Project: Distributed Commerce Gateway

Part_5

Distributed NoSQL Database Systems Implementation

CSE 512 Distributed Database Systems 2023

Professor - *Bharatesh Chakravarthi*

Submitted by:

Hitaxi Mistry - 1222126334 - hmistry1@asu.edu

Naga Venkata Dharani Chinta - 1228137224 - nvchinta@asu.edu

Monalisa Dokania - 122534460 - mdokania@asu.edu

Rajkumar Kakumanu - 1225696602 - rkakuman@asu.edu

Submission Date - Nov 27, 2023

ASU School of Computing and Augmented Intelligence

Ira A. Fulton School of Engineering, Arizona State University, Tempe

Abstract

In this assignment, we have worked with MongoDB, a distributed No-SQL database system management and performing different CRUD operations on the collections created in the database. The main idea behind this part of the project is to understand and learn the storage and retrieval process.

For our topic in the project “E-commerce”, we are utilizing the same schema which we used for the other parts in PostgreSQL. We have explained the README.MD file in the directory to guide the process of running the code and obtaining the required outputs. It is very important to install the libraries, import them, and run the mongodb image in the docker container. The term CRUD represents “Create, Read, Update and Delete”. As we are working with the NoSQL database, the schema of the database is not strict and it is not structured as tables and relationships, rather we are working with json data type.

Implementation

The database name is “e-commerce” and the collections created are mentioned below in the list.

```
# Database name and collections
DATABASE_NAME = "e_commerce"
COLLECTION_LIST = ["Users", "Categories", "Products", "Orders", "OrderDetails", "Transactions", "Reviews", "Inventory", "Shipping"]
```

✓ 0.2s Python

The collections have been created using the function. The mock_data for each collection has been generated using the https://www.mockaroo.com/ and uploaded along with the code in a json format.

In order to showcase the functionality of our NoSQL database, we have implemented sample CRUD operations for each collection. The entire code can be viewed in the

part-5 directory and we have mentioned some of the important sample queries in this report, which is crucial from the user perspective.

CRUD operations

Create:

The collections are created similar to the below displayed code. The data is inserted using the mock_data.json file.

Example:1

In this example, the collection “Users” is created and the user information is added into it.

```
# Insert data into Users collection
#users_collection.delete_many({})
users_collection = db["Users"]

# Load mock data from a JSON file
with open('users_mock_data.json', 'r') as file:
    mock_data = json.load(file)

insert_result = users_collection.insert_many(mock_data)

print(insert_result)
print("Data inserted successfully!")
```

✓ 0.3s Pytho

```
<pymongo.results.InsertManyResult object at 0x114a36ef0>
Data inserted successfully!
```

Example:2

In this example, the collection “Shipping” is created and the user information is added into it.

```
# Insert data into Shipping collection
shipping_collection = db["Shipping"]

# Load mock data from a JSON file
with open('shipping_mock_data.json', 'r') as file:
    mock_data = json.load(file)

insert_result = shipping_collection.insert_many(mock_data)

print(insert_result)
print("Data inserted successfully!")
```

✓ 0.3s Python

```
<pymongo.results.InsertManyResult object at 0x115deafe0>
Data inserted successfully!
```

Example: 3

In this example, the collection “Orders” is created and the user information is added into it.

```python
# Insert data into Orders collection
#orders_collection.delete_many({})
orders_collection = db["Orders"]

# Load mock data from a JSON file
with open('order_mock_data.json', 'r') as file:
    mock_data = json.load(file)

insert_result = orders_collection.insert_many(mock_data)

print(insert_result)
print("Data inserted successfully!")
```

✓ 0.4s Python

```
<pymongo.results.InsertManyResult object at 0x115d235e0>
Data inserted successfully!
```

Read:

In this operation, the documents are read from the collection and according to the user/developer’s specific requirements the data is fetched and printed. Each example has different information fetched from the collection, considering the realistic needs.

Example: 1

Here, the orders having the product_quantity to be 10 are fetched from the collection ‘Order_details’ and displayed.

```python
# Query for orders having quantity 10
query = {"quantity": 10 }

# Fetch data based on the query
cursor = orders_details_collection.find(query)

# Print the results
for user_data in cursor:
    print(user_data)
```

✓ 0.3s Python

```
{'_id': ObjectId('65525e567f69c7083d39f599'), 'order_detail_id': 11, 'order_id': 11, 'product_id': 11, 'quantity': 10, 'subtotal': 282.05}
{'_id': ObjectId('65525e567f69c7083d39f5a3'), 'order_detail_id': 21, 'order_id': 21, 'product_id': 21, 'quantity': 10, 'subtotal': 156.41}
{'_id': ObjectId('65525e567f69c7083d39f5b4'), 'order_detail_id': 38, 'order_id': 38, 'product_id': 38, 'quantity': 10, 'subtotal': 586.52}
{'_id': ObjectId('65525e567f69c7083d39f5b8'), 'order_detail_id': 42, 'order_id': 42, 'product_id': 42, 'quantity': 10, 'subtotal': 308.83}
{'_id': ObjectId('65525e567f69c7083d39f5bd'), 'order_detail_id': 47, 'order_id': 47, 'product_id': 47, 'quantity': 10, 'subtotal': 762.35}
{'_id': ObjectId('65525e567f69c7083d39f5be'), 'order_detail_id': 48, 'order_id': 48, 'product_id': 48, 'quantity': 10, 'subtotal': 216.02}
{'_id': ObjectId('65525e567f69c7083d39f5c3'), 'order_detail_id': 53, 'order_id': 53, 'product_id': 53, 'quantity': 10, 'subtotal': 799.39}
{'_id': ObjectId('65525e567f69c7083d39f5cb'), 'order_detail_id': 61, 'order_id': 61, 'product_id': 61, 'quantity': 10, 'subtotal': 150.37}
{'_id': ObjectId('65525e567f69c7083d39f5cd'), 'order_detail_id': 63, 'order_id': 63, 'product_id': 63, 'quantity': 10, 'subtotal': 49.15}
{'_id': ObjectId('65525e567f69c7083d39f5e3'), 'order_detail_id': 85, 'order_id': 85, 'product_id': 85, 'quantity': 10, 'subtotal': 566.17}
{'_id': ObjectId('65525e567f69c7083d39f5e9'), 'order_detail_id': 91, 'order_id': 91, 'product_id': 91, 'quantity': 10, 'subtotal': 410.63}
```

Example: 2

Here, the transactions having the payment_status to be ‘refunded’ are fetched from the collection ‘Transactions’ and displayed.

```python
# Query for the payment_status to be refunded
query = {"payment_status": 'refunded' }

# Fetch data based on the query
cursor = transaction_collection.find(query)
# Print the results
for user_data in cursor:
    print(user_data)
```

✓ 0.3s Python

```
{'_id': ObjectId('65525ef67f69c7083d39f65b'), 'transaction_id': 8, 'order_id': 3, 'transaction_date': '9/6/2014', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f65f'), 'transaction_id': 5, 'order_id': 1, 'transaction_date': '11/10/2020', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f663'), 'transaction_id': 3, 'order_id': 9, 'transaction_date': '12/13/2017', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f669'), 'transaction_id': 0, 'order_id': 3, 'transaction_date': '11/18/2017', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f66e'), 'transaction_id': 6, 'order_id': 6, 'transaction_date': '7/6/2017', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f67a'), 'transaction_id': 2, 'order_id': 10, 'transaction_date': '5/9/2019', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f686'), 'transaction_id': 0, 'order_id': 9, 'transaction_date': '6/27/2018', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f687'), 'transaction_id': 7, 'order_id': 7, 'transaction_date': '5/21/2012', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f688'), 'transaction_id': 0, 'order_id': 2, 'transaction_date': '9/7/2016', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f691'), 'transaction_id': 8, 'order_id': 9, 'transaction_date': '2/23/2017', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f692'), 'transaction_id': 2, 'order_id': 4, 'transaction_date': '5/3/2012', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f695'), 'transaction_id': 6, 'order_id': 1, 'transaction_date': '7/5/2011', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f69b'), 'transaction_id': 7, 'order_id': 7, 'transaction_date': '11/3/2016', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6a2'), 'transaction_id': 5, 'order_id': 0, 'transaction_date': '2/7/2015', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6a6'), 'transaction_id': 4, 'order_id': 6, 'transaction_date': '1/4/2020', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6ac'), 'transaction_id': 2, 'order_id': 5, 'transaction_date': '12/4/2016', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6b1'), 'transaction_id': 5, 'order_id': 5, 'transaction_date': '1/31/2021', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6b2'), 'transaction_id': 2, 'order_id': 6, 'transaction_date': '3/11/2017', 'payment_status': 'refunded'}
{'_id': ObjectId('65525ef67f69c7083d39f6b5'), 'transaction_id': 0, 'order_id': 3, 'transaction_date': '12/28/2014', 'payment_status': 'refunded'}
```

Example: 3

Here, the count of the products having the ratings to be 1 is performed from the collection ‘Reviews’ and displayed.

```python
# Query for product having the ratings 1
query = {"rating": 1 }

# Fetch data based on the query
cursor = list(review_collection.find(query))
print("The number of products having the rating as 1: "+ str(len(cursor)))

#cursor = review_collection.find(query)
# Print the results
#for user_data in cursor:
#    print(user_data)
```

✓ 0.2s Pythor

```
The number of products having the rating as 1: 19
```

Example: 4

Here, the inventories having the warehouse location to be ‘Arizona’ (AZ) are fetched from the collection ‘Inventory’ and displayed along with the count value.

```
# Query for inventory in AZ
query = {"warehouse_location": 'AZ' }

# Fetch data based on the query
cursor = list(inventory_collection.find(query))
print("The number of products having the rating as 1: "+ str(len(cursor)))

cursor = inventory_collection.find(query)
# Print the results
for user_data in cursor:
    print(user_data)
```

✓ 0.2s Python

```
The number of products having the rating as 1: 3
{'_id': ObjectId('655260597f69c7083d39f7ea'), 'product_id': 749, 'inventory_id': 75, 'warehouse_location': 'AZ', 'stock_quantity': 721,
'created_at': '6/20/2018 13:07:23'}
{'_id': ObjectId('655260597f69c7083d39f816'), 'product_id': 550, 'inventory_id': 56, 'warehouse_location': 'AZ', 'stock_quantity': 499,
'created_at': '11/28/2021 01:55:00'}
{'_id': ObjectId('655260597f69c7083d39f81d'), 'product_id': 199, 'inventory_id': 60, 'warehouse_location': 'AZ', 'stock_quantity': 662,
'created_at': '1/26/2011 04:33:31'}
```

Update:

Example: 1

Here, the orders having the shipping_status to be ‘awaiting_pickup’ are updated to ‘in_transit’ and the count of updated documents is depicted in the image below, along with the code.

```
# Specify the shipping status to be awaiting_pickup
status_to_update = 'awaiting_pickup'

# Updating the value to 'in_transit'
new_status = 'in_transit'

# Update the price for the specified product
result = shipping_collection.update_many({"status": status_to_update}, {"$set": {"status": new_status }})

# Print the update result
print(f"Matched {result.matched_count} document(s) and modified {result.modified_count} document(s)")
```

✓ 0.2s Python

```
Matched 26 document(s) and modified 26 document(s)
```

Example: 2

Here, the user with id 100 wants to update their address to ‘541 E MAIN ST’ in the collection ‘Users’. One thing to note is that instead of update_many, we have update_one function as only 1 document will be modified.

```
# Update the address for a specific user
query = {"user_id": 100 }  # Replace with the actual user ID or any other unique identifier
new_address = {"$set": {"address": "541 E MAIN ST"}}

# Perform the update
result = users_collection.update_one(query, new_address)

# Print the update result
print(f"Matched {result.matched_count} document(s) and modified {result.modified_count} document(s)")
```

✓ 0.4s

```
Matched 1 document(s) and modified 1 document(s)
```

Example: 3

Here, the inventory having the id as 65 has the change in the stock value, so the value has to be updated. So, fetching the inventory with the matching id is updated to have the stock value to be 635 and we can see only 1 document is updated.

```
# Specify the inventory ID
id_to_update = 65

# Specifying the other value
new_stock = 635

# Update the price for the specified product
result = inventory_collection.update_one({"inventory_id": id_to_update}, {"$set": {"stock_quantity": new_stock }})

# Print the update result
print(f"Matched {result.matched_count} document(s) and modified {result.modified_count} document(s)")
```

✓ 0.2s

```
Matched 1 document(s) and modified 1 document(s)
```

Example: 4

In the products collection, due to the black Friday deals there is a discount coming up in the price, so for the specific product with id as 1, the new price is set and it is $600.00.

```
# Specify the product ID or any other unique identifier to identify the product
product_id_to_update = 1

# Specify the new price for the product
new_price = 600  # Replace with the new price

# Update the price for the specified product
result = products_collection.update_one({"product_id": product_id_to_update}, {"$set": {"product_price": new_price }})

# Print the update result
print(f"Matched {result.matched_count} document(s) and modified {result.modified_count} document(s)")
```

✓ 0.3s

```
Matched 1 document(s) and modified 1 document(s)
```

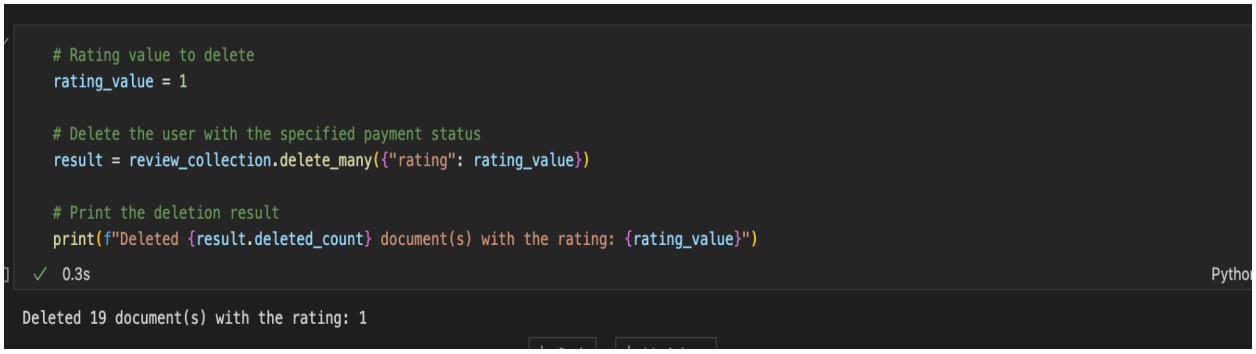
Delete:

Example: 1

Here, the transactions having the payment status as declined are deleted from the transactions collection and we can notice that 37 documents have been deleted.

Example: 2

Due to the rating 1 of certain products in the review collection, we decided to discontinue the products and hence, it needed to be deleted from the collections of reviews too.



Example: 3

Here, the product we decided to discontinue selling is ‘Pancetta’, hence it is deleted from the products collection.

From the above examples, it is clear that the NoSQL provides multiple functionalities like lightning fast queries, no strict schema and also the flexibility with which the developers can use it.