MANCHESTER
1824
The University of Manchester

Amazone

DATA70141 | UNDERSTANDING DATABASES

STUDENT IDS: 11356488 | 11358614 | 11361155 | 11150456 | 11493842

Introduction

With Amazone's strategic initiative to expand its service offerings to include fresh groceries, the company has entered into a partnership with Morrizon, a prominent UK grocery retailer.

This collaboration aims to introduce same-day and instant grocery delivery services, initially launching in Manchester.

To support this expansion, our team has been engaged to develop an integrated online platform that seamlessly incorporates both the existing and new business models.

The platform will facilitate efficient operations across six selected Morrizon grocery stores, enabling instant pick-up and delivery by Amazon's partnered delivery drivers.

The data modeling and implementation strategy for the enhanced platform is discussed in the upcoming slides.

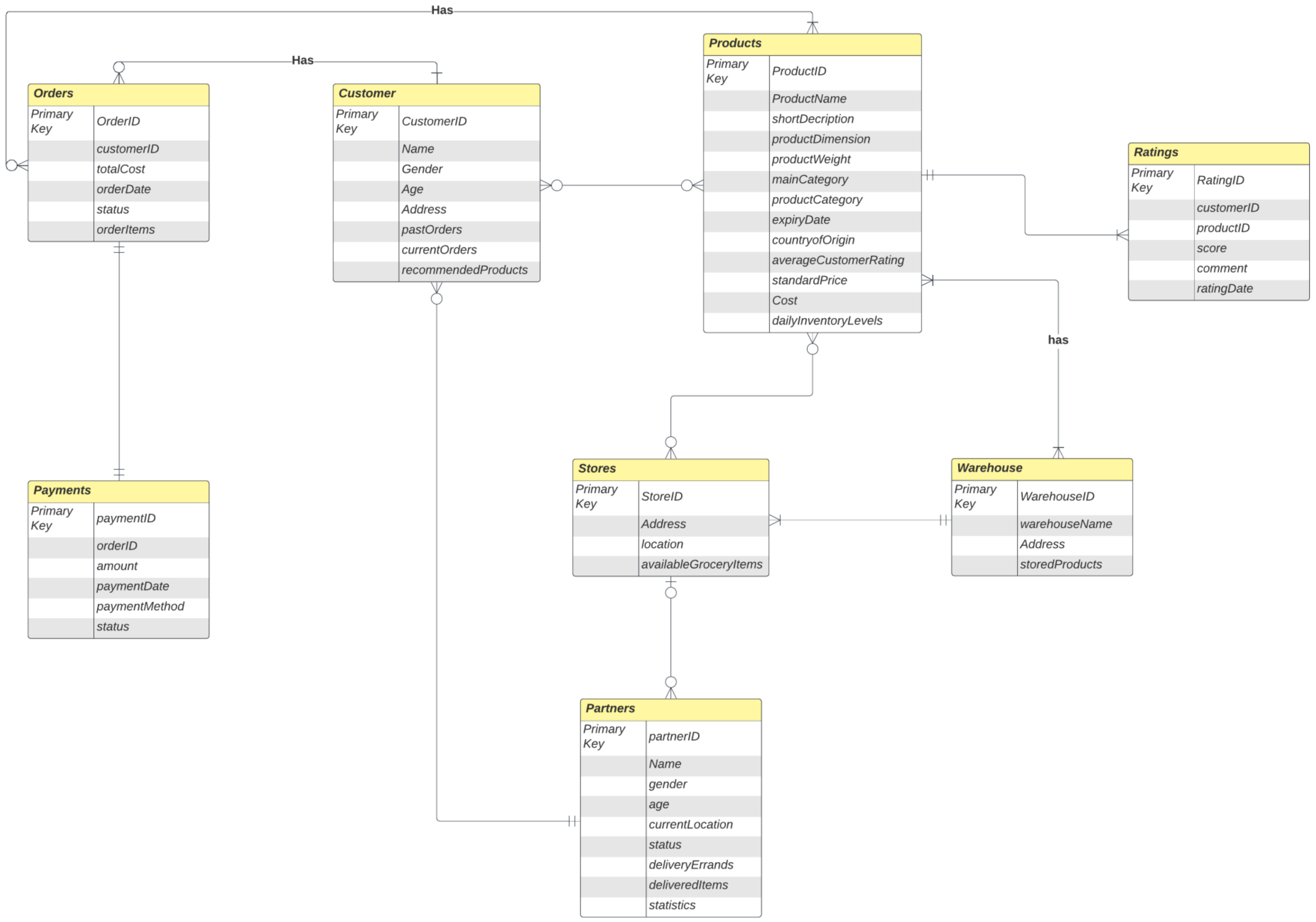
Partitioning of Tasks

TASKS	
Database Modelling and Design	All 5 members exchanged ideas
Designing Collections 1.Product 2.Customers 3.Orders 4.Rating 5.Store 6.Warehouse 7.Partners 8.Payments	Product – Vedaant Customer – Kashish, Tanya Orders – All contributed Rating - Rohit Store - Abhishek Warehouse - Abhishek Partners – Tanya Payments - Kashish
Populating Data into Collections	All 5 members
Querying	All 5 members divided 2 queries each
Querying Using MongoDB pipelines	Abhishek
Querying Using Python Scripts	Vedaant, Kashish, Tanya
Visualizations using matplotlib	Kashish, Tanya, Rohit
Entity Relationship Diagram	Abhishek, Kashish
Presentation	All 5 members

Database Design and Implementation

- Entity Relationship Diagram
- Description of reasons behind design decisions

ERD


Has
Has
has
Orders
Primary Key
OrderID
customerID
totalCost
orderDate
status
orderItems
Customer
Primary Key
CustomerID
Name
Gender
Age
Address
pastOrders
currentOrders
recommendedProducts
Products
Primary Key
ProductID
ProductName
shortDecription
productDimension
productWeight
mainCategory
productCategory
expiryDate
countryofOrigin
averageCustomerRating
standardPrice
Cost
dailyInventoryLevels
Ratings
Primary Key
RatingID
customerID
productID
score
comment
ratingDate
Payments
Primary Key
paymentID
orderID
amount
paymentDate
paymentMethod
status
Stores
Primary Key
StoreID
Address
location
availableGroceryItems
Warehouse
Primary Key
WarehouseID
warehouseName
Address
storedProducts
Partners
Primary Key
partnerID
Name
gender
age
currentLocation
status
deliveryErrands
deliveredItems
statistics

Description of reasons behind design decisions

- **Customer Schema**
  - Unique customer identification using ObjectId.
  - Multiple addresses per customer for flexibility.
  - References to past and current orders, ratings, and recommended products for personalized services.
- **Order Schema**
  - Direct link to customers for easy tracking.
  - Detailed order items, including product ID and quantity, for comprehensive order management.
- **Product Schema**
  - Unique product identification and detailed attributes for varied product types.
  - Inventory levels tracking linked to warehouse locations for efficient stock management.
- **Warehouse Schema**
  - Location-based organization for effective inventory distribution.
  - Links to stored products for quick access and management.
- **Rating Schema**
  - Connections to both customers and products for accurate feedback collection and analysis.
- **Partner (Delivery) Schema**
  - Real-time location tracking for efficient delivery management.
  - Detailed delivery history for performance tracking and payout calculations.
- **Store Schema**
  - Physical address and geolocation for optimized store placement and delivery routing.
  - Inventory details for available grocery items for stock level management.

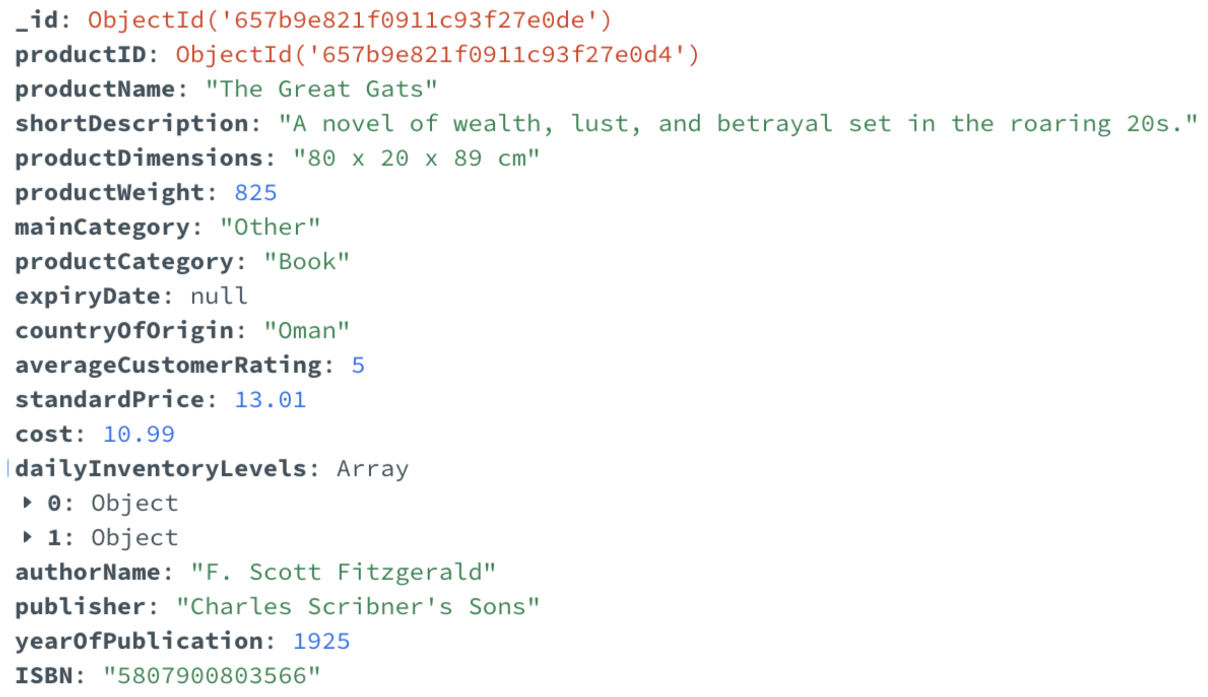
COLLECTIONS

-Sample data implemented

Products

Product Collection: Key Attributes and functions

- Faker library for data insertion, **country of Origin, ISBN, product dimensions and weight.**
- **Product Name, Description, Price** and specific attributes like author name, artist name etc., has been curated manually to make sure the data is as real as possible.
- **bson.ObjectId()** used for generation of **productID**
- **Products have been categorized into Other and Fresh**



```
_id: ObjectId('657b9e821f0911c93f27e0de')
productID: ObjectId('657b9e821f0911c93f27e0d4')
productName: "The Great Gats"
shortDescription: "A novel of wealth, lust, and betrayal set in the roaring 20s."
productDimensions: "80 x 20 x 89 cm"
productWeight: 825
mainCategory: "Other"
productCategory: "Book"
expiryDate: null
countryOfOrigin: "Oman"
averageCustomerRating: 5
standardPrice: 13.01
cost: 10.99
dailyInventoryLevels: Array
  ▸ 0: Object
  ▸ 1: Object
authorName: "F. Scott Fitzgerald"
publisher: "Charles Scribner's Sons"
yearOfPublication: 1925
ISBN: "5807900803566"
```

Customers

Customer Collection: Key Attributes and functions

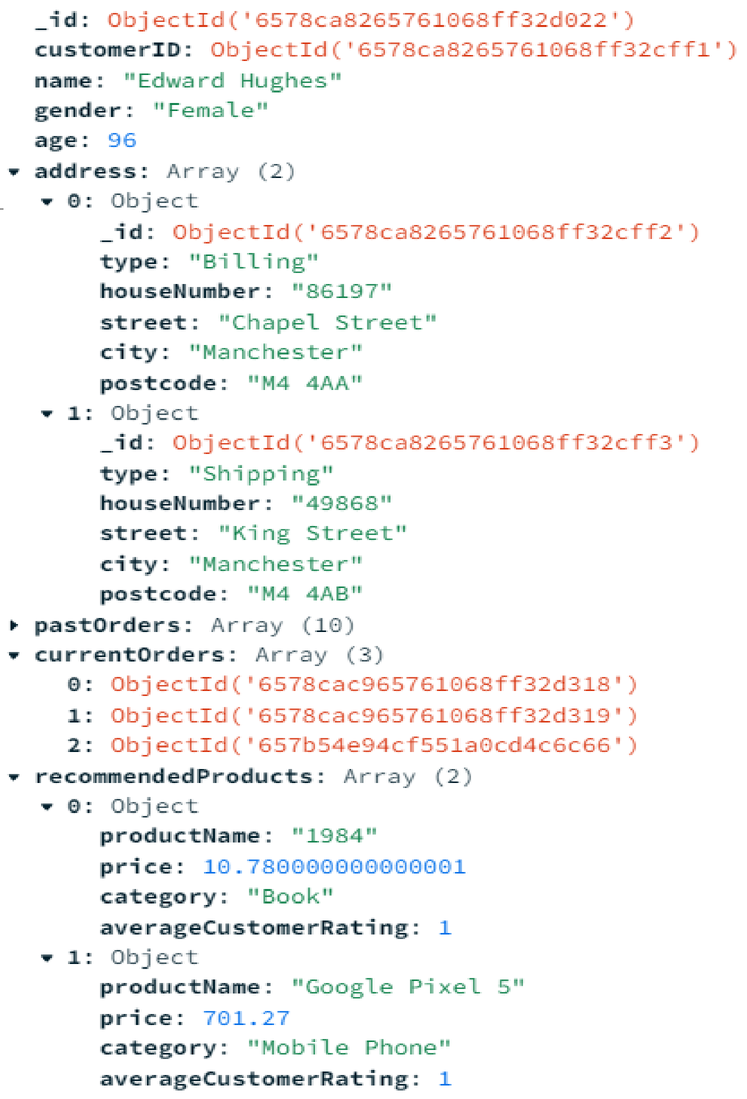
- Faker library for data insertion, name, age, **Manchester specific addresses**

- Each Customer can have separate billing and shipping addresses or a common billing/shipping address.

- **currentOrders** attribute is an array of current in-transit orders which references the **'Orders'** collection.

- **pastOrders** attribute is an array of delivered orders which references the **'Orders'** collection

- We designed a code to update the **recommendedProducts** feature based on the top-rated products for each customer



```
  _id: ObjectId('6578ca8265761068ff32d022')
  customerID: ObjectId('6578ca8265761068ff32cff1')
  name: "Edward Hughes"
  gender: "Female"
  age: 96
▾ address: Array (2)
  ▾ 0: Object
      _id: ObjectId('6578ca8265761068ff32cff2')
      type: "Billing"
      houseNumber: "86197"
      street: "Chapel Street"
      city: "Manchester"
      postcode: "M4 4AA"
  ▾ 1: Object
      _id: ObjectId('6578ca8265761068ff32cff3')
      type: "Shipping"
      houseNumber: "49868"
      street: "King Street"
      city: "Manchester"
      postcode: "M4 4AB"
▸ pastOrders: Array (10)
▾ currentOrders: Array (3)
    0: ObjectId('6578cac965761068ff32d318')
    1: ObjectId('6578cac965761068ff32d319')
    2: ObjectId('657b54e94cf551a0cd4c6c66')
▾ recommendedProducts: Array (2)
  ▾ 0: Object
      productName: "1984"
      price: 10.780000000000001
      category: "Book"
      averageCustomerRating: 1
  ▾ 1: Object
      productName: "Google Pixel 5"
      price: 701.27
      category: "Mobile Phone"
      averageCustomerRating: 1
```

Orders

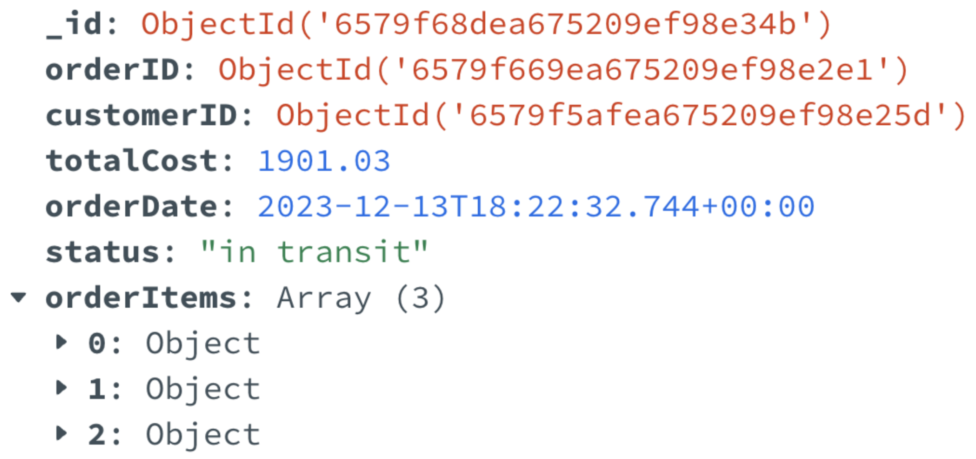
Collection Orders. Attributes:
Order details like totalCost, orderDate and transit status.

- The attribute totalCost denotes the cost of the order paid by the buyer.

- orderDate denotes the date in the format yyyy-mm-dd(year month date), along with the exact time of the purchase.

- Status mentions whether the order has been delivered or is still in transit.

- While orderItems shows the number of individual products that are part of a given order. This array has a minimum of 1 product per order.



```
_id: ObjectId('6579f68dea675209ef98e34b')
orderID: ObjectId('6579f669ea675209ef98e2e1')
customerID: ObjectId('6579f5afea675209ef98e25d')
totalCost: 1901.03
orderDate: 2023-12-13T18:22:32.744+00:00
status: "in transit"
▾ orderItems: Array (3)
  ▸ 0: Object
  ▸ 1: Object
  ▸ 2: Object
```

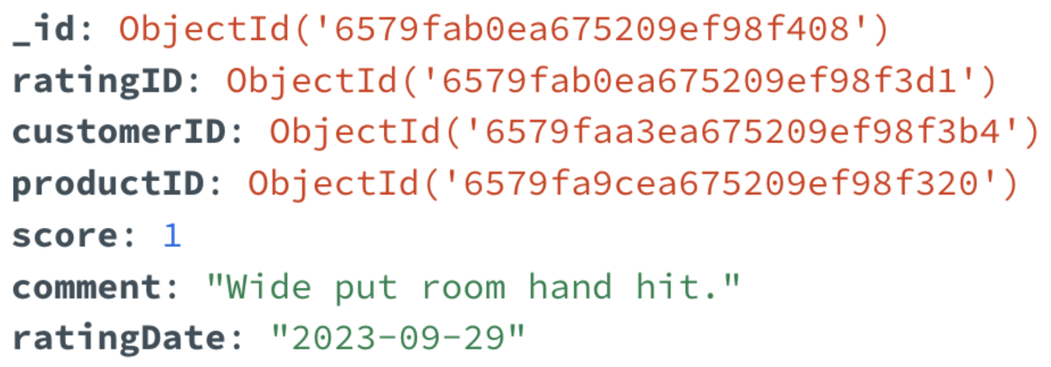
Ratings

Collection Ratings. Attributes related to Ratings:

- score indicates a rating score that the customer gives after they have received the product, on a scale of 1 to 5.

- comment is a string that the customer inputs along with the score, in the form of a product review.

- ratingDate denotes the date on which the customer rated and reviewed the product.



```
_id: ObjectId('6579fab0ea675209ef98f408')
ratingID: ObjectId('6579fab0ea675209ef98f3d1')
customerID: ObjectId('6579faa3ea675209ef98f3b4')
productID: ObjectId('6579fa9cea675209ef98f320')
score: 1
comment: "Wide put room hand hit."
ratingDate: "2023-09-29"
```

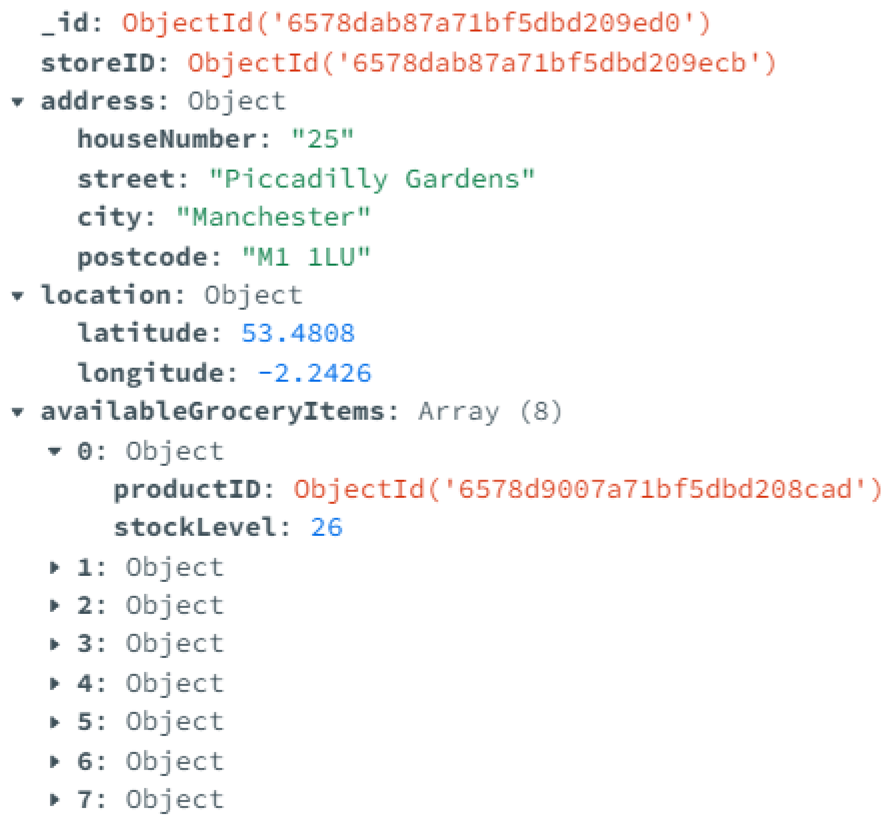
Stores

Collection **Stores**
Attributes:
- **storeID**: An identifier for the morrizons store.

- **address**: An object containing details about the store's address.

- **location**: An object providing the geographical coordinates of the store.

- **availableGroceryItems**: An array containing information about the available grocery items in store. Each item is represented as an object with:
  - **productID** – identifier of the grocery product.
  - **stockLevel** – quantiy of the product available in the store.



```
  _id: ObjectId('6578dab87a71bf5dbd209ed0')
  storeID: ObjectId('6578dab87a71bf5dbd209ecb')
▾ address: Object
    houseNumber: "25"
    street: "Piccadilly Gardens"
    city: "Manchester"
    postcode: "M1 1LU"
▾ location: Object
    latitude: 53.4808
    longitude: -2.2426
▾ availableGroceryItems: Array (8)
  ▾ 0: Object
      productID: ObjectId('6578d9007a71bf5dbd208cad')
      stockLevel: 26
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
  ▸ 5: Object
  ▸ 6: Object
  ▸ 7: Object
```

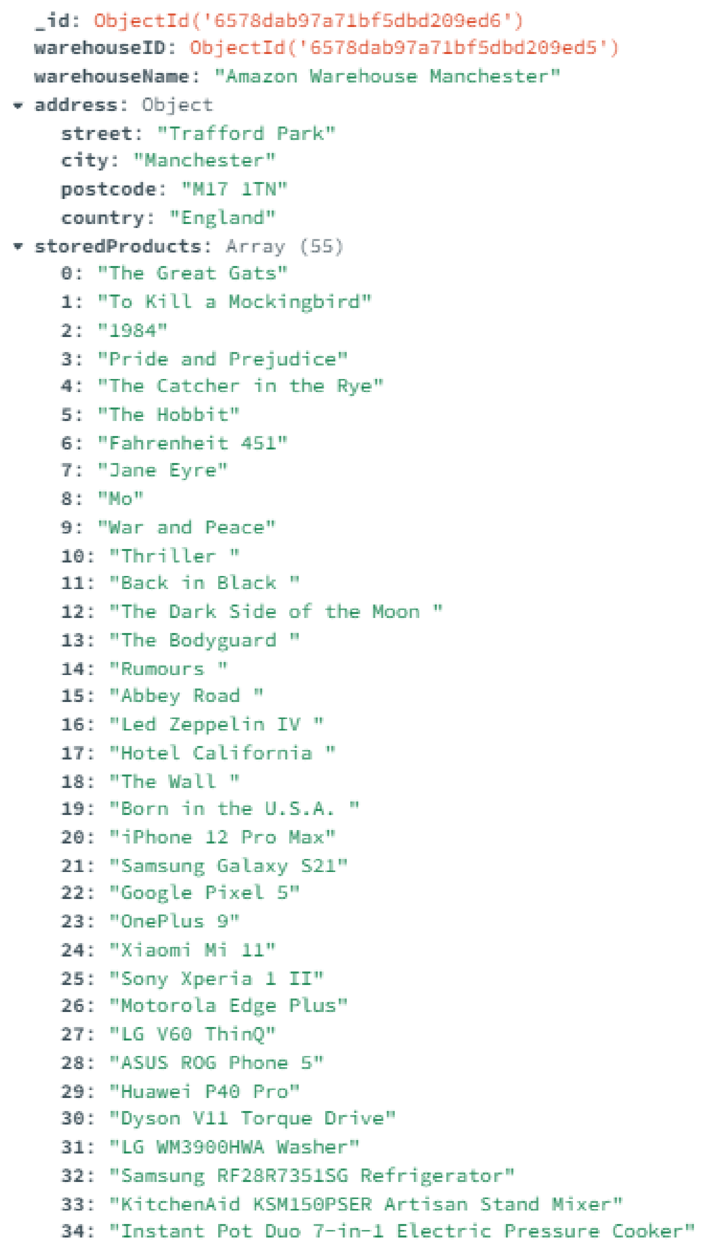
Warehouse

Collection **Warehouse**

Attributes:

- **warehouseID:** unique identifier for the warehouse
- **address:** An object containing details about the warehouse's address.
- **storedProducts:** A list of product names that are stored in the warehouse.

Warehouse fetches all product names from the products collection and creates a list (product_names_list) containing the product names.



```
  _id: ObjectId('6578dab97a71bf5dbd209ed6')
  warehouseID: ObjectId('6578dab97a71bf5dbd209ed5')
  warehouseName: "Amazon Warehouse Manchester"
▾ address: Object
    street: "Trafford Park"
    city: "Manchester"
    postcode: "M17 1TN"
    country: "England"
▾ storedProducts: Array (55)
    0: "The Great Gats"
    1: "To Kill a Mockingbird"
    2: "1984"
    3: "Pride and Prejudice"
    4: "The Catcher in the Rye"
    5: "The Hobbit"
    6: "Fahrenheit 451"
    7: "Jane Eyre"
    8: "Mo"
    9: "War and Peace"
    10: "Thriller "
    11: "Back in Black "
    12: "The Dark Side of the Moon "
    13: "The Bodyguard "
    14: "Rumours "
    15: "Abbey Road "
    16: "Led Zeppelin IV "
    17: "Hotel California "
    18: "The Wall "
    19: "Born in the U.S.A. "
    20: "iPhone 12 Pro Max"
    21: "Samsung Galaxy S21"
    22: "Google Pixel 5"
    23: "OnePlus 9"
    24: "Xiaomi Mi 11"
    25: "Sony Xperia 1 II"
    26: "Motorola Edge Plus"
    27: "LG V60 ThinQ"
    28: "ASUS ROG Phone 5"
    29: "Huawei P40 Pro"
    30: "Dyson V11 Torque Drive"
    31: "LG WM3900HWA Washer"
    32: "Samsung RF28R7351SG Refrigerator"
    33: "KitchenAid KSM150PSER Artisan Stand Mixer"
    34: "Instant Pot Duo 7-in-1 Electric Pressure Cooker"
```

Partners

Collection Partners. Attributes related to partners:

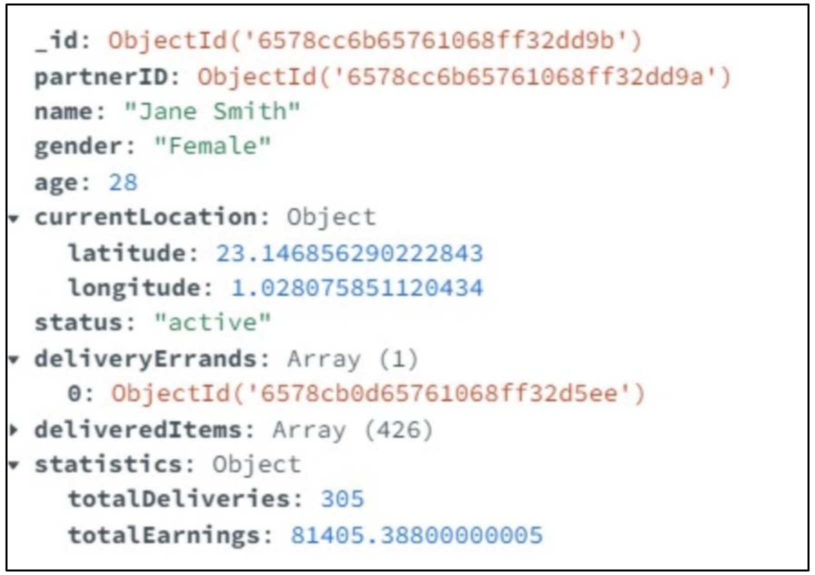
- currentLocation provides the latitude and longitude details of the delivery partner

- The status defines whether the delivery partner is active or idle.

- If the partner is active then deliveryErrand field gives the objectID of the product that the partner is delivering.

- DeliveredItems is an array of products that the delivery partner has already delivered.

- Statistics displays the total deliveries made and the total earnings of the partner.

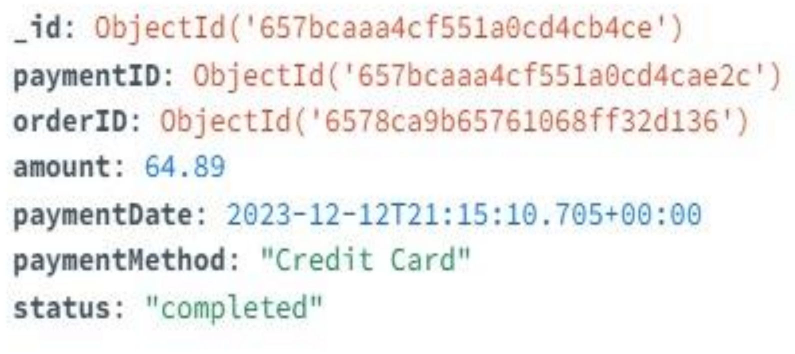


```
 _id: ObjectId('6578cc6b65761068ff32dd9b')
 partnerID: ObjectId('6578cc6b65761068ff32dd9a')
 name: "Jane Smith"
 gender: "Female"
 age: 28
▾ currentLocation: Object
   latitude: 23.146856290222843
   longitude: 1.028075851120434
 status: "active"
▾ deliveryErrands: Array (1)
   0: ObjectId('6578cb0d65761068ff32d5ee')
▸ deliveredItems: Array (426)
▾ statistics: Object
   totalDeliveries: 305
   totalEarnings: 81405.38800000005
```

Payments

Collection **Payments**
Attributes:
- **paymentID:** ObjectID is generated for paymentID
- **orderID**: The ID of the order
- **paymentMethod**: A randomly chosen payment method (Credit Card, PayPal, Bitcoin)

The code assumes that the "totalCost" attribute is present in orders and generates payments based on that. Additionally, we are using randomization for payment method.



```
_id: ObjectId('657bcaaa4cf551a0cd4cb4ce')
paymentID: ObjectId('657bcaaa4cf551a0cd4cae2c')
orderID: ObjectId('6578ca9b65761068ff32d136')
amount: 64.89
paymentDate: 2023-12-12T21:15:10.705+00:00
paymentMethod: "Credit Card"
status: "completed"
```

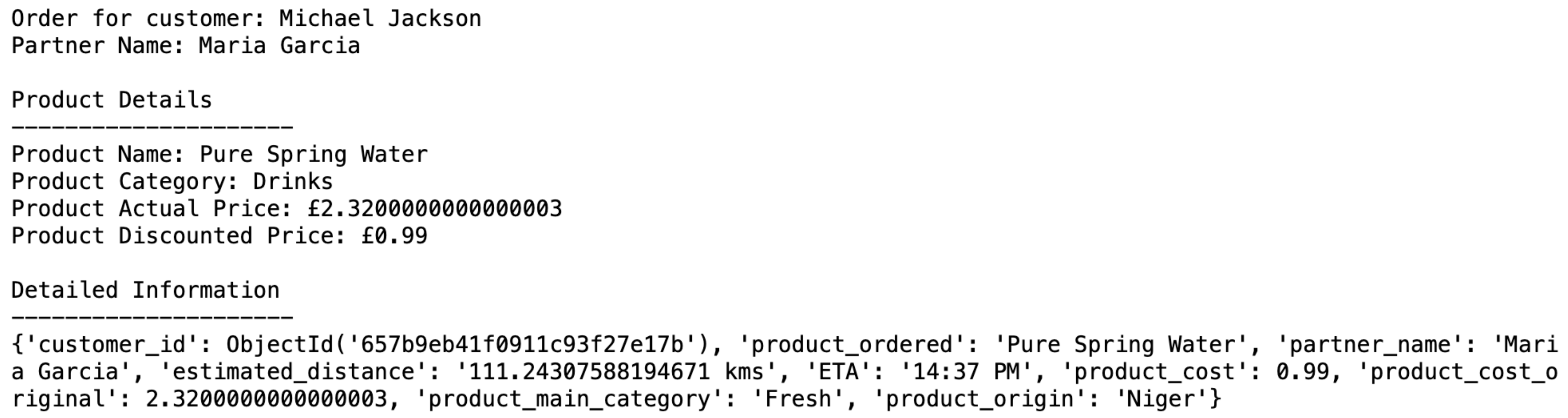
QUERIES

- Sample Queries implemented
- Query Results

Query: 1(a)

Customer ordering a product, adding it to the cart and making payment.

- For implementing the above query, python is used with the pymongo library.
- **Efficient Partner Allocation:** Assigns delivery partners based on location and ratings, ensuring quick and quality service.
- **ETA with Geospatial Data:** Uses geospatial analysis for real-time delivery ETA, improving customer experience.

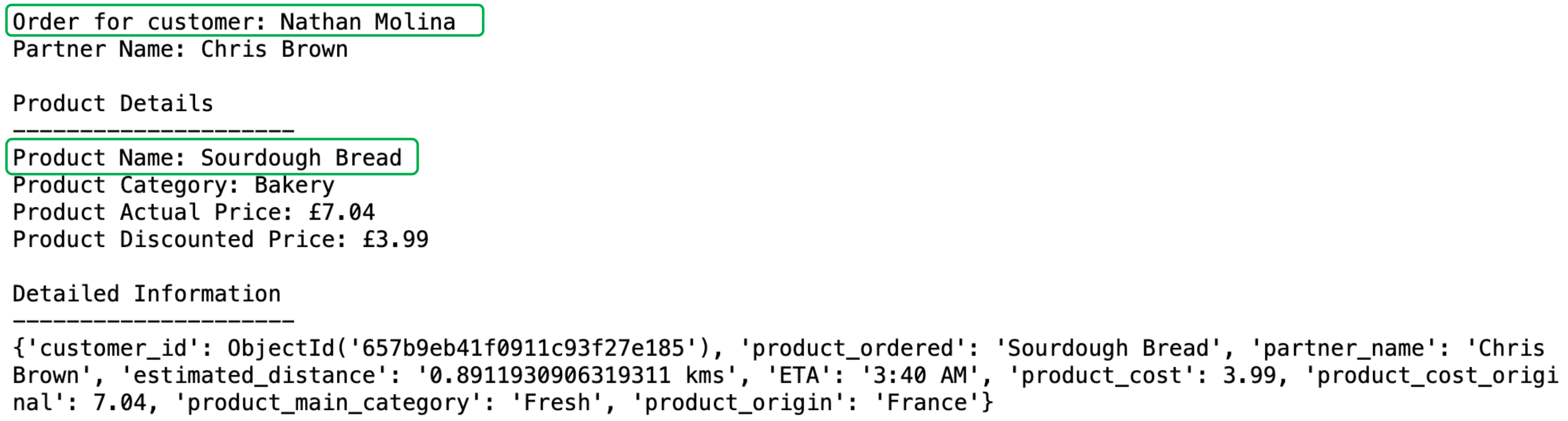


```
Order for customer: Michael Jackson
Partner Name: Maria Garcia

Product Details
---------------------
Product Name: Pure Spring Water
Product Category: Drinks
Product Actual Price: £2.3200000000000003
Product Discounted Price: £0.99

Detailed Information
---------------------
{'customer_id': ObjectId('657b9eb41f0911c93f27e17b'), 'product_ordered': 'Pure Spring Water', 'partner_name': 'Mari
a Garcia', 'estimated_distance': '111.24307588194671 kms', 'ETA': '14:37 PM', 'product_cost': 0.99, 'product_cost_o
riginal': 2.3200000000000003, 'product_main_category': 'Fresh', 'product_origin': 'Niger'}
```

Query: 1(b)

Customer ordering a product, adding it to the cart and making payment.

- For implementing the above query, python is used with the pymongo library.
- **Optimized Fresh Product Delivery System:** The query selects a customer and fresh product randomly and assigns the closest active delivery partner for timely and satisfactory service.
- **ETA with Geospatial Data:** Uses the same geospatial analysis based on the haversine formula for real-time delivery ETA, improving customer experience.

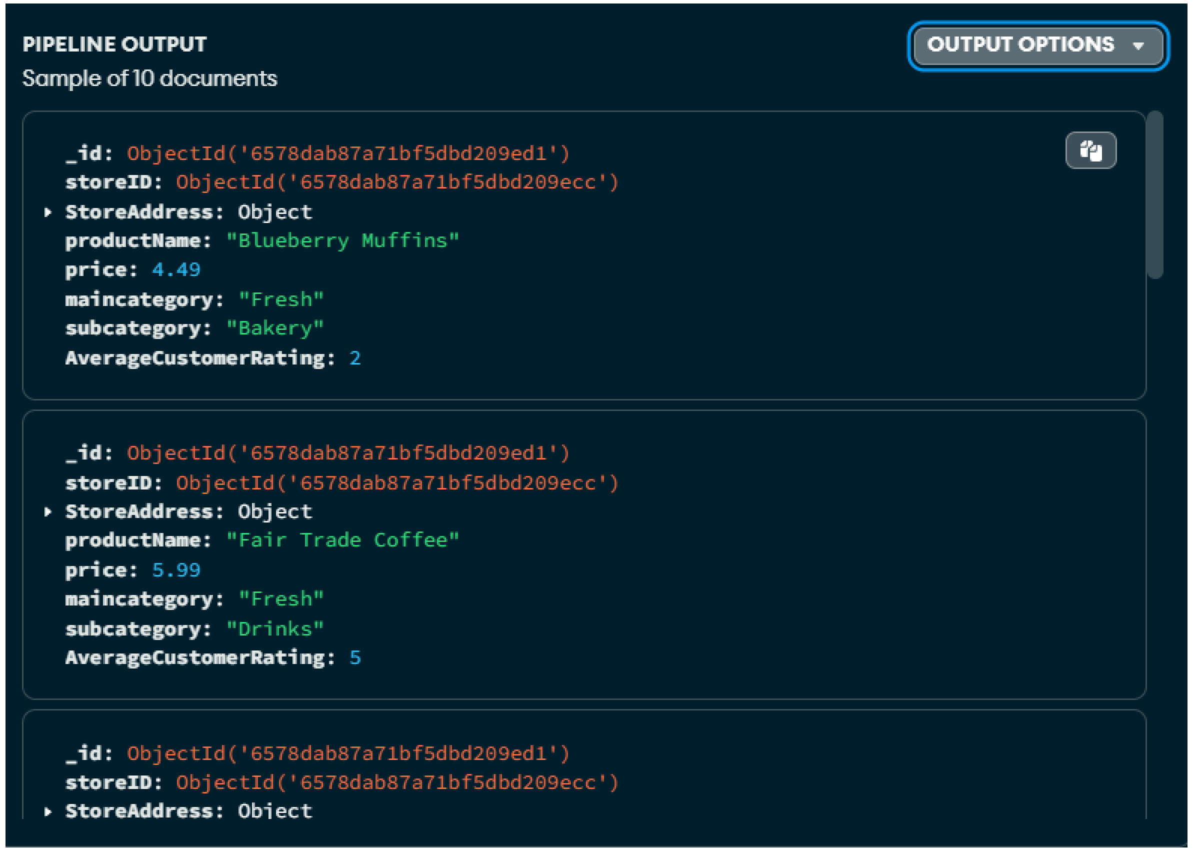


```
Order for customer: Nathan Molina
Partner Name: Chris Brown

Product Details
---------------------
Product Name: Sourdough Bread
Product Category: Bakery
Product Actual Price: £7.04
Product Discounted Price: £3.99

Detailed Information
---------------------
{'customer_id': ObjectId('657b9eb41f0911c93f27e185'), 'product_ordered': 'Sourdough Bread', 'partner_name': 'Chris
Brown', 'estimated_distance': '0.8911930906319311 kms', 'ETA': '3:40 AM', 'product_cost': 3.99, 'product_cost_origi
nal': 7.04, 'product_main_category': 'Fresh', 'product_origin': 'France'}
```

Query: 2

User searching for available fresh products. The products are displayed based on the user's location.

- The query was designed to locate grocery stores near a specified geospatial location.
- Type of Products we are looking for is only **"Fresh"** category.
- Using **aggregation pipeline** the information was extracted about fresh products in grocery stores based on the user's location and the nearest store.
- Stages used:-
  $geonear
  $unwind
  $lookup
  $match
  $project



Sample of 10 documents

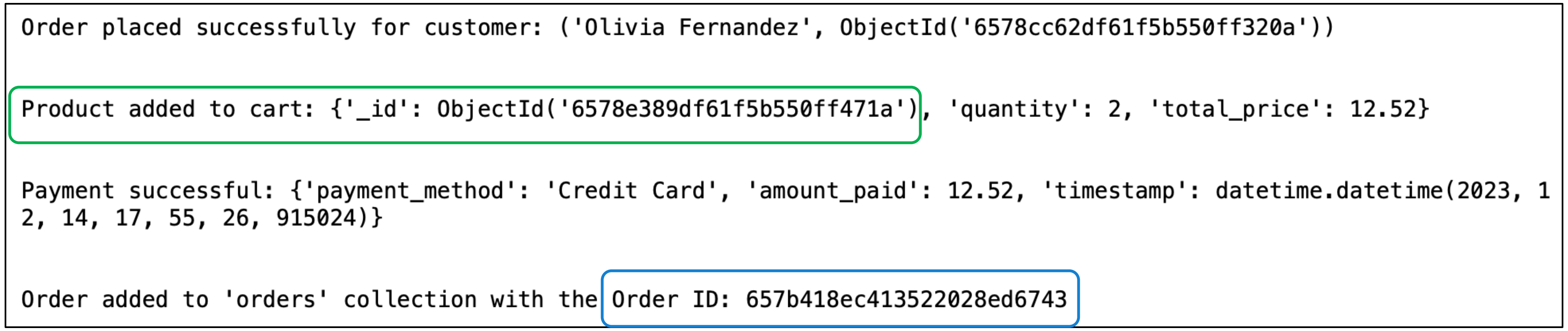
```
_id: ObjectId('6578dab87a71bf5dbd209ed1')
storeID: ObjectId('6578dab87a71bf5dbd209ecc')
▸ StoreAddress: Object
productName: "Blueberry Muffins"
price: 4.49
maincategory: "Fresh"
subcategory: "Bakery"
AverageCustomerRating: 2

_id: ObjectId('6578dab87a71bf5dbd209ed1')
storeID: ObjectId('6578dab87a71bf5dbd209ecc')
▸ StoreAddress: Object
productName: "Fair Trade Coffee"
price: 5.99
maincategory: "Fresh"
subcategory: "Drinks"
AverageCustomerRating: 5

_id: ObjectId('6578dab87a71bf5dbd209ed1')
storeID: ObjectId('6578dab87a71bf5dbd209ecc')
▸ StoreAddress: Object
```

Query: 3(a)

Customer ordering a product, adding it to the cart and making payment.

- For implementing the above query, python is used with the pymongo library.

- 'customers', 'products', and 'orders' collections are used to define the function simulating a customer ordering a product, addition of products to the cart and making the payment.

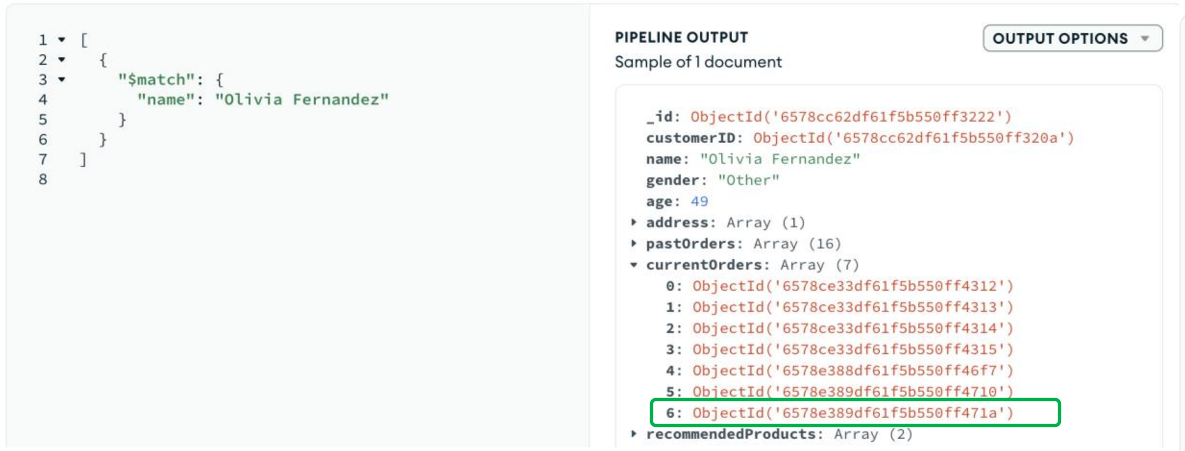


```
Order placed successfully for customer: ('Olivia Fernandez', ObjectId('6578cc62df61f5b550ff320a'))

Product added to cart: {'_id': ObjectId('6578e389df61f5b550ff471a'), 'quantity': 2, 'total_price': 12.52}

Payment successful: {'payment_method': 'Credit Card', 'amount_paid': 12.52, 'timestamp': datetime.datetime(2023, 1
2, 14, 17, 55, 26, 915024)}

Order added to 'orders' collection with the Order ID: 657b418ec413522028ed6743
```

Query: 3(a)

Customer ordering a product, adding it to the cart and making payment.

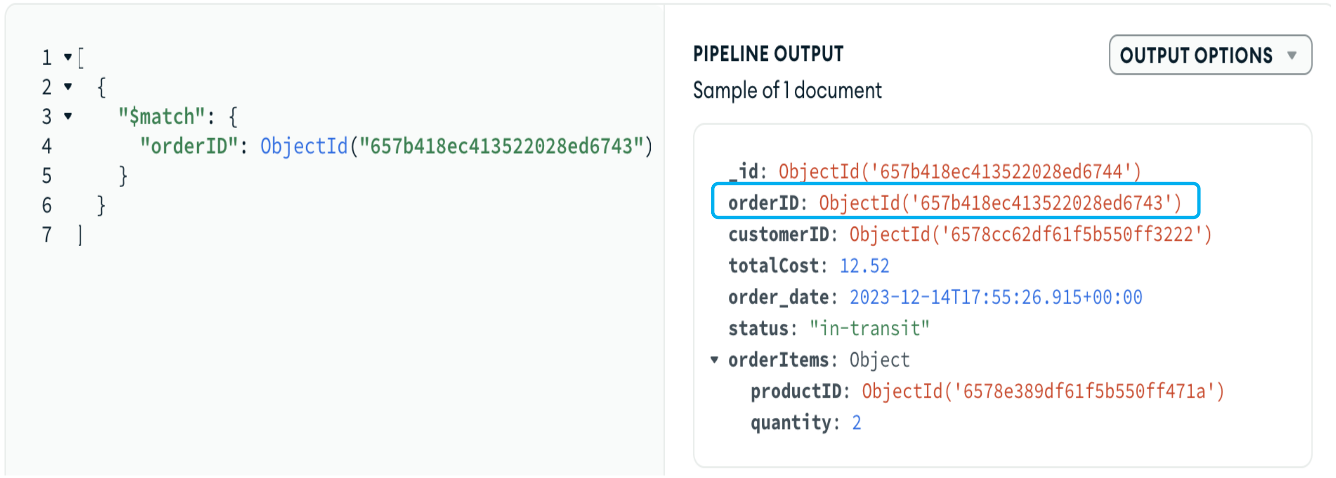
- After the payment is done, the customers collection is updated where the order details are updated in the currentOrder field with the productID of the ordered item.
- The Orders collection is also updated with the addition of a new document with the order details.



```
[
  {
    "$match": {
      "name": "Olivia Fernandez"
    }
  }
]
```

PIPELINE OUTPUT
Sample of 1 document

```
  _id: ObjectId('6578cc62df61f5b550ff3222')
  customerID: ObjectId('6578cc62df61f5b550ff320a')
  name: "Olivia Fernandez"
  gender: "Other"
  age: 49
▸ address: Array (1)
▸ pastOrders: Array (16)
▾ currentOrders: Array (7)
    0: ObjectId('6578ce33df61f5b550ff4312')
    1: ObjectId('6578ce33df61f5b550ff4313')
    2: ObjectId('6578ce33df61f5b550ff4314')
    3: ObjectId('6578ce33df61f5b550ff4315')
    4: ObjectId('6578e388df61f5b550ff46f7')
    5: ObjectId('6578e389df61f5b550ff4710')
    6: ObjectId('6578e389df61f5b550ff471a')
▸ recommendedProducts: Array (2)
```



```
[
  {
    "$match": {
      "orderID": ObjectId("657b418ec413522028ed6743")
    }
  }
]
```

PIPELINE OUTPUT
Sample of 1 document

```
  _id: ObjectId('657b418ec413522028ed6744')
  orderID: ObjectId('657b418ec413522028ed6743')
  customerID: ObjectId('6578cc62df61f5b550ff3222')
  totalCost: 12.52
  order_date: 2023-12-14T17:55:26.915+00:00
  status: "in-transit"
▾ orderItems: Object
    productID: ObjectId('6578e389df61f5b550ff471a')
    quantity: 2
```

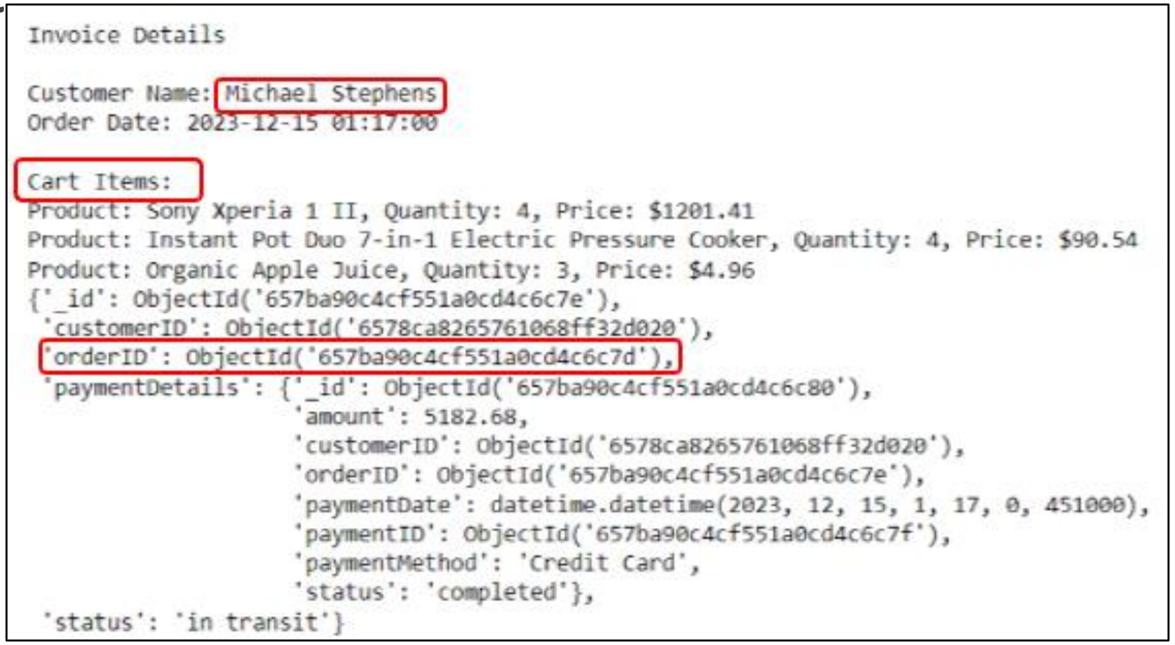
Query: 3(b)

A Customer orders multiple products, adding them to the cart, and making payment.

For implementing the query , we integrated python's pymongo library with MongoDB compass and developed a code utilising the aggregation pipeline.

'Customers, 'Products', 'Order's and collections were used to define the logic to add items to the customer specific **shopping cart**

Python code output:

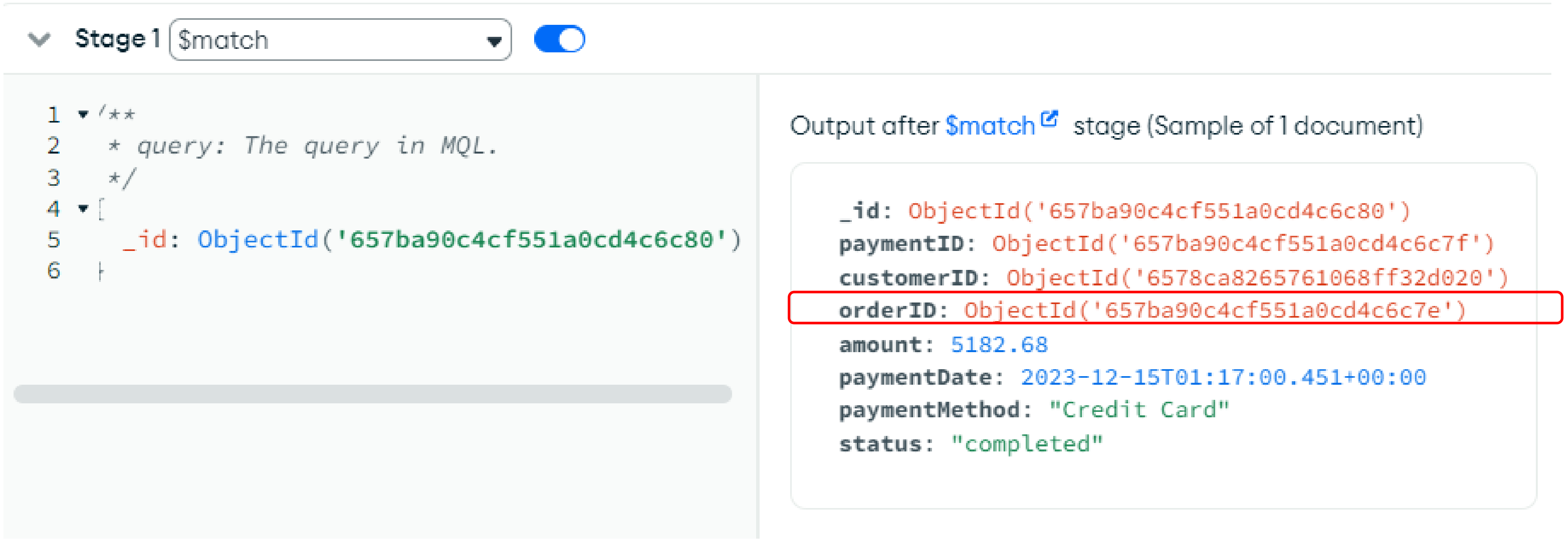


```
Invoice Details

Customer Name: Michael Stephens
Order Date: 2023-12-15 01:17:00

Cart Items:
Product: Sony Xperia 1 II, Quantity: 4, Price: $1201.41
Product: Instant Pot Duo 7-in-1 Electric Pressure Cooker, Quantity: 4, Price: $90.54
Product: Organic Apple Juice, Quantity: 3, Price: $4.96
{'_id': ObjectId('657ba90c4cf551a0cd4c6c7e'),
 'customerID': ObjectId('6578ca8265761068ff32d020'),
 'orderID': ObjectId('657ba90c4cf551a0cd4c6c7d'),
 'paymentDetails': {'_id': ObjectId('657ba90c4cf551a0cd4c6c80'),
                    'amount': 5182.68,
                    'customerID': ObjectId('6578ca8265761068ff32d020'),
                    'orderID': ObjectId('657ba90c4cf551a0cd4c6c7e'),
                    'paymentDate': datetime.datetime(2023, 12, 15, 1, 17, 0, 451000),
                    'paymentID': ObjectId('657ba90c4cf551a0cd4c6c7f'),
                    'paymentMethod': 'Credit Card',
                    'status': 'completed'},
 'status': 'in transit'}
```

Query: 3(b)

A Customer orders multiple products, adding them to the cart, and making payment.

'Payments' collection was **updated** with the payment details when a customer checks out the shopping cart items.



```
/**
 * query: The query in MQL.
 */
{
  _id: ObjectId('657ba90c4cf551a0cd4c6c80')
}
```

```
_id: ObjectId('657ba90c4cf551a0cd4c6c80')
paymentID: ObjectId('657ba90c4cf551a0cd4c6c7f')
customerID: ObjectId('6578ca8265761068ff32d020')
orderID: ObjectId('657ba90c4cf551a0cd4c6c7e')
amount: 5182.68
paymentDate: 2023-12-15T01:17:00.451+00:00
paymentMethod: "Credit Card"
status: "completed"
```

Query: 3(b)

A Customer orders <u>multiple products</u>, adding them to the cart, and making payment.

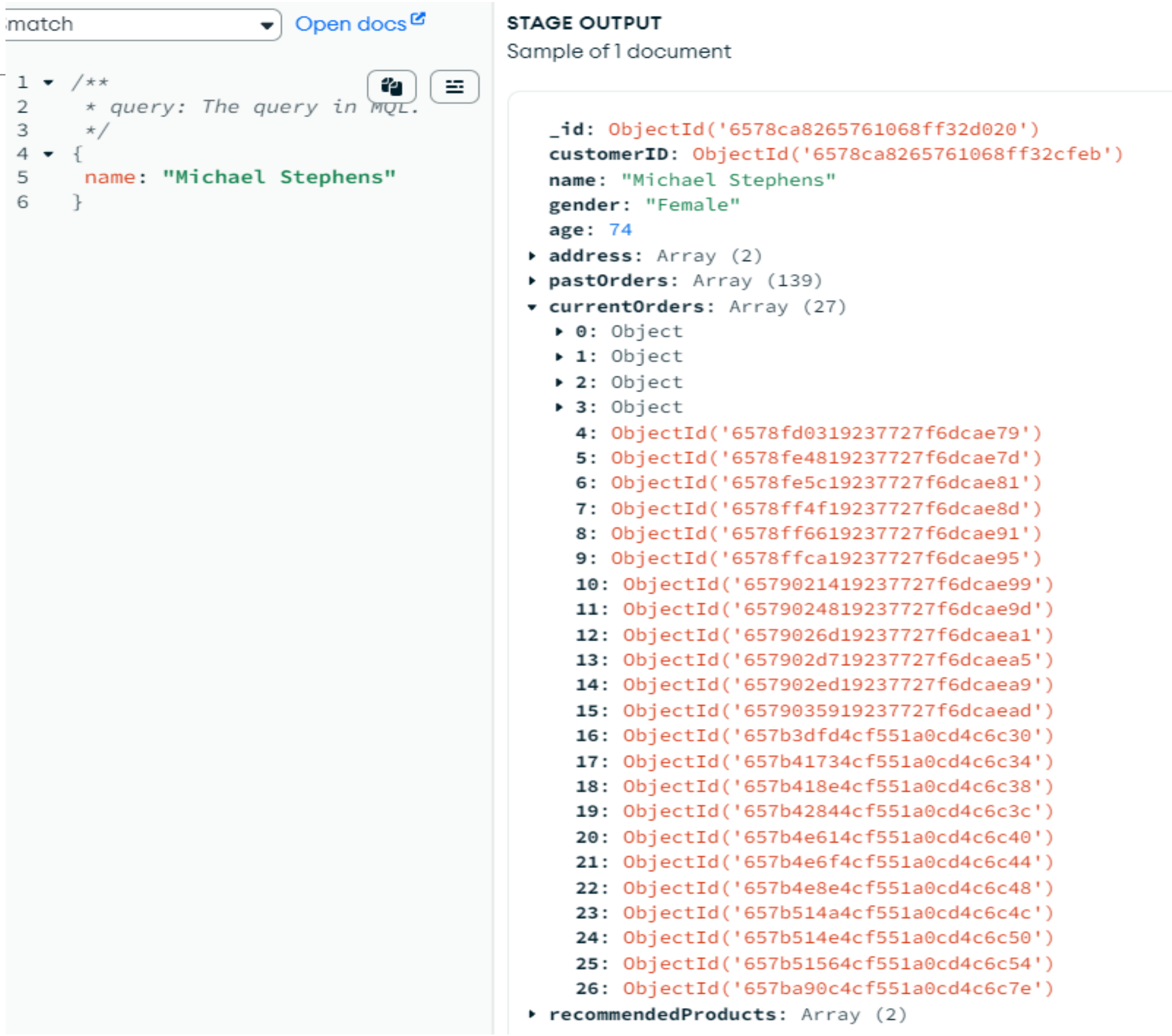
We used an aggregation pipeline to query the **<u>invoice details</u>** of the newly placed order , including <u>customer details, order details, and payment details.</u>

'Orders' collection gets the updated order details for the newly added current order having in-transit status.

'Customer' collection has a feature 'currentOrders' which also gets updated as soon as the order is placed.

<u>Stages used:</u>

$match (to match orderID) $lookup (to join associated payment details) $project (to output the details.)

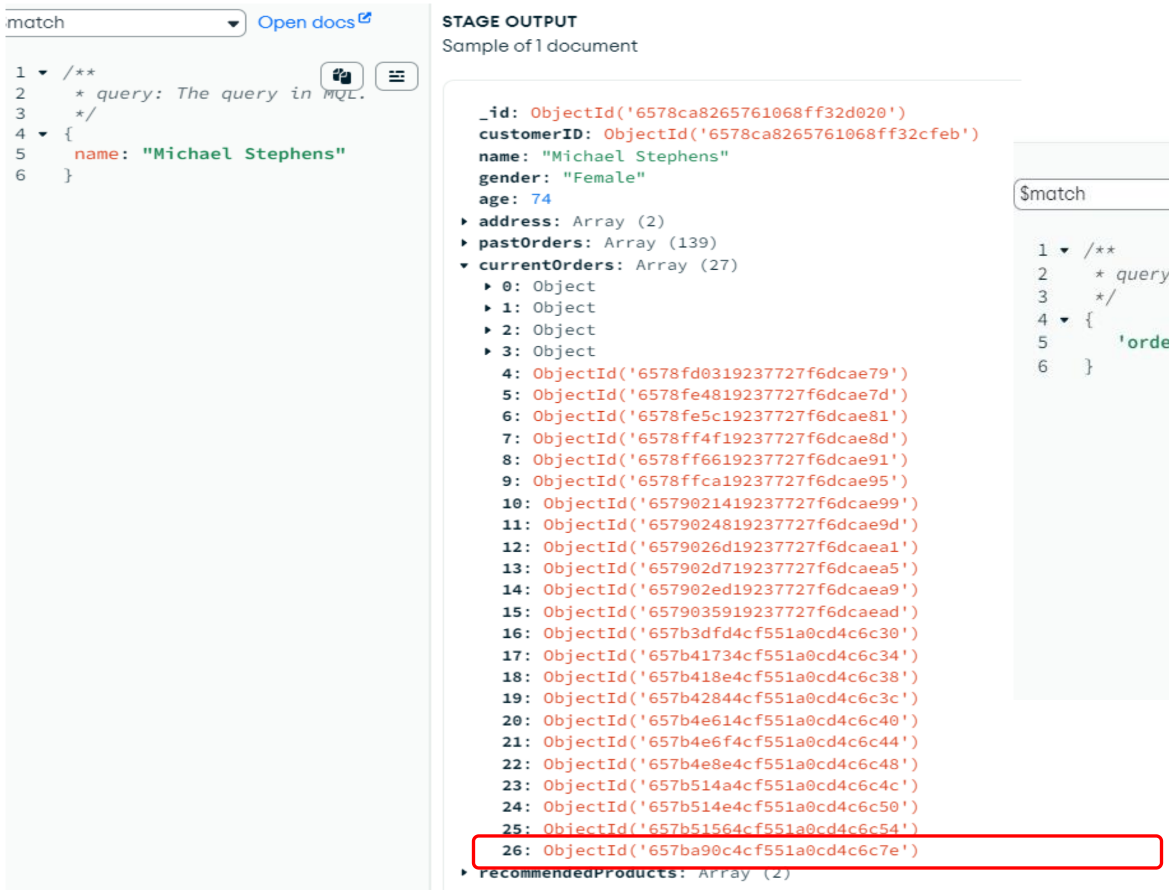


```
/**
 * query: The query in MQL.
 */
{
  name: "Michael Stephens"
}
```

```
_id: ObjectId('6578ca8265761068ff32d020')
customerID: ObjectId('6578ca8265761068ff32cfeb')
name: "Michael Stephens"
gender: "Female"
age: 74
▸ address: Array (2)
▸ pastOrders: Array (139)
▾ currentOrders: Array (27)
  ▸ 0: Object
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
    4: ObjectId('6578fd0319237727f6dcae79')
    5: ObjectId('6578fe4819237727f6dcae7d')
    6: ObjectId('6578fe5c19237727f6dcae81')
    7: ObjectId('6578ff4f19237727f6dcae8d')
    8: ObjectId('6578ff6619237727f6dcae91')
    9: ObjectId('6578ffca19237727f6dcae95')
    10: ObjectId('6579021419237727f6dcae99')
    11: ObjectId('6579024819237727f6dcae9d')
    12: ObjectId('6579026d19237727f6dcaea1')
    13: ObjectId('657902d719237727f6dcaea5')
    14: ObjectId('657902ed19237727f6dcaea9')
    15: ObjectId('6579035919237727f6dcaead')
    16: ObjectId('657b3dfd4cf551a0cd4c6c30')
    17: ObjectId('657b41734cf551a0cd4c6c34')
    18: ObjectId('657b418e4cf551a0cd4c6c38')
    19: ObjectId('657b42844cf551a0cd4c6c3c')
    20: ObjectId('657b4e614cf551a0cd4c6c40')
    21: ObjectId('657b4e6f4cf551a0cd4c6c44')
    22: ObjectId('657b4e8e4cf551a0cd4c6c48')
    23: ObjectId('657b514a4cf551a0cd4c6c4c')
    24: ObjectId('657b514e4cf551a0cd4c6c50')
    25: ObjectId('657b51564cf551a0cd4c6c54')
    26: ObjectId('657ba90c4cf551a0cd4c6c7e')
▸ recommendedProducts: Array (2)
```

Query: 3(b)

A Customer orders multiple products, adding them to the cart, and making payment.

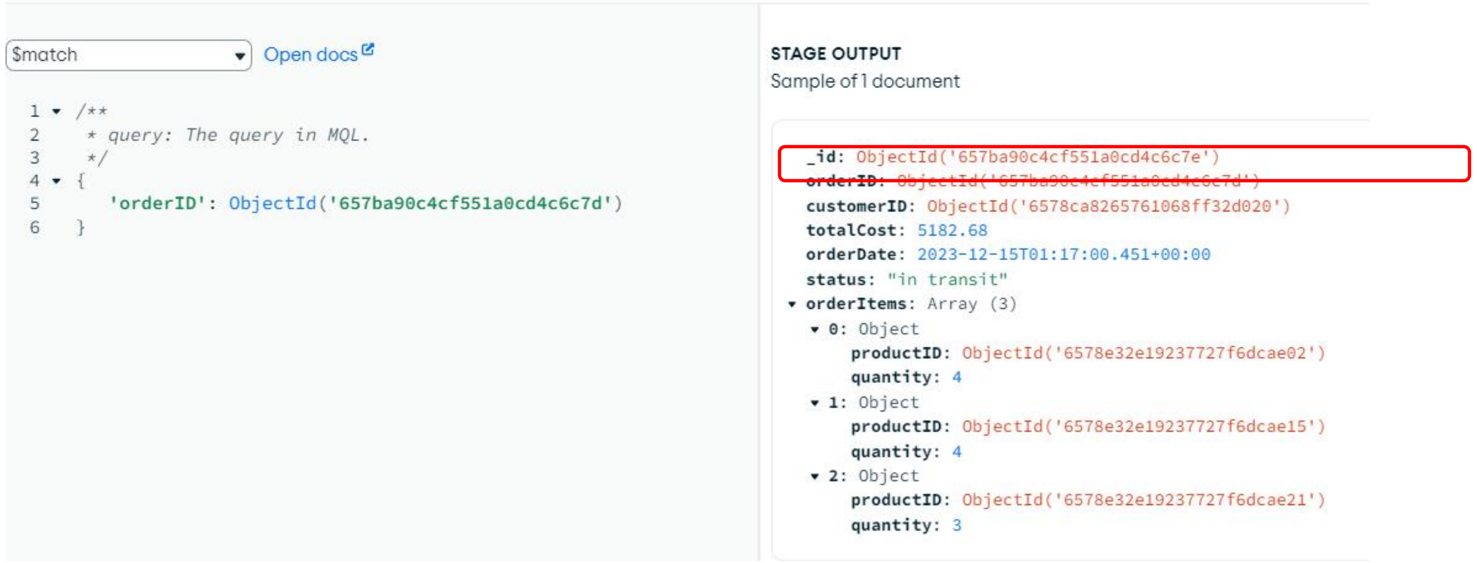


match   Open docs

```
/**
 * query: The query in MQL.
 */
{
  name: "Michael Stephens"
}
```

STAGE OUTPUT
Sample of 1 document

```
_id: ObjectId('6578ca8265761068ff32d020')
customerID: ObjectId('6578ca8265761068ff32cfeb')
name: "Michael Stephens"
gender: "Female"
age: 74
▸ address: Array (2)
▸ pastOrders: Array (139)
▾ currentOrders: Array (27)
  ▸ 0: Object
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
    4: ObjectId('6578fd0319237727f6dcae79')
    5: ObjectId('6578fe4819237727f6dcae7d')
    6: ObjectId('6578fe5c19237727f6dcae81')
    7: ObjectId('6578ff4f19237727f6dcae8d')
    8: ObjectId('6578ff6619237727f6dcae91')
    9: ObjectId('6578ffca19237727f6dcae95')
    10: ObjectId('6579021419237727f6dcae99')
    11: ObjectId('6579024819237727f6dcae9d')
    12: ObjectId('6579026d19237727f6dcaea1')
    13: ObjectId('657902d719237727f6dcaea5')
    14: ObjectId('657902ed19237727f6dcaea9')
    15: ObjectId('6579035919237727f6dcaead')
    16: ObjectId('657b3dfd4cf551a0cd4c6c30')
    17: ObjectId('657b41734cf551a0cd4c6c34')
    18: ObjectId('657b418e4cf551a0cd4c6c38')
    19: ObjectId('657b42844cf551a0cd4c6c3c')
    20: ObjectId('657b4e614cf551a0cd4c6c40')
    21: ObjectId('657b4e6f4cf551a0cd4c6c44')
    22: ObjectId('657b4e8e4cf551a0cd4c6c48')
    23: ObjectId('657b514a4cf551a0cd4c6c4c')
    24: ObjectId('657b514e4cf551a0cd4c6c50')
    25: ObjectId('657b51564cf551a0cd4c6c54')
    26: ObjectId('657ba90c4cf551a0cd4c6c7e')
▸ recommendedProducts: Array (2)
```



$match   Open docs

```
/**
 * query: The query in MQL.
 */
{
  'orderID': ObjectId('657ba90c4cf551a0cd4c6c7d')
}
```

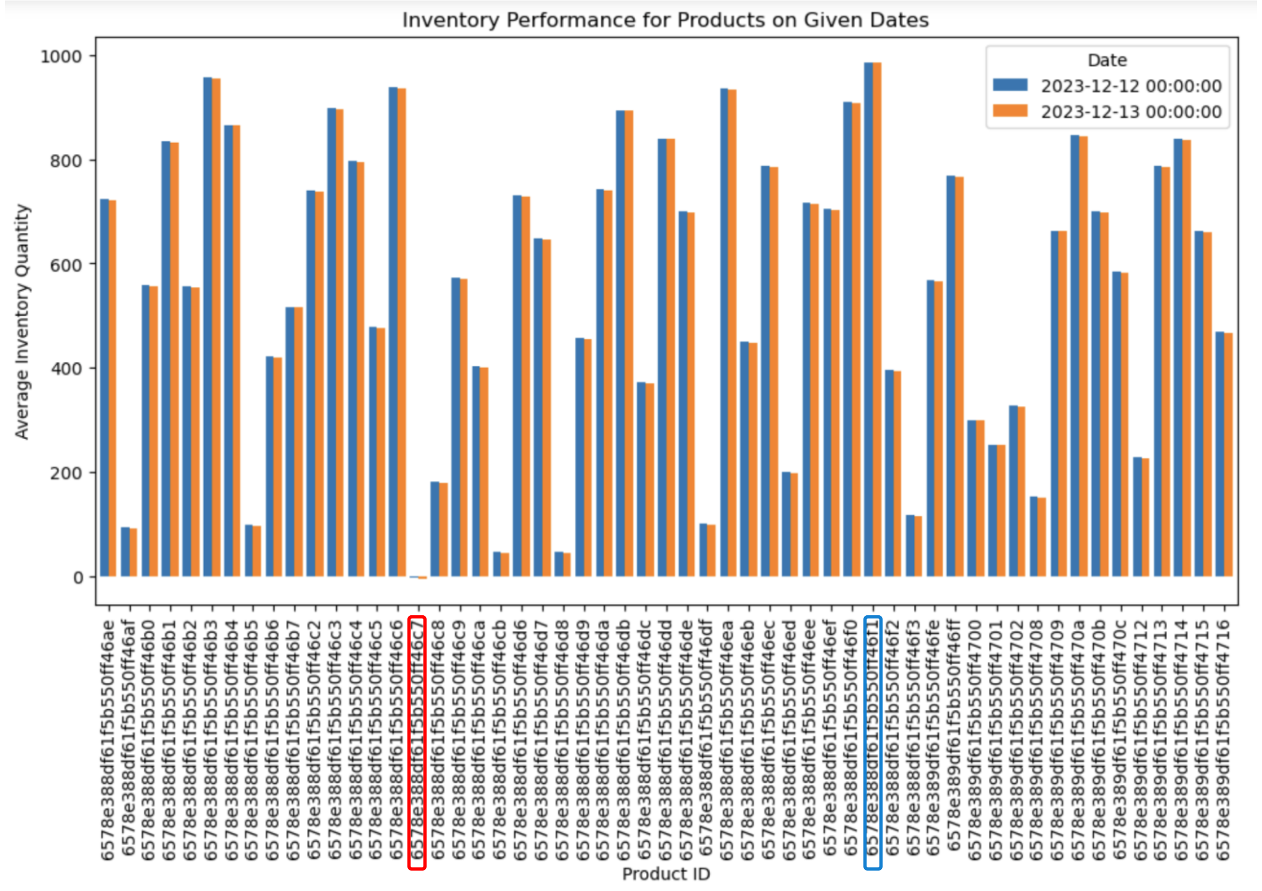
STAGE OUTPUT
Sample of 1 document

```
_id: ObjectId('657ba90c4cf551a0cd4c6c7e')
orderID: ObjectId('657ba90c4cf551a0cd4c6c7d')
customerID: ObjectId('6578ca8265761068ff32d020')
totalCost: 5182.68
orderDate: 2023-12-15T01:17:00.451+00:00
status: "in transit"
▾ orderItems: Array (3)
  ▾ 0: Object
      productID: ObjectId('6578e32e19237727f6dcae02')
      quantity: 4
  ▾ 1: Object
      productID: ObjectId('6578e32e19237727f6dcae15')
      quantity: 4
  ▾ 2: Object
      productID: ObjectId('6578e32e19237727f6dcae21')
      quantity: 3
```

Query: 4(a)

Manager checking inventory performance and visualising using Pandas Matplotlib

- For implementing the above query, python is used with the pymongo library along with the aggregation pipeline “unwind”, “group” and “sort” commands.
- The inventory_query is created where we have used the “unwind” on dailyInventoryLevels field to separate out all the inventory attributes.
- ”group” is used to group the results based on the productt_id and date to calculate the average inventory over a period of two days.
- To visualise the above query, we have converted the result to a Pandas DataFrame and then Pivot the DataFrame for plotting a grouped bar chart.

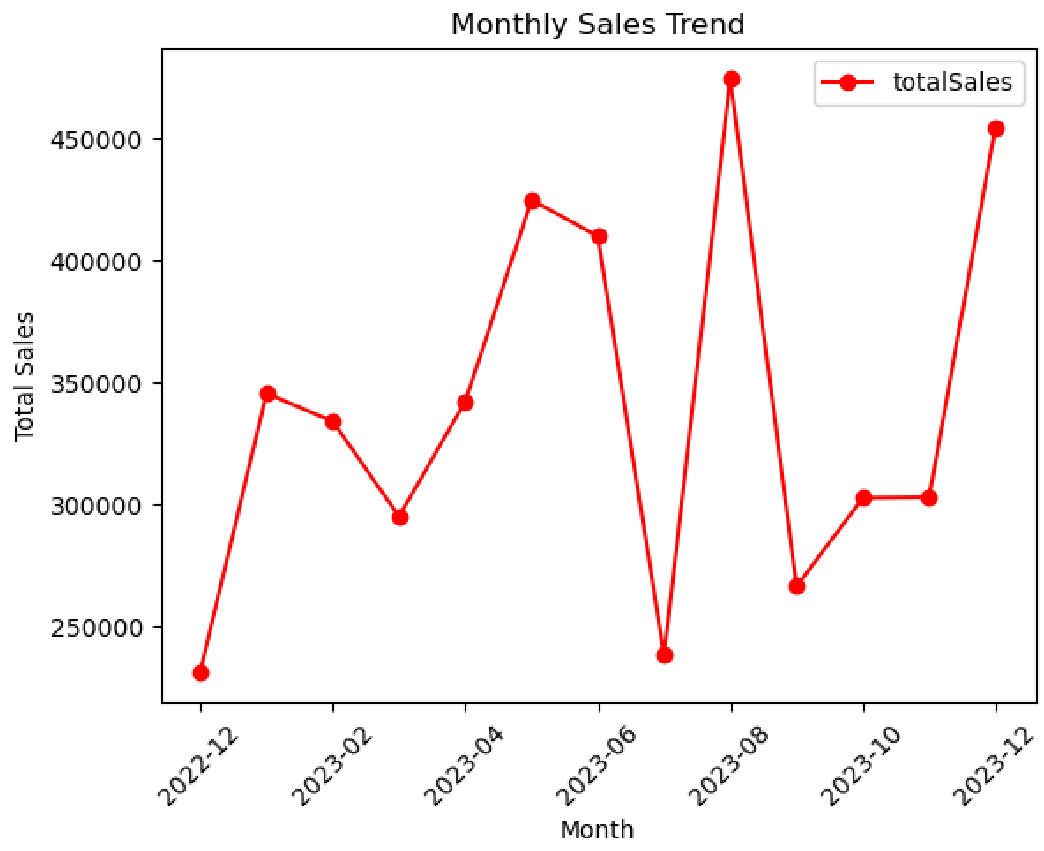


Query: 4(b)

Manager checking sales performance and visualising using Pandas Matplotlib

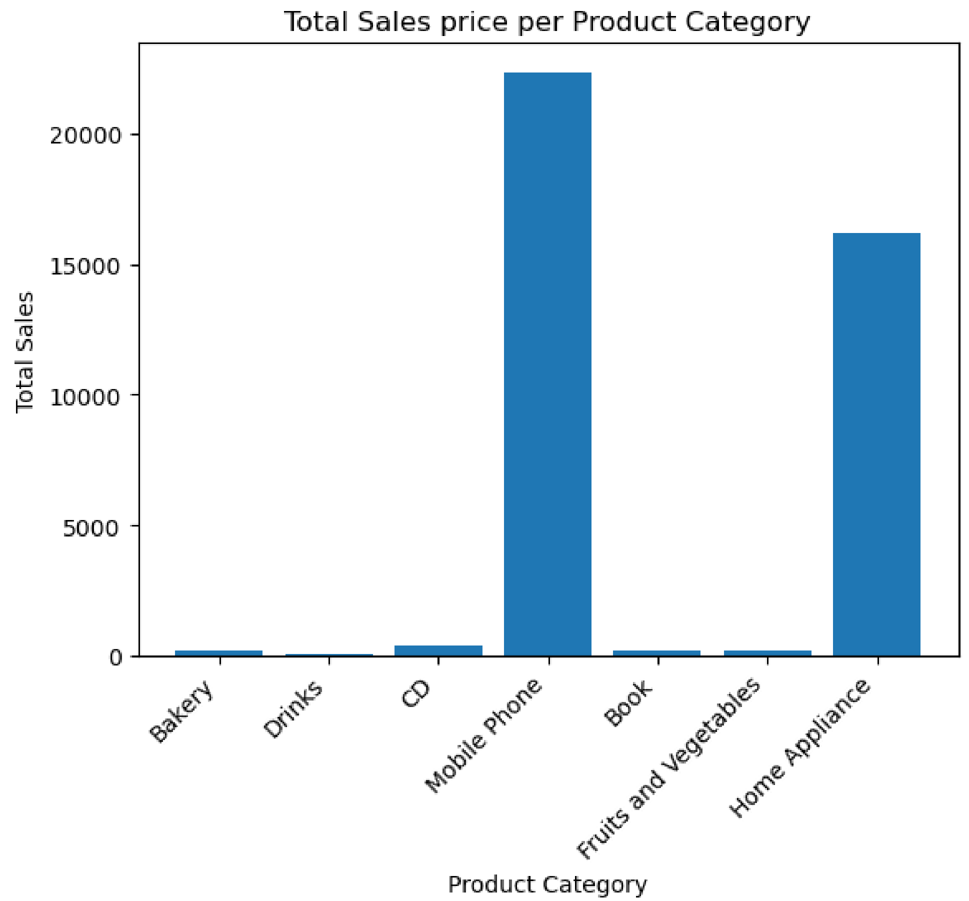
- The query was designed to measure the monthly sales performance for Amazone's products.
- The monthly sales prices are evaluated based on "**past orders**" and **"current order"** order price.
- Using **aggregation pipeline** to give the monthly sales value and visualised using pandas and matplotlib.
- Stages used:-
  $group
  $sum
  $project
  $sort

```
    totalSales    month
0    231293.46  2022-12
1    345423.90  2023-01
2    333987.84  2023-02
3    294973.33  2023-03
4    342217.92  2023-04
5    424690.56  2023-05
6    409793.32  2023-06
7    238198.72  2023-07
8    474558.78  2023-08
9    266391.45  2023-09
10   302614.74  2023-10
11   302892.71  2023-11
12   454542.10  2023-12
```



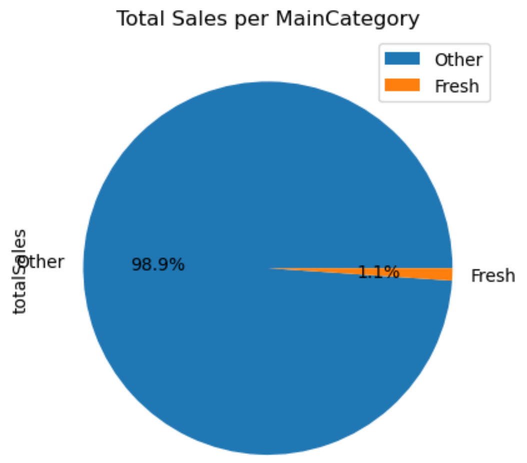
Query: 4(b)

Manager checking sales performance and visualising using Pandas Matplotlib



```
Total Sales per Product Category:

                     _id  totalSales

0                 Bakery      155.24

1                     CD      359.93

2         Home Appliance    16171.45

3  Fruits and Vegetables      177.34

4                   Book      192.64

5           Mobile Phone    22346.18

6                 Drinks       88.94
```

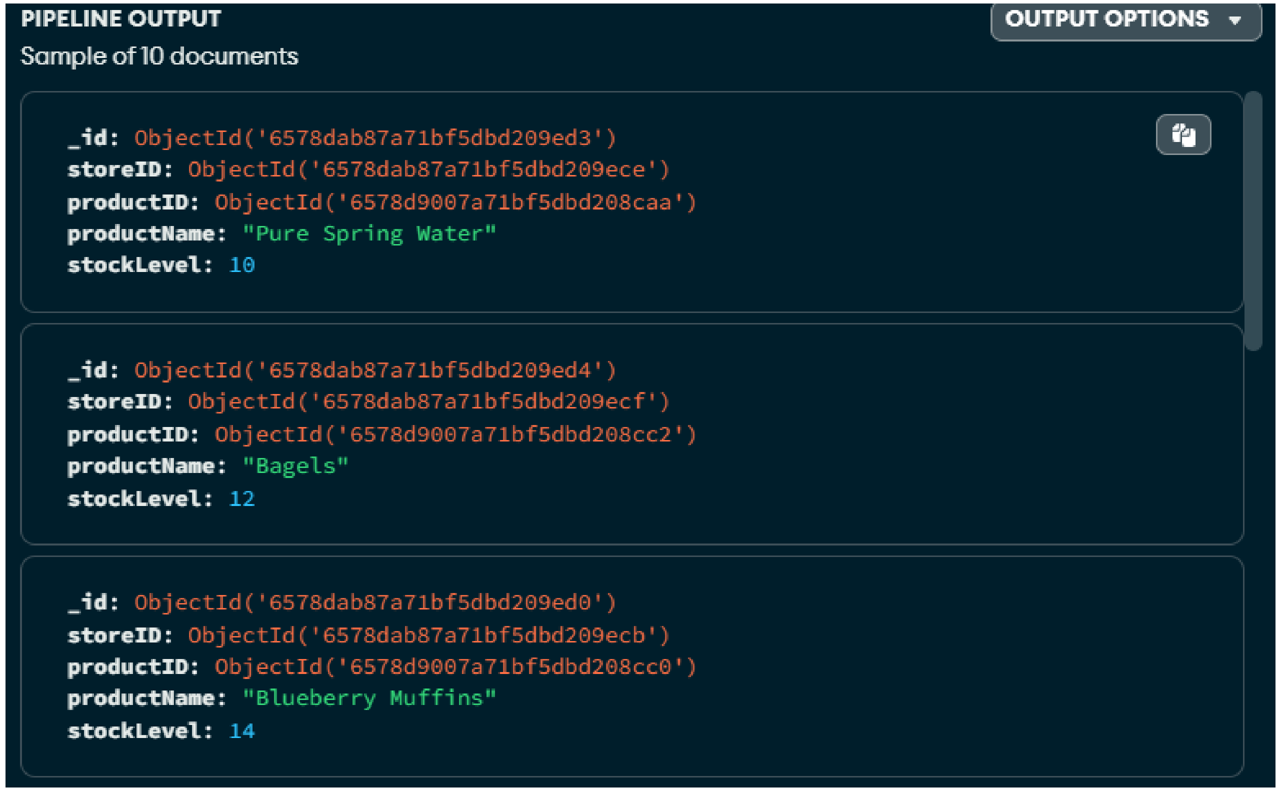
```
Total Sales per MainCategory:
     _id  totalSales
0  Other    39070.20
1  Fresh      421.52
```



Query: 5(a)

Filtering Morrizons stores with low stock level products (<100 items)

- This scenario was important to **identify and manage low stock level grocery items.** After identifying the stores can restock from the respective warehouses.
- The presented output is achieved using MongoDB **aggregation pipeline.**
- Grocery items with a stock level **below or equal to 100** from the "availableGroceryItems" are obtained.
- Collection used: **Stores and products**
- Stages used:
  - $unwind
  - $lookup
  - $project
  - $sort
  - $limit



Sample of 10 documents

```
_id: ObjectId('6578dab87a71bf5dbd209ed3')
storeID: ObjectId('6578dab87a71bf5dbd209ece')
productID: ObjectId('6578d9007a71bf5dbd208caa')
productName: "Pure Spring Water"
stockLevel: 10
```

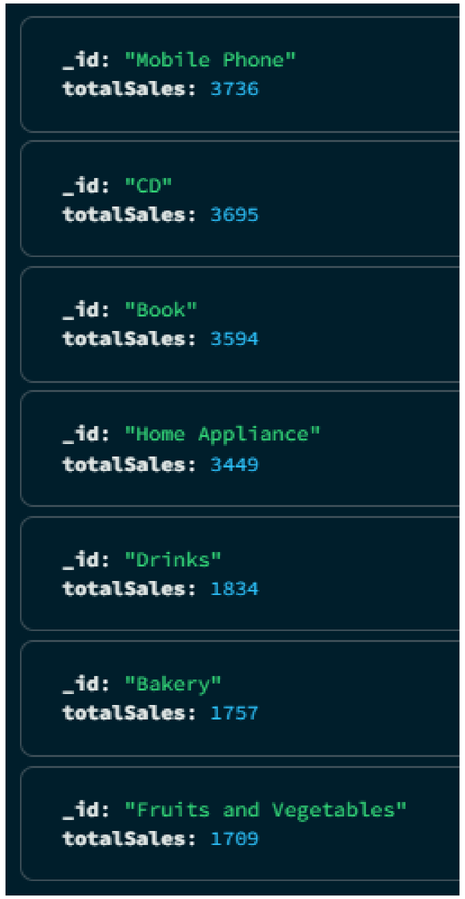
```
_id: ObjectId('6578dab87a71bf5dbd209ed4')
storeID: ObjectId('6578dab87a71bf5dbd209ecf')
productID: ObjectId('6578d9007a71bf5dbd208cc2')
productName: "Bagels"
stockLevel: 12
```

```
_id: ObjectId('6578dab87a71bf5dbd209ed0')
storeID: ObjectId('6578dab87a71bf5dbd209ecb')
productID: ObjectId('6578d9007a71bf5dbd208cc0')
productName: "Blueberry Muffins"
stockLevel: 14
```

Query: 5(b)

Finding Number of Sales per Category (Quantity sold)

- By calculating and sorting the total sales for each product category, the organization gains valuable insights into the performance of different product types and enabling the organization to meet customer demands more effectively.
- MongoDB **aggregation pipeline** is designed to calculate and display the total sales based on **"orderItems"** within the collection.
- Collection used: **Orders**
- Stages:
    - $unwind
    - $lookup
    - $group
    - $sort



```
_id: "Mobile Phone"
totalSales: 3736

_id: "CD"
totalSales: 3695

_id: "Book"
totalSales: 3594

_id: "Home Appliance"
totalSales: 3449

_id: "Drinks"
totalSales: 1834

_id: "Bakery"
totalSales: 1757

_id: "Fruits and Vegetables"
totalSales: 1709
```

Query: 5(c)

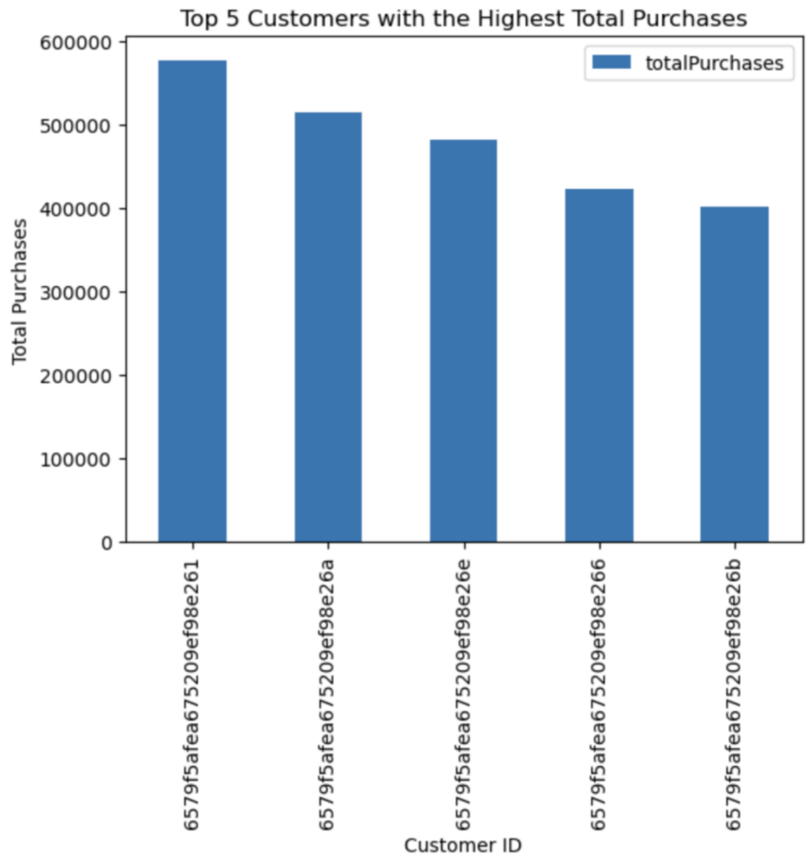
Top 5 customers with highest total purchases

- For implementing the above query, python is used with the pymongo library “limit” "group"and “sort” commands.

- top_customers_query is created when total purchases are sorted in the ascending order and the limit is set to 5, for obtaining the top 5 customers with highest total purchases.

- Moreover to visualise the actual numbers those 5 numbers were displayed on a bar chart.
- Collection used: **Orders**

- Stages:
  - Group
  - Sort
  - Limit



THANK YOU