**Aggregation & Indexing**

Use assignment10

Db.createCollection(‘orders’)

db.orders.insertMany([

{ "product": "Laptop", "category": "Electronics", "quantity": 3, "price": 1000, "purchaseDate": ISODate("2023-08-01") },

{ "product": "Chair", "category": "Furniture", "quantity": 10, "price": 50, "purchaseDate": ISODate("2023-08-01") },

{ "product": "Tablet", "category": "Electronics", "quantity": 5, "price": 600, "purchaseDate": ISODate("2023-08-02") },

{ "product": "Desk", "category": "Furniture", "quantity": 2, "price": 300, "purchaseDate": ISODate("2023-08-02") },

{ "product": "Phone", "category": "Electronics", "quantity": 8, "price": 700, "purchaseDate": ISODate("2023-08-03") }

])

**Aggregation query**

db.orders.aggregate([

{

$group: {

\_id: "$category",

totalRevenue: { $sum: { $multiply: ["$quantity", "$price"] } }

}

}

])

**Indexing queries**

db.orders.createIndex({ "purchaseDate": 1 })

db.orders.find({ "purchaseDate": ISODate("2023-08-01") })

**theory**

. Aggregation:

Purpose: Aggregation operations are used to process multiple documents and return computed results, often useful for generating reports, statistics, and analytics.

Aggregation Pipeline: In MongoDB, aggregation is achieved using a pipeline, where each stage transforms the documents in some way.

Code Explanation:

db.orders.aggregate([...]) initializes the aggregation pipeline on the "orders" collection.

The $group stage groups the documents based on a specified field, here "category".

Within $group, \_id: "$category" groups orders by category (e.g., "Electronics" and "Furniture").

The field totalRevenue is calculated for each category by multiplying quantity and price for each document and summing up the total revenue for that category.

The final result provides the total revenue for each category.

2. Indexing:

Purpose: Indexing improves the performance of search queries by reducing the amount of data MongoDB needs to scan. Proper indexing can make retrieval faster, especially for large datasets.

Code Explanation:

db.orders.createIndex({ "purchaseDate": 1 }) creates an index on the "purchaseDate" field in ascending order (indicated by 1).

When the find query db.orders.find({ "purchaseDate": ISODate("2023-08-01") }) is run, MongoDB uses the index on "purchaseDate" to quickly locate documents from the specified date.

This index minimizes scan time, especially when searching for a specific date in large datasets, resulting in faster and more efficient query performance.

Summary

In this example:

Aggregation enables calculation of total revenue by category, providing valuable insights into sales distribution across product types.

Indexing on purchaseDate optimizes the search for orders made on a specific date, which is crucial in scenarios where orders span across multiple dates or years. Together, these features enhance data analysis and query efficiency within MongoDB.