Understanding Indexes in MongoDB: A Beginner's Guide

What are Indexes?

Imagine you have a library with thousands of books scattered on shelves. When someone asks for a specific book, you would typically search through each shelf until you find it. In a database like MongoDB, indexes act like a library catalog. They help MongoDB quickly find the data you're looking for without scanning every document.

Indexes in MongoDB are special data structures that store a small portion of the data set in an easy-to-traverse form. They improve the speed of data retrieval operations on a MongoDB database by reducing the number of documents MongoDB needs to scan to find the required documents. Understanding how indexes work and when to use them is crucial for optimizing query performance in MongoDB.

Types of indexes in mongodb:

- Single Field Index
- Compound Index
- Multikey Index
- Geospatial Index
- Text Indexes
- Hashed Indexes
- Clustered Indexes

Creating Indexes

Indexes can be created using the `createIndex()` method in MongoDB. Index creation can be performed either using the

MongoDB shell or within your application code. Here's how you can create an index in the MongoDB shell:

db.collection.createIndex({ field: 1 });

Index Strategies

When designing indexes for MongoDB, consider the following strategies:

- **Equality Queries**: Index fields used in queries with equality operators ($eq).
- **Range Queries**: Index fields used in queries with range operators ($gt, $lt).
- **Sort Queries**: Index fields used in queries with sort operations.

Monitoring and Maintaining Indexes

MongoDB provides various methods to monitor and maintain indexes:

- db.collection.getIndexes(): Lists all indexes on a collection.
- db.collection.dropIndex(): Removes an index from a collection.
- Index Intersection: MongoDB can use multiple indexes to fulfill a query.

Now, Let's walk through an example where we create a database, a collection, insert some documents, and then demonstrate how indexes work in MongoDB.

Step-by-Step Example: Using Indexes in MongoDB

Step 1: Setup MongoDB Environment

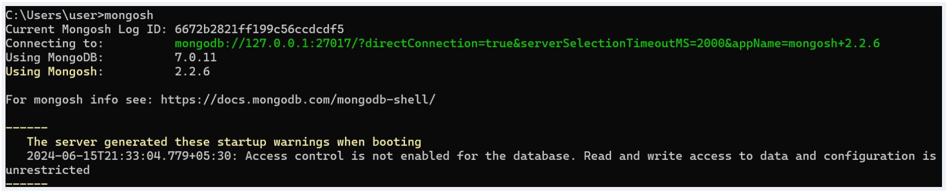
First, ensure MongoDB is installed and running on your system. You can start the MongoDB server using the following command:

>mongod

Step 2: Connect to MongoDB

Open a new terminal or command prompt window and connect to MongoDB using the MongoDB shell:

>mongosh



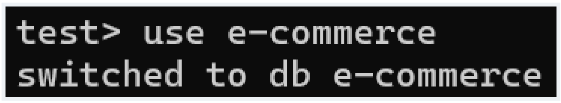
```
C:\Users\user>mongosh
Current Mongosh Log ID: 6672b2821ff199c56ccdcdf5
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.2.6
Using MongoDB:          7.0.11
Using Mongosh:          2.2.6

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

------
   The server generated these startup warnings when booting
   2024-06-15T21:33:04.779+05:30: Access control is not enabled for the database. Read and write access to data and configuration is
unrestricted
------
```

Step 3: Create a Database and Collection

Let's create a database called e-commerce and a collection called products where we will store information about products:

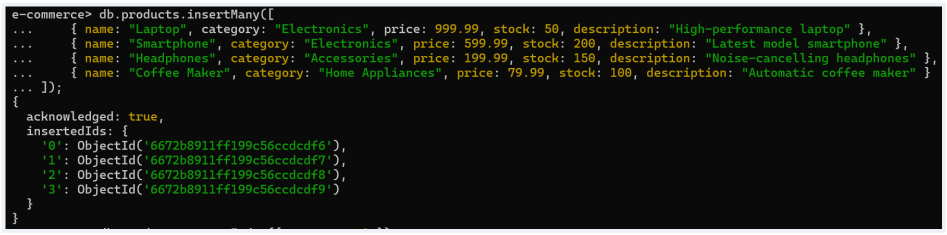
// Switch to 'library' database (creates if not exists)

>use e-commerce



```
test> use e-commerce
switched to db e-commerce
```

Insert Sample Data in the collection as shown below:



```
e-commerce> db.products.insertMany([
...     { name: "Laptop", category: "Electronics", price: 999.99, stock: 50, description: "High-performance laptop" },
...     { name: "Smartphone", category: "Electronics", price: 599.99, stock: 200, description: "Latest model smartphone" },
...     { name: "Headphones", category: "Accessories", price: 199.99, stock: 150, description: "Noise-cancelling headphones" },
...     { name: "Coffee Maker", category: "Home Appliances", price: 79.99, stock: 100, description: "Automatic coffee maker" }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6672b8911ff199c56ccdcdf6'),
    '1': ObjectId('6672b8911ff199c56ccdcdf7'),
    '2': ObjectId('6672b8911ff199c56ccdcdf8'),
    '3': ObjectId('6672b8911ff199c56ccdcdf9')
  }
}
```

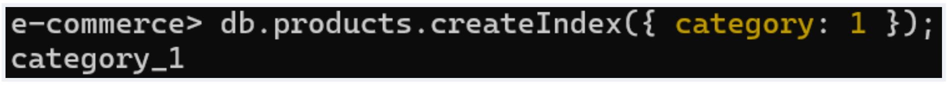
Step 4:

Create Indexes Based on Queries

Index on Category:

If you frequently query products by category, create an index on the category field

>db.products.createIndex({ category: 1 });

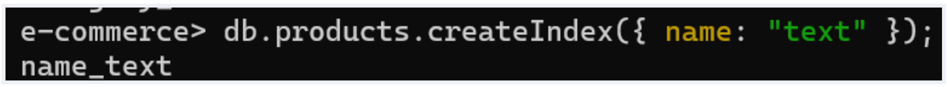


```
e-commerce> db.products.createIndex({ category: 1 });
category_1
```

Index on Name for Search:

If you have a search functionality to find products by name, create a text index on the name field.

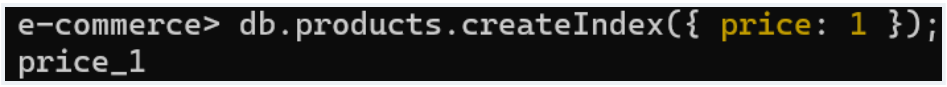
>db.products.createIndex({ name: "text" });



```
e-commerce> db.products.createIndex({ name: "text" });
name_text
```

Index for Sorting by Price:

>db.products.createIndex({ price: 1 });



```
e-commerce> db.products.createIndex({ price: 1 });
price_1
```

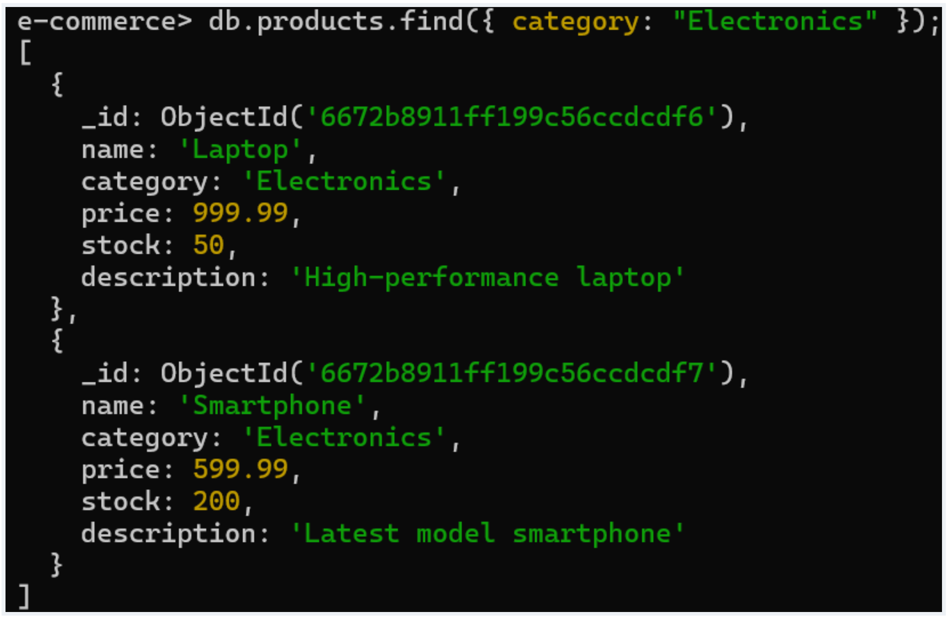
Now, its time to use queries

Use Queries Efficiently for the above created indexes:

Find Products by Category:

>db.products.find({ category: "Electronics" });

(this should execute and provide the details with the category electronics as shown below)

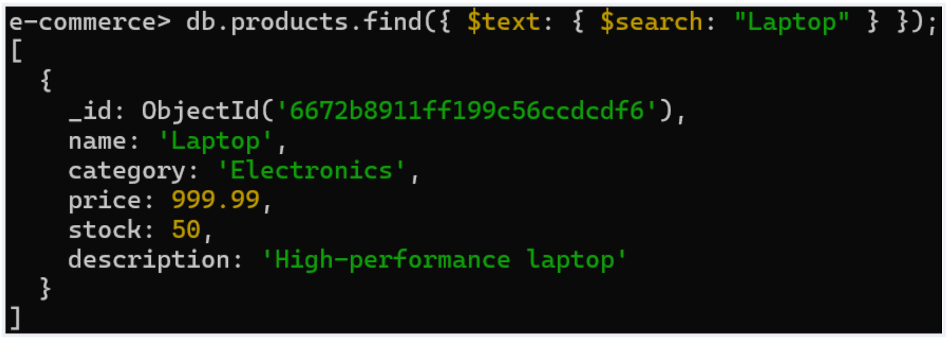


```
e-commerce> db.products.find({ category: "Electronics" });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf6'),
    name: 'Laptop',
    category: 'Electronics',
    price: 999.99,
    stock: 50,
    description: 'High-performance laptop'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf7'),
    name: 'Smartphone',
    category: 'Electronics',
    price: 599.99,
    stock: 200,
    description: 'Latest model smartphone'
  }
]
```

This query will use the index on category and be much faster than scanning the entire collection.

Search Products by Name:

db.products.find({ $text: { $search: "Laptop" } });



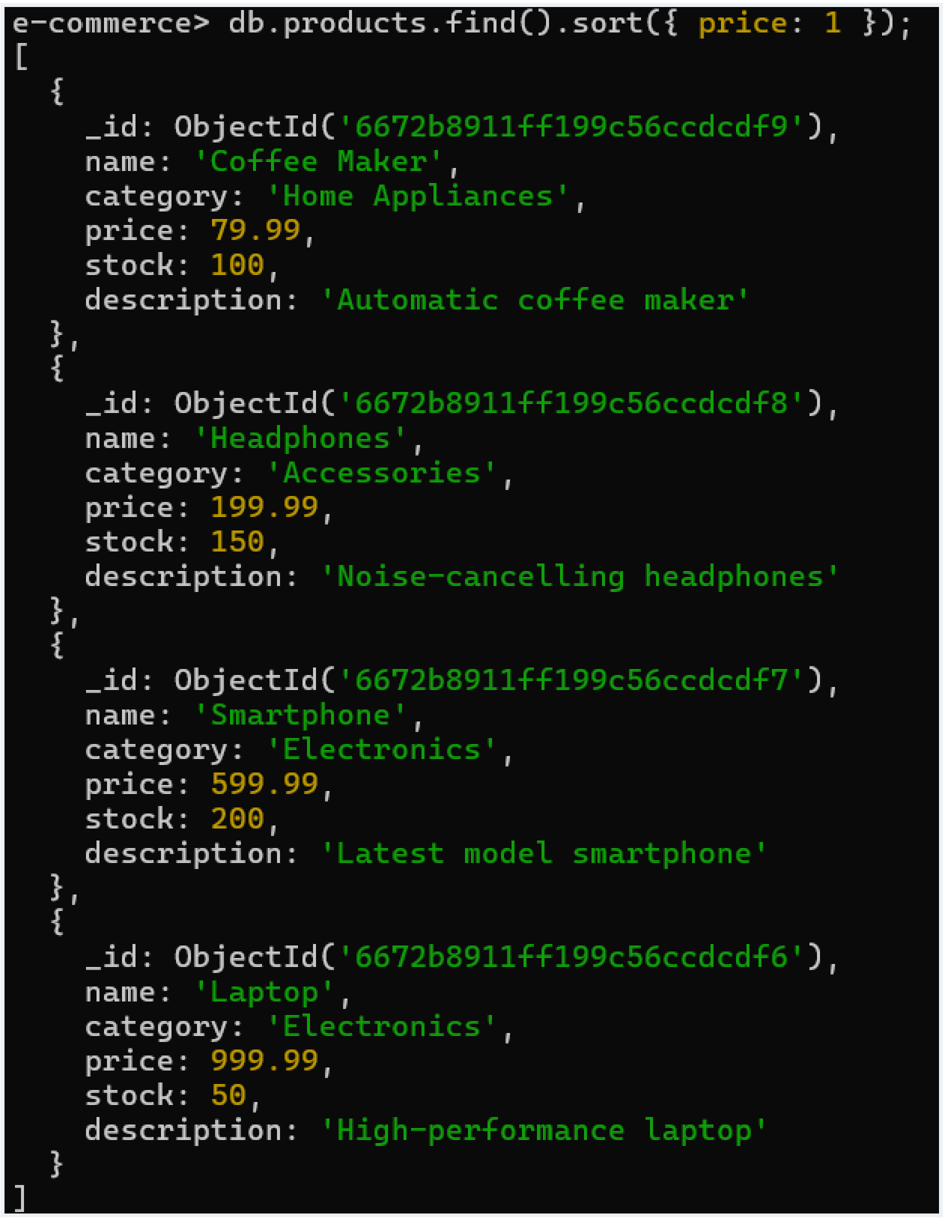
```
e-commerce> db.products.find({ $text: { $search: "Laptop" } });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf6'),
    name: 'Laptop',
    category: 'Electronics',
    price: 999.99,
    stock: 50,
    description: 'High-performance laptop'
  }
]
```

This query will use the text index on name.

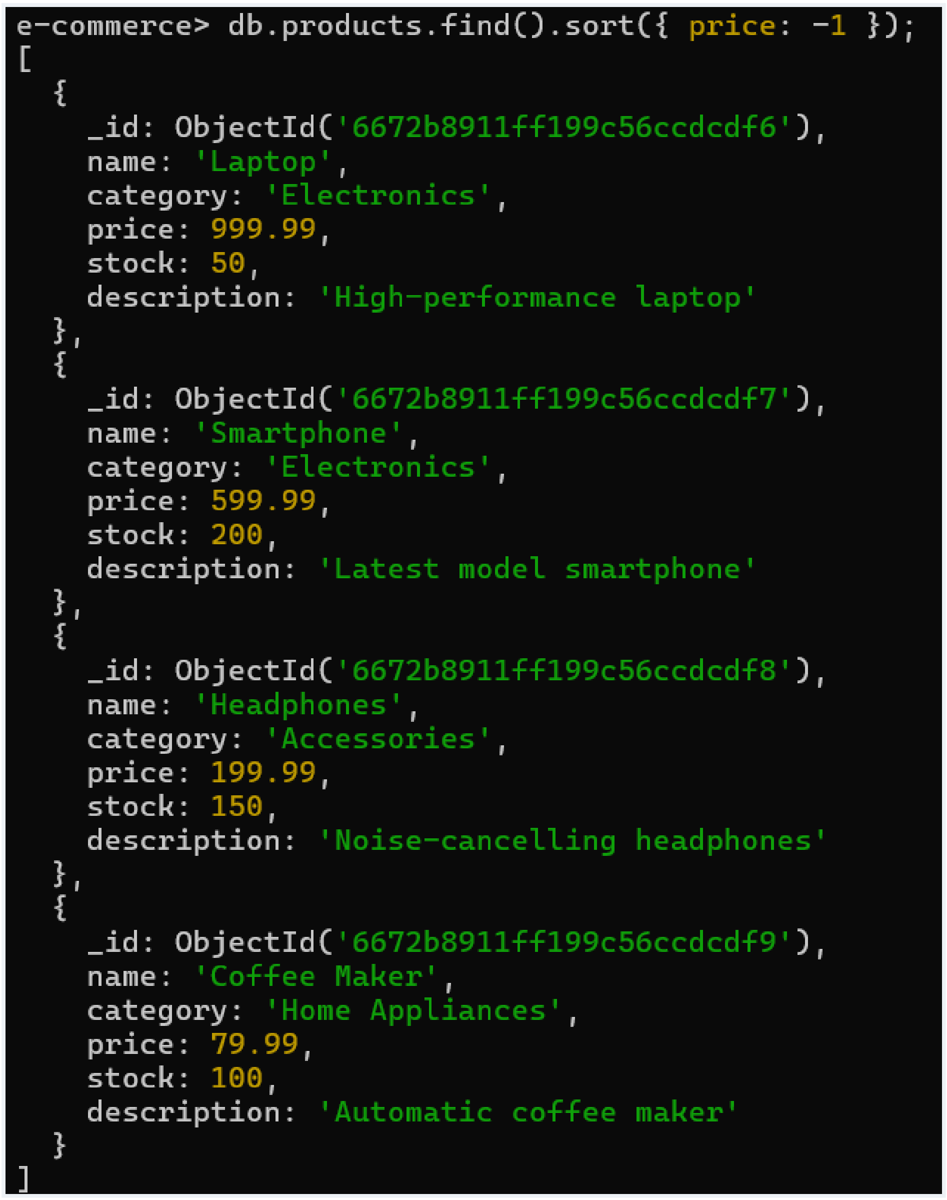
Sort Products by Price:

db.products.find().sort({ price: 1 }); ascending order

db.products.find().sort({ price: -1 }); descending order



```
e-commerce> db.products.find().sort({ price: 1 });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf9'),
    name: 'Coffee Maker',
    category: 'Home Appliances',
    price: 79.99,
    stock: 100,
    description: 'Automatic coffee maker'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf8'),
    name: 'Headphones',
    category: 'Accessories',
    price: 199.99,
    stock: 150,
    description: 'Noise-cancelling headphones'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf7'),
    name: 'Smartphone',
    category: 'Electronics',
    price: 599.99,
    stock: 200,
    description: 'Latest model smartphone'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf6'),
    name: 'Laptop',
    category: 'Electronics',
    price: 999.99,
    stock: 50,
    description: 'High-performance laptop'
  }
]
```



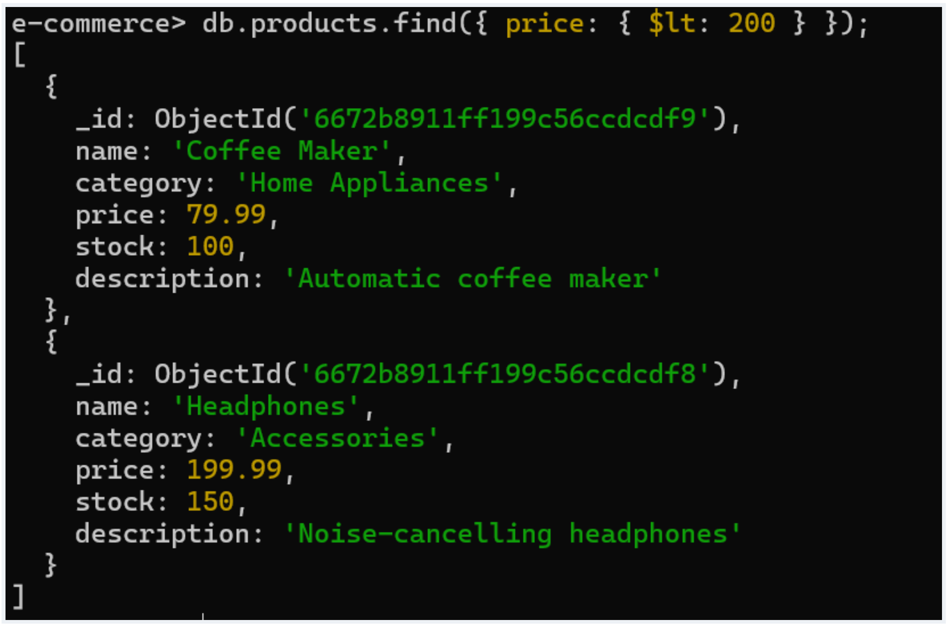
```
e-commerce> db.products.find().sort({ price: -1 });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf6'),
    name: 'Laptop',
    category: 'Electronics',
    price: 999.99,
    stock: 50,
    description: 'High-performance laptop'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf7'),
    name: 'Smartphone',
    category: 'Electronics',
    price: 599.99,
    stock: 200,
    description: 'Latest model smartphone'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf8'),
    name: 'Headphones',
    category: 'Accessories',
    price: 199.99,
    stock: 150,
    description: 'Noise-cancelling headphones'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf9'),
    name: 'Coffee Maker',
    category: 'Home Appliances',
    price: 79.99,
    stock: 100,
    description: 'Automatic coffee maker'
  }
]
```

This query will use the index on price to sort the results efficiently.

Query for Products with Price Less Than a Specific Value

To find products where the price is less than $200:

>db.products.find({ price: { $lt: 200 } });

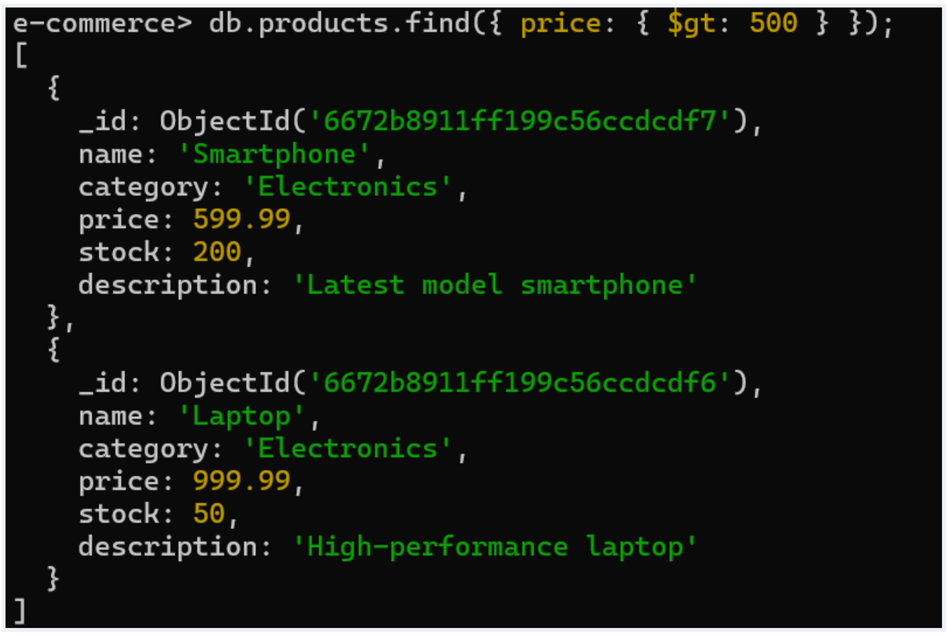


```
e-commerce> db.products.find({ price: { $lt: 200 } });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf9'),
    name: 'Coffee Maker',
    category: 'Home Appliances',
    price: 79.99,
    stock: 100,
    description: 'Automatic coffee maker'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf8'),
    name: 'Headphones',
    category: 'Accessories',
    price: 199.99,
    stock: 150,
    description: 'Noise-cancelling headphones'
  }
]
```

Query for Products with Price Greater Than a Specific Value

To find products where the price is greater than $500:

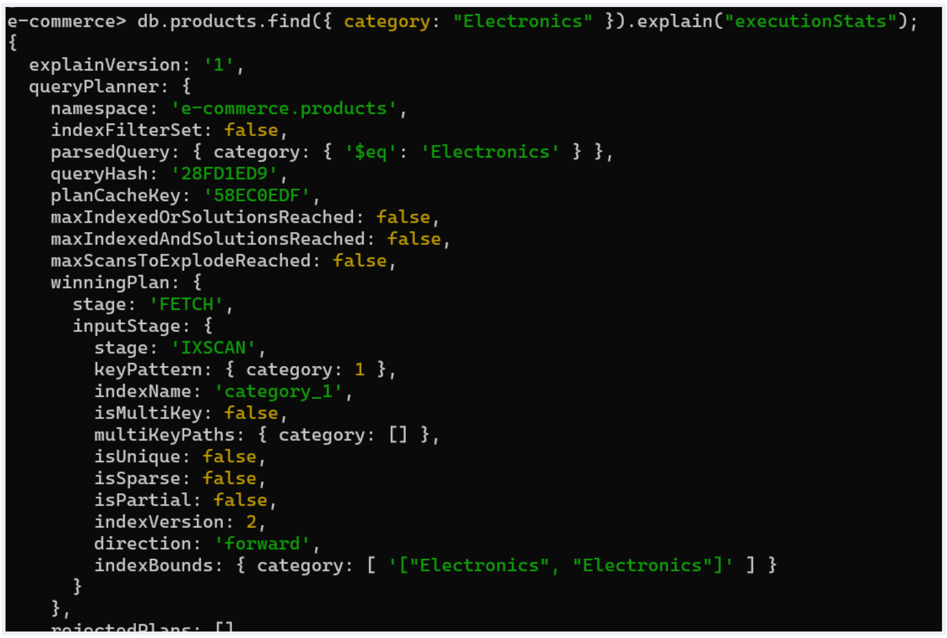
>db.products.find({ price: { $gt: 500 } });



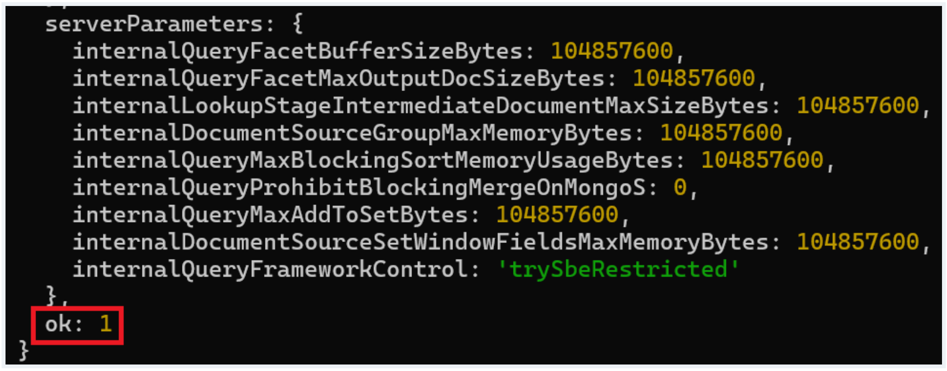
```
e-commerce> db.products.find({ price: { $gt: 500 } });
[
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf7'),
    name: 'Smartphone',
    category: 'Electronics',
    price: 599.99,
    stock: 200,
    description: 'Latest model smartphone'
  },
  {
    _id: ObjectId('6672b8911ff199c56ccdcdf6'),
    name: 'Laptop',
    category: 'Electronics',
    price: 999.99,
    stock: 50,
    description: 'High-performance laptop'
  }
]
```

Monitor Index Performance:

db.products.find({ category: "Electronics" }).explain("executionStats");



```
e-commerce> db.products.find({ category: "Electronics" }).explain("executionStats");
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'e-commerce.products',
    indexFilterSet: false,
    parsedQuery: { category: { '$eq': 'Electronics' } },
    queryHash: '28FD1ED9',
    planCacheKey: '58EC0EDF',
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { category: 1 },
        indexName: 'category_1',
        isMultiKey: false,
        multiKeyPaths: { category: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { category: [ '["Electronics", "Electronics"]' ] }
      }
    },
```



```
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted'
  },
  ok: 1
}
```

This will show you the query plan and whether the index is being used.

Conclusion

By understanding your application's query patterns and strategically creating indexes, you can significantly improve the performance and efficiency of your MongoDB database in real-life scenarios.