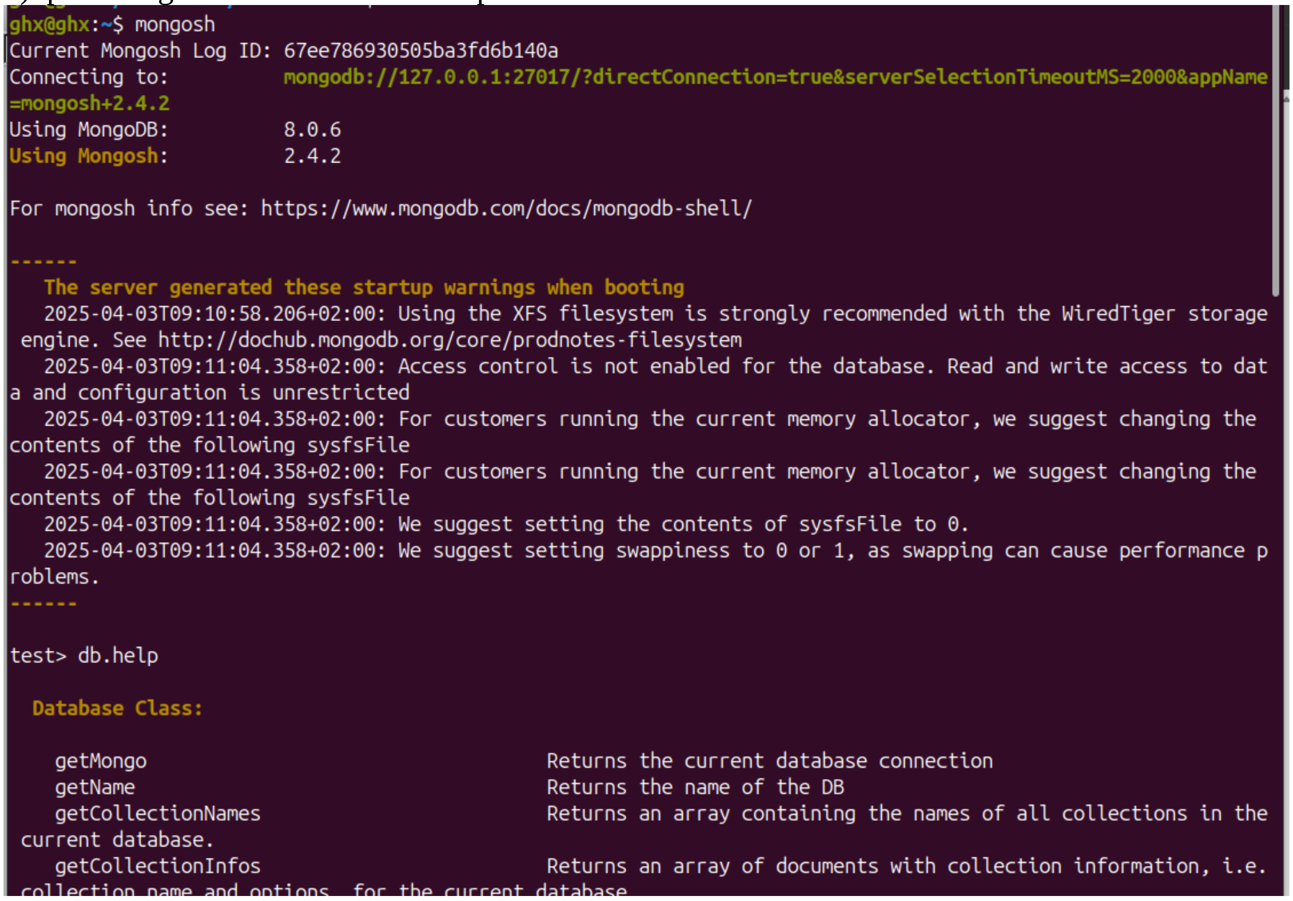
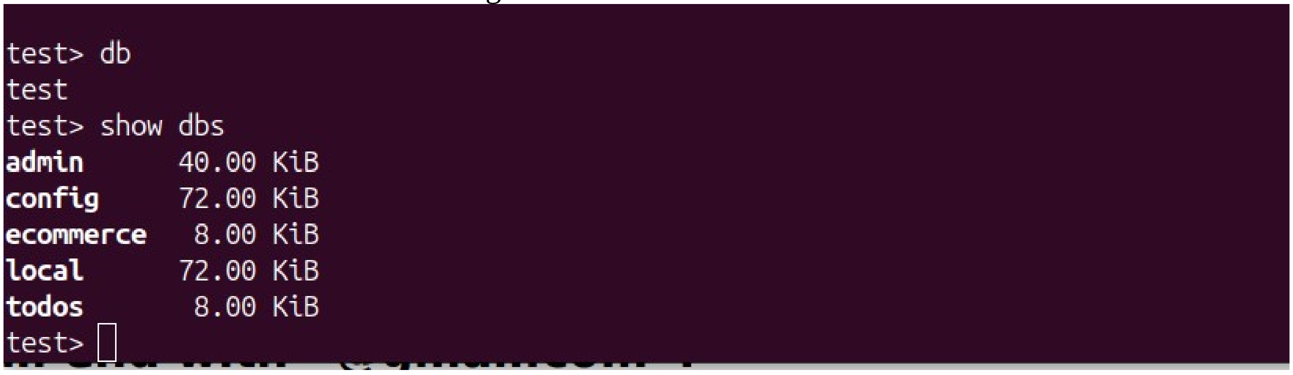
1)Install Mongo Database.
Done
2)open mongo shell and view the help.



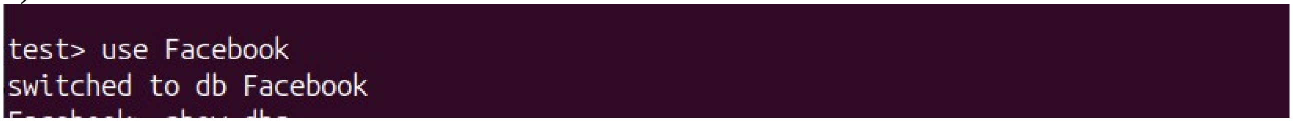
```
ghx@ghx:~$ mongosh
Current Mongosh Log ID: 67ee786930505ba3fd6b140a
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName
=mongosh+2.4.2
Using MongoDB:          8.0.6
Using Mongosh:          2.4.2

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

------
   The server generated these startup warnings when booting
   2025-04-03T09:10:58.206+02:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage
 engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2025-04-03T09:11:04.358+02:00: Access control is not enabled for the database. Read and write access to dat
a and configuration is unrestricted
   2025-04-03T09:11:04.358+02:00: For customers running the current memory allocator, we suggest changing the
contents of the following sysfsFile
   2025-04-03T09:11:04.358+02:00: For customers running the current memory allocator, we suggest changing the
contents of the following sysfsFile
   2025-04-03T09:11:04.358+02:00: We suggest setting the contents of sysfsFile to 0.
   2025-04-03T09:11:04.358+02:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance p
roblems.
------

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
```

3)identify your current working database and show list of available databases.
Identify your current working database : using db
show list of available databases : using show dbs



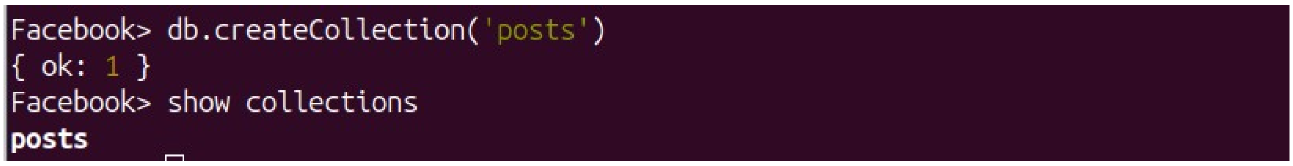
```
test> db
test
test> show dbs
admin       40.00 KiB
config      72.00 KiB
ecommerce    8.00 KiB
local       72.00 KiB
todos        8.00 KiB
test>
```

4) create a new database called “Facebook” and use it.

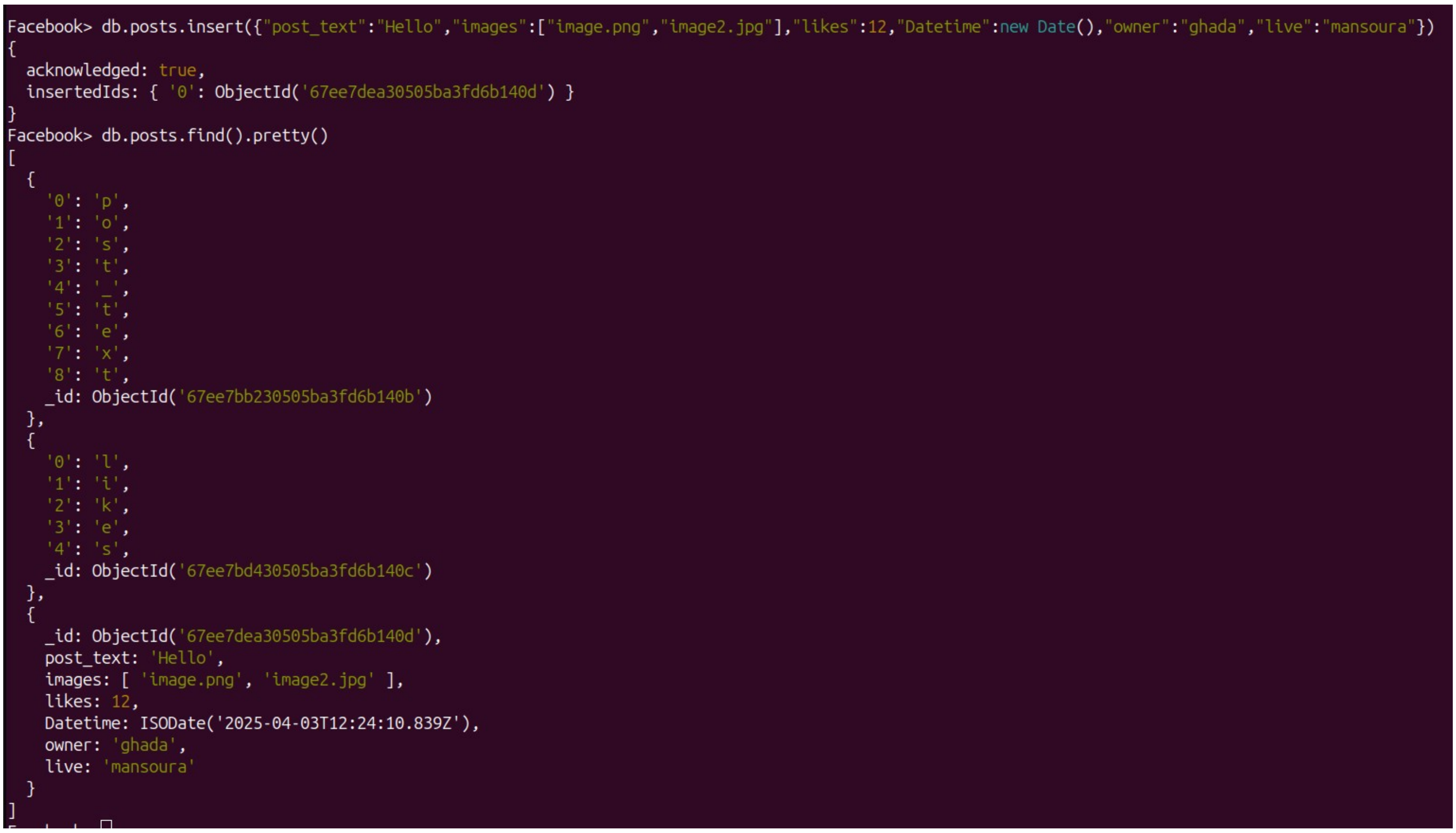


```
test> use Facebook
switched to db Facebook
```

5)Create Collection with name “posts” which has facebook post properties
[“post_text”,”images”,”likes”,”comments”, ”Datetime”,”owner”,”live”]

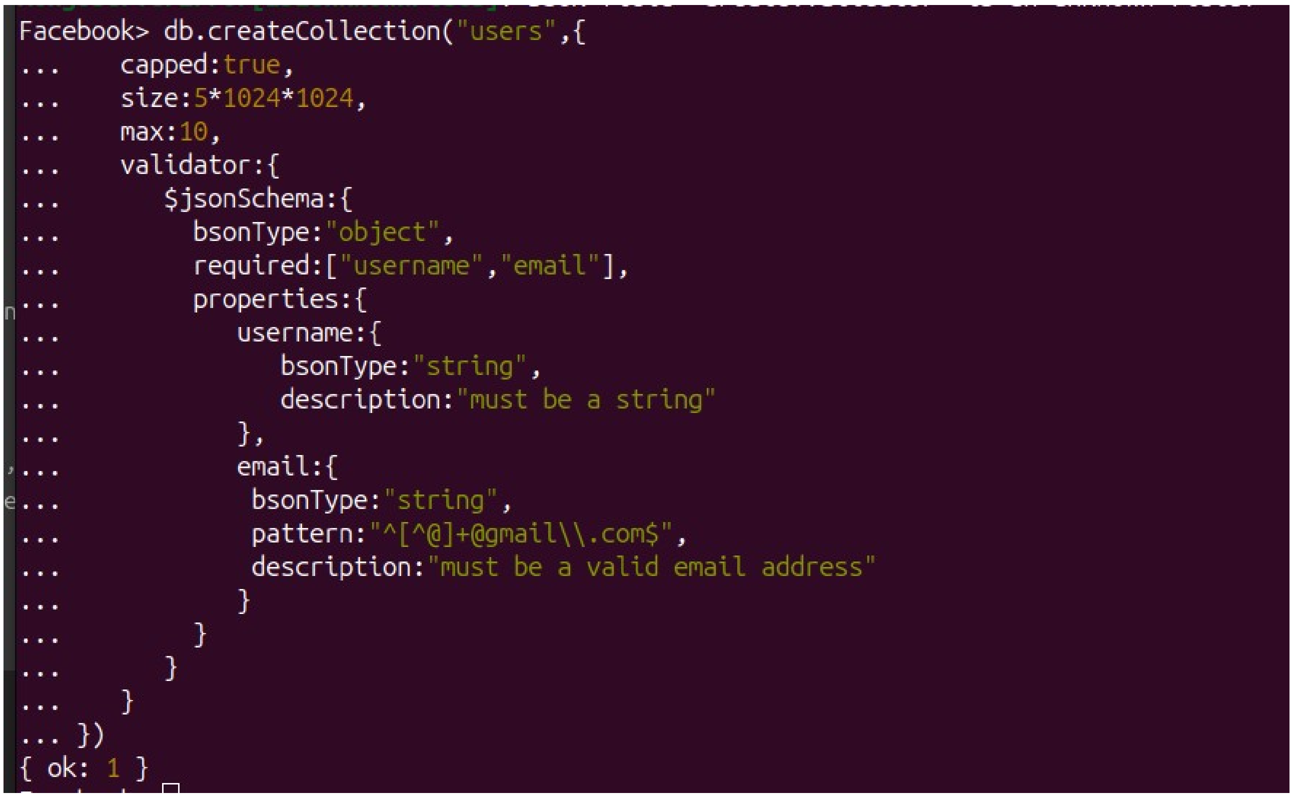


```
Facebook> db.createCollection('posts')
{ ok: 1 }
Facebook> show collections
posts
```



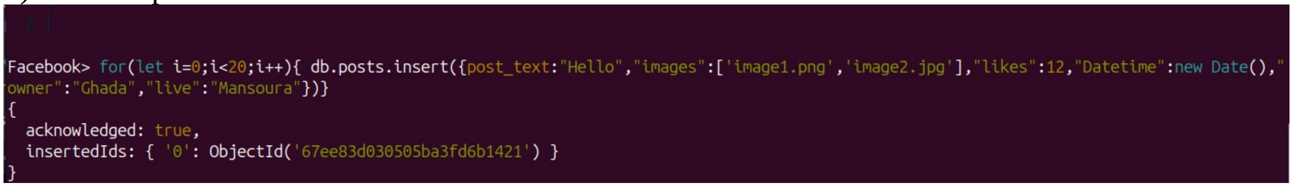
```
Facebook> db.posts.insert({"post_text":"Hello","images":["image.png","image2.jpg"],"likes":12,"Datetime":new Date(),"owner":"ghada","live":"mansoura"})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('67ee7dea30505ba3fd6b140d') }
}
Facebook> db.posts.find().pretty()
[
  {
    '0': 'p',
    '1': 'o',
    '2': 's',
    '3': 't',
    '4': '_',
    '5': 't',
    '6': 'e',
    '7': 'x',
    '8': 't',
    _id: ObjectId('67ee7bb230505ba3fd6b140b')
  },
  {
    '0': 'l',
    '1': 'i',
    '2': 'k',
    '3': 'e',
    '4': 's',
    _id: ObjectId('67ee7bd430505ba3fd6b140c')
  },
  {
    _id: ObjectId('67ee7dea30505ba3fd6b140d'),
    post_text: 'Hello',
    images: [ 'image.png', 'image2.jpg' ],
    likes: 12,
    Datetime: ISODate('2025-04-03T12:24:10.839Z'),
    owner: 'ghada',
    live: 'mansoura'
  }
]
```

6)Create Capped Collection with name users with Size 5 MB ,
10 users Maximum and must has username field “String”
and email end with “@gmail.com”.



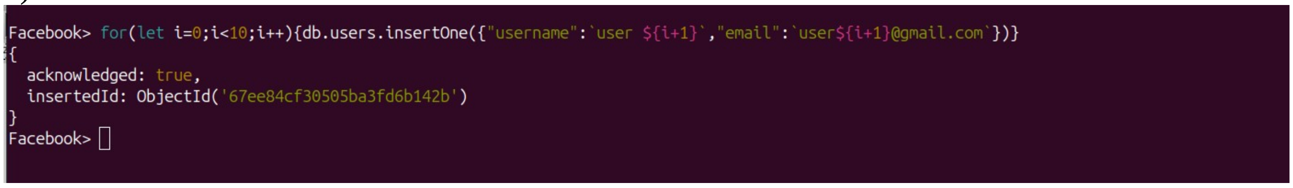
```
Facebook> db.createCollection("users",{
...     capped:true,
...     size:5*1024*1024,
...     max:10,
...     validator:{
...        $jsonSchema:{
...          bsonType:"object",
...          required:["username","email"],
...          properties:{
...             username:{
...                bsonType:"string",
...                description:"must be a string"
...             },
...             email:{
...              bsonType:"string",
...              pattern:"^[^@]+@gmail\\.com$",
...              description:"must be a valid email address"
...             }
...          }
...        }
...     }
... })
{ ok: 1 }
```

7)Insert 20 post “ordered Insert”.



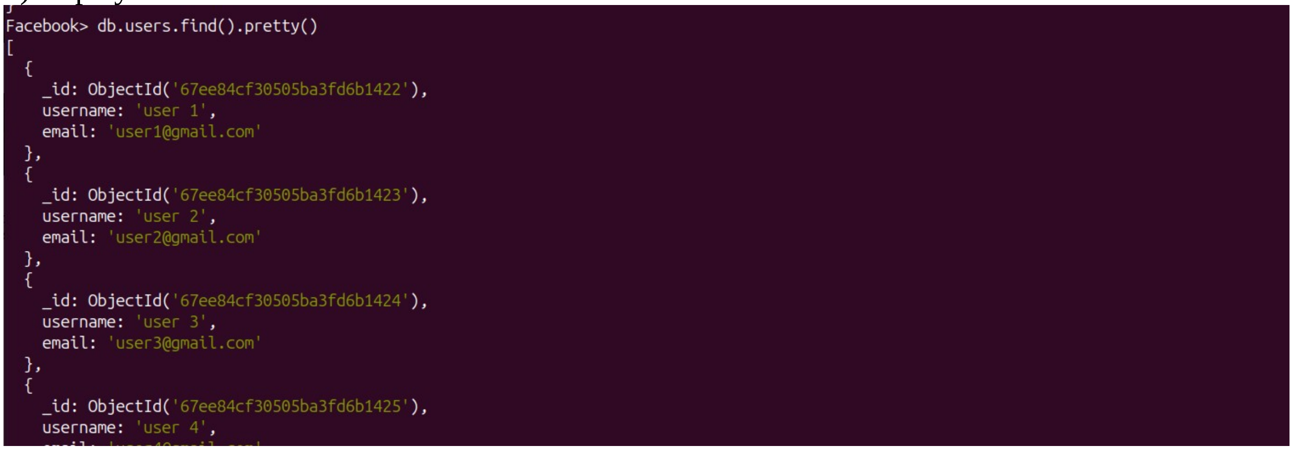
```
Facebook> for(let i=0;i<20;i++){ db.posts.insert({post_text:"Hello","images":['image1.png','image2.jpg'],"likes":12,"Datetime":new Date(),"
owner":"Ghada","live":"Mansoura"})}
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('67ee83d030505ba3fd6b1421') }
}
```

8)Insert 10 users.



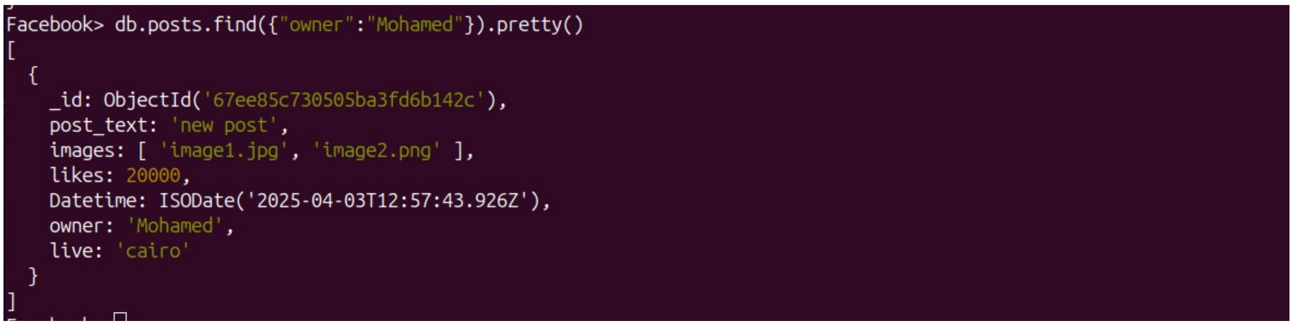
```
Facebook> for(let i=0;i<10;i++){db.users.insertOne({"username":`user ${i+1}`,"email":`user${i+1}@gmail.com`})}
{
  acknowledged: true,
  insertedId: ObjectId('67ee84cf30505ba3fd6b142b')
}
Facebook> 
```

9)Display all users.



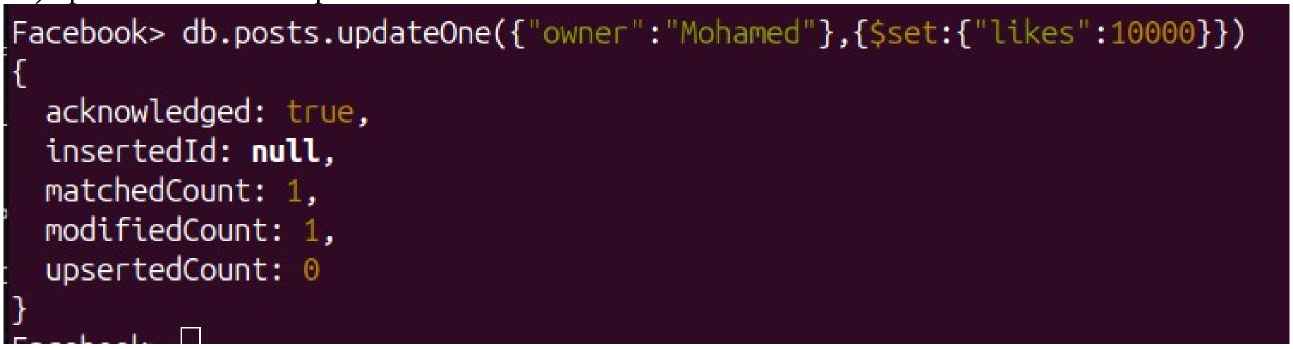
```
Facebook> db.users.find().pretty()
[
  {
    _id: ObjectId('67ee84cf30505ba3fd6b1422'),
    username: 'user 1',
    email: 'user1@gmail.com'
  },
  {
    _id: ObjectId('67ee84cf30505ba3fd6b1423'),
    username: 'user 2',
    email: 'user2@gmail.com'
  },
  {
    _id: ObjectId('67ee84cf30505ba3fd6b1424'),
    username: 'user 3',
    email: 'user3@gmail.com'
  },
  {
    _id: ObjectId('67ee84cf30505ba3fd6b1425'),
    username: 'user 4',
```

10)Display user “Mohamed” posts



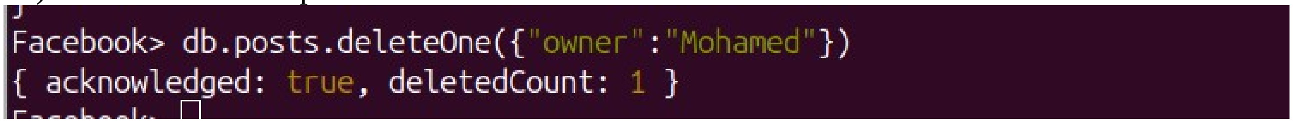
```
Facebook> db.posts.find({"owner":"Mohamed"}).pretty()
[
  {
    _id: ObjectId('67ee85c730505ba3fd6b142c'),
    post_text: 'new post',
    images: [ 'image1.jpg', 'image2.png' ],
    likes: 20000,
    Datetime: ISODate('2025-04-03T12:57:43.926Z'),
    owner: 'Mohamed',
    live: 'cairo'
  }
]
```

11)Update Mohamed ‘s posts set likes 10000



```
Facebook> db.posts.updateOne({"owner":"Mohamed"},{$set:{"likes":10000}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

12)delete Mohamed ‘s posts



```
Facebook> db.posts.deleteOne({"owner":"Mohamed"})
{ acknowledged: true, deletedCount: 1 }
```

13)what is sharding ?

- Sharding helps to scale horizontally by breaking up data into smaller chunks and distributing them across various servers or nodes. Each chunk contains a subset of the data

14)What is replication?

Replication refers to the process of creating multiple copies of data to ensure high availability and data redundancy

15)what is failover mechanism ?

A failover mechanism is a process that automatically switches to a backup system or node when the primary system or node fails

16)what is embedded documents ?

embedded documents refer to subdocuments that are stored inside a parent document.

17)What ACID?

ACID is a set of properties that guarantee database transactions , it stand for:

- Atomicity: Ensures that a transaction is either fully completed or not executed at all

- Consistency: Guarantees that a transaction brings the database from one valid state to another valid state, maintaining database integrity.

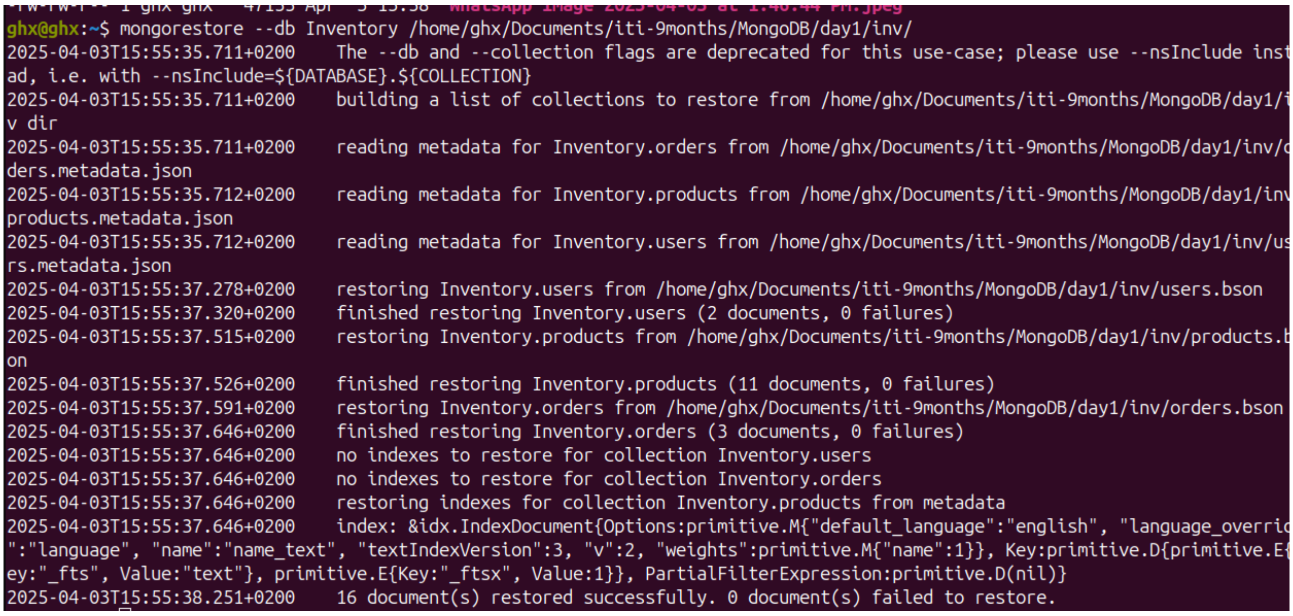
- Isolation: Ensures that concurrent transactions do not interfere with each other, and that transactions appear to be executed in isolation.

- Durability: Ensures that once a transaction is committed, it remains in the system even in the event of a failure.

18) User Management Methods?

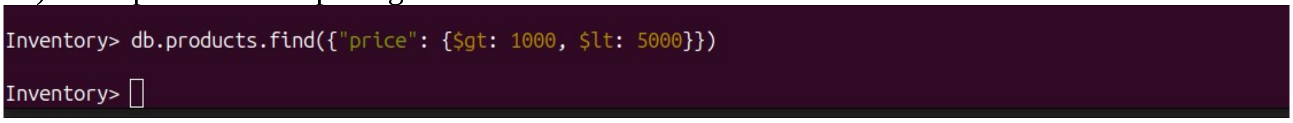
User management involves creating, managing, and assigning roles to users to control access to databases and collections.

19)Import Inventory Database using this command in terminal
mongorestore --db Inventory path_to_Inventory_folder.



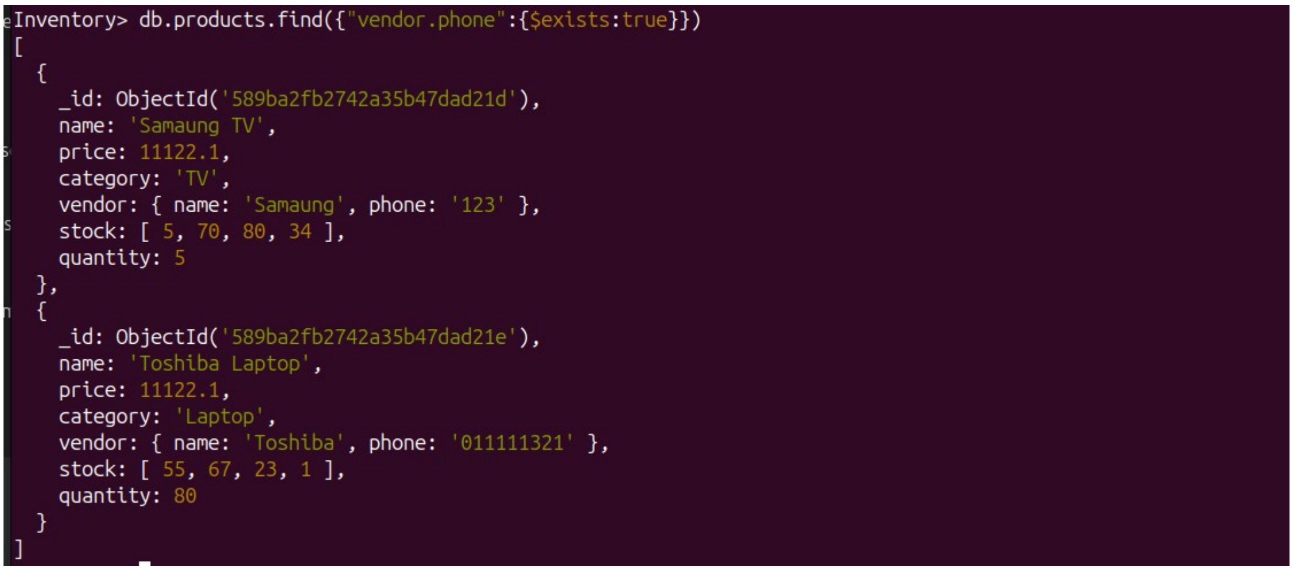
```
ghx@ghx:~$ mongorestore --db Inventory /home/ghx/Documents/iti-9months/MongoDB/day1/inv/
2025-04-03T15:55:35.711+0200    The --db and --collection flags are deprecated for this use-case; please use --nsInclude inst
ad, i.e. with --nsInclude=${DATABASE}.${COLLECTION}
2025-04-03T15:55:35.711+0200    building a list of collections to restore from /home/ghx/Documents/iti-9months/MongoDB/day1/i
v dir
2025-04-03T15:55:35.711+0200    reading metadata for Inventory.orders from /home/ghx/Documents/iti-9months/MongoDB/day1/inv/o
ders.metadata.json
2025-04-03T15:55:35.712+0200    reading metadata for Inventory.products from /home/ghx/Documents/iti-9months/MongoDB/day1/inv
products.metadata.json
2025-04-03T15:55:35.712+0200    reading metadata for Inventory.users from /home/ghx/Documents/iti-9months/MongoDB/day1/inv/us
rs.metadata.json
2025-04-03T15:55:37.278+0200    restoring Inventory.users from /home/ghx/Documents/iti-9months/MongoDB/day1/inv/users.bson
2025-04-03T15:55:37.320+0200    finished restoring Inventory.users (2 documents, 0 failures)
2025-04-03T15:55:37.515+0200    restoring Inventory.products from /home/ghx/Documents/iti-9months/MongoDB/day1/inv/products.b
on
2025-04-03T15:55:37.526+0200    finished restoring Inventory.products (11 documents, 0 failures)
2025-04-03T15:55:37.591+0200    restoring Inventory.orders from /home/ghx/Documents/iti-9months/MongoDB/day1/inv/orders.bson
2025-04-03T15:55:37.646+0200    finished restoring Inventory.orders (3 documents, 0 failures)
2025-04-03T15:55:37.646+0200    no indexes to restore for collection Inventory.users
2025-04-03T15:55:37.646+0200    no indexes to restore for collection Inventory.orders
2025-04-03T15:55:37.646+0200    restoring indexes for collection Inventory.products from metadata
2025-04-03T15:55:37.646+0200    index: &idx.IndexDocument{Options:primitive.M{"default_language":"english", "language_overri
":"language", "name":"name_text", "textIndexVersion":3, "v":2, "weights":primitive.M{"name":1}}, Key:primitive.D{primitive.E{
ey:"_fts", Value:"text"}, primitive.E{Key:"_ftsx", Value:1}}, PartialFilterExpression:primitive.D(nil)}
2025-04-03T15:55:38.251+0200    16 document(s) restored successfully. 0 document(s) failed to restore.
```

20)Select products with price greater than 1000 and less than 5000.

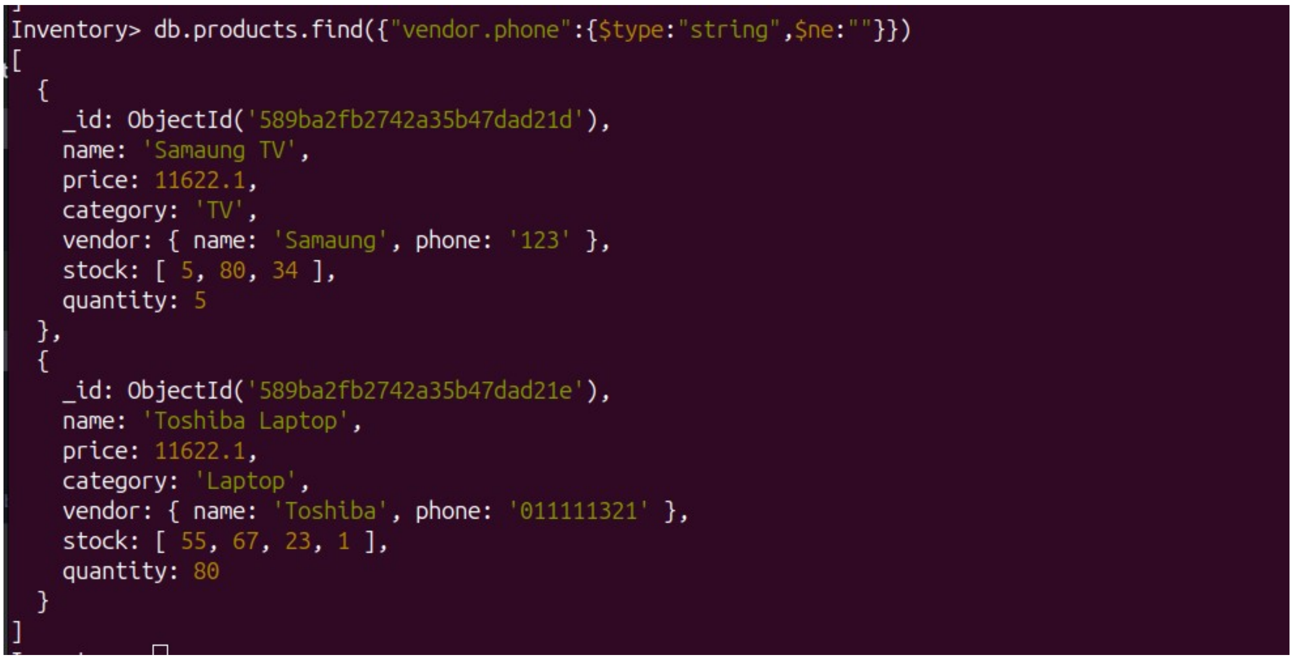


```
Inventory> db.products.find({"price": {$gt: 1000, $lt: 5000}})

Inventory> 
```

21. Display products which have phone number for vedor “using2 ways”

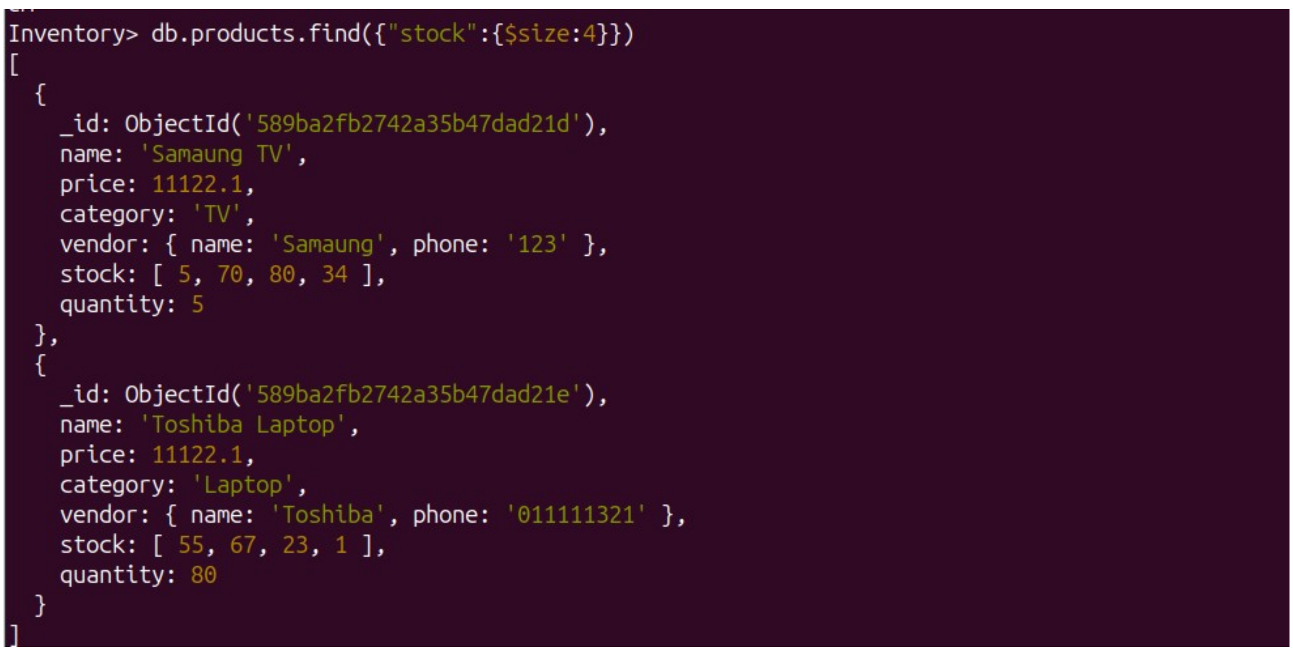


```
Inventory> db.products.find({"vendor.phone":{$exists:true}})
[
  {
    _id: ObjectId('589ba2fb2742a35b47dad21d'),
    name: 'Samaung TV',
    price: 11122.1,
    category: 'TV',
    vendor: { name: 'Samaung', phone: '123' },
    stock: [ 5, 70, 80, 34 ],
    quantity: 5
  },
  {
    _id: ObjectId('589ba2fb2742a35b47dad21e'),
    name: 'Toshiba Laptop',
    price: 11122.1,
    category: 'Laptop',
    vendor: { name: 'Toshiba', phone: '011111321' },
    stock: [ 55, 67, 23, 1 ],
    quantity: 80
  }
]
```



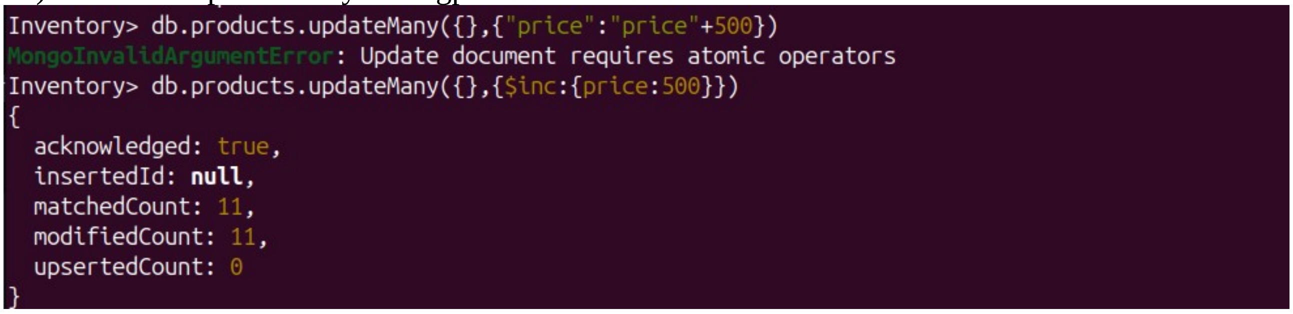
```
Inventory> db.products.find({"vendor.phone":{$type:"string",$ne:""}})
[
  {
    _id: ObjectId('589ba2fb2742a35b47dad21d'),
    name: 'Samaung TV',
    price: 11622.1,
    category: 'TV',
    vendor: { name: 'Samaung', phone: '123' },
    stock: [ 5, 80, 34 ],
    quantity: 5
  },
  {
    _id: ObjectId('589ba2fb2742a35b47dad21e'),
    name: 'Toshiba Laptop',
    price: 11622.1,
    category: 'Laptop',
    vendor: { name: 'Toshiba', phone: '011111321' },
    stock: [ 55, 67, 23, 1 ],
    quantity: 80
  }
]
```

22) Display products which available in 4 stocks at the same time



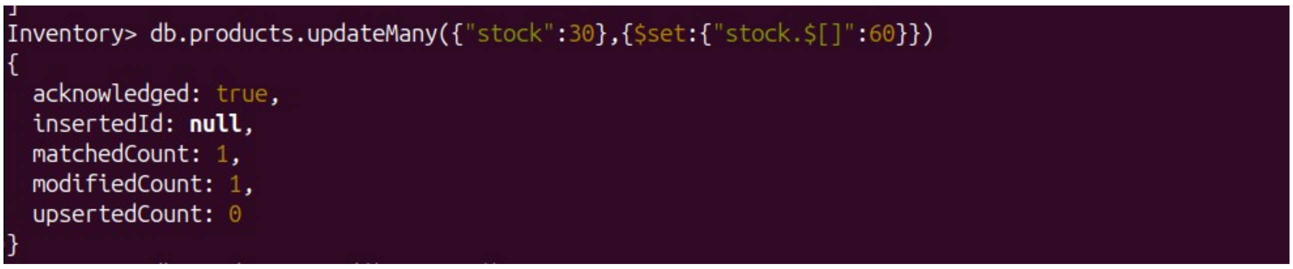
```
Inventory> db.products.find({"stock":{$size:4}})
[
  {
    _id: ObjectId('589ba2fb2742a35b47dad21d'),
    name: 'Samaung TV',
    price: 11122.1,
    category: 'TV',
    vendor: { name: 'Samaung', phone: '123' },
    stock: [ 5, 70, 80, 34 ],
    quantity: 5
  },
  {
    _id: ObjectId('589ba2fb2742a35b47dad21e'),
    name: 'Toshiba Laptop',
    price: 11122.1,
    category: 'Laptop',
    vendor: { name: 'Toshiba', phone: '011111321' },
    stock: [ 55, 67, 23, 1 ],
    quantity: 80
  }
]
```

23) increase all products by 500 egp



```
Inventory> db.products.updateMany({},{"price":"price"+500})
MongoInvalidArgumentError: Update document requires atomic operators
Inventory> db.products.updateMany({},{$inc:{price:500}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 11,
  modifiedCount: 11,
  upsertedCount: 0
}
```

24) replace stock #30 with #60 in all products.



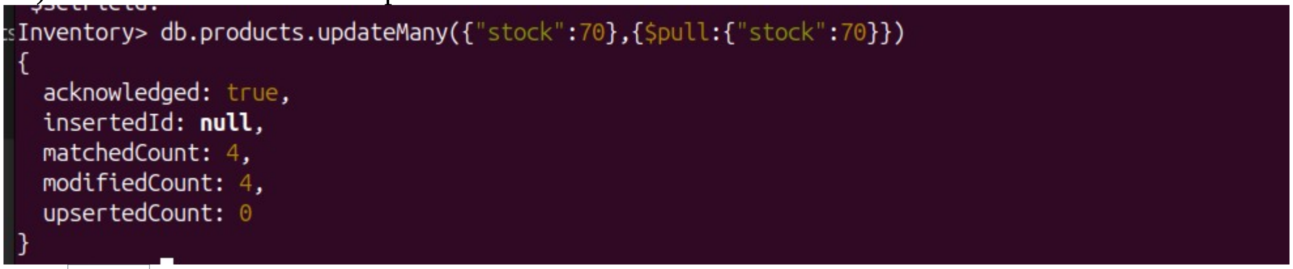
```
Inventory> db.products.updateMany({"stock":30},{$set:{"stock.$[]":60}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

The $[] is called the all-positional operator.

It acts as a placeholder to update all elements in the stock array that match the query condition (30 in this case).

It will replace every occurrence of 30 with 60 in the array.

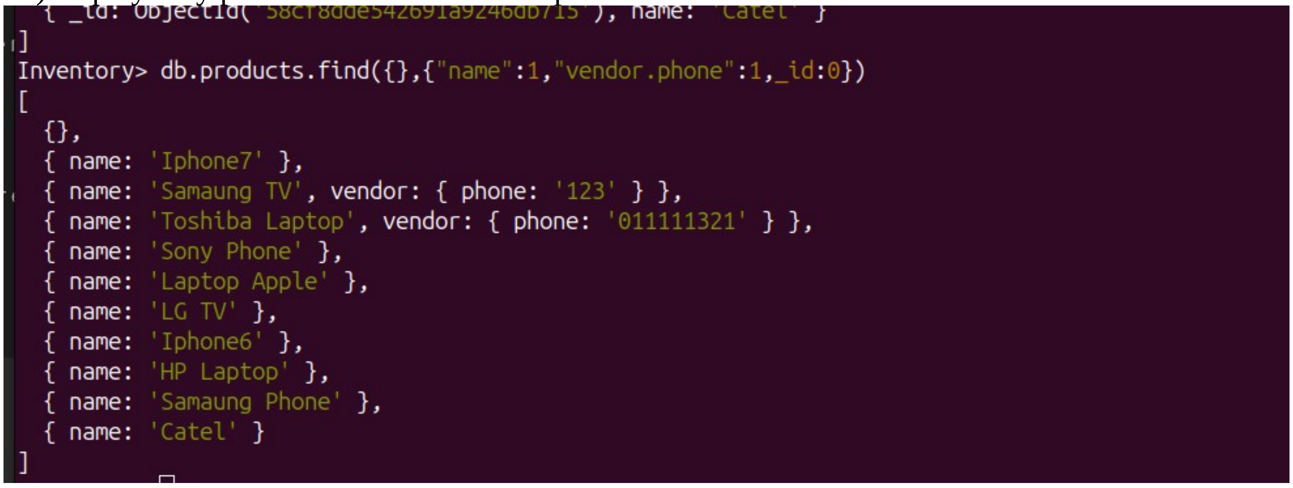
25) remove stock 70 from all products.



```
Inventory> db.products.updateMany({"stock":70},{$pull:{"stock":70}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

The $pull operator removes from an existing array all instances of a value or values that match a specified condition.

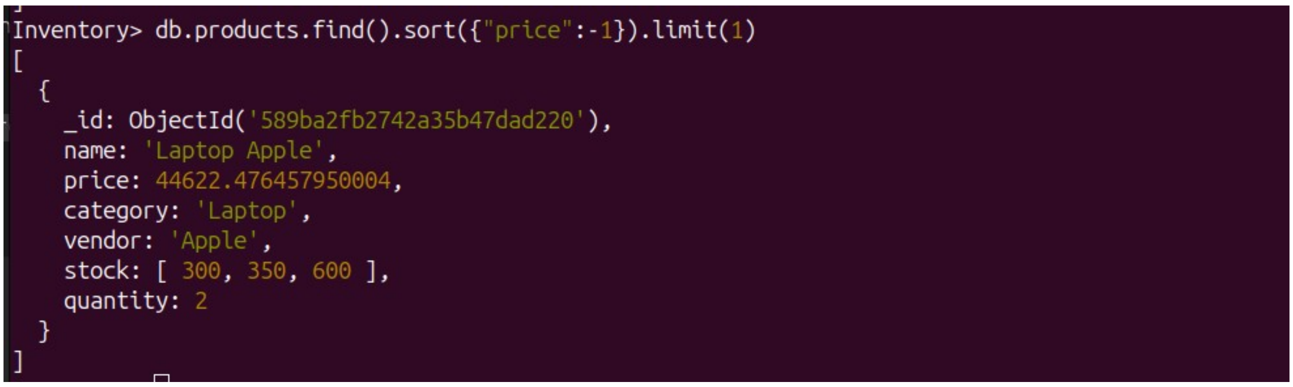
26) display only product name and vendor phone number.



```
  { _id: ObjectId('58cf8dde542691a9246db715'), name: 'Catel' }
]
Inventory> db.products.find({},{"name":1,"vendor.phone":1,_id:0})
[
  {},
  { name: 'Iphone7' },
  { name: 'Samaung TV', vendor: { phone: '123' } },
  { name: 'Toshiba Laptop', vendor: { phone: '011111321' } },
  { name: 'Sony Phone' },
  { name: 'Laptop Apple' },
  { name: 'LG TV' },
  { name: 'Iphone6' },
  { name: 'HP Laptop' },
  { name: 'Samaung Phone' },
  { name: 'Catel' }
]
```

First one for filteration I want to include all elements that have name so I put name 1 , phone put vendor.phone 1 and exclude _id so I put it 0.

27) display the most expensive product.



```
Inventory> db.products.find().sort({"price":-1}).limit(1)
[
  {
    _id: ObjectId('589ba2fb2742a35b47dad220'),
    name: 'Laptop Apple',
    price: 44622.476457950004,
    category: 'Laptop',
    vendor: 'Apple',
    stock: [ 300, 350, 600 ],
    quantity: 2
  }
]
```

-1 : for descending order
1 : for ascending order
sort products by price descending and get only the first one