

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	2
2.	Connect to the the MongoDB environment.....	2
3.	General Details and practice preparation (5)	3
4.	Import products data into MongoDB (15)	3
5.	Verify the loaded data in MongoDB (20)	3
6.	CRUD operations in MongoDB collections (40)	4
7.	Using indexes (5)	5
8.	Architecture and monitoring (15)	5

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.
- 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**

```
D:\ProjectFiles\epam-mongodb-docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS          NAMES
1bbb6b51e865  mongo                                "docker-entrypoint.s..." 6 minutes ago  Up 7 seconds  27017/tcp      mongo
d35f8bb7fd76  postgres:13                         "docker-entrypoint.s..." 8 days ago    Exited (1)    8 days ago
cblc80cdd9f96  dpape/pgadmin4:latest              "/entrypoint.sh"        8 days ago    Exited (0)    8 days ago    2_docker_sql-pgadmin-1
755ffccf711e  vsc-rdb-alpha-7bca59193fdb3007554aea848f4b41924e16b17fa5c6c107abb332bff998dae "/bin/sh -c 'echo Co..." 9 days ago    Exited (0)    9 days ago    laughing_archimedes
b529bf651d6f  obsidiandynamics/kafdrop           "/kafdrop.sh"          2 weeks ago   Exited (143)  2 weeks ago   kafdrop
252d20cf8804  mysql                                "docker-entrypoint.s..." 3 weeks ago   Exited (0)    2 weeks ago   mysql
12b4f1db420a  ofrir119/kafka:2.4.0               "supervisord -n"       3 weeks ago   Exited (0)    2 weeks ago   kafka
clae9e16c57  bitnami/spark:3.5                  "/opt/bitnami/script..." 7 weeks ago   Exited (137)  7 weeks ago   spark-1
7231de0a2c07  bitnami/spark:3.5                  "/opt/bitnami/script..." 7 weeks ago   Exited (137)  7 weeks ago   spark-worker-2
782151ade179  bitnami/spark:3.5                  "/opt/bitnami/script..." 7 weeks ago   Exited (137)  7 weeks ago   spark-worker-1
72c06633b331  bde2020/hive:2.3.2-postgresql-metastore "entrypoint.sh /opt/..." 2 months ago  Exited (255)  5 weeks ago   hive3-hive-metastore-1
4f57854biaf3  gethue/hue                          "/startup.sh"           2 months ago  Exited (255)  5 weeks ago   hive3-hue-1
f91dd7da7d59  bde2020/hive-metastore-postgresql:2.3.0 "/docker-entrypoint..." 2 months ago  Exited (255)  5 weeks ago   hive3-hive-metastore-post
gresql-1
e32dc8e01691  bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8 "/entrypoint.sh /run..." 2 months ago  Exited (255)  2 months ago   hive3-namenode-1
8f0875df76d8  bde2020/hives:2.3.2-postgresql-metastore "entrypoint.sh /bin/..." 2 months ago  Exited (255)  2 months ago   hive3-hive-server-1
59d8889fe986  bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8 "/entrypoint.sh /run..." 2 months ago  Exited (255)  2 months ago   hive3-datanode-1

D:\ProjectFiles\epam-mongodb-docker exec -it mongo /bin/bash
root@1bbb6b51e865:/#
```

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
```

```
D:\ProjectFiles\epam-mongodb>docker exec -it mongo /bin/bash
root@1bbb6b51e865:/# mongoimport --host localhost --port 27017 -d epam -c products ./data/products.json
2024-01-28T21:29:10.970+0000    connected to: mongodb://localhost:27017/
2024-01-28T21:29:11.001+0000    11 document(s) imported successfully. 0 document(s) failed to import.
root@1bbb6b51e865:/#
```

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()

```
test> db.help()
Database Class:
getMongo           Returns the current database connection
getname            Returns the name of the db
getCollectionsNames 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, type, etc.
runCommand         Runs an arbitrary command against the current database.
adminCommand       Runs a specified admin/diagnostic pipeline which does not require authentication.
aggregate          Returns another database without modifying the db variable in the current database.
getCollection       Returns a collection or a view object that is functionally equivalent to the current database.
dropDatabase       Removes the current database, deleting the associated data files.
createUser         Creates a new user for the database on which the method is run.
updateUser         Updates the user's profile on the database on which you run the method.
logout             Ends the current authentication session. This function has no effect if you are not authenticated.
dropUser           Removes the user from the current database.
removeUser         Removes all users from the current database.
grantRolesToUser   Grants additional roles to a user.
revokeRolesFromUser Removes a one or more roles from a user on the current database.
getUsers           Returns user information for a specified user. Run this method on the current database.
createCollection   Create new collection
createEncryptedCollection Create new collection with a list of encrypted fields each with a key and value.
createView         Create new view.
createRole         Creates a new role.
updateRole         Updates the role's profile on the database on which you run the method.
dropRole          Removes the role from the current database.
dropAllRoles       Removes all roles from the current database.
grantRolesToRole   Grants additional privileges to a role.
revokeRolesFromRole Removes a one or more privileges from a role on the current database.
revokePrivilegesFromRole Removes a one or more privileges from a role on the current database.
getRoles           Returns role information for a specified role. Run this method on the current database.
currentOp         Returns information for all the roles in the database.
killOp            aggregation using $currentOp operator. Returns a document.
shutdownServer    Calls the shutdown command. Terminates an operation as specified by the $killOp command.
fsyncLock          Calls the fsync command. Forces the mongod to flush all pending writes to the disk.
serverStatus       Returns the db version, uses the buildInfo command.
useMaster          returns the db serverid, uses the buildInfo command
hello              Calls the hello command
serverBuildInfo    returns the db serverid, uses the buildInfo command
serverStatus       returns the server status, uses the serverStatus command
stats              db stats, uses the dbstats command
hostInfo           Calls the hostInfo command
serverCmdFlags     returns the db servercmdflags, uses the getCmdFlags 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
cloneDatabase       deprecated, non-functional
cloneCollection     deprecated, non-functional
copyDatabase         deprecated, non-functional
commandHelp        returns the db commandHelp, uses the passed in command with help:
listCommands        returns the db listCommands command
getLastError        Calls the getLastError command
getLastErrorObj     Calls the getLastErrorObj command
printShardingStatus Calls sh.status(verbose)
printSecondaryReplicationInfo Prints secondary replication information
getReplicationInfo Returns replication information
getReplicationInfoObj Returns replication information
printSlaveReplicationInfo Prints slave replication information
printSecondaryReplicationInfo Prints secondary replication information
setSecondaryOplog DEPRECATED, use db.printSecondaryReplicationInfo
setSecondaryOplog DEPRECATED, use db.printSecondaryReplicationInfo
watch              opens a change stream cursor on the database.
checkMetadataConsistency Runs a SQL query against Atlas Data Lake. Note: this is an experimental feature.
Returns a cursor with information about metadata inconsistencies
```

Collection

db.collection.help()

```
test> db.collection.help()
Collection Class:
aggregate          Calculates aggregate values for the data in a collection or a view.
bulkWrite          Performs multiple write operations with controls for order of execution.
count              Returns the count of documents that would match a find() query for the collection.
countDocuments     Returns the count of documents that match the query for a collection or view.
deleteMany         Removes all documents that match the filter from a collection.
deleteOne          Removes a single document from a collection.
distinct           Finds the distinct values for a specified field across a single collection.
estimatedDocumentCount Returns the count of all documents in a collection or view.
find               Selects documents in a collection or view.
findAndModify      Modifies and returns a single document.
findOne            Selects documents in a collection or view.
renameCollection   Renames a collection.
findOneAndDelete   Deletes a single document based on the filter and sort criteria, returning the deleted document.
findOneAndUpdate   Updates a single document based on the filter and sort criteria.
insert             Inserts a document or documents into a collection.
insertMany         Inserts multiple documents into a collection.
insertOne          Inserts a document into a collection.
isCapped           Checks if a collection is capped.
lastErrorObject     Removes documents from a collection.
replaceOne         Replaces a single document within the collection based on the filter.
update            Modifies an existing document or documents in a collection.
updateMany         Updates all documents that match the specified filter for a collection.
updateOne          Updates a single document within the collection based on the filter.
compact            Compacts structured encryption data.
convertToCapped    Calls {convertToCapped: 'coll', size: maxbytes} command.
createIndex        Creates one index on a collection.
createIndexes      Creates one or more indexes on a collection.
ensureIndex        Returns an array that holds a list of documents that identify and describe the collection.
getIndexKeys       Returns an array of key patterns for indexes defined on collection.
getIndexes         Returns an array of key patterns for indexes defined on collection.
getIndexesSpecs    Returns an array of key patterns for indexes defined on collection.
getIndices         Returns an array of key patterns for indexes defined on collection.
getIndexesKeys     Returns an array of key patterns for indexes defined on collection.
dropIndexes        Drops or removes the specified index from a collection.
dropIndex          Drops or removes the specified index from a collection.
totalIndexSize     Reports the total size used by the indexes on a collection.
reIndex            Rebuilds all existing indexes on a collection.
reIndex           get current database.
getDB              Returns the MongoDB object.
getMongo           This method provides a wrapper around the size output of the collStats (i.e. the total amount of storage allocated to this collection for document storage).
dataSize           The total size in bytes of the data in the collection plus the size of every document.
storageSize        Removes a collection or view from the database.
totalSize          Returns collection infos if the collection exists or null otherwise.
drop              Returns the name of the collection prefixed with the database name.
exists             Returns the name of the collection.
getFullName        Returns a command with the given name where the first param is the collection name.
getName            Returns information on the query plan.
runCommand         Returns statistics about the collection.
explain           returns the statsCmd aggregation for the collection. Takes an options object.
stats             Initializes an ordered bulk command. Returns an instance of Bulk.
latencyStats       Initializes an unordered bulk command. Returns an instance of Bulk.
initializeOrderedBulkop Returns an interface to access the query plan cache for a collection. The first parameter is the collection name.
initializeUnorderedBulkop Returns an interface to access the query plan cache for a collection. The first parameter is the collection name.
mapReduce          Calls the mapReduce command.
validate           Calls the validate command. Default full value is false.
getShardVersion    Calls the getShardVersion command.
getShardDistribution Prints the data distribution statistics for a sharded collection.
watch             Opens a change stream cursor on the collection.
hideIndex          Hides an existing index from the query planner.
unhideIndex        Unhides an existing index from the query planner.
analyzeShardKey    Returns metrics for evaluating a shard key. That is, 'key' can be a candidate for a shard key.
configureQueryAnalyzer Configures the query analyzer.
checkMetadataConsistency Returns a cursor with information about metadata inconsistencies.
getSearchIndexes   Returns an array that holds a list of documents that identify and describe the collection.
createSearchIndex  Creates a search index on a collection.
createSearchIndexes Creates one or more search indexes on a collection.
dropSearchIndex    Drops or removes the specified search index from a collection.
updateSearchIndex  Updates the specified search index.
```

- Find options in collections

db.collection.find().help()

```
test> db.collection.find().help()
collection cursor:
  addOptions      Adds OP_QUERY wire protocol flags, such as the tailable flag.
  allowDiskUse    Sets the 'allowDiskUse' option. If no argument is passed, the
  allowPartialResults Sets the 'allowPartialResults' option to true.
  collation        Specifies the collation for the cursor returned by the db.collection.find().
  comment          Adds a comment field to the query.
  count            Counts the number of documents referenced by a cursor.
  hasNext          cursor.hasNext() returns true if the cursor returned by the
  returned.         returned. To check if a document is in the cursor's batch without waiting, use tryNext instead.
  limit            Call this method on a query to override MongoDB's default in
  limit            use the limit() method on a cursor to specify the maximum number
  max              Specifies the exclusive upper bound for a specific index in
  maxWaitTimeMS   Set a maxWaitTimeMS on a tailing cursor query to allow to c
  min              Specifies the inclusive lower bound for a specific index in
  next             The next document in the cursor returned by the db.collection
  n               without waiting, use tryNext instead.
  noCursorTimeout Instructs the server to avoid closing a cursor automatically.
  options          Sets the options for the cursor.
  readPref         Append readPref() to a cursor to control how the client route
  readPref         modifies the cursor to return index keys rather than the doc
  size             A count of the number of documents that match the db.collection
  tailable         Marks the cursor as tailable.
  maxScan          deprecated, non-functional
  showRecordid     Modifies the output of a query by adding a field recordid to
  readConcern      Specify a read concern for the db.collection.find() method.
```

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

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

db.getName()

```
test> db.getName()
test
test> _
```

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

use epam

```
test> use epam
switched to db epam
epam>
```

db.getCollectionNames()

```
epam> db.getCollectionNames()
[ 'products' ]
epam> _
```

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

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

```
epam> db.products.find()
{
  "_id": "a53",
  "name": "A53 phone",
  "brand": "ACME",
  "type": "phone",
  "price": 200,
  "warranty_years": 1,
  "available": true
},
{
  "_id": ObjectId("107d9d5d719ddef170f18f8e"),
  "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("107d9d5d719ddef170f18f8e"),
  "name": "A53 case green",
  "type": [ "Accessory", "case" ],
  "color": "green",
  "price": 15,
  "warranty_years": 0
},
{
}
```

```
epam> db.products.countDocuments()
11
epam> _
```


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('507d95d5719dbef170f15bfa'),
    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('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: 'objId(507d95d5719dbef170f15bfb)', name: 'Phone Extended Warranty', price: 18 },
  { _id: 'objId(507d95d5719dbef170f15bf9)', name: 'AC3 Series Charger', price: 19 },
  { _id: 'objId(507d95d5719dbef170f15bfc)', name: 'AC3 Case Black', price: 12.5 },
  { _id: 'objId(507d95d5719dbef170f15bfa)', name: 'AC3 Case Green', price: 12 },
  { _id: 'objId(507d95d5719dbef170f15bfd)', name: 'AC3 Case Red', price: 12 },
  { _id: 'objId(507d95d5719dbef170f15c01)', name: 'Phone Service Basic Plan' },
  { _id: 'objId(507d95d5719dbef170f15bfe)', name: 'Phone Service Core Plan' },
  { _id: 'objId(507d95d5719dbef170f15bff)', name: 'Phone Service Family Plan' },
  { _id: 'objId(507d95d5719dbef170f15c00)', 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'
```

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()

```
test> db.serverStatus()
{
  host: '192b6b51e865',
  version: '7.0.8',
  process: 'mongod',
  pid: Long('1'),
  uptime: 1541,
  uptimeMillis: Long('15417357'),
  uptimeEstimate: Long('15417'),
  localTime: ISODate('2024-01-29T01:31:46.457Z'),
  asserts: {
    regular: 0,
    warning: 0,
    msg: 0,
    user: 0,
    tripwire: 0,
    rollovers: 0
  },
  batchedDeletes: {
    batches: 4,
    docs: 7,
    stagedSizeBytes: 2230,
    timeInBatchMillis: 2,
    refetchesDueToViel: 0
  },
  catalogStats: {
    collections: 1,
    capped: 0,
    clustered: 0,
    timeseries: 0,
    views: 0,
    internalCollections: 3,
    internalViews: 0,
    cories: 0,
    queryableEncryption: 0
  },
  collectionCatalog: { numScansDueToMissingMapping: Long('0') },
  connections: {
    current: 1,
    available: 838855,
    totalCreated: 15,
    rejected: 0,
    active: 2,
    threaded: 1,
    exhaustedMaster: 0,
    exhaustedHello: 0,
    awaitingTopologyChanges: 1
  },
  electionMetrics: {
    stepdownCmd: { 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') },
    freeElect: { called: Long('0'), successful: Long('0') },
    numStepDownsCausedByHigherTerm: Long('0'),
    numCatchups: Long('0'),
    numCatchupsSucceeded: Long('0'),
    numCatchupsAlreadyCaughtUp: Long('0'),
    numCatchupsSkipped: Long('0'),
    numCatchupsTimedOut: Long('0'),
    numCatchupsFailedWithError: Long('0'),
    numCatchupsFailedWithHigherTerm: Long('0'),
    numCatchupsFailedWithRep1SetAbortPrimaryCatchupCmd: Long('0'),
    averageCatchups: 0
  },
  extraInfo: {

```

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

db.currentOp()

```
test> db.currentOp()
{
  inprog: [
    {
      type: 'op',
      host: '1b9b6b51e865:27017',
      desc: 'conn14',
      connectionId: 14,
      client: '127.0.0.1:56362',
      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: '5.15.133.1-microsoft-standard-WSL2',
          type: 'linux'
        }
      },
      active: true,
      currentOpTime: '2024-01-29T01:33:59.420+00:00',
      threaded: true,
      opid: 194577,
      lsid: {
        id: UUID('7c5ae0aa-28a0-4fdd-ae98-762493495e3f'),
        uid: Binary.createFromBase64('47DEqpJ8HBSa+/Tlmm+SJCeuqERkmsNMpJWZG3hsuFU=', 0)
      },
      secs_running: Long('0'),
      microsecs_running: Long('654'),
      op: 'command',
      ns: 'admin.$cmd.aggregate',
      redacted: false,
      command: {
        aggregate: 1,
        pipeline: [
          {
            '$currentOp': {
              allUsers: true,
              idleConnections: false,
              truncateOps: false
            }
          },
          { '$match': {} }
        ]
      },
      cursor: {},
      lsid: { id: UUID('7c5ae0aa-28a0-4fdd-ae98-762493495e3f') },
      $db: 'admin'
    },
    {
      type: 'op',
      host: '1b9b6b51e865:27017',
      desc: 'JournalFlusher',
      active: true,
      currentOpTime: '2024-01-29T01:33:59.421+00:00',
      opid: 194578,
      op: 'none',
      ns: '',
      redacted: false,
      command: {}
    }
  ]
}
```

- 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> =
```