Question 1

use MongoDBtest

switched to db MongoDBtest

db.store.insert({  
 name: 'fan',  
 description: '20v fan',  
 price: '100',  
 category: 'electronic',  
 stock: 100  
})

db.store.insertMany([  
 {  
 name: 'fan',  
 description: '20v fan',  
 price: 100,  
 category: 'electronic',  
 stockQuantity: 100  
 },  
 {  
 name: 'shirt',  
 description: 'Cotton shirt',  
 price: 25.99,  
 category: 'clothing',  
 stockQuantity: 50  
 },  
 {  
 name: 'phone',  
 description: 'Smartphone',  
 price: 599.99,  
 category: 'electronic',  
 stockQuantity: 20  
 },  
 {  
 name: 'watch',  
 description: 'Water-resistant watch',  
 price: 199.99,  
 category: 'accessories',  
 stockQuantity: 80  
 },  
]);

db.store.insertMany([  
 {  
 name: 'headphones',  
 description: 'Wireless headphones',  
 price: 89.99,  
 category: 'electronic',  
 stockQuantity: 30  
 },  
 {  
 name: 'backpack',  
 description: 'Waterproof backpack',  
 price: 49.99,  
 category: 'accessories',  
 stockQuantity: 60  
 },  
 {  
 name: 'television',  
 description: 'Smart TV',  
 price: 899.99,  
 category: 'electronic',  
 stockQuantity: 10  
 },  
 {  
 name: 'jeans',  
 description: 'Slim-fit jeans',  
 price: 59.99,  
 category: 'clothing',  
 stockQuantity: 70  
 },  
 {  
 name: 'earrings',  
 description: 'Silver earrings',  
 price: 29.99,  
 category: 'accessories',  
 stockQuantity: 90  
 }  
]);

Question 2b

db.store.aggregate([  
 {  
 $match: {  
 name: 'fan' // Filter for the specific product 'fan'  
 }  
 },  
 {  
 $addFields: {  
 stockQuantity: { $subtract: ['$stockQuantity', 1] } // Decrement the stock quantity by 1  
 }  
 }  
])

Question 2c

db.store.aggregate([{$group:{\_id:"$category"}},{$sort: { "price": 1 } }])

o/p:- {

\_id: 'accessories'

}

{

\_id: 'electronic'

}

{

\_id: 'clothing'

}

Question 2d

db.store.aggregate([  
 {  
 $match: {  
 price: { $gte: 50, $lte: 200 }, // Specify the price range (from $50 to $200)  
 stockQuantity: { $gt: 0 } // Filter for products with stock quantity greater than 0  
 }  
 }  
])

o/p :- {

\_id: ObjectId("64b232a382d2f2a6f40f8a15"),

name: 'fan',

description: '20v fan',

price: 100,

category: 'electronic',

stockQuantity: 100

}

{

\_id: ObjectId("64b232a382d2f2a6f40f8a18"),

name: 'watch',

description: 'Water-resistant watch',

price: 199.99,

category: 'accessories',

stockQuantity: 80

}

Question 2e

db.store.aggregate([  
 {  
 $match: {  
 \_id: ObjectId('64b232c782d2f2a6f40f8a1d')  
 }  
 },  
 {  
 $project: {  
 "name": 1,  
 "description": 1,  
 "price": 1,  
 "category": 1,  
 "stockQuantity": 1  
 }  
 }  
])

o/p :- db.store.aggregate([  
 {  
 $match: {  
 \_id: ObjectId('64b232c782d2f2a6f40f8a1d')  
 }  
 },  
 {  
 $project: {  
 "name": 1,  
 "description": 1,  
 "price": 1,  
 "category": 1,  
 "stockQuantity": 1  
 }  
 }  
])

Question 3

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

⦁ user can filter by specific category fast.

⦁ when order is placed, updates are done quickly into the data base.

⦁ By indexing fields used for reporting purposes, such as sales data or inventory levels, the system can generate up-to-date reports and analytics insights promptly.

Question 4

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

* Capped Collections: Capped collections are fixed-size collections in MongoDB that maintain insertion order and automatically overwrite older documents when the collection reaches its size limit. They are useful for scenarios where you only need to retain a fixed amount of the most recent data, such as logs or real-time event streams. Capped collections provide a scalable approach for managing high-velocity data with a bounded storage footprint.
* Sharding: Sharding is a technique used to distribute data across multiple machines (shards) in a MongoDB cluster. It allows horizontal scaling by partitioning the data based on a shard key. Each shard contains a subset of the data, and MongoDB automatically routes queries to the appropriate shard. Sharding enables distributing the load across multiple servers, increasing the system's capacity to handle larger datasets and higher traffic volumes.
* Indexes: MongoDB's indexing capabilities play a crucial role in scaling read operations. By creating appropriate indexes on frequently queried fields, you can significantly improve query performance, reduce response times, and efficiently scale read-intensive workloads. MongoDB supports various types of indexes, including single-field indexes, compound indexes, geospatial indexes, and text indexes.