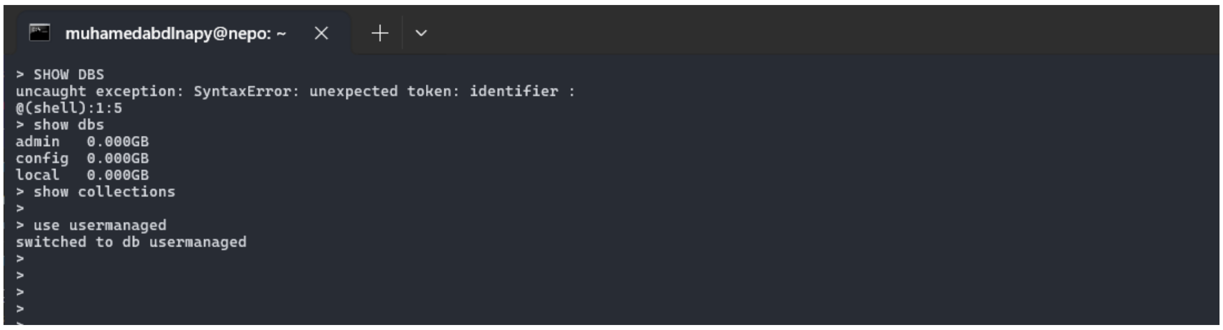
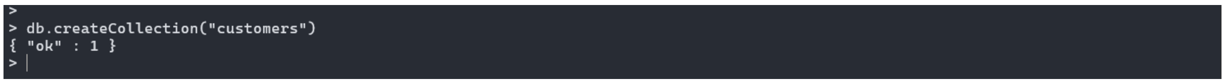
MongoDB

Lab1

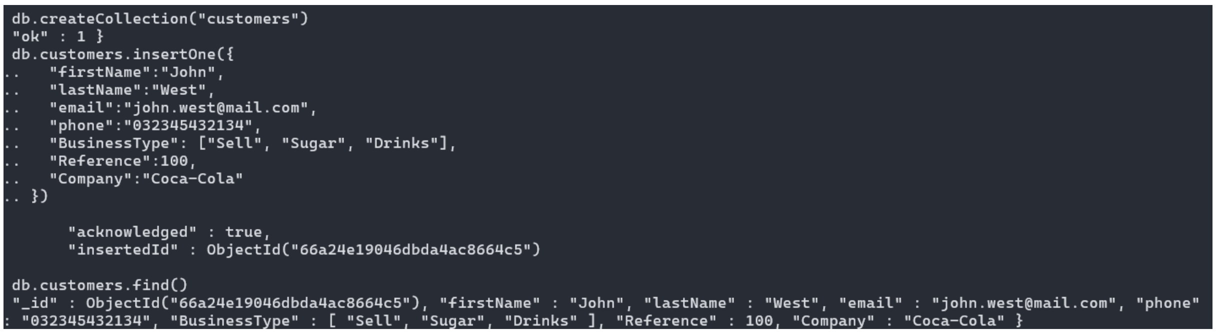
(1) Create Database


muhamedabdlnapy@nepo: ~

```
> SHOW DBS
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:5
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> show collections
>
> use usermanaged
switched to db usermanaged
>
>
>
>
```

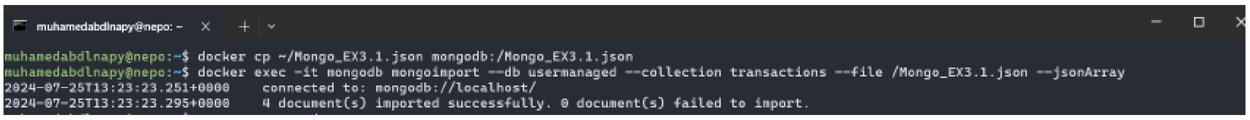

```
>
> db.createCollection("customers")
{ "ok" : 1 }
> |
```

(2) Create a Collection & Insert a Record

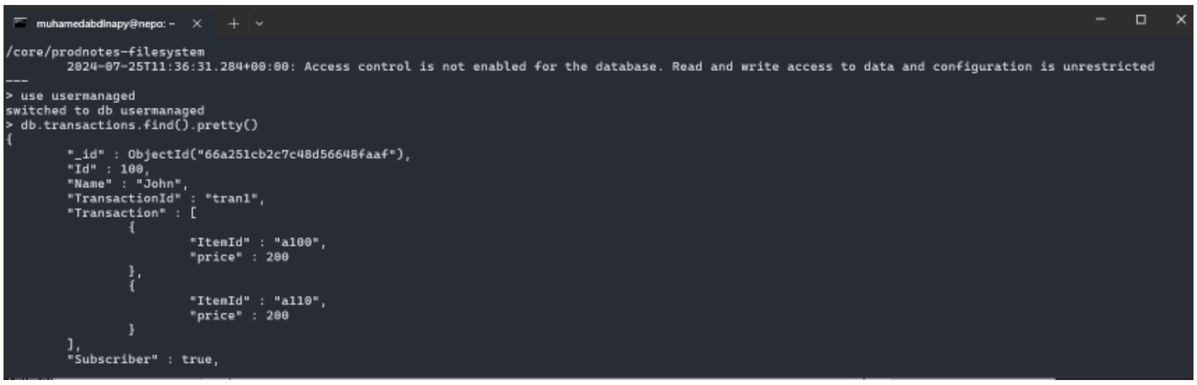

```
db.createCollection("customers")
"ok" : 1 }
db.customers.insertOne({
..    "firstName":"John",
..    "lastName":"West",
..    "email":"john.west@mail.com",
..    "phone":"032345432134",
..    "BusinessType": ["Sell", "Sugar", "Drinks"],
..    "Reference":100,
..    "Company":"Coca-Cola"
.. })

      "acknowledged" : true,
      "insertedId" : ObjectId("66a24e19046dbda4ac8664c5")

db.customers.find()
"_id" : ObjectId("66a24e19046dbda4ac8664c5"), "firstName" : "John", "lastName" : "West", "email" : "john.west@mail.com", "phone"
: "032345432134", "BusinessType" : [ "Sell", "Sugar", "Drinks" ], "Reference" : 100, "Company" : "Coca-Cola" }
```

(3) Bulk Load JSON File

3-1. Create a collection called transactions in usermanaged ,bulk load the data from a json file(import Mongo_EX3.1.json).

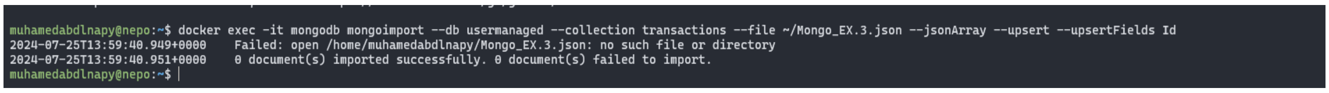

muhamedabdlnapy@nepo: ~

```
muhamedabdlnapy@nepo:~$ docker cp ~/Mongo_EX3.1.json mongodb:/Mongo_EX3.1.json
muhamedabdlnapy@nepo:~$ docker exec -it mongodb mongoimport --db usermanaged --collection transactions --file /Mongo_EX3.1.json --jsonArray
2024-07-25T13:23:23.251+0000    connected to: mongodb://localhost/
2024-07-25T13:23:23.295+0000    4 document(s) imported successfully. 0 document(s) failed to import.
```

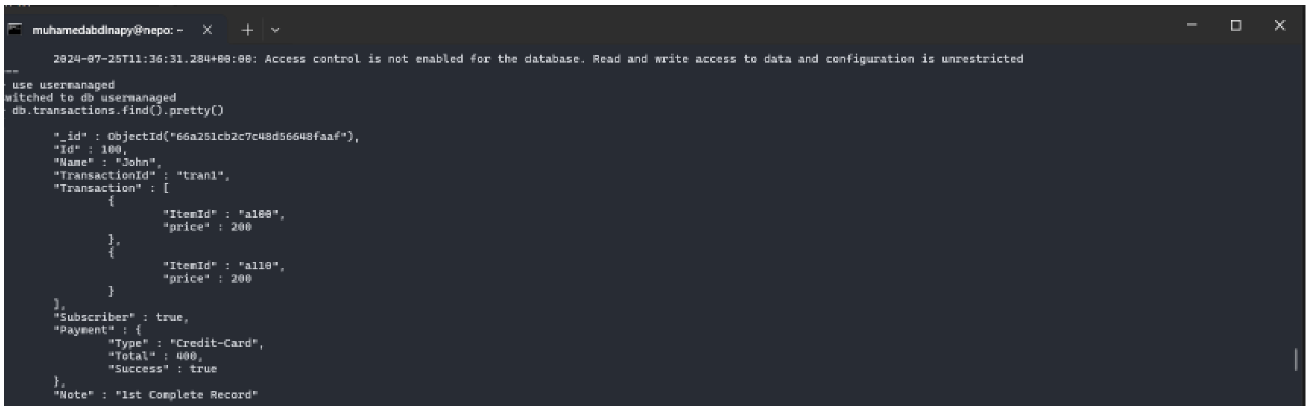


```
/core/prodnotes-filesystem
        2024-07-25T11:36:31.284+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
> use usermanaged
switched to db usermanaged
> db.transactions.find().pretty()
{
        "_id" : ObjectId("66a251cb2c7c48d56648faaf"),
        "Id" : 100,
        "Name" : "John",
        "TransactionId" : "tran1",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
```

3-3. Upsert the record from the new (import Mongo_EX3.3.json)



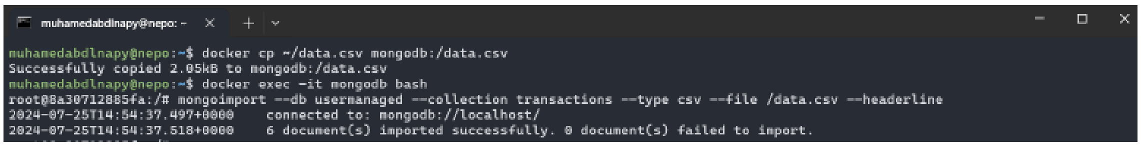
```
muhamedabdlnapy@nepo:~$ docker exec -it mongodb mongoimport --db usermanaged --collection transactions --file ~/Mongo_EX.3.json --jsonArray --upsert --upsertFields Id
2024-07-25T13:59:40.949+0000    Failed: open /home/muhamedabdlnapy/Mongo_EX.3.json: no such file or directory
2024-07-25T13:59:40.951+0000    0 document(s) imported successfully. 0 document(s) failed to import.
muhamedabdlnapy@nepo:~$ |
```



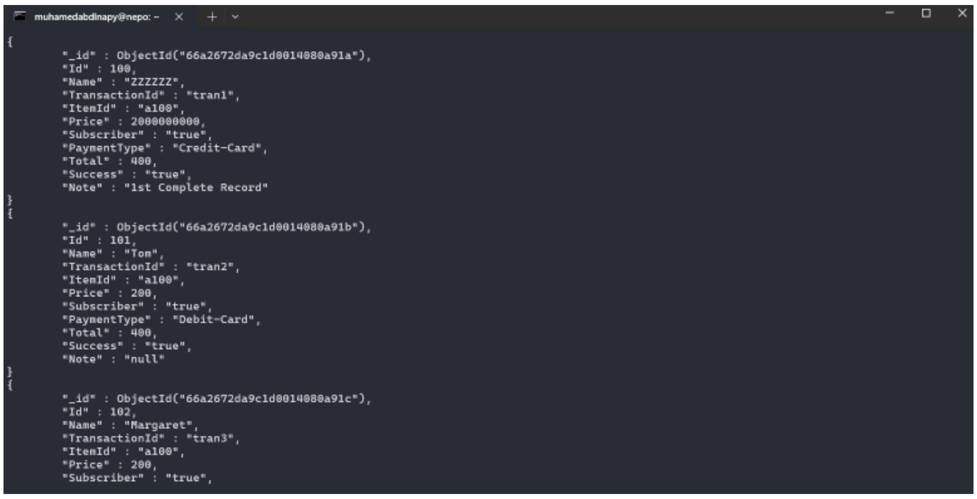
```
        2024-07-25T11:36:31.284+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
--
 use usermanaged
witched to db usermanaged
 db.transactions.find().pretty()

        "_id" : ObjectId("66a251cb2c7c48d56648faaf"),
        "Id" : 100,
        "Name" : "John",
        "TransactionId" : "tran1",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Credit-Card",
                "Total" : 400,
                "Success" : true
        },
        "Note" : "1st Complete Record"
```

<u>(4) Bulk Load CSV File</u>

4-1. Create a collection and load data from a CSV file will multiple rows. Define the keys from the header row.



```
muhamedabdlnapy@nepo:~$ docker cp ~/data.csv mongodb:/data.csv
Successfully copied 2.05kB to mongodb:/data.csv
muhamedabdlnapy@nepo:~$ docker exec -it mongodb bash
root@8a30712885fa:/# mongoimport --db usermanaged --collection transactions --type csv --file /data.csv --headerline
2024-07-25T14:54:37.497+0000    connected to: mongodb://localhost/
2024-07-25T14:54:37.518+0000    6 document(s) imported successfully. 0 document(s) failed to import.
```

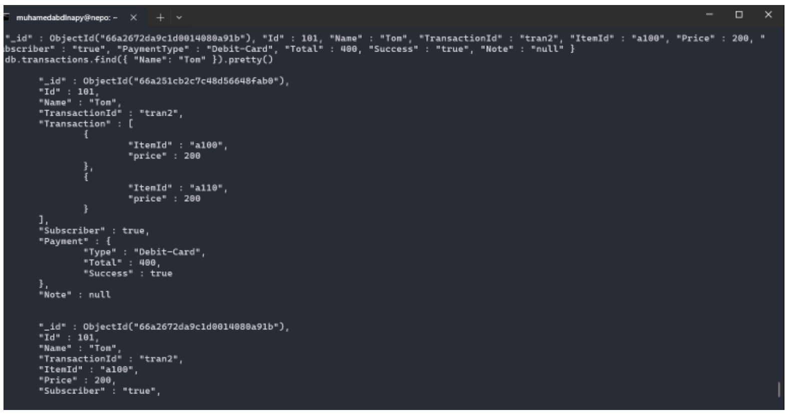


```
{
        "_id" : ObjectId("66a2672da9c1d0014080a91a"),
        "Id" : 100,
        "Name" : "ZZZZZZ",
        "TransactionId" : "tran1",
        "ItemId" : "a100",
        "Price" : 2000000000,
        "Subscriber" : "true",
        "PaymentType" : "Credit-Card",
        "Total" : 400,
        "Success" : "true",
        "Note" : "1st Complete Record"
}
{
        "_id" : ObjectId("66a2672da9c1d0014080a91b"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "ItemId" : "a100",
        "Price" : 200,
        "Subscriber" : "true",
        "PaymentType" : "Debit-Card",
        "Total" : 400,
        "Success" : "true",
        "Note" : "null"
}
{
        "_id" : ObjectId("66a2672da9c1d0014080a91c"),
        "Id" : 102,
        "Name" : "Margaret",
        "TransactionId" : "tran3",
        "ItemId" : "a100",
        "Price" : 200,
        "Subscriber" : "true",
```

(5) Query MongoDB with Conditions

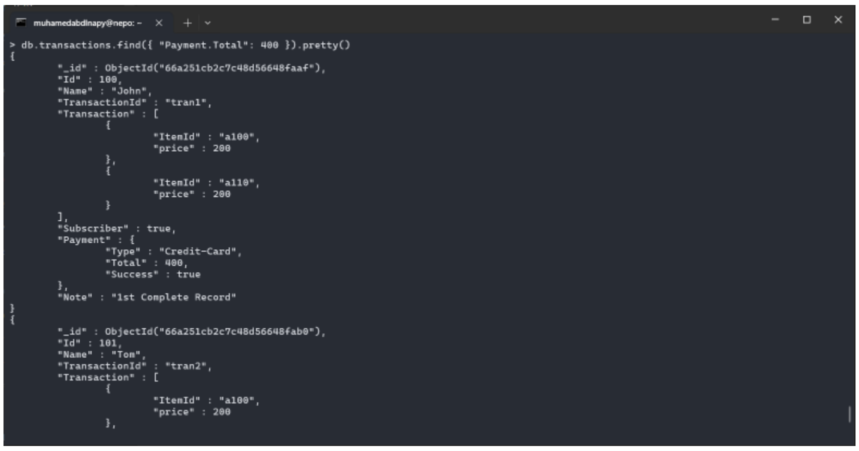
This question uses the collection (transactions) created in Exercise 3.

5-1. Find any record where Name is Tom



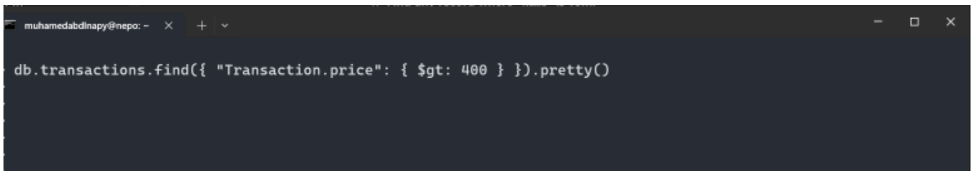
```
"_id" : ObjectId("66a2672da9c1d0014080a91b"), "Id" : 101, "Name" : "Tom", "TransactionId" : "tran2", "ItemId" : "a100", "Price" : 200, "
bscriber" : "true", "PaymentType" : "Debit-Card", "Total" : 400, "Success" : "true", "Note" : "null" }
db.transactions.find({ "Name": "Tom" }).pretty()

        "_id" : ObjectId("66a251cb2c7c48d56648fab0"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Debit-Card",
                "Total" : 400,
                "Success" : true
        },
        "Note" : null


        "_id" : ObjectId("66a2672da9c1d0014080a91b"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "ItemId" : "a100",
        "Price" : 200,
        "Subscriber" : "true",
```

5-2. Find any record where total payment amount (Payment.Total) is 400.



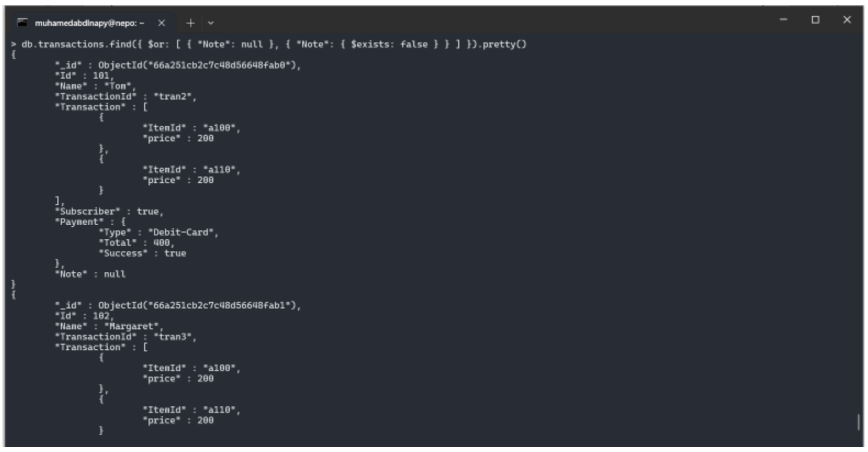
```
> db.transactions.find({ "Payment.Total": 400 }).pretty()
{
        "_id" : ObjectId("66a251cb2c7c48d56648faaf"),
        "Id" : 100,
        "Name" : "John",
        "TransactionId" : "tran1",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Credit-Card",
                "Total" : 400,
                "Success" : true
        },
        "Note" : "1st Complete Record"
}
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab0"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
```

5-3. Find any record where price (Transaction.price) is greater than 400.



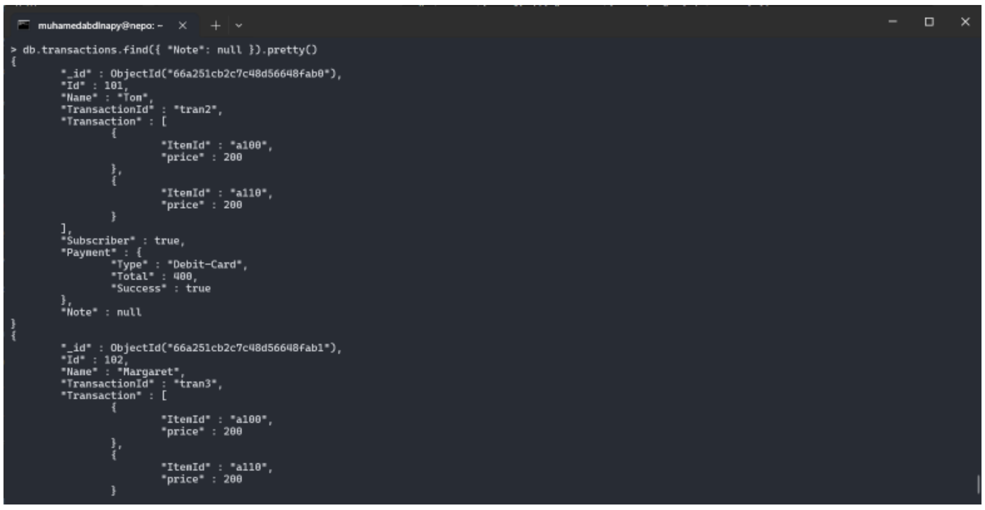
```
db.transactions.find({ "Transaction.price": { $gt: 400 } }).pretty()
```

5-4. Find any record where Note iwqs null or the key itself is missing.



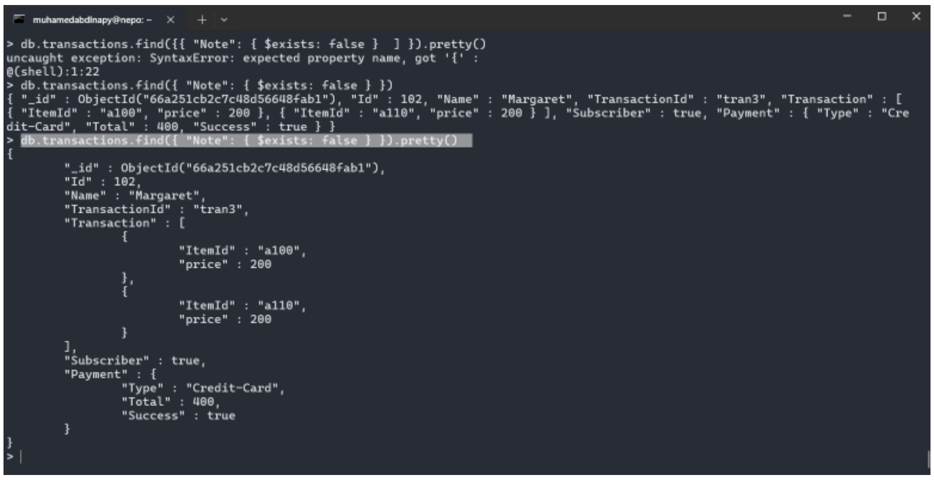
```
> db.transactions.find({ $or: [ { "Note": null }, { "Note": { $exists: false } } ] }).pretty()
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab0"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Debit-Card",
                "Total" : 400,
                "Success" : true
        },
        "Note" : null
}
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab1"),
        "Id" : 102,
        "Name" : "Margaret",
        "TransactionId" : "tran3",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
```

5-5. Find any record where Note exists and its value is null.



```
> db.transactions.find({ "Note": null }).pretty()
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab0"),
        "Id" : 101,
        "Name" : "Tom",
        "TransactionId" : "tran2",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Debit-Card",
                "Total" : 400,
                "Success" : true
        },
        "Note" : null
}
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab1"),
        "Id" : 102,
        "Name" : "Margaret",
        "TransactionId" : "tran3",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
```

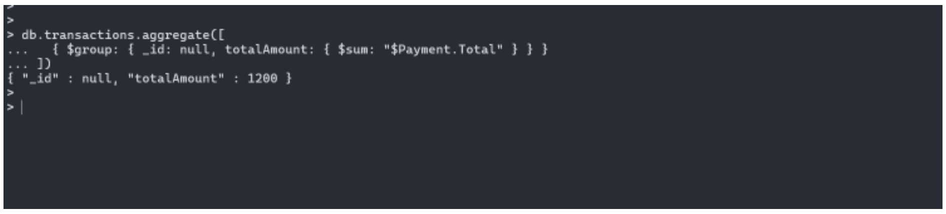
5-6. Find any record where the Note key does not exist.



```
> db.transactions.find({{ "Note": { $exists: false }  ] }).pretty()
uncaught exception: SyntaxError: expected property name, got '{' :
@(shell):1:22
> db.transactions.find({ "Note": { $exists: false } })
{ "_id" : ObjectId("66a251cb2c7c48d56648fab1"), "Id" : 102, "Name" : "Margaret", "TransactionId" : "tran3", "Transaction" : [
{ "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price" : 200 } ], "Subscriber" : true, "Payment" : { "Type" : "Cre
dit-Card", "Total" : 400, "Success" : true } }
> db.transactions.find({ "Note": { $exists: false } }).pretty()
{
        "_id" : ObjectId("66a251cb2c7c48d56648fab1"),
        "Id" : 102,
        "Name" : "Margaret",
        "TransactionId" : "tran3",
        "Transaction" : [
                {
                        "ItemId" : "a100",
                        "price" : 200
                },
                {
                        "ItemId" : "a110",
                        "price" : 200
                }
        ],
        "Subscriber" : true,
        "Payment" : {
                "Type" : "Credit-Card",
                "Total" : 400,
                "Success" : true
        }
}
> |
```

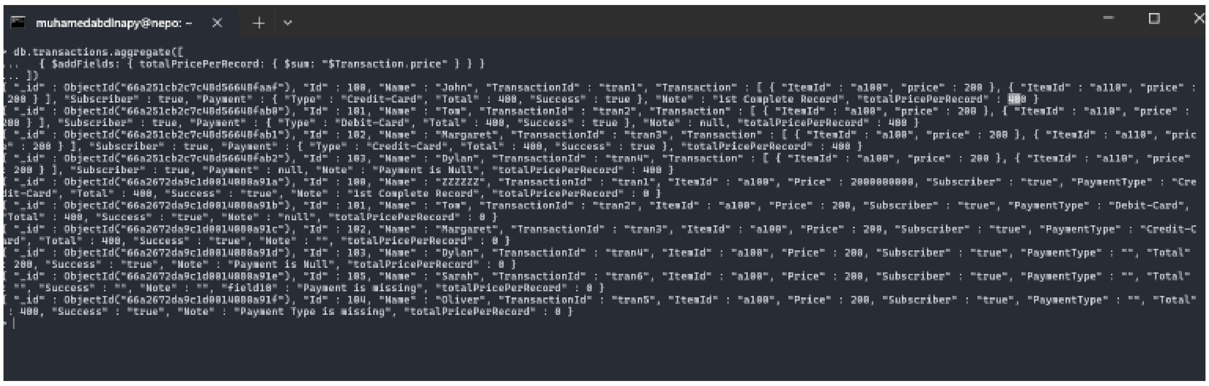
6) Aggregation with MongoDB

6-1. Calculate the total transaction amount by adding up Payment.Total in all records.



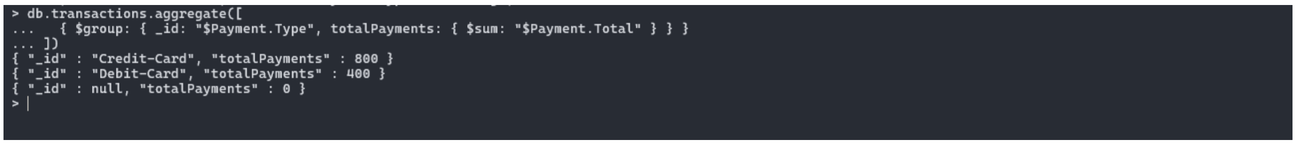
```
>
> db.transactions.aggregate([
...     { $group: { _id: null, totalAmount: { $sum: "$Payment.Total" } } }
... ])
{ "_id" : null, "totalAmount" : 1200 }
>
> |
```

6-2. Get the total price per record by adding up the price values in the Transaction array (Transaction.price).



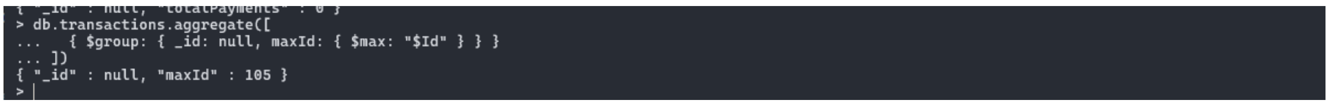
```
> db.transactions.aggregate([
...    { $addFields: { totalPricePerRecord: { $sum: "$Transaction.price" } } }
... ])
{ "_id" : ObjectId("66a251cb2c7c48d56648faaf"), "Id" : 100, "Name" : "John", "TransactionId" : "tran1", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price" :
 200 } ], "Subscriber" : true, "Payment" : { "Type" : "Credit-Card", "Total" : 400, "Success" : true }, "Note" : "1st Complete Record", "totalPricePerRecord" : 400 }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab0"), "Id" : 101, "Name" : "Tom", "TransactionId" : "tran2", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price" :
200 } ], "Subscriber" : true, "Payment" : { "Type" : "Debit-Card", "Total" : 400, "Success" : true }, "Note" : null, "totalPricePerRecord" : 400 }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab1"), "Id" : 102, "Name" : "Margaret", "TransactionId" : "tran3", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "pric
e" : 200 } ], "Subscriber" : true, "Payment" : { "Type" : "Credit-Card", "Total" : 400, "Success" : true }, "totalPricePerRecord" : 400 }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab2"), "Id" : 103, "Name" : "Dylan", "TransactionId" : "tran4", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price"
: 200 } ], "Subscriber" : true, "Payment" : null, "Note" : "Payment is Null", "totalPricePerRecord" : 400 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91a"), "Id" : 100, "Name" : "ZZZZZZ", "TransactionId" : "tran1", "ItemId" : "a100", "Price" : 2000000000, "Subscriber" : "true", "PaymentType" : "Cre
dit-Card", "Total" : 400, "Success" : "true", "Note" : "1st Complete Record", "totalPricePerRecord" : 0 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91b"), "Id" : 101, "Name" : "Tom", "TransactionId" : "tran2", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "Debit-Card",
"Total" : 400, "Success" : "true", "Note" : "null", "totalPricePerRecord" : 0 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91c"), "Id" : 102, "Name" : "Margaret", "TransactionId" : "tran3", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "Credit-C
ard", "Total" : 400, "Success" : "true", "Note" : "", "totalPricePerRecord" : 0 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91d"), "Id" : 103, "Name" : "Dylan", "TransactionId" : "tran4", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "", "Total"
: 200, "Success" : "true", "Note" : "Payment is Null", "totalPricePerRecord" : 0 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91e"), "Id" : 105, "Name" : "Sarah", "TransactionId" : "tran6", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "", "Total"
: "", "Success" : "", "Note" : "", "field10" : "Payment is missing", "totalPricePerRecord" : 0 }
{ "_id" : ObjectId("66a2672da9c1d0014808a91f"), "Id" : 104, "Name" : "Oliver", "TransactionId" : "tran5", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "", "Total"
: 400, "Success" : "true", "Note" : "Payment Type is missing", "totalPricePerRecord" : 0 }
> |
```

6-3. Calculate total payments (Payment.Total) for each payment type (Payment.Type).



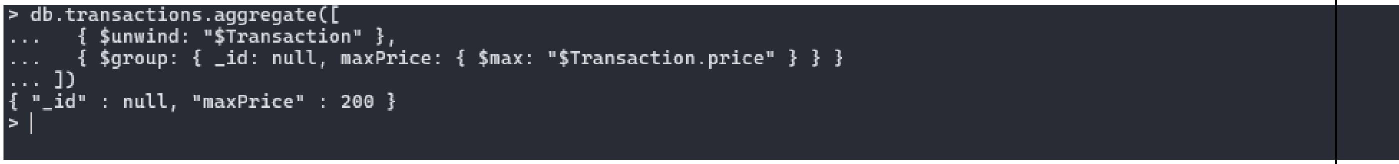
```
> db.transactions.aggregate([
...   { $group: { _id: "$Payment.Type", totalPayments: { $sum: "$Payment.Total" } } }
... ])
{ "_id" : "Credit-Card", "totalPayments" : 800 }
{ "_id" : "Debit-Card", "totalPayments" : 400 }
{ "_id" : null, "totalPayments" : 0 }
> |
```

6-4. Find the max Id.



```
{ "_id" : null, "totalPayments" : 0 }
> db.transactions.aggregate([
...   { $group: { _id: null, maxId: { $max: "$Id" } } }
... ])
{ "_id" : null, "maxId" : 105 }
> |
```

6-5. Find the max price (Transaction.price).

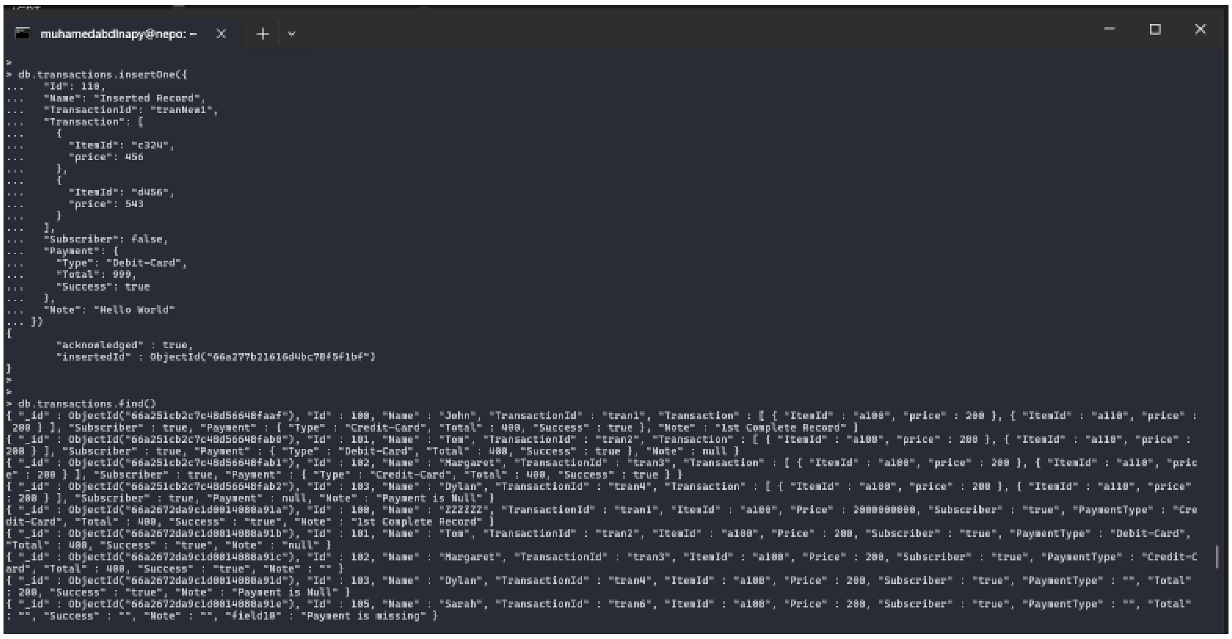


```
> db.transactions.aggregate([
...   { $unwind: "$Transaction" },
...   { $group: { _id: null, maxPrice: { $max: "$Transaction.price" } } }
... ])
{ "_id" : null, "maxPrice" : 200 }
> |
```

(7) CRUD Operations

This question uses the collection (transactions)that created in Exercise 3.

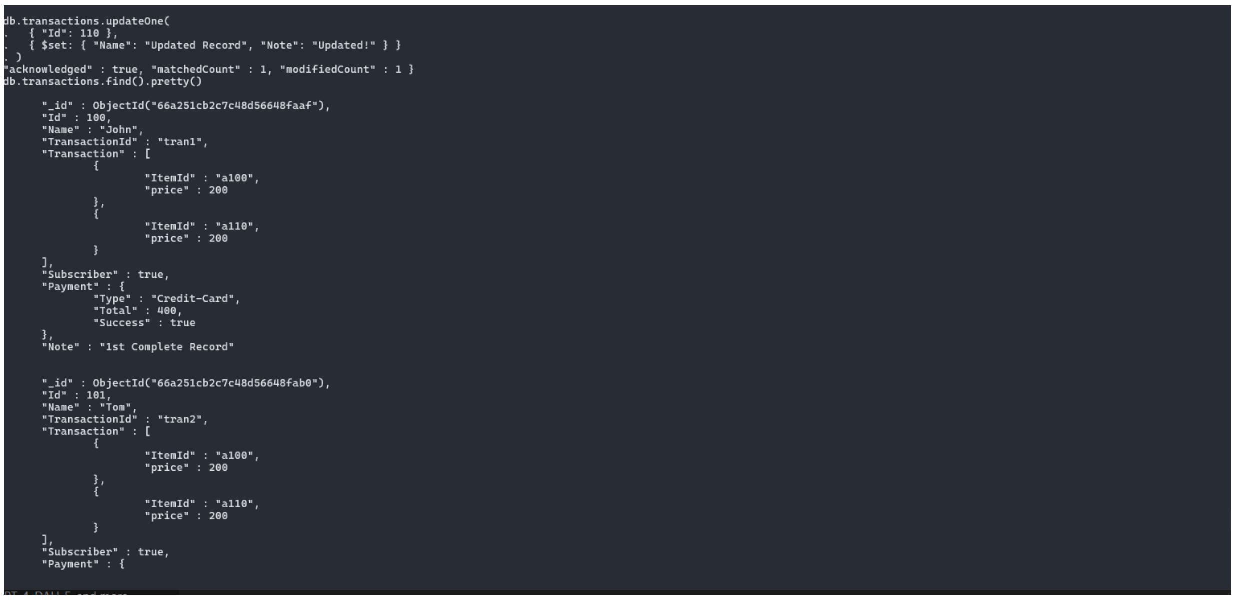
7-1. Insert a record below



muhamedabdlnapy@nepo: ~

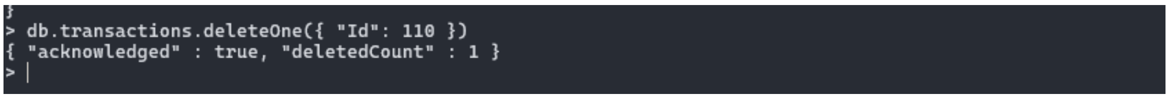
```
>
> db.transactions.insertOne({
...   "Id": 110,
...   "Name": "Inserted Record",
...   "TransactionId": "tranNew1",
...   "Transaction": [
...     {
...       "ItemId": "c324",
...       "price": 456
...     },
...     {
...       "ItemId": "d456",
...       "price": 543
...     }
...   ],
...   "Subscriber": false,
...   "Payment": {
...     "Type": "Debit-Card",
...     "Total": 999,
...     "Success": true
...   },
...   "Note": "Hello World"
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("66a277b21616d4bc78f5f1bf")
}
>
>
> db.transactions.find()
{ "_id" : ObjectId("66a251cb2c7c48d56648faaf"), "Id" : 100, "Name" : "John", "TransactionId" : "tran1", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price" :
 200 } ], "Subscriber" : true, "Payment" : { "Type" : "Credit-Card", "Total" : 400, "Success" : true }, "Note" : "1st Complete Record" }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab0"), "Id" : 101, "Name" : "Tom", "TransactionId" : "tran2", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price" :
200 } ], "Subscriber" : true, "Payment" : { "Type" : "Debit-Card", "Total" : 400, "Success" : true }, "Note" : null }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab1"), "Id" : 102, "Name" : "Margaret", "TransactionId" : "tran3", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "pric
e" : 200 } ], "Subscriber" : true, "Payment" : { "Type" : "Credit-Card", "Total" : 400, "Success" : true } }
{ "_id" : ObjectId("66a251cb2c7c48d56648fab2"), "Id" : 103, "Name" : "Dylan", "TransactionId" : "tran4", "Transaction" : [ { "ItemId" : "a100", "price" : 200 }, { "ItemId" : "a110", "price"
: 200 } ], "Subscriber" : true, "Payment" : null, "Note" : "Payment is Null" }
{ "_id" : ObjectId("66a2672da9c1d0014088a91a"), "Id" : 100, "Name" : "ZZZZZZ", "TransactionId" : "tran1", "ItemId" : "a100", "Price" : 2000000000, "Subscriber" : "true", "PaymentType" : "Cre
dit-Card", "Total" : 400, "Success" : "true", "Note" : "1st Complete Record" }
{ "_id" : ObjectId("66a2672da9c1d0014088a91b"), "Id" : 101, "Name" : "Tom", "TransactionId" : "tran2", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "Debit-Card",
"Total" : 400, "Success" : "true", "Note" : "null" }
{ "_id" : ObjectId("66a2672da9c1d0014088a91c"), "Id" : 102, "Name" : "Margaret", "TransactionId" : "tran3", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "Credit-C
ard", "Total" : 400, "Success" : "true", "Note" : "" }
{ "_id" : ObjectId("66a2672da9c1d0014088a91d"), "Id" : 103, "Name" : "Dylan", "TransactionId" : "tran4", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "", "Total"
: 200, "Success" : "true", "Note" : "Payment is Null" }
{ "_id" : ObjectId("66a2672da9c1d0014088a91e"), "Id" : 105, "Name" : "Sarah", "TransactionId" : "tran6", "ItemId" : "a100", "Price" : 200, "Subscriber" : "true", "PaymentType" : "", "Total"
: "", "Success" : "", "Note" : "", "field10" : "Payment is missing" }
```

7-2. Updating the new inserted record above. Make Name='Updated Record' & Note='Updated!'



```
db.transactions.updateOne(
.   { "Id": 110 },
.   { $set: { "Name": "Updated Record", "Note": "Updated!" } }
. )
"acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
db.transactions.find().pretty()

      "_id" : ObjectId("66a251cb2c7c48d56648faaf"),
      "Id" : 100,
      "Name" : "John",
      "TransactionId" : "tran1",
      "Transaction" : [
              {
                      "ItemId" : "a100",
                      "price" : 200
              },
              {
                      "ItemId" : "a110",
                      "price" : 200
              }
      ],
      "Subscriber" : true,
      "Payment" : {
              "Type" : "Credit-Card",
              "Total" : 400,
              "Success" : true
      },
      "Note" : "1st Complete Record"


      "_id" : ObjectId("66a251cb2c7c48d56648fab0"),
      "Id" : 101,
      "Name" : "Tom",
      "TransactionId" : "tran2",
      "Transaction" : [
              {
                      "ItemId" : "a100",
                      "price" : 200
              },
              {
                      "ItemId" : "a110",
                      "price" : 200
              }
      ],
      "Subscriber" : true,
      "Payment" : {
```

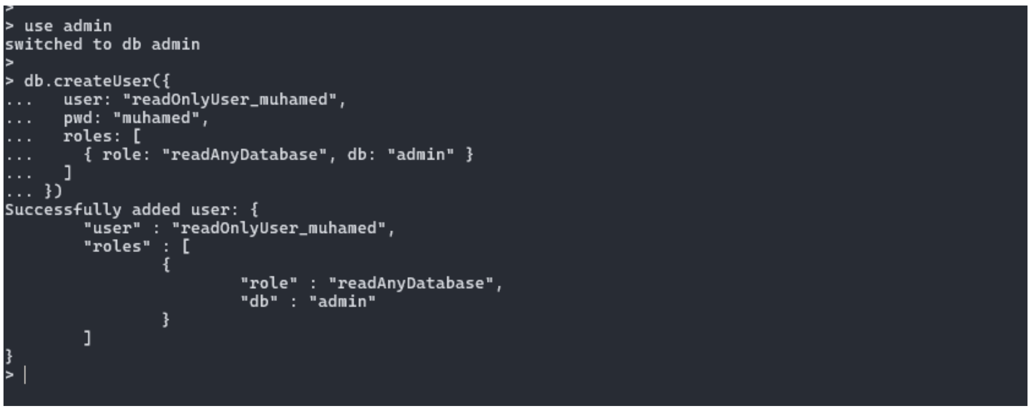
7-3. Delete the record inserted above by using Id



```
}
> db.transactions.deleteOne({ "Id": 110 })
{ "acknowledged" : true, "deletedCount" : 1 }
> |
```

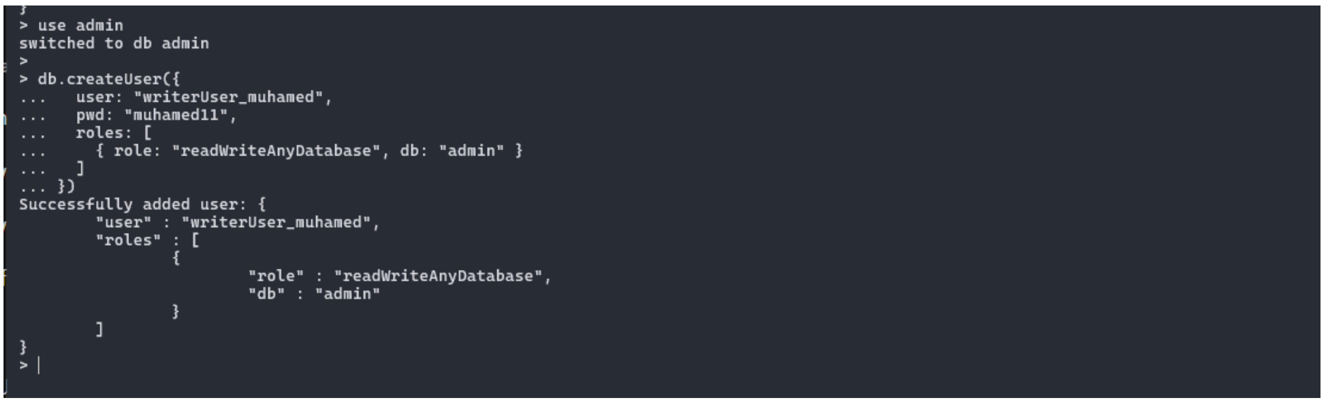
(8) User Creation

8-1. Create a read only user who can query records from collections from all databases.



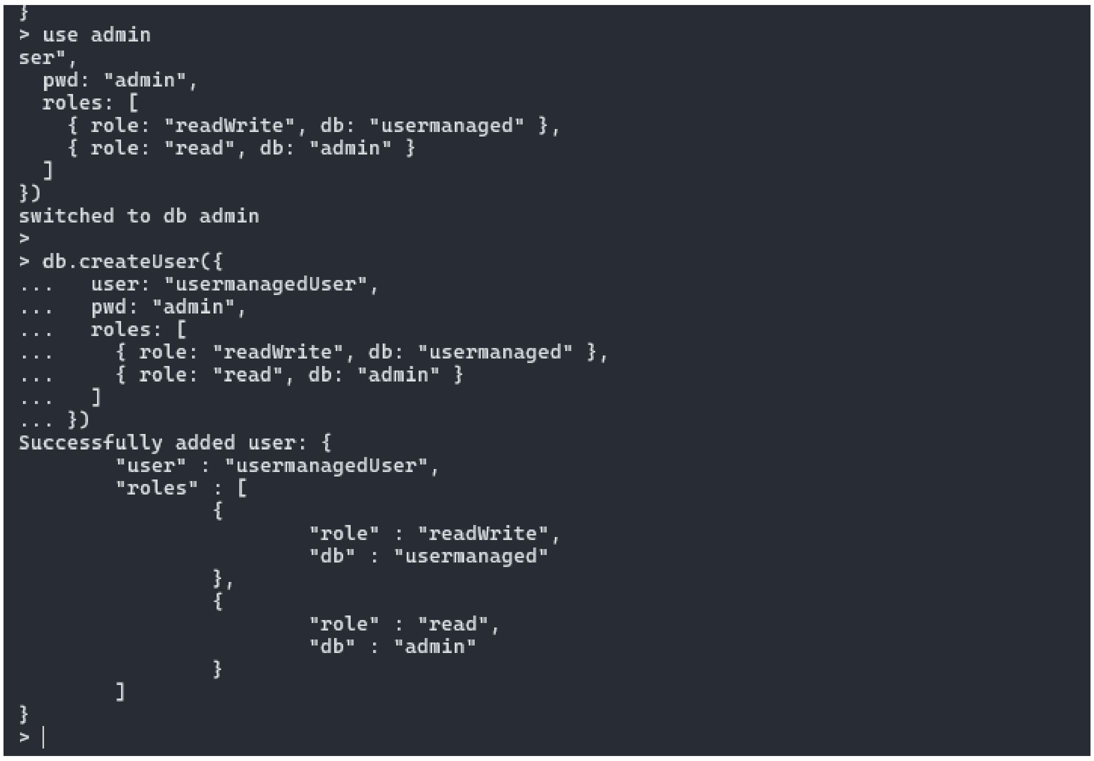
```
>
> use admin
switched to db admin
>
> db.createUser({
...    user: "readOnlyUser_muhamed",
...    pwd: "muhamed",
...    roles: [
...      { role: "readAnyDatabase", db: "admin" }
...    ]
... })
Successfully added user: {
        "user" : "readOnlyUser_muhamed",
        "roles" : [
                {
                        "role" : "readAnyDatabase",
                        "db" : "admin"
                }
        ]
}
> |
```

8-2. Create a writer user who can create collections and do CRUD operations in any collections.



```
}
> use admin
switched to db admin
>
> db.createUser({
...    user: "writerUser_muhamed",
...    pwd: "muhamed11",
...    roles: [
...      { role: "readWriteAnyDatabase", db: "admin" }
...    ]
... })
Successfully added user: {
        "user" : "writerUser_muhamed",
        "roles" : [
                {
                        "role" : "readWriteAnyDatabase",
                        "db" : "admin"
                }
        ]
}
> |
```

8-3. Create a usermanaged user who can do the writer operation in the usermanaged database and read only for the rest of the databases.



```
}
> use admin
ser",
  pwd: "admin",
  roles: [
    { role: "readWrite", db: "usermanaged" },
    { role: "read", db: "admin" }
  ]
})
switched to db admin
>
> db.createUser({
...   user: "usermanagedUser",
...   pwd: "admin",
...   roles: [
...     { role: "readWrite", db: "usermanaged" },
...     { role: "read", db: "admin" }
...   ]
... })
Successfully added user: {
        "user" : "usermanagedUser",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "usermanaged"
                },
                {
                        "role" : "read",
                        "db" : "admin"
                }
        ]
}
> |
```