MANCHESTER 1824
The University of Manchester

DATA70141 - Understanding Databases
Amazone Database Design
Team 19

Minh Duc Pham
Emre Coskuner
Bader Elden Rada Shalata
Yuhang Xie
Hanxi Yu
Murui Xiao
Yuxiang Sun

MANCHESTER 1824
The University of Manchester

Table of Contents

- Project Goals and Aspirations
- Major tasks of the Project
- Collection Relationship Diagram and Database Schema
- Implement a demonstration database
- Design and implement queries to demonstrate the database

Project Goals and Aspirations

Design a NoSQL database for the Amazone online shopping website

- Analyze the requirements and develop the project plan
- Design a Collection Relationship Diagram for the Database
- Design the target Database Schema

Implement and demonstrate the database

- Implement a demonstration database with MongoDB
- Design and develop the database for MongoDB demonstration

- The project was executed using Agile methodologies to foster transparency, adaptability, and regular inspection within the project team
- The project was conducted by a squad of 7 analysts and engineers
- Progress was regularly monitored and updated regularly through appropriate Scrum ceremonies and artifacts
- Tools such as Trello, Microsoft Teams, and Miro were adopted to enhance project management and collaboration

MANCHESTER 1824
The University of Manchester

Major tasks of the Project

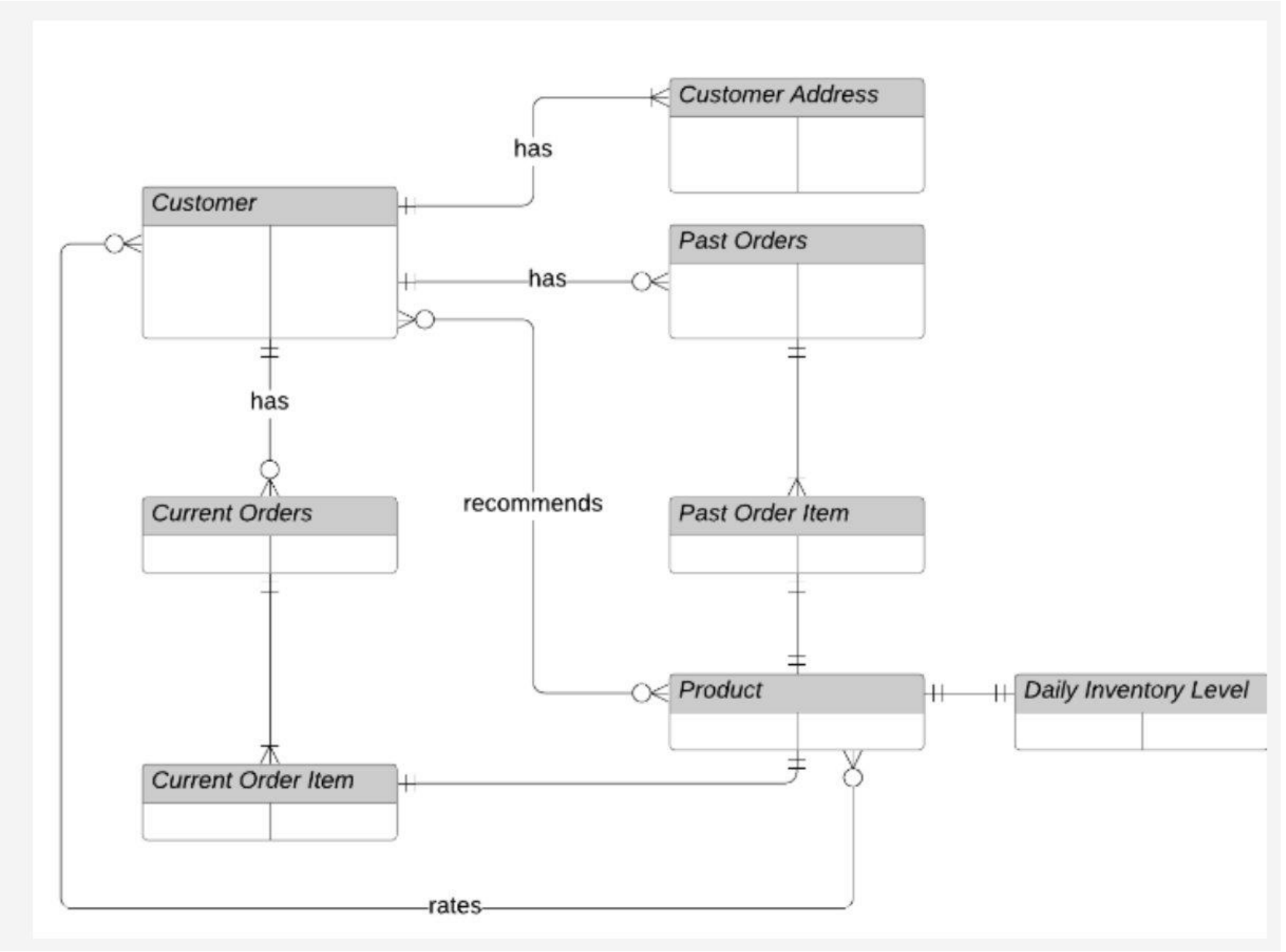
Major Tasks	Sub-tasks	Person In Charge
Requirements Analysis and Project Plan Development	▶ Analyze the given requirements for each collection and their relationships ▶ Identify required action items ▶ Develop a project plan to develop the database	▶ Team 19's analysts and developers
Design Collection Relationship Diagram	▶ Asses the current state of Amazone's old database ▶ Develop Collection Relationship Diagram ▶ Evaluate the differences between the old and new diagram	▶ Emre Coskuner ▶ Bader Elden Rada Shalata ▶ Hanxi Yu ▶ Yuxiang Sun
Design target Database Schema	▶ Develop the new Database Schema for Amazone ▶ Adopt the design of Embedding and Referencing techniques	▶ Emre Coskuner ▶ Bader Elden Rada Shalata ▶ Hanxi Yu ▶ Yuxiang Sun
Develop the database	▶ Evaluate and implement the defined Schema ▶ Generate and upload relevant document for each collection ▶ Index collections for efficient query	▶ Yuhang Xie ▶ Minh Duc Pham
Demonstrate the database with Queries	▶ Design queries to demonstrate the database ▶ Optimize scripts to enhance queries' performance	▶ Yuhang Xie ▶ Murui Xiao

All the action items were iteratively and incrementally revised and developed via Iterations

Old ERD

There are three main parts in the old ERD, 'Customer', 'Order' and ’ Product'.

The new schema is designed to support both the existing business and fresh product delivery services.



MANCHESTER 1824
The University of Manchester

New CRD

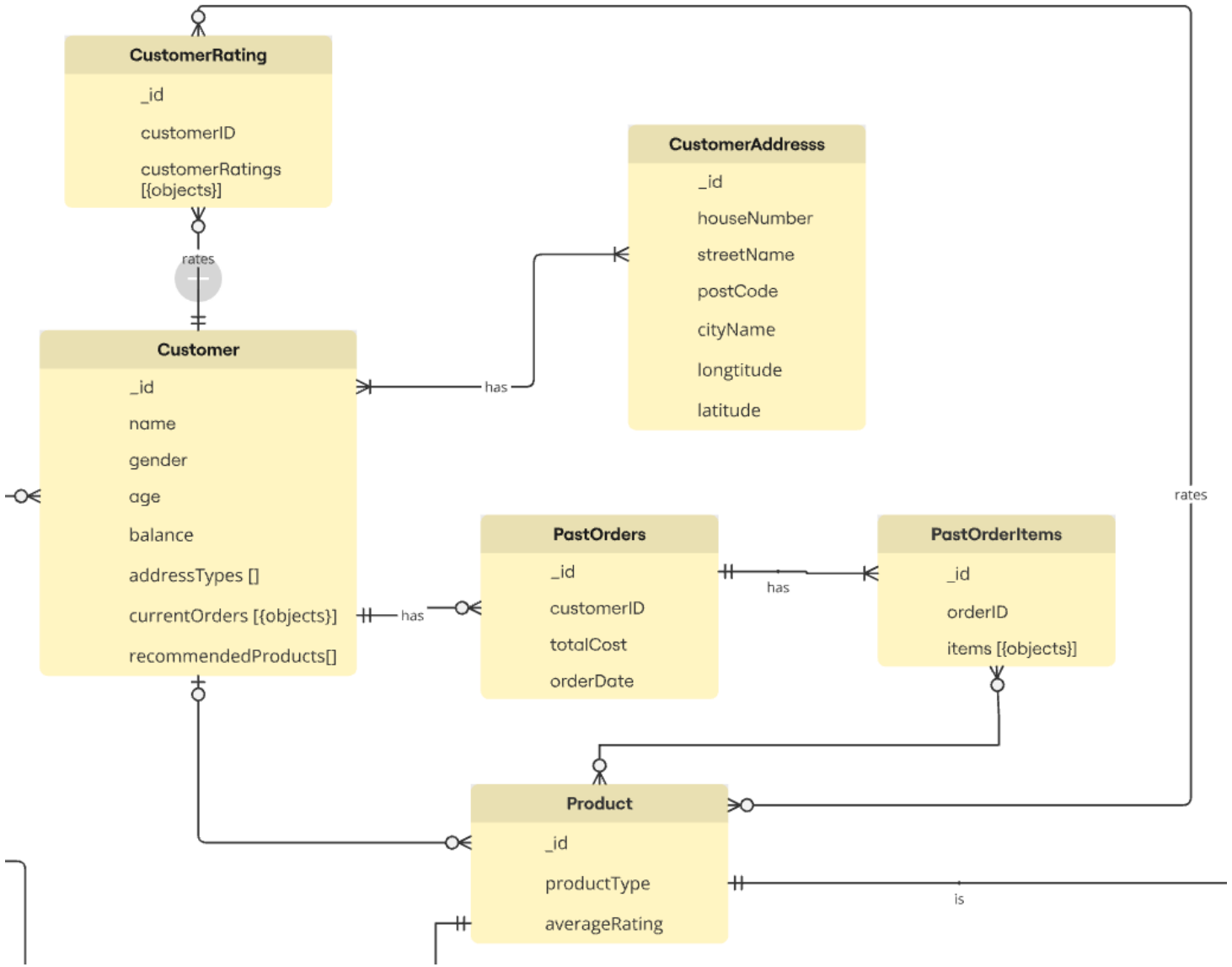
Schema modifications:

1. Embed 'Current Orders' and ’Current order item ‘ into 'Customer' collection

2. 'Product' collection is divided into 'Fresh' Product collection and the 'Other' Product collection.

3. Each category of products is stored in independent collection.

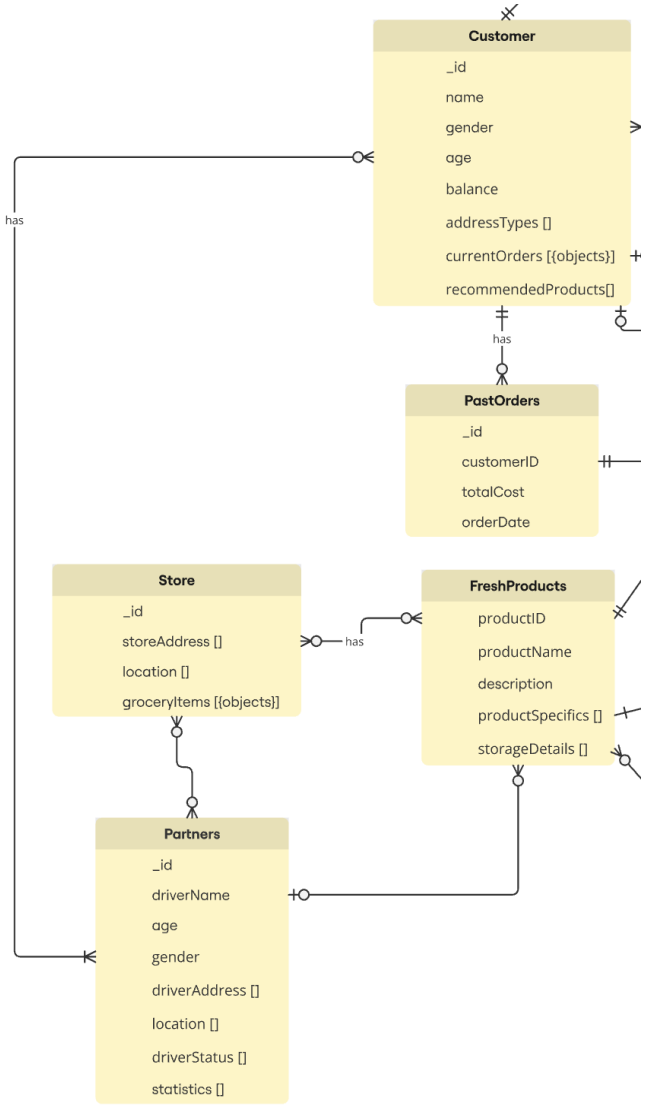
Customer related collections and relations




```
addresstype : Object
  shippingAddressID : ObjectId('6759d28aaf1004dc18de97aa')
  billingAddressID : ObjectId('6759d28aaf1004dc18de97aa')

currentOrders : Array (2)
  0: Object
    orderID : ObjectId('6759dd6ecbe1fc1a3c2d27dc')
    totalcost : 3
    deliveryPartnerID : ObjectId('675cc341b92053bcd803986b')
    products : Array (1)
      0: Object
        productID : ObjectId('6758c8a6333cf916c9322b62')
        productType : "Fresh"
        productName : "Almond Danish"
        quantityBought : 1
    orderDate : 2024-12-28T00:00:00.000+00:00
recommendedProducts : Array (2)
  0: Object
    productName : "Chocolate Croissant"
    averageRating : 3.67
    productID : ObjectId('6758c8a6333cf916c9322b61')
```

Fresh products related collections and relations

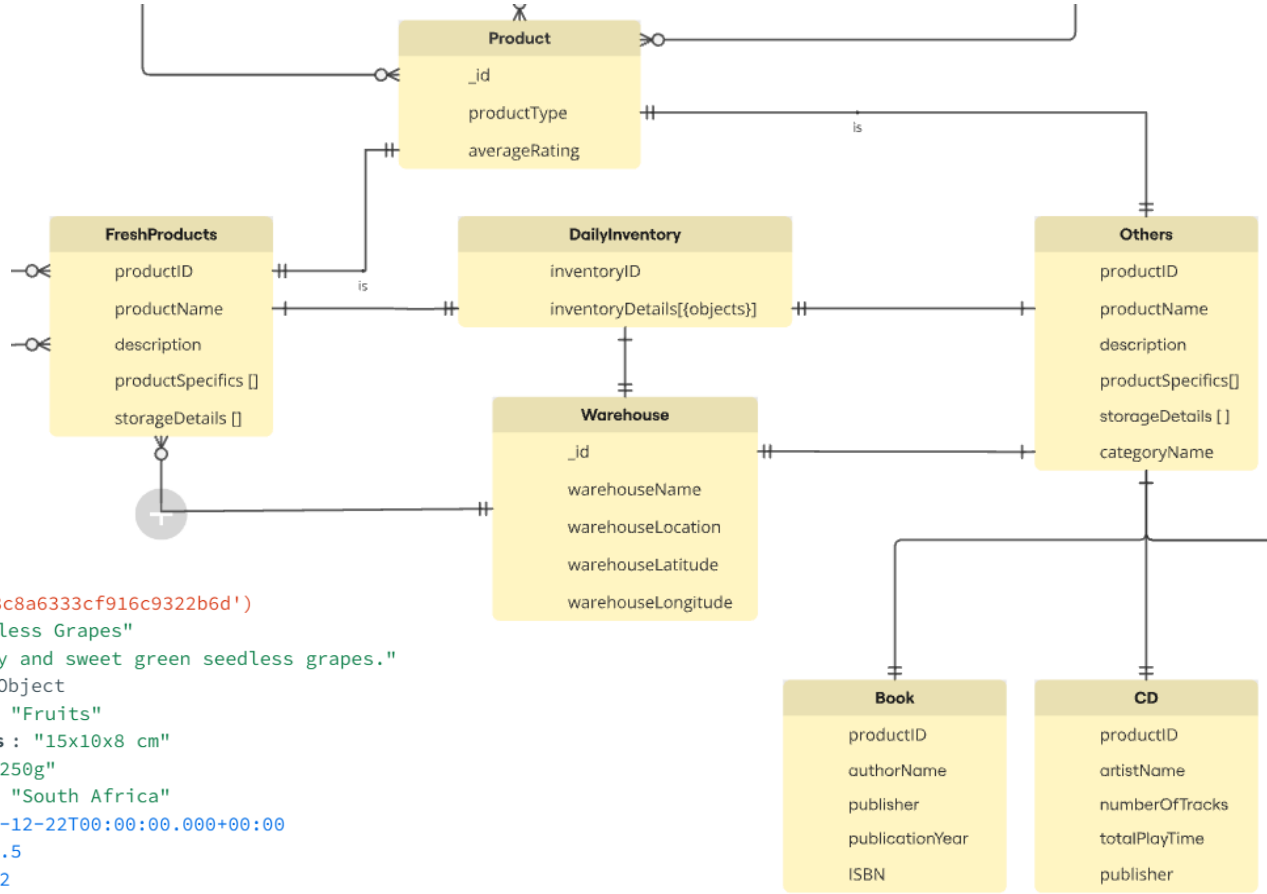



```
_id: ObjectId('6759c5a3154e92d1bb477993')
storeAddress : Object
    buildingNumber : 101
    streetName : "High Street"
    postCode : "M1 4AA"
    city : "Manchester"
location : Object
    latitude : 53.4808
    longitude : -2.2426
groceryItems : Array (2)
  0: Object
      productID : ObjectId('6758c8a6333cf916c9322b61')
      productName : "Chocolate Croissant"
      quantityInStore : 150
```

```
_id: ObjectId('6759b93457119e3dcc6cfb37')
driverName : "Cristiano Ronaldo"
age : 35
gender : "Male"
driverAddress : Object
    houseNumber : 12
    streetName : "Elm Street"
    city : "Manchester"
    postcode : "M1 1AB"
driverStatus : Object
    isActive : true
    onRoute : true
    pickedOrderID : ObjectId('6759dd6ecbe1fc1a3c2d27da')
    ETA : 2.2
location : Object
    latitude : 53.482
    longitude : -2.2445
statistics : Object
    averageRating : 4.8
    totalDeliveries : 1200
    earnings : 45000.5
```

```
_id: ObjectId('6758c8a6333cf916c9322b61')
productName : "Chocolate Croissant"
description : "A buttery and flaky croissant filled with rich chocolate."
productSpecifics : Object
    productCategory : "Bakery"
    productDimensions : "12x8x4 cm"
    productWeight : "85g"
    countryOfOrigin : "France"
    expiryDate : 2024-12-15T00:00:00.000+00:00
    standardPrice : 3
    costInMorrizon : 2
storageDetails : Object
    storageID : ObjectId('67589d831814df7d9eb60c9b')
    storageName : "Fresh Products Warehouse"
    storageLocation : "123 Deansgate, Manchester, UK"
```

Product related collections and relations




```
_id: ObjectId('6758c8a6333cf916c9322b9b')
productName : "Braun Multiquick Hand Blender"
description : "Compact and versatile hand blender for soups, smoothies, and more."
▾ productSpecifics : Object
   productDimensions : "7x7x39 cm"
   shippingWeight : "1.8kg"
   averagePrice : 80
   supplierCost : 65
▾ storageDetails : Object
   storageID : ObjectId('67589d831814df7d9eb60c9a')
   storageName : "Dale Street Warehouse"
   storageLocation : "Dale Street, Northern Quarter, Manchester, UK"
categoryName : "HomeAppliance"
```

```
_id: ObjectId('6758c8a6333cf916c9322b94')
color : "Black"
voltage : "220-240V"
style : "Digital Air Fryer"
```

```
_id: ObjectId('6758c8a6333cf916c9322b6d')
productName : "Seedless Grapes"
description : "Juicy and sweet green seedless grapes."
▾ productSpecifics : Object
   productCategory : "Fruits"
   productDimensions : "15x10x8 cm"
   productWeight : "250g"
   countryOfOrigin : "South Africa"
   expiryDate : 2024-12-22T00:00:00.000+00:00
   standardPrice : 2.5
   costInMorrizon : 2
▾ storageDetails : Object
   storageID : ObjectId('67589d831814df7d9eb60c9b')
   storageName : "Fresh Products Warehouse"
   storageLocation : "123 Deansgate, Manchester, UK"
```

Schema, JSON Files Structure

1 ObjectID for Unique Document Identification

To ensure unique identification for each document in the collection.

- Efficient for indexing and retrieval.
- Globally unique, avoiding conflicts in distributed systems.

2 Arrays for Collections of Related Data

To represent relationships like one-to-many or many-to-many within the same document.

- Simplifies queries by storing related entities together.
- Preserves the logical grouping of data.

Schema, JSON Files Structure

3 References for Related Entities

To establish relationships between the current collection and other collections without duplicating data.

- Enables linking between entities while maintaining data consistency.
- Reduces redundancy and improves storage efficiency.

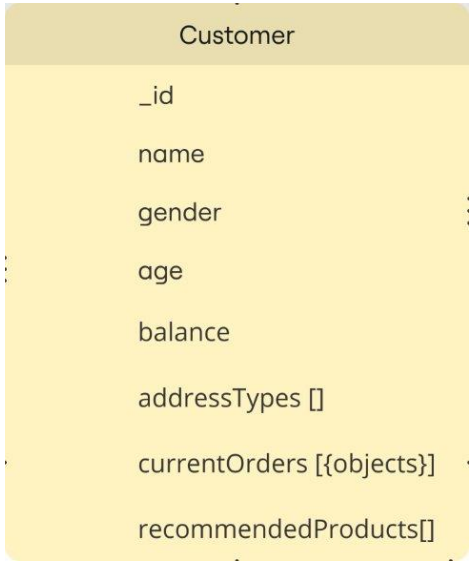
4 Nested Objects for Related Fields

To logically group related fields together in a single structure.

- Improves readability and maintainability.
- Avoids flat and repetitive designs.

Collections and Schemas

- Customer




```
_id: ObjectId('6759d559b156995774e8b89d')
name : "Taylor Swift"
gender : "Female"
age : "30"
balance : "1900"
addresstype : Object
  shippingAddressID : ObjectId('6759d28aaf1004dc18de97ac')
  billingAddressID : ObjectId('6759d28aaf1004dc18de97ac')
currentOrders : Array (2)
  0: Object
  1: Object
recommendedProducts : Array (2)
  0: Object
  1: Object
```

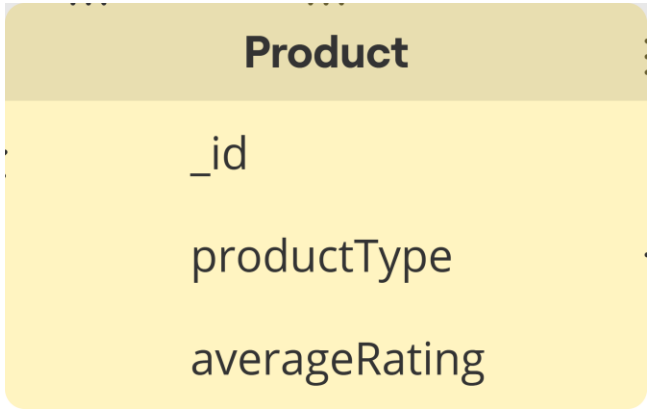
```
Customers
{
        "_id":  "ObjectID",
        "name": "string",
        "gender": "string",
        "age": "int",
        "balance": "int",
        "addresstype":
        {
                "shippingAddressID": "ref<customerAddresses._id>",
                "billingAddressID": "ref<customerAddresses._id>"
        },

        "currentOrders": "Object"
        [
                {
                        "orderid": "ObjectID",
                        "totalcost": "int",
                        "deliveryPartnerID": "ref<partners._id>",
                        "products": "Object"
                        [
                                {
                                        "productID": "ref<products._id>",
                                        "productType": "ref<products.productType>",
                                        "productName": "ref<freshProducts/others.productName>",
                                        "quantityBought":"int"
                                }
                        ],
                        "orderDate": "ISODate"
                }
        ],

        "recommendedProducts": "Object"
        [
                {
                        "productID": "ref<products._id>",
                        "productName": "ref<freshProducts/others.productName>",
                        "averageRating": "ref<products.averageRating>"
                }
        ]
}
```

Collections and Schemas

- Product




```
_id: ObjectId('6758c8a6333cf916c9322b61')
productType : "Fresh"
averageRating : 3.67
```

```
Product
{
        "_id": "ObjectID",
        "productType": "string",
        "averageRating": "double"

}
```

Collections and Schemas

- FreshProducts

FreshProducts

productID

productName

description

productSpecifics []

storageDetails []

```
_id: ObjectId('6758c8a6333cf916c9322b6d')
productName : "Seedless Grapes"
description : "Juicy and sweet green seedless grapes."
▾ productSpecifics : Object
    productCategory : "Fruits"
    productDimensions : "15x10x8 cm"
    productWeight : "250g"
    countryOfOrigin : "South Africa"
    expiryDate : 2024-12-22T00:00:00.000+00:00
    standardPrice : 2.5
    costInMorrizon : 2
▾ storageDetails : Object
    storageID : ObjectId('67589d831814df7d9eb60c9b')
    storageName : "Fresh Products Warehouse"
    storageLocation : "123 Deansgate, Manchester, UK"
```

```
FreshProducts
{
        "_id": "ref<product._id>",
        "productName":"string",
        "description": "string",
        "productSpecifics":
        {
                "productCategory": "string",
                "productDimensions": "string",
                "productWeight": "string",
                "countryOfOrigin": "string",
                "expiryDate": "ISODate",
                "standardPrice": "int",
                "costInMorrizon": "int",
        },

        "storageDetails":
        {
                "storageID" : "ref<warehouses._id>",
                "storageName": "ref<warehouses.warehouseName>",
                "storageLocation": "ref<warehouses.warehouseLocation>"
        }
}
```

Table Relations

- Partners

Partners
_id
driverName
age
gender
driverAddress []
location []
driverStatus []
statistics []

```
_id: ObjectId('6759b93457119e3dcc6cfb37')
driverName : "Cristiano Ronaldo"
age : 35
gender : "Male"
▾ driverAddress : Object
    houseNumber : 12
    streetName : "Elm Street"
    city : "Manchester"
    postcode : "M1 1AB"
▾ driverStatus : Object
    isActive : true
    onRoute : true
    pickedOrderID : ObjectId('6759dd6ecbe1fc1a3c2d27da')
    ETA : 2.2
▾ location : Object
    latitude : 53.482
    longitude : -2.2445
▾ statistics : Object
    averageRating : 4.8
    totalDeliveries : 1200
    earnings : 45000.5
```

Partners

```
{
        "_id": "ObjectId",
        "driverName": "string",
        "age" : "int",
        "gender": "string",
        "driverAddress":
        {
                "houseNumber": "int",
                "streetName": "string",
                "city": "string",
                "postcode": "string"
        },

        "driverStatus":
        {

                "isActive": "boolean",
                "onRoute": "boolean",
                "pickedOrderID": "ref<customers.currentOrders.orderID>",
                "ETA": "double"

        },

        "location":
        {
                "latitude":"double",
                "longtitude":"double"
        },

        "statistics":
        {
                "averageRating": "double",
                "totalDeliveries": "int",
                "earnings": "double"

        }
}
```

MANCHESTER 1824
The University of Manchester

Implement a demonstration database

New Database Schema

Real World Objects

The data for the platform were generated based on the newly-defined schema and relevant information of real - world objects.
All the generated data were stored in JSON files

Information that was both manually and computationally generated

- Some attributes were manually collected such as addresses, coordinates, product descriptions
- The information were all consolidated and transformed into JSON files

Information that was generated via Python Scripts

- Customers' ratings were randomly generated in a scale from 1 to 5
- ETA for the delivery of Fresh Products were automatically calculated
- Recommended Products were generated based on our algorithm

Load the data into the database 2

All the generated data in JSON format files were uploaded into MongoDB via Python Scripts

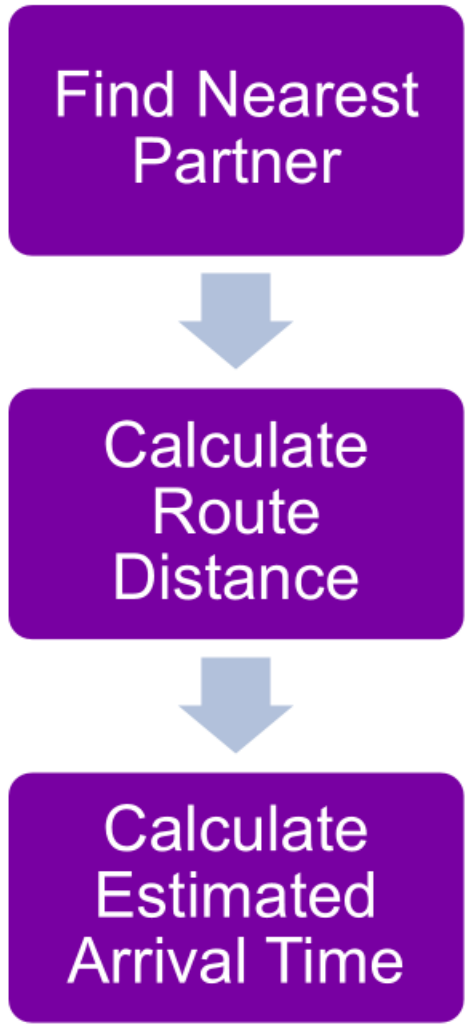
- Several iterations of testing were conducted to make sure that the demonstration database met the given requirements

Indexing collections for efficient querying 3

Geospatial attributes were utilized for indexing. Adopting this technique would enable the developer to gain certain advantages while dev eloping queries. The details of our usage and further evaluations are mentioned in the next slide

Indexing – Geospatial Attributes

- **Faster Query Performance**
  - Precomputes spatial relationships for reduced computational overhead.
- **Accurate Distance Calculations**
  - Uses spherical geometry for distance calculations.
- **Application in our case**
  - Search the nearest delivery partner to a store or customer.
  - Calculate the distance between points.



Usage of Indexing Geo-attributes

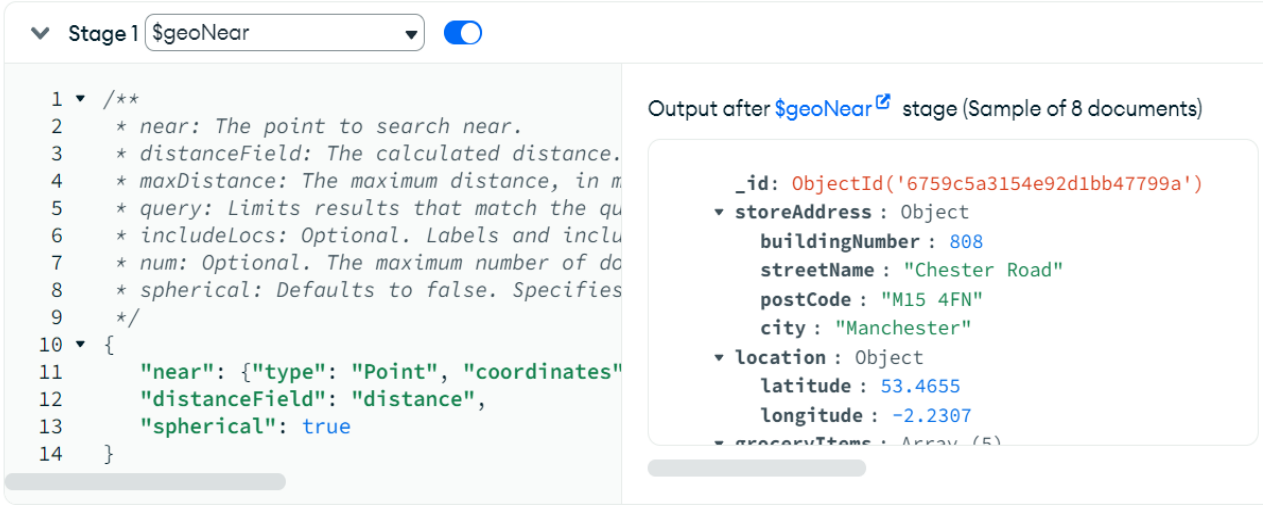
```
db.stores.createIndex({ location: "2dsphere" })
```

In this task, it is important to look up:

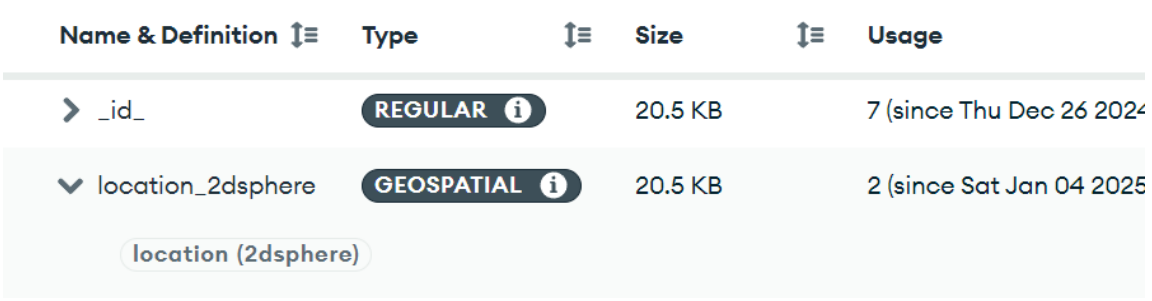
1) the stores near to the customer

2) the driver(partner) near to the store

3) the distance of delivery



So, indexing the geo-attributes could benefit the process of **Finding Nearest Objects** and **Calculate Route and Estimated Time**.



Queries – Customer Searching Products

Results of looking up


```
[
    {
        "customer": "John Doe",
        "nearest 5 stores": [
            {
                "address": "Chester Road",
                "postcode": "M15 4FN"
            },
            {
                "address": "Whitworth Street",
                "postcode": "M1 6EL"
            },
            {
                "address": "Oxford Road",
                "postcode": "M1 5AN"
            },
"fresh products available": [
    {
        "productName": "Carrots",
        "description": "Fresh and crunchy carr
        "price": 0.8
    },
    {
        "productName": "Spinach",
        "description": "Fresh leafy spinach, pa
        "price": 1.5
    },
```


Step 1: Find the nearest 5 stores
Using $geoNear to find the nearest 5 stores to the customers shipping address.

```python
def find_nearest_stores(db, customer_location, limit=5):
    """Find nearest stores to a customer's location."""
    pipeline = [
        {"$geoNear": {
            "near": {"type": "Point", "coordinates": [customer_location["longitude"], customer_location["latitude"]]},
            "distanceField": "distance",
            "spherical": True
        }},
        {"$limit": limit}
    ]
    return list(db.stores.aggregate(pipeline))
```

Step 2: Get available fresh products in the stores
Find all products that are in stock of the stores

Core Functions:
$geoNear, to calculate the distance
$project, to form the format of output

Queries – Create a New Fresh Order

Results of creating and assigning an order


```
{
    "Order Details": {
        "Time": "2024-12-14T17:47:45.356748",
        "Customer": "Olivia Wilde",
        "Product": "Almond Danish",
        "Quantity": 2,
        "Store": {
            "Store ID": "6759c5a3154e92d1bb477993",
            "Location": "High Street",
            "City": "Manchester",
            "Postcode": "M1 4AA"
        },
        "Assigned Driver": {
            "Name": "Virgil van Dijk",
            "Rating": 4.3,
            "TotalDeliveries": 1009
        },
        "Delivery Details": {
            "Total Distance (miles)": 5.86,
            "Estimated Delivery Time (minutes)": 5
        }
    }
```


Core Functions:
$geoNear, to calculate the distance
$lookup, to connect different collections

Step 1: Check the availability of the demanded product
Find if the product is in stock in the nearest 5 stores.
If available, create a new order for the customer

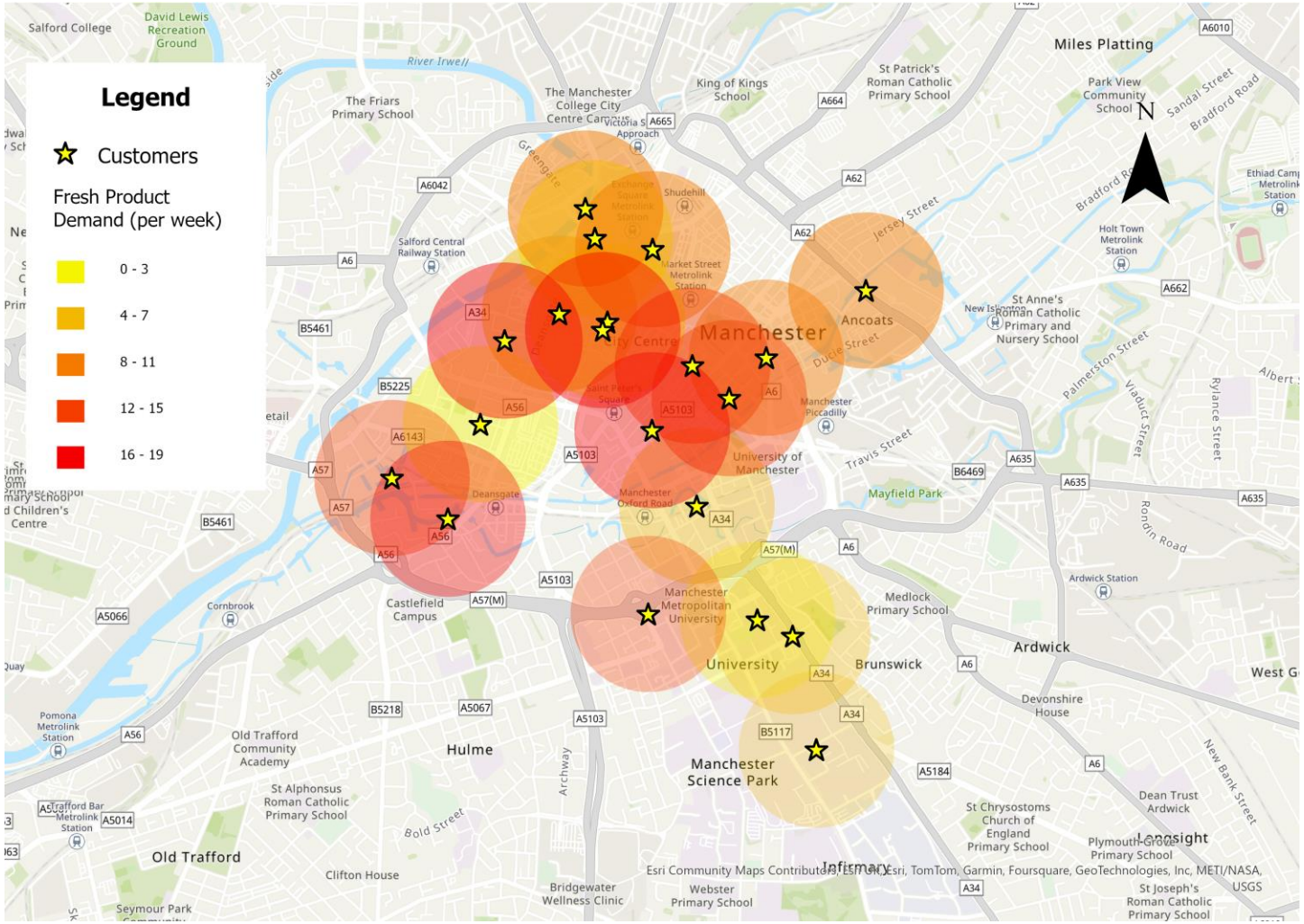
Step 2: Assign the order to a partner
Find the nearest partner to the store who is not occupied.
If there is any available partner, assign this order to the partner

---

Step 3: Calculate the route and time
Get geolocation of partner, store and customer's shipping address. Calculate the distance between these locations and calculate estimated arrival time.

Step 4: Update the document
Using $project function to form the output format of the required information

Queries – Demand Distribution on Map



Step 1: Calculate the Demand of Single customer

Sum up the order amount of every customer and the money they spent

Step 2: Get the Geospatial Data

Find the longitude and latitude of the shipping address of each customer

Step 3: Plot on Map

Using the cluster algorithm to analysis the communities of customer and the demand for fresh product delivery of each community

Core Functions:

$count, to calculate the amount of customer

'cluster_finder', from ArcGIS, to plot the map

Queries – Customer Searching orders

```
[
    {
        "Customer": "Kim Jane",
        "Balance": 2500,
        "Total Order Cost": 12,
        "Order Count": 2,
        "Orders": [
            {
                "Order ID": "6759dd6ecbe1fc1a3c2d27dc",
                "OrderCost": 3,
                "Products": [
                    {
                        "Product": "Almond Danish",
                        "Quantity": 1,
                        "productID": "6758c8a6333cf916c9322b62",
                        "productType": "Fresh"
                    }
                ]
            },
            {
                "Order ID": "6759dd6ecbe1fc1a3c2d27dd",
                "OrderCost": 9,
                "Products": [
                    {
                        "Product": "Pride and Prejudice",
                        "Quantity": 1,
                        "productID": "6758c8a6333cf916c9322b76",
                        "productType": "Other"
                    }
                ]
            }
        ]
    },
    {
        "Customer": "Liam Neeson",
        "Balance": 3500,
        "Total Order Cost": 968,
        "Order Count": 2,
```

```
'''
    Optional conditional field: orderCount: customer's order count
'''
def sucess_current_order(database, condition=None):
    collection = database['customers']
    pipeline = [
        {
            '$unwind': {
                'path': '$currentOrders'  # expand currentOrders
            }
        }, {
            "$group": {
                "_id": "$_id",  # group by _id
                "name": {"$first": "$name"},  # name
                "balance": {"$first": {"$toInt": "$balance"}},  # transfrom balance to int
                "totalCostSum": {"$sum": "$currentOrders.totalcost"},  # sum totalcost
                "orders": {"$push": "$currentOrders"},  # products
                "orderCount": {"$sum": 1}  # calculate order count
            }
        }, {
            "$match": {  # Filter items where totalCostSum is less than balance
                "$expr": {"$lt": ["$totalCostSum", "$balance"]}
            }
        }, {
            "$project": {
                "_id": 1,
                "name": 1,
                "balance": 1,
                "totalCostSum": 1,
                "orders": 1,
                "orderCount": 1
            }
        }
    ]

    if condition:
        pipeline.append({
            "$match": condition
        })
```

Step 1:Calculate the total sales and balance

$group to group by id to calculate total sales

Step 2: choose order balance > sales

All of this is successful payment's orders

Step 3: Optional Step: other condition

Determine whether there are other filter conditions based on the incoming parameters.

By constructing a function, only the individual conditional statements are required, and the successful payment of orders under various conditions is automatically output.

```
q1 = sucess_current_order(db)

condition2 = {"orders.products.productType": "Fresh"}
q2 = sucess_current_order(db, condition2)
```

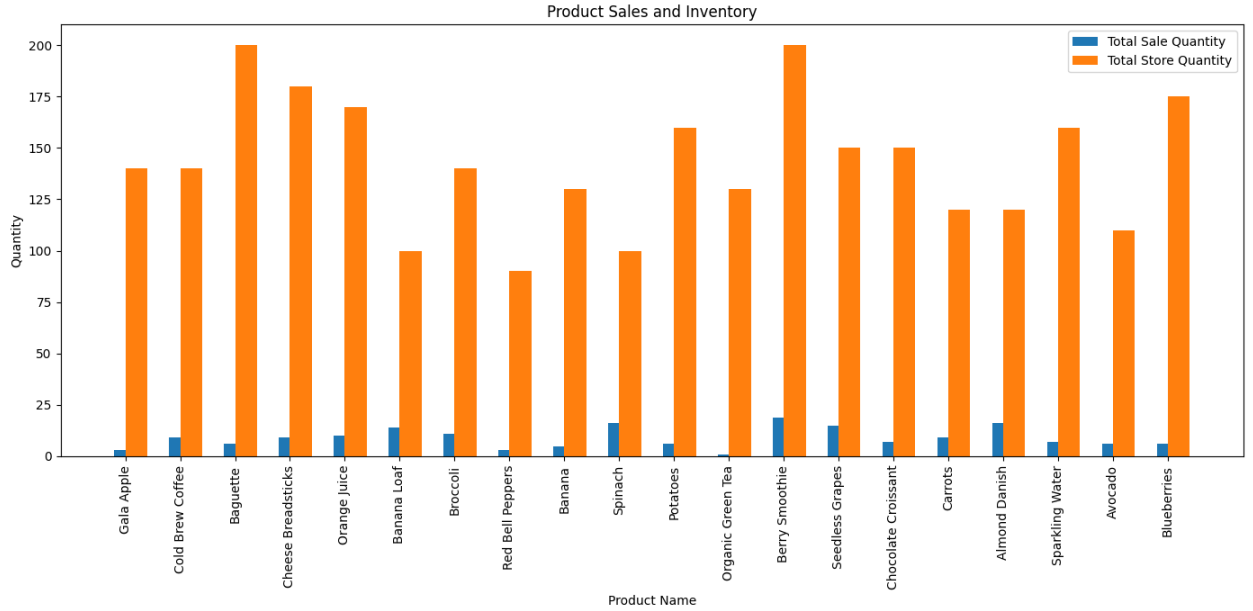
Queries – checking sales and inventory performance.

```python
def calculate_sales_and_inventory(collection_past_order_items):
    pipeline = [
        {
            '$unwind': {
                'path': '$items'
            }
        }, {
            "$lookup": {
                "from": "products",
                "localField": "items.productID",
                "foreignField": "_id",
                "as": "product_info"
            }
        }, {
            "$unwind": {
                "path": "$product_info"
            }
        }, {
            "$match": {
                "product_info.productType": "Fresh"  # choose productType is Fresh
            }
        }, {
            "$group": {
                "_id": "$items.productID",  # group by productID
                "totalsaleQuantity": {"$sum": "$items.itemQuantity"},
                "productName": {"$first": "$items.productName"},
                "productType": {"$first": "$product_info.productType"}
            }
        }, {
            "$lookup": {
                "from": "stores"
```

```python
# Calculate the sales volume and inventory of each product
sales_and_inventory = calculate_sales_and_inventory(collection_past_order_items)
print("Sales and Inventory:", sales_and_inventory)

# store the data in a json file
save_to_json(sales_and_inventory, 'sales_and_inventory.json')

# visualize the data
visualize_data(sales_and_inventory)
```



Step 1:Assign the category to product

Find the category from product collection by product_id to PastOrderItem

Step 2: Calculate the sales of product

Sum the sales group by product

Step 3: Assign the store information to a product

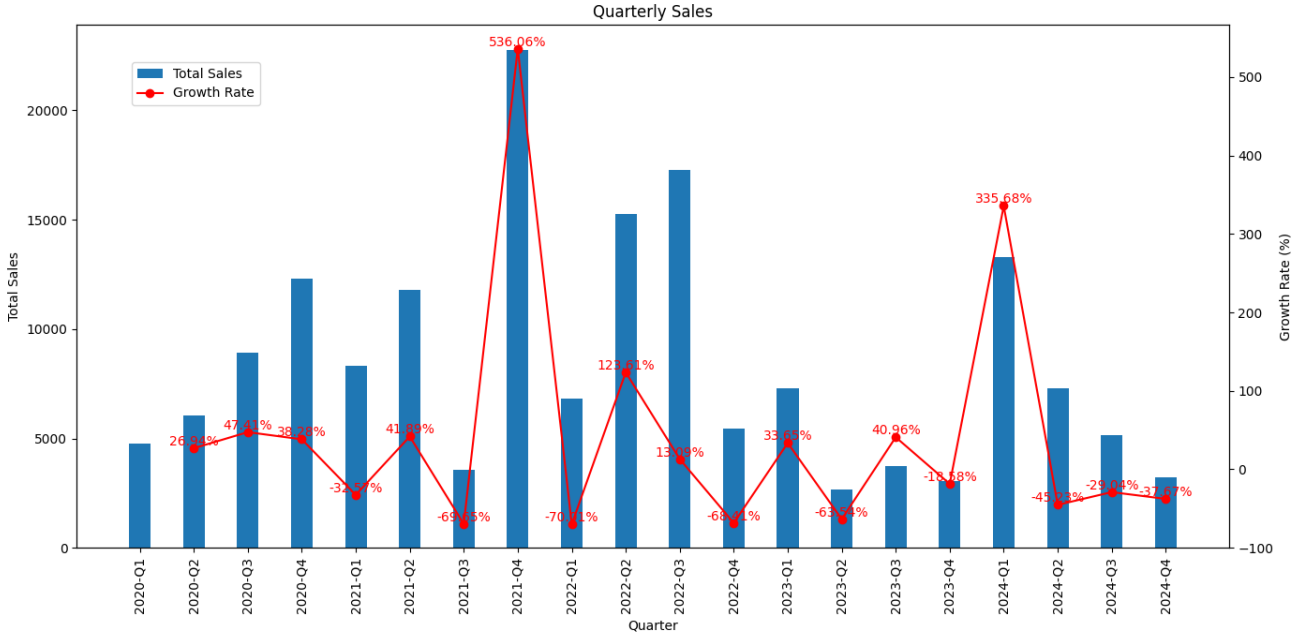
Finding inventory information from the store collection

Queries – Sales

```python
def calculate_quarterly_sales(collection, condition=None):
    pipeline = [
        {
            "$group": {
                "_id": {"year": {"$year": "$orderDate"}, "quarter": {"$ceil": {"$divide": [{"$month":
                "totalSales": {"$sum": "$totalCost"}
            }
        }, {
            "$sort": {
                "_id.year": 1,
                "_id.quarter": 1
            }
        }, {
            "$project": {
                "_id": 0,
                "quarter": {
                    "$concat": [
                        {"$toString": "$_id.year"}, "-Q",
                        {"$toString": "$_id.quarter"}
                    ]
                },
                "totalSales": 1
            }
        }
    ]

    if condition:
        pipeline.insert(0, {
            "$match": condition
        })

    result = collection.aggregate(pipeline)
    return list(result)
```



Step 1:Calculate the sales by year day month quarterly

Divided into months, years and quarters according to orderDate

Step 2: Calculate the growth rate of time

Calculate the growth rate

Conditional Automative Function

```python
def calculate_growth_rate(data):
    growth_rates = []
    for i in range(1, len(data)):
        previous = data[i - 1]['totalSales']
        current = data[i]['totalSales']
        growth_rate = ((current - previous) / previous) * 100 if previous != 0 else 0
        growth_rates.append(growth_rate)
    return growth_rates
```

```python
# Enquiry on monthly sales between 2020-06-20 and 2022-01-15
condition_example = {"orderDate": {"$gte": datetime(2020, 6, 20), "$lt": datetime(2022, 1, 15)}}
monthly_sales_con = calculate_monthly_sales(collection_past_order, condition_example)
print("Monthly Sales (2020-06-20 to 2022-01-15):", monthly_sales_con)
```

```python
def calculate_monthly_sales(collection, condition=None):...

def calculate_daily_sales(collection, condition=None):...

def calculate_total_sales(collection, condition=None):...

def calculate_yearly_sales(collection, condition=None):...

def calculate_quarterly_sales(collection, condition=None):...

def calculate_growth_rate(data):...
```

MANCHESTER 1824
The University of Manchester

Conclusion

- We have successfully designed and implemented a NoSQL database tailored for the Amazone online shopping platform, addressing both existing business operations and the integration of fresh product delivery services.
- By reengineering the schema, we achieved a more efficient structure with embedded relationships, improved query performance, and better data consistency through indexing and geospatial attributes.
- The database supports essential functionalities, such as customer searches, real-time order creation, and demand distribution analysis. These advancements optimize operational workflows and provide valuable insights for strategic decision-making.
- This project not only demonstrates technical proficiency in NoSQL database design but also showcases its practical application in solving real-world e-commerce challenges

Thank you for your attention