

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

```
db.users.insertMany([
  {
    userId: "R003",
    name: "Rahul",
    email: "rahul@example.com",
    age: 28,
    address: {
      city: "Chicago",
      state: "IL",
      zip: "60601"
    },
    createdAt: new Date("2024-03-01T12:00:00Z")
  },
  {
    userId: "R004",
    name: "Emma",
    email: "emma@example.com",
    age: 29,
    address: {
      city: "San Francisco",
```

```
    state: "CA",
    zip: "94101"
  },
  createdAt: new Date("2024-03-02T14:00:00Z")
}
]);
```

// 2. Create orders collection and insert sample data

```
db.orders.insertMany([
  {
    orderId: "ORD003",
    userId: "R003",
    orderDate: new Date("2025-01-10T10:15:00Z"),
    items: [
      {
        productId: "P003",
        quantity: 3,
        price: 200
      },
      {
        productId: "P004",
        quantity: 2,
        price: 150
      }
    ],
    totalAmount: 850,
    status: "Shipped"
  }
]);
```

```
},  
{  
  orderId: "ORD004",  
  userId: "R004",  
  orderDate: new Date("2025-01-12T16:00:00Z"),  
  items: [  
    {  
      productId: "P005",  
      quantity: 1,  
      price: 300  
    },  
    {  
      productId: "P006",  
      quantity: 2,  
      price: 120  
    }  
  ],  
  totalAmount: 540,  
  status: "Delivered"  
}  
]);
```

```
mongosh mongodb://127.0.0.1:27027/shopping_platform> use shopping_platform
switched to db shopping_platform

...   status: "Shipped"
...   },
...   {
...     orderId: "ORD004",
...     userId: "R004",
...     orderDate: new Date("2025-01-12T16:00:00Z"),
...     items: [
...       {
...         productId: "P005",
...         quantity: 1,
...         price: 300
...       },
...       {
...         productId: "P006",
...         quantity: 2,
...         price: 120
...       }
...     ],
...     totalAmount: 540,
...     status: "Delivered"
...   }
... ];
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('679333303c1bb1b71b0d8192'),
    '1': ObjectId('679333303c1bb1b71b0d8193')
  }
}
shopping_platform> |
```

// 3. Create products collection and insert sample data

```
db.products.insertMany([
{
  productId: "P007",
  name: "Gaming Headset",
  category: "Electronics",
  price: 250,
  stock: 100,
  ratings: [
    {
      userId: "R003",
```

```
        rating: 4.8
    },
    {
        userId: "R004",
        rating: 4.2
    }
]
},
{
    productId: "P008",
    name: "Smartwatch",
    category: "Wearables",
    price: 300,
    stock: 50,
    ratings: [
        {
            userId: "R004",
            rating: 4.5
        }
    ]
}
]);
```

```
mongosh mongodb://127.0.0.0:27020/shopping_platform?authSource=admin
> use shopping_platform;
> insertMany([
  {
    productId: "P001",
    name: "Smartwatch",
    category: "Wearables",
    price: 300,
    stock: 50,
    ratings: [
      {
        userId: "R004",
        rating: 4.2
      }
    ]
  },
  {
    productId: "P008",
    name: "Smartwatch",
    category: "Wearables",
    price: 300,
    stock: 50,
    ratings: [
      {
        userId: "R004",
        rating: 4.5
      }
    ]
  }
]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('679336383c1bb1b71b0d8194'),
    '1': ObjectId('679336383c1bb1b71b0d8195')
  }
}
shopping_platform> |
fwd-i-search: _
```

// 4. Create warehouses collection with geospatial index

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

```
db.warehouses.insertMany([
```

```
{
```

```

warehouseId: "W003",

location: {

  type: "Point",

  coordinates: [41.8781, -87.6298]

},

products: ["P007", "P008", "P009"]

}

]);

```

```

mongosh mongodb://127.0.0.1:27020
> use shopping_platform
shopping_platform> db.warehouses.insertMany([
...   {
...     warehouseId: "W003",
...     location: {
...       type: "Point",
...       coordinates: [41.8781, -87.6298]
...     },
...     products: ["P007", "P008", "P009"]
...   }
... ]);
{
  acknowledged: true,
  insertedIds: { '_id': ObjectId('6793385d3c1bb1b71b0d819a') }
}
shopping_platform> |

deletedCount: 0,
upsertedCount: 0,
upsertedIds: {},
insertedIds: { '_id': ObjectId('679338023c1bb1b71b0d8198') }
}
Write Errors: [
  WriteError {
    err: {
      index: 1,
      code: 16755,
      errmsg: 'Can\'t extract geo keys: { "_id": ObjectId(\'679338023c1bb1b71b0d8199\'), "warehouseId": "W004", "location": { type: "Point", coordinates: [ 37.7749, -122.4194 ] }, "products": [ "P008", "P010" ] } Longitude/latitude is out of bounds, lng: 37.7749 lat: -122.419',
      errInfo: undefined,
      op: {
        warehouseId: 'W004',
        location: { type: 'Point', coordinates: [ 37.7749, -122.4194 ] },
        products: [ 'P008', 'P010' ],
        _id: ObjectId('679338023c1bb1b71b0d8199')
      }
    }
  }
]
shopping_platform> db.warehouses.createIndex({ location: "2dsphere" });
location_2dsphere
shopping_platform>

shopping_platform> db.warehouses.insertMany([
...   {
...     warehouseId: "W003",
...     location: {
...       type: "Point",
...       coordinates: [41.8781, -87.6298]
...     },
...     products: ["P007", "P008", "P009"]
...   }
... ]);
{
  acknowledged: true,
  insertedIds: { '_id': ObjectId('6793385d3c1bb1b71b0d819a') }
}
shopping_platform> |

```

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  
}  
}  
]);
```

```
mongosh mongodb://127.0.0.1:27027/shopping_platform>
...   }
...   },
...   {
...     $match: {
...       averageRating: { $gte: 4 }
...     }
...   },
...   {
...     $project: {
...       _id: 0,
...       productId: "$_id.productId",
...       name: "$_id.name",
...       category: "$_id.category",
...       price: "$_id.price",
...       stock: "$_id.stock",
...       averageRating: 1
...     }
...   }
... }
... ]);
[
  {
    averageRating: 4.5,
    productId: 'P007',
    name: 'Gaming Headset',
    category: 'Electronics',
    price: 250,
    stock: 100
  },
  {
    averageRating: 4.5,
    productId: 'P008',
    name: 'Smartwatch',
    category: 'Wearables',
    price: 300,
    stock: 50
  }
]
shopping_platform>
```

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.

[

```
{
  orderId: "ORD001",
  orderDate: ISODate("2024-12-10T14:32:00Z"),
  totalAmount: 250,
  status: "Delivered",
  userDetails: {
    name: "John Doe"
  },
  items: [
    {
      productId: "P001",
      quantity: 2,
      price: 100
    },
    {
      productId: "P002",
      quantity: 1,
      price: 50
    }
  ],
},
{
  orderId: "ORD002",
  orderDate: ISODate("2024-12-15T09:45:00Z"),
  totalAmount: 100,
  status: "Processing",
```

```
userDetails: {  
  name: "Jane Smith"  
},  
items: [  
  {  
    productId: "P001",  
    quantity: 1,  
    price: 100  
  }  
]  
}
```

```
mongosh mongodb://127.0.0.1:27017/?
Command Prompt: mongosh
mongodb://127.0.0.1:27017/?
directConnection=true&serverSelectionTimeoutMS=20
00
ctrl+alt+1
...   orderDate: ISODate("2024-12-15T09:45:00Z"),
...   totalAmount: 100,
...   status: "Processing",
...   userDetails: {
...     name: "Jane Smith"
...   },
...   items: [
...     {
...       productId: "P001",
...       quantity: 1,
...       price: 100
...     }
...   ]
... }
... ]
[
  {
    orderId: 'ORD001',
    orderDate: ISODate('2024-12-10T14:32:00.000Z'),
    totalAmount: 250,
    status: 'Delivered',
    userDetails: { name: 'John Doe' },
    items: [
      { productId: 'P001', quantity: 2, price: 100 },
      { productId: 'P002', quantity: 1, price: 50 }
    ]
  },
  {
    orderId: 'ORD002',
    orderDate: ISODate('2024-12-15T09:45:00.000Z'),
    totalAmount: 100,
    status: 'Processing',
    userDetails: { name: 'Jane Smith' },
    items: [ { productId: 'P001', quantity: 1, price: 100 } ]
  }
]
shopping_platform> |
```

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" });
```

```
shopping_platform> db.warehouses.createIndex({ location: "2dsphere" });  
location_2dsphere  
shopping_platform> |
```

```
db.warehouses.aggregate([  
  
  {  
  
    $geoNear: {  
  
      near: {  
  
        type: "Point",  
  
        coordinates: [41.8781, -87.6298]  
  
      },  
  
      distanceField: "distance",  
  
      maxDistance: 50000,  
  
      spherical: true,  
  
      query: { products: "P007" }  
  
    }  
  
  },  
  
  {  
  
    $project: {
```

```
_id: 0,  
warehouseId: 1,  
distance: { $round: ["$distance", 2] },  
products: 1,  
location: 1  
}  
}  
]);
```

```

mongosh mongodb://127.0.0.1:27020 > use shopping_platform
shopping_platform> db.products.aggregate([
  {
    $near: {
      type: "Point",
      coordinates: [41.8781, -87.6298]
    },
    distanceField: "distance",
    maxDistance: 50000,
    spherical: true,
    query: { products: "P007" }
  },
  {
    $project: {
      _id: 0,
      warehouseId: 1,
      distance: { $round: ["$distance", 2] },
      products: 1,
      location: 1
    }
  }
]);
[
  {
    warehouseId: 'W003',
    location: { type: 'Point', coordinates: [ 41.8781, -87.6298 ] },
    products: [ 'P007', 'P008', 'P009' ],
    distance: 0
  },
  {
    warehouseId: 'W003',
    location: { type: 'Point', coordinates: [ 41.8781, -87.6298 ] },
    products: [ 'P007', 'P008', 'P009' ],
    distance: 0
  },
  {
    warehouseId: 'W003',
    location: { type: 'Point', coordinates: [ 41.8781, -87.6298 ] },
    products: [ 'P007', 'P008', 'P009' ],
    distance: 0
  }
]
shopping_platform> |

```