

**Name: Prachi Shinde**

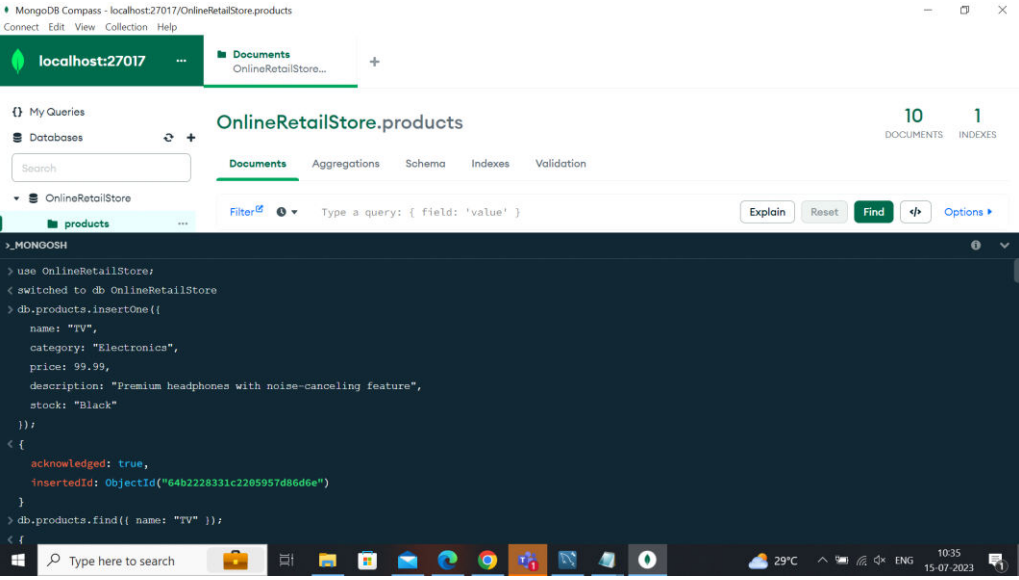
1. Design the MongoDB database schema to store the necessary information for the online retail store.

created "OnlineRetailStore" Database and "products" Collection.

2. Implement the necessary MongoDB queries for the following operations:

**a. Insert a new product into the database.**

**Result=>**

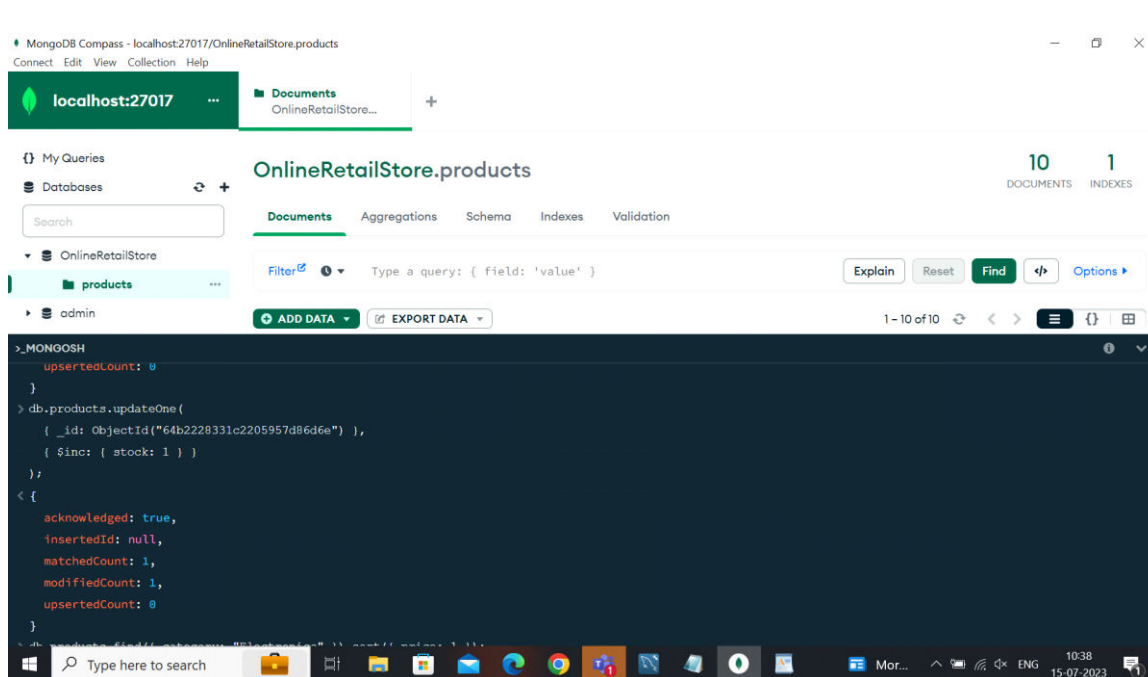


The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'localhost:27017/OnlineRetailStore.products'. The left sidebar shows the 'OnlineRetailStore' database and the 'products' collection. The main panel displays the 'products' collection with 10 documents and 1 index. The 'Documents' tab is active, showing a list of documents. The bottom panel shows the MongoDB Shell with the following commands and output:

```
> use OnlineRetailStore;
< switched to db OnlineRetailStore
> db.products.insertOne({
  name: "TV",
  category: "Electronics",
  price: 99.99,
  description: "Premium headphones with noise-canceling feature",
  stock: "Black"
});
< {
  acknowledged: true,
  insertedId: ObjectId("64b2228331c2205957d86d6e")
}
> db.products.find({ name: "TV" });
< f
```

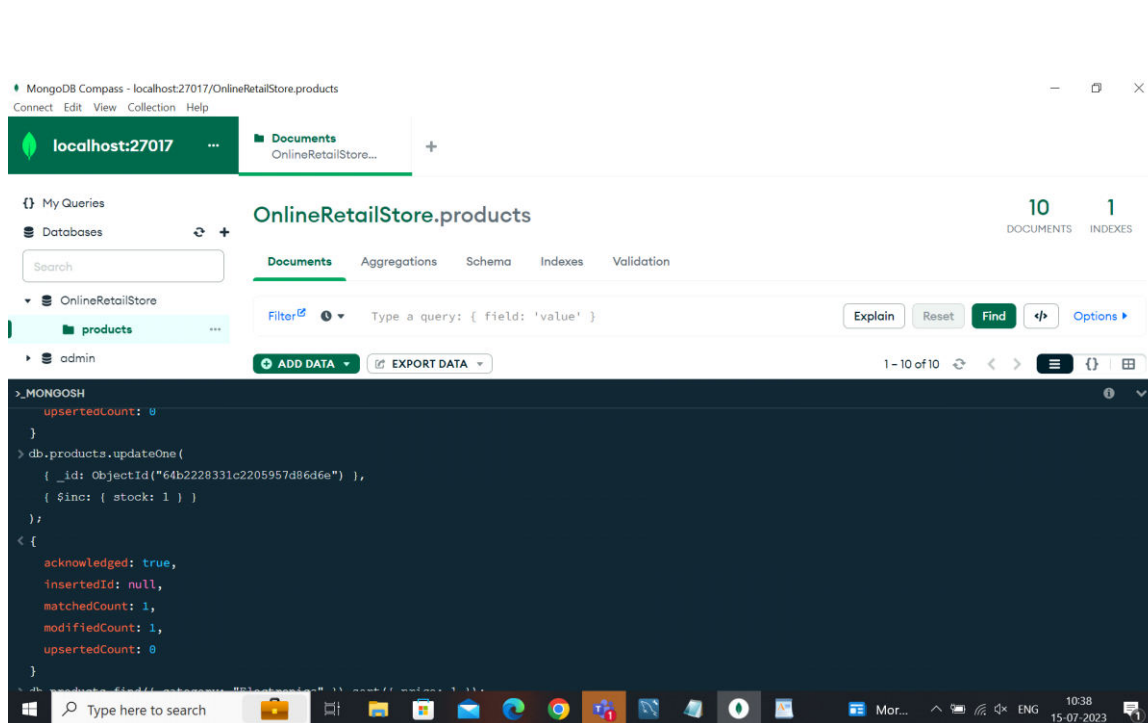
**b1. Update the stock quantity of a product when an order is placed.**

**Result=>**



**b2. Update the stock quantity of a product when an order is cancelled.**

**Result=>**



**c. Retrieve a list of products in a specific category, sorted by price.**

Result=>

The screenshot shows the MongoDB Compass interface for the 'OnlineRetailStore.products' collection. The 'Documents' tab is active, displaying a list of products. A filter is applied: 'category: "Electronics"'. The results are sorted by 'price' in descending order. The first two documents are visible:

```
> db.products.find({ category: "Electronics" }).sort({ price: -1 });
```

```
{
  "_id": ObjectId("64b22679da350c28fc817911"),
  "name": "Bluetooth Speaker",
  "description": "Lightweight and portable Bluetooth speaker with built-in microphone",
  "category": "Electronics",
  "price": "$59.99 ",
  "stock": 40
}
```

```
{
  "_id": ObjectId("64b22679da350c28fc817913"),
  "name": "Camera",
  "description": "Compact digital camera with high-resolution image capture",
  "category": "Electronics",
  "price": "$199.99 ",
  "stock": 10
}
```

d. Search for products within a given price range and display only those with available stock.

Result=>

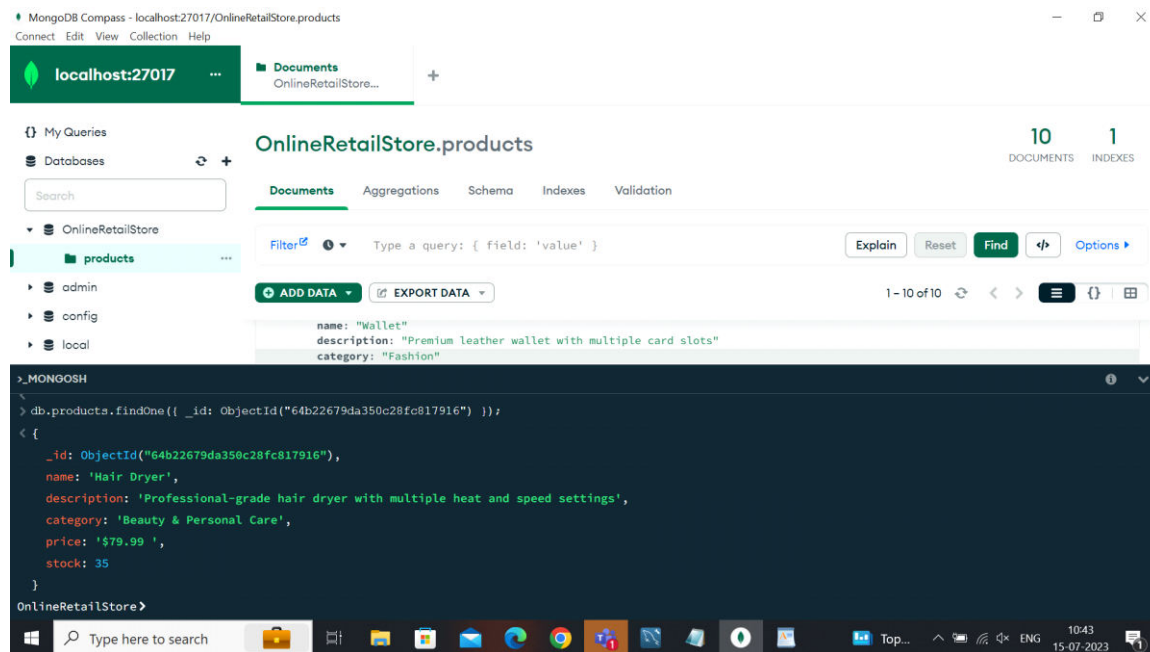
The screenshot shows the MongoDB Compass interface for the 'OnlineRetailStore.products' collection. The 'Documents' tab is active, displaying a list of products. A filter is applied: 'category: "Electronics"'. The results are sorted by 'price' in descending order. The first two documents are visible:

```
> db.products.find({ category: "Electronics" }).sort({ price: -1 });
```

```
{
  "_id": ObjectId("64b22679da350c28fc81790f"),
  "name": "Water Bottle",
  "description": "Durable stainless steel water bottle with leak-proof cap",
  "category": "Home & Kitchen",
  "price": "$19.99 "
}
```

e. Retrieve the details of a specific product by its unique identifier.

Result=>



3. Explain how MongoDB's indexing features can improve the performance of the inventory system.

MongoDB's indexing features improve performance by enabling faster query execution, efficient sorting and ordering, optimized query plans, covered queries, multikey indexes for arrays, sparse indexes for missing values, and text indexes for full-text search. These features minimize data scanning, reduce I/O operations, and enhance the inventory system's overall performance.

4. Discuss the scalability options available in MongoDB to handle increasing data and traffic.

MongoDB offers scalability options such as sharding for horizontal scaling, replica sets for high availability, auto-sharding for simplified distribution of data, data partitioning for optimized query performance, read preference and tagging for control over read operations, Mongos router for query routing, and cluster management tools for monitoring and optimization.