Zhastay Yeltay

MONGODB PRACTICE

Note: The requirements for the practical task are just texts or scripts/codes are in bold, the other simple italic and yellowed text and screenshots are my answers.

CONTENTS

1. NOTES REGARDING THE PRACTICE

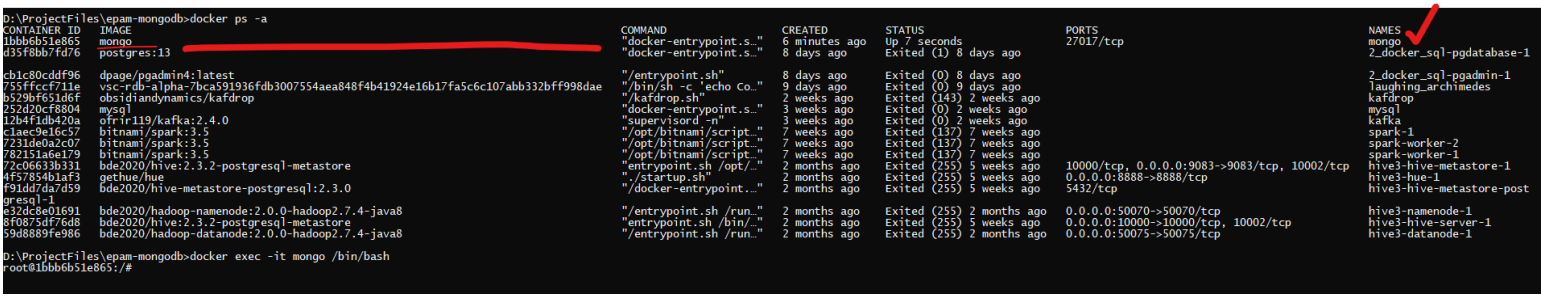
- When you see the dollar sign in a command you need to execute, note this is just a command prompt indication. You do not need to actually write the "$", as it is not part of the command. o For all practices below – write all the commands you have used in a "Practice answers document".

  When you are complete, you can submit this document for review.

2. CONNECT TO THE THE MONGODB ENVIRONMENT

- Verify the "MongoDB" environment is up and running: **docker ps -a**

- Open a BASH session to the practice environment **docker exec -it Mongo /bin/bash**



3. GENERAL DETAILS AND PRACTICE PREPARATION (5)

- 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.

I downloaded the file and moved it into project directory. After that send it to docker mongodb location using cmd-bash.

1bbb6b51e865 – this is my mongodb's id.

docker cp products.json 1bbb6b51e865:./data/products.json

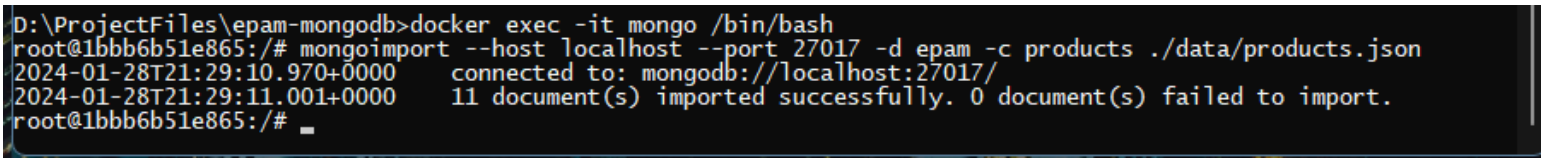
```
D:\ProjectFiles\epam-mongodb>docker cp products.json 1bbb6b51e865:./data/products.json
Successfully copied 4.61kB to 1bbb6b51e865:./data/products.json

D:\ProjectFiles\epam-mongodb>
```

4. IMPORT PRODUCTS DATA INTO MONGODB (15)

- Import the products information from the JSON file you have loaded into MongoDB.
  - Import into a collection named “products” and a database name “epam”
  - 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.

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



5. VERIFY THE LOADED DATA IN MONGODB (20)

- Login to MongoDB
  - Do we have to specify the hostname and port number? Why?
  - What is the MongoDB version?

I used just mongosh command for new version of mongodb.
Also you can specify –host and –port or other parameters if it needs. In my case I can connect to mongodb just using mongosh command with default parameters localhost:27017.

```
root@1bbb6b51e865:/# mongosh
Current Mongosh Log ID: 65b6ca82d95f5468ff57db1a
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.1.1
Using MongoDB:          7.0.5
Using Mongosh:          2.1.1

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2024-01-28T21:14:49.312+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-01-28T21:14:50.208+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-01-28T21:14:50.208+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
   2024-01-28T21:14:50.208+00:00: vm.max_map_count is too low
------

test>
test> db.version()
7.0.5
test>
```

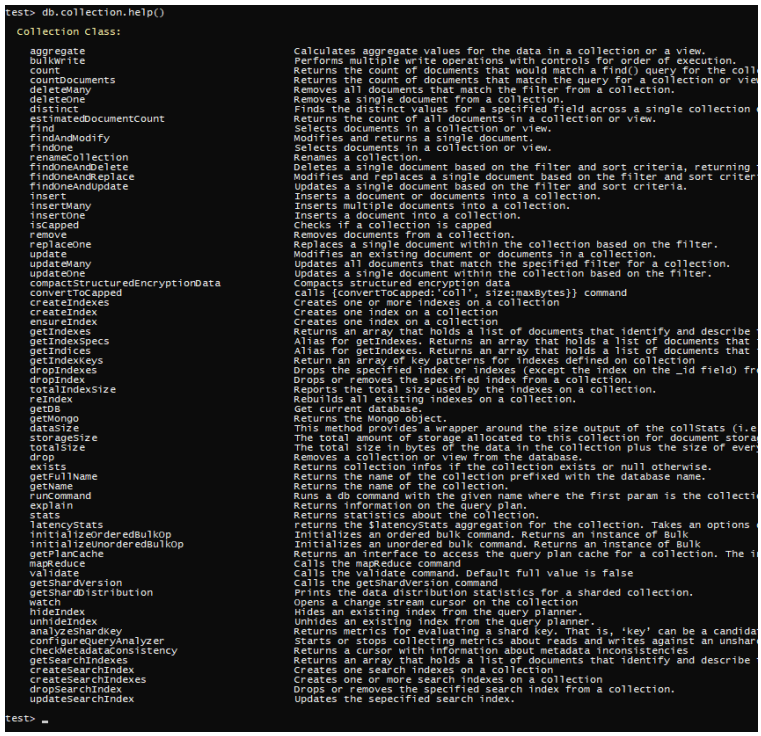
Version is 7.0.5

- Check – what options are available in MongoDB for the following:
  - Databases

Db.help()

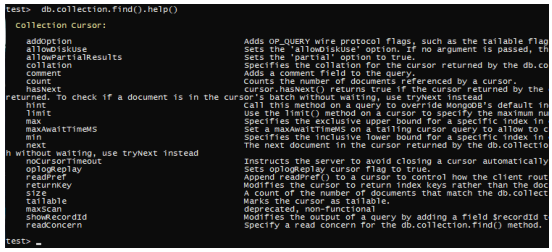
- Collection

db.collection.help()



- Find options in collections

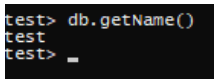
db.collection.find().help()



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

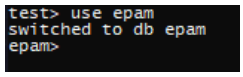
- Check – Which databases currently exist in this MongoDB instance?

db.getName()

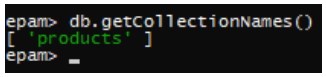


- Switch to use the database named “epam” o Check – Which collections currently exist in the database “epam”?

use epam

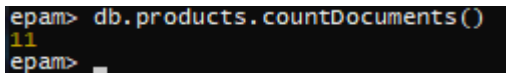


db.getCollectionNames()



- List all data in the collection “products” o Check – How many documents currently exist in this collection?

db.products.find(), db.products.countDocuments()



6. CRUD OPERATIONS IN MONGODB COLLECTIONS (40)

- 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({
  _id: "ac9",
  name: "AC9 Phone",
  brand: "ACME",
  type: "phone",
  price: 333,
  warranty_years: 0.25,
  available: true
})
```

```
epam> db.products.insertOne({
...   _id: "ac9",
...   name: "AC9 Phone",
...   brand: "ACME",
...   type: "phone",
...   price: 333,
...   warranty_years: 0.25,
...   available: true
... })
{ acknowledged: true, insertedId: 'ac9' }
epam> db.products.find()
[
```

- 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)

```
db.products.find().skip(2).limit(10)
```

```
epam> db.products.find().skip(2).limit(10)
[
  {
    _id: ObjectId('507d95d5719dbef170f15bfa'),
    name: 'AC3 Case Green',
    type: [ 'accessory', 'case' ],
    color: 'green',
    price: 12,
    warranty_years: 0
  },
  {
    _id: 'ac7',
    name: 'AC7 Phone',
    brand: 'ACME',
    type: 'phone',
    price: 320,
    warranty_years: 1,
    available: false
  },
  {
```

```
for (const myDoc of db.products.find().skip(2).limit(10) ) {
  for (var key in myDoc) {
    print(key + ": " + JSON.stringify(myDoc[key]));
  }
```

print("\n");
}

```
epam> for (const myDoc of db.products.find().skip(2).limit(10) ) {
...    for (var key in myDoc) {
...      print(key + ": " + JSON.stringify(myDoc[key]));
...    }
...    print("\n");
... }
_id: "507d95d5719dbef170f15bfa"
name: "AC3 Case Green"
type: ["accessory","case"]
color: "green"
price: 12
warranty_years: 0


_id: "ac7"
name: "AC7 Phone"
brand: "ACME"
type: "phone"
price: 320
warranty_years: 1
available: false


_id: "507d95d5719dbef170f15bfd"
name: "AC3 Case Red"
type: ["accessory","case"]
color: "red"
price: 12
warranty_years: 0.25
available: true
for: "ac3"
```

- Query 2:
  - Display only the “name” and “brand” fields for each product.

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

```
epam> db.products.find({}, {"name": 1, "brand": 1, "_id": 0})
[
  { name: 'AC3 Phone', brand: 'ACME' },
  { name: 'Phone Service Basic Plan' },
  { name: 'AC3 Case Green' },
  { name: 'AC7 Phone', brand: 'ACME' },
  { name: 'AC3 Case Red' },
  { name: 'AC3 Case Black' },
  { name: 'AC3 Series Charger' },
  { name: 'Cable TV Basic Service Package' },
  { name: 'Phone Service Family Plan' },
  { name: 'Phone Extended Warranty' },
  { name: 'Phone Service Core Plan' },
  { name: 'AC9 Phone', brand: 'ACME' }
]
epam>
```

- 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?

db.products.find({}, {"_id": 1, "limits": 1}).limit(10)

```
epam> db.products.find({}, {"_id": 1, "limits": 1}).limit(10)
[
  { _id: 'ac3' },
  {
    _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('507d95d5719dbef170f15bfa') },
  { _id: 'ac7' },
  { _id: ObjectId('507d95d5719dbef170f15bfd') },
  { _id: ObjectId('507d95d5719dbef170f15bfc') },
  { _id: ObjectId('507d95d5719dbef170f15bf9') },
  { _id: ObjectId('507d95d5719dbef170f15c01') },
  {
    _id: ObjectId('507d95d5719dbef170f15c00'),
    limits: {
      voice: { units: 'minutes', n: 1200, over_rate: 0.05 },
      data: { n: 'unlimited', over_rate: 0 },
      sms: { n: 'unlimited', over_rate: 0 }
    }
  },
  { _id: ObjectId('507d95d5719dbef170f15bfb') }
]
epam>
```

I will say that it shows if this key exists. In other cases as we can see it does not.

- Query 4:
  - Display the IDs, names and prices of all products of which prices are greater or equal to 200.

db.products.find({"price": {"$gte": 200}}, {"_id": 1, "name": 1, "price": 1})

```
epam> db.products.find({"price": {"$gte": 200}}, {"_id": 1, "name": 1, "price": 1})
[
  { _id: 'ac3', name: 'AC3 Phone', price: 200 },
  { _id: 'ac7', name: 'AC7 Phone', price: 320 },
  { _id: 'ac9', name: 'AC9 Phone', price: 333 }
]
epam> _
```

- 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)

db.products.find({}, {"_id": 1, "name": 1, "price": 1}).sort({"price": -1, "name": 1})

```
epam> db.products.find({}, { _id: 1, name: 1, price: 1}).sort({ price: -1, name: 1})
[
  { _id: 'ac9', name: 'AC9 Phone', price: 333 },
  { _id: 'ac7', name: 'AC7 Phone', price: 320 },
  { _id: 'ac3', name: 'AC3 Phone', price: 200 },
  {
    _id: ObjectId('607d95d5719dbef170f15bf8'),
    name: 'Phone Extended Warranty',
    price: 38
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bf9'),
    name: 'AC3 Series Charger',
    price: 19
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bfc'),
    name: 'AC3 Case Black',
    price: 12.5
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bfa'),
    name: 'AC3 Case Green',
    price: 12
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bfd'),
    name: 'AC3 Case Red',
    price: 12
  },
  {
    _id: ObjectId('607d95d5719dbef170f15c01'),
    name: 'Cable TV Basic Service Package'
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bfe'),
    name: 'Phone Service Basic Plan'
  },
  {
    _id: ObjectId('607d95d5719dbef170f15bff'),
    name: 'Phone Service Core Plan'
  },
  {
    _id: ObjectId('607d95d5719dbef170f15c00'),
    name: 'Phone Service Family Plan'
  }
]
epam> _
```

- Query 6:
  - Write a query that displays how many products we have of type “service”. (Check the field which is named “type”)

db.products.countDocuments({"type": "service"})

```
epam> db.products.countDocuments({"type": "service"})
3
epam> _
```

- Updating records
  - General questions
    - Can we update the “_id” field? Why so?

I tried to change as same value, it performed without errors and but no modifications. Also I tried to change it different value, it doesn’t work.

```
epam> db.products.update(
...     {_id: "ac9"},
...     {$set: {"_id": "ac9"}},
...     {multi: false}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
epam> db.products.update(
...     {_id: "ac9"},
...     {$set: {"_id": "ac10"}},
...     {multi: false}
... )
MongoServerError: Performing an update on the path '_id' would modify the immutable field '_id'
epam> db.products.update(
```

No, you can not change _id value. You will get error like that:
MongoServerError: Performing an update on the path '_id' would modify the immutable field '_id'.

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

It works like set in sql. The $set operator replaces the value of a field with the specified value. We change specific values using this keyword.

We can not do that. We will get this error:
MongoInvalidArgumentError: Update document requires atomic operators

- When should use the "*multi*" keyword?

It updates all data which are meet criteria. And it works as options.

- Please perform a query after each of the following updates to verify you have updates the documents as expected.
- Update 1:
  - Update product with ID "ac3", so that he will now have only the following field values:
  - company: "EPAM"
  - item: "MongoDB"

```
db.products.update(
  {"_id": "ac3"},
  {$set: {"company": "EPAM", "item": "MongoDB"}},
  {multi: false}
)
```

```
epam> db.products.update(
...     {"_id": "ac3"},
...     {$set: {"company": "EPAM", "item": "MongoDB"}},
...     {multi: false}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
epam> db.products.find({ "_id": "ac3" })
[
  {
    _id: 'ac3',
    name: 'AC3 Phone',
    brand: 'ACME',
    type: 'phone',
    price: 200,
    warranty_years: 1,
    available: true,
    company: 'EPAM',
    item: 'MongoDB'
  }
]
epam>
```

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.

```
db.products.update(
  {"name": /ac3/i},
  {$set: {"subtype": "AC3"}},
  {multi: true}
)
```

```
epam> db.products.update(
...     {"name": /ac3/i},
...     {$set: {"subtype": "AC3"}},
...     {multi: true}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
epam>
```

- Deleting records
  - Remove all records of type “service”.

```
epam> db.products.deleteMany({type: "service"})
{ acknowledged: true, deletedCount: 3 }
epam>
```

```
epam> db.products.countDocuments()
9
epam>
```

7. USING INDEXES (5)

- Create an index for the “price” field

db.products.createIndex({"price": 1})

```
epam> db.products.createIndex({"price": 1})
price_1
epam>
```

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

db.products.createIndex({"type": 1, "subtype": 1 })

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

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

db.products.createIndex({"name": "text"})

```
epam> db.products.createIndex({"name": "text"})
name_text
epam>
```

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

Text indexes have an advantage over a index because of its specialized support for full-text search features like relevance rating, word matches, and language-specific features.

8. ARCHITECTURE AND MONITORING (15)

- Consult the guidelines document if required for assistance on the following requirements.
- Run a command which describes the current MongoDB node.

db.isMaster() or db.hello()

```
epam> db.isMaster()
{
  ismaster: true,
  topologyVersion: {
    processId: ObjectId('65b6c3c9f5a58c53dd77e1bf'),
    counter: Long('0')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2024-01-29T00:41:01.554Z'),
  logicalSessionTimeoutMinutes: 30,
  connectionId: 10,
  minWireVersion: 0,
  maxWireVersion: 21,
  readOnly: false,
  ok: 1,
  isWritablePrimary: true
}
epam>
```

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

db.hello().localTime

```
epam> db.hello().localTime
ISODate('2024-01-29T01:29:17.181Z')
epam>
```

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

db.serverStatus()

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

db.currentOp()

- Check – are replication sets currently enabled?

rs.status()

No, replication sets are not.

```
test> rs.status()
MongoServerError: not running with --replSet
test> db.adminCommand( { replSetGetStatus: 1, initialSync: 1 } )
MongoServerError: not running with --replSet
test>
```