**MongoDB Exercises**

**Scenario: Online Shopping Platform**

You are managing a MongoDB database for an online shopping platform. The database contains the following collections:

1. users: Stores user details.
2. orders: Stores order information.
3. products: Stores product information.

**Creating Database:**

*// Create and use the shopping database*

use shopping\_platform

*// 1. Create users collection and insert sample data*

db.users.insertMany([

{

userId: "U001",

name: "John Doe",

email: "john.doe@example.com",

age: 28,

address: {

city: "New York",

state: "NY",

zip: "10001"

},

createdAt: new Date("2024-01-01T10:00:00Z")

},

{

userId: "U002",

name: "Jane Smith",

email: "jane.smith@example.com",

age: 32,

address: {

city: "Los Angeles",

state: "CA",

zip: "90001"

},

createdAt: new Date("2024-01-02T15:30:00Z")

}

]);

*// 2. Create orders collection and insert sample data*

db.orders.insertMany([

{

orderId: "ORD001",

userId: "U001",

orderDate: new Date("2024-12-10T14:32:00Z"),

items: [{

productId: "P001",

quantity: 2,

price: 100

},

{

productId: "P002",

quantity: 1,

price: 50

}],

totalAmount: 250,

status: "Delivered"

},

{

orderId: "ORD002",

userId: "U002",

orderDate: new Date("2024-12-15T09:45:00Z"),

items: [

{

productId: "P001",

quantity: 1,

price: 100

}],

totalAmount: 100,

status: "Processing"

}]);

*// 3. Create products collection and insert sample data*

db.products.insertMany([

{

productId: "P001",

name: "Wireless Mouse",

category: "Electronics",

price: 50,

stock: 200,

ratings: [{

userId: "U002",

rating: 4.5

},

{

userId: "U003",

rating: 3.0

}]},

{

productId: "P002",

name: "Bluetooth Keyboard",

category: "Electronics",

price: 80,

stock: 150,

ratings: [

{

userId: "U001",

rating: 5.0

}]}]);

*// 4. Create warehouses collection with geospatial index*

db.warehouses.createIndex({ location: "2dsphere" });

*// Insert warehouse data*

db.warehouses.insertMany([

{

warehouseId: "W001",

location: {

type: "Point",

coordinates: [-74.006, 40.7128] *// New York*

},

products: ["P001", "P002", "P003"]

},

{

warehouseId: "W002",

location: {

type: "Point",

coordinates: [-118.2437, 34.0522] *// Los Angeles*

},

products: ["P001", "P002"]

}

])

**Queries**

**1. Find High-Spending Users**

**Write a query to find users who have spent more than $500 in total across all**

**their orders.**

**Hint: Use $lookup to join the users and orders collections and calculate the total**

**spending.**

db.users.aggregate([

{

$lookup: {

from: "orders",

localField: "userId",

foreignField: "userId",

as: "userOrders"

}

},

{

$addFields: {

totalSpent: {

$sum: "$userOrders.totalAmount"

}

}

},

{

$match: {

totalSpent: { $gt: 500 }

}

},

{

$project: {

userId: 1,

name: 1,

email: 1,

totalSpent: 1

}

}]);

**2. List Popular Products by Average Rating**

**Retrieve products that have an average rating greater than or equal to 4.**

**Hint: Use $unwind to flatten the ratings array and $group to calculate the**

**average rating.**

db.products.aggregate([

{

$unwind: "$ratings"

},

{

$group: {

\_id: {

productId: "$productId",

name: "$name",

category: "$category",

price: "$price",

stock: "$stock"

},

averageRating: { $avg: "$ratings.rating" }

}

},

{

$match: {

averageRating: { $gte: 4 }

}

},

{

$project: {

\_id: 0,

productId: "$\_id.productId",

name: "$\_id.name",

category: "$\_id.category",

price: "$\_id.price",

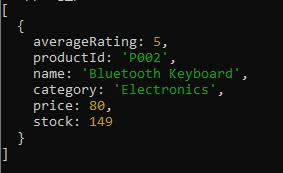
stock: "$\_id.stock",

averageRating: 1

}

}

]);



**3. Search for Orders in a Specific Time Range**

**Find all orders placed between "2024-12-01" and "2024-12-31". Ensure the result**

**includes the user name for each order.**

**Hint: Use $match with a date range filter and $lookup to join with the users**

**collection.**

db.orders.aggregate([

{

$match: {

orderDate: {

$gte: ISODate("2024-12-01T00:00:00Z"),

$lte: ISODate("2024-12-31T23:59:59Z")

}

}

},

{

$lookup: {

from: "users",

localField: "userId",

foreignField: "userId",

as: "userDetails"

}

},

{

$unwind: "$userDetails"

},

{

$project: {

orderId: 1,

orderDate: 1,

totalAmount: 1,

status: 1,

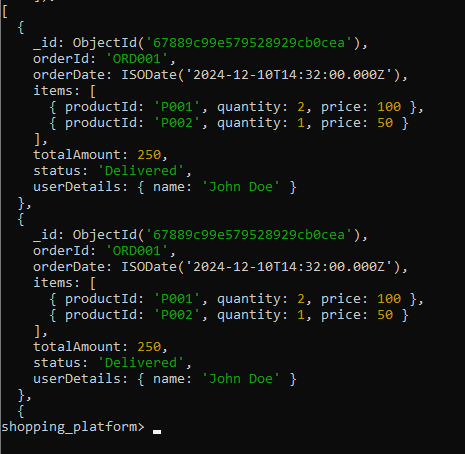
"userDetails.name": 1,

items: 1

}

}

]);



**4. Update Stock After Order Completion**

**When an order is placed, reduce the stock of each product by the quantity in the**

**order. For example, if 2 units of P001 were purchased, decrement its stock by 2.**

**Hint: Use $inc with updateOne or updateMany.**

db.orders.find({ orderId: "ORD001" }).forEach(function(order) {

order.items.forEach(function(item) {

db.products.updateOne(

{ productId: item.productId },

{ $inc: { stock: -item.quantity } }

);

});

});

**5. Find Nearest Warehouse**

**Assume there’s a warehouses collection with geospatial data:**

**{ "warehouseId": "W001",**

**"location": { "type": "Point", "coordinates": [-74.006,**

**40.7128] },**

**"products": ["P001", "P002", "P003"] }**

**Find the nearest warehouse within a 50-kilometer radius that stocks "P001".**

**Hint: Use the $geoNear aggregation stage with a filter on the products array.**

db.warehouses.createIndex({ location: "2dsphere" });



db.warehouses.aggregate([

{

$geoNear: {

near: {

type: "Point",

coordinates: [-74.006, 40.7128]

},

distanceField: "distance",

maxDistance: 50000,

spherical: true,

query: { products: "P001" }

}

},

{

$project: {

\_id: 0,

warehouseId: 1,

distance: { $round: ["$distance", 2] },

products: 1,

location: 1

}

}

]);

