**Scenario Overview: eCommerce Database**

We'll create a MongoDB database to store information about:

1. **Products** – Details about the products sold.
2. **Customers** – Information about registered customers.

**Step 1: Setting Up the Database and Collections**

Let's first **create the database** and **collections**.

**Switch to or Create a Database**

use eCommerce;

**Create Collections**

db.createCollection("products");

db.createCollection("customers");

**Step 2: Inserting Documents**

**Insert Products**

Let's insert some products into the products collection.

db.products.insertMany([

{

"name": "Laptop",

"brand": "Apple",

"price": 1200,

"inStock": true,

"categories": ["electronics", "computers"],

"ratings": [5, 4, 5]

},

{

"name": "Smartphone",

"brand": "Samsung",

"price": 800,

"inStock": true,

"categories": ["electronics", "mobiles"],

"ratings": [4, 4, 4]

},

{

"name": "Headphones",

"brand": "Sony",

"price": 200,

"inStock": false,

"categories": ["electronics", "audio"],

"ratings": [5, 5, 4, 4]

}

]);

**Insert Customers**

db.customers.insertMany([

{

"name": "Alice Johnson",

"email": "alice@example.com",

"age": 28,

"purchases": [

{ "product": "Laptop", "price": 1200, "date": "2023-11-15" }

]

},

{

"name": "Bob Smith",

"email": "bob@example.com",

"age": 32,

"purchases": [

{ "product": "Smartphone", "price": 800, "date": "2024-01-05" }

]

}

]);

**Step 3: Reading (Querying) Data**

**Retrieve All Products**

db.products.find();

**Find Products by Condition**

* Find all products that are **in stock**:

db.products.find({ "inStock": true });

* Find all products with a price **greater than 500**:

db.products.find({ "price": { "$gt": 500 } });

* Find products belonging to the **"electronics"** category:

db.products.find({ "categories": "electronics" });

**Retrieve Specific Fields**

* Only show product name and price, excluding \_id:

db.products.find({}, { "name": 1, "price": 1, "\_id": 0 });

**Sort and Limit Results**

* Sort products by price **descending** and limit to **2 results**:

db.products.find().sort({ "price": -1 }).limit(2);

**Step 4: Updating Documents**

**Update a Single Document**

* Change the price of the **Laptop**:

db.products.updateOne(

{ "name": "Laptop" },

{ "$set": { "price": 1100 } }

);

**Update Multiple Documents**

* Set inStock to **true** for all products in the **"audio"** category:

db.products.updateMany(

{ "categories": "audio" },

{ "$set": { "inStock": true } }

);

**Add a New Field to Documents**

* Add a new field discount for all products:

db.products.updateMany(

{},

{ "$set": { "discount": 0 } }

);

**Step 5: Deleting Documents**

**Delete a Single Document**

* Remove the product named **Headphones**:

db.products.deleteOne({ "name": "Headphones" });

**Delete Multiple Documents**

* Remove all products with price **less than 500**:

db.products.deleteMany({ "price": { "$lt": 500 } });

**Remove All Documents from a Collection**

db.products.deleteMany({});

**Step 6: Advanced Querying**

**Using Logical Operators**

* Find products that are either from **Apple** or cost **more than 800**:

db.products.find({

"$or": [

{ "brand": "Apple" },

{ "price": { "$gt": 800 } }

]

});

**Querying Embedded Documents**

* Find customers who purchased a **Laptop**:

db.customers.find({ "purchases.product": "Laptop" });

**Using Array Operators**

* Find products with an average rating of **4 or higher**:

db.products.find({

"ratings": { "$elemMatch": { "$gte": 4 } }

});

**Step 7: Working with Cursors**

**Iterating Through Results Using Cursors**

var cursor = db.products.find();

while (cursor.hasNext()) {

printjson(cursor.next());

}

* This prints each product as a JSON object.

**Step 8: Practical Examples**

**Example 1: Product Search with Sorting and Limiting**

* Display the top 3 most expensive products:

db.products.find().sort({ "price": -1 }).limit(3);

**Example 2: Complex Customer Query**

* Find customers who bought a product in **2024**:

db.customers.find({

"purchases.date": { "$regex": "^2024" }

});

**Example 3: Updating Nested Fields**

* Add a new purchase to **Alice Johnson's** record:

db.customers.updateOne(

{ "name": "Alice Johnson" },

{ "$push": { "purchases": { "product": "Headphones", "price": 200, "date": "2024-02-10" } } }

);