MANCHESTER 1824
The University of Manchester

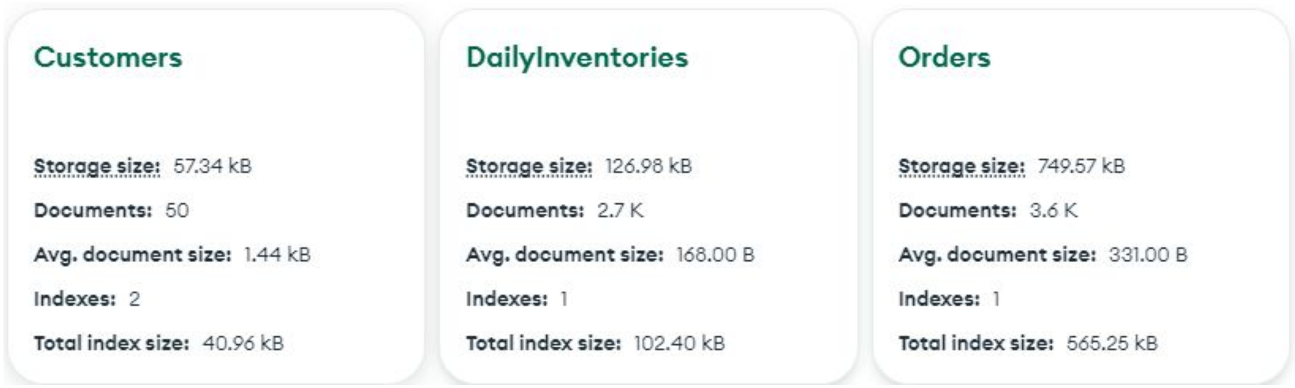
Amazone - NoSQL
Project

Bagus Pranata

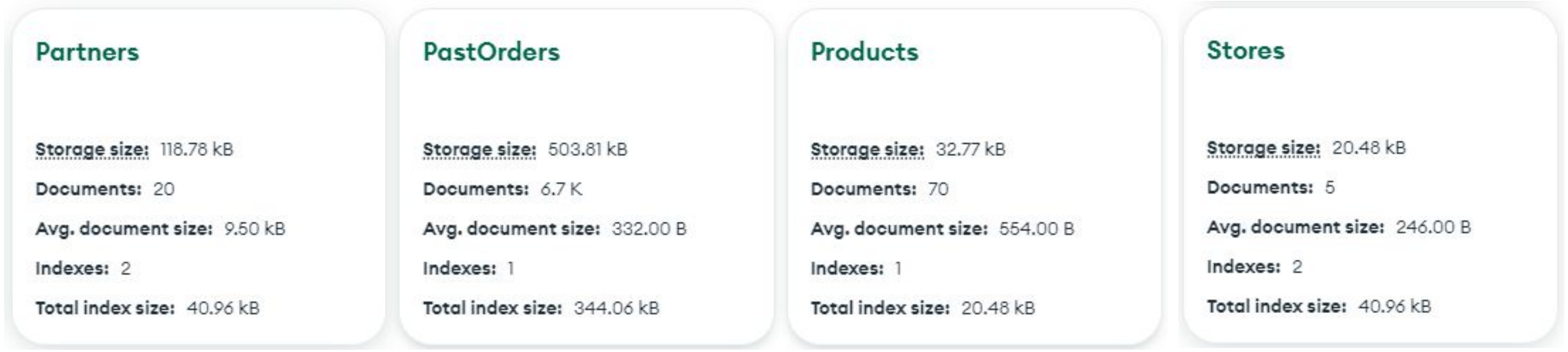
1

Schema Design

Schema Overview



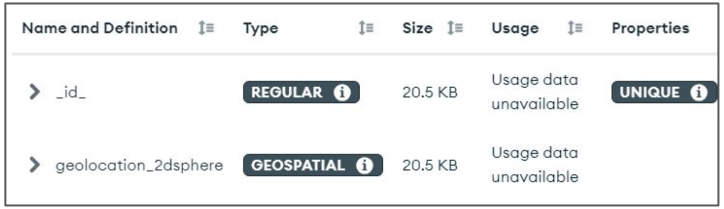
	Customers	DailyInventories	Orders
Storage size:	57.34 kB	126.98 kB	749.57 kB
Documents:	50	2.7 K	3.6 K
Avg. document size:	1.44 kB	168.00 B	331.00 B
Indexes:	2	1	1
Total index size:	40.96 kB	102.40 kB	565.25 kB



	Partners	PastOrders	Products	Stores
Storage size:	118.78 kB	503.81 kB	32.77 kB	20.48 kB
Documents:	20	6.7 K	70	5
Avg. document size:	9.50 kB	332.00 B	554.00 B	246.00 B
Indexes:	2	1	1	2
Total index size:	40.96 kB	344.06 kB	20.48 kB	40.96 kB

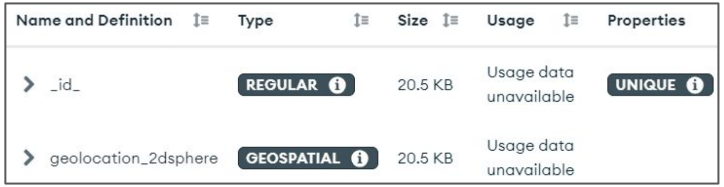
Schema Index

Customers



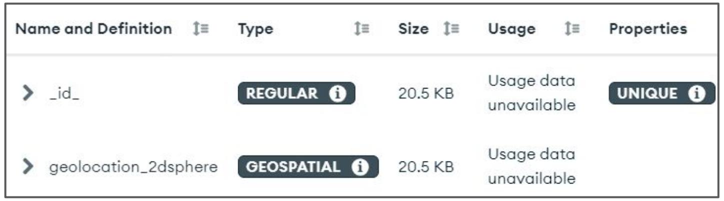
Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	20.5 KB	Usage data unavailable	UNIQUE
geolocation_2dsphere	GEOSPATIAL	20.5 KB	Usage data unavailable	

Partners



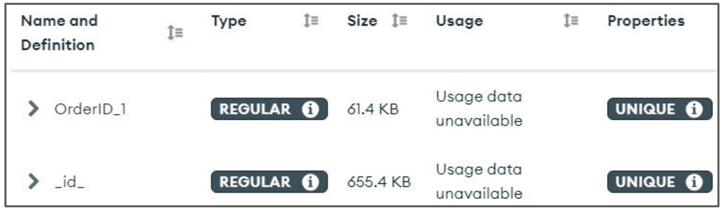
Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	20.5 KB	Usage data unavailable	UNIQUE
geolocation_2dsphere	GEOSPATIAL	20.5 KB	Usage data unavailable	

Stores



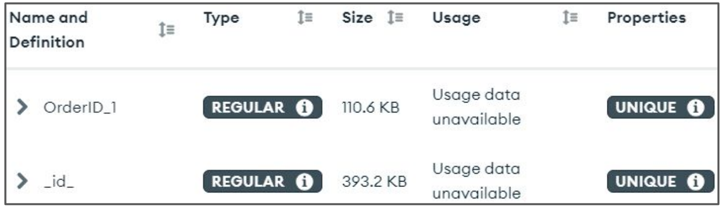
Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	20.5 KB	Usage data unavailable	UNIQUE
geolocation_2dsphere	GEOSPATIAL	20.5 KB	Usage data unavailable	

Orders



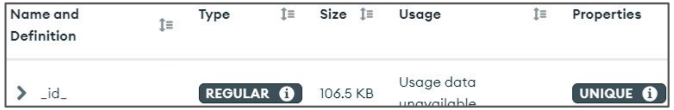
Name and Definition	Type	Size	Usage	Properties
OrderID_1	REGULAR	61.4 KB	Usage data unavailable	UNIQUE
_id_	REGULAR	655.4 KB	Usage data unavailable	UNIQUE

PastOrders



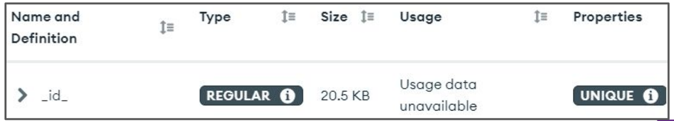
Name and Definition	Type	Size	Usage	Properties
OrderID_1	REGULAR	110.6 KB	Usage data unavailable	UNIQUE
_id_	REGULAR	393.2 KB	Usage data unavailable	UNIQUE

DailyInventories



Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	106.5 KB	Usage data unavailable	UNIQUE

Products



Name and Definition	Type	Size	Usage	Properties
_id_	REGULAR	20.5 KB	Usage data unavailable	UNIQUE

Customers and Products Collection Schema

```
Customers {
_id           : objectID
Name          : string,
Gender        : string,
Age           : integer,
Addresses  : [Array of addresses]
Geolocation: [Object]
                  { latitude: "double",
                   longitude: "double" },
Recommended_products: [Array]
                  ref<Products._id>],
Orders: [Array]
              orderID: ref<Orders._id>
}
```

Note

🟦 : Referencing

```
Products {
_id             : objectID,
Category     : string,
Name           : string,
short_Description: string,
storeID        : ref<Stores._id>,
Product_dimension: string
Product_weight_or_quantity: string,
Expiry_date: string,
Country_of_origin: string,
Standard_price_to_customers: integer,
Cost_of_products: integer,
Type            : string,
Ratings: [Array]
                _id     : objectID,
                customerID: ref<Customers._id>,
                Rating: integer,
                Review: string,
                Timestamp: Timestamp}
```

Note:
For '<u>Other</u>' product, they will have their own additional & specific product attributes

MANCHESTER 1824 The University of Manchester

Orders and PastOrders Collection Schema

Orders are any orders made in the same year (in this case 2023), older than that, it will go to **PastOrders** given the status is 'Paid' and 'Delivered' or 'Returned and 'Refunded' or 'Cancelled' and 'Unpaid'

Orders {

```
_id          : objectID,
OrderID      : string,
customerID: ref<Customers._id>,
Items        : [Array]
               productID   : ref<Products_id>,
               Quantity    : integer,
storeID      : objectID ref<Stores._id>,
partnerID    : ref<Partners._id>,
Order_date: Timestamp,
Order_status: string,
Payment_status: string
}
```

PastOrders {

```
_id          : objectID,
OrderID      : string,
customerID: ref<Customers._id>,
Items        : [Array]
               productID   : ref<Products_id>,
               Quantity    : integer,
storeID      : objectID ref<Stores._id>,
partnerID    : ref<Partners._id>,
Order_date: Timestamp,
Order_status: string,
Payment_status: string
}
```

Note

🟦 : Referencing

Stores, Partners, & Daily Inventories Collection Schema

```
Stores {
_id           : objectID,
Name          : string,
Geolocation: [Object]
                { latitude: "double",
                 longitude: "double" },
Address       : String,
Items         : [Array]
              ref<Products._id> }
```

```
Daily Inventories {
_id           : objectID,
productID     : ref<Products._id>,
Date          : date timestamp,
Inventory_quantity: integer,
Storeage_warehouse_name: string,
Geolocation: [Object]
                { latitude: "double", longitude: "double" },
Address       : string}
```

```
Partners {
_id           : objectID,
Name          : string,
Gender        : string,
Age           : integer,
Status        : string,
Geolocation: [Object]
                { latitude: "double",
               longitude: "double" }
 Addresses : [Array of address]
Rating        : integer,
Statistics    : [Object]
              TotalofWorkingHours: integer,
              NumberofDeliveries : integer,
Deliveries    : [Array]
              ref<Orders._id>
}
```

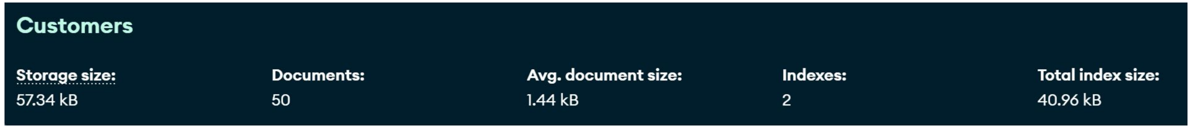
Note

🟦 : Referencing

2

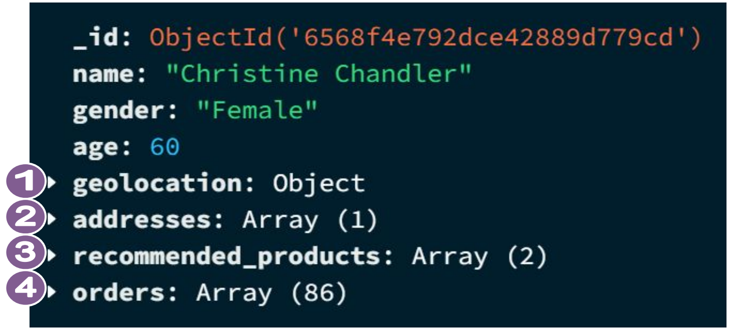
Sample Data Implemented

Customers Sample Data

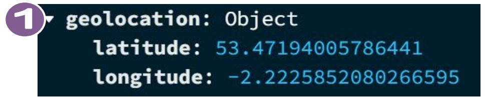


Customers

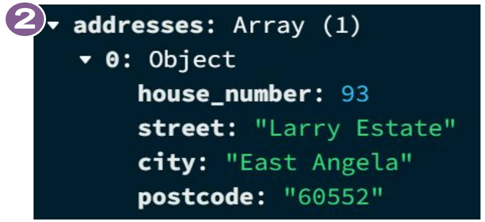
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
57.34 kB	50	1.44 kB	2	40.96 kB



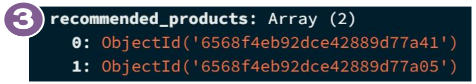
```
_id: ObjectId('6568f4e792dce42889d779cd')
name: "Christine Chandler"
gender: "Female"
age: 60
1 ▸ geolocation: Object
2 ▸ addresses: Array (1)
3 ▸ recommended_products: Array (2)
4 ▸ orders: Array (86)
```



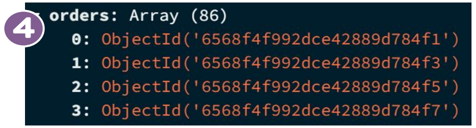
```
1 ▾ geolocation: Object
      latitude: 53.47194005786441
      longitude: -2.2225852080266595
```



```
2 ▾ addresses: Array (1)
    ▾ 0: Object
        house_number: 93
        street: "Larry Estate"
        city: "East Angela"
        postcode: "60552"
```



```
3 recommended_products: Array (2)
    0: ObjectId('6568f4eb92dce42889d77a41')
    1: ObjectId('6568f4eb92dce42889d77a05')
```

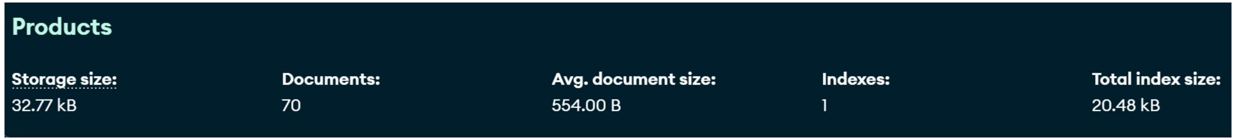


```
4 orders: Array (86)
    0: ObjectId('6568f4f992dce42889d784f1')
    1: ObjectId('6568f4f992dce42889d784f3')
    2: ObjectId('6568f4f992dce42889d784f5')
    3: ObjectId('6568f4f992dce42889d784f7')
```

Assumptions and details

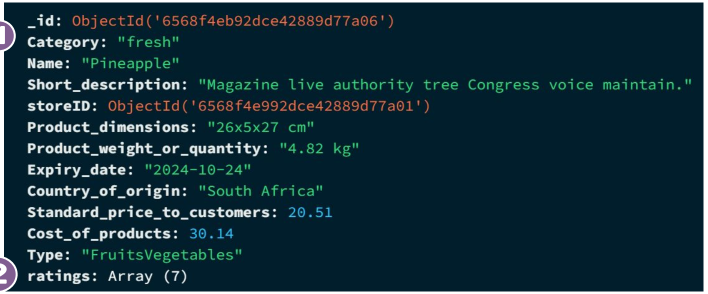
- Geolocation of Customer are scattered around base locations (5 region within manchester)
- Addresses and other details generated by Faker python library.
- Name and Orders in Customers collection are orders within 365 days. Older than that, it goes to 'PastOrders' collection
- No short ID since all process involving ID are most likely done server-side so ObjectId would suffice
- Addresses are object since one customer may have >1 address

Products Sample Data



Products

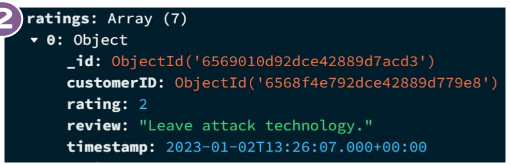
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
32.77 kB	70	554.00 B	1	20.48 kB



1

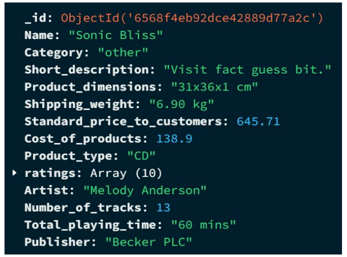
```
_id: ObjectId('6568f4eb92dce42889d77a06')
Category: "fresh"
Name: "Pineapple"
Short_description: "Magazine live authority tree Congress voice maintain."
storeID: ObjectId('6568f4e992dce42889d77a01')
Product_dimensions: "26x5x27 cm"
Product_weight_or_quantity: "4.82 kg"
Expiry_date: "2024-10-24"
Country_of_origin: "South Africa"
Standard_price_to_customers: 20.51
Cost_of_products: 30.14
Type: "FruitsVegetables"
ratings: Array (7)
```

2



2

```
ratings: Array (7)
▾ 0: Object
    _id: ObjectId('6569010d92dce42889d7acd3')
    customerID: ObjectId('6568f4e792dce42889d779e8')
    rating: 2
    review: "Leave attack technology."
    timestamp: 2023-01-02T13:26:07.000+00:00
```

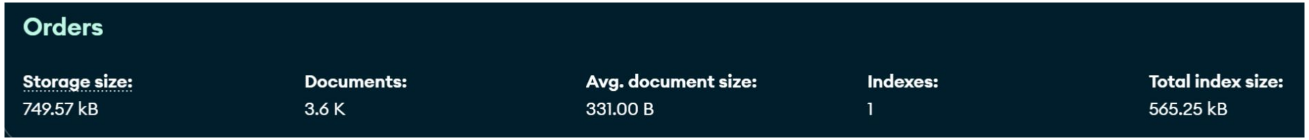


```
  _id: ObjectId('6568f4eb92dce42889d77a2c')
  Name: "Sonic Bliss"
  Category: "other"
  Short_description: "Visit fact guess bit."
  Product_dimensions: "31x36x1 cm"
  Shipping_weight: "6.90 kg"
  Standard_price_to_customers: 645.71
  Cost_of_products: 138.9
  Product_type: "CD"
▸ ratings: Array (10)
  Artist: "Melody Anderson"
  Number_of_tracks: 13
  Total_playing_time: "60 mins"
  Publisher: "Becker PLC"
```

Assumptions and details

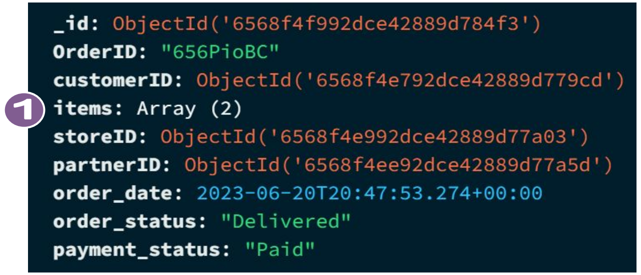
- Category: Fresh or Other ('Other' has their own additional attribute such as Book with attribute 'author name', 'publisher', 'year of publication', 'ISBN', etc)
- Ratings: displayed as an array that showing all of ratings given to each product, referenced to <Customers._id>.

Orders and PastOrders Sample Data

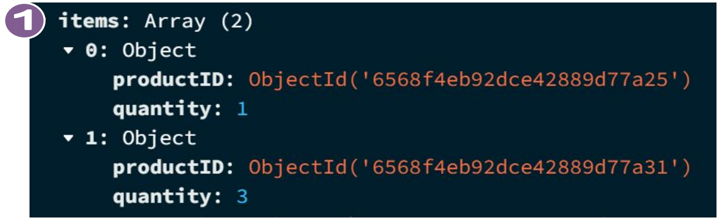


Orders

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
749.57 kB	3.6 K	331.00 B	1	565.25 kB



```
_id: ObjectId('6568f4f992dce42889d784f3')
OrderID: "656PioBC"
customerID: ObjectId('6568f4e792dce42889d779cd')
items: Array (2)
storeID: ObjectId('6568f4e992dce42889d77a03')
partnerID: ObjectId('6568f4ee92dce42889d77a5d')
order_date: 2023-06-20T20:47:53.274+00:00
order_status: "Delivered"
payment_status: "Paid"
```

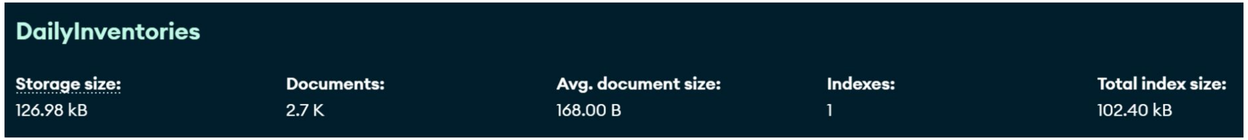


```
items: Array (2)
 ▾ 0: Object
     productID: ObjectId('6568f4eb92dce42889d77a25')
     quantity: 1
 ▾ 1: Object
     productID: ObjectId('6568f4eb92dce42889d77a31')
     quantity: 3
```

Assumptions and details

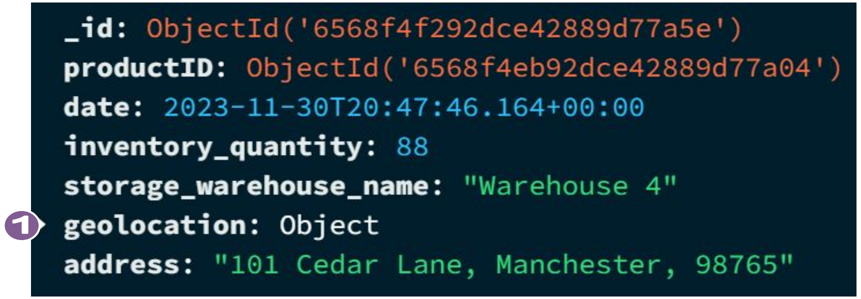
- OrderID provides a human-readable identification system for efficient communication in various scenarios, such as addressing customer complaints.
- Orders are varied from 3 years ago up until present, which then further filtered, anything above 365 days with status of ‘Paid’ and ‘Delivered’ goes to the ‘PastOrders’
- Possible order statuses are ‘Delivered’, ‘In ‘Cart’, ‘Returned’
- Possible payment statuses are ‘Paid’, ‘Unpaid’, ‘Refunded’

Daily Inventories Sample Data

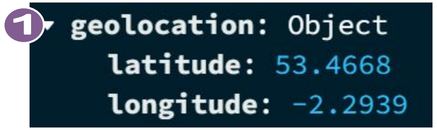


DailyInventories

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
126.98 kB	2.7 K	168.00 B	1	102.40 kB



```
_id: ObjectId('6568f4f292dce42889d77a5e')
productID: ObjectId('6568f4eb92dce42889d77a04')
date: 2023-11-30T20:47:46.164+00:00
inventory_quantity: 88
storage_warehouse_name: "Warehouse 4"
1 geolocation: Object
address: "101 Cedar Lane, Manchester, 98765"
```

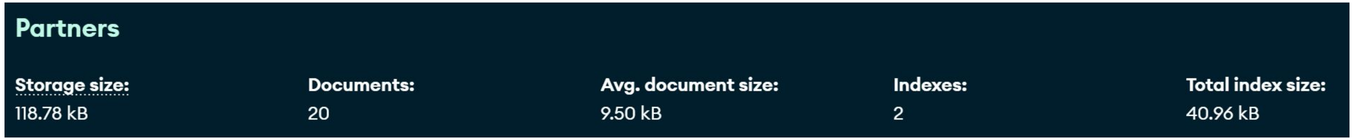


```
1 geolocation: Object
     latitude: 53.4668
     longitude: -2.2939
```

Assumptions and details

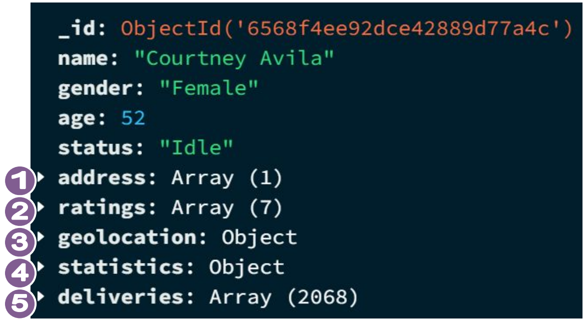
- Geolocation of warehouse are scattered around base locations (5 region within manchester)
- Address and other details generated by Faker python library.
- Address is simple string since a warehouse will only have one address

Partners Sample Data

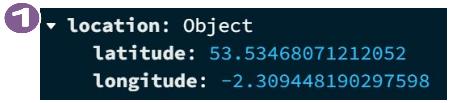


Partners

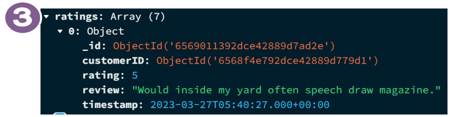
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
118.78 kB	20	9.50 kB	2	40.96 kB



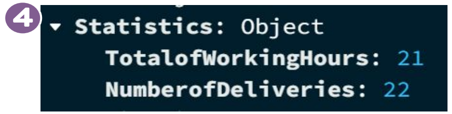
```
_id: ObjectId('6568f4ee92dce42889d77a4c')
name: "Courtney Avila"
gender: "Female"
age: 52
status: "Idle"
1 ▸ address: Array (1)
2 ▸ ratings: Array (7)
3 ▸ geolocation: Object
4 ▸ statistics: Object
5 ▸ deliveries: Array (2068)
```



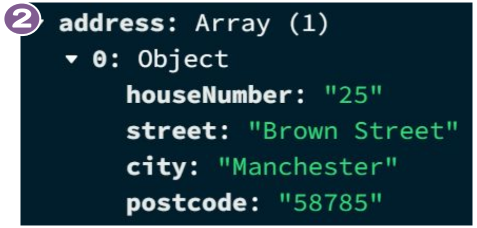
```
1 ▾ location: Object
      latitude: 53.53468071212052
      longitude: -2.309448190297598
```



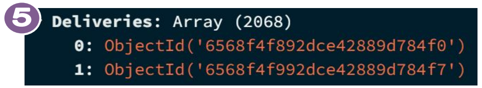
```
3 ▾ ratings: Array (7)
    ▾ 0: Object
        _id: ObjectId('6569011392dce42889d7ad2e')
        customerID: ObjectId('6568f4e792dce42889d779d1')
        rating: 5
        review: "Would inside my yard often speech draw magazine."
        timestamp: 2023-03-27T05:40:27.000+00:00
```



```
4 ▾ Statistics: Object
      TotalofWorkingHours: 21
      NumberofDeliveries: 22
```



```
2 address: Array (1)
    ▾ 0: Object
        houseNumber: "25"
        street: "Brown Street"
        city: "Manchester"
        postcode: "58785"
```

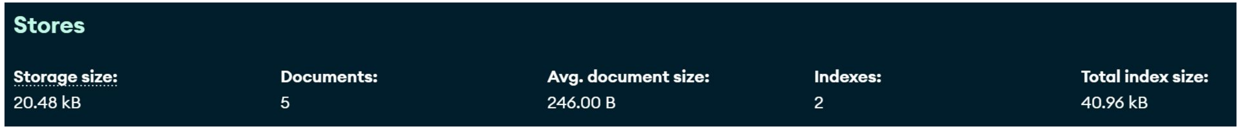


```
5 Deliveries: Array (2068)
    0: ObjectId('6568f4f892dce42889d784f0')
    1: ObjectId('6568f4f992dce42889d784f7')
```

Assumptions and details

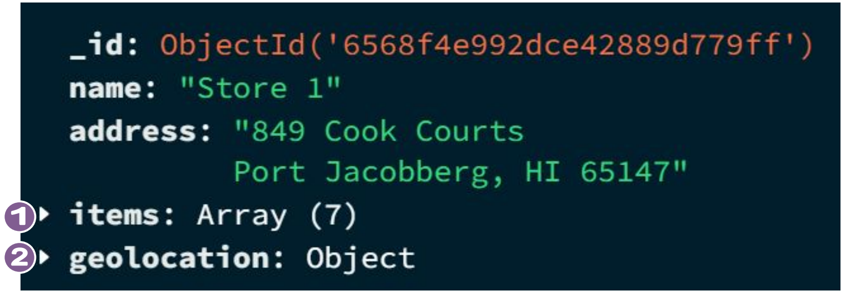
- Geolocation of Partners are scattered around base locations (5 region within manchester)
- Addresses and other details generated by Faker python library.
- Deliveries consists of order deliveries done by the partners. Might be cleansed every three years or so, since the management is on Morrizons, but Amazone needs the statistics to calculate partners' efficiency and perhaps giving annual bonus to the highest performing partners
- No short ID since Partners are managed by Morrizons and Amazone-side, we just need to find idle partners
- Addresses are object since one customer may have >1 address

Stores Sample Data

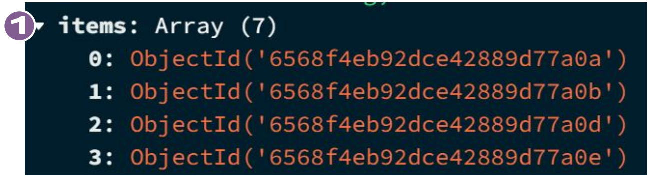


Stores

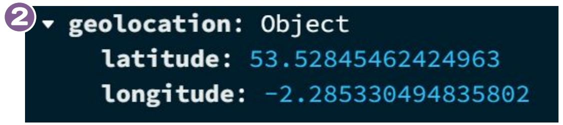
Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	5	246.00 B	2	40.96 kB



```
  _id: ObjectId('6568f4e992dce42889d779ff')
  name: "Store 1"
  address: "849 Cook Courts
           Port Jacobberg, HI 65147"
1 ▸ items: Array (7)
2 ▸ geolocation: Object
```



```
1 ▾ items: Array (7)
      0: ObjectId('6568f4eb92dce42889d77a0a')
      1: ObjectId('6568f4eb92dce42889d77a0b')
      2: ObjectId('6568f4eb92dce42889d77a0d')
      3: ObjectId('6568f4eb92dce42889d77a0e')
```



```
2 ▾ geolocation: Object
      latitude: 53.52845462424963
      longitude: -2.285330494835802
```

Assumptions and details

- Geolocation of Stores are scattered around base locations (5 region within manchester)
- Address and other details generated by Faker python library.
- Items consist of items sold in that particular store
- Addresses are simple string since one store only has one address

3

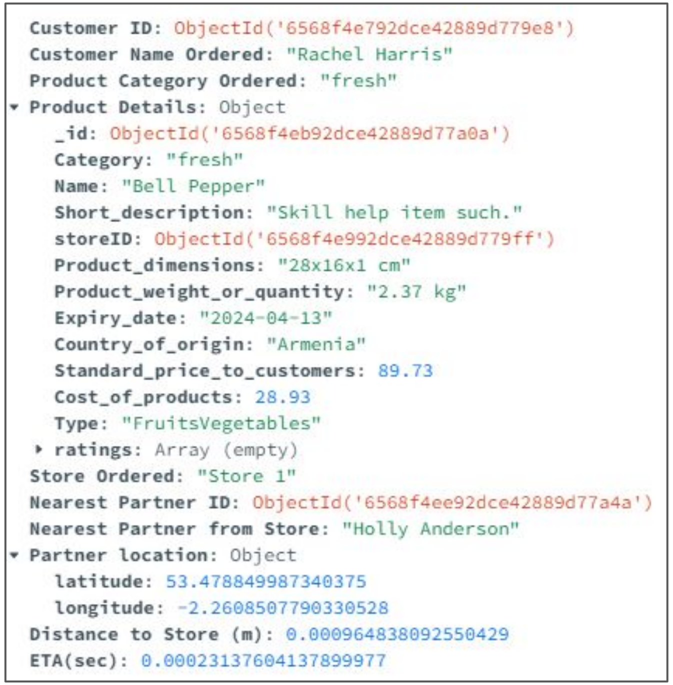
Query & Results

Query-1 : Customer Order Fresh Product, Assign a Pickup Delivery, showing the detail of products & partners.

Aggregation Pipeline

```
db.Customers.aggregate([ { $match: { _id: ObjectId("6568f4e792dce42889d779e8") } },
 { $lookup: { from: "Stores",
   let: { customerLocation: "$geolocation" },
   pipeline: [
     { $geoNear: { near: "$$customerLocation", distanceField: "distance", spherical: true } },
     { $match: { name: "Store 1" } },
     { $unwind: "$items" },
     { $lookup: { from: "Products", localField: "items", foreignField: "_id", as: "itemDetails" } },
     { $unwind: "$itemDetails" },
     { $match: { "itemDetails._id": ObjectId("6568f4eb92dce42889d77a0a") } } ],    as: "store"  } },
 { $unwind: "$store" },
 { $lookup: { from: "Partners",
   let: { storeLocation: "$store.geolocation" },
   pipeline: [
     { $geoNear: { near: "$$storeLocation", distanceField: "distance", spherical: true } },
     { $match: { status: "Active" } },
     { $sort: { distance: 1 } },
     { $limit: 1 } ],    as: "partner"  } },
 { $unwind: "$partner" },
 { $project: { _id: 0,
   "Customer ID": "$_id",
   "Customer Name Ordered": "$name",
   "Product Category Ordered": "$store.itemDetails.Category",
    "Product Details": "$store.itemDetails",
   "Store Ordered": "$store.name",
   "Nearest Partner ID": "$partner._id",
   "Nearest Partner from Store": "$partner.name",
   "Partner rating": "$partner.ratings",
   "Partner location": "$partner.location",
   "Distance to Store (m)": "$partner.distance",
   "ETA(sec)": { $divide: ["$partner.distance", 4.17] } } }]);
```

Results

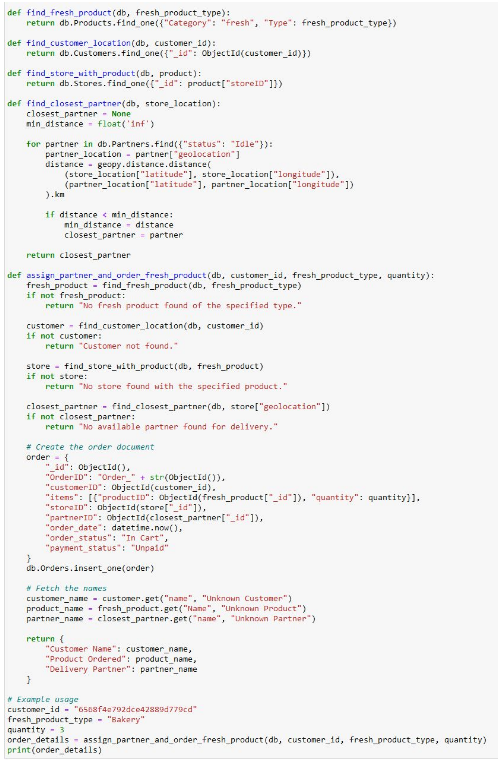


```
Customer ID: ObjectId('6568f4e792dce42889d779e8')
Customer Name Ordered: "Rachel Harris"
Product Category Ordered: "fresh"
▾ Product Details: Object
    _id: ObjectId('6568f4eb92dce42889d77a0a')
    Category: "fresh"
    Name: "Bell Pepper"
    Short_description: "Skill help item such."
    storeID: ObjectId('6568f4e992dce42889d779ff')
    Product_dimensions: "28x16x1 cm"
    Product_weight_or_quantity: "2.37 kg"
    Expiry_date: "2024-04-13"
    Country_of_origin: "Armenia"
    Standard_price_to_customers: 89.73
    Cost_of_products: 28.93
    Type: "FruitsVegetables"
  ▸ ratings: Array (empty)
Store Ordered: "Store 1"
Nearest Partner ID: ObjectId('6568f4ee92dce42889d77a4a')
Nearest Partner from Store: "Holly Anderson"
▾ Partner location: Object
    latitude: 53.478849987340375
    longitude: -2.2608507790330528
Distance to Store (m): 0.000964838092550429
ETA(sec): 0.00023137604137899977
```

Execution Time: 5ms

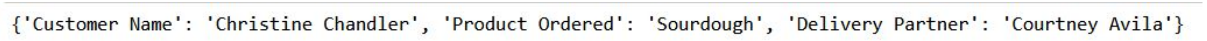
Query-1 : Customer Order Fresh Product, Assign a Pickup Delivery, showing the detail of products & partners.

CRUD on Python



```python
def find_fresh_product(db, fresh_product_type):
    return db.Products.find_one({"Category": "fresh", "Type": fresh_product_type})

def find_customer_location(db, customer_id):
    return db.Customers.find_one({"_id": ObjectId(customer_id)})

def find_store_with_product(db, product):
    return db.Stores.find_one({"_id": product["storeID"]})

def find_closest_partner(db, store_location):
    closest_partner = None
    min_distance = float('inf')

    for partner in db.Partners.find({"status": "Idle"}):
        partner_location = partner["geolocation"]
        distance = geopy.distance.distance(
            (store_location["latitude"], store_location["longitude"]),
            (partner_location["latitude"], partner_location["longitude"])
        ).km

        if distance < min_distance:
            min_distance = distance
            closest_partner = partner

    return closest_partner

def assign_partner_and_order_fresh_product(db, customer_id, fresh_product_type, quantity):
    fresh_product = find_fresh_product(db, fresh_product_type)
    if not fresh_product:
        return "No fresh product found of the specified type."

    customer = find_customer_location(db, customer_id)
    if not customer:
        return "Customer not found."

    store = find_store_with_product(db, fresh_product)
    if not store:
        return "No store found with the specified product."

    closest_partner = find_closest_partner(db, store["geolocation"])
    if not closest_partner:
        return "No available partner found for delivery."

    # Create the order document
    order = {
        "_id": ObjectId(),
        "OrderID": "Order_" + str(ObjectId()),
        "customerID": ObjectId(customer_id),
        "items": [{"productID": ObjectId(fresh_product["_id"]), "quantity": quantity}],
        "storeID": ObjectId(store["_id"]),
        "partnerID": ObjectId(closest_partner["_id"]),
        "order_date": datetime.now(),
        "order_status": "In Cart",
        "payment_status": "Unpaid"
    }
    db.Orders.insert_one(order)

    # Fetch the names
    customer_name = customer.get("name", "Unknown Customer")
    product_name = fresh_product.get("Name", "Unknown Product")
    partner_name = closest_partner.get("name", "Unknown Partner")

    return {
        "Customer Name": customer_name,
        "Product Ordered": product_name,
        "Delivery Partner": partner_name
    }

# Example usage
customer_id = "6568f4e792dce42889d779cd"
fresh_product_type = "Bakery"
quantity = 3
order_details = assign_partner_and_order_fresh_product(db, customer_id, fresh_product_type, quantity)
print(order_details)
```

Results



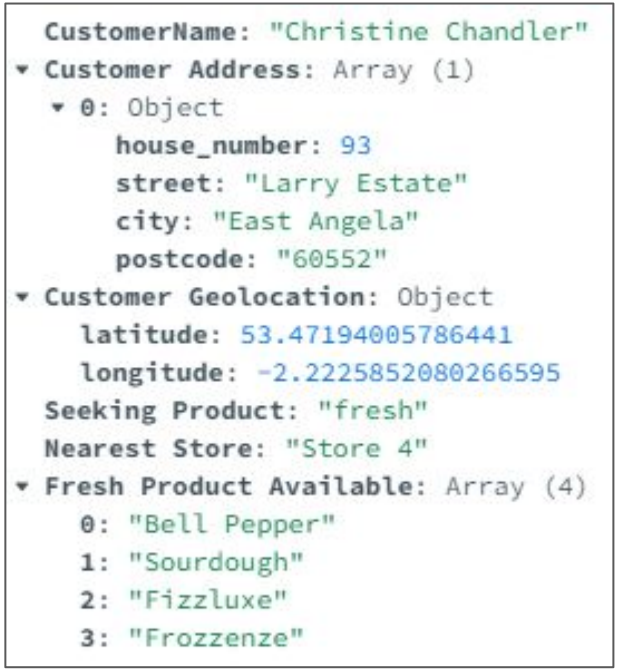
```
{'Customer Name': 'Christine Chandler', 'Product Ordered': 'Sourdough', 'Delivery Partner': 'Courtney Avila'}
```

Query-2 : Cust searching for available fresh products. The products should be displayed based on the user's location.

Aggregation Pipeline

```
db.Customers.aggregate([ { $match: { _id: ObjectId("6568f4e792dce42889d779cd") } },
 { $lookup: { from: "Stores",
   let: { customerLocation: "$geolocation" },
   pipeline: [
    { $geoNear: { near: "$$customerLocation", distanceField: "dist.calculated",
      includeLocs: "dist.geolocation", spherical: true } },
    { $limit: 1 },
    { $lookup: { from: "Products", localField: "items", foreignField: "_id", as: "productDetails" } },
    { $unwind: "$productDetails" } ],   as: "nearestStore" } },
 { $unwind: "$nearestStore" },
 { $match: { "nearestStore.productDetails.Category": "fresh" } },
 { $group: {  _id: "$_id",
   name: { $first: "$name" },
   addresses: { $first: "$addresses" },
   Category: { $first: "$nearestStore.productDetails.Category" },
   geolocation: { $first: "$geolocation" },
   nearestStoreName: { $first: "$nearestStore.name" },
   freshProductsAvailable: { $push: "$nearestStore.productDetails.Name" } } },
 { $project: {    _id: 0,
   CustomerName: "$name",
   "Customer Address": "$addresses",
   "Customer Geolocation": "$geolocation",
   "Seeking Product": "$Category",
   "Nearest Store": "$nearestStoreName",
   "Fresh Product Available": "$freshProductsAvailable" } }]);
```

Results



```
  CustomerName: "Christine Chandler"
▾ Customer Address: Array (1)
  ▾ 0: Object
      house_number: 93
      street: "Larry Estate"
      city: "East Angela"
      postcode: "60552"
▾ Customer Geolocation: Object
    latitude: 53.4719400578644l
    longitude: -2.225852080266595
  Seeking Product: "fresh"
  Nearest Store: "Store 4"
▾ Fresh Product Available: Array (4)
    0: "Bell Pepper"
    1: "Sourdough"
    2: "Fizzluxe"
    3: "Frozzenze"
```

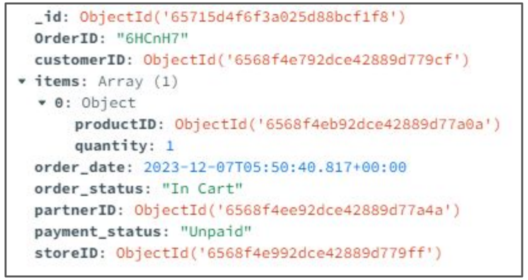
Execution Time: 2ms

Query-3 : Customer Ordering Product, Adding to Cart, Making Payment

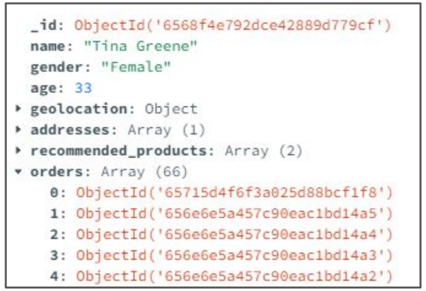
Aggregation Pipeline

```
db.Customers.aggregate([{$match: { _id: ObjectId("6568f4e792dce42889d779cf"),},},
  {$project: { customerID: "$_id",
                  order_date: new Date(),
                  items: [{productID: ObjectId("6568f4eb92dce42889d77a0a"),
                            quantity: 1,},],
                   storeID: ObjectId("6568f4e992dce42889d779ff"),
                   partnerID: ObjectId("6568f4ee92dce42889d77a4a"),
                   order_date: new Date(),
                   order_status: "In Cart",
                   payment_status: "Unpaid",},},
  {$merge: {into: "Orders",
                 whenMatched: "merge",
                 whenNotMatched: "insert",},])

db.Customers.aggregate([{$match: {_id: ObjectID("6568f4e792dce42889d779cf") } },
  {$lookup: {from: "Orders",
     localField: "_id",
     foreignField: "customerID",
     as: "customerOrders" } },
  {$unwind: "$customerOrders"},
  {$sort: {"customerOrders.order_date": -1}},
  {$group: {_id: "$_id",
     latestOrder: { $first: "$customerOrders._id" },
     orders: { $push: "$customerOrders._id" }}},
  {$project: {_id: 1, orders: "$orders"}},
  {$merge: "Customers" }])

db.Orders.aggregate([{ $match: {customerID: ObjectId("6568f4e792dce42889d779cf") }},
  {$sort: {order_date: -1}},
  {$limit: 1}},
  {$set: {payment_status: "Paid",  order_status: "On Delivery"}},
   {$merge: {into: "Orders",  whenMatched: "merge"}}])
```

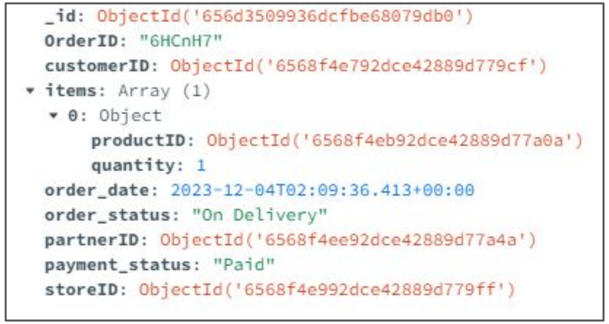
Results



```
_id: ObjectId('65715d4f6f3a025d88bcf1f8')
OrderID: "6HCnH7"
customerID: ObjectId('6568f4e792dce42889d779cf')
▾ items: Array (1)
  ▾ 0: Object
      productID: ObjectId('6568f4eb92dce42889d77a0a')
      quantity: 1
order_date: 2023-12-07T05:50:40.817+00:00
order_status: "In Cart"
partnerID: ObjectId('6568f4ee92dce42889d77a4a')
payment_status: "Unpaid"
storeID: ObjectId('6568f4e992dce42889d779ff')
```



```
_id: ObjectId('6568f4e792dce42889d779cf')
name: "Tina Greene"
gender: "Female"
age: 33
▸ geolocation: Object
▸ addresses: Array (1)
▸ recommended_products: Array (2)
▾ orders: Array (66)
    0: ObjectId('65715d4f6f3a025d88bcf1f8')
    1: ObjectId('656e6e5a457c90eac1bd14a5')
    2: ObjectId('656e6e5a457c90eac1bd14a4')
    3: ObjectId('656e6e5a457c90eac1bd14a3')
    4: ObjectId('656e6e5a457c90eac1bd14a2')
```

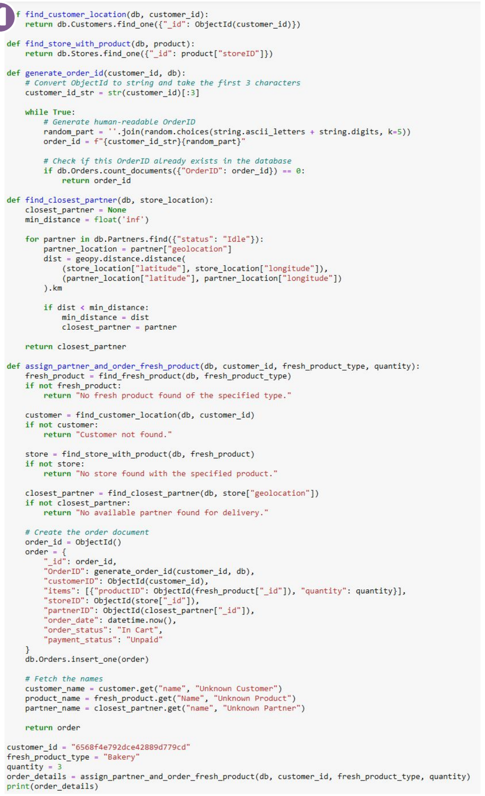


```
_id: ObjectId('656d3509936dcfbe68079db0')
OrderID: "6HCnH7"
customerID: ObjectId('6568f4e792dce42889d779cf')
▾ items: Array (1)
  ▾ 0: Object
      productID: ObjectId('6568f4eb92dce42889d77a0a')
      quantity: 1
order_date: 2023-12-04T02:09:36.413+00:00
order_status: "On Delivery"
partnerID: ObjectId('6568f4ee92dce42889d77a4a')
payment_status: "Paid"
storeID: ObjectId('6568f4e992dce42889d779ff')
```

MANCHESTER
1824
The University of Manchester

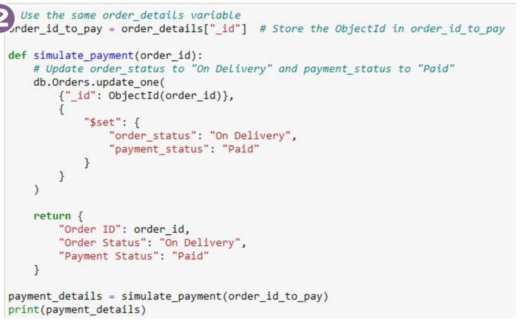
Query-3 : Customer Ordering Product, Adding to Cart, Making Payment

CRUD on Python



1

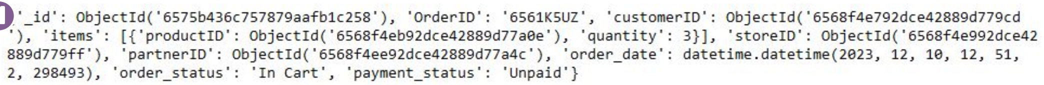
```python
f find_customer_location(db, customer_id):
    return db.Customers.find_one({"_id": ObjectId(customer_id)})

def find_store_with_product(db, product):
    return db.Stores.find_one({"_id": product["storeID"]})

def generate_order_id(customer_id, db):
    # Convert ObjectId to string and take the first 3 characters
    customer_id_str = str(customer_id)[:3]

    while True:
        # Generate human-readable OrderID
        random_part = ''.join(random.choices(string.ascii_letters + string.digits, k=5))
        order_id = f"{customer_id_str}{random_part}"

        # Check if this OrderID already exists in the database
        if db.Orders.count_documents({"OrderID": order_id}) == 0:
            return order_id

def find_closest_partner(db, store_location):
    closest_partner = None
    min_distance = float('inf')

    for partner in db.Partners.find({"status": "Idle"}):
        partner_location = partner["geolocation"]
        dist = geopy.distance.distance(
            (store_location["latitude"], store_location["longitude"]),
            (partner_location["latitude"], partner_location["longitude"])
        ).km

        if dist < min_distance:
            min_distance = dist
            closest_partner = partner

    return closest_partner

def assign_partner_and_order_fresh_product(db, customer_id, fresh_product_type, quantity):
    fresh_product = find_fresh_product(db, fresh_product_type)
    if not fresh_product:
        return "No fresh product found of the specified type."

    customer = find_customer_location(db, customer_id)
    if not customer:
        return "Customer not found."

    store = find_store_with_product(db, fresh_product)
    if not store:
        return "No store found with the specified product."

    closest_partner = find_closest_partner(db, store["geolocation"])
    if not closest_partner:
        return "No available partner found for delivery."

    # Create the order document
    order_id = ObjectId()
    order = {
        "_id": order_id,
        "OrderID": generate_order_id(customer_id, db),
        "customerID": ObjectId(customer_id),
        "items": [{"productID": ObjectId(fresh_product["_id"]), "quantity": quantity}],
        "storeID": ObjectId(store["_id"]),
        "partnerID": ObjectId(closest_partner["_id"]),
        "order_date": datetime.now(),
        "order_status": "In Cart",
        "payment_status": "Unpaid"
    }
    db.Orders.insert_one(order)

    # Fetch the names
    customer_name = customer.get("name", "Unknown Customer")
    product_name = fresh_product.get("Name", "Unknown Product")
    partner_name = closest_partner.get("name", "Unknown Partner")

    return order

customer_id = "6568f4e792dce42889d779cd"
fresh_product_type = "Bakery"
quantity = 3
order_details = assign_partner_and_order_fresh_product(db, customer_id, fresh_product_type, quantity)
print(order_details)
```



2

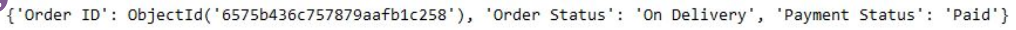
```python
 Use the same order_details variable
rder_id_to_pay = order_details["_id"]  # Store the ObjectId in order_id_to_pay

def simulate_payment(order_id):
    # Update order_status to "On Delivery" and payment_status to "Paid"
    db.Orders.update_one(
        {"_id": ObjectId(order_id)},
        {
            "$set": {
                "order_status": "On Delivery",
                "payment_status": "Paid"
            }
        }
    )

    return {
        "Order ID": order_id,
        "Order Status": "On Delivery",
        "Payment Status": "Paid"
    }

payment_details = simulate_payment(order_id_to_pay)
print(payment_details)
```

Results



1

```
'_id': ObjectId('6575b436c757879aafb1c258'), 'OrderID': '6561K5UZ', 'customerID': ObjectId('6568f4e792dce42889d779cd
'), 'items': [{'productID': ObjectId('6568f4eb92dce42889d77a0e'), 'quantity': 3}], 'storeID': ObjectId('6568f4e992dce42
889d779ff'), 'partnerID': ObjectId('6568f4ee92dce42889d77a4c'), 'order_date': datetime.datetime(2023, 12, 10, 12, 51,
2, 298493), 'order_status': 'In Cart', 'payment_status': 'Unpaid'}
```



2

```
{'Order ID': ObjectId('6575b436c757879aafb1c258'), 'Order Status': 'On Delivery', 'Payment Status': 'Paid'}
```

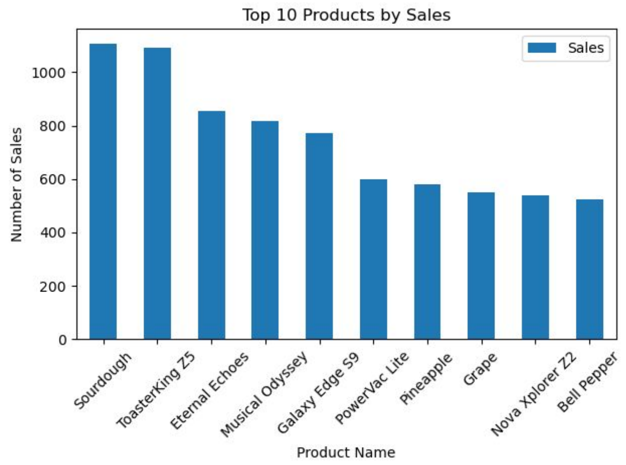
Query-4 : Manager Checking Sales

CRUD on Python

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the data
orders_df = pd.read_json('AMAZONE.Orders.json')

# Filter orders with 'Paid' payment_status and 'Delivered' order_status
filtered_orders_df = orders_df[(orders_df['payment_status'] == 'Paid') & (orders_df['order_status'] == 'Delivered')]

# Explode the items field
filtered_orders_df = filtered_orders_df.explode('items')

# Normalize the productID field and quantity
filtered_orders_df['productID'] = filtered_orders_df['items'].apply(lambda x: x['productID']['$oid'])
filtered_orders_df['quantity'] = filtered_orders_df['items'].apply(lambda x: x['quantity'])

products_df = pd.read_json('AMAZONE.Products.json')
products_df['_id'] = products_df['_id'].apply(lambda x: x['$oid'])

# Merge the Orders and Products dataframes
merged_df = filtered_orders_df.merge(products_df, left_on='productID', right_on='_id', how='left')

# Group by product name and sum the quantities for each product to get total sales
product_sales = merged_df.groupby('Name')['quantity'].sum().reset_index(name='Sales')

# Sort the products by sales
product_sales_sorted = product_sales.sort_values(by='Sales', ascending=False)

# Get the top 10 products with the highest sales
top_10_products = product_sales_sorted.head(10)
print(top_10_products)

plt.figure(figsize=(10, 6))
top_10_products.plot(kind='bar', x='Name', y='Sales', title='Top 10 Products by Sales (Paid and On Delivery)')
plt.xlabel('Product Name')
plt.ylabel('Number of Sales')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

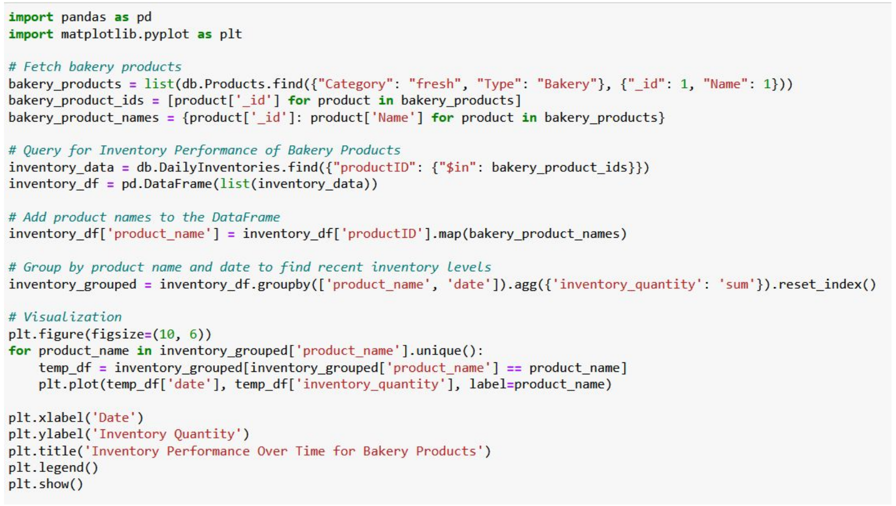
Results

```
              Name  Sales
37       Sourdough   1107
41   ToasterKing Z5   1091
12   Eternal Echoes    854
23  Musical Odyssey    816
15   Galaxy Edge S9    771
30    PowerVac Lite    601
28        Pineapple    582
17            Grape    550
25   Nova Xplorer Z2    540
0       Bell Pepper    525

<Figure size 1000x600 with 0 Axes>
```



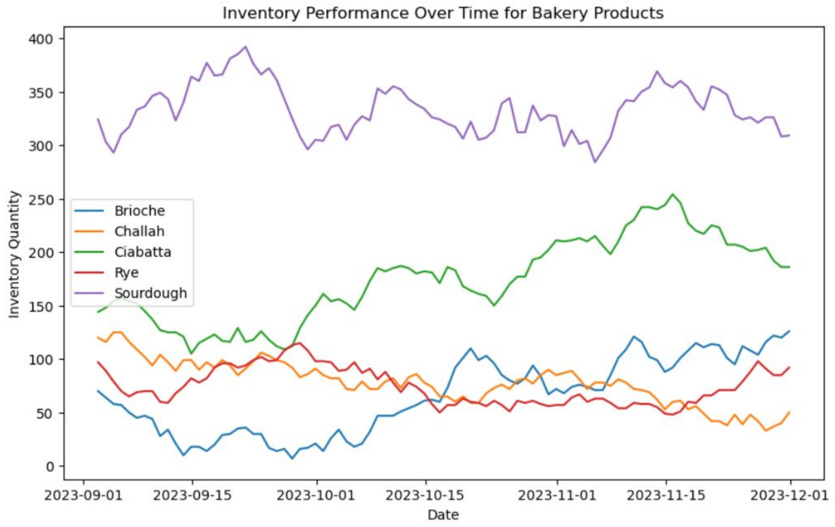
Query-4 : Manager Checking Inventory

CRUD on Python



```python
import pandas as pd
import matplotlib.pyplot as plt

# Fetch bakery products
bakery_products = list(db.Products.find({"Category": "fresh", "Type": "Bakery"}, {"_id": 1, "Name": 1}))
bakery_product_ids = [product['_id'] for product in bakery_products]
bakery_product_names = {product['_id']: product['Name'] for product in bakery_products}

# Query for Inventory Performance of Bakery Products
inventory_data = db.DailyInventories.find({"productID": {"$in": bakery_product_ids}})
inventory_df = pd.DataFrame(list(inventory_data))

# Add product names to the DataFrame
inventory_df['product_name'] = inventory_df['productID'].map(bakery_product_names)

# Group by product name and date to find recent inventory levels
inventory_grouped = inventory_df.groupby(['product_name', 'date']).agg({'inventory_quantity': 'sum'}).reset_index()

# Visualization
plt.figure(figsize=(10, 6))
for product_name in inventory_grouped['product_name'].unique():
    temp_df = inventory_grouped[inventory_grouped['product_name'] == product_name]
    plt.plot(temp_df['date'], temp_df['inventory_quantity'], label=product_name)

plt.xlabel('Date')
plt.ylabel('Inventory Quantity')
plt.title('Inventory Performance Over Time for Bakery Products')
plt.legend()
plt.show()
```

Results

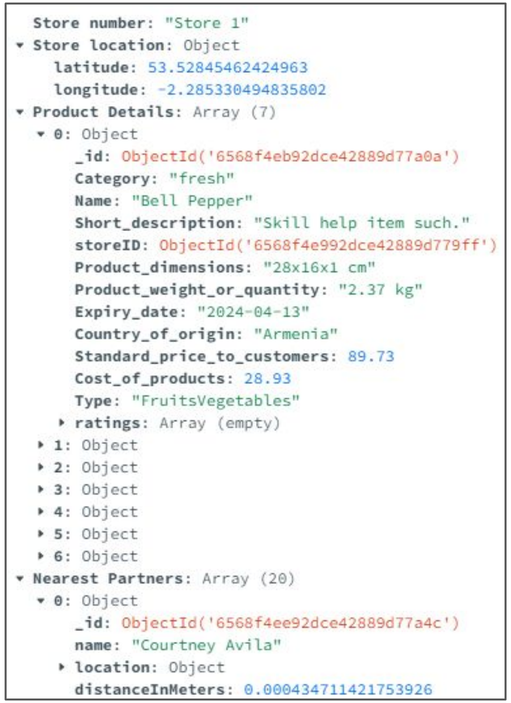


Query-5 (Custom query) : Customers are checking what is being sold at Store-1 and if there's any partner near to pick it up

Aggregation Pipeline

```
db.Stores.aggregate([ { $match: { name: "Store 1" } },
 { $lookup: { from: "Partners",
    let: { storeLocation: "$geolocation" },
    pipeline: [
      { $geoNear: { near: "$$storeLocation",
         distanceField: "dist.calculated",
         includeLocs: "dist.geolocation",
         spherical: true,}},
      { $project: { _id: 1,
         name: 1,
         location: 1,
         distanceInMeters: "$dist.calculated", } },],
    as: "NearestPartners", } },
 { $lookup: {
    from: "Products",
    localField: "items",
    foreignField: "_id",
    as: "ProductDetails", } },
 { $project: {
    _id: 0,
  "Store number": "$name",
    "Store location": "$geolocation",
    "Product Details": "$ProductDetails",
    "Nearest Partners": "$NearestPartners", }  }]);
```

Results



```
  Store number: "Store 1"
▾ Store location: Object
    latitude: 53.52845462424963
    longitude: -2.285330494835802
▾ Product Details: Array (7)
  ▾ 0: Object
      _id: ObjectId('6568f4eb92dce42889d77a0a')
      Category: "fresh"
      Name: "Bell Pepper"
      Short_description: "Skill help item such."
      storeID: ObjectId('6568f4e992dce42889d779ff')
      Product_dimensions: "28x16x1 cm"
      Product_weight_or_quantity: "2.37 kg"
      Expiry_date: "2024-04-13"
      Country_of_origin: "Armenia"
      Standard_price_to_customers: 89.73
      Cost_of_products: 28.93
      Type: "FruitsVegetables"
    ▸ ratings: Array (empty)
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
  ▸ 5: Object
  ▸ 6: Object
▾ Nearest Partners: Array (20)
  ▾ 0: Object
      _id: ObjectId('6568f4ee92dce42889d77a4c')
      name: "Courtney Avila"
    ▸ location: Object
      distanceInMeters: 0.000434711421753926
```

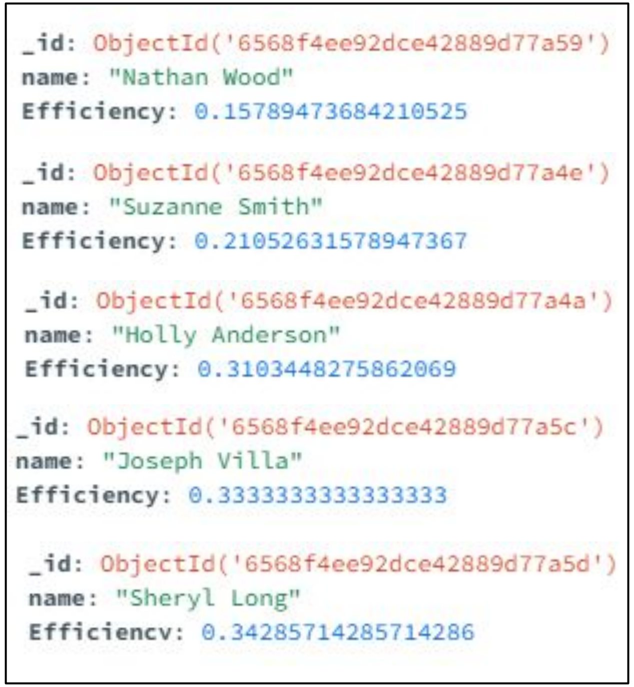
Execution Time: 3ms

Query-6 (Custom query) : Amazone checking to see which of their Partners are inefficient

Aggregation Pipeline

```
db.Partners.aggregate([
 {$project: {
    _id: 1,
    name: 1,
    Efficiency: {
      $divide: [
        "$statistics.NumberofDeliveries",
        "$statistics.TotalofWorkingHours",],},},},
 {$match:
    //Efficiency less than 0.5 is inefficient
    {Efficiency: {$lt: 0.5,},},},
 {$sort: {Efficiency: 1,},},])
```

Results



```
_id: ObjectId('6568f4ee92dce42889d77a59')
name: "Nathan Wood"
Efficiency: 0.15789473684210525

_id: ObjectId('6568f4ee92dce42889d77a4e')
name: "Suzanne Smith"
Efficiency: 0.21052631578947367

_id: ObjectId('6568f4ee92dce42889d77a4a')
name: "Holly Anderson"
Efficiency: 0.3103448275862069

_id: ObjectId('6568f4ee92dce42889d77a5c')
name: "Joseph Villa"
Efficiency: 0.3333333333333333

_id: ObjectId('6568f4ee92dce42889d77a5d')
name: "Sheryl Long"
Efficiencv: 0.34285714285714286
```

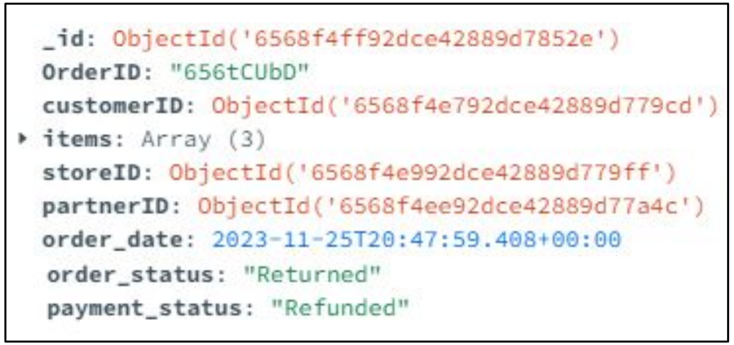
MANCHESTER 1824
The University of Manchester

Query-7 (Custom query) : Customer returns their latest delivery and receives refund

Aggregation Pipeline

```
db.Customers.aggregate([
 {$match: {_id: ObjectId("6568f4e792dce42889d779cd"),},},
 {$unwind: "$orders",},
 {$lookup: {
    from: "Orders",
    localField: "orders",
    foreignField: "_id",
    as: "orderDetails",},},
 {$unwind: "$orderDetails",},
 {$match: {"orderDetails.order_status": "Delivered",},},
 {$sort: {"orderDetails.order_date": -1,},},
 {$group: {
    _id: "$_id",
    latestDeliveredOrder: {
      $first: "$orderDetails",},},},
 {$set:{
      "latestDeliveredOrder.order_status":
        "Returned",
      "latestDeliveredOrder.payment_status":
        "Refunded",},},
   {$replaceWith: "$latestDeliveredOrder",},
 {$merge: {into: {db: "AMAZONE", coll: "Orders",},
    whenMatched: "merge",
    whenNotMatched: "discard",},},])
```

Results



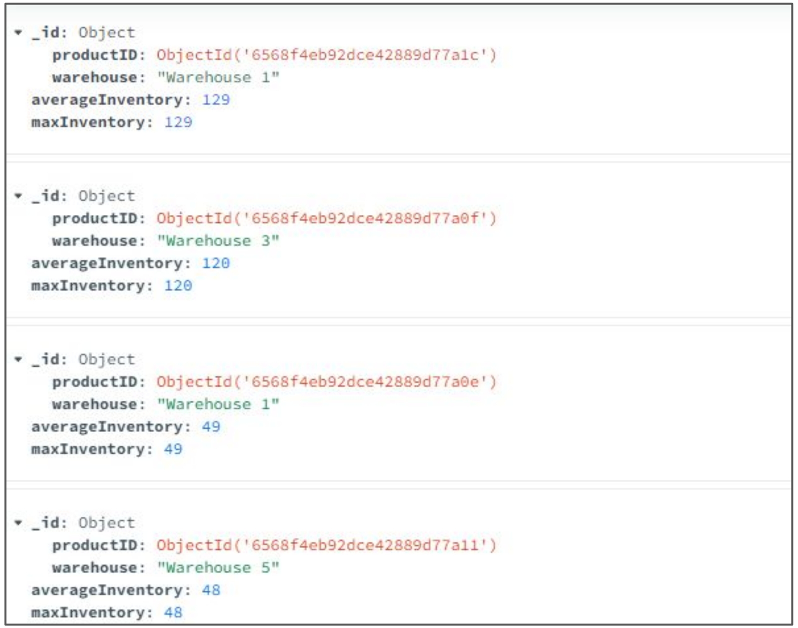
```
  _id: ObjectId('6568f4ff92dce42889d7852e')
  OrderID: "656tCUbD"
  customerID: ObjectId('6568f4e792dce42889d779cd')
▸ items: Array (3)
  storeID: ObjectId('6568f4e992dce42889d779ff')
  partnerID: ObjectId('6568f4ee92dce42889d77a4c')
  order_date: 2023-11-25T20:47:59.408+00:00
  order_status: "Returned"
  payment_status: "Refunded"
```

Query-8 (Custom query) : specified date range, providing the average and maximum inventory quantities for each product in each warehouse

Aggregation Pipeline

```
db.getCollection('DailyInventories').aggregate([
   {$match:
{date: {$gte: ISODate('2023-09-01T00:00:00.000Z'),
        $lte: ISODate('2023-09-02T23:59:59.999Z')}}},
   {$group:
{_id: {productID: '$productID',
warehouse: '$storage_warehouse_name'},
averageInventory: {
      $avg: '$inventory_quantity'},
      maxInventory: {
      $max: '$inventory_quantity'}}}],
  { maxTimeMS: 60000, allowDiskUse: true });
```

Results



```
▾ _id: Object
    productID: ObjectId('6568f4eb92dce42889d77a1c')
    warehouse: "Warehouse 1"
  averageInventory: 129
  maxInventory: 129

▾ _id: Object
    productID: ObjectId('6568f4eb92dce42889d77a0f')
    warehouse: "Warehouse 3"
  averageInventory: 120
  maxInventory: 120

▾ _id: Object
    productID: ObjectId('6568f4eb92dce42889d77a0e')
    warehouse: "Warehouse 1"
  averageInventory: 49
  maxInventory: 49

▾ _id: Object
    productID: ObjectId('6568f4eb92dce42889d77a11')
    warehouse: "Warehouse 5"
  averageInventory: 48
  maxInventory: 48
```

Thank you

MANCHESTER 1824
The University of Manchester