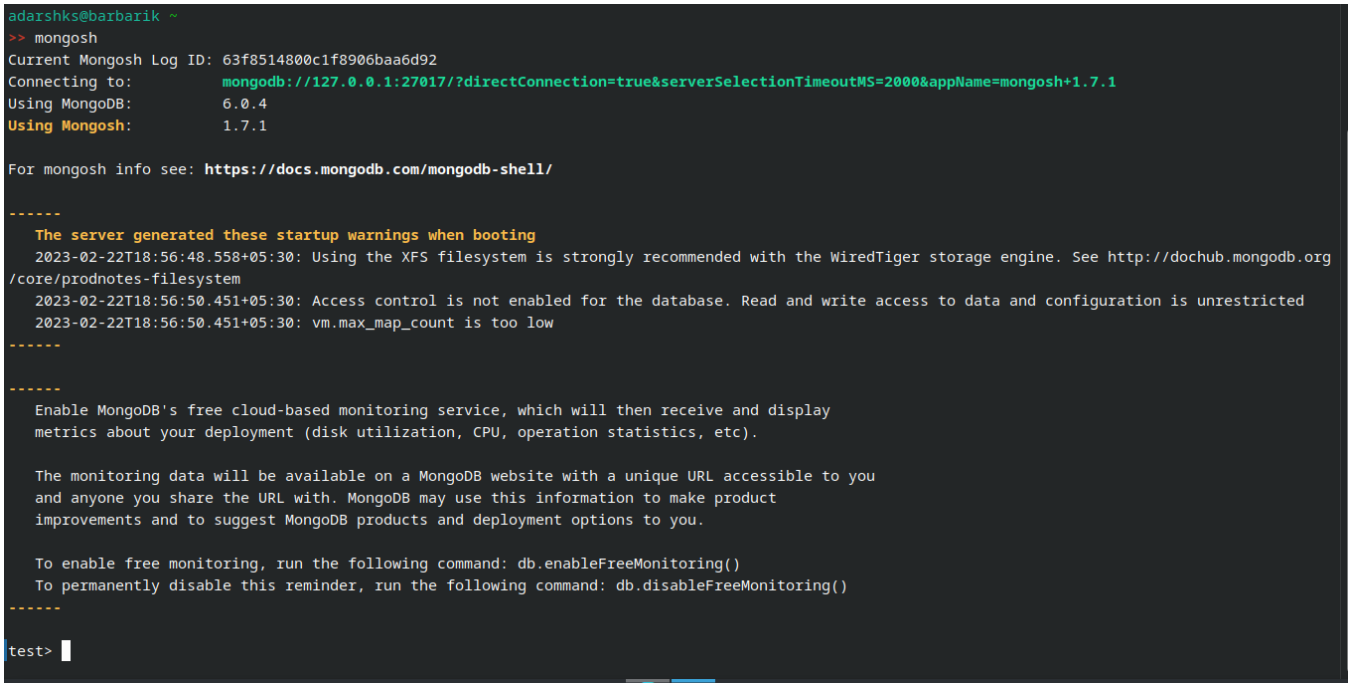
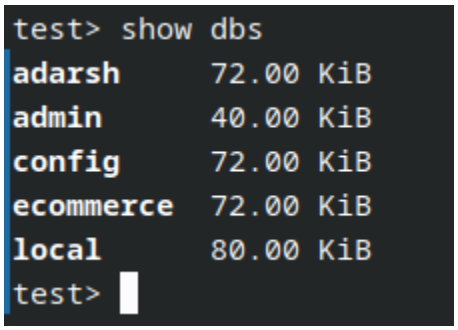
CRUD operations in MongoDB

Starting mongo server in terminal

To start the mongo daemon in the terminal run the command mongo (in windows) or mongosh (in Linux)
In Linux systems it looks like this :



Run the command show dbs command shows databases(DBs) which are present in the system, and are having at least one document, this is a constraint in MongoDB that only those DBs are retained which are having at least one Document in them.



The Default DBs that come with mongodb are admin, config & local. The remaining DBs that are visible in the screenshot are created by me.

On using command : `use phoneCart`

if : db with name phoneCart is already present then it uses that

else : it creates a new db named phoneCart

```
test> use phoneCart
switched to db phoneCart
phoneCart>
```

But as we know that if we do not add any document to the DB then it wont retain in the system, we can see that by show dbs command :

```
phoneCart> show dbs
adarsh      72.00 KiB
admin       40.00 KiB
config     108.00 KiB
ecommerce   72.00 KiB
local       80.00 KiB
phoneCart>
```

Adding Document to collection : **Create** operation

First switch to the database in which you want to add the document.

insertOne() method

This method is used to insert only one document in the database.
Syntax : `db.<collection_Name>.insertOne({key : "value"});`

If the collection collection_Name exists then it will insert the document into that but if it doesn't then it will create that collection into the DB. And the mongoDB will return an object containing the Object ID assigned to that document. Ex : on running the command

```
db.smartPhones.insertOne({name: "Samsung s23", price: 100000, rating: 4.5, qty: 233, sold: 98});
```

We see the following

```
phoneCart> db.smartPhones.insertOne({name: "Samsung s23", price: 100000, rating: 4.5, qty: 233, sold: 98});
{
  acknowledged: true,
  insertedId: ObjectId("63f9828c546ace9c767dde0b")
}
phoneCart>
```

insertMany() method

This method is used to insert multiple documents in the database. We have to keep the objects inside an array.

Syntax : `db.<collection_Name>.insertMany([{key1 : ”value1”} , {key2 : ”value2”} , {key3 : ”value3”}]);`
Here {key1 : ”value1”} , {key2 : ”value2”} , {key3 : ”value3”} are three different objects and are kept in an array.

db.smartPhones.insertMany([{name: "Samsung s23 ultra", price: 100000, rating: 4.8, qty: 233, sold: 98}, {name: "iPhone 14", price: 129000, rating: 4.4, qty: 133, sold: 598}, {name: "Xiaomi 13", price: 47000, rating: 3.5, qty: 633, sold: 58, hasIRBlaster:true}])

```
phoneCart> db.smartPhones.insertMany([{name: "Samsung s23 ultra", price: 100000, rating: 4.8, qty: 233,
sold: 98}, {name: "iPhone 14", price: 129000, ratirating: 4.4, qty: 133, sold: 598}, {name: "Xiaomi 13",
 price: 47000, rating: 3.5, qty: 633, sold: 58, hasIRBlaster:true}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("63f987db546ace9c767dde0f"),
    '1': ObjectId("63f987db546ace9c767dde10"),
    '2': ObjectId("63f987db546ace9c767dde11")
  }
}
phoneCart>
```

We can see that 2 ObjectIDs are returned which denote the three documents that are inserted into the Database.

By the way you can see the list of all the collections present in the DB with the command :
show collections;

```
phoneCart> show collections;
smartPhones
phoneCart>
```

Do Note that

- Mongo gives a unique id to all the elements that are added in the DB, so if we add the same valued documents multiple times then it is added in the DB with different ids.
- Primary(1°) key in MongoDB is ObjectId assigned by MongoDB itself
- MongoDB doesn't enforce a schema on the tables (like in SQL we have to insert data in the same column, format, ...), it's more like JS Objects.

Reading from Database : **Read** operation

The `find()` method

To read all documents from the DB
`db.items.find()`

```
phoneCart> db.smartPhones.find()
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde10"),
    name: 'iPhone 14',
    price: 129000,
    ratirating: 4.4,
    qty: 133,
    sold: 598
  },
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.5,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```

limit() method

The `limit()` method, here provided a value will limit the o/p result to the first 2 elements in the DB.

```
phoneCart> db.smartPhones.find().limit(2);
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  }
]
phoneCart>
```

Using Filter in find()

`db.items.find({"rating" : 4.8})` will return the document that exactly matches the filter provided in the find method if no document matches then no o/p is returned

```
phoneCart> db.smartPhones.find({"rating" : 4.8})
[
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  }
]
phoneCart>
```

Some Complex Filters

`gte` greater than equal to

```
phoneCart> db.smartPhones.find({"price" : {$gte: 100000}})
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde10"),
    name: 'iPhone 14',
    price: 129000,
    ratirating: 4.4,
    qty: 133,
    sold: 598
  }
]
phoneCart>
```

`gt` greater than

```
phoneCart> db.smartPhones.find({"price" : {$gt: 100000}})
[
  {
    _id: ObjectId("63f987db546ace9c767dde10"),
    name: 'iPhone 14',
    price: 129000,
    ratirating: 4.4,
    qty: 133,
    sold: 598
  }
]
phoneCart>
```

`lt` less than

```
phoneCart> db.smartPhones.find({"price" : {$lt: 100000}})
[
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.5,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```

`lte` less than equal to

```
phoneCart> db.smartPhones.find({"price" : {$lte: 100000}})
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.5,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```

And Operator(,)

```
phoneCart> db.smartPhones.find({"rating": {$gt: 4}, "price": {$lte: 100000}})
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  }
]
phoneCart>
```

`or` Operator

```
phoneCart> db.smartPhones.find({ $or: [ {rating: {$gt: 4}}, {"qty": {$gte: 500}} ] })
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 100000,
    rating: 4.5,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 100000,
    rating: 4.8,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.5,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```

Update DB

`updateOne()` method

`db.<Collection_name>.updateOne({filterObject}, {$set : {vlaueToBeChanged}})`

The first element which matches the filterObject is updated.

db.smartPhones.updateOne({name: "Xiaomi 13"}, {$set: {rating:3.9}})

```
phoneCart> db.smartPhones.updateOne({name: "Xiaomi 13"}, {$set: {rating:3.9}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

You can verify if the update has been done or not

```
phoneCart> db.smartPhones.find({name: "Xiaomi 13"})
[
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.9,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```

`updateMany()` method

`db.<Collection_name>.updateMany({filterObject}, {$set : {vlaueToBeChanged}})`

All the documents which matches the filter object is updated

```
phoneCart> db.smartPhones.updateMany({price: 100000}, {$set: {price : 90000, rating : 4.6}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
```

You can verify if the update has been done or not

```
phoneCart> db.smartPhones.find({price: 90000})
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 90000,
    rating: 4.6,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 90000,
    rating: 4.6,
    qty: 233,
    sold: 98
  }
]
phoneCart>
```

`delete` operation

`deleteOne()` method

```
phoneCart> db.smartPhones.deleteOne({name: "iPhone 14"})
{ acknowledged: true, deletedCount: 1 }
```

Now of we try to `find()` method with the same object we will not get anything as they are deleted

```
phoneCart> db.smartPhones.find({name: "iPhone 14"})

phoneCart>
```

`deleteMany()` method

Now on having a look at all the documents in the Collection :

```
phoneCart> db.smartPhones.find();
[
  {
    _id: ObjectId("63f9828c546ace9c767dde0b"),
    name: 'Samsung s23',
    price: 90000,
    rating: 4.6,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde0f"),
    name: 'Samsung s23 ultra',
    price: 90000,
    rating: 4.6,
    qty: 233,
    sold: 98
  },
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.9,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
```

After using the `deleteMany()` method we get :

```
phoneCart> db.smartPhones.deleteMany({sold : {$gt : 90}});
{ acknowledged: true, deletedCount: 2 }
```

Now on running the `find()` command we get :

```
phoneCart> db.smartPhones.find();
[
  {
    _id: ObjectId("63f987db546ace9c767dde11"),
    name: 'Xiaomi 13',
    price: 47000,
    rating: 3.9,
    qty: 633,
    sold: 58,
    hasIRBlaster: true
  }
]
phoneCart>
```