Savitribai Phule Pune University, Pune

Department of Technology

B.Sc. Data Science

SEMESTER - VI

Subject: NoSQL Databases - Lab

Full Name:

Roll No:

Submission Date:

Internal Examiner Signature	**External Examiner Signature**	**Course Coordinator Signature**

LAB File of NoSQL Databases

INDEX PAGE

Sr. No	Title of Lab	Lab Date	Submission Date	Signature / Remark
1	Create a database and collection using use and createCollection.	20-02-2025	24-02-2025	
2	Insert documents using insertOne and insertMany, and read them using find.	24-02-2025	03-03-2025	
3	Use conditional operators like $gt, $in, and $ne to filter documents.	03-03-2025	10-03-2025	
4	Update documents using $set, $inc, $rename, and $unset	10-03-2025	17-03-2025	
5	Delete documents using deleteOne() and deleteMany() with conditions.	17-03-2025	24-03-2025	
6	Sort, limit, and skip documents in query results.	17-03-2025	24-03-2025	
7	Create and modify arrays using $set and $push	24-03-2025	21-04-2025	
8	Use aggregation pipeline operators like $group and $sort	24-03-2025	21-04-2025	
9	Create, view, and delete indexes using createIndex, getIndexes, and dropIndex.	21-04-2025	28-04-2025	

10	Use distinct() and countDocuments() to retrieve data insights.	21-04-2025	28-04-2025	
11	Compare estimatedDocumentCount() with countDocuments() for performance analysis.	28-04-2025	30-04-2025	
12	Create a user and manage roles using createUser, updateUser, and grantRolesToUser.	28-04-2025	30-04-2025	

Experiment 1 :

Create a database and collection using use and createCollection.

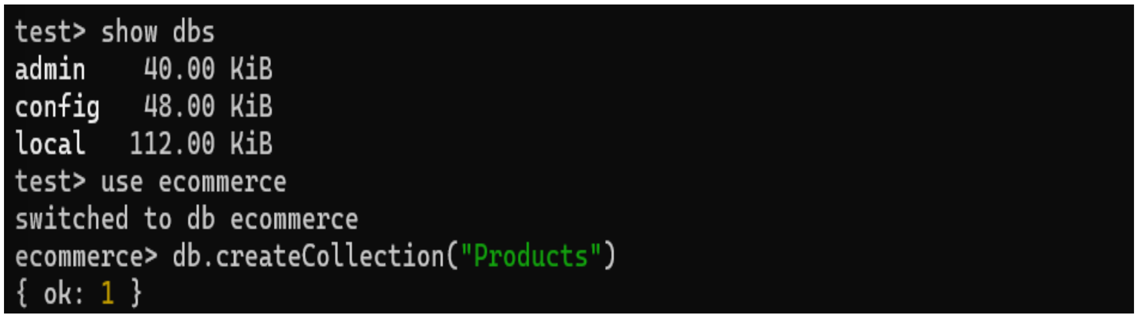
Aim : To create a MongoDB database and collection using the use command and createCollection() method for managing product data in an e-commerce application.

Objective :

- To understand the process of initializing a new database in MongoDB.
- To learn how to create a collection explicitly using the createCollection() method.
- To establish the foundational structure for storing and managing e-commerce product information.

a) Create a database named `ecommerce`.

b) Create a collection named `products`.



```
test> show dbs
admin    40.00 KiB
config   48.00 KiB
local   112.00 KiB
test> use ecommerce
switched to db ecommerce
ecommerce> db.createCollection("Products")
{ ok: 1 }
```

Experiment 2:

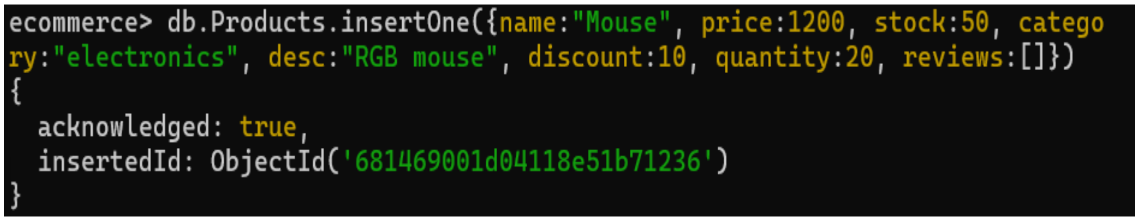
Insert documents using insertOne and insertMany, and read them using find.

Aim : To perform data insertion operations using insertOne() and insertMany() methods and retrieve data using the find() method in MongoDB.

Objective :

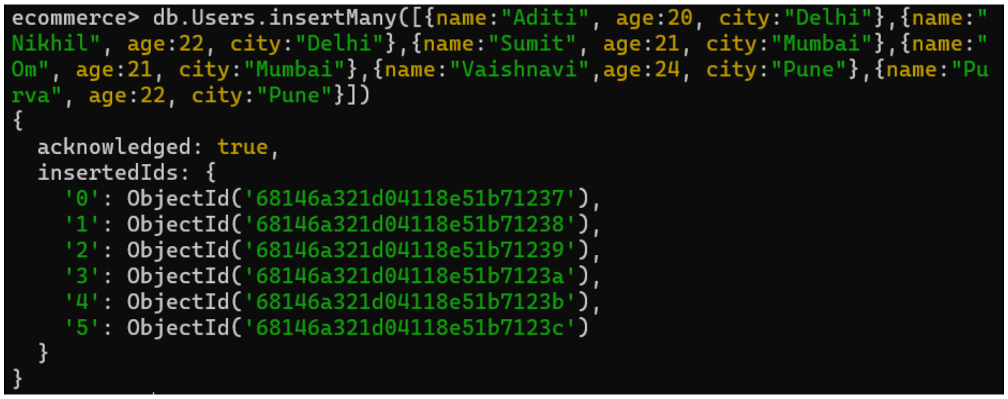
- To learn how to insert a single document into a MongoDB collection using insertOne().
- To understand how to insert multiple documents at once using insertMany().
- To practice retrieving all documents from a collection using the find() method.
- To gain hands-on experience with basic CRUD operations in MongoDB.

a) **Insert a product document into `products`.**



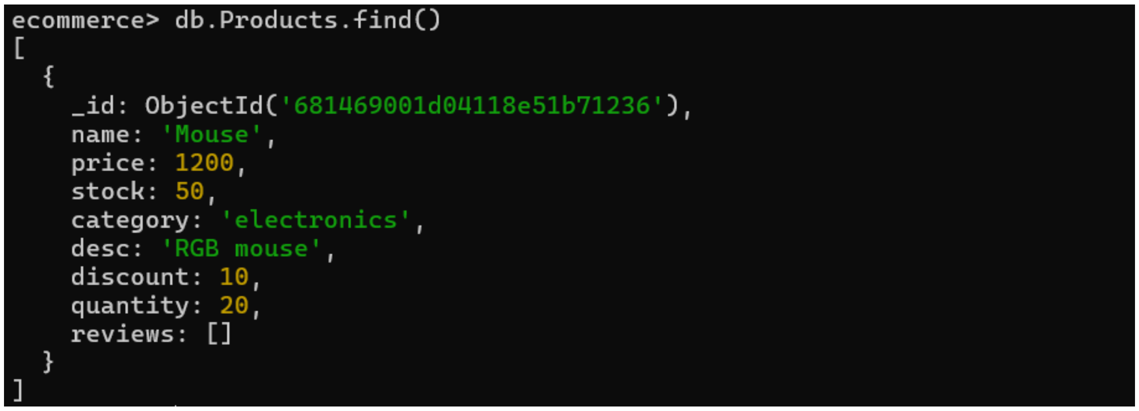
```
ecommerce> db.Products.insertOne({name:"Mouse", price:1200, stock:50, catego
ry:"electronics", desc:"RGB mouse", discount:10, quantity:20, reviews:[]})
{
  acknowledged: true,
  insertedId: ObjectId('681469001d04118e51b71236')
}
```

b) **Insert multiple user documents into a `users` collection.**



```
ecommerce> db.Users.insertMany([{name:"Aditi", age:20, city:"Delhi"},{name:"
Nikhil", age:22, city:"Delhi"},{name:"Sumit", age:21, city:"Mumbai"},{name:"
Om", age:21, city:"Mumbai"},{name:"Vaishnavi",age:24, city:"Pune"},{name:"Pu
rva", age:22, city:"Pune"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68146a321d04118e51b71237'),
    '1': ObjectId('68146a321d04118e51b71238'),
    '2': ObjectId('68146a321d04118e51b71239'),
    '3': ObjectId('68146a321d04118e51b7123a'),
    '4': ObjectId('68146a321d04118e51b7123b'),
    '5': ObjectId('68146a321d04118e51b7123c')
  }
}
```

c) **Display all documents from the `products` collection.**



```
ecommerce> db.Products.find()
[
  {
    _id: ObjectId('681469001d04118e51b71236'),
    name: 'Mouse',
    price: 1200,
    stock: 50,
    category: 'electronics',
    desc: 'RGB mouse',
    discount: 10,
    quantity: 20,
    reviews: []
  }
]
```

Experiment 3:

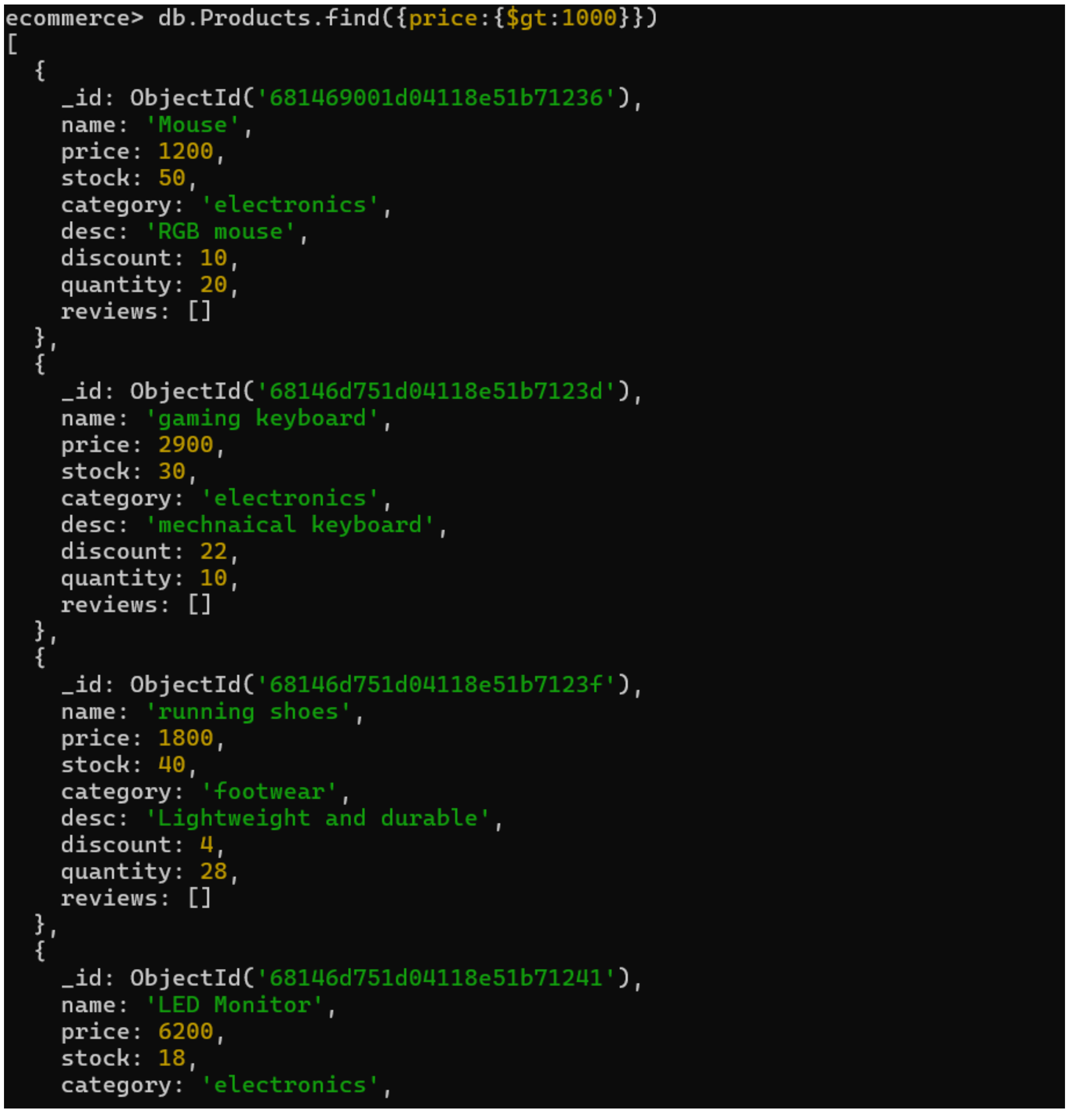
Use conditional operators like $gt, $in, and $ne to filter documents.

Aim : To use conditional query operators such as $gt, $in, and $ne in MongoDB to filter and retrieve documents based on specific criteria.

Objective :

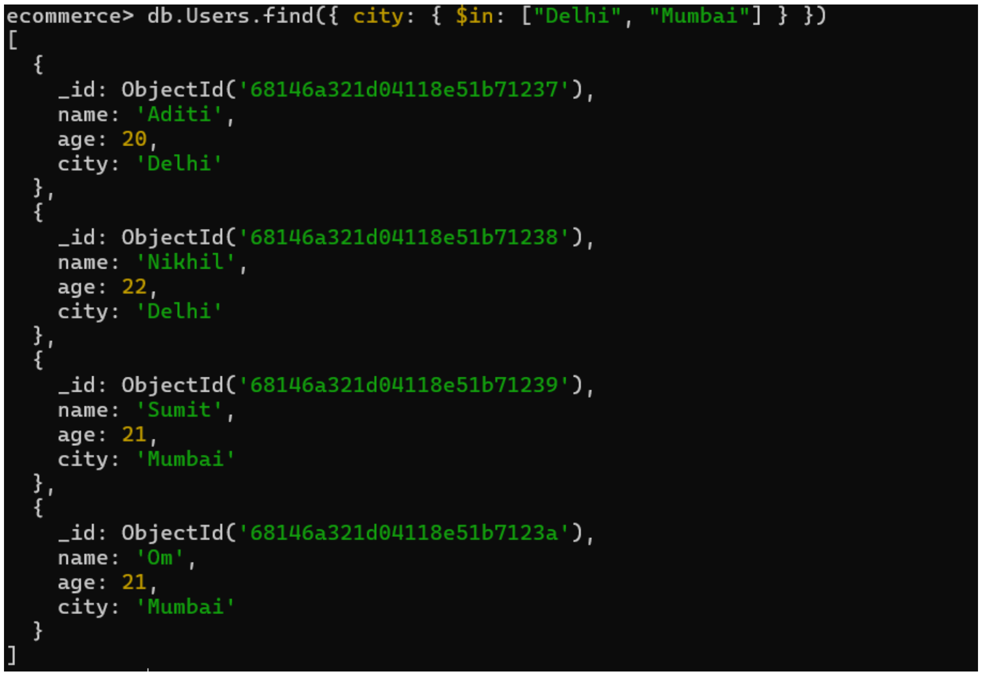
- To understand how to filter documents using the $gt (greater than) operator.
- To learn the application of the $in operator for matching values from a list.
- To apply the $ne (not equal) operator to exclude certain values from query results.
- To retrieve targeted data from MongoDB collections based on conditional logic.

  a) Display products with price greater than 1000.



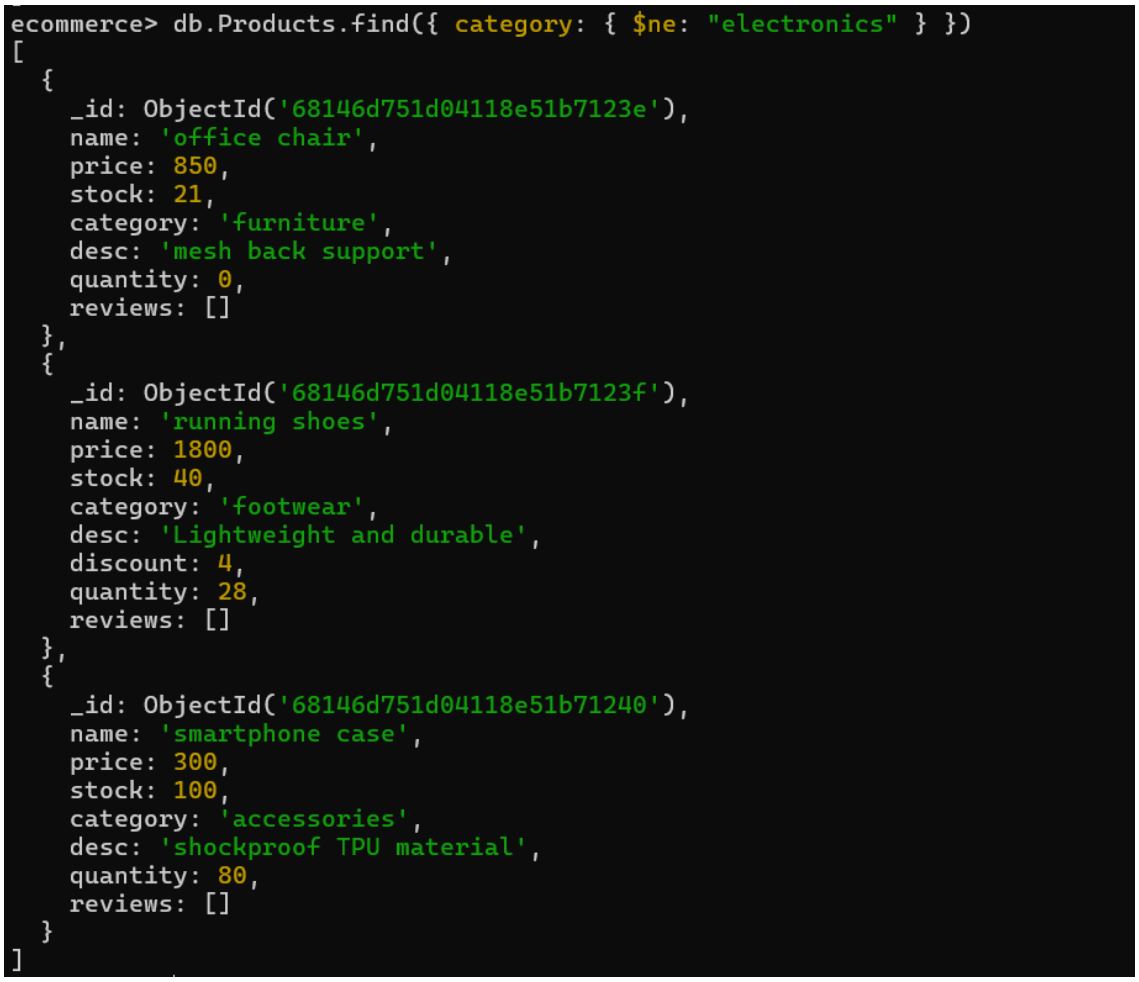
```
ecommerce> db.Products.find({price:{$gt:1000}})
[
  {
    _id: ObjectId('681469001d04118e51b71236'),
    name: 'Mouse',
    price: 1200,
    stock: 50,
    category: 'electronics',
    desc: 'RGB mouse',
    discount: 10,
    quantity: 20,
    reviews: []
  },
  {
    _id: ObjectId('68146d751d04118e51b7123d'),
    name: 'gaming keyboard',
    price: 2900,
    stock: 30,
    category: 'electronics',
    desc: 'mechnaical keyboard',
    discount: 22,
    quantity: 10,
    reviews: []
  },
  {
    _id: ObjectId('68146d751d04118e51b7123f'),
    name: 'running shoes',
    price: 1800,
    stock: 40,
    category: 'footwear',
    desc: 'Lightweight and durable',
    discount: 4,
    quantity: 28,
    reviews: []
  },
  {
    _id: ObjectId('68146d751d04118e51b71241'),
    name: 'LED Monitor',
    price: 6200,
    stock: 18,
    category: 'electronics',
```

b) Find users whose city is either 'Delhi' or 'Mumbai'.



```
ecommerce> db.Users.find({ city: { $in: ["Delhi", "Mumbai"] } })
[
  {
    _id: ObjectId('68146a321d04118e51b71237'),
    name: 'Aditi',
    age: 20,
    city: 'Delhi'
  },
  {
    _id: ObjectId('68146a321d04118e51b71238'),
    name: 'Nikhil',
    age: 22,
    city: 'Delhi'
  },
  {
    _id: ObjectId('68146a321d04118e51b71239'),
    name: 'Sumit',
    age: 21,
    city: 'Mumbai'
  },
  {
    _id: ObjectId('68146a321d04118e51b7123a'),
    name: 'Om',
    age: 21,
    city: 'Mumbai'
  }
]
```

c) List products not from category 'electronics'.



```
ecommerce> db.Products.find({ category: { $ne: "electronics" } })
[
  {
    _id: ObjectId('68146d751d04118e51b7123e'),
    name: 'office chair',
    price: 850,
    stock: 21,
    category: 'furniture',
    desc: 'mesh back support',
    quantity: 0,
    reviews: []
  },
  {
    _id: ObjectId('68146d751d04118e51b7123f'),
    name: 'running shoes',
    price: 1800,
    stock: 40,
    category: 'footwear',
    desc: 'Lightweight and durable',
    discount: 4,
    quantity: 28,
    reviews: []
  },
  {
    _id: ObjectId('68146d751d04118e51b71240'),
    name: 'smartphone case',
    price: 300,
    stock: 100,
    category: 'accessories',
    desc: 'shockproof TPU material',
    quantity: 80,
    reviews: []
  }
]
```

Experiment 4 :

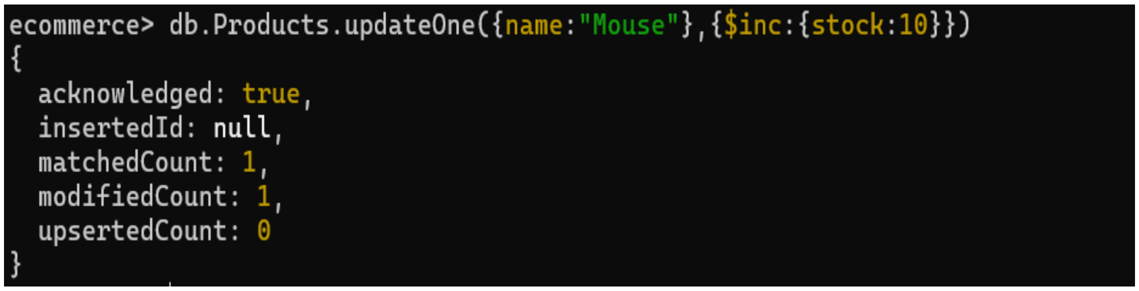
Update documents using $set, $inc, $rename, and $unset.

Aim : To perform document updates in MongoDB using operators like $set, $inc, $rename, and $unset to modify fields efficiently.

Objective :

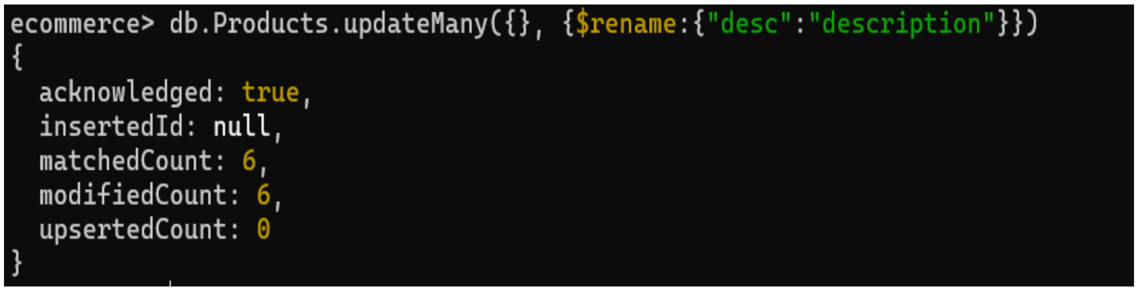
- To understand how to increment numeric values using the $inc operator.
- To learn how to rename a field in a document using the $rename operator.
- To explore the $unset operator for removing fields from documents.
- To apply the $set operator for updating or adding new fields.

a) Update product stock by incrementing it by 10.



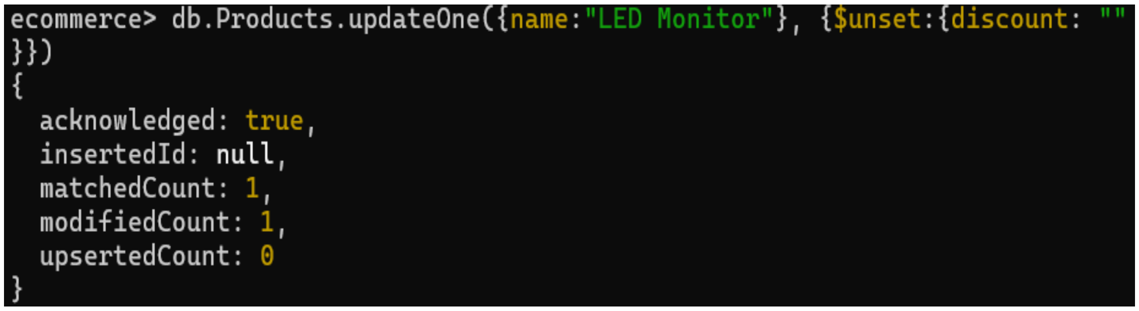
```
ecommerce> db.Products.updateOne({name:"Mouse"},{$inc:{stock:10}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

b) Rename field `desc` to `description`.



```
ecommerce> db.Products.updateMany({}, {$rename:{"desc":"description"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 6,
  modifiedCount: 6,
  upsertedCount: 0
}
```

c) Remove the `discount` field from a product.



```
ecommerce> db.Products.updateOne({name:"LED Monitor"}, {$unset:{discount: ""
}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Experiment 5 :

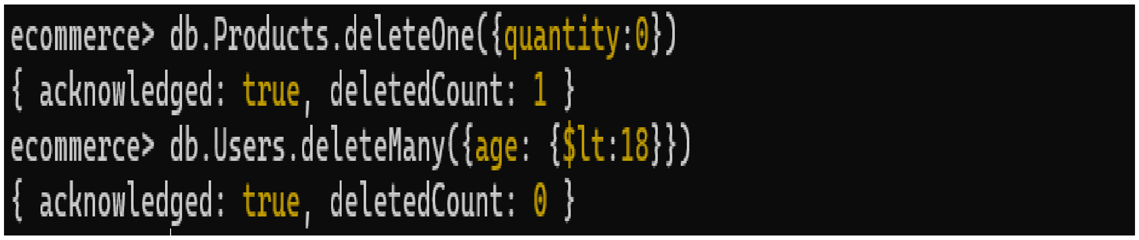
Delete documents using deleteOne() and deleteMany() with conditions.

Aim : To learn how to delete specific documents from a MongoDB collection using deleteOne() and deleteMany() based on given conditions.

Objective :

- To understand how to use deleteOne() to remove a single document that matches a condition (e.g., product with quantity 0).
- To learn how to use deleteMany() to delete multiple documents that match a condition (e.g., users below age 18).
- To practice applying conditions while deleting data from a database.

a) Delete one product with quantity 0.
b) Delete all users below age 18.



```
ecommerce> db.Products.deleteOne({quantity:0})
{ acknowledged: true, deletedCount: 1 }
ecommerce> db.Users.deleteMany({age: {$lt:18}})
{ acknowledged: true, deletedCount: 0 }
```

Experiment 6 :

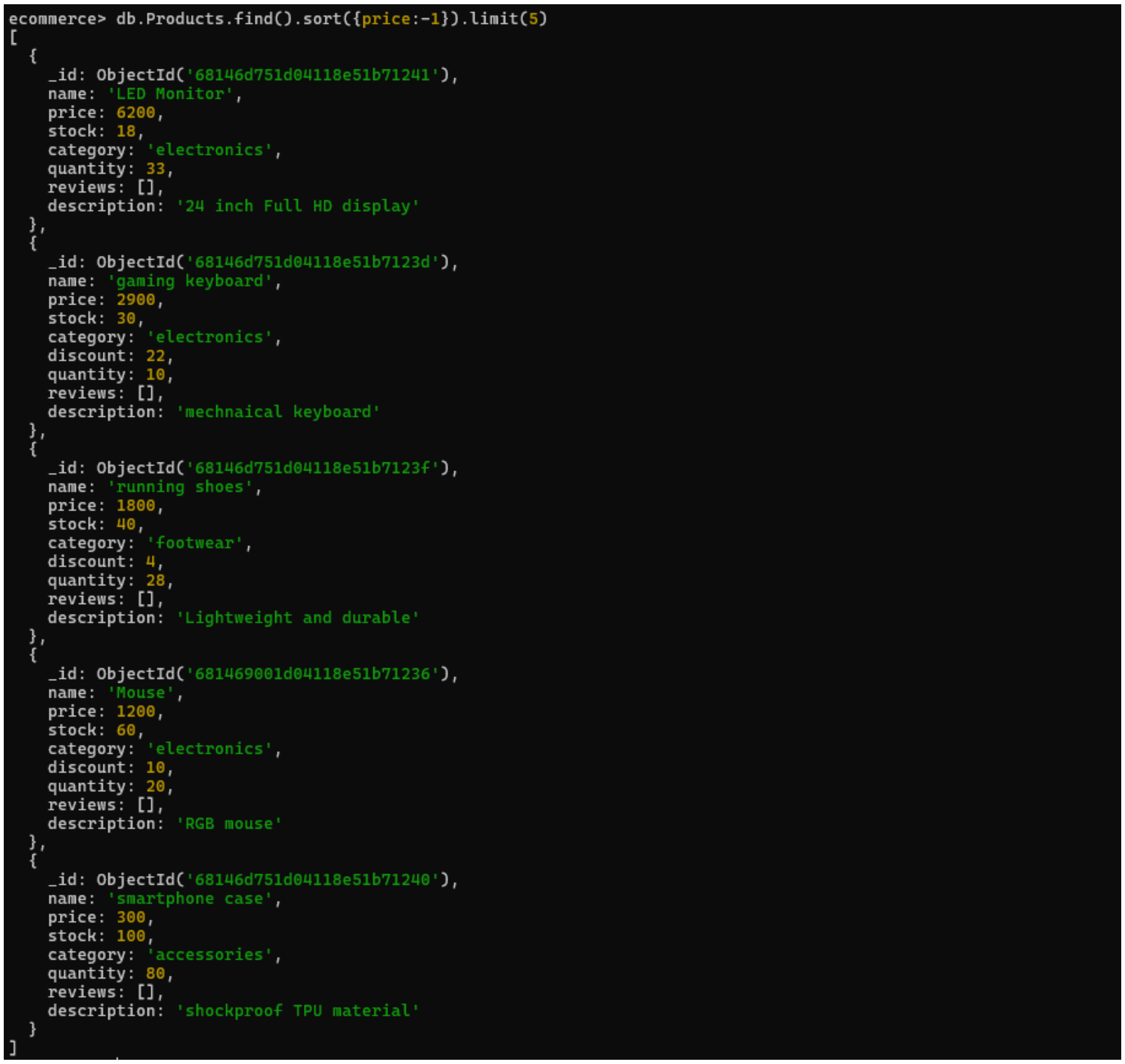
Sort, limit, and skip documents in query results.

Aim : To learn how to sort, limit, and skip documents in MongoDB query results.

Objective :

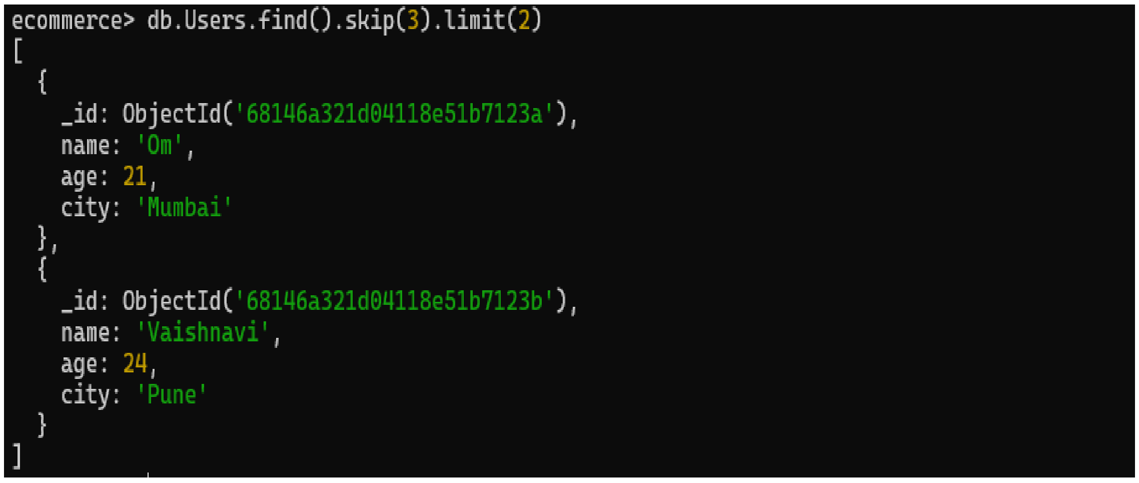
- To understand how to sort documents based on a specific field (e.g., price in descending order).
- To learn how to use limit() to restrict the number of results (e.g., top 5 products).
- To practice using skip() to skip a number of documents and then display the next few (e.g., skip first 3 users and show the next 2).

a) Display top 5 products sorted by price in descending order.



```
ecommerce> db.Products.find().sort({price:-1}).limit(5)
[
  {
    _id: ObjectId('68146d751d04118e51b71241'),
    name: 'LED Monitor',
    price: 6200,
    stock: 18,
    category: 'electronics',
    quantity: 33,
    reviews: [],
    description: '24 inch Full HD display'
  },
  {
    _id: ObjectId('68146d751d04118e51b7123d'),
    name: 'gaming keyboard',
    price: 2900,
    stock: 30,
    category: 'electronics',
    discount: 22,
    quantity: 10,
    reviews: [],
    description: 'mechnaical keyboard'
  },
  {
    _id: ObjectId('68146d751d04118e51b7123f'),
    name: 'running shoes',
    price: 1800,
    stock: 40,
    category: 'footwear',
    discount: 4,
    quantity: 28,
    reviews: [],
    description: 'Lightweight and durable'
  },
  {
    _id: ObjectId('681469001d04118e51b71236'),
    name: 'Mouse',
    price: 1200,
    stock: 60,
    category: 'electronics',
    discount: 10,
    quantity: 20,
    reviews: [],
    description: 'RGB mouse'
  },
  {
    _id: ObjectId('68146d751d04118e51b71240'),
    name: 'smartphone case',
    price: 300,
    stock: 100,
    category: 'accessories',
    quantity: 80,
    reviews: [],
    description: 'shockproof TPU material'
  }
]
```

b) Skip first 3 users and display the next 2.



```
ecommerce> db.Users.find().skip(3).limit(2)
[
  {
    _id: ObjectId('68146a321d04118e51b7123a'),
    name: 'Om',
    age: 21,
    city: 'Mumbai'
  },
  {
    _id: ObjectId('68146a321d04118e51b7123b'),
    name: 'Vaishnavi',
    age: 24,
    city: 'Pune'
  }
]
```

Experiment 7 :

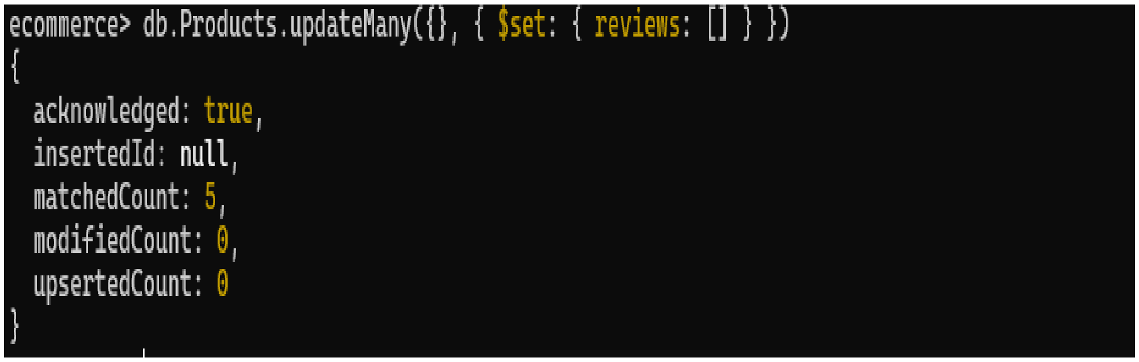
Create and modify arrays using $set and $push.

Aim : To learn how to create and update array fields in MongoDB documents using $set and $push.

Objective :

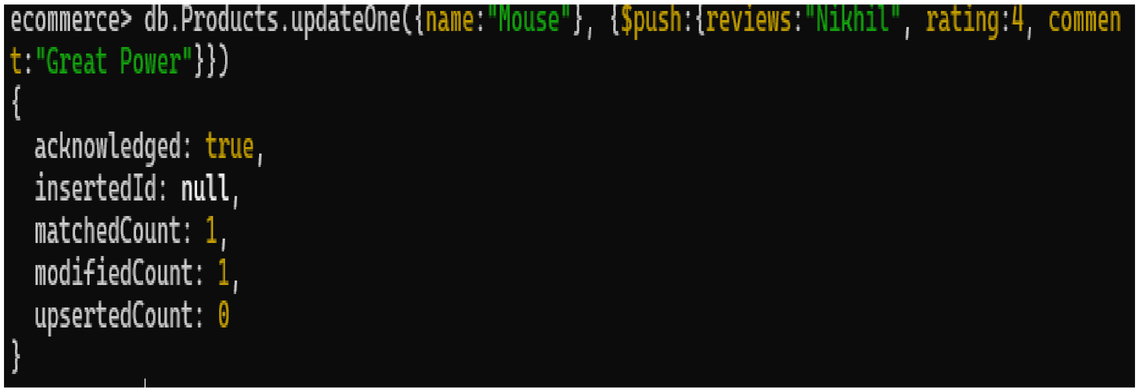
- To use the $set operator to add an empty reviews array to all product documents.
- To use the $push operator to insert a review object into the reviews array of a specific product.
- To understand how arrays are handled and updated in MongoDB.

a) Add an empty `reviews` array to all products.



```
ecommerce> db.Products.updateMany({}, { $set: { reviews: [] } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 0,
  upsertedCount: 0
}
```

b) Push a review object into the `reviews` array of a specific product.



```
ecommerce> db.Products.updateOne({name:"Mouse"}, {$push:{reviews:"Nikhil", rating:4, commen
t:"Great Power"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Experiment 8 :

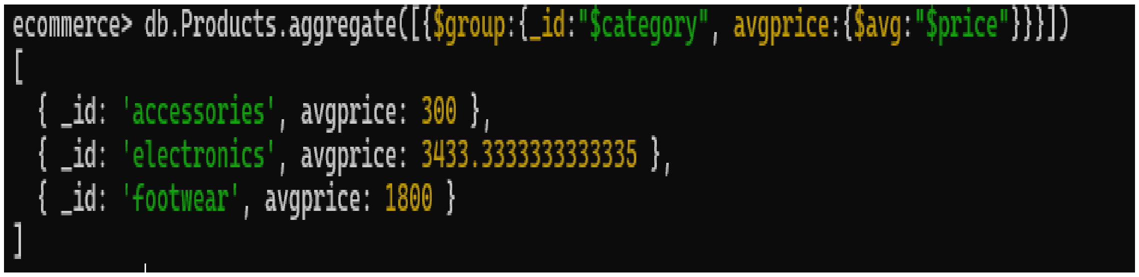
Use aggregation pipeline operators like $group and $sort.

Aim : To understand and apply MongoDB aggregation pipeline operators like $group and $sort to analyze and summarize data.

Objective :

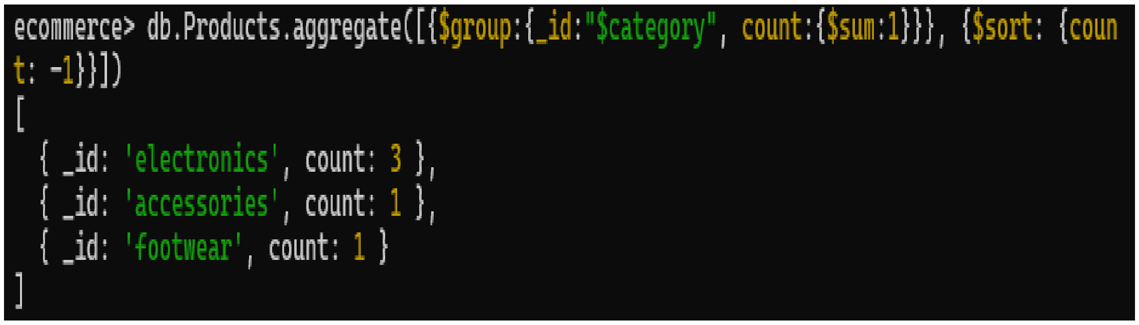
- To group products by category and calculate the average price using the $group operator.
- To sort product categories by the total number of products in descending order using $sort.
- To write an aggregation query on the ecommerce collection to summarize review data.
- To learn how to use aggregation to generate reports and insights from data.

a) Group products by category and calculate average price.



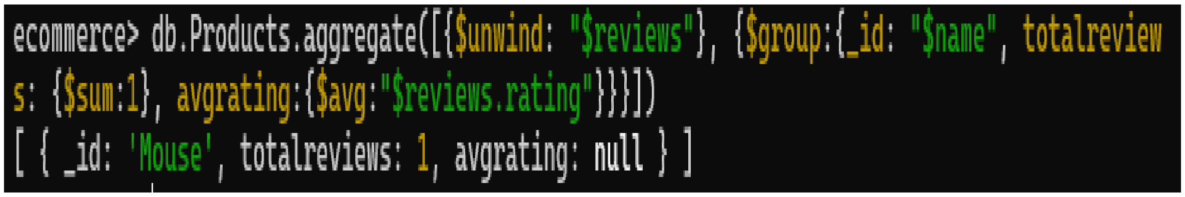
```
ecommerce> db.Products.aggregate([{$group:{_id:"$category", avgprice:{$avg:"$price"}}}])
[
  { _id: 'accessories', avgprice: 300 },
  { _id: 'electronics', avgprice: 3433.3333333333335 },
  { _id: 'footwear', avgprice: 1800 }
]
```

b) Sort categories by total number of products in descending order.



```
ecommerce> db.Products.aggregate([{$group:{_id:"$category", count:{$sum:1}}}, {$sort: {coun
t: -1}}])
[
  { _id: 'electronics', count: 3 },
  { _id: 'accessories', count: 1 },
  { _id: 'footwear', count: 1 }
]
```

c) Using the `ecommerce` collection, write a query to display reviews summary.



```
ecommerce> db.Products.aggregate([{$unwind: "$reviews"}, {$group:{_id: "$name", totalreview
s: {$sum:1}, avgrating:{$avg:"$reviews.rating"}}}])
[ { _id: 'Mouse', totalreviews: 1, avgrating: null } ]
```

Experiment 9 :

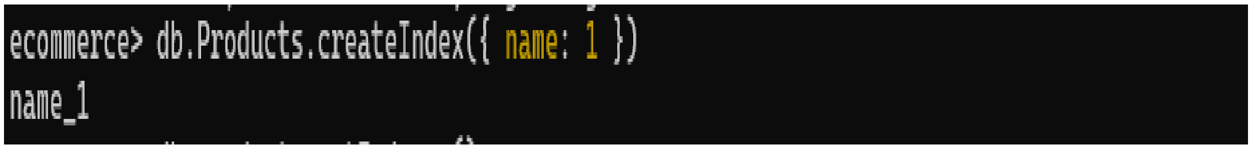
Create, view, and delete indexes using createIndex, getIndexes, and dropIndex.

Aim : To learn how to create, view, and delete indexes in MongoDB to improve query performance.

Objective :

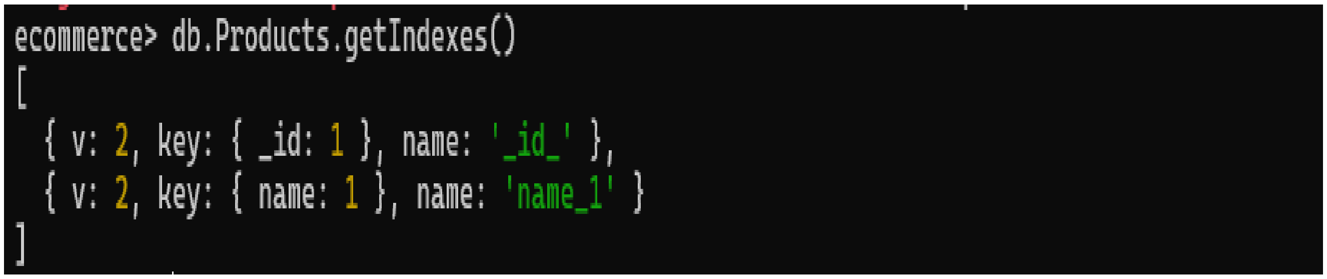
- To create an index on the productName field using createIndex().
- To view all existing indexes on the products collection using getIndexes().
- To delete the index on productName using dropIndex().
- To understand the role of indexes in optimizing database queries.

a) Create an index on `productName`.



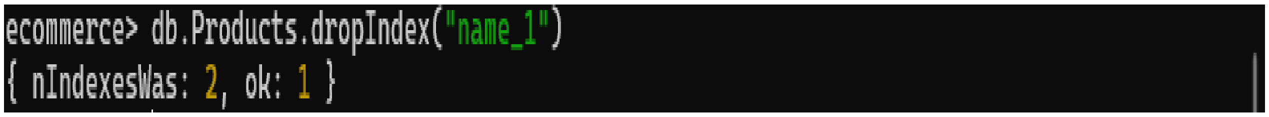
```
ecommerce> db.Products.createIndex({ name: 1 })
name_1
```

b) View all indexes on `products` collection.



```
ecommerce> db.Products.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1' }
]
```

c) Drop the index on `productName`.



```
ecommerce> db.Products.dropIndex("name_1")
{ nIndexesWas: 2, ok: 1 }
```

Experiment 10 :

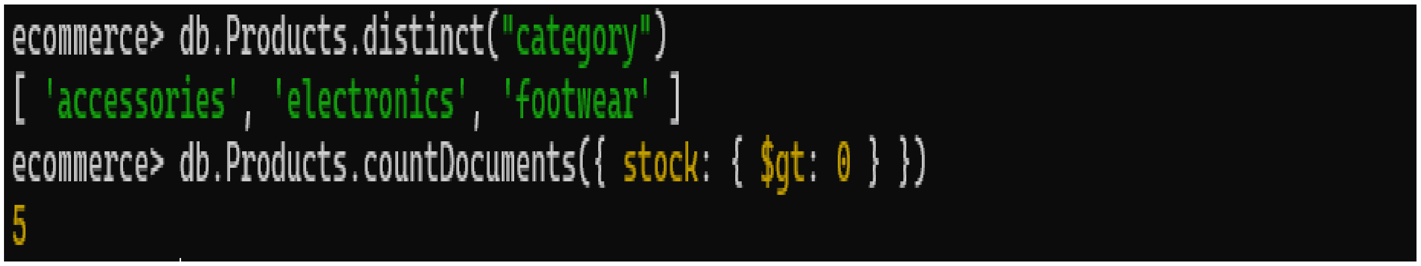
Use distinct() and countDocuments() to retrieve data insights.

Aim : To learn how to retrieve unique values and count documents in MongoDB collections.

Objective :

- To use distinct() to find all unique categories in the products collection.
- To use countDocuments() to count how many products are currently in stock.
- To understand how these functions help in getting useful data insights.

a) **Find all distinct categories in `products`.**
b) **Count how many products are in stock.**



```
ecommerce> db.Products.distinct("category")
[ 'accessories', 'electronics', 'footwear' ]
ecommerce> db.Products.countDocuments({ stock: { $gt: 0 } })
5
```

Experiment 11:

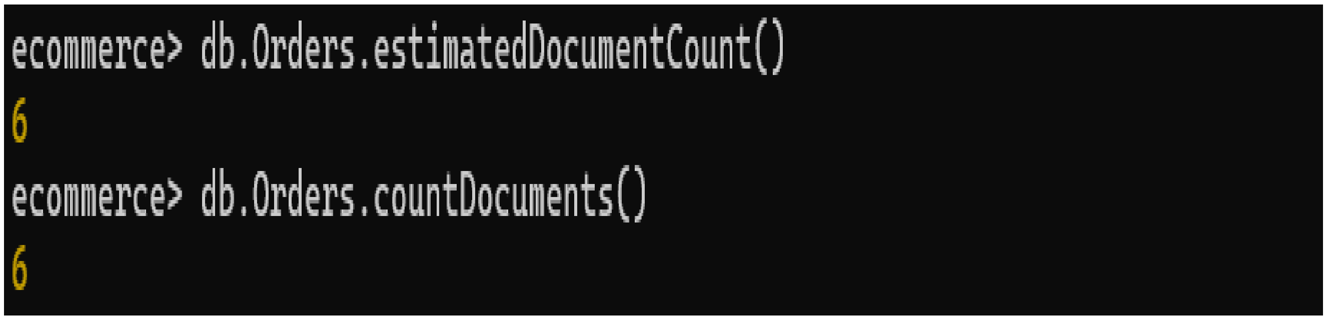
Compare estimatedDocumentCount() with countDocuments() for performance analysis.

Aim : To understand the difference between estimatedDocumentCount() and countDocuments() in terms of accuracy and performance.

Objective :

- To use estimatedDocumentCount() to get a quick estimate of documents in the orders collection.
- To use countDocuments() to get the exact number of documents with more accuracy.
- To compare the speed and use cases of both methods.
- To learn when to use each method based on the requirement (speed vs. accuracy).

  a) **Find the estimated document count for `orders` collection.**
  b) **Find the exact count using `countDocuments()` and compare.**



```
ecommerce> db.Orders.estimatedDocumentCount()
6
ecommerce> db.Orders.countDocuments()
6
```

Experiment 12:

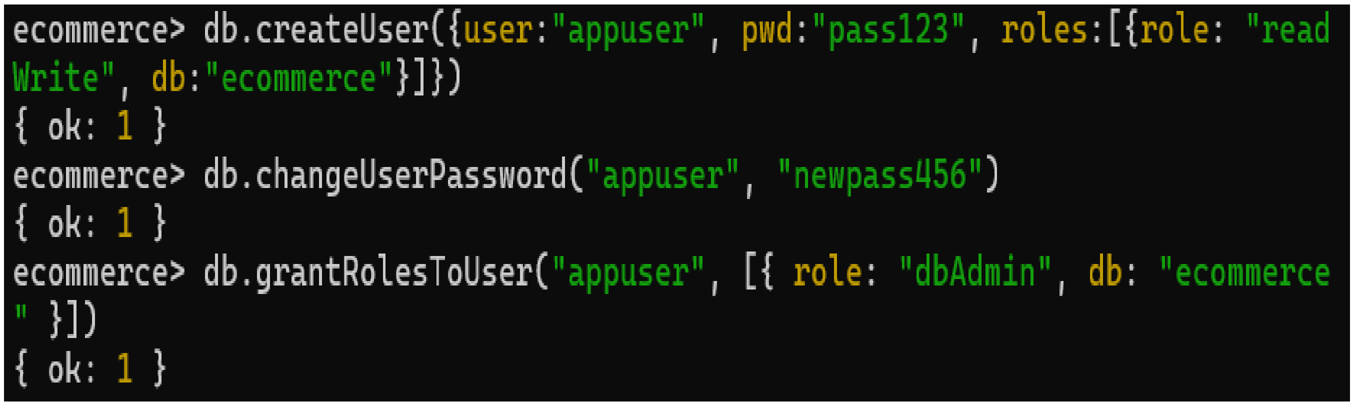
Create a user and manage roles using createUser, updateUser, and grantRolesToUser.

Aim : To learn how to create and manage MongoDB users and assign roles for access control.

Objective :

- To create a new user with readWrite access using createUser().
- To update the password of an existing user using updateUser().
- To grant an admin role to a user using grantRolesToUser().
- To understand how user roles control access and permissions in MongoDB.

a) **Create a new user with readWrite access.**
b) **Change the password of an existing user.**
c) **Grant admin role to a user.**



```
ecommerce> db.createUser({user:"appuser", pwd:"pass123", roles:[{role: "read
Write", db:"ecommerce"}]})
{ ok: 1 }
ecommerce> db.changeUserPassword("appuser", "newpass456")
{ ok: 1 }
ecommerce> db.grantRolesToUser("appuser", [{ role: "dbAdmin", db: "ecommerce
" }])
{ ok: 1 }
```