

Rajveer Singh

Computer System Technician, Loyalist College

COMP1031: Introduction to Data Communication

Aditya Saxena

Screenshots for portfolio:-

## 1. Creating MongoDB database

### 1.1. Code

```
use Ecommerce
```

### 1.2. Screenshot

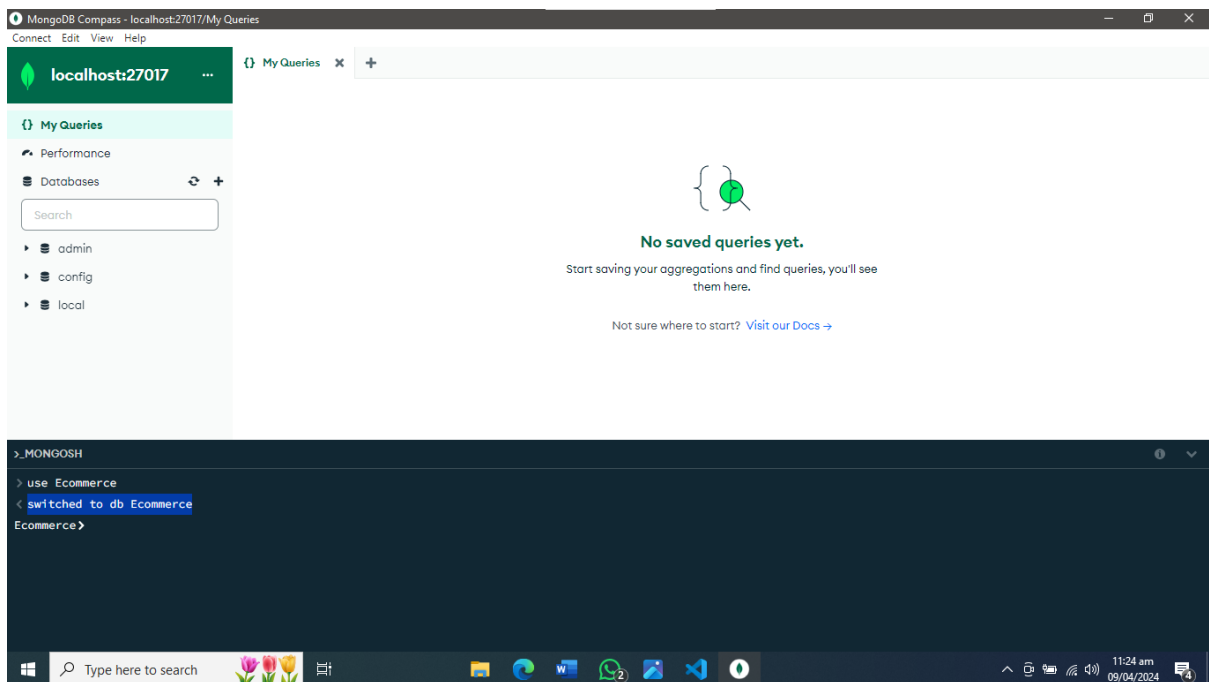


Figure 1: Creating a database

## 2. Creating the 5 collections using mongosh

### 2.1. Mongosh commands

```
db.createCollection("Products");

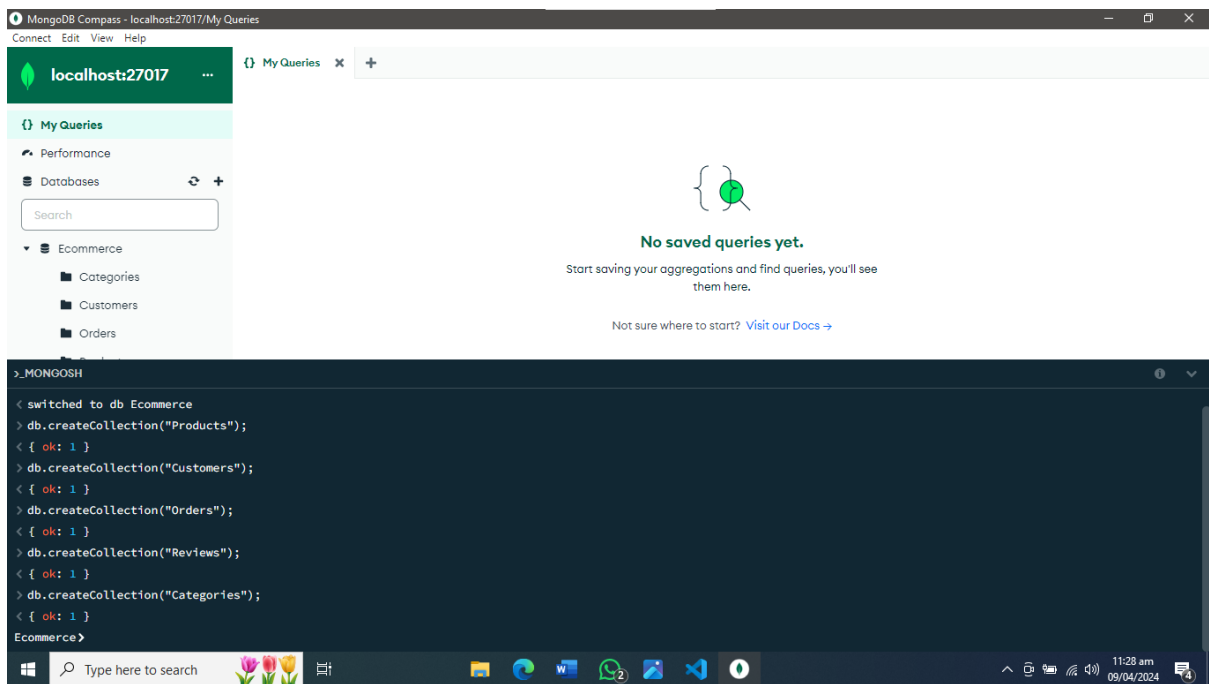
db.createCollection("Customers");

db.createCollection("Orders");

db.createCollection("Reviews");

db.createCollection("Categories");
```

### 2.2. Screenshot



## 3. Creating Document in Product Collection

### 3.1. Mongosh commands

```
const productData = {  
  name: "Laptop",  
  price: 70000,  
  brand: "DELL",  
  category: "Electronics"  
};  
db.Products.insertOne(productData);
```

### 3.2. Screenshots

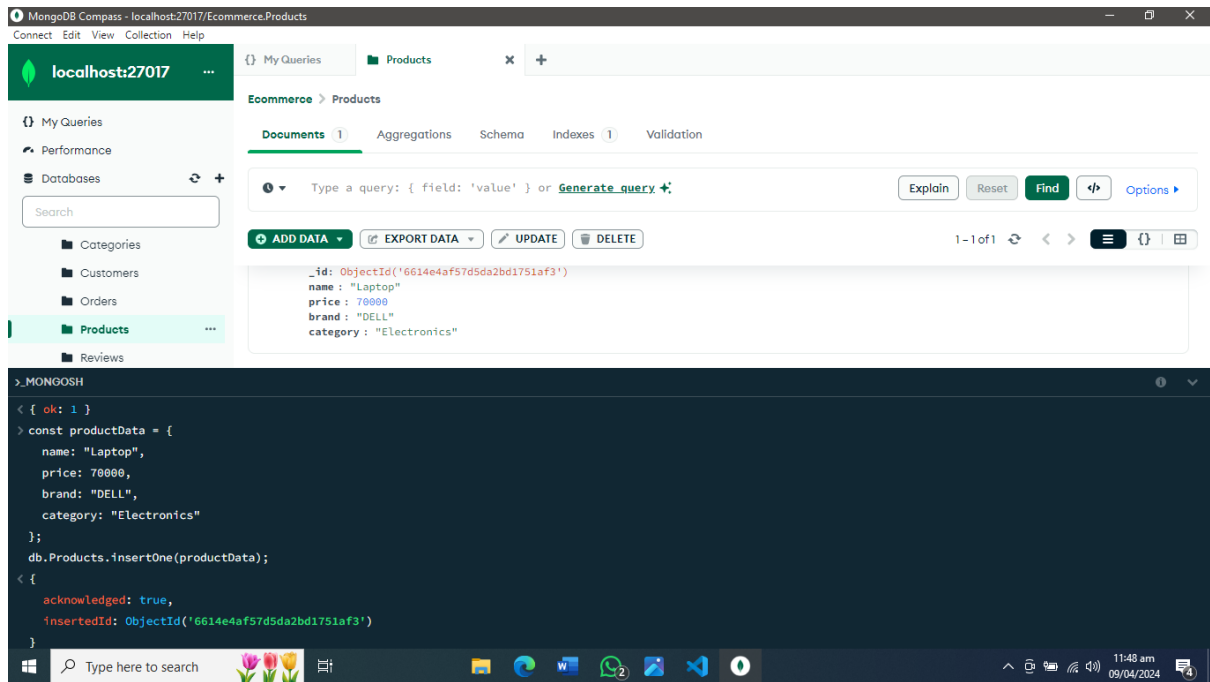


Figure 2: Creating the document in Product collection using `insertOne()`

## 4. Creating 2 documents in Products Collection

### 4.1. Mongosh commands

```
const customersData = [
  {
    name: "Doe",
    email: "doe@gamil.com",
    age: 20
  },
  {
    name: "Jane",
    email: "jane@gmail.com",
    age: 25
  }
];

db.Customers.insertMany(customersData);
```

### 4.2. Screenshot

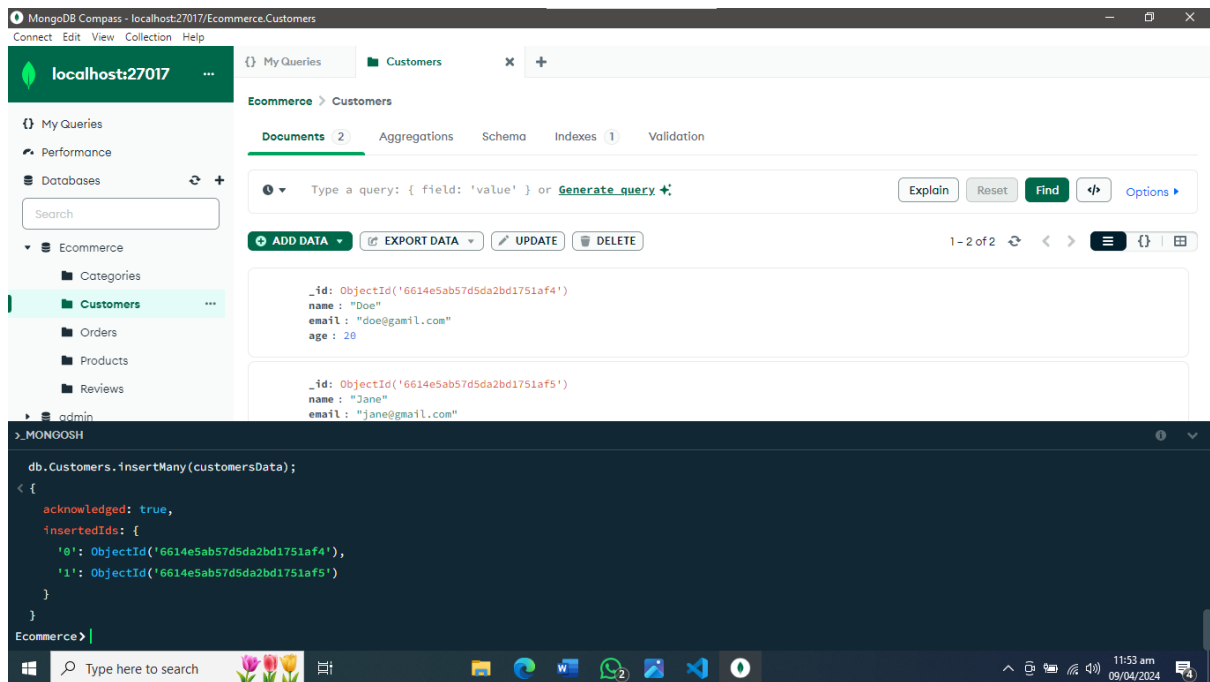


Figure 3: Creating 2 documents in Customers Collection using insertMany()

## 5. Populating collections with 10 documents

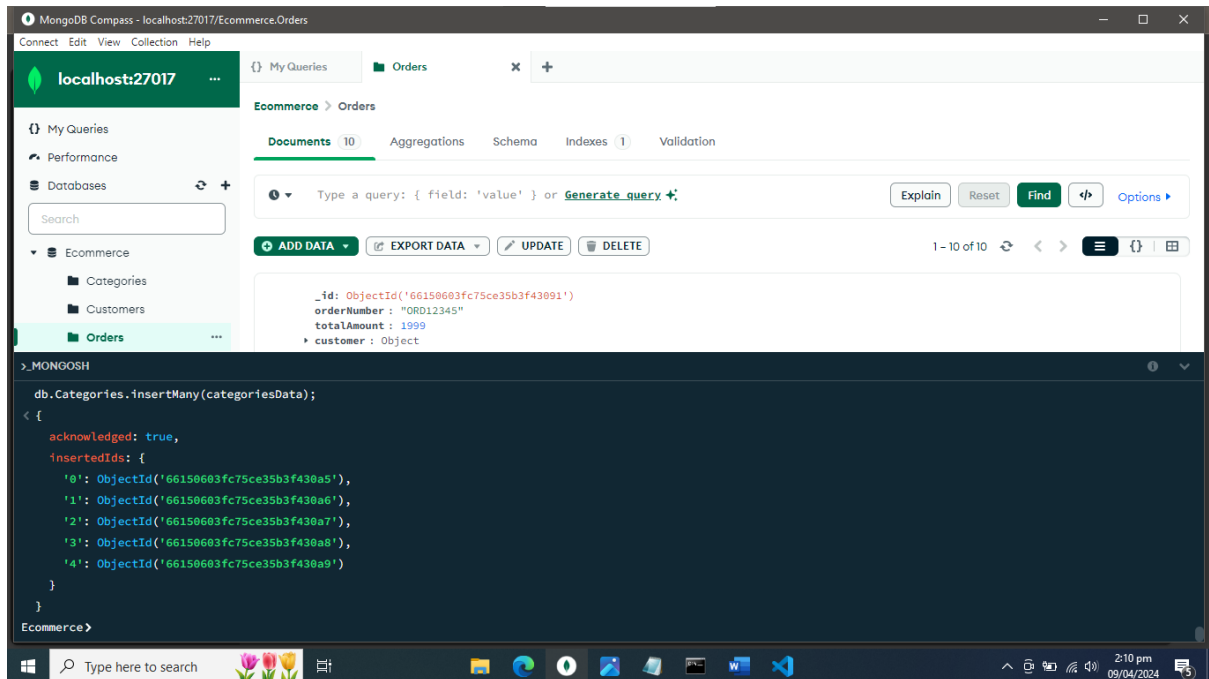


Figure 4: Populating collections with 10 documents each

## 6. Update document using updateOne()

Screenshot

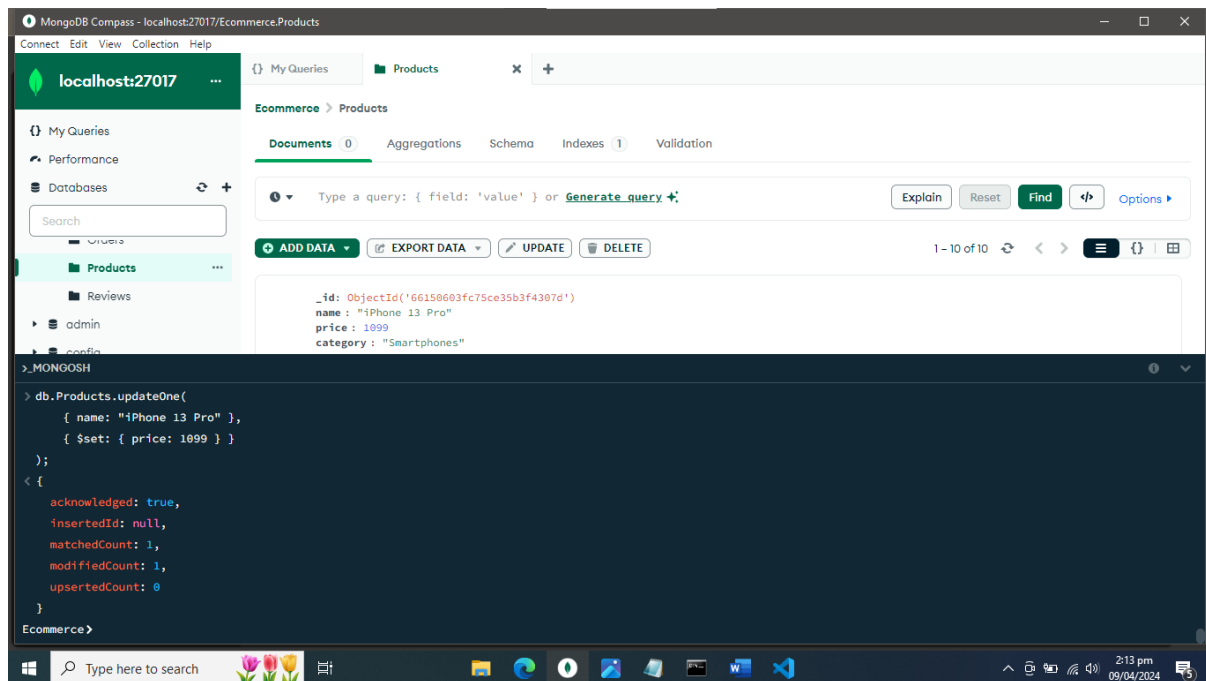


Figure 5: Updating document using updateOne() in mongosh

## 7. Update multiple documents using updateMany()

Screenshot

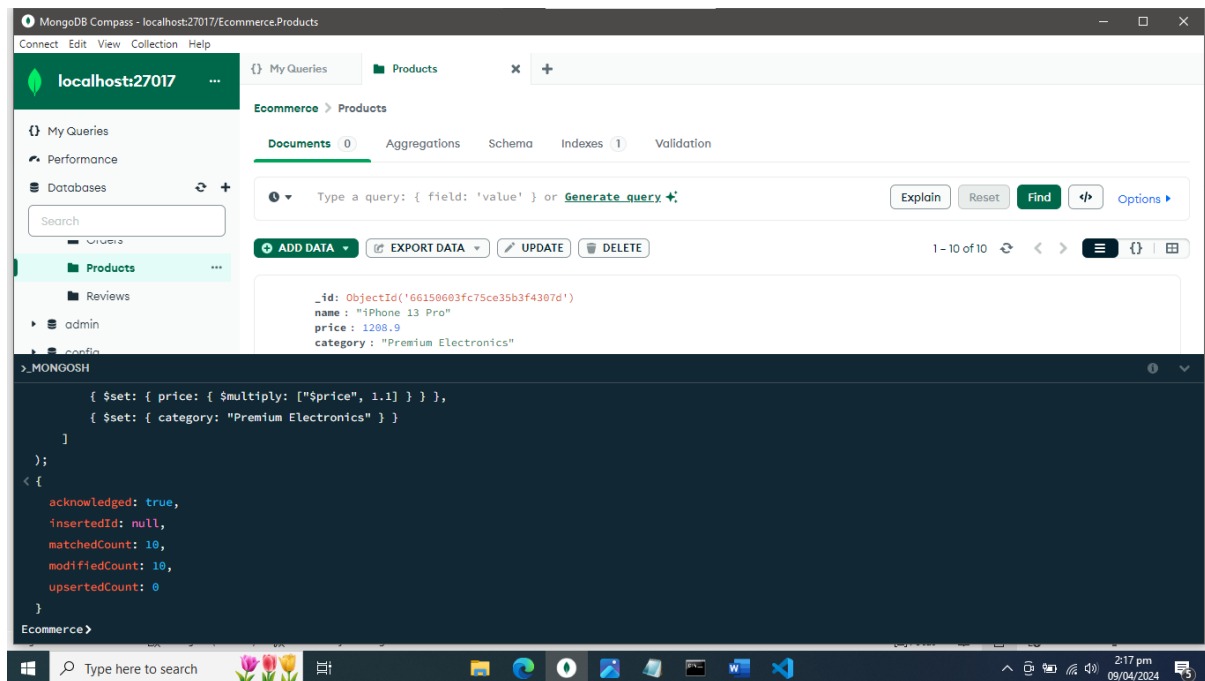


Figure 6: updating multiple documents using updateMany() command

## 8. Reading One document

### Screenshot

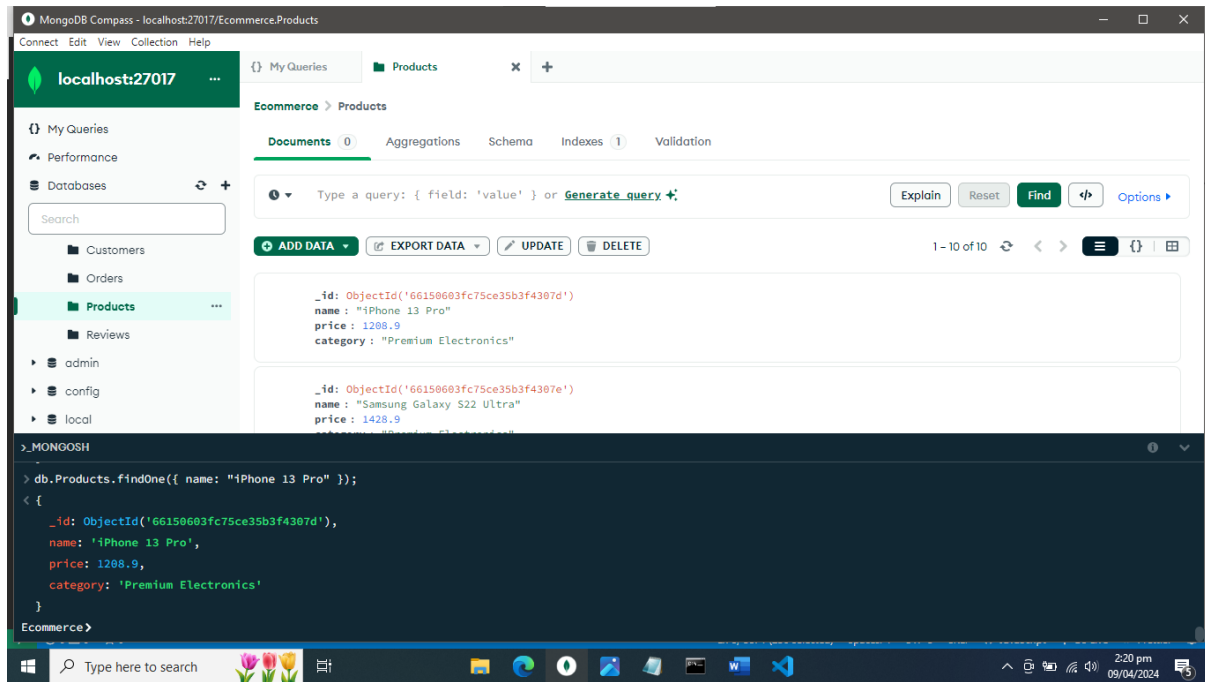
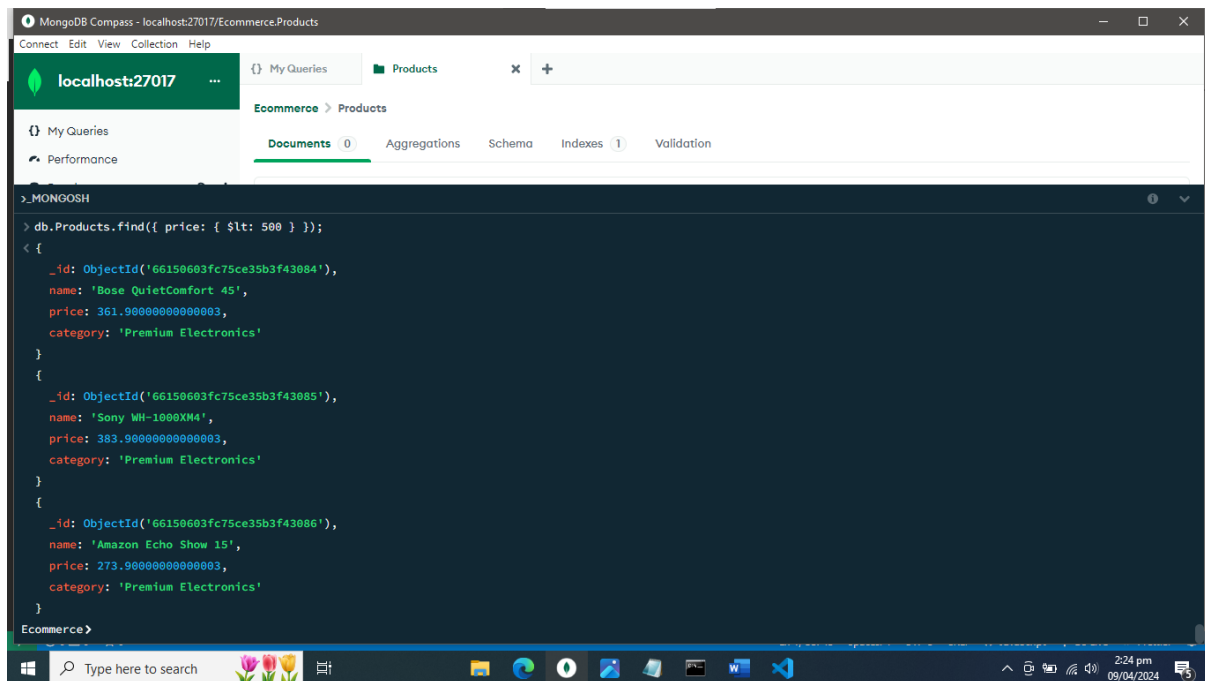


Figure 7: Reading One document using `findOne` in mongosh



## 9. Read Multiple documents using find()

### Screenshot



The screenshot shows the MongoDB Compass interface with the 'Products' collection selected. The 'Documents' tab is active, and the mongosh terminal displays the result of a find() query. The query filters documents where the price is less than 500. The results show three documents, each with an \_id, name, price, and category.

```
> db.Products.find({ price: { $lt: 500 } });
< {
  _id: ObjectId('66150603fc75ce35b3f43084'),
  name: 'Bose QuietComfort 45',
  price: 361.90000000000003,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43085'),
  name: 'Sony WH-1000XM4',
  price: 383.90000000000003,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43086'),
  name: 'Amazon Echo Show 15',
  price: 273.90000000000003,
  category: 'Premium Electronics'
}
Ecommerce>
```

Figure 8: Read Multiple documents using find() in mongosh

## 10. Delete One Document

### Screenshot

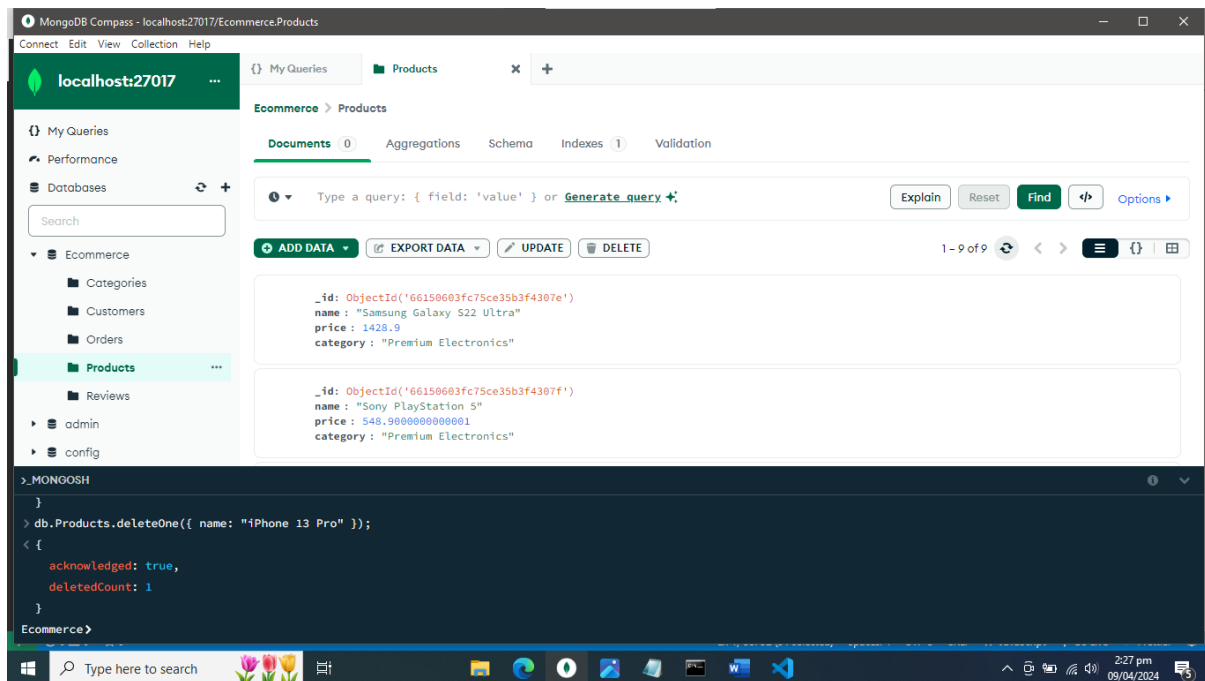


Figure 9: Delete One document using deleteOne() in mongosh

## 11. Delete Multiple Documents

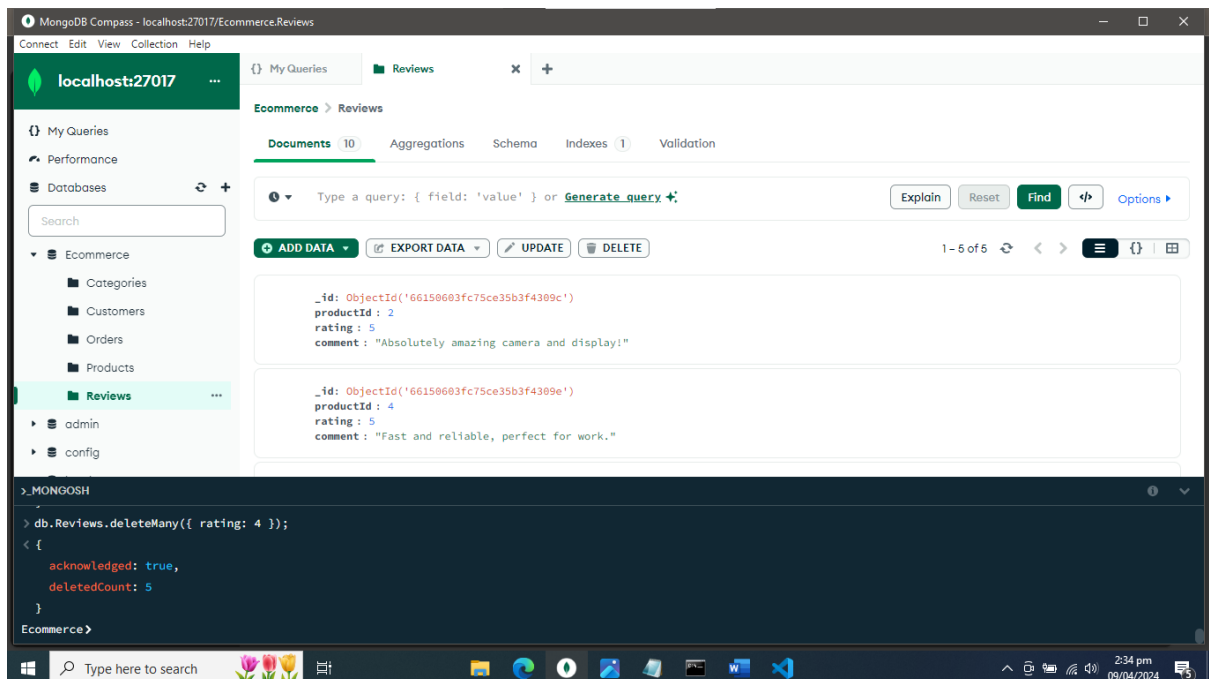
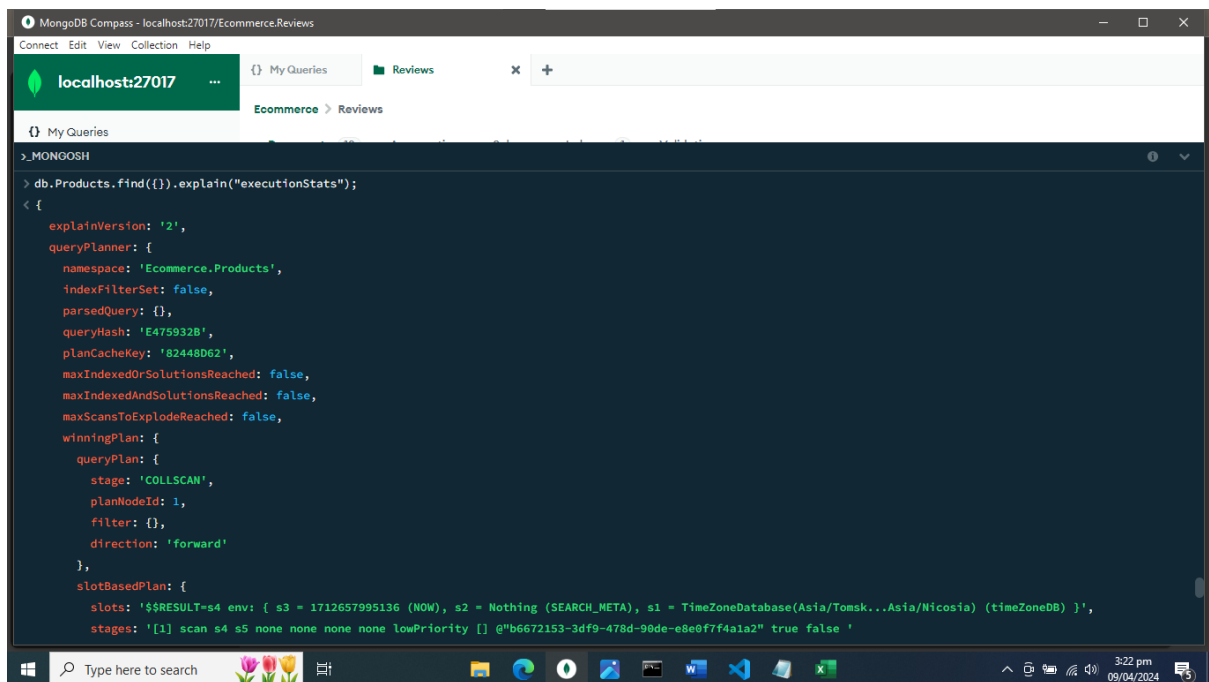


Figure 10: Deleting multiple documents in mongosh

## 12. Execution stats

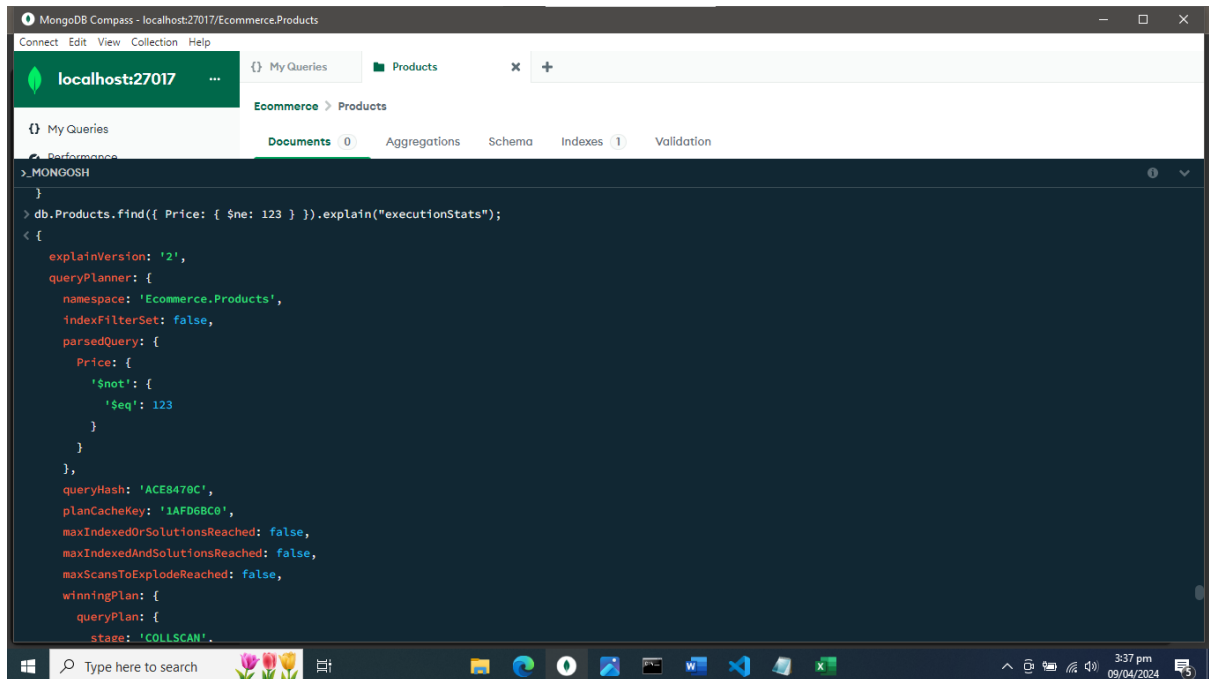


The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'localhost:27017' and the database 'Ecommerce.Reviews'. The left sidebar shows the 'My Queries' tab. The main area displays the command `db.Products.find().explain("executionStats");` and its output, which is a JSON document detailing the query execution plan and statistics.

```
> db.Products.find().explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'Ecommerce.Products',
    indexFilterSet: false,
    parsedQuery: {},
    queryHash: 'E475932B',
    planCacheKey: '82448062',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'COLLSCAN',
        planNodeId: 1,
        filter: {},
        direction: 'forward'
      },
      slotBasedPlan: {
        slots: '$$RESULT=s4 env: { s3 = 1712657995136 (NOW), s2 = Nothing (SEARCH_META), s1 = TimeZoneDatabase(Asia/Tomsk...Asia/Moscow) (timeZoneDB) }',
        stages: '[1] scan s4 s5 none none none none lowPriority [] @"b6672153-3df9-478d-90de-e8e0f7f4a1a2" true false '
```

Figure 11: Execution stats of first query

## 12.1. Find documents where a category field equals Electronics - Query 2



The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'localhost:27017/Ecommerce.Products'. The left sidebar shows 'My Queries' and 'Products'. The main panel displays the 'Documents' tab for the 'Products' collection. The query editor shows the following command:

```
> db.Products.find({ Price: { $ne: 123 } }).explain("executionStats");
```

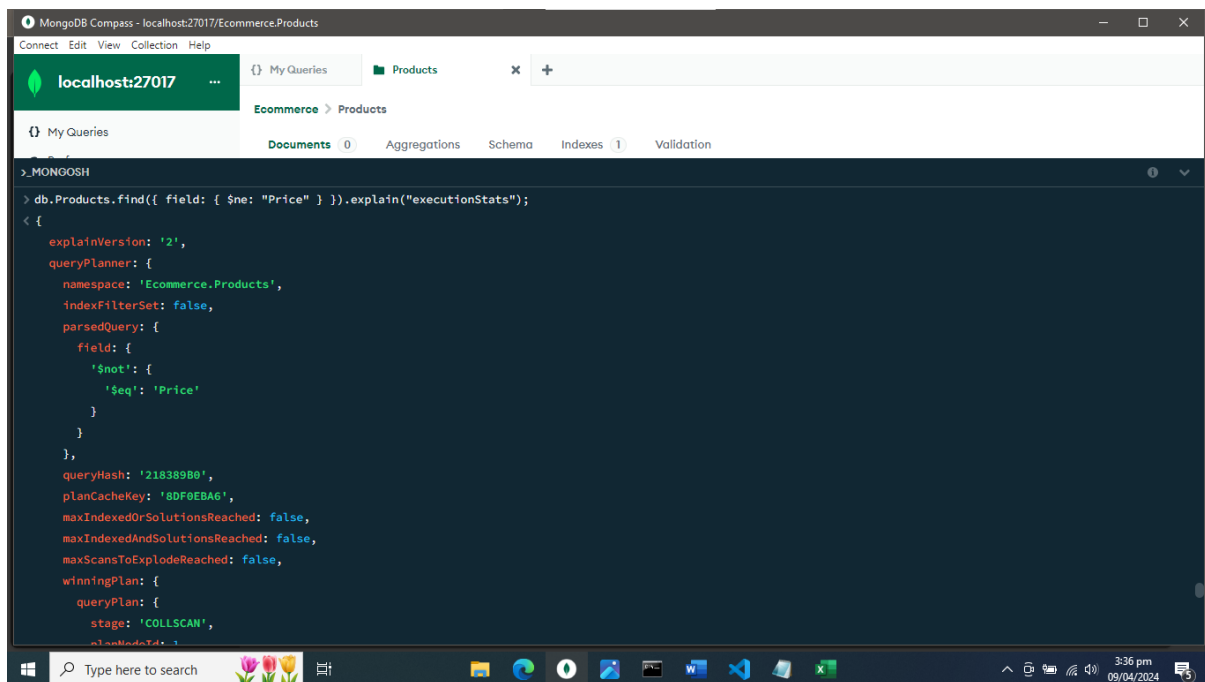
The output shows the execution statistics for the query:

```
< {
  explainVersion: '2',
  queryPlanner: {
    namespace: 'Ecommerce.Products',
    indexFilterSet: false,
    parsedQuery: {
      Price: {
        '$not': {
          '$eq': 123
        }
      }
    },
    queryHash: 'ACE8479C',
    planCacheKey: '1AFD68C0',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'COLLSCAN',
```

The Windows taskbar at the bottom shows the time as 3:37 pm on 09/04/2024.

Figure 12: Execution stats for second query

## 12.2. Find documents where a Price field does not equal 123 – Query 3



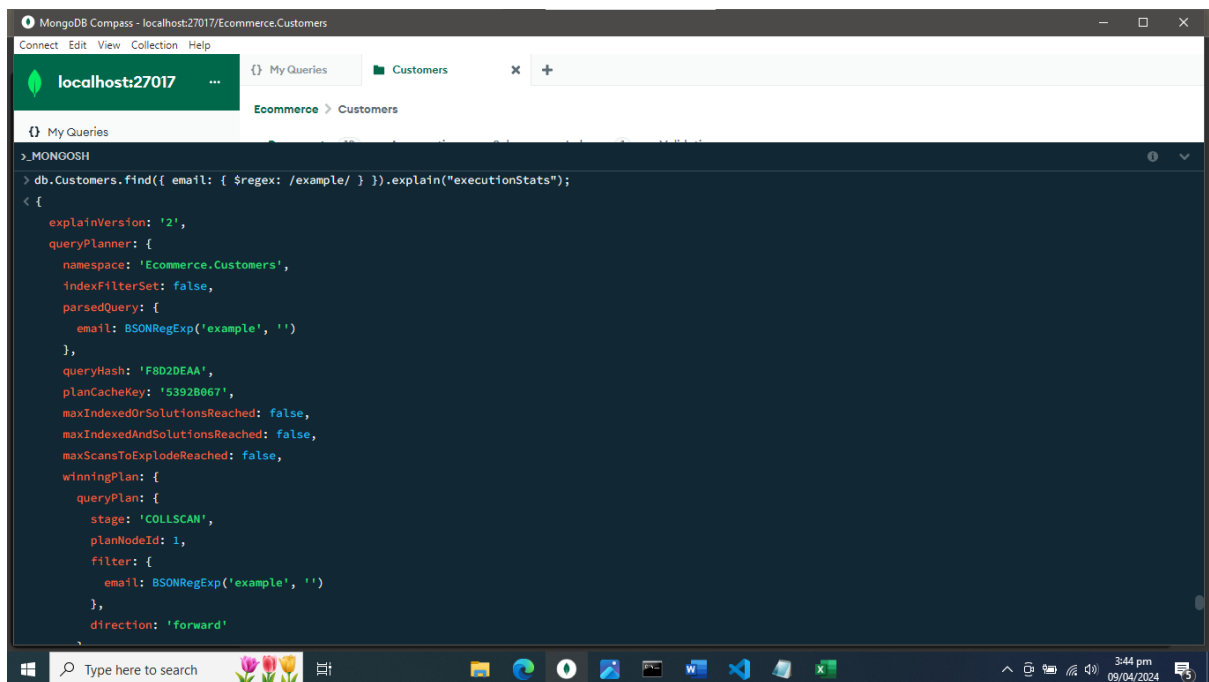
The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'localhost:27017' and the database 'Ecommerce'. The 'Products' collection is selected. The 'My Queries' tab is active, showing a query: `db.Products.find({ field: { $ne: "Price" } }).explain("executionStats");`. The 'Documents' tab is selected, showing the execution statistics for the query. The statistics are as follows:

```
> MONGODB
> db.Products.find({ field: { $ne: "Price" } }).explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'Ecommerce.Products',
    indexFilterSet: false,
    parsedQuery: {
      field: {
        '$not': {
          '$eq': 'Price'
        }
      }
    },
    queryHash: '218389B0',
    planCacheKey: '8DF0EBA6',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'COLLSCAN',

```

Figure 13: Execution stats for 3rd query

### 12.3. Find documents where a email field matches a regular expression- Query 4



The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'localhost:27017/Ecommerce.Customers'. The left sidebar shows the 'Customers' collection. The main editor displays a MongoDB query: `db.Customers.find({ email: { $regex: /example/ } }).explain("executionStats");`. The output shows the execution statistics for this query, including the query plan, which is a COLLSCAN. The status bar at the bottom shows the time as 3:44 pm on 09/04/2024.

```
> MONGODB
> db.Customers.find({ email: { $regex: /example/ } }).explain("executionStats");
{
  explainVersion: '2',
  queryPlanner: {
    namespace: 'Ecommerce.Customers',
    indexFilterSet: false,
    parsedQuery: {
      email: BSONRegExp('example', '')
    },
    queryHash: 'F8D2DEAA',
    planCacheKey: '5392B067',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      queryPlan: {
        stage: 'COLLSCAN',
        planNodeId: 1,
        filter: {
          email: BSONRegExp('example', '')
        },
        direction: 'forward'
      },

```

Figure 14: Execution stats for 4th query

## 12.4. Find documents with a field value greater than a specific value.

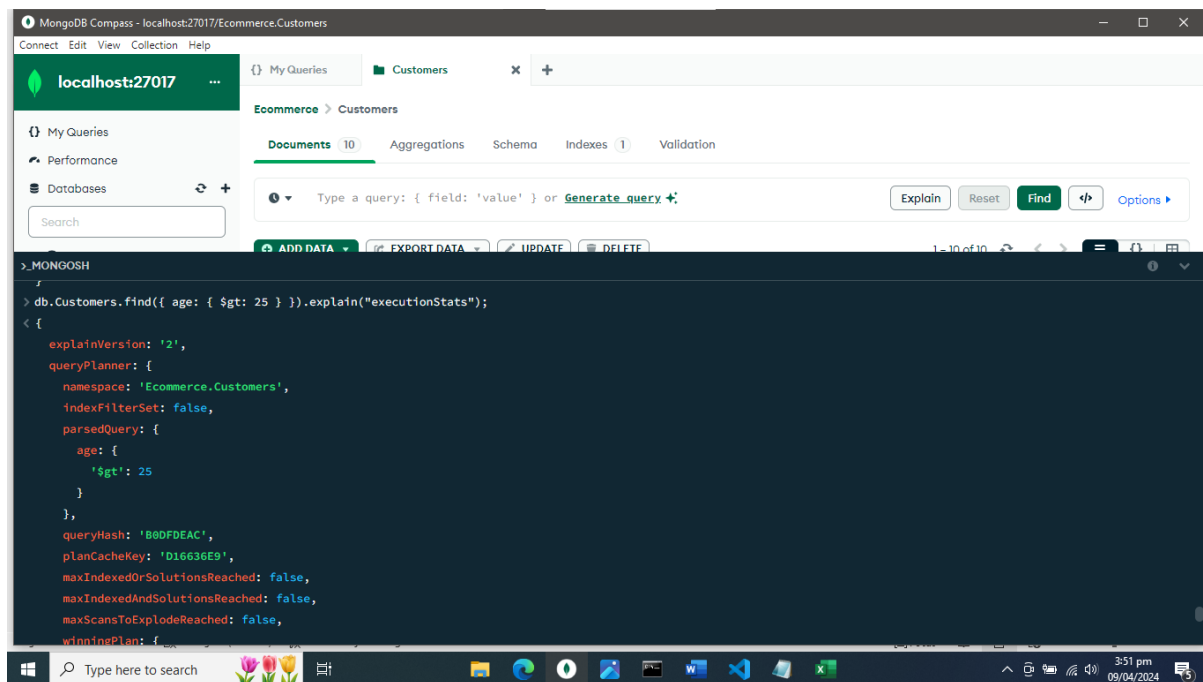


Figure 15: Execution stats for 5th query



## 13. Creating indexes based on the workload excel file

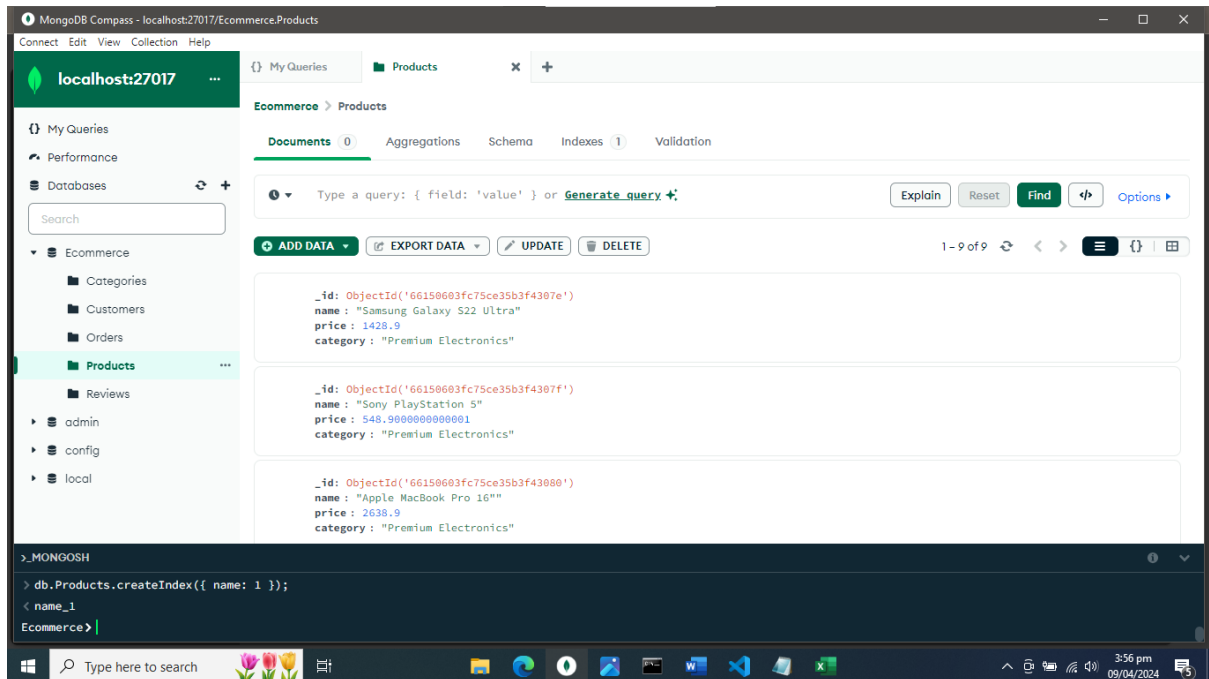


Figure 16: Creating indexes

## 14. Checking execution stats again

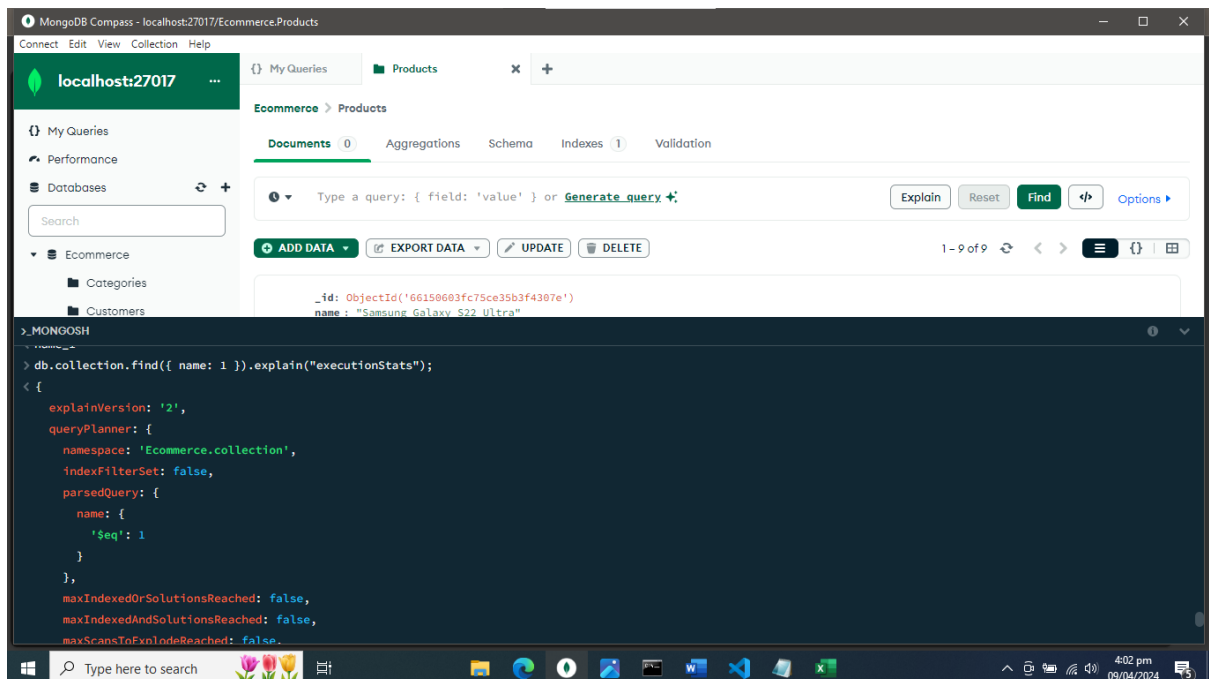


Figure 17: Checking execution states again

## 15. Mongodb aggregation functions

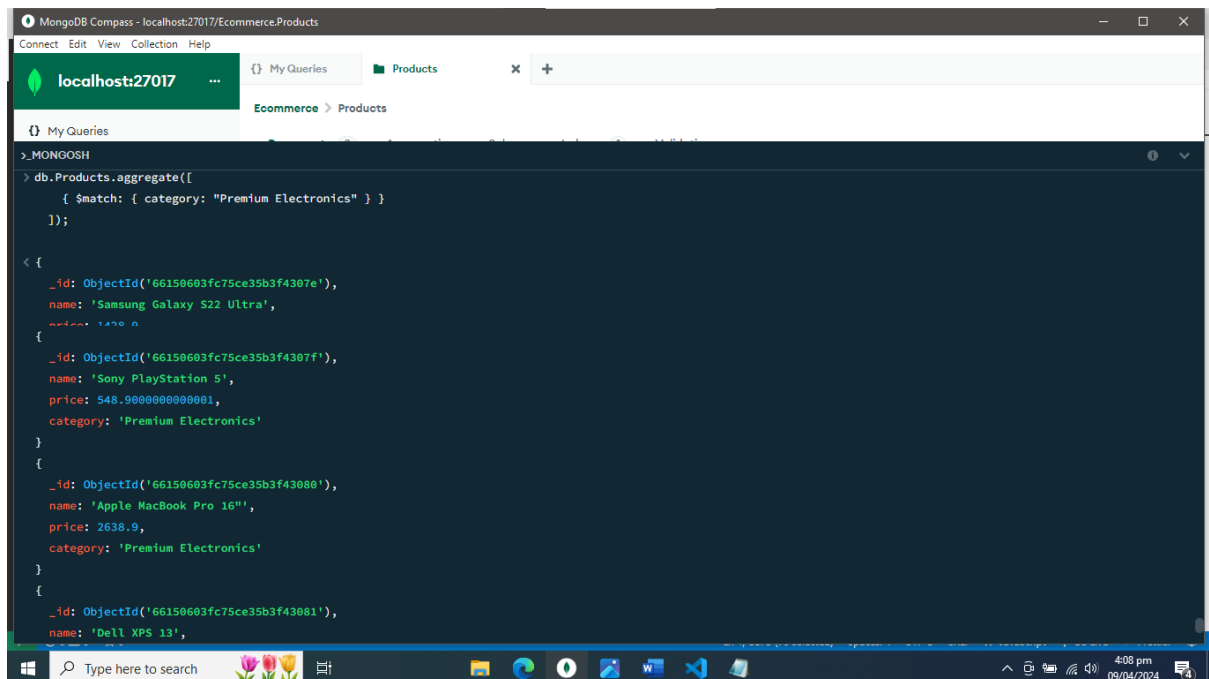


Figure 18: match aggregate function

### 15.1. group aggregation function

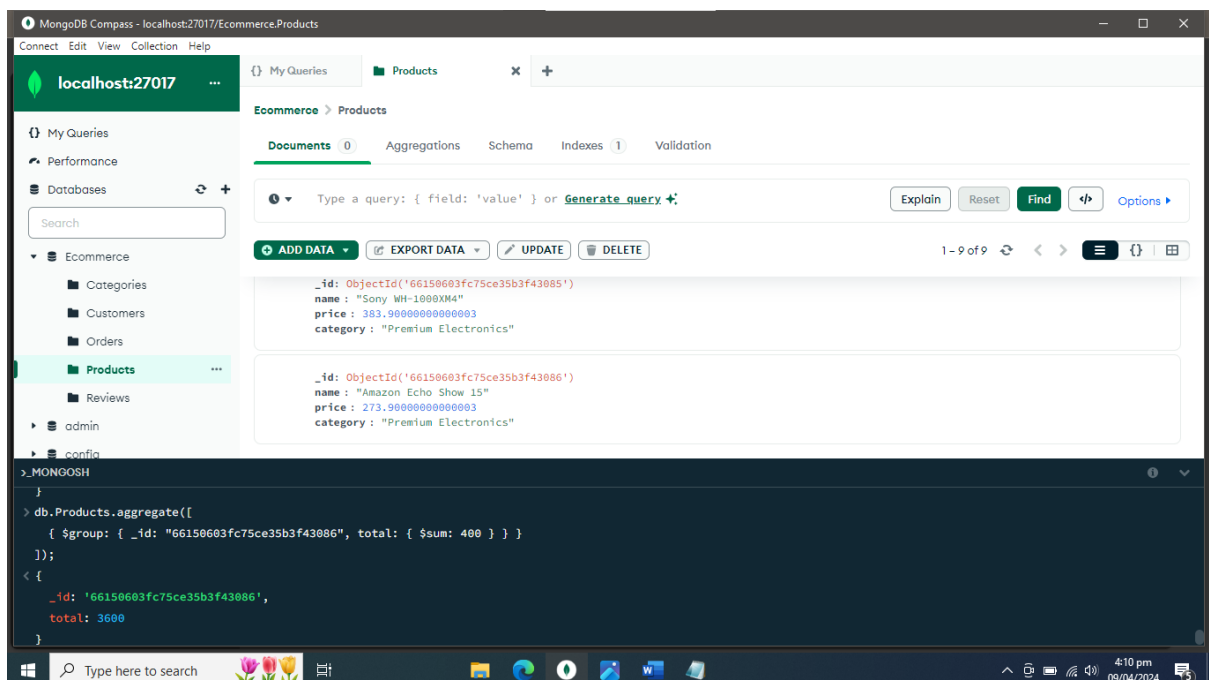
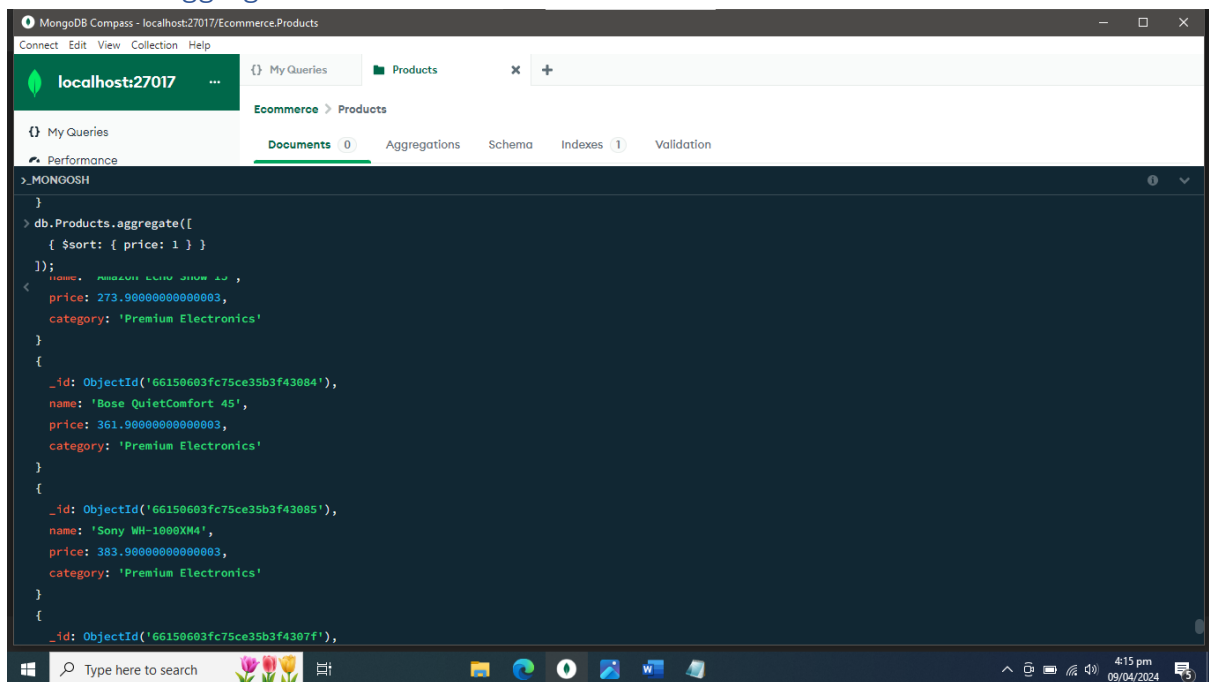


Figure 19: group aggregation function

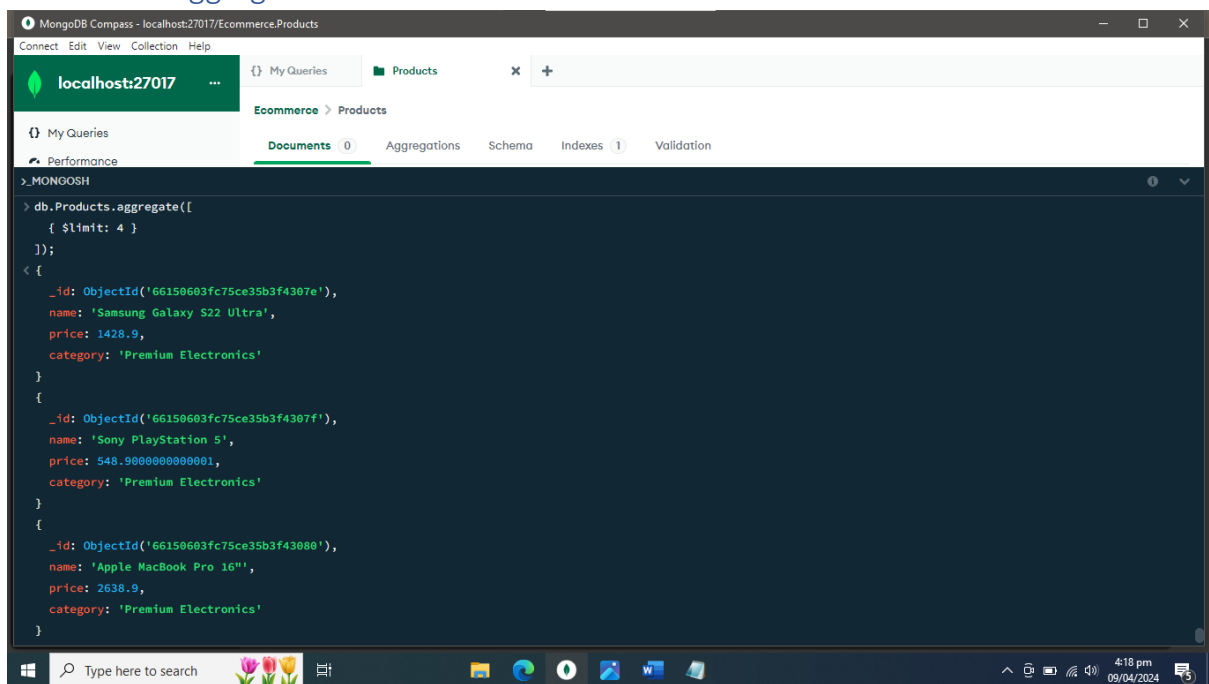
## 15.2. sort aggregation function



```
> MONGODB
>
> db.Products.aggregate([
  { $sort: { price: 1 } }
]);
{
  name: 'Amazon Echo Show 10',
  price: 273.90000000000003,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43084'),
  name: 'Bose QuietComfort 45',
  price: 361.90000000000003,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43085'),
  name: 'Sony WH-1000XM4',
  price: 383.90000000000003,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f4307f'),
  name: 'Apple MacBook Pro 16"',
  price: 2638.9,
  category: 'Premium Electronics'
}
```

Figure 20: sort aggregate function

## 15.3. limit aggregation function



```
> MONGODB
>
> db.Products.aggregate([
  { $limit: 4 }
]);
{
  _id: ObjectId('66150603fc75ce35b3f4307e'),
  name: 'Samsung Galaxy S22 Ultra',
  price: 1428.9,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f4307f'),
  name: 'Sony PlayStation 5',
  price: 548.90000000000001,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43080'),
  name: 'Apple MacBook Pro 16"',
  price: 2638.9,
  category: 'Premium Electronics'
}
{
  _id: ObjectId('66150603fc75ce35b3f43081'),
  name: 'Bose QuietComfort 45',
  price: 361.90000000000003,
  category: 'Premium Electronics'
}
```

Figure 21: limit aggregation function

## 15.4. project aggregation function

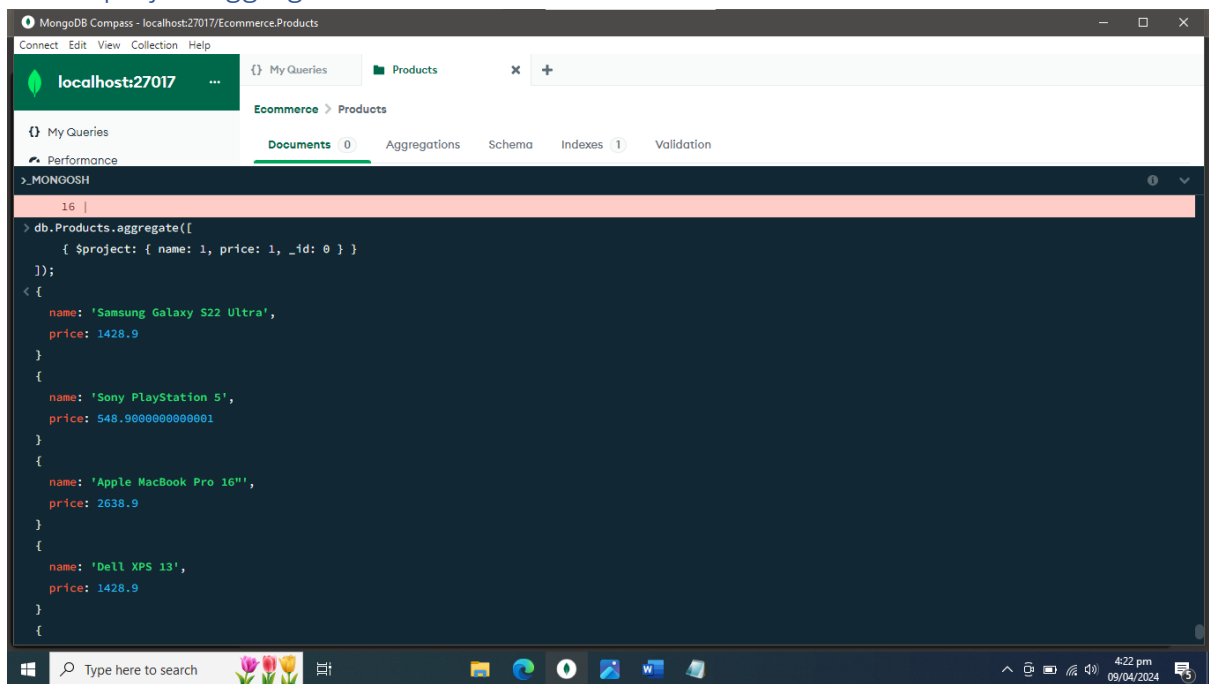


Figure 22: project aggregation function

## 15.5. count aggregation function

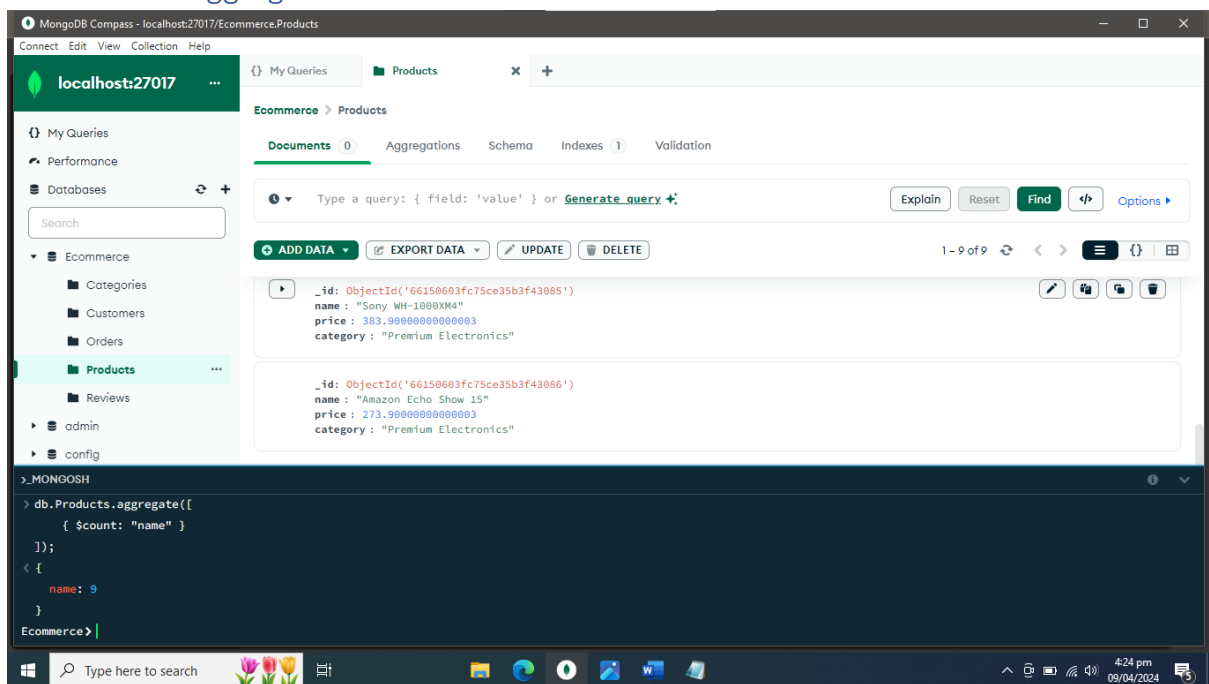


Figure 23: count aggregation function

## 15.6. sum aggregation function

The screenshot displays the MongoDB Compass interface for a local database at localhost:27017. The 'Ecommerce' database is selected, and the 'Products' collection is open. The 'Documents' tab is active, showing two product documents. Below the documents, the MongoDB Shell (MONGOSH) is open, showing an aggregation query and its output.

**Products Collection Documents:**

- `{ name: "Sony WH-1000XM4", price: 383.90000000000003, category: "Premium Electronics" }`
- `{ _id: ObjectId('66150603fc75ce35b3f43086'), name: "Amazon Echo Show 15", price: 273.90000000000003, category: "Premium Electronics" }`

**MongoDB Shell (MONGOSH) Query and Output:**

```
> db.Products.aggregate([
  { $group: { _id: null, totalPrice: { $sum: "$price" } } }
]);
< {
  _id: null,
  totalPrice: 14653.1
}
```

The output shows the result of the aggregation query, which groups the products by their category and calculates the total price for each group. The result is a single document with `_id: null` and `totalPrice: 14653.1`.

Figure 24: insert aggregation function

## 16. MongoDB aggregation pipelines

### 16.1. Calculate Average Price by Category- pipeline 1

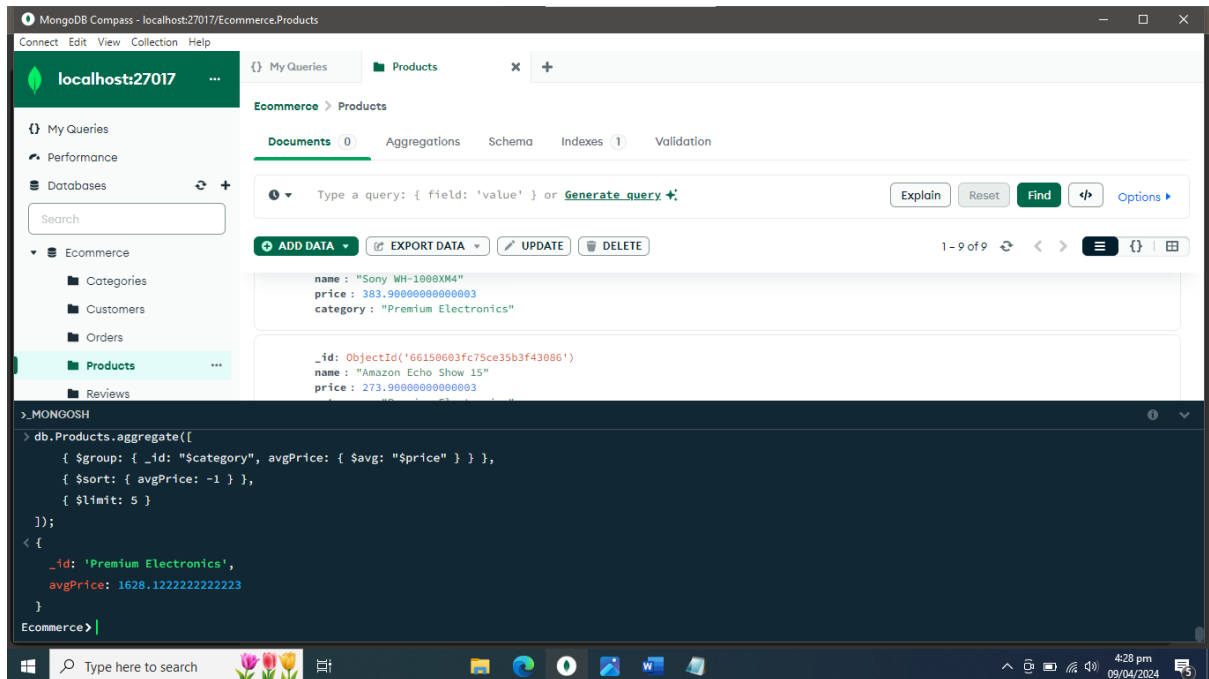


Figure 25: average price by category pipeline

### 16.2. Count Products by Category- pipeline 2

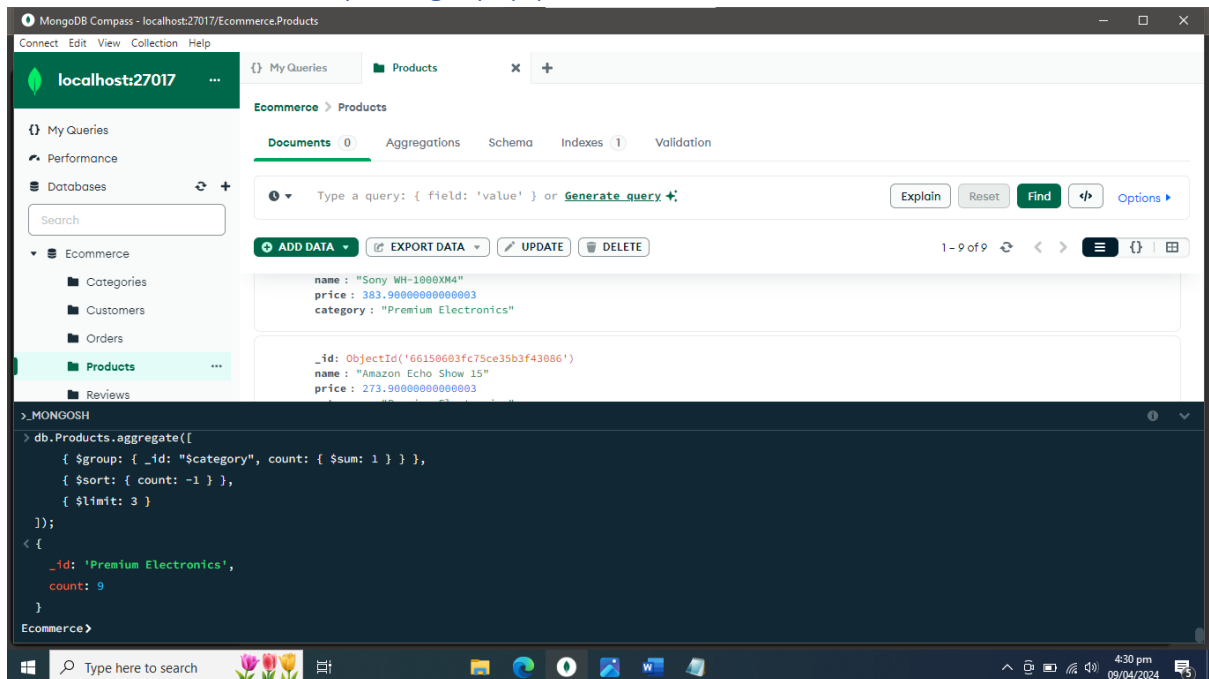


Figure 26: Count products by category - pipeline 2

## 16.3. Calculate Total Revenue - pipeline 3

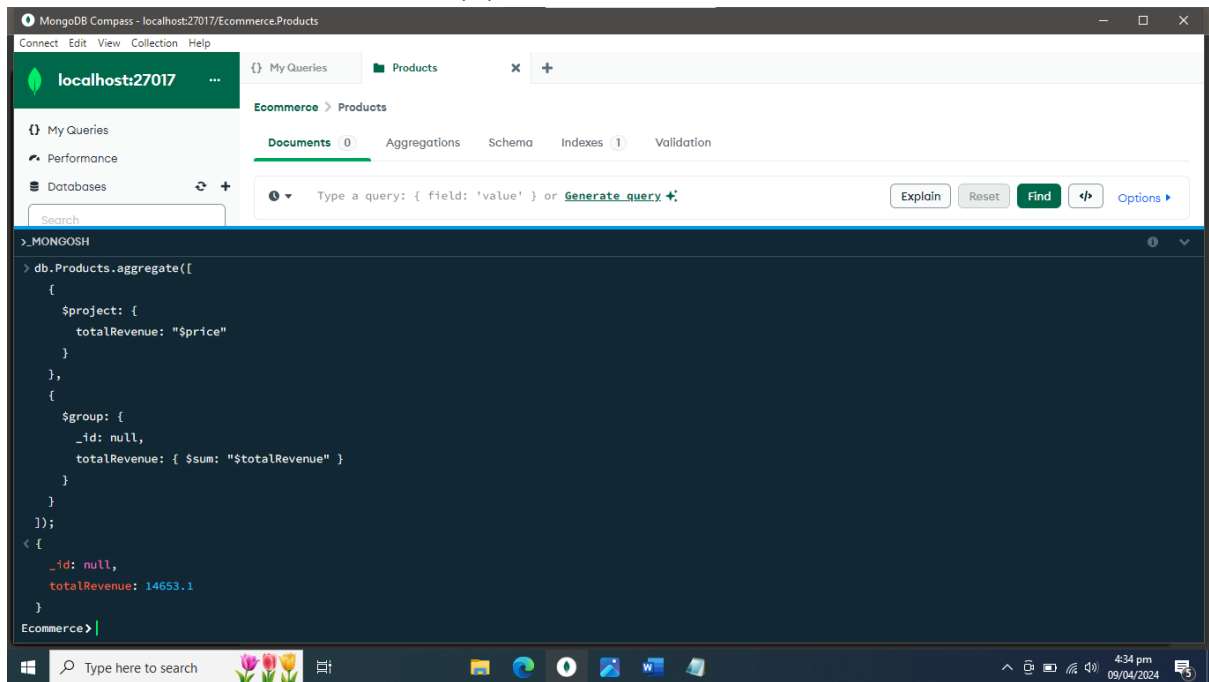


Figure 27: calculate total revenue - pipeline 3