Aggregation and indexing are powerful features in MongoDB used to perform complex data processing and to optimize query performance.

Let's look at some examples using a sample `ecommerce` database with a `sales` collection that contains documents related to sales transactions.

Step 1: Sample Data Setup

First, we'll create a sample dataset for aggregation and indexing examples.

javascript
Copy code

```
use ecommerce

db.sales.insertMany([
    { item: "laptop", quantity: 2, price: 800, customer: "Alice",
region: "North", date: new Date("2024-10-01") },
    { item: "phone", quantity: 5, price: 300, customer: "Bob", region:
"South", date: new Date("2024-10-02") },
    { item: "tablet", quantity: 3, price: 450, customer: "Alice",
region: "North", date: new Date("2024-10-05") },
    { item: "laptop", quantity: 1, price: 800, customer: "Charlie",
region: "East", date: new Date("2024-10-06") },
    { item: "phone", quantity: 2, price: 300, customer: "David",
region: "West", date: new Date("2024-10-07") },
    { item: "laptop", quantity: 1, price: 800, customer: "Alice",
region: "North", date: new Date("2024-10-10") }
])
```

Step 2: Aggregation Queries

1. Total Sales Revenue by Item

This query calculates the total revenue for each item type.

javascript
Copy code

```
db.sales.aggregate([
    {
        $group: {
            _id: "$item",
            totalRevenue: { $sum: { $multiply: ["$quantity", "$price"]
} }
        }
    }
])
```

Explanation: This query groups documents by the `item` field and calculates `totalRevenue` by multiplying `quantity` and `price` for each document and summing up the result within each group.

2. Total Quantity Sold by Region

This query aggregates the total quantity of items sold per region.

javascript
Copy code

```
db.sales.aggregate([
    {
        $group: {
            _id: "$region",
            totalQuantity: { $sum: "$quantity" }
        }
    }
])
```

Explanation: This groups the documents by `region` and sums up the `quantity` field for each region.

3. Monthly Sales for Each Customer

Calculate the monthly sales for each customer to see their purchasing behavior.

javascript
Copy code

```
db.sales.aggregate([
    {
        $group: {
            _id: { customer: "$customer", month: { $month: "$date" }
},
            totalSales: { $sum: { $multiply: ["$quantity", "$price"] }
}
        }
    }
])
```

Explanation: Here, the `$group` stage uses a compound key `_id` containing both `customer` and the month extracted from `date`. This gives the monthly sales for each customer.

4. Average Purchase Amount by Item

This query calculates the average amount spent per item.

javascript
Copy code

```
db.sales.aggregate([
    {
        $group: {
            _id: "$item",
            averagePurchaseAmount: { $avg: { $multiply: ["$quantity",
"$price"] } }
        }
    }
])
```

Step 3: Indexing

Indexes can significantly improve the performance of queries. Let's create indexes on fields used frequently in filtering, sorting, or grouping.

1. Create an Index on the `item` Field

This index improves the performance of queries that filter or sort by the `item` field.

javascript
Copy code

```
db.sales.createIndex({ item: 1 })
```

2. Create a Compound Index on `customer` and `date`

This compound index is useful for queries that involve both the `customer` and `date` fields.

javascript
Copy code

```
db.sales.createIndex({ customer: 1, date: -1 })
```

Explanation: The compound index supports queries that include both `customer` and `date`, such as looking up sales for a specific customer within a date range or sorting sales by date.

Example Query Using the Index

After creating indexes, MongoDB will utilize them for queries involving these fields. Here's an example:

javascript
Copy code

```
db.sales.find({ customer: "Alice" }).sort({ date: -1 })
```

With the compound index on `{ customer: 1, date: -1 }`, this query will run faster than without the index, especially with a large dataset.

Summary

- **Aggregation**: Useful for summarizing data. We used `$group` to calculate totals, averages, and monthly statistics.
- **Indexing**: Helps optimize query performance. We created single-field and compound indexes for frequently used fields in our queries.

Aggregation and indexing are essential for handling complex queries efficiently in MongoDB.