MongoDB Exercises

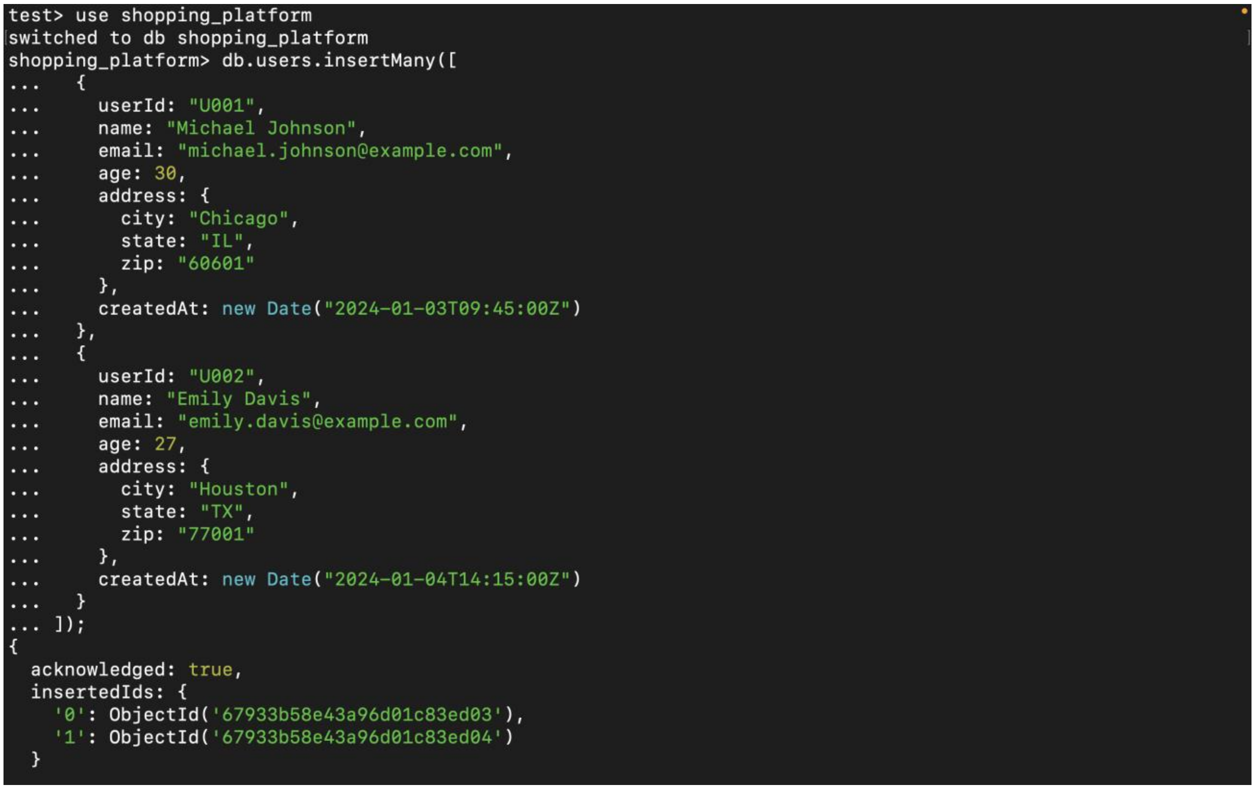
Scenario: Online Shopping Platform

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.

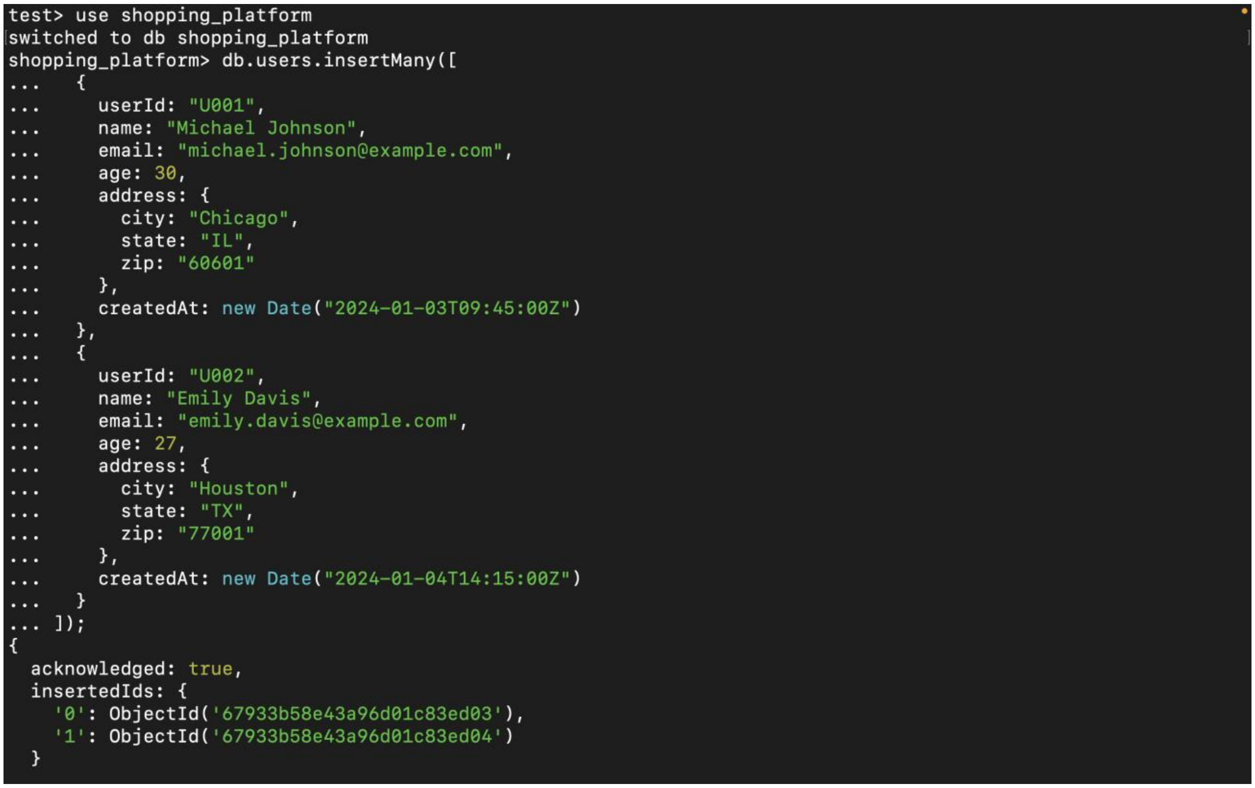
Creating Database:

```
// Create and use the shopping database
use shopping_platform

// 1. Create users collection and insert sample data

db.users.insertMany([
  {
    userId: "U001",
    name: "Michael Johnson",
    email: "michael.johnson@example.com",
    age: 30,
    address: {
      city: "Chicago",
      state: "IL",
      zip: "60601"
    },
    createdAt: new Date("2024-01-03T09:45:00Z")
  },
  {
    userId: "U002",
    name: "Emily Davis",
    email: "emily.davis@example.com",
    age: 27,
    address: {
      city: "Houston",
      state: "TX",
      zip: "77001"
    },
    createdAt: new Date("2024-01-04T14:15:00Z")
  }
]);
```



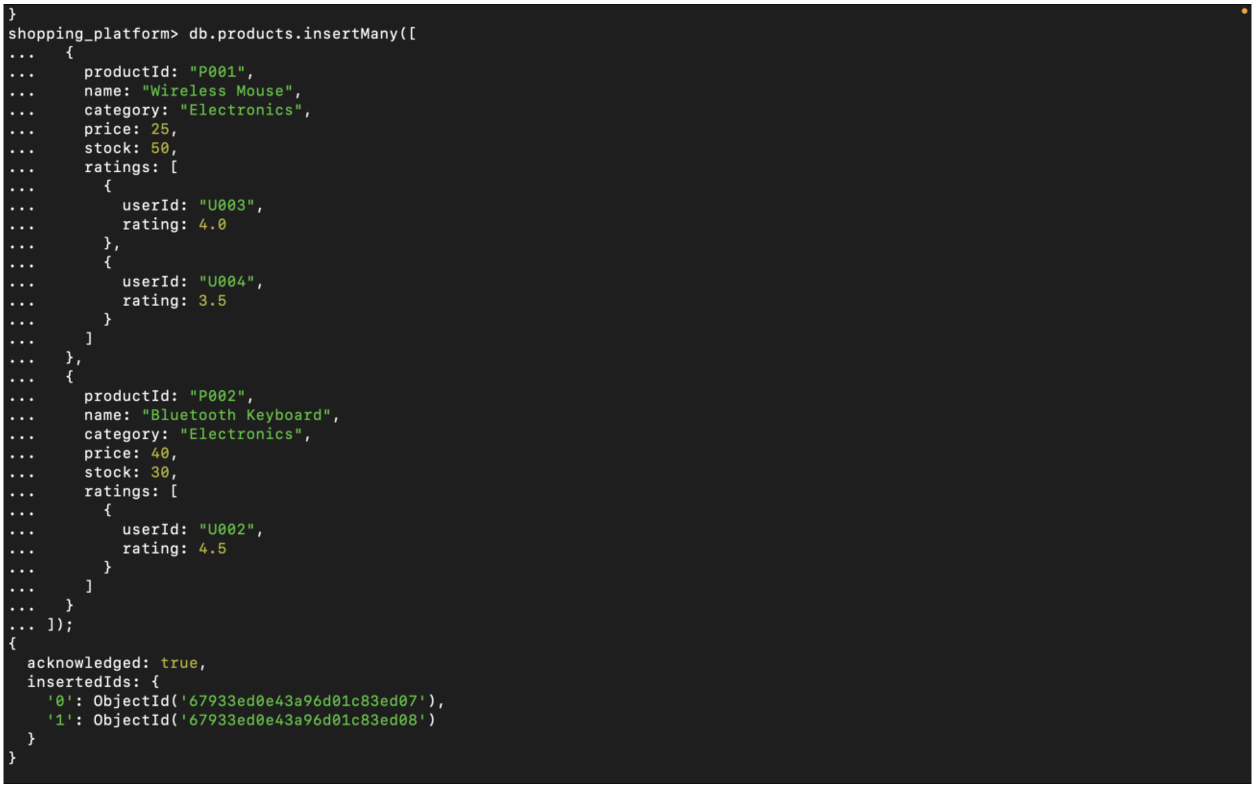
```
test> use shopping_platform
switched to db shopping_platform
shopping_platform> db.users.insertMany([
...   {
...     userId: "U001",
...     name: "Michael Johnson",
...     email: "michael.johnson@example.com",
...     age: 30,
...     address: {
...       city: "Chicago",
...       state: "IL",
...       zip: "60601"
...     },
...     createdAt: new Date("2024-01-03T09:45:00Z")
...   },
...   {
...     userId: "U002",
...     name: "Emily Davis",
...     email: "emily.davis@example.com",
...     age: 27,
...     address: {
...       city: "Houston",
...       state: "TX",
...       zip: "77001"
...     },
...     createdAt: new Date("2024-01-04T14:15:00Z")
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67933b58e43a96d01c83ed03'),
    '1': ObjectId('67933b58e43a96d01c83ed04')
  }
```

```
// 2. Create orders collection and insert sample data
db.orders.insertMany([
  {
    orderId: "ORD003",
    userId: "U003",
    orderDate: new Date("2024-12-20T10:15:00Z"),
    items: [
      {
        productId: "P003",
        quantity: 3,
        price: 75
      },
      {
        productId: "P004",
        quantity: 2,
        price: 40
      }
    ],
    totalAmount: 305,
    status: "Shipped"
  },
  {
    orderId: "ORD004",
    userId: "U004",
    orderDate: new Date("2024-12-22T16:30:00Z"),
    items: [
      {
        productId: "P005",
        quantity: 4,
        price: 60
      }
    ],
    totalAmount: 240,
    status: "Pending"
  }
]);
```



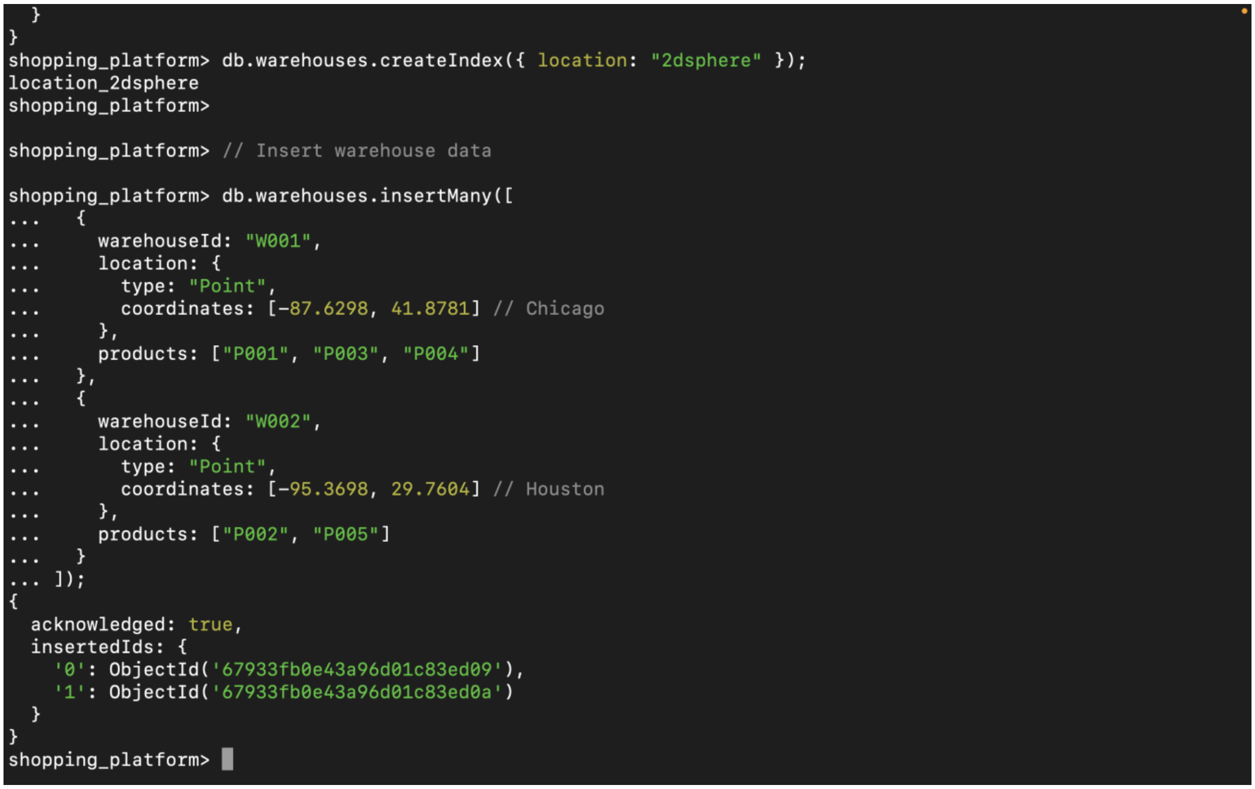
```
test> use shopping_platform
switched to db shopping_platform
shopping_platform> db.users.insertMany([
...   {
...     userId: "U001",
...     name: "Michael Johnson",
...     email: "michael.johnson@example.com",
...     age: 30,
...     address: {
...       city: "Chicago",
...       state: "IL",
...       zip: "60601"
...     },
...     createdAt: new Date("2024-01-03T09:45:00Z")
...   },
...   {
...     userId: "U002",
...     name: "Emily Davis",
...     email: "emily.davis@example.com",
...     age: 27,
...     address: {
...       city: "Houston",
...       state: "TX",
...       zip: "77001"
...     },
...     createdAt: new Date("2024-01-04T14:15:00Z")
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67933b58e43a96d01c83ed03'),
    '1': ObjectId('67933b58e43a96d01c83ed04')
  }
```

```
// 3. Create products collection and insert sample data
db.products.insertMany([

 {

   productId: "P001",

   name: "Wireless Mouse",

   category: "Electronics",

   price: 25,

   stock: 50,

   ratings: [

    {

      userId: "U003",

      rating: 4.0

    },

    {

      userId: "U004",
        rating: 3.5
      }
    ]
  },
  {
    productId: "P002",
    name: "Bluetooth Keyboard",
    category: "Electronics",
    price: 40,
    stock: 30,
    ratings: [
      {
        userId: "U002",
        rating: 4.5
      }
    ]
  }
]);
```



```
}
shopping_platform> db.products.insertMany([
...   {
...     productId: "P001",
...     name: "Wireless Mouse",
...     category: "Electronics",
...     price: 25,
...     stock: 50,
...     ratings: [
...       {
...         userId: "U003",
...         rating: 4.0
...       },
...       {
...         userId: "U004",
...         rating: 3.5
...       }
...     ]
...   },
...   {
...     productId: "P002",
...     name: "Bluetooth Keyboard",
...     category: "Electronics",
...     price: 40,
...     stock: 30,
...     ratings: [
...       {
...         userId: "U002",
...         rating: 4.5
...       }
...     ]
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67933ed0e43a96d01c83ed07'),
    '1': ObjectId('67933ed0e43a96d01c83ed08')
  }
}
```

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

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


// Insert warehouse data

db.warehouses.insertMany([

 {

   warehouseId: "W001",

   location: {

     type: "Point",

     coordinates: [-87.6298, 41.8781] // Chicago

   },

   products: ["P001", "P003", "P004"]

 },

 {
   warehouseId: "W002",
   location: {
    type: "Point",
    coordinates: [-95.3698, 29.7604] // Houston
   },
   products: ["P002", "P005"]
  }
]);
```



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

shopping_platform> // Insert warehouse data

shopping_platform> db.warehouses.insertMany([
...   {
...     warehouseId: "W001",
...     location: {
...       type: "Point",
...       coordinates: [-87.6298, 41.8781] // Chicago
...     },
...     products: ["P001", "P003", "P004"]
...   },
...   {
...     warehouseId: "W002",
...     location: {
...       type: "Point",
...       coordinates: [-95.3698, 29.7604] // Houston
...     },
...     products: ["P002", "P005"]
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67933fb0e43a96d01c83ed09'),
    '1': ObjectId('67933fb0e43a96d01c83ed0a')
  }
}
shopping_platform> 
```

Queries

1. Find High-Spending Users

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

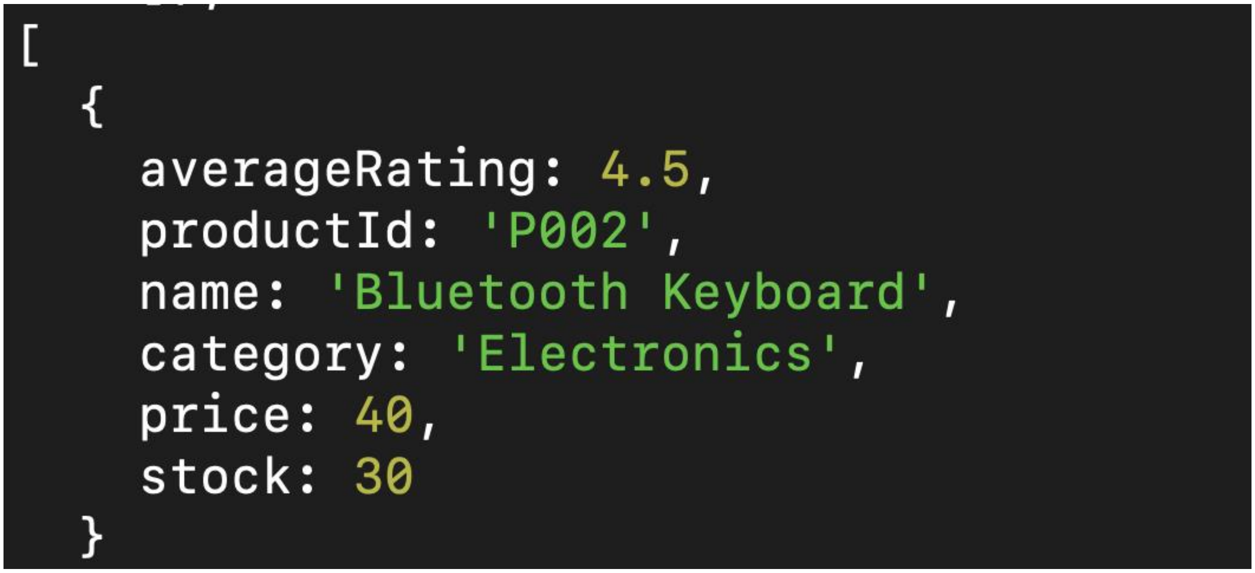
Hint: Use $lookup to join the users and orders collections and calculate the total Spending.

db.users.aggregate([

```
 {
  $lookup: {
   from: "orders",
   localField: "userId",
   foreignField: "userId",
   as: "userOrders"
  }
 },
 {
  $addFields: {
   totalSpent: {
    $sum: "$userOrders.totalAmount"
   }
  }
 },
 {
  $match: {
   totalSpent: { $gt: 200 }
  }
 },
 {
  $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
    }
  }
]);
```



```
[
  {
    averageRating: 4.5,
    productId: 'P002',
    name: 'Bluetooth Keyboard',
    category: 'Electronics',
    price: 40,
    stock: 30
  }
```

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

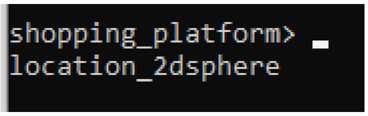
When an order is placed, reduce the stock of each product by the quantity in the

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" });
```

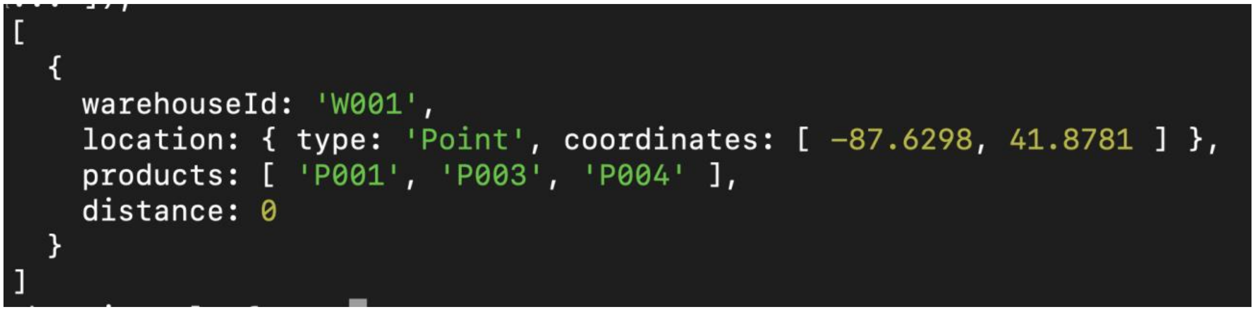


```
shopping_platform> 
location_2dsphere
```

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

```
db.warehouses.aggregate([
  {
    $geoNear: {
      near: {
        type: "Point",
        coordinates: [-87.6298, 41.8781]
      },
      distanceField: "distance",
      maxDistance: 50000,
      spherical: true,
      query: { products: "P001" }
    }
  },
  {
```

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



```
... ]);
[
  {
    warehouseId: 'W001',
    location: { type: 'Point', coordinates: [ -87.6298, 41.8781 ] },
    products: [ 'P001', 'P003', 'P004' ],
    distance: 0
  }
]
```