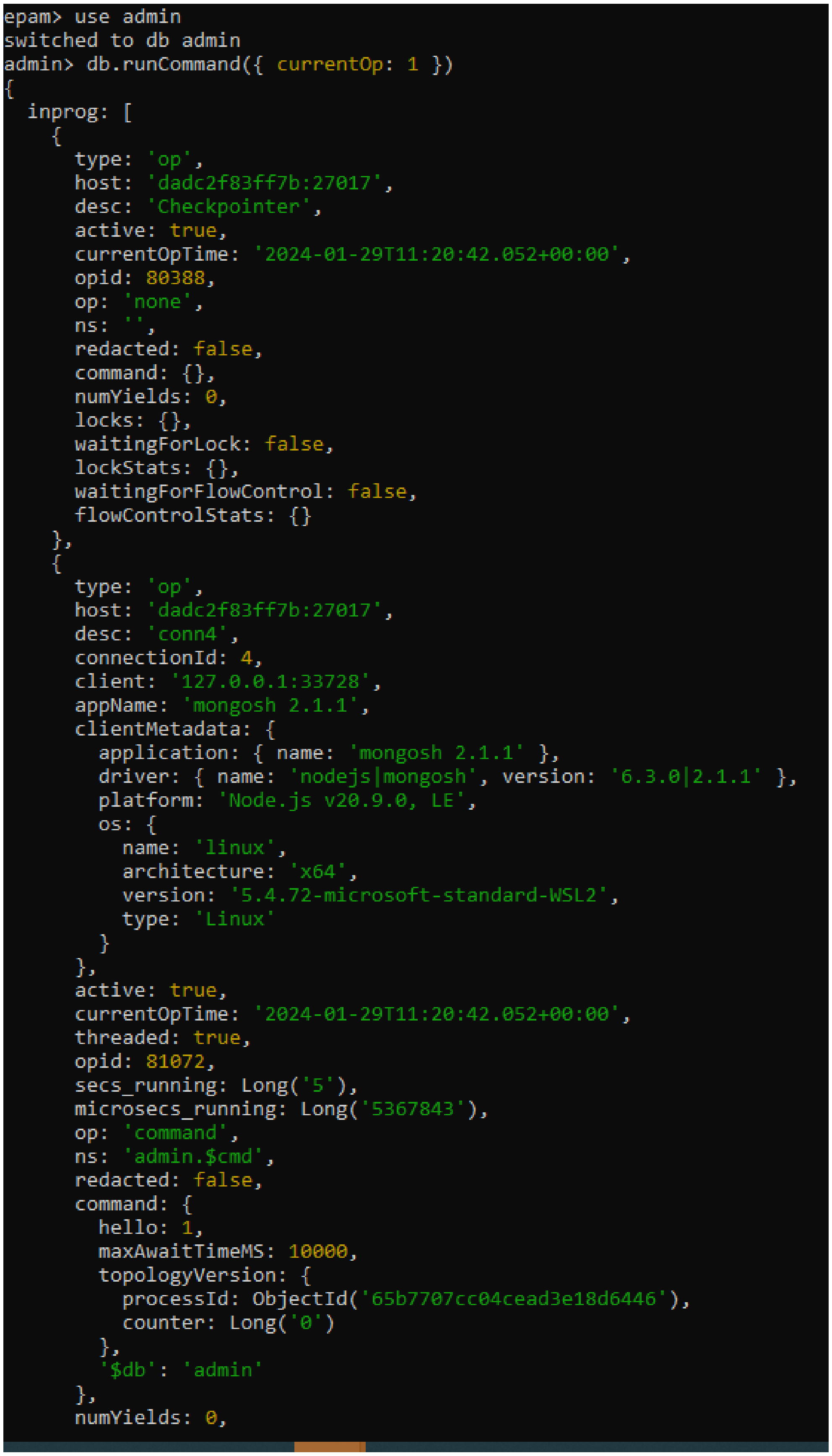


```
epam> db.stats()
{
  db: 'epam',
  collections: Long('1'),
  views: Long('0'),
  objects: Long('13'),
  avgObjSize: 197.07692307692307,
  dataSize: 2562,
  storageSize: 36864,
  indexes: Long('4'),
  indexSize: 98304,
  totalSize: 135168,
  scaleFactor: Long('1'),
  fsUsedSize: 21613555712,
  fsTotalSize: 269490393088,
  ok: 1
```



```
epam> db.runCommand({ serverStatus: 1 })
{
  host: 'dadc2f83ff7b',
  version: '7.0.5',
  process: 'mongod',
  pid: Long('1'),
  uptime: 6285,
  uptimeMillis: Long('6284891'),
  uptimeEstimate: Long('6284'),
  localTime: ISODate('2024-01-29T11:16:25.777Z'),
  asserts: {
    regular: 0,
    warning: 0,
    msg: 0,
    user: 19,
    tripwire: 0,
    rollovers: 0
  },
  batchedDeletes: {
    batches: 0,
    docs: 0,
    stagedSizeBytes: 0,
    timeInBatchMillis: 0,
    refetchesDueToYield: 0
  },
  catalogStats: {
    collections: 1,
    capped: 0,
    clustered: 0,
    timeseries: 0,
    views: 0,
    internalCollections: 3,
    internalViews: 0,
    csfle: 0,
    queryableEncryption: 0
  },
  collectionCatalog: { numScansDueToMissingMapping: Long('0') },
  connections: {
    current: 7,
    available: 838853,
    totalCreated: 10,
    rejected: 0,
    active: 2,
    threaded: 7,
    exhaustIsMaster: 0,
    exhaustHello: 1,
    awaitingTopologyChanges: 1
  },
  electionMetrics: {
    stepUpCmd: { called: Long('0'), successful: Long('0') },
    priorityTakeover: { called: Long('0'), successful: Long('0') },
    catchUpTakeover: { called: Long('0'), successful: Long('0') },
    electionTimeout: { called: Long('0'), successful: Long('0') },
    freezeTimeout: { called: Long('0'), successful: Long('0') },
    numStepDownsCausedByHigherTerm: Long('0'),
    numCatchUps: Long('0'),
    numCatchUpsSucceeded: Long('0'),
    numCatchUpsAlreadyCaughtUp: Long('0'),
    numCatchUpsSkipped: Long('0'),
```

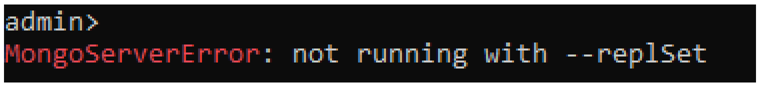
o Display information about all currently running operations in the database instance.



```
epam> use admin
switched to db admin
admin> db.runCommand({ currentOp: 1 })
{
  inprog: [
    {
      type: 'op',
      host: 'dadc2f83ff7b:27017',
      desc: 'Checkpointer',
      active: true,
      currentOpTime: '2024-01-29T11:20:42.052+00:00',
      opid: 80388,
      op: 'none',
      ns: '',
      redacted: false,
      command: {},
      numYields: 0,
      locks: {},
      waitingForLock: false,
      lockStats: {},
      waitingForFlowControl: false,
      flowControlStats: {}
    },
    {
      type: 'op',
      host: 'dadc2f83ff7b:27017',
      desc: 'conn4',
      connectionId: 4,
      client: '127.0.0.1:33728',
      appName: 'mongosh 2.1.1',
      clientMetadata: {
        application: { name: 'mongosh 2.1.1' },
        driver: { name: 'nodejs|mongosh', version: '6.3.0|2.1.1' },
        platform: 'Node.js v20.9.0, LE',
        os: {
          name: 'linux',
          architecture: 'x64',
          version: '5.4.72-microsoft-standard-WSL2',
          type: 'Linux'
        }
      },
      active: true,
      currentOpTime: '2024-01-29T11:20:42.052+00:00',
      threaded: true,
      opid: 81072,
      secs_running: Long('5'),
      microsecs_running: Long('5367843'),
      op: 'command',
      ns: 'admin.$cmd',
      redacted: false,
      command: {
        hello: 1,
        maxAwaitTimeMS: 10000,
        topologyVersion: {
          processId: ObjectId('65b7707cc04cead3e18d6446'),
          counter: Long('0')
        },
        '$db': 'admin'
      },
      numYields: 0,
```

o Check – are replication sets currently enabled?

No rep.sets curently enabled



```
admin>
MongoServerError: not running with --replSet
```