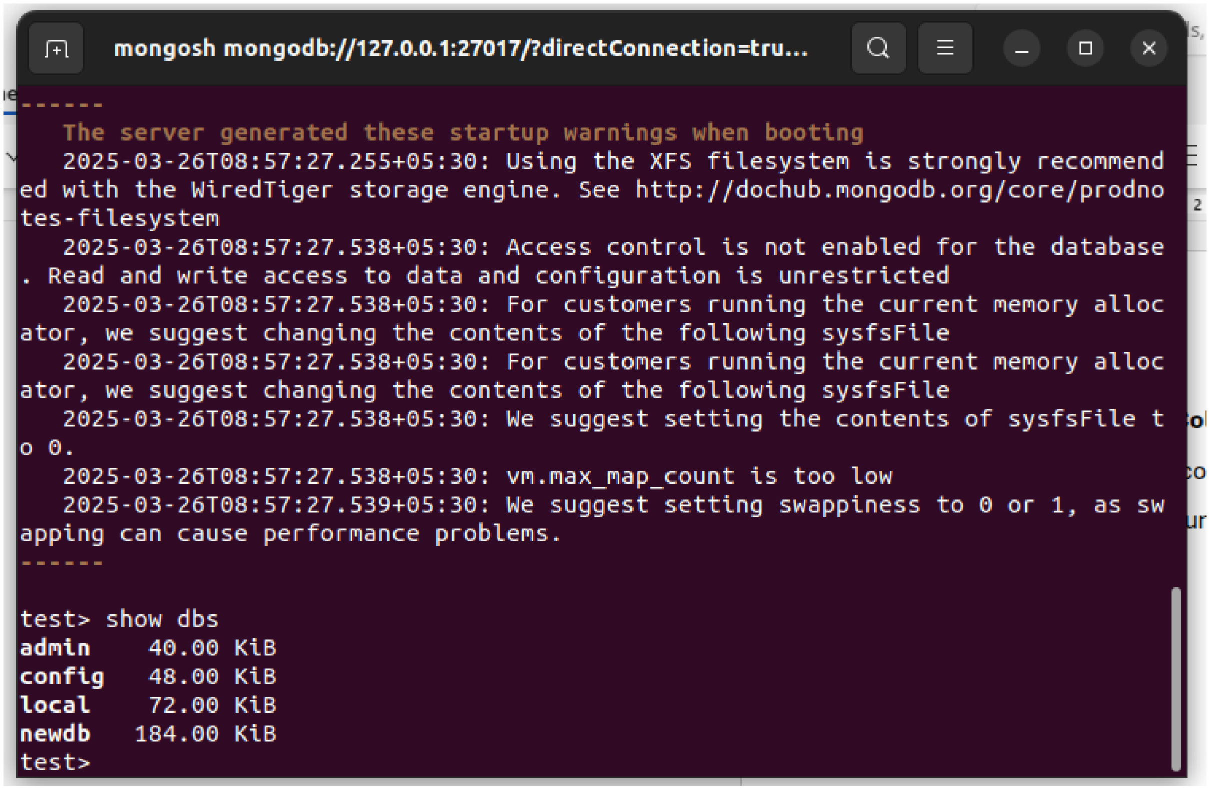
What is a Database and Collection in MongoDB?

Database: A container for collections, similar to a database in SQL.

Collection: A group of documents, like a table in SQL.



```
------
   The server generated these startup warnings when booting
   2025-03-26T08:57:27.255+05:30: Using the XFS filesystem is strongly recommend
ed with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodno
tes-filesystem
   2025-03-26T08:57:27.538+05:30: Access control is not enabled for the database
. Read and write access to data and configuration is unrestricted
   2025-03-26T08:57:27.538+05:30: For customers running the current memory alloc
ator, we suggest changing the contents of the following sysfsFile
   2025-03-26T08:57:27.538+05:30: For customers running the current memory alloc
ator, we suggest changing the contents of the following sysfsFile
   2025-03-26T08:57:27.538+05:30: We suggest setting the contents of sysfsFile t
o 0.
   2025-03-26T08:57:27.538+05:30: vm.max_map_count is too low
   2025-03-26T08:57:27.539+05:30: We suggest setting swappiness to 0 or 1, as sw
apping can cause performance problems.
------

test> show dbs
admin    40.00 KiB
config   48.00 KiB
local    72.00 KiB
newdb   184.00 KiB
test>
```

Show dbs – Create or switch to a database

db.createCollection("newdb") - Create a collection

Document in MongoDB

A document is the basic unit of data storage in MongoDB. It is a JSON-like structure that contains field-value pairs.

```
{
 "_id": ObjectId("507f1f77bcf86cd799439011"),
 "name": “Brindha",
 }
```

Documents -

A **document** is the basic unit of data storage in MongoDB. It is a JSON-like structure that contains field-value pairs.

```
newdb> db.newproducts.find()
[
 { _id: ObjectId('67e3a4198ea3f781676b140b'), name: 'Laptop' },
 {
  _id: ObjectId('67e3a56a8ea3f781676b140c'),
  name: 'Laptop',
  price: 34300
 },
 {
  _id: ObjectId('67e3a5908ea3f781676b140d'),
  name: 'Laptop',
  price: 34300
 },
 {
  _id: ObjectId('67e3bfc58ea3f781676b140f'),
  name: 'Laptop',
  price: 34300
 },
 {
  _id: ObjectId('67e3bfc58ea3f781676b1410'),
  name: 'Laptop1',
  price: 34300
 },
 {
  _id: ObjectId('67e3bff68ea3f781676b1411'),
  name: 'Laptop1',
  price: 34300
 }
```

What is BSON?

- BSON (Binary JSON) is the internal storage format of MongoDB.
- It supports additional data types like Date, Decimal128, and Binary.

```
{ _id: new ObjectId('67e3a4198ea3f781676b140b'), name: 'Laptop' },
{ _id: new ObjectId('67e3a56a8ea3f781676b140c'), name: 'Laptop', price: 34300 },
{ _id: new ObjectId('67e3a5908ea3f781676b140d'), name: 'Laptop', price: 34300 },
{ _id: new ObjectId('67e3bfc58ea3f781676b140f'), name: 'Laptop', price: 34300 },
{ _id: new ObjectId('67e3bfc58ea3f781676b1410'), name: 'Laptop1', price: 34300 },
{ _id: new ObjectId('67e3bff68ea3f781676b1411'), name: 'Laptop1', price: 34300 };
```

Bson:

```
[
 { _id: ObjectId("67e3a4198ea3f781676b140b"), name: "Laptop" },
 { _id: ObjectId("67e3a56a8ea3f781676b140c"), name: "Laptop", price:
NumberInt(34300) },
 { _id: ObjectId("67e3a5908ea3f781676b140d"), name: "Laptop", price:
NumberInt(34300) },
 { _id: ObjectId("67e3bfc58ea3f781676b140f"), name: "Laptop", price:
NumberInt(34300) },
 { _id: ObjectId("67e3bfc58ea3f781676b1410"), name: "Laptop1", price:
NumberInt(34300) },
 { _id: ObjectId("67e3bff68ea3f781676b1411"), name: "Laptop1", price:
NumberInt(34300) }
]
```

MongoDB installation:
In Linux:

```
sudo apt install -y mongodb-org
```

After installation, start the MongoDB service:

```
sudo systemctl start mongod
```

To check mongodb running status

sudo systemctl status mongod

Version

mongod –version

Connect to shell

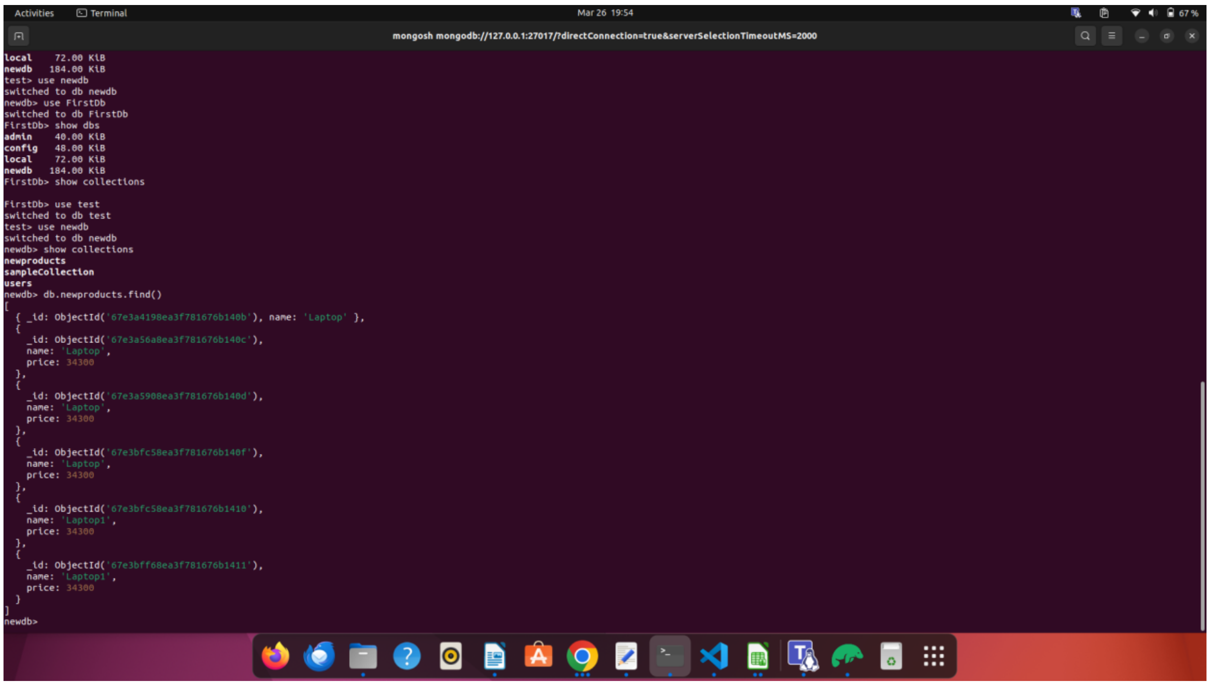
Mongosh

Restart the mongodb shell

sudo systemctl restart mongod

Commands;

Db.newproducts.find()- Retrieves all the data from the collection


Activities
Terminal
Mar 26 19:54
67 %
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

```
local     72.00 KiB
newdb    184.00 KiB
test> use newdb
switched to db newdb
newdb> use FirstDb
switched to db FirstDb
FirstDb> show dbs
admin     40.00 KiB
config    48.00 KiB
local     72.00 KiB
newdb    184.00 KiB
FirstDb> show collections

FirstDb> use test
switched to db test
test> use newdb
switched to db newdb
newdb> show collections
newproducts
sampleCollection
users
newdb> db.newproducts.find()
[
  { _id: ObjectId('67e3a4198ea3f781676b140b'), name: 'Laptop' },
  {
    _id: ObjectId('67e3a56a8ea3f781676b140c'),
    name: 'Laptop',
    price: 34300
  },
  {
    _id: ObjectId('67e3a5908ea3f781676b140d'),
    name: 'Laptop',
    price: 34300
  },
  {
    _id: ObjectId('67e3bfc58ea3f781676b140f'),
    name: 'Laptop',
    price: 34300
  },
  {
    _id: ObjectId('67e3bfc58ea3f781676b1410'),
    name: 'Laptop1',
    price: 34300
  },
  {
    _id: ObjectId('67e3bff68ea3f781676b1411'),
    name: 'Laptop1',
    price: 34300
  }
]
newdb>
```

How to Connect to MongoDB using a Connection String?

Local Connection

mongodb://localhost:27017

Remote Connection

mongodb://username:password@host:port/database

Example in Node.js:

```
const mongoose = require('mongoose');
mongoose.connect("mongodb://localhost:27017/newdb", { useNewUrlParser: true,
useUnifiedTopology: true})
  .then(() => console.log ("Connected to MongoDB"))
  .catch (err => console.log(err));
```

Database & Collection in MongoDB

Database in MongoDB

```
use  newdb;
```

Create collection

db.createCollection("users");

Inserting Data into a Collection (Creates Collection If Not Exists)
db.newproducts.insertOne({name: "Mobile", price:25,000.00});

MONGODB CRUD OPERATION

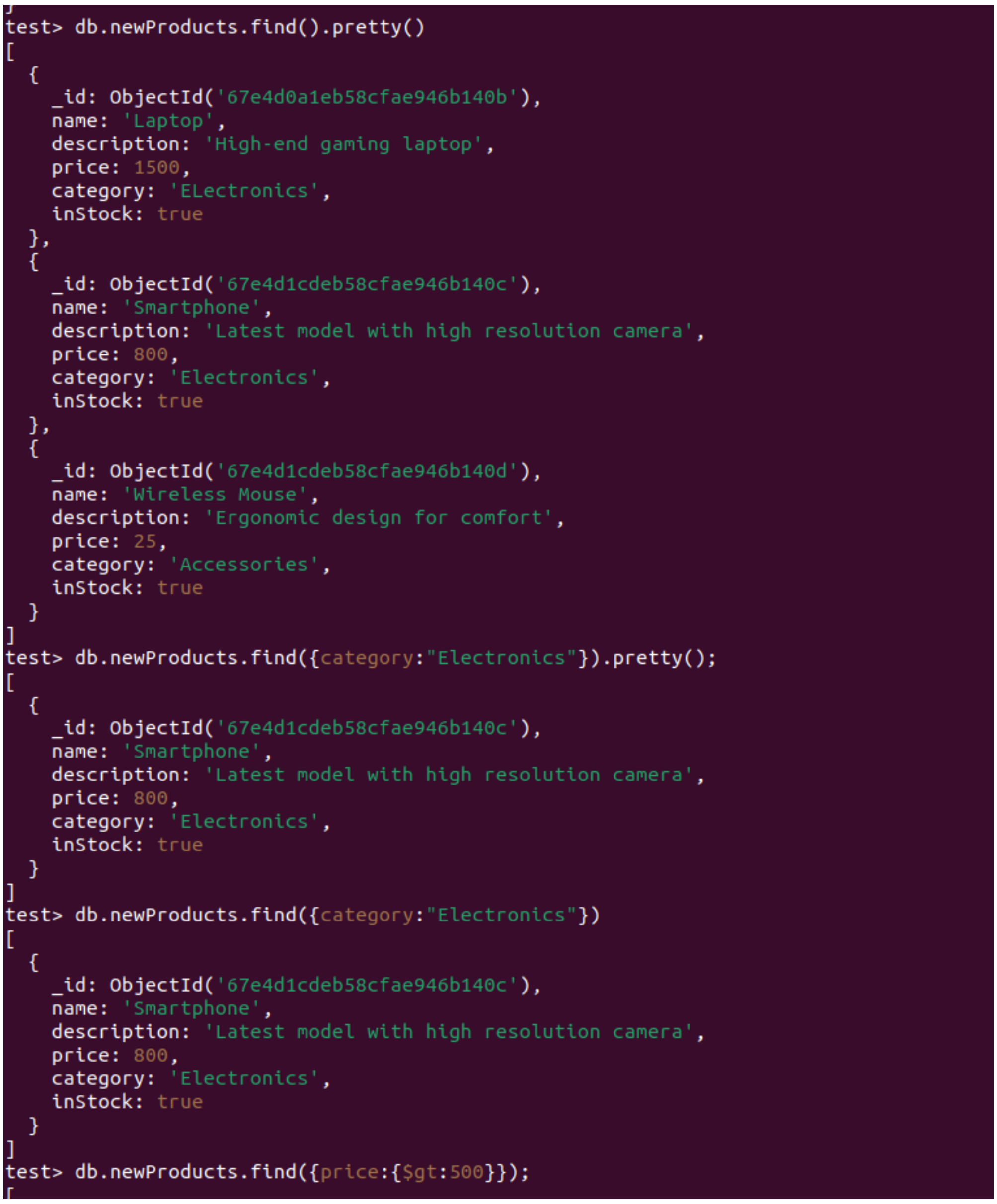
test> db.newProducts.find().pretty() - Find all documents

```
[
  {
    _id: ObjectId('67e4d0a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140d'),
    name: 'Wireless Mouse',
    description: 'Ergonomic design for comfort',
    price: 25,
    category: 'Accessories',
    inStock: true
  }
]
```

Pretty() in Mongodb:-

The. pretty() method in MongoDB is used to format query results in a more readable, structured JSON format.

```
test> db.newProducts.find({category:"Electronics"}).pretty();
[
 {
   _id: ObjectId('67e4d1cdeb58cfae946b140c'),
   name: 'Smartphone',
   description: 'Latest model with high resolution camera',
   price: 800,
   category: 'Electronics',
   inStock: true
 }
]
test> db.newProducts.find({category:"Electronics"}) - finds based on the category
[
 {
   _id: ObjectId('67e4d1cdeb58cfae946b140c'),
   name: 'Smartphone',
   description: 'Latest model with high resolution camera',
   price: 800,
   category: 'Electronics',
   inStock: true
 }
]
test> db.newProducts.find({price:{$gt:500}}) - Finds the document based on the
condition
[
 {
   _id: ObjectId('67e4d0a1eb58cfae946b140b'),
   name: 'Laptop',
   description: 'High-end gaming laptop',
   price: 1500,
   category: 'ELectronics',
   inStock: true
 },
 {
   _id: ObjectId('67e4d1cdeb58cfae946b140c'),
   name: 'Smartphone',
   description: 'Latest model with high resolution camera',
   price: 800,
   category: 'Electronics',
   inStock: true
 }
]
```

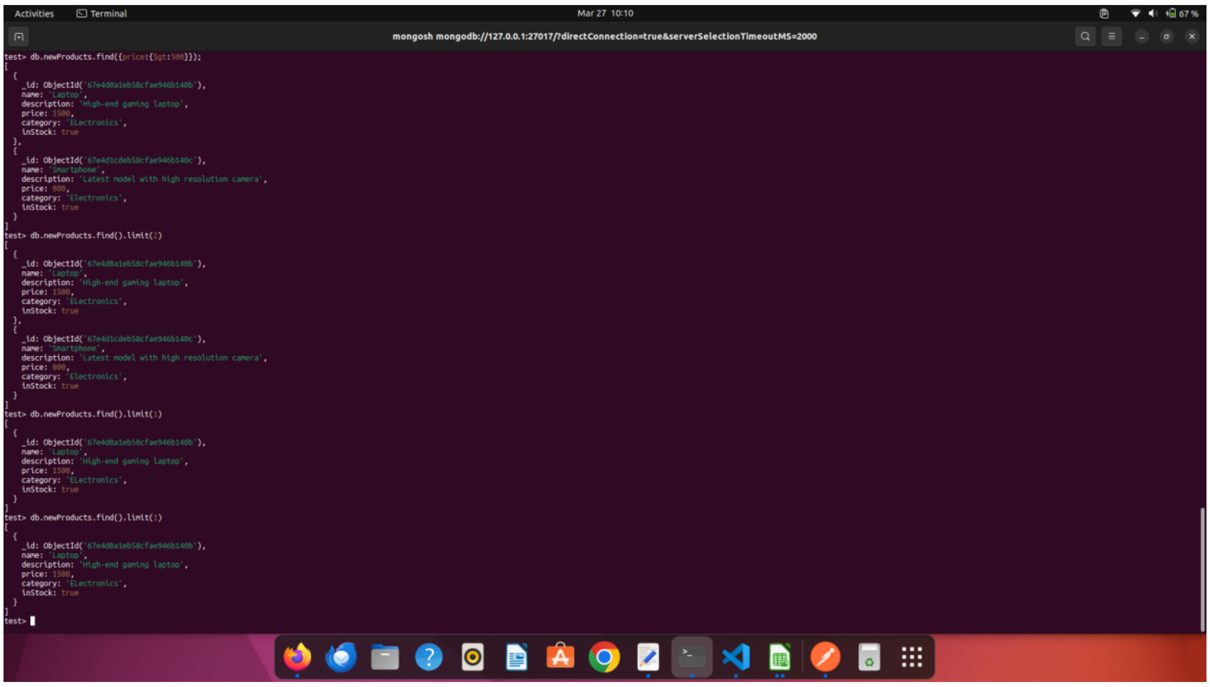


```
test> db.newProducts.find().pretty()
[
  {
    _id: ObjectId('67e4d0a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140d'),
    name: 'Wireless Mouse',
    description: 'Ergonomic design for comfort',
    price: 25,
    category: 'Accessories',
    inStock: true
  }
]
test> db.newProducts.find({category:"Electronics"}).pretty();
[
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  }
]
test> db.newProducts.find({category:"Electronics"})
[
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  }
]
test> db.newProducts.find({price:{$gt:500}});
```

27/03/2025

CRUD OPERATION: -

READ OPERATION: -



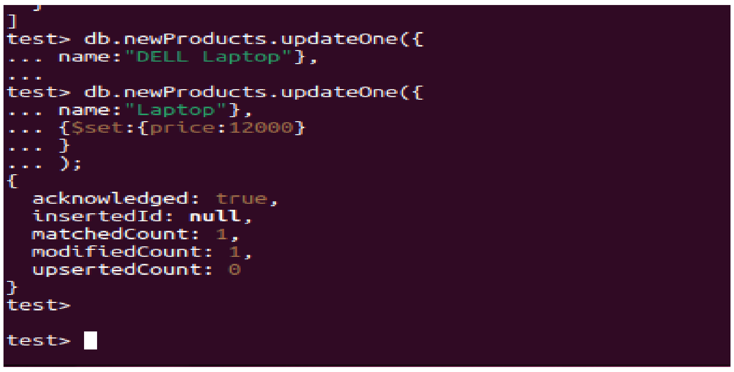
```
test> db.newProducts.find({price:{$gt:500}});
[
  {
    _id: ObjectId('67e4d8a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  }
]
test> db.newProducts.find().limit(2)
[
  {
    _id: ObjectId('67e4d8a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: true
  }
]
test> db.newProducts.find().limit(1)
[
  {
    _id: ObjectId('67e4d8a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  }
]
test> db.newProducts.find().limit(1)
[
  {
    _id: ObjectId('67e4d8a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 1500,
    category: 'ELectronics',
    inStock: true
  }
]
test> 
```

Activities
Terminal
Mar 27 10:10
67 %
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

3. Update Operations

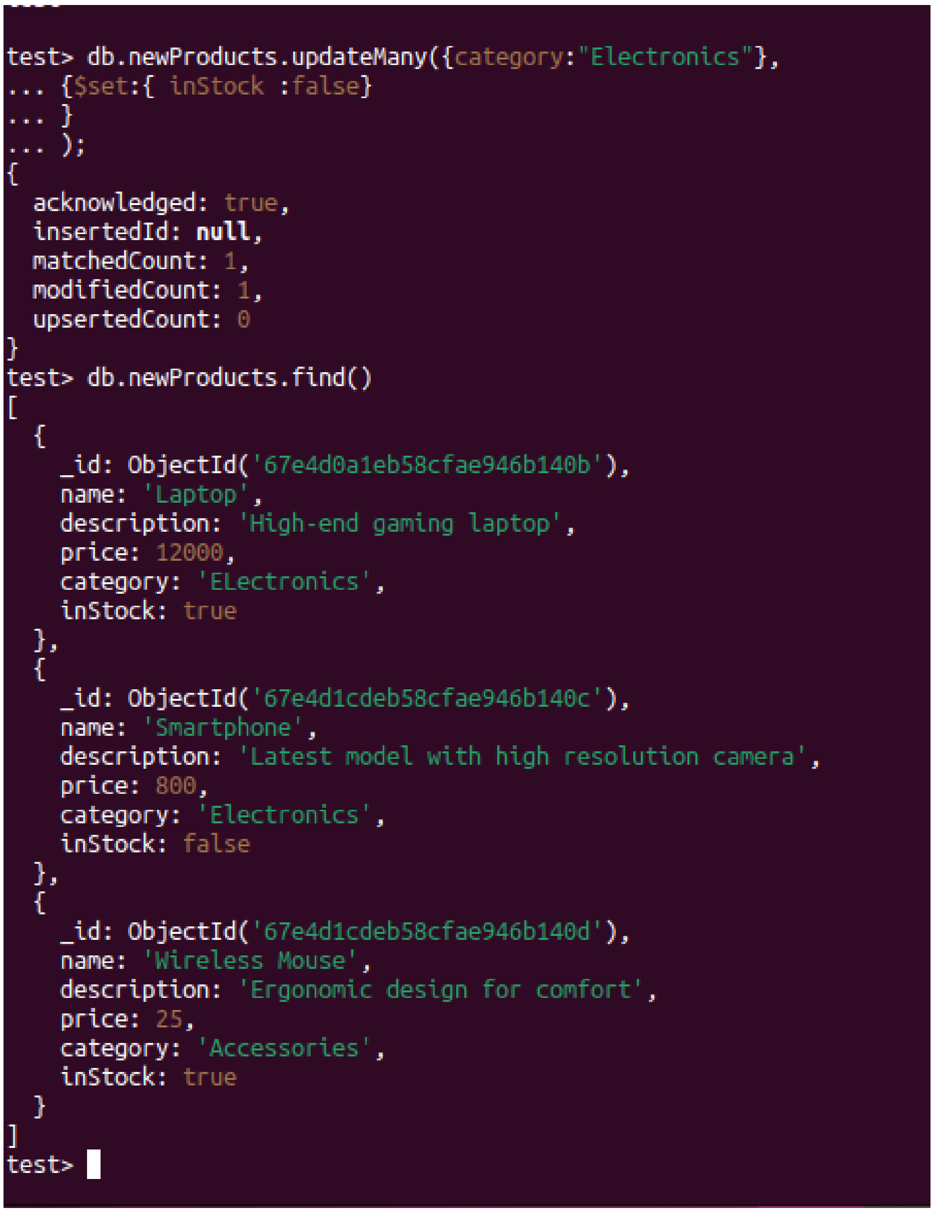
To update documents, we use the updateOne(), updateMany(), and replaceOne() methods.

UpdateOne()



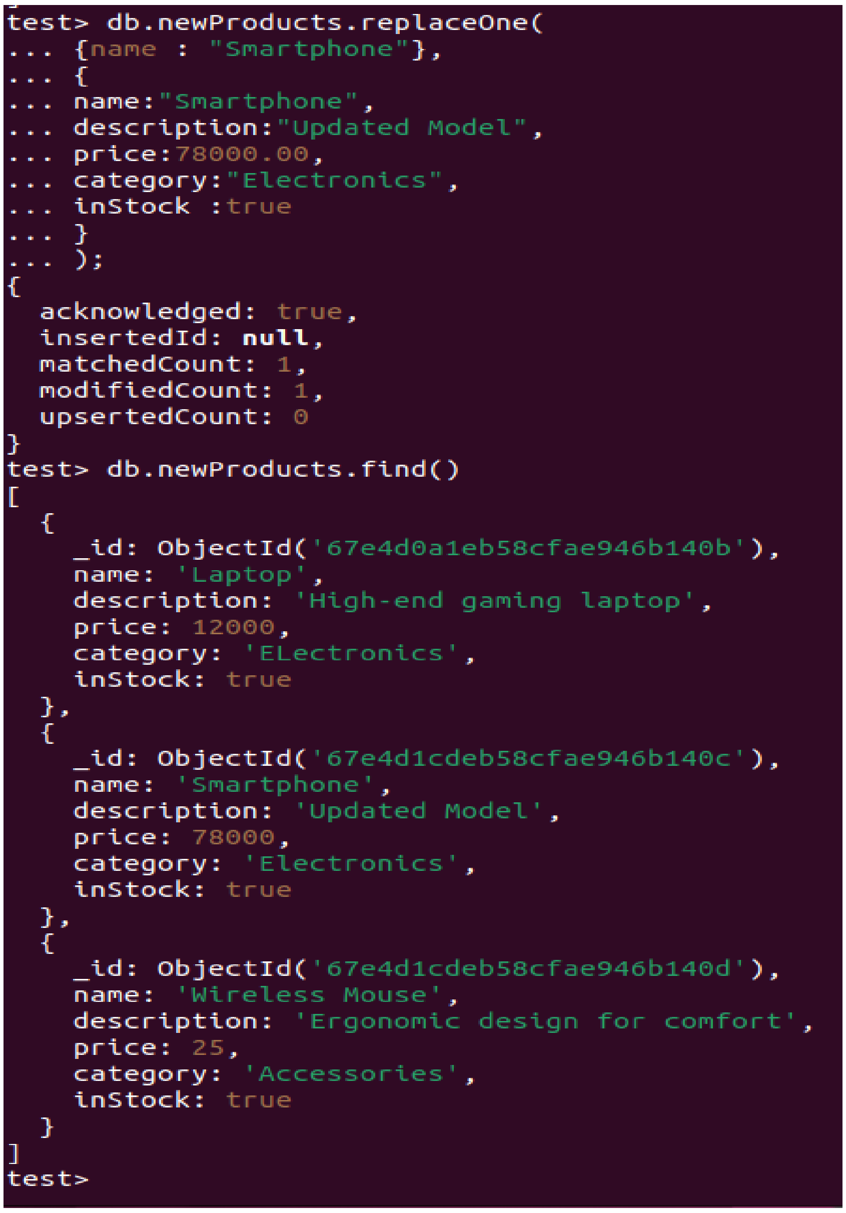
```
]
test> db.newProducts.updateOne({
... name:"DELL Laptop"},
...
test> db.newProducts.updateOne({
... name:"Laptop"},
... {$set:{price:12000}
... }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test>

test> 
```

updateMany()

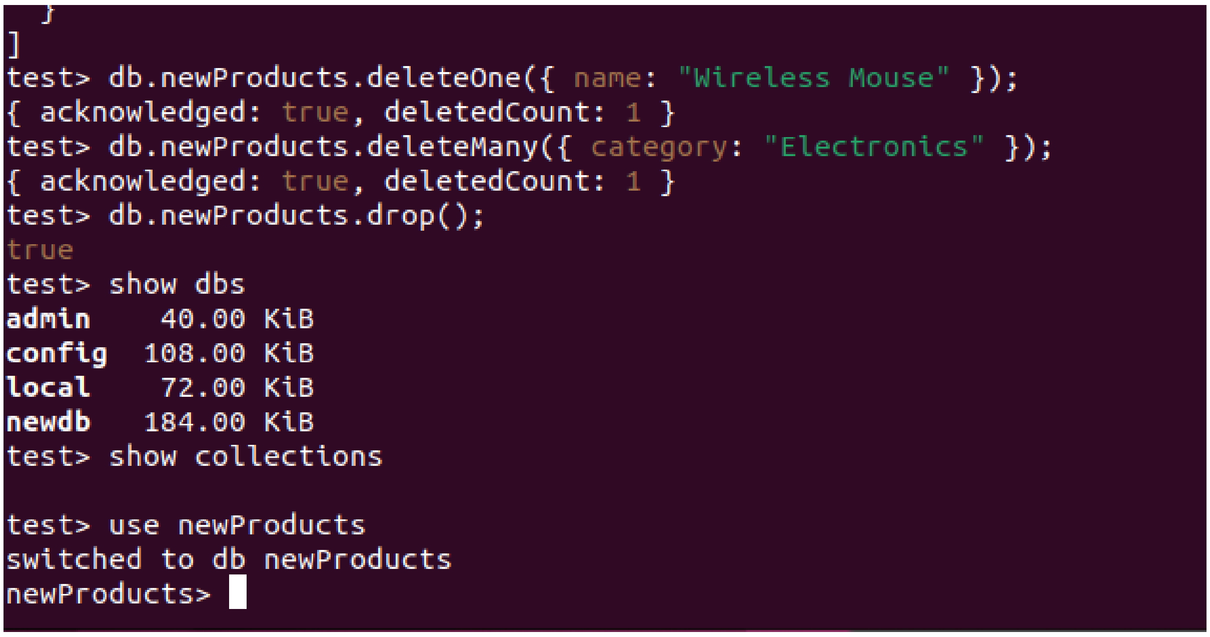


```
test> db.newProducts.updateMany({category:"Electronics"},
... {$set:{ inStock :false}
... }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> db.newProducts.find()
[
  {
    _id: ObjectId('67e4d0a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 12000,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Latest model with high resolution camera',
    price: 800,
    category: 'Electronics',
    inStock: false
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140d'),
    name: 'Wireless Mouse',
    description: 'Ergonomic design for comfort',
    price: 25,
    category: 'Accessories',
    inStock: true
  }
]
test> 
```

replaceOne()


```
test> db.newProducts.replaceOne(
... {name : "Smartphone"},
... {
... name:"Smartphone",
... description:"Updated Model",
... price:78000.00,
... category:"Electronics",
... inStock :true
... }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test> db.newProducts.find()
[
  {
    _id: ObjectId('67e4d0a1eb58cfae946b140b'),
    name: 'Laptop',
    description: 'High-end gaming laptop',
    price: 12000,
    category: 'ELectronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140c'),
    name: 'Smartphone',
    description: 'Updated Model',
    price: 78000,
    category: 'Electronics',
    inStock: true
  },
  {
    _id: ObjectId('67e4d1cdeb58cfae946b140d'),
    name: 'Wireless Mouse',
    description: 'Ergonomic design for comfort',
    price: 25,
    category: 'Accessories',
    inStock: true
  }
]
test>
```

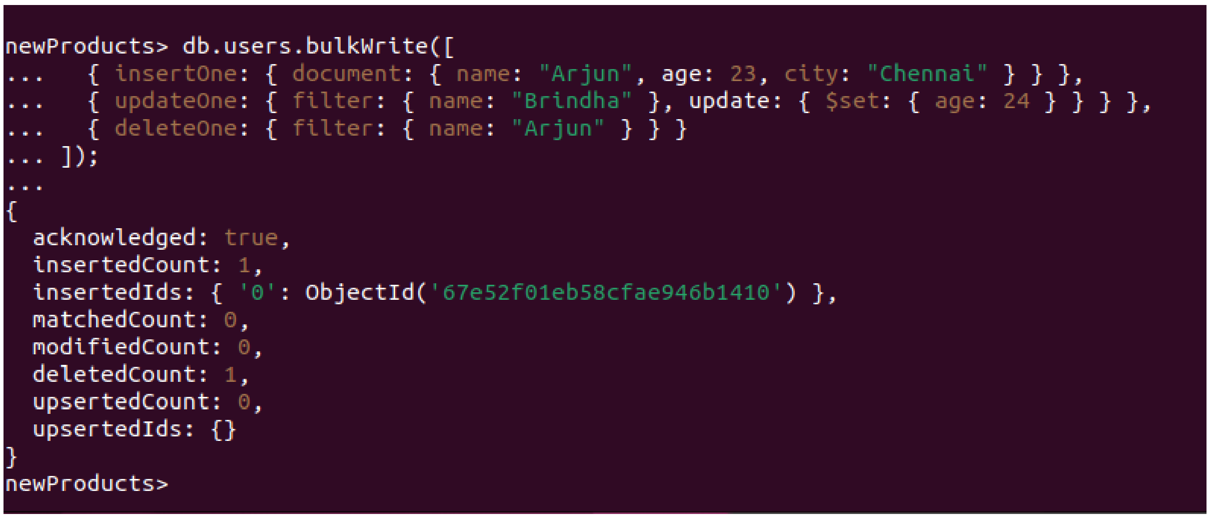
deleteOne(),deleteMany(),drop()



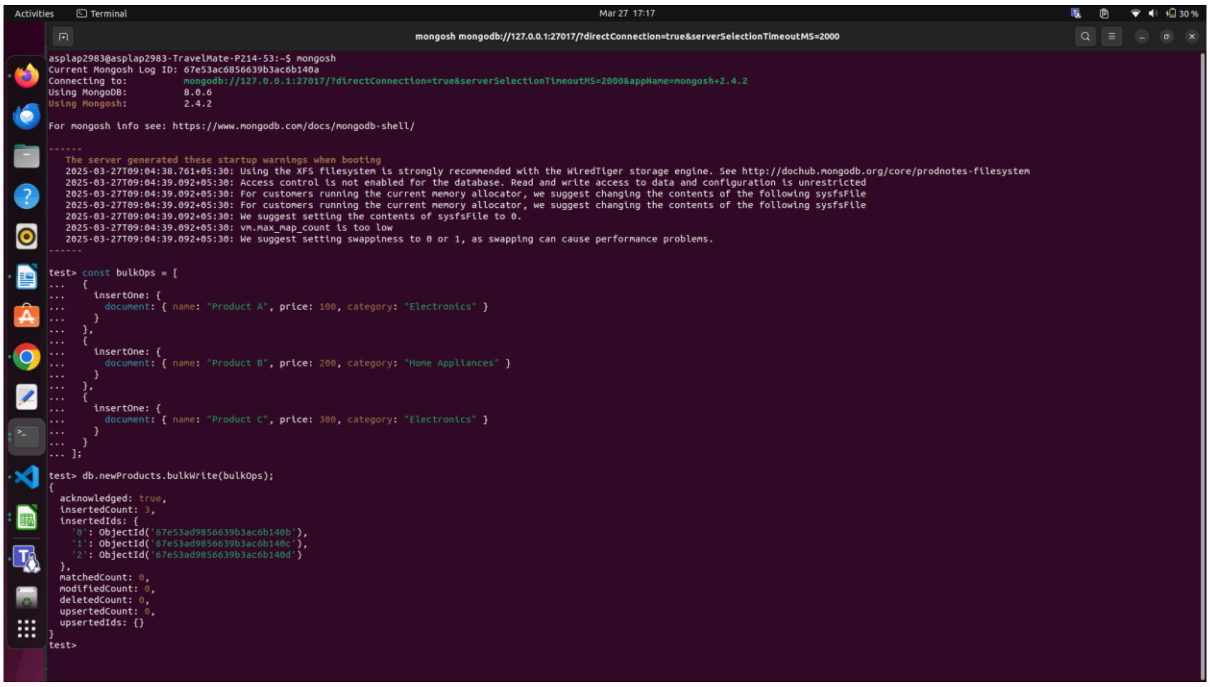
```
  }
]
test> db.newProducts.deleteOne({ name: "Wireless Mouse" });
{ acknowledged: true, deletedCount: 1 }
test> db.newProducts.deleteMany({ category: "Electronics" });
{ acknowledged: true, deletedCount: 1 }
test> db.newProducts.drop();
true
test> show dbs
admin     40.00 KiB
config   108.00 KiB
local     72.00 KiB
newdb    184.00 KiB
test> show collections

test> use newProducts
switched to db newProducts
newProducts> 
```

BulkWrite()

```
db.users.bulkWrite([
 { insertOne: {document: {name: "Arjun", age: 23, city: "Chennai"}}},
 { updateOne: {filter: {name: "Brindha"}, update: {$set: {age: 24}}}},
 { deleteOne: {filter: {name: "Arjun"}}}
]);
```



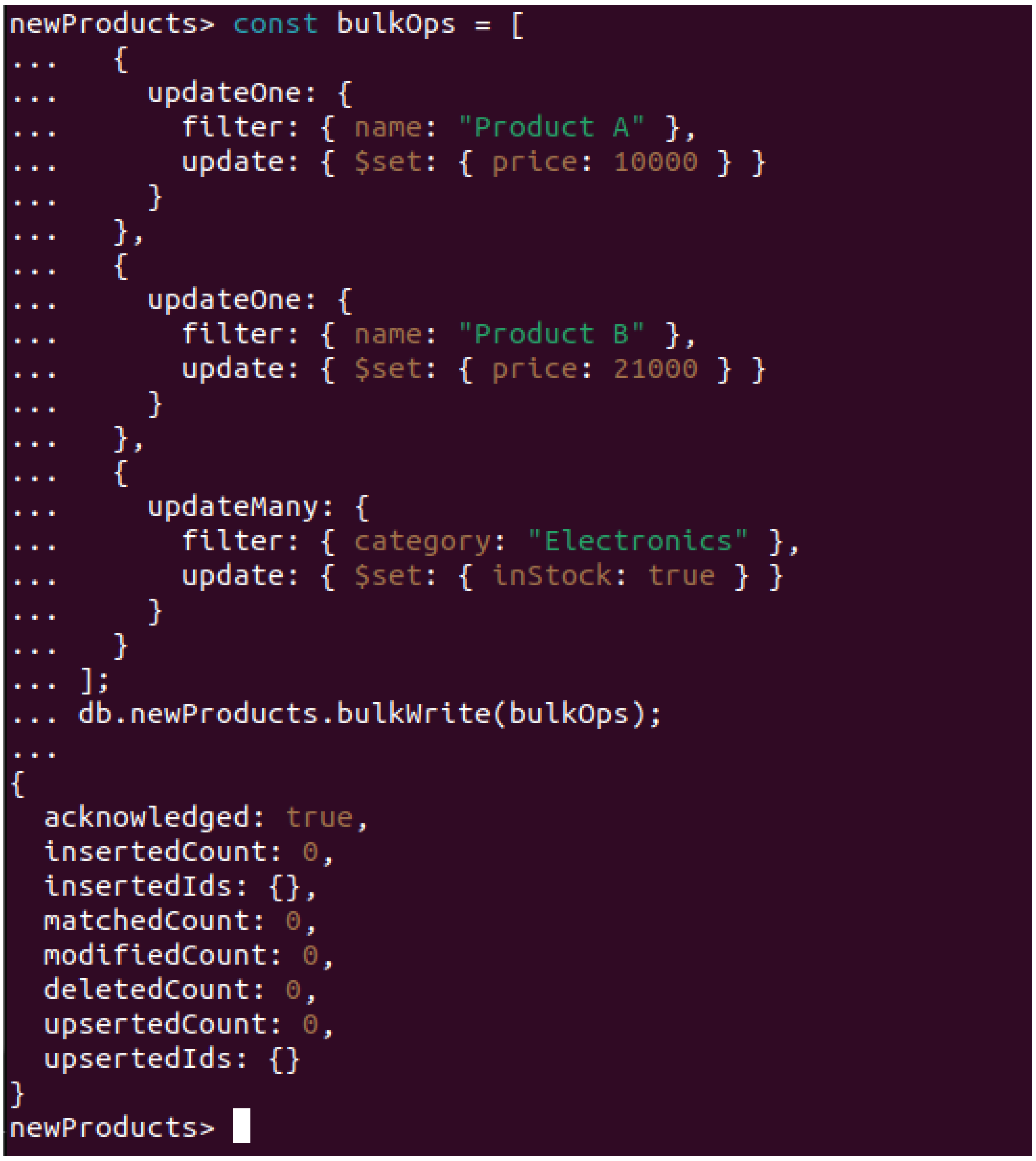
```
newProducts> db.users.bulkWrite([
...   { insertOne: { document: { name: "Arjun", age: 23, city: "Chennai" } } },
...   { updateOne: { filter: { name: "Brindha" }, update: { $set: { age: 24 } } } },
...   { deleteOne: { filter: { name: "Arjun" } } }
... ]);
...
{
  acknowledged: true,
  insertedCount: 1,
  insertedIds: { '0': ObjectId('67e52f01eb58cfae946b1410') },
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 1,
  upsertedCount: 0,
  upsertedIds: {}
}
newProducts>
```


Activities
Terminal
Mar 27 17:17
30 %
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

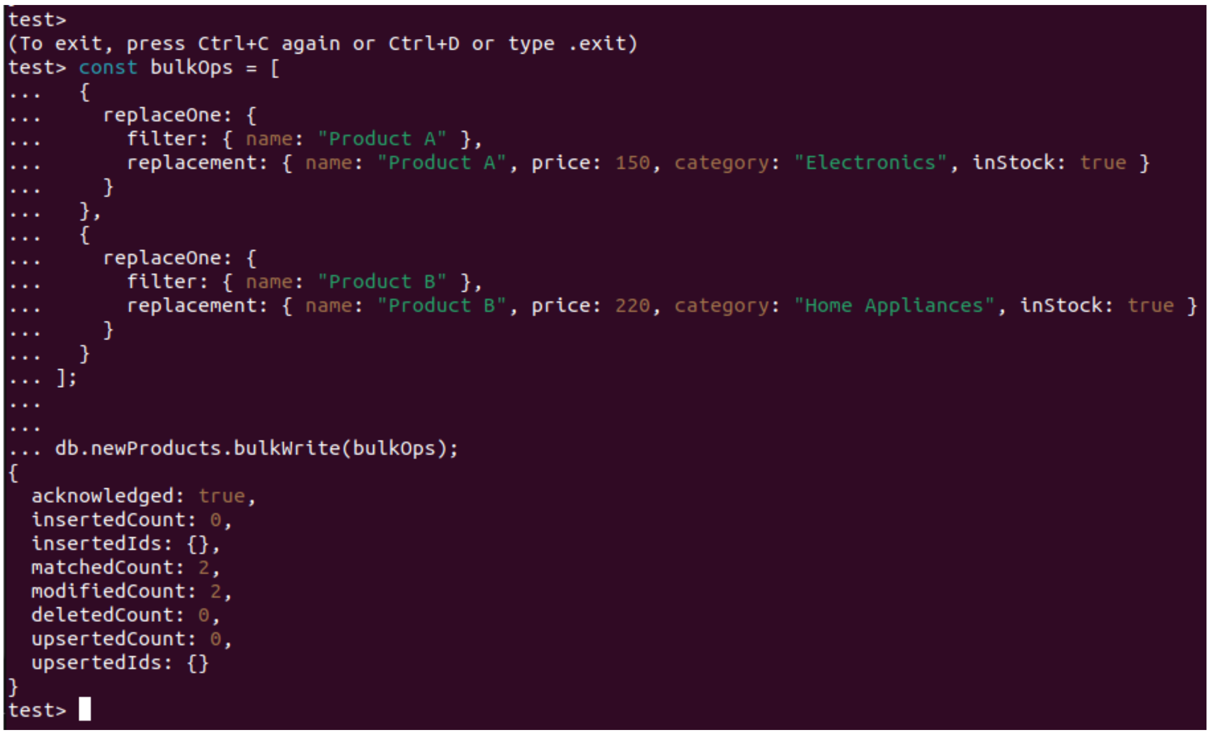
```
asplap2983@asplap2983-TravelMate-P214-53:~$ mongosh
Current Mongosh Log ID: 67e53ac6856639b3ac6b140a
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.4.2
Using MongoDB:          8.0.6
Using Mongosh:          2.4.2

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

------
   The server generated these startup warnings when booting
   2025-03-27T09:04:38.761+05:30: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2025-03-27T09:04:39.092+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2025-03-27T09:04:39.092+05:30: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
   2025-03-27T09:04:39.092+05:30: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
   2025-03-27T09:04:39.092+05:30: We suggest setting the contents of sysfsFile to 0.
   2025-03-27T09:04:39.092+05:30: vm.max_map_count is too low
   2025-03-27T09:04:39.092+05:30: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
------

test> const bulkOps = [
...   {
...     insertOne: {
...       document: { name: "Product A", price: 100, category: "Electronics" }
...     }
...   },
...   {
...     insertOne: {
...       document: { name: "Product B", price: 200, category: "Home Appliances" }
...     }
...   },
...   {
...     insertOne: {
...       document: { name: "Product C", price: 300, category: "Electronics" }
...     }
...   }
... ];

test> db.newProducts.bulkWrite(bulkOps);
{
  acknowledged: true,
  insertedCount: 3,
  insertedIds: {
    '0': ObjectId('67e53ad9856639b3ac6b140b'),
    '1': ObjectId('67e53ad9856639b3ac6b140c'),
    '2': ObjectId('67e53ad9856639b3ac6b140d')
  },
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {}
}
test>
```

```
const bulkOps = [
...   {
...     insertOne: {
...       document: {name: "Product A", price: 100, category: "Electronics" }
...     }
...   },
...   {
...     insertOne: {
...       document: {name: "Product B", price: 200, category: "Home Appliances" }
...     }
...   },
...   {
...     insertOne: {
...       document: {name: "Product C", price: 300, category: "Electronics" }
...     }
...   }
... ];

test> db.newProducts.bulkWrite(bulkOps);
{
  acknowledged: true,
  insertedCount: 3,
  insertedIds: {
    '0': ObjectId('67e53ad9856639b3ac6b140b'),
    '1': ObjectId('67e53ad9856639b3ac6b140c'),
    '2': ObjectId('67e53ad9856639b3ac6b140d')
```

```
 },
 matchedCount: 0,
 modifiedCount: 0,
 deletedCount: 0,
 upsertedCount: 0,
 upsertedIds: {}
}
```



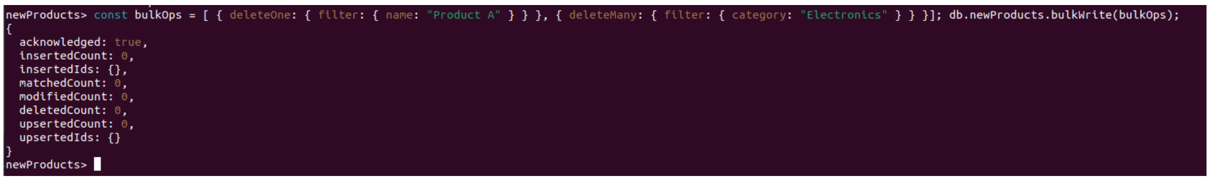
```
newProducts> const bulkOps = [
...   {
...     updateOne: {
...       filter: { name: "Product A" },
...       update: { $set: { price: 10000 } }
...     }
...   },
...   {
...     updateOne: {
...       filter: { name: "Product B" },
...       update: { $set: { price: 21000 } }
...     }
...   },
...   {
...     updateMany: {
...       filter: { category: "Electronics" },
...       update: { $set: { inStock: true } }
...     }
...   }
... ];
... db.newProducts.bulkWrite(bulkOps);
...
{
  acknowledged: true,
  insertedCount: 0,
  insertedIds: {},
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {}
}
newProducts> 
```



```
test>
(To exit, press Ctrl+C again or Ctrl+D or type .exit)
test> const bulkOps = [
...   {
...     replaceOne: {
...       filter: { name: "Product A" },
...       replacement: { name: "Product A", price: 150, category: "Electronics", inStock: true }
...     }
...   },
...   {
...     replaceOne: {
...       filter: { name: "Product B" },
...       replacement: { name: "Product B", price: 220, category: "Home Appliances", inStock: true }
...     }
...   }
... ];
...
...
... db.newProducts.bulkWrite(bulkOps);
{
  acknowledged: true,
  insertedCount: 0,
  insertedIds: {},
  matchedCount: 2,
  modifiedCount: 2,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {}
}
test> 
```

replaceOne()
newProducts>

```
... const bulkOps = [
...   {
...     replaceOne: {
...       filter: {name: "Product A"},
...       replacement: {name: "Product A", price: 15000, category: "Electronics", inStock: true}
...     }
...   },
...   {
...     replaceOne: {
...       filter: {name: "Product B"},
...       replacement: {name: "Product B", price: 22000, category: "Home Appliances", inStock: true}
...     }
...   }
... ];
...
...
... db.newProducts.bulkWrite(bulkOps);
{
  acknowledged: true,
  insertedCount: 0,
  insertedIds: {},
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {}
}
```



```
newProducts> const bulkOps = [ { deleteOne: { filter: { name: "Product A" } } }, { deleteMany: { filter: { category: "Electronics" } } }]; db.newProducts.bulkWrite(bulkOps);
{
  acknowledged: true,
  insertedCount: 0,
  insertedIds: {},
  matchedCount: 0,
  modifiedCount: 0,
  deletedCount: 0,
  upsertedCount: 0,
  upsertedIds: {}
}
newProducts> 
```

Aggregation Pipelines

Certainly! Below is a list of each MongoDB Aggregation Pipeline stage with its one-line definition:

Aggregation Pipeline STAGES

1. **$match**:

   Filters documents based on specified conditions (similar to find queries).

2. **$group**:

Groups documents by a specified field and computes aggregate values (sum, count, average, etc.).

3. **$sort**:

   Sorts the documents based on specified fields in ascending or descending order.

4. **$project**:

   Reshapes each document by including, excluding, or transforming fields.

5. **$limit**:

   Limits the number of documents passed through the pipeline.

6. **$skip**:
   Skips a specified number of documents before passing the remaining documents.
7. **$unwind**:
   Deconstructs an array field to output a document for each element in the array.
8. **$lookup**:
   Joins documents from another collection based on a common field.
9. **$addFields**:
   Adds new fields or modifies existing fields in the documents.
10. **$count**:
    Counts the number of documents passing through the pipeline and outputs a count field.

    $facet:
    Allows multiple aggregation pipelines to run in parallel and outputs the results as different fields.
    $sample:
    a. Randomly selects a specified number of documents from the collection.
11. **$merge**:
    a. Writes the results of the aggregation pipeline to a specified collection.
12. **$replaceRoot**:
    a. Replaces the root document with a new document or sub-document.

1. $match

Definition: Filters documents based on specified conditions (similar to find queries).

Example:

```
db.products.aggregate([
 { $match: { category: "Electronics" } }
]);
```

2. $group

Definition: Groups documents by a specified field and computes aggregate values (sum, count, average, etc.).

Example:

```
db.products.aggregate([
 {
   $group: {
     _id: "$category",
     totalPrice: {$sum: "$price" },
     count: {$sum: 1 }
   }
 }
]);
```

3. $sort

Definition: Sorts the documents based on specified fields in ascending or descending order.

Example:

```
db.products.aggregate([
 { $sort: { price: -1 } }
]);
```

4. $project

Definition: Reshapes each document by including, excluding, or transforming fields.

Example:

```
db.products.aggregate([
 {
   $project: {
     name: 1,
     price: 1,
     discountPrice: { $multiply: ["$price", 0.9] }
   }
 }
]);
```

5. $limit

Definition: Limits the number of documents passed through the pipeline.

Example:

```
db.products.aggregate([
 { $limit: 5 }
]);
```

6. $skip

Definition: Skips a specified number of documents before passing the remaining documents.

Example:

```
db.products.aggregate([
 { $skip: 3 }
]);
```

7. $unwind

Definition: Deconstructs an array field to output a document for each element in the array.

Example:

```
db.orders.aggregate([
 { $unwind: "$items" }
]);
```

8. $lookup

Definition: Joins documents from another collection based on a common field.

Example:

```
db.orders.aggregate([
 {
  $lookup: {
   from: "products",
   localField: "productId",
   foreignField: "_id",
   as: "productDetails"
  }
 }
]);
```

9. $addFields

Definition: Adds new fields or modifies existing fields in the documents.

Example:

```
db.products.aggregate([
 {
  $addFields: {
      salePrice: { $multiply: ["$price", 0.8] }
    }
  }
]);
```

10. $count

Definition: Counts the number of documents passing through the pipeline and outputs a count field.

Example:

```
db.products.aggregate([
 { $count: "totalProducts" }
]);
```

11. $facet

Allows multiple aggregation pipelines to run in parallel and outputs the results as different fields.

Example:

```
db.orders.aggregate([
 {
   $facet: {
     totalSales: [
       { $group: { _id: null, totalAmount: {$sum: "$total" } } }
     ],
     totalOrders: [
       { $count: "total" }
     ]
   }
```

13. $sample

Definition: Randomly selects a specified number of documents from the collection.

Example:

```
db.products.aggregate([
 { $sample: { size: 3 } }  // Randomly select 3 products
]);
```

14. $merge

Definition: Writes the results of the aggregation pipeline to a specified collection.

Example:

```
db.products.aggregate([
 {
   $group: {
     _id: "$category",
     totalPrice: { $sum: "$price" }
   }
 },
 { $merge: { into: "aggregated_results" } }
]);
```

15. $replaceRoot

Definition: Replaces the root document with a new document or sub-document.

Example:

```
db.orders.aggregate([
 {
   $replaceRoot: { newRoot: "$productDetails" }
 }
]);
```

DAY – 03

1. Selecting the Database

```
use newProduts
db.createCollection("newProducts")
```

3. Inserting Data into the Collection

```
db.newProducts.insertOne({
  name: "Smartphone",
  brand: "BrandA",
  price: 399.99,
  inStock: true,
  releaseDate: new Date("2025-05-01")
})
```

Insert Multiple Documents:

```
db.newProducts.insertMany([
  { name: "Laptop", brand: "BrandB", price: 899.99, inStock: true, releaseDate: new
Date("2025-06-01") },
  { name: "Smartwatch", brand: "BrandC", price: 149.99, inStock: false, releaseDate:
new Date("2025-04-15") },
  { name: "Tablet", brand: "BrandD", price: 299.99, inStock: true, releaseDate: new
Date("2025-07-10") }
])
```

4. Querying the Data

Find All Documents:

You are using the following command to find all documents:

```
db.newProducts.find()
```

Find All with Pretty Formatting:

```
db.newProducts.find(). pretty()
```

Find Documents with a Specific Condition:

```
db.newProducts.find({ inStock: true }).pretty()
```

Find a Specific Product by Name:

```
db.newProducts.find({ name: "Smartphone" }).pretty()
```

Find Products by Price Range:

```
db.newProducts.find({ price: { $gt: 200 } }).pretty()
```

5. Updating Documents

You can update documents using updateOne(), updateMany(), or replaceOne().

Update a Single Document

‘

```
db.newProducts.updateOne(
 { name: "Smartphone" },   // Filter
 { $set: { price: 379.99 } }  // Update operation
)
```

Update Multiple Documents:

```
db.newProducts.updateMany(
 { price: { $gt: 500 } },  // Filter
```

```
  { $set: { inStock: false } }  // Update operation
)
```

6. Deleting Documents

Delete a Single Document:

For example, delete the "Smartwatch" product:

```
db.newProducts.deleteOne({ name: "Smartwatch" })
```

Delete Multiple Documents:

For example, delete all products that are not in stock:

```
db.newProducts.deleteMany({ inStock: false })
```

7. Indexing

Indexes improve the performance of queries.

Create an Index on the name Field:

```
db.newProducts.createIndex({ name: 1 })
```

This will create an ascending index on the name field.

Create a Compound Index on price and brand:

```
db.newProducts.createIndex({ price: 1, brand: 1 })
```

8. Aggregation Pipeline

MongoDB’s aggregation framework allows you to perform complex queries and data transformations. Here's an example of how you can use it.

```
db.newProducts.aggregate([
 { $match: { inStock: true } },
 { $group: { _id: null, avgPrice: { $avg: "$price" } } }
])
```

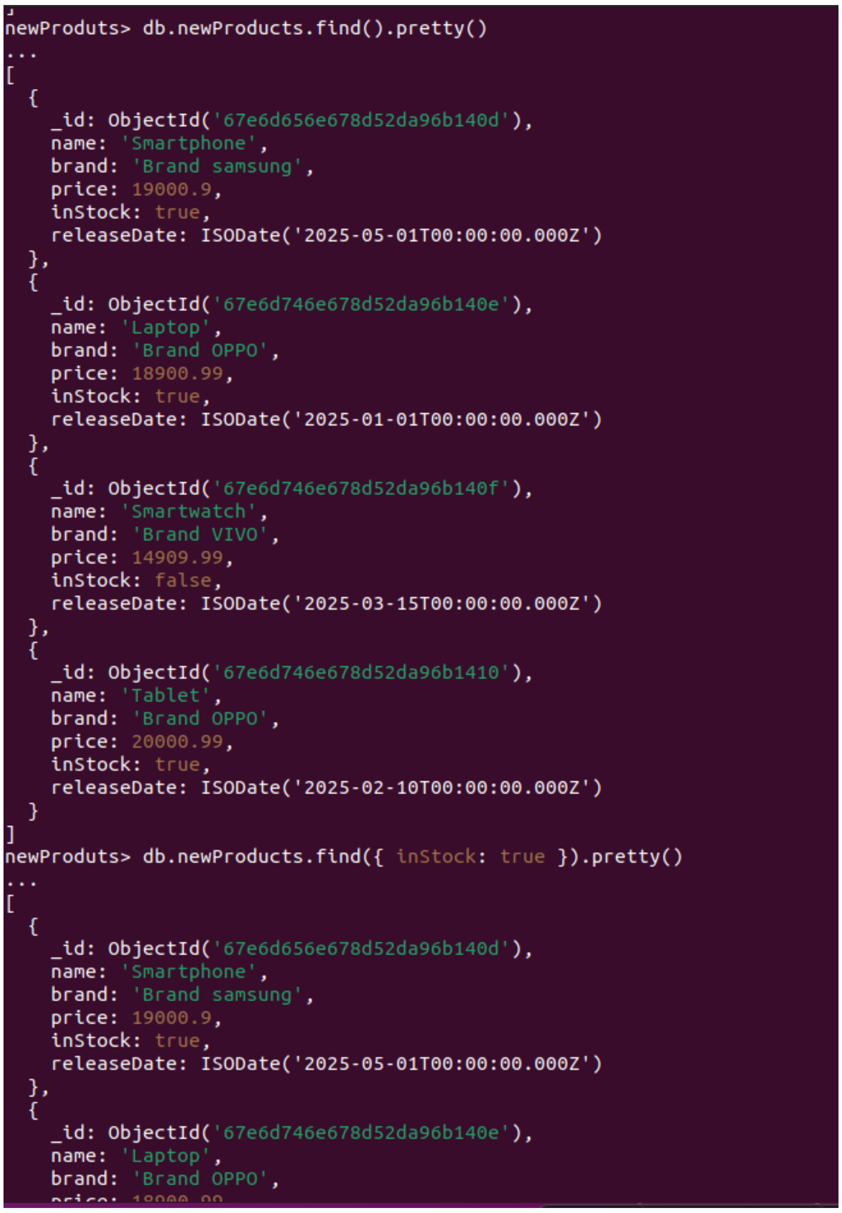
```
db.newProducts.aggregate([
 { $group: { _id: "$brand", productCount: { $sum: 1 } } }

)
```

9. Data Validation (Defining Schema with Validation)

While MongoDB doesn't require a strict schema, you can set up schema validation to enforce certain rules.

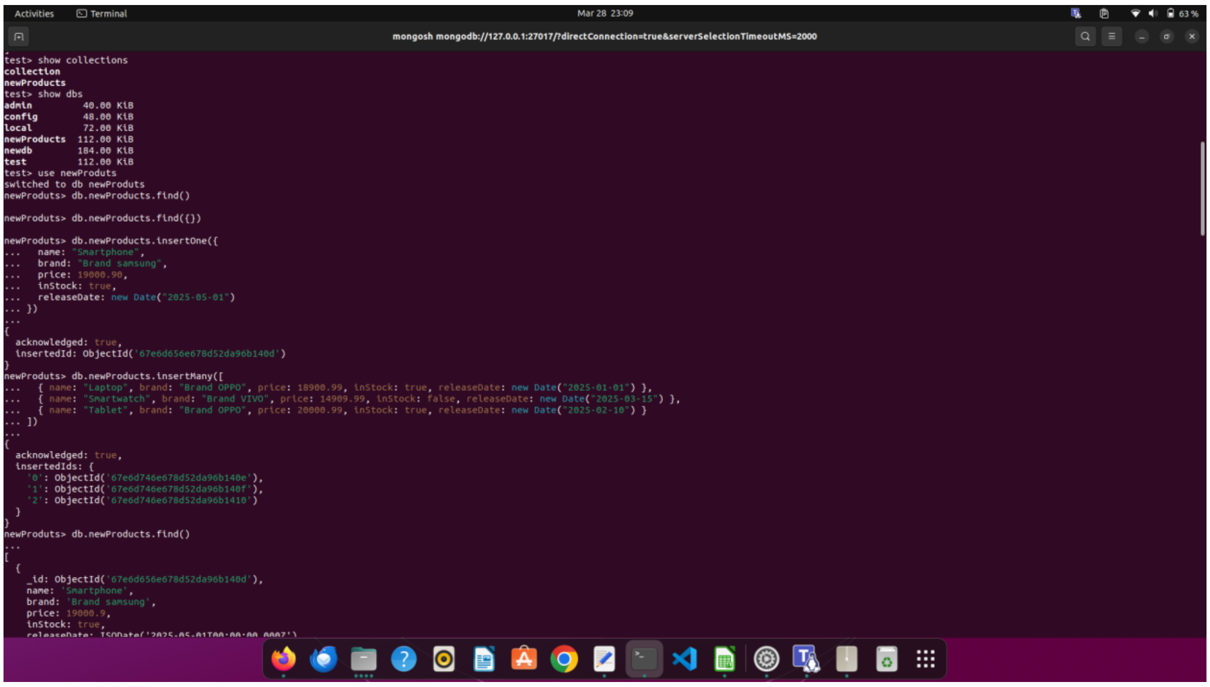
Define Validation for newProducts Collection:

For example, you can require that each product document should have a name (string), price (number), and inStock (boolean).



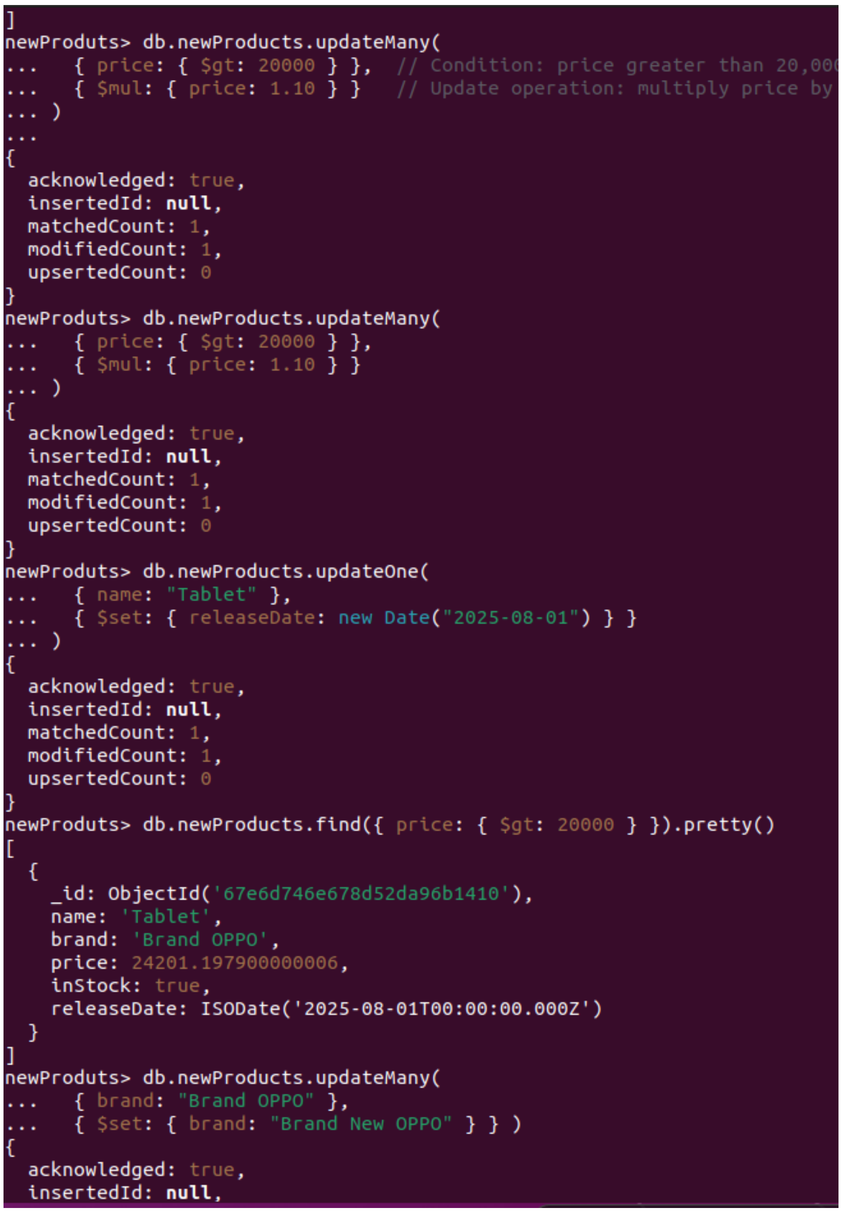
```
newProduts> db.newProducts.find().pretty()
...
[
  {
    _id: ObjectId('67e6d656e678d52da96b140d'),
    name: 'Smartphone',
    brand: 'Brand samsung',
    price: 19000.9,
    inStock: true,
    releaseDate: ISODate('2025-05-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140e'),
    name: 'Laptop',
    brand: 'Brand OPPO',
    price: 18900.99,
    inStock: true,
    releaseDate: ISODate('2025-01-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140f'),
    name: 'Smartwatch',
    brand: 'Brand VIVO',
    price: 14909.99,
    inStock: false,
    releaseDate: ISODate('2025-03-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b1410'),
    name: 'Tablet',
    brand: 'Brand OPPO',
    price: 20000.99,
    inStock: true,
    releaseDate: ISODate('2025-02-10T00:00:00.000Z')
  }
]
newProduts> db.newProducts.find({ inStock: true }).pretty()
...
[
  {
    _id: ObjectId('67e6d656e678d52da96b140d'),
    name: 'Smartphone',
    brand: 'Brand samsung',
    price: 19000.9,
    inStock: true,
    releaseDate: ISODate('2025-05-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140e'),
    name: 'Laptop',
    brand: 'Brand OPPO',
```

```
db.createCollection("newProducts", {
 validator: {
  $jsonSchema: {
   bsonType: "object",
   required: ["name", "price", "inStock"],
   properties: {
    name: { bsonType: "string" },
    price: { bsonType: "double" },
    inStock: { bsonType: "bool" },
    releaseDate: { bsonType: "date" }
   }
  }
 }
})
```

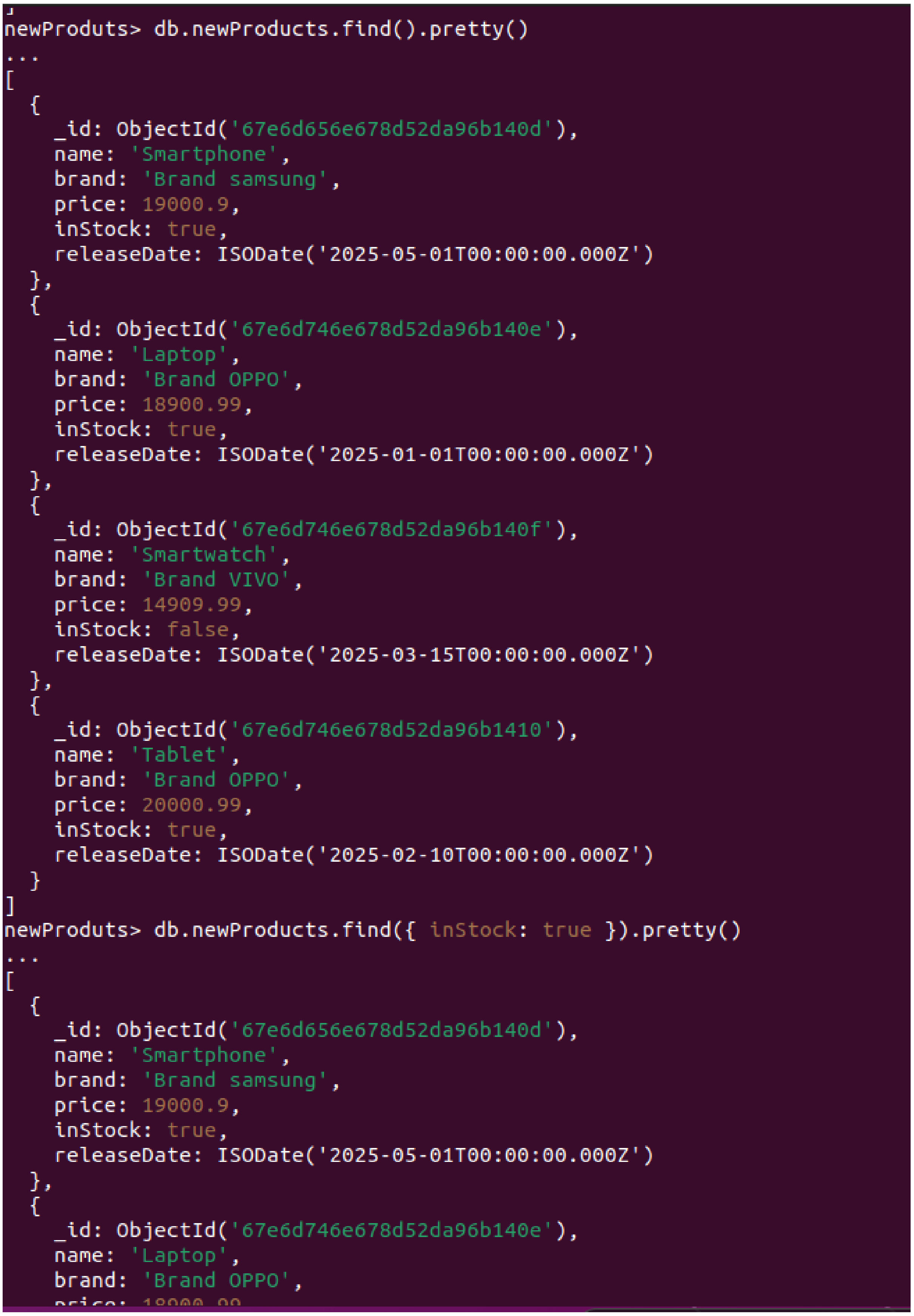


```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
```

```
test> show collections
collection
newProducts
test> show dbs
admin         40.00 KiB
config        48.00 KiB
local         72.00 KiB
newProducts  112.00 KiB
newdb        184.00 KiB
test         112.00 KiB
test> use newProduts
switched to db newProduts
newProduts> db.newProducts.find()

newProduts> db.newProducts.find({})

newProduts> db.newProducts.insertOne({
...   name: "Smartphone",
...   brand: "Brand samsung",
...   price: 19000.90,
...   inStock: true,
...   releaseDate: new Date("2025-05-01")
... })
...
{
  acknowledged: true,
  insertedId: ObjectId('67e6d656e678d52da96b140d')
}
newProduts> db.newProducts.insertMany([
...   { name: "Laptop", brand: "Brand OPPO", price: 18900.99, inStock: true, releaseDate: new Date("2025-01-01") },
...   { name: "Smartwatch", brand: "Brand VIVO", price: 14909.99, inStock: false, releaseDate: new Date("2025-03-15") },
...   { name: "Tablet", brand: "Brand OPPO", price: 20000.99, inStock: true, releaseDate: new Date("2025-02-10") }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67e6d746e678d52da96b140e'),
    '1': ObjectId('67e6d746e678d52da96b140f'),
    '2': ObjectId('67e6d746e678d52da96b1410')
  }
}
newProduts> db.newProducts.find()
...
[
  {
    _id: ObjectId('67e6d656e678d52da96b140d'),
    name: 'Smartphone',
    brand: 'Brand samsung',
    price: 19000.9,
    inStock: true,
    releaseDate: ISODate('2025-05-01T00:00:00.000Z')
```



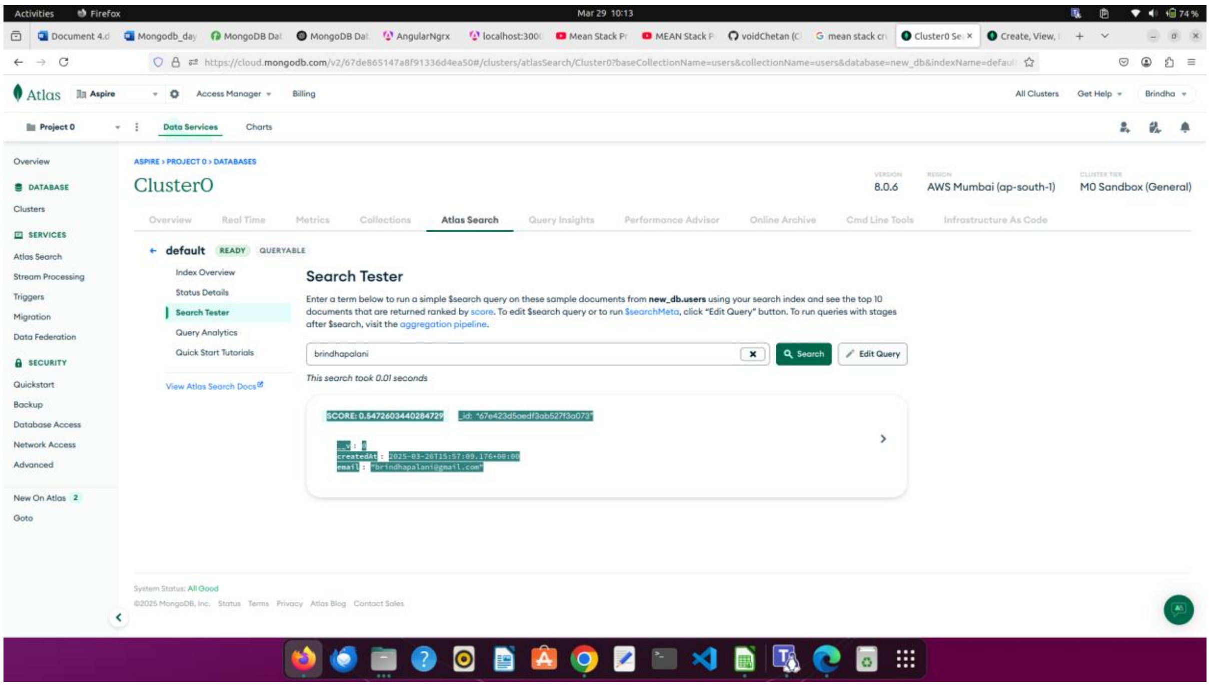
```
]
newProduts> db.newProducts.updateMany(
...   { price: { $gt: 20000 } },   // Condition: price greater than 20,00
...   { $mul: { price: 1.10 } }   // Update operation: multiply price by
... )
...
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
newProduts> db.newProducts.updateMany(
...   { price: { $gt: 20000 } },
...   { $mul: { price: 1.10 } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
newProduts> db.newProducts.updateOne(
...   { name: "Tablet" },
...   { $set: { releaseDate: new Date("2025-08-01") } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
newProduts> db.newProducts.find({ price: { $gt: 20000 } }).pretty()
[
  {
    _id: ObjectId('67e6d746e678d52da96b1410'),
    name: 'Tablet',
    brand: 'Brand OPPO',
    price: 24201.197900000006,
    inStock: true,
    releaseDate: ISODate('2025-08-01T00:00:00.000Z')
  }
]
newProduts> db.newProducts.updateMany(
...   { brand: "Brand OPPO" },
...   { $set: { brand: "Brand New OPPO" } } )
{
  acknowledged: true,
  insertedId: null,
```



```
newProduts> db.newProducts.find().pretty()
...
[
  {
    _id: ObjectId('67e6d656e678d52da96b140d'),
    name: 'Smartphone',
    brand: 'Brand samsung',
    price: 19000.9,
    inStock: true,
    releaseDate: ISODate('2025-05-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140e'),
    name: 'Laptop',
    brand: 'Brand OPPO',
    price: 18900.99,
    inStock: true,
    releaseDate: ISODate('2025-01-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140f'),
    name: 'Smartwatch',
    brand: 'Brand VIVO',
    price: 14909.99,
    inStock: false,
    releaseDate: ISODate('2025-03-15T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b1410'),
    name: 'Tablet',
    brand: 'Brand OPPO',
    price: 20000.99,
    inStock: true,
    releaseDate: ISODate('2025-02-10T00:00:00.000Z')
  }
]
newProduts> db.newProducts.find({ inStock: true }).pretty()
...
[
  {
    _id: ObjectId('67e6d656e678d52da96b140d'),
    name: 'Smartphone',
    brand: 'Brand samsung',
    price: 19000.9,
    inStock: true,
    releaseDate: ISODate('2025-05-01T00:00:00.000Z')
  },
  {
    _id: ObjectId('67e6d746e678d52da96b140e'),
    name: 'Laptop',
    brand: 'Brand OPPO',
```

Advanced

Indexing in users database in mongodb atlas



Enter a term below to run a simple $search query on these sample documents from **new_db.users** using your search index and see the top 10 documents that are returned ranked by score. To edit $search query or to run $searchMeta, click "Edit Query" button. To run queries with stages after $search, visit the aggregation pipeline.

brindhapalani

This search took 0.01 seconds

```
SCORE: 0.5472603440284729   _id: "67e423d5aedf3ab527f3a073"

__v : 0
createdAt : 2025-03-26T15:57:09.176+00:00
email : "brindhapalani@gmail.com"
```

use companyDB

db.employees.getIndexes()

```
db.employees.insertMany([
  { name: "Amit", age: 30, city: "Mumbai", email: "amit@example.com", position: "Manager" },
  { name: "Priya", age: 25, city: "Delhi", email: "priya@example.com", position: "Software Engineer" },
  { name: "Rahul", age: 35, city: "Bangalore", email: "rahul@example.com", position: "Data Scientist" },
  { name: "Sneha", age: 28, city: "Chennai", email: "sneha@example.com", position: "HR Executive" },
  { name: "Vikram", age: 40, city: "Hyderabad", email: "vikram@example.com", position: "Product Manager" },
  { name: "Anjali", age: 32, city: "Kolkata", email: "anjali@example.com", position: "Marketing Head" }
])

db.employees.insertMany([
  { name: "Amit", age: 30, city: "Mumbai", email: "amit@example.com", position: "Manager" },
  { name: "Priya", age: 25, city: "Delhi", email: "priya@example.com", position: "Software Engineer" },
  { name: "Rahul", age: 35, city: "Bangalore", email: "rahul@example.com", position: "Data Scientist" },
  { name: "Sneha", age: 28, city: "Chennai", email: "sneha@example.com", position: "HR Executive" },
  { name: "Vikram", age: 40, city: "Hyderabad", email: "vikram@example.com", position: "Product Manager" },
  { name: "Anjali", age: 32, city: "Kolkata", email: "anjali@example.com", position: "Marketing Head" }
])

db.employees.insertMany([
  { name: "Amit", age: 30, city: "Mumbai", email: "amit@example.com", position: "Manager" },
  { name: "Priya", age: 25, city: "Delhi", email: "priya@example.com", position: "Software Engineer" },
  { name: "Rahul", age: 35, city: "Bangalore", email: "rahul@example.com", position: "Data Scientist" },
  { name: "Sneha", age: 28, city: "Chennai", email: "sneha@example.com", position: "HR Executive" },
  { name: "Vikram", age: 40, city: "Hyderabad", email: "vikram@example.com", position: "Product Manager" },
  { name: "Anjali", age: 32, city: "Kolkata", email: "anjali@example.com", position: "Marketing Head" }
])


db.employees.createIndex({ name: 1 })

db.employees.createIndex({ email: 1 }, { unique: true })
```

db.employees.createIndex({ age: 1, city: 1 })

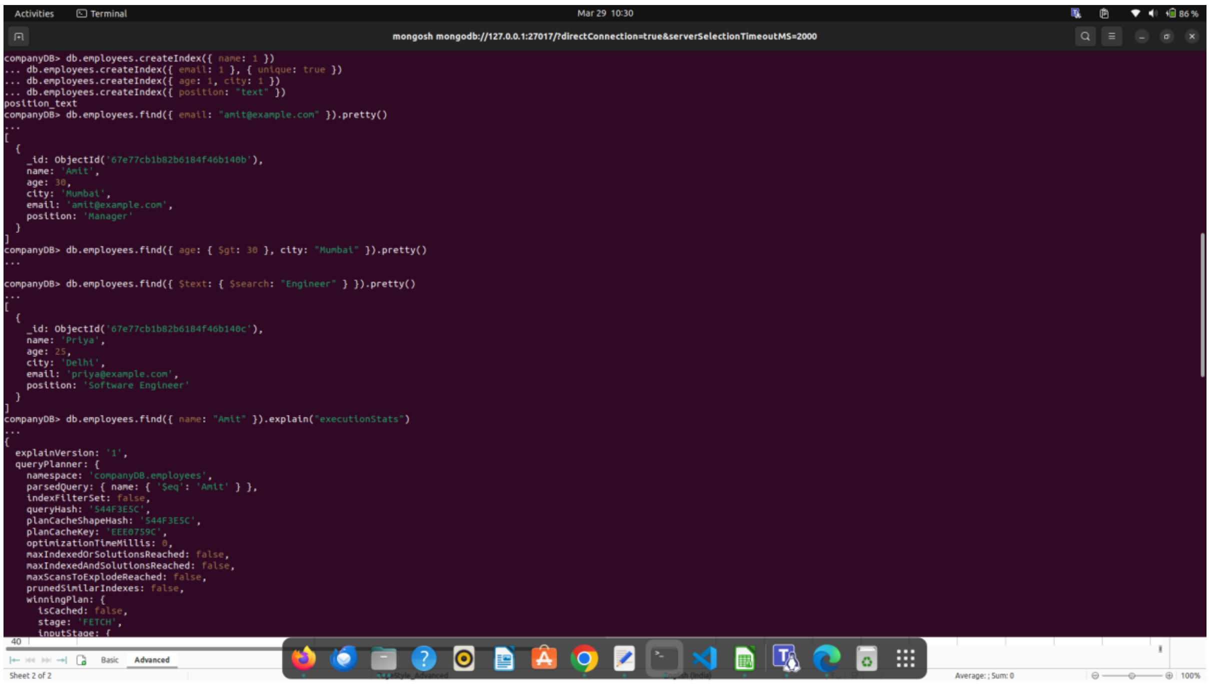
db.employees.createIndex({ position: "text" })

db.employees.find({ email: "amit@example.com" }).pretty()

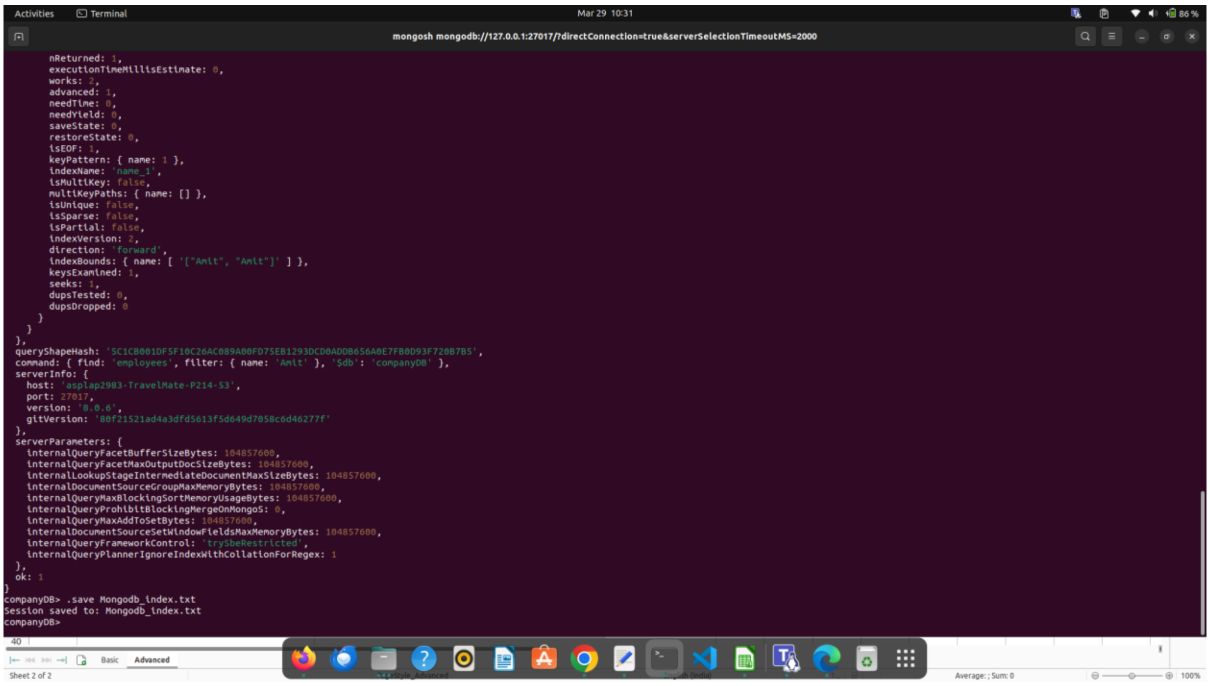
db.employees.find({ age: { $gt: 30 }, city: "Mumbai" }).pretty()

db.employees.find({ $text: { $search: "Engineer" } }).pretty()

db.employees.find({ name: "Amit" }).explain("executionStats")

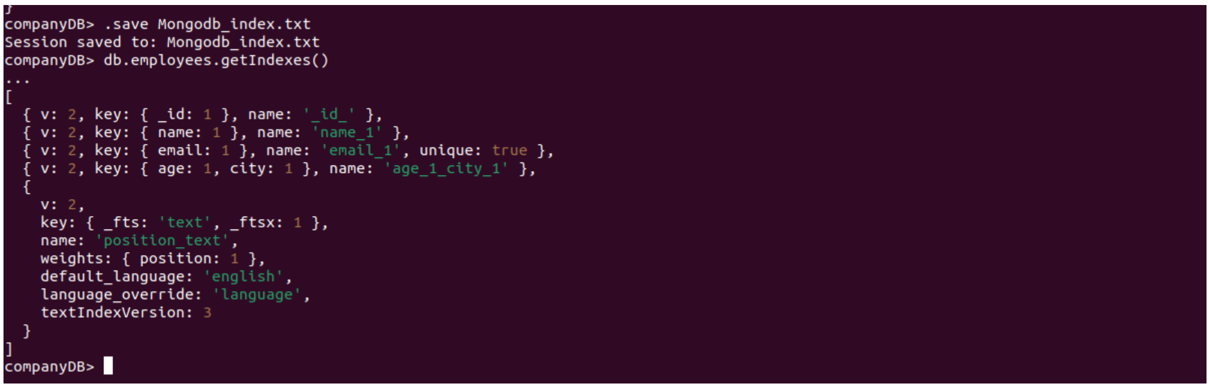


```
companyDB> db.employees.createIndex({ name: 1 })
... db.employees.createIndex({ email: 1 }, { unique: true })
... db.employees.createIndex({ age: 1, city: 1 })
... db.employees.createIndex({ position: "text" })
position_text
companyDB> db.employees.find({ email: "amit@example.com" }).pretty()
...
[
  {
    _id: ObjectId('67e77cb1b82b6184f46b140b'),
    name: 'Amit',
    age: 30,
    city: 'Mumbai',
    email: 'amit@example.com',
    position: 'Manager'
  }
]
companyDB> db.employees.find({ age: { $gt: 30 }, city: "Mumbai" }).pretty()
...

companyDB> db.employees.find({ $text: { $search: "Engineer" } }).pretty()
...
[
  {
    _id: ObjectId('67e77cb1b82b6184f46b140c'),
    name: 'Priya',
    age: 25,
    city: 'Delhi',
    email: 'priya@example.com',
    position: 'Software Engineer'
  }
]
companyDB> db.employees.find({ name: "Amit" }).explain("executionStats")
...
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'companyDB.employees',
    parsedQuery: { name: { '$eq': 'Amit' } },
    indexFilterSet: false,
    queryHash: '544F3E5C',
    planCacheShapeHash: '544F3E5C',
    planCacheKey: 'EEE0759C',
    optimizationTimeMillis: 0,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      stage: 'FETCH',
      inputStage: {
```



```
        nReturned: 1,
        executionTimeMillisEstimate: 0,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 0,
        restoreState: 0,
        isEOF: 1,
        keyPattern: { name: 1 },
        indexName: 'name_1',
        isMultiKey: false,
        multiKeyPaths: { name: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { name: [ '["Amit", "Amit"]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  queryShapeHash: '5C1CB001DF5F10C26AC089A00FD75EB1293DCD0ADDB656A0E7FB0D93F720B7B5',
  command: { find: 'employees', filter: { name: 'Amit' }, '$db': 'companyDB' },
  serverInfo: {
    host: 'asplap2983-TravelMate-P214-53',
    port: 27017,
    version: '8.0.6',
    gitVersion: '80f21521ad4a3dfd5613f5d649d7058c6d46277f'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted',
    internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
  },
  ok: 1
}
companyDB> .save Mongodb_index.txt
Session saved to: Mongodb_index.txt
companyDB>
```

getIndex()



```
}
companyDB> .save Mongodb_index.txt
Session saved to: Mongodb_index.txt
companyDB> db.employees.getIndexes()
...
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1' },
  { v: 2, key: { email: 1 }, name: 'email_1', unique: true },
  { v: 2, key: { age: 1, city: 1 }, name: 'age_1_city_1' },
  {
    v: 2,
    key: { _fts: 'text', _ftsx: 1 },
    name: 'position_text',
    weights: { position: 1 },
    default_language: 'english',
    language_override: 'language',
    textIndexVersion: 3
  }
]
companyDB> 
```

db.employees.dropIndexes() - dropping the indexes

Aggrgation using group and

```
db.employees.aggregate([

 {

   $group: {

     _id: "$city",

     totalEmployees: { $sum: 1 }

   }

 }

]);
```

```
db.employees.aggregate([

 {
  $group: {
   _id: "$city",
   avgSalary: { $avg: "$salary" }
  }
 }
```

Text Search in MongoDB

MongoDB provides powerful text search capabilities to perform full-text search on string content within documents.

Creating a Text Index:

```
db.collection.createIndex({ field: "text" })

db.collection.find({ $text: { $search: "your search term" } })
```

Example:

```
db.articles.createIndex({ title: "text", content: "text" })
db.articles.find({ $text: { $search: "MongoDB text search" } })
```

Sharding in MongoDB

Sharding is a method used to distribute data across multiple servers to handle large datasets and high throughput operations. It ensures horizontal scaling, meaning the data is spread across multiple machines (shards) rather than being stored on a single server. MongoDB uses sharding to maintain high availability, improve query performance, and distribute read/write load efficiently.

Shard Keys

A shard key is a field or combination of fields that MongoDB uses to distribute documents across shards. The choice of a shard key impacts the efficiency of data distribution and query performance.

A good shard key:

- Ensures even data distribution across shards
- Supports efficient query routing

- Avoids performance bottlenecks

Example of defining a shard key:

```
db.employees.createIndex({ employeeId: 1 });
sh.shardCollection("companyDB.employees", { employeeId: 1 });
```

Hashed Sharding

In hashed sharding, MongoDB applies a hash function to the shard key values and distributes documents based on the computed hash. This ensures a more even distribution of data, preventing hotspots where certain values may be more frequently queried.

Advantages:

- Even distribution of data across shards
- Prevents performance issues caused by uneven data distribution

Example of hashed sharding:

```
sh.shardCollection("companyDB.employees", { employeeId: "hashed" });
```

Range Sharding

Range sharding distributes documents based on a continuous range of shard key values. Each shard stores a defined range of values, making range-based queries efficient.

Advantages:

- Efficient range queries since related data is stored together
- Good for sequentially accessed data

Disadvantages:

- Can lead to uneven data distribution if certain ranges have more data than others

Example of range sharding:

```
sh.shardCollection("companyDB.orders", { orderDate: 1 });
```

Using the Balancer in MongoDB

The balancer is responsible for redistributing chunks (data partitions) among shards to maintain an even distribution of data. It automatically moves chunks from overloaded shards to underloaded ones.

Key Points about the Balancer:

- Runs in the background to monitor chunk distribution
- Uses a migration process to move chunks between shards
- Can be manually enabled or disabled

Commands to control the balancer:

- **Check balancer status:**sh.getBalancerState();

- **Enable the balancer:**

```
sh.startBalancer();
```

- **Disable the balancer:**

```
sh.stopBalancer();
```

- **Manually move a chunk:**

```
sh.moveChunk("companyDB.employees", { employeeId: 1000 }, "shard02");
```

Handling Transactions in MongoDB

MongoDB supports multi-document ACID transactions, allowing multiple operations across multiple collections and documents to be executed atomically. Transactions are useful for scenarios where multiple operations must either all succeed or all fail together.

Starting a Transaction

To start a transaction, a **session** is required. A session helps maintain the transaction state.

Example: Performing a Multi-Document Transaction

```
// Start a session
session = db.getMongo().startSession();
```

```
// Start a transaction
session.startTransaction();

try {
   // Get session-aware database and collections
   employeesCollection = session.getDatabase("companyDB").employees;
   salariesCollection = session.getDatabase("companyDB").salaries;

   // Insert a new employee
   employeesCollection.insertOne(
      { _id: 101, name: "John Doe", department: "Finance" }
   );

   // Insert the employee's salary record
   salariesCollection.insertOne(
      { employee_id: 101, salary: 60000 }
   );

   // Commit the transaction
   session.commitTransaction();
   print("Transaction committed successfully");
} catch (error) {
   // If an error occurs, abort the transaction
   session.abortTransaction();
   print("Transaction aborted due to an error:", error);
} finally {
   // End the session
   session.endSession();
}
```

Key Transaction Commands in mongosh

- **Start a session:**

```
session = db.getMongo().startSession();
```

- **Start a transaction:**

```
session.startTransaction();
```

- **Commit a transaction:**

```
session.commitTransaction();
```

- **Abort a transaction:**

```
session.abortTransaction();
```

- **End a session:**

```
session.endSession();
```

Transaction Properties

- **Atomicity:** Either all operations succeed, or none are applied.
- **Consistency:** The database remains in a valid state before and after the transaction.
- **Isolation:** Transactions are isolated from each other.
- **Durability:** Committed transactions persist even in case of failure.

MongoDB Security

MongoDB provides various security mechanisms to ensure data integrity and prevent unauthorized access. Below are key security aspects:

1. Authentication in MongoDB

Authentication ensures that only authorized users can access the database.

Creating an Admin User

```
use admin;
db.createUser({
   user: "admin",
   pwd: "securepassword",
   roles: [{ role: "root", db: "admin" }]
});
```

Enabling Authentication

To enable authentication, modify the MongoDB configuration file (mongod.conf):

```
security:
  authorization: enabled
```

Restart MongoDB:

```
sudo systemctl restart mongod
```

Connect with authentication:

```
mongosh -u admin -p securepassword --authenticationDatabase admin
```

2. Role-Based Access Control (RBAC)

RBAC ensures that users have only the necessary privileges.

Creating a User with Read-Only Access

```
use companyDB;
db.createUser({
    user: "readUser",
    pwd: "readPass",
    roles: [{ role: "read", db: "companyDB" }]
});
```

Creating a User with Read and Write Access

```
db.createUser({
    user: "editor",
    pwd: "editPass",
    roles: [{ role: "readWrite", db: "companyDB" }]
});
```

Granting an Admin Role

```
db.grantRolesToUser("editor", [{ role: "dbAdmin", db: "companyDB" }]);
```

3. Secure Connections with TLS/SSL

To enable secure connections, MongoDB can be configured to use TLS/SSL.

Generate SSL Certificates (Self-Signed)

```
openssl req -newkey rsa:2048 -nodes -keyout mongo-key.pem -x509 -days 365 -out mongo-cert.pem
```

Enable TLS/SSL in mongod.conf

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/mongodb/mongo-cert.pem
```

Connect to MongoDB with SSL

```
mongosh --tls --tlsCAFile /etc/mongodb/mongo-cert.pem --host localhost
```

4. Enabling IP Whitelisting

Restrict access to MongoDB by allowing only specific IP addresses.

Modify mongod.conf

```
net:
  bindIp: 127.0.0.1,192.168.1.100
```

Restart MongoDB:

```
sudo systemctl restart mongod
```

5. Encrypting Data at Rest

MongoDB Enterprise supports **encryption at rest** using the WiredTiger storage engine.

Enable Encryption in mongod.conf

```
security:
  enableEncryption: true
  encryptionKeyFile: /etc/mongodb/mongo-keyfile
```

Restart MongoDB:

```
sudo systemctl restart mongod
```

6. Auditing and Logging

Enable auditing to track database access and changes.

Enable Audit Logging in mongod.conf

View audit logs:

```
cat /var/log/mongodb/audit.log
```

7. Backup and Disaster Recovery

Always back up your data to avoid data loss.

Backup a Database

```
mongodump --db companyDB --out /backup/
```

Restore a Database

```
mongorestore --db companyDB /backup/companyDB
```

Authentication in MongoDB

Authentication ensures that only authorized users can access the database. MongoDB provides different authentication mechanisms such as SCRAM, LDAP, Kerberos, and x.509 certificates.

1. **Enable Authentication**

2. Authentication is disabled by default. To enable it, start the MongoDB server with authentication enabled using the --auth flag.

3. **Create an Admin User**

The first user created should have administrative privileges. The admin database is used for user authentication and authorization.

4. **Create Application Users**

Application users can be created with specific roles to limit their access.

5. **Login as an Authenticated User**

Once authentication is enabled, every user must log in using a valid username and password.

Role-Based Access Control (RBAC)

MongoDB implements RBAC to control user actions. Roles define the level of access for users.

1. **Built-in Roles**
    a. read: Allows read-only access to a database.
    b. readWrite: Allows both read and write operations on a database.
    c. dbAdmin: Provides administrative access to manage a specific database.
    d. userAdmin: Grants permissions to create and manage users.
    e. clusterAdmin: Used for managing the entire cluster.
2. **Custom Roles**

Custom roles can be created with specific permissions for better security and access control.

Connecting Securely to MongoDB with Authentication

1. **Using Username and Password**

When authentication is enabled, users must provide credentials to connect.

2. **Using x.509 Certificates**

MongoDB supports x.509 certificates for client authentication in secure environments.

3. **Using LDAP or Kerberos**

Enterprise versions of MongoDB support integration with LDAP and Kerberos for centralized authentication.

4. Using TLS/SSL for Secure Connections

TLS/SSL encrypts communication between MongoDB clients and servers to prevent data interception.