CET341 – ADVANCED DATA TECHNOLOGIES

PAKO BALEFI

239507561

ASSIGNMENT TWO

BOTSWANA ACCOUNTANCY COLLEGE (FRANCISTOWN)

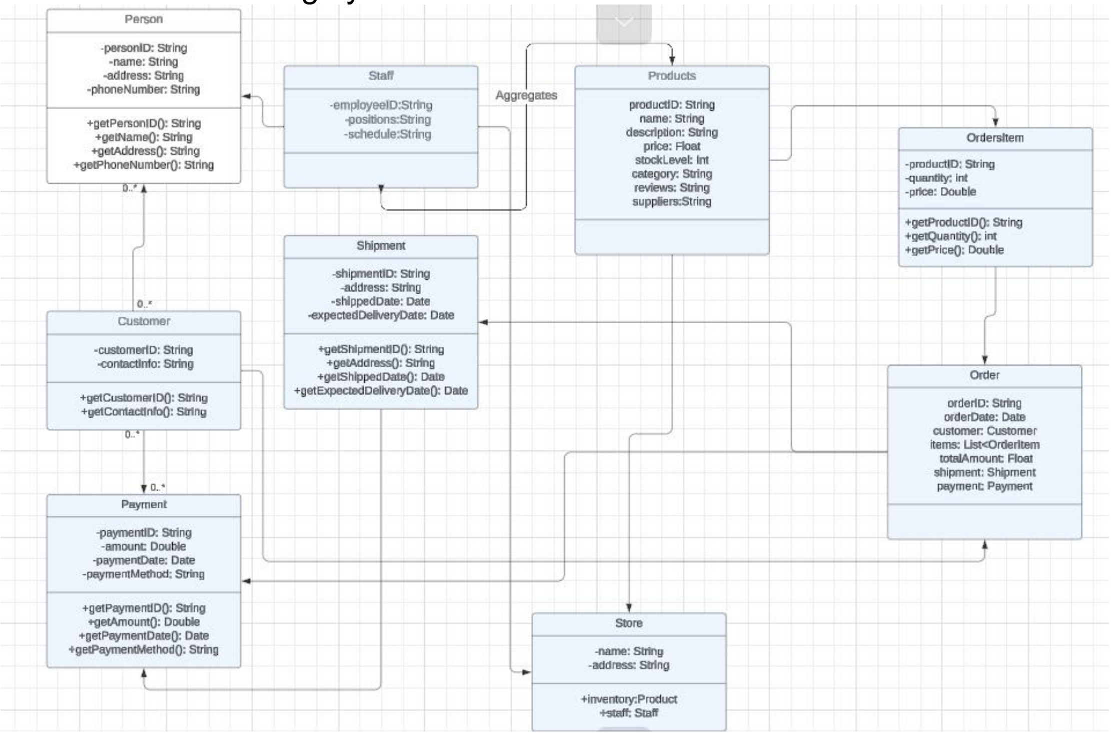
TASK 1

The existing online store system lacks a comprehensive and organized structure to manage its operations efficiently. There are several pain points identified, including disjointed data management, lack of scalability, and difficulty in maintaining data integrity. These challenges hinder the store's ability to effectively serve its customers, manage inventory, and streamline internal processes.

Key Issues:

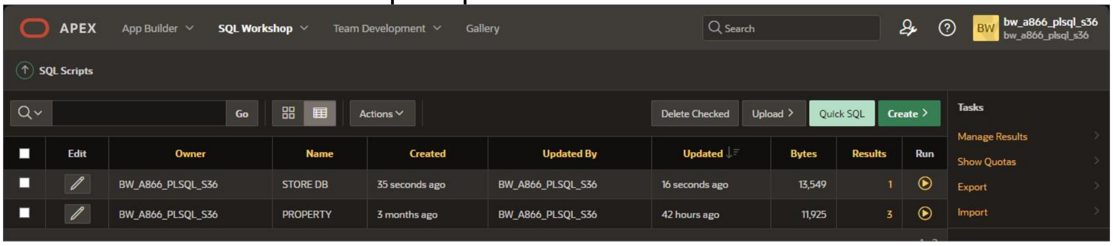
Disjointed Data Management: Currently, the store's data is scattered across various systems and databases, making it challenging to maintain consistency and accuracy. Product information, customer details, orders, and staff records are stored separately, leading to redundancy and data inconsistencies.
Limited Scalability: The current system struggles to accommodate growing data volumes and user traffic. As the store expands its product offerings and customer base, scalability becomes a critical concern. The existing architecture lacks the flexibility to scale horizontally or vertically to meet increasing demands.
Data Integrity Challenges: Ensuring data integrity is paramount for any online store system. However, the current setup lacks robust mechanisms to enforce data integrity rules effectively. There is a risk of data duplication, inconsistent records, and integrity violations, compromising the reliability of the system.

Proposed Solution:
To address these challenges, we propose redesigning the online store system with a robust and scalable architecture. The system will leverage modern database technologies to streamline data management, enhance scalability, and enforce data integrity.



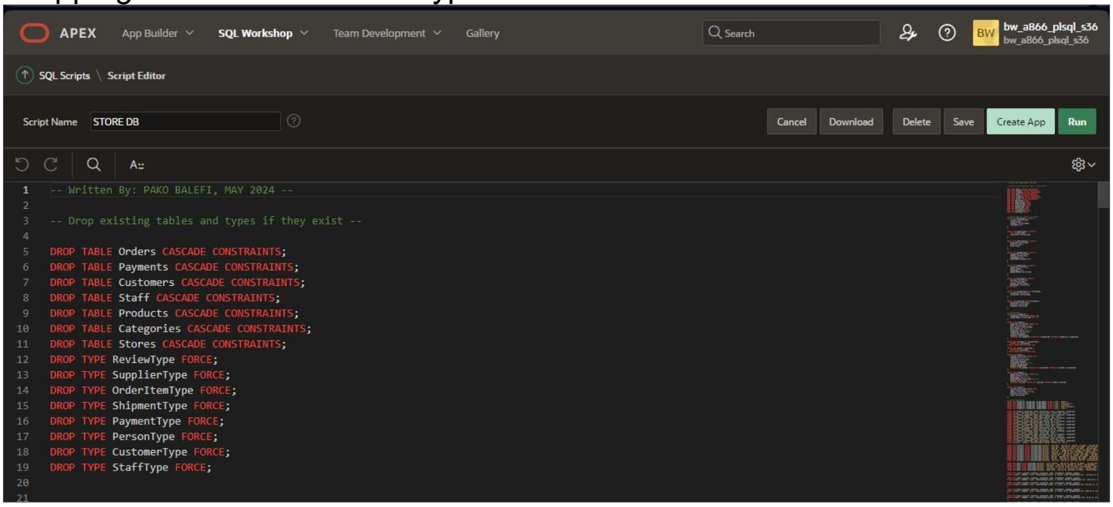
TASK 2

Creation of the database sql script named STORE DB



☐	Edit	Owner	Name	Created	Updated By	Updated	Bytes	Results	Run
☐	✎	BW_A866_PLSQL_S36	STORE DB	35 seconds ago	BW_A866_PLSQL_S36	16 seconds ago	13,549	1	▶
☐	✎	BW_A866_PLSQL_S36	PROPERTY	3 months ago	BW_A866_PLSQL_S36	42 hours ago	11,925	3	▶

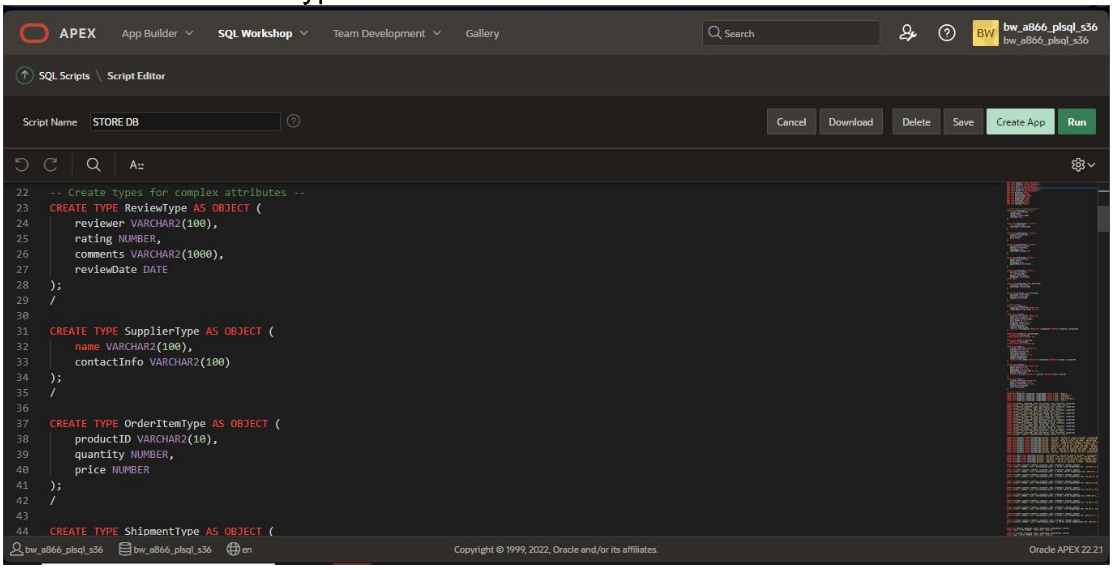
Dropping tables and the user types is as follows



Script Name: STORE DB

```
-- Written By: PAKO BALEFI, MAY 2024 --

-- Drop existing tables and types if they exist --

DROP TABLE Orders CASCADE CONSTRAINTS;
DROP TABLE Payments CASCADE CONSTRAINTS;
DROP TABLE Customers CASCADE CONSTRAINTS;
DROP TABLE Staff CASCADE CONSTRAINTS;
DROP TABLE Products CASCADE CONSTRAINTS;
DROP TABLE Categories CASCADE CONSTRAINTS;
DROP TABLE Stores CASCADE CONSTRAINTS;
DROP TYPE ReviewType FORCE;
DROP TYPE SupplierType FORCE;
DROP TYPE OrderItemType FORCE;
DROP TYPE ShipmentType FORCE;
DROP TYPE PaymentType FORCE;
DROP TYPE PersonType FORCE;
DROP TYPE CustomerType FORCE;
DROP TYPE StaffType FORCE;
```

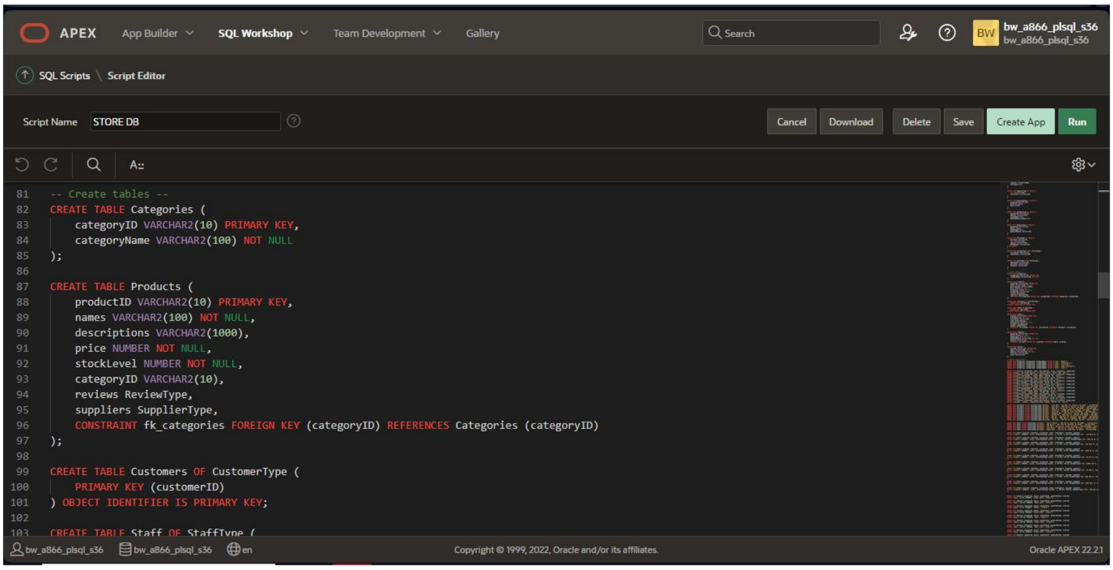
Creation of the user types is as follows



Script Name: STORE DB

```
-- Create types for complex attributes --
CREATE TYPE ReviewType AS OBJECT (
    reviewer VARCHAR2(100),
    rating NUMBER,
    comments VARCHAR2(1000),
    reviewDate DATE
);
/

CREATE TYPE SupplierType AS OBJECT (
    name VARCHAR2(100),
    contactInfo VARCHAR2(100)
);
/

CREATE TYPE OrderItemType AS OBJECT (
    productID VARCHAR2(10),
    quantity NUMBER,
    price NUMBER
);
/

CREATE TYPE ShipmentType AS OBJECT (
```

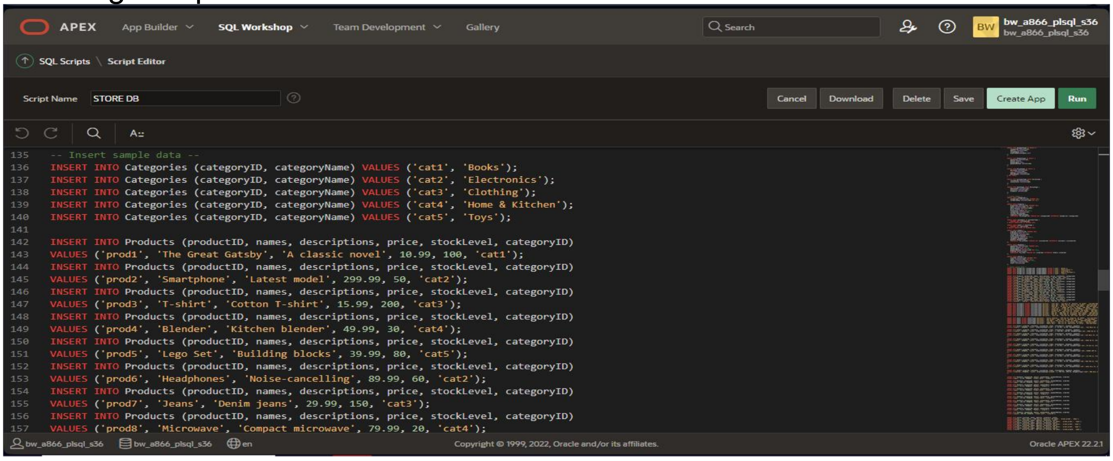
Creation of the tables is as follows



Script Name: STORE DB

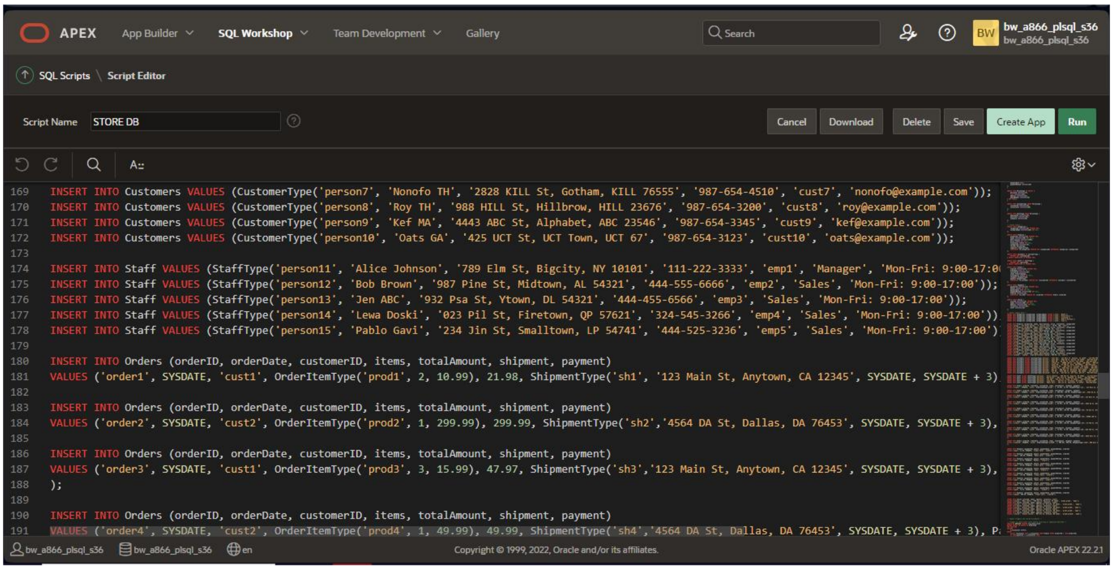
```sql
-- Create tables --
CREATE TABLE Categories (
    categoryID VARCHAR2(10) PRIMARY KEY,
    categoryName VARCHAR2(100) NOT NULL
);

CREATE TABLE Products (
    productID VARCHAR2(10) PRIMARY KEY,
    names VARCHAR2(100) NOT NULL,
    descriptions VARCHAR2(1000),
    price NUMBER NOT NULL,
    stockLevel NUMBER NOT NULL,
    categoryID VARCHAR2(10),
    reviews ReviewType,
    suppliers SupplierType,
    CONSTRAINT fk_categories FOREIGN KEY (categoryID) REFERENCES Categories (categoryID)
);

CREATE TABLE Customers OF CustomerType (
    PRIMARY KEY (customerID)
) OBJECT IDENTIFIER IS PRIMARY KEY;

CREATE TABLE Staff OF StaffType (
```

Inserting sample data into the tables



Script Name: STORE DB

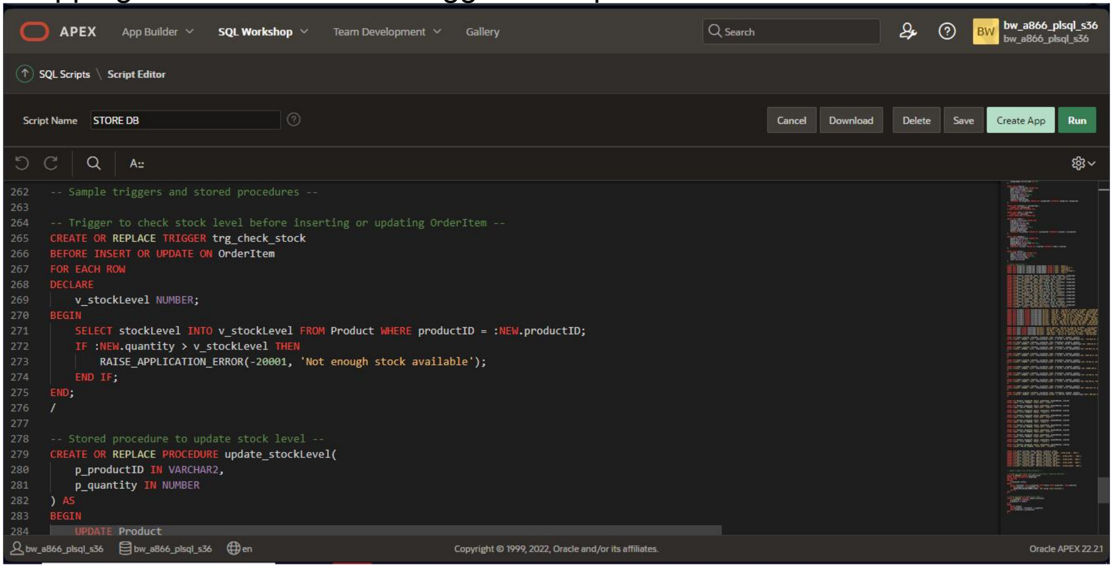
```sql
-- Insert sample data --
INSERT INTO Categories (categoryID, categoryName) VALUES ('cat1', 'Books');
INSERT INTO Categories (categoryID, categoryName) VALUES ('cat2', 'Electronics');
INSERT INTO Categories (categoryID, categoryName) VALUES ('cat3', 'Clothing');
INSERT INTO Categories (categoryID, categoryName) VALUES ('cat4', 'Home & Kitchen');
INSERT INTO Categories (categoryID, categoryName) VALUES ('cat5', 'Toys');

INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod1', 'The Great Gatsby', 'A classic novel', 10.99, 100, 'cat1');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod2', 'Smartphone', 'Latest model', 299.99, 50, 'cat2');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod3', 'T-shirt', 'Cotton T-shirt', 15.99, 200, 'cat3');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod4', 'Blender', 'Kitchen blender', 49.99, 30, 'cat4');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod5', 'Lego Set', 'Building blocks', 39.99, 80, 'cat5');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod6', 'Headphones', 'Noise-cancelling', 89.99, 60, 'cat2');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod7', 'Jeans', 'Denim jeans', 29.99, 150, 'cat3');
INSERT INTO Products (productID, names, descriptions, price, stockLevel, categoryID)
VALUES ('prod8', 'Microwave', 'Compact microwave', 79.99, 20, 'cat4');
```



Script Name: STORE DB

```sql
INSERT INTO Customers VALUES (CustomerType('person7', 'Nonofo TH', '2828 KILL St, Gotham, KILL 76555', '987-654-4510', 'cust7', 'nonofo@example.com'));
INSERT INTO Customers VALUES (CustomerType('person8', 'Roy TH', '988 HILL St, Hillbrow, HILL 23676', '987-654-3200', 'cust8', 'roy@example.com'));
INSERT INTO Customers VALUES (CustomerType('person9', 'Kef MA', '4443 ABC St, Alphabet, ABC 23546', '987-654-3345', 'cust9', 'kef@example.com'));
INSERT INTO Customers VALUES (CustomerType('person10', 'Oats GA', '425 UCT St, UCT Town, UCT 67', '987-654-3123', 'cust10', 'oats@example.com'));

INSERT INTO Staff VALUES (StaffType('person11', 'Alice Johnson', '789 Elm St, Bigcity, NY 10101', '111-222-3333', 'emp1', 'Manager', 'Mon-Fri: 9:00-17:0
INSERT INTO Staff VALUES (StaffType('person12', 'Bob Brown', '987 Pine St, Midtown, AL 54321', '444-555-6666', 'emp2', 'Sales', 'Mon-Fri: 9:00-17:00'));
INSERT INTO Staff VALUES (StaffType('person13', 'Jen ABC', '932 Psa St, Ytown, DL 54321', '444-455-6566', 'emp3', 'Sales', 'Mon-Fri: 9:00-17:00'));
INSERT INTO Staff VALUES (StaffType('person14', 'Lewa Doski', '023 Pil St, Firetown, QP 57621', '324-545-3266', 'emp4', 'Sales', 'Mon-Fri: 9:00-17:00'))
INSERT INTO Staff VALUES (StaffType('person15', 'Pablo Gavi', '234 Jin St, Smalltown, LP 54741', '444-525-3236', 'emp5', 'Sales', 'Mon-Fri: 9:00-17:00')

INSERT INTO Orders (orderID, orderDate, customerID, items, totalAmount, shipment, payment)
VALUES ('order1', SYSDATE, 'cust1', OrderItemType('prod1', 2, 10.99), 21.98, ShipmentType('sh1', '123 Main St, Anytown, CA 12345', SYSDATE, SYSDATE + 3)

INSERT INTO Orders (orderID, orderDate, customerID, items, totalAmount, shipment, payment)
VALUES ('order2', SYSDATE, 'cust2', OrderItemType('prod2', 1, 299.99), 299.99, ShipmentType('sh2','4564 DA St, Dallas, DA 76453', SYSDATE, SYSDATE + 3),

INSERT INTO Orders (orderID, orderDate, customerID, items, totalAmount, shipment, payment)
VALUES ('order3', SYSDATE, 'cust1', OrderItemType('prod3', 3, 15.99), 47.97, ShipmentType('sh3','123 Main St, Anytown, CA 12345', SYSDATE, SYSDATE + 3),
);

INSERT INTO Orders (orderID, orderDate, customerID, items, totalAmount, shipment, payment)
VALUES ('order4', SYSDATE, 'cust2', OrderItemType('prod4', 1, 49.99), 49.99, ShipmentType('sh4','4564 DA St, Dallas, DA 76453', SYSDATE, SYSDATE + 3), P
```

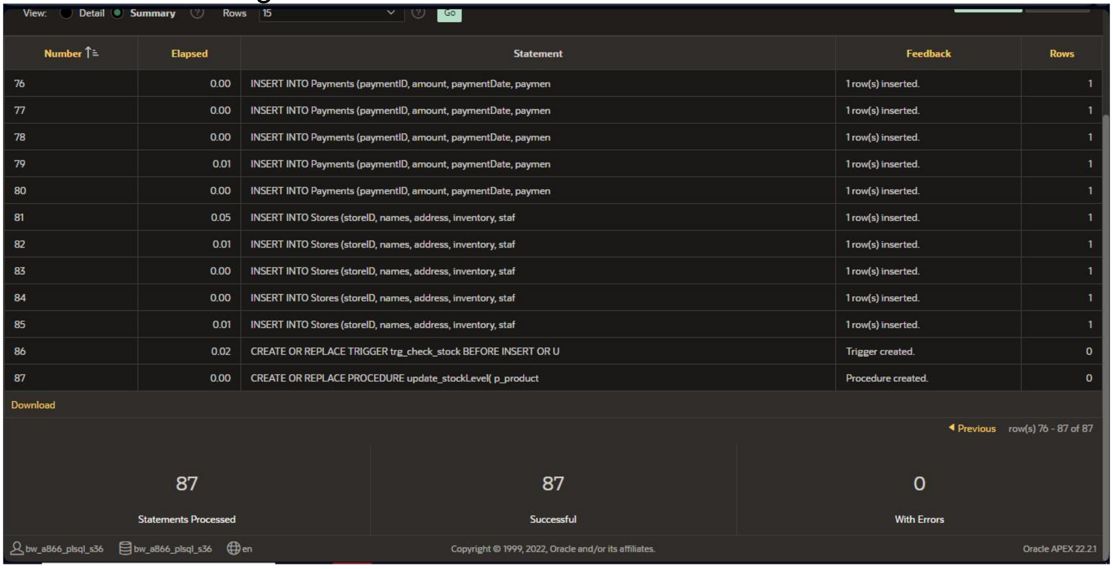
Dropping and creation of the triggers and procedure is as follows



Script Name: STORE DB

```
-- Sample triggers and stored procedures --

-- Trigger to check stock level before inserting or updating OrderItem --
CREATE OR REPLACE TRIGGER trg_check_stock
BEFORE INSERT OR UPDATE ON OrderItem
FOR EACH ROW
DECLARE
    v_stockLevel NUMBER;
BEGIN
    SELECT stockLevel INTO v_stockLevel FROM Product WHERE productID = :NEW.productID;
    IF :NEW.quantity > v_stockLevel THEN
        RAISE_APPLICATION_ERROR(-20001, 'Not enough stock available');
    END IF;
END;
/

-- Stored procedure to update stock level --
CREATE OR REPLACE PROCEDURE update_stockLevel(
    p_productID IN VARCHAR2,
    p_quantity IN NUMBER
) AS
BEGIN
    UPDATE Product
```

Showing that the database sql script is working without errors as the results are show on the figure below

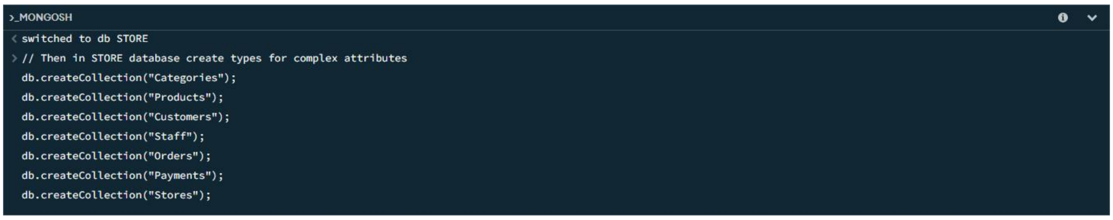


Number	Elapsed	Statement	Feedback	Rows
76	0.00	INSERT INTO Payments (paymentID, amount, paymentDate, paymen	1 row(s) inserted.	1
77	0.00	INSERT INTO Payments (paymentID, amount, paymentDate, paymen	1 row(s) inserted.	1
78	0.00	INSERT INTO Payments (paymentID, amount, paymentDate, paymen	1 row(s) inserted.	1
79	0.01	INSERT INTO Payments (paymentID, amount, paymentDate, paymen	1 row(s) inserted.	1
80	0.00	INSERT INTO Payments (paymentID, amount, paymentDate, paymen	1 row(s) inserted.	1
81	0.05	INSERT INTO Stores (storeID, names, address, inventory, staf	1 row(s) inserted.	1
82	0.01	INSERT INTO Stores (storeID, names, address, inventory, staf	1 row(s) inserted.	1
83	0.00	INSERT INTO Stores (storeID, names, address, inventory, staf	1 row(s) inserted.	1
84	0.00	INSERT INTO Stores (storeID, names, address, inventory, staf	1 row(s) inserted.	1
85	0.01	INSERT INTO Stores (storeID, names, address, inventory, staf	1 row(s) inserted.	1
86	0.02	CREATE OR REPLACE TRIGGER trg_check_stock BEFORE INSERT OR U	Trigger created.	0
87	0.00	CREATE OR REPLACE PROCEDURE update_stockLevel(p_product	Procedure created.	0

87	87	0
Statements Processed	Successful	With Errors

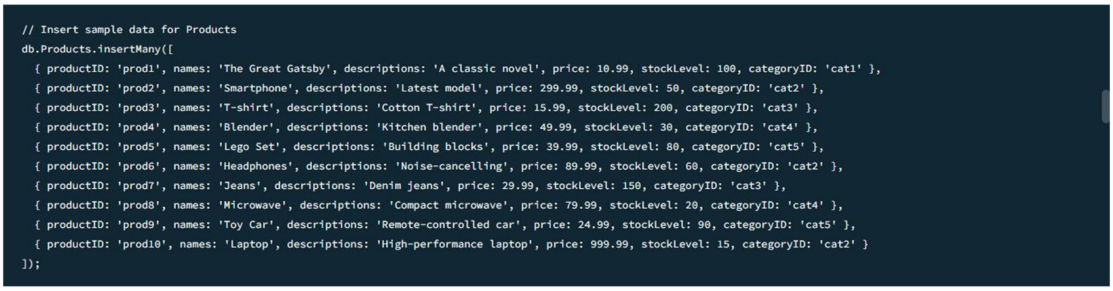
TASK 3

Creation of the database STORE and its collections using the mongo shell is as follows



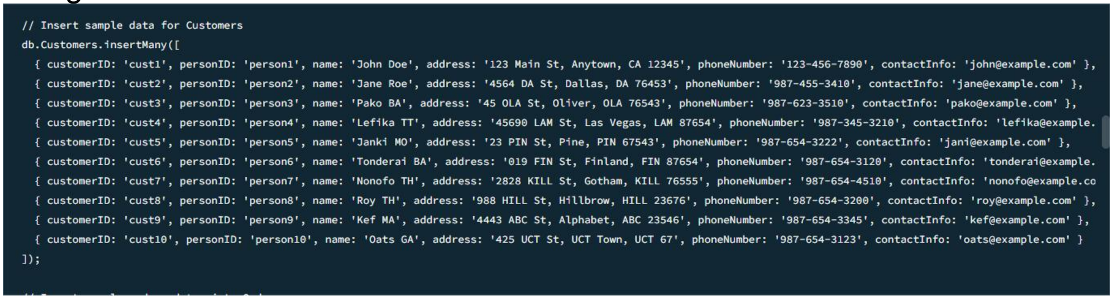
```
>_MONGOSH
< switched to db STORE
> // Then in STORE database create types for complex attributes
  db.createCollection("Categories");
  db.createCollection("Products");
  db.createCollection("Customers");
  db.createCollection("Staff");
  db.createCollection("Orders");
  db.createCollection("Payments");
  db.createCollection("Stores");
```

Inserting data in the Products collection that have been created using mongo shell is as follows



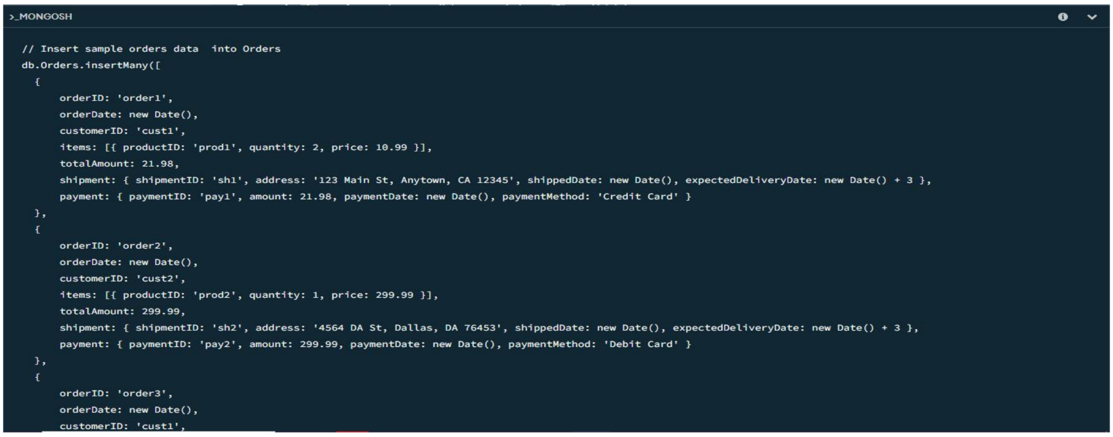
```
// Insert sample data for Products
db.Products.insertMany([
  { productID: 'prod1', names: 'The Great Gatsby', descriptions: 'A classic novel', price: 10.99, stockLevel: 100, categoryID: 'cat1' },
  { productID: 'prod2', names: 'Smartphone', descriptions: 'Latest model', price: 299.99, stockLevel: 50, categoryID: 'cat2' },
  { productID: 'prod3', names: 'T-shirt', descriptions: 'Cotton T-shirt', price: 15.99, stockLevel: 200, categoryID: 'cat3' },
  { productID: 'prod4', names: 'Blender', descriptions: 'Kitchen blender', price: 49.99, stockLevel: 30, categoryID: 'cat4' },
  { productID: 'prod5', names: 'Lego Set', descriptions: 'Building blocks', price: 39.99, stockLevel: 80, categoryID: 'cat5' },
  { productID: 'prod6', names: 'Headphones', descriptions: 'Noise-cancelling', price: 89.99, stockLevel: 60, categoryID: 'cat2' },
  { productID: 'prod7', names: 'Jeans', descriptions: 'Denim jeans', price: 29.99, stockLevel: 150, categoryID: 'cat3' },
  { productID: 'prod8', names: 'Microwave', descriptions: 'Compact microwave', price: 79.99, stockLevel: 20, categoryID: 'cat4' },
  { productID: 'prod9', names: 'Toy Car', descriptions: 'Remote-controlled car', price: 24.99, stockLevel: 90, categoryID: 'cat5' },
  { productID: 'prod10', names: 'Laptop', descriptions: 'High-performance laptop', price: 999.99, stockLevel: 15, categoryID: 'cat2' }
]);
```

Inserting data in the Customers collection that have been created using mongo shell is as follows



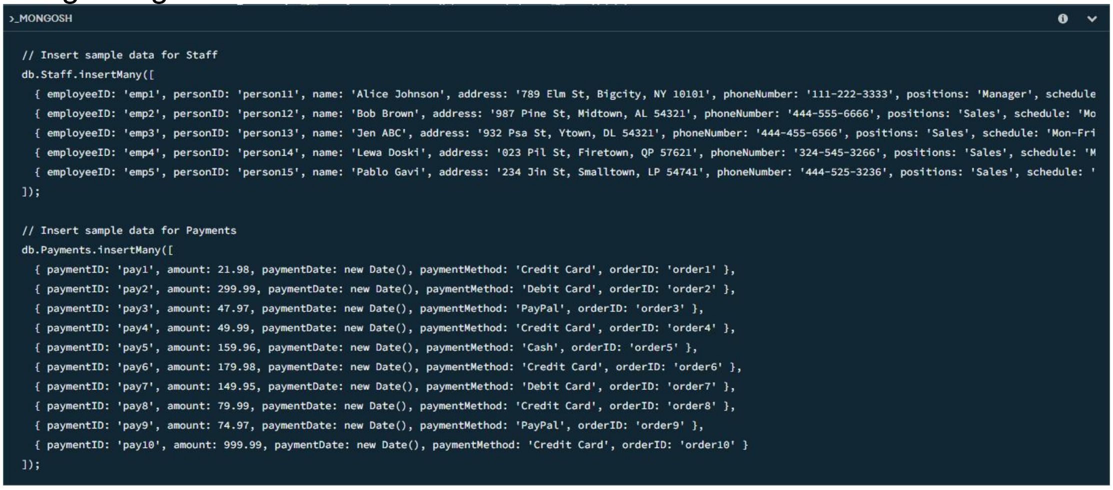
```
// Insert sample data for Customers
db.Customers.insertMany([
  { customerID: 'cust1', personID: 'person1', name: 'John Doe', address: '123 Main St, Anytown, CA 12345', phoneNumber: '123-456-7890', contactInfo: 'john@example.com' },
  { customerID: 'cust2', personID: 'person2', name: 'Jane Roe', address: '4564 DA St, Dallas, DA 76453', phoneNumber: '987-455-3410', contactInfo: 'jane@example.com' },
  { customerID: 'cust3', personID: 'person3', name: 'Pako BA', address: '45 OLA St, Oliver, OLA 76543', phoneNumber: '987-623-3510', contactInfo: 'pako@example.com' },
  { customerID: 'cust4', personID: 'person4', name: 'Lefika TT', address: '45690 LAM St, Las Vegas, LAM 87654', phoneNumber: '987-345-3210', contactInfo: 'lefika@example.
  { customerID: 'cust5', personID: 'person5', name: 'Janki MO', address: '23 PIN St, Pine, PIN 67543', phoneNumber: '987-654-3222', contactInfo: 'jani@example.com' },
  { customerID: 'cust6', personID: 'person6', name: 'Tonderai BA', address: '019 FIN St, Finland, FIN 87654', phoneNumber: '987-654-3120', contactInfo: 'tonderai@example.
  { customerID: 'cust7', personID: 'person7', name: 'Nonofo TH', address: '2828 KILL St, Gotham, KILL 76555', phoneNumber: '987-654-4510', contactInfo: 'nonofo@example.co
  { customerID: 'cust8', personID: 'person8', name: 'Roy TH', address: '988 HILL St, Hillbrow, HILL 23676', phoneNumber: '987-654-3200', contactInfo: 'roy@example.com' },
  { customerID: 'cust9', personID: 'person9', name: 'Kef MA', address: '4443 ABC St, Alphabet, ABC 23546', phoneNumber: '987-654-3345', contactInfo: 'kef@example.com' },
  { customerID: 'cust10', personID: 'person10', name: 'Oats GA', address: '425 UCT St, UCT Town, UCT 67', phoneNumber: '987-654-3123', contactInfo: 'oats@example.com' }
]);
```

Inserting data in the Orders collection that have been created using mongo shell is as follows



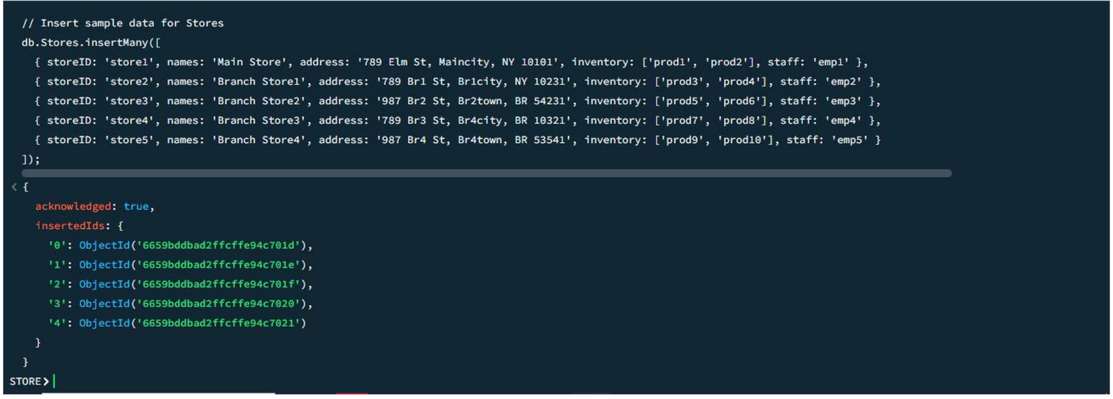
```
>_MONGOSH

  // Insert sample orders data  into Orders
  db.Orders.insertMany([
    {
        orderID: 'order1',
        orderDate: new Date(),
        customerID: 'cust1',
        items: [{ productID: 'prod1', quantity: 2, price: 10.99 }],
        totalAmount: 21.98,
        shipment: { shipmentID: 'sh1', address: '123 Main St, Anytown, CA 12345', shippedDate: new Date(), expectedDeliveryDate: new Date() + 3 },
        payment: { paymentID: 'pay1', amount: 21.98, paymentDate: new Date(), paymentMethod: 'Credit Card' }
    },
    {
        orderID: 'order2',
        orderDate: new Date(),
        customerID: 'cust2',
        items: [{ productID: 'prod2', quantity: 1, price: 299.99 }],
        totalAmount: 299.99,
        shipment: { shipmentID: 'sh2', address: '4564 DA St, Dallas, DA 76453', shippedDate: new Date(), expectedDeliveryDate: new Date() + 3 },
        payment: { paymentID: 'pay2', amount: 299.99, paymentDate: new Date(), paymentMethod: 'Debit Card' }
    },
    {
        orderID: 'order3',
        orderDate: new Date(),
        customerID: 'cust1',
```

Inserting data in the Staff and Payments collection that have been created using mongo shell is as follows



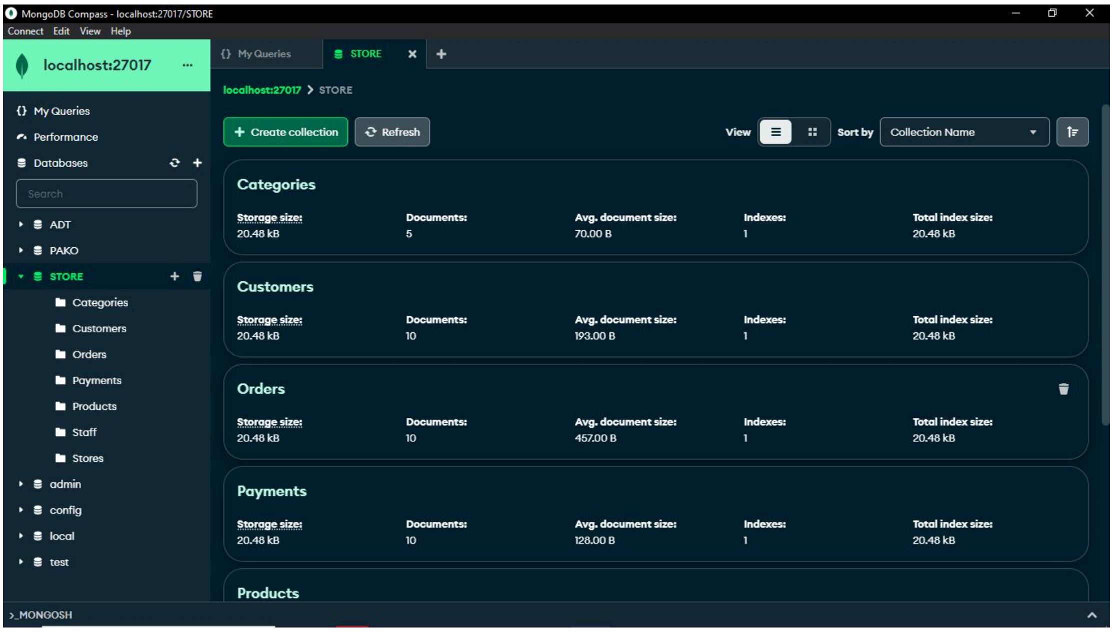
```
// Insert sample data for Staff
db.Staff.insertMany([
  { employeeID: 'emp1', personID: 'person11', name: 'Alice Johnson', address: '789 Elm St, Bigcity, NY 10101', phoneNumber: '111-222-3333', positions: 'Manager', schedule
  { employeeID: 'emp2', personID: 'person12', name: 'Bob Brown', address: '987 Pine St, Midtown, AL 54321', phoneNumber: '444-555-6666', positions: 'Sales', schedule: 'Mo
  { employeeID: 'emp3', personID: 'person13', name: 'Jen ABC', address: '932 Psa St, Ytown, DL 54321', phoneNumber: '444-455-6566', positions: 'Sales', schedule: 'Mon-Fri
  { employeeID: 'emp4', personID: 'person14', name: 'Lewa Doski', address: '023 Pil St, Firetown, QP 57621', phoneNumber: '324-545-3266', positions: 'Sales', schedule: 'M
  { employeeID: 'emp5', personID: 'person15', name: 'Pablo Gavi', address: '234 Jin St, Smalltown, LP 54741', phoneNumber: '444-525-3236', positions: 'Sales', schedule: '
]);

// Insert sample data for Payments
db.Payments.insertMany([
  { paymentID: 'pay1', amount: 21.98, paymentDate: new Date(), paymentMethod: 'Credit Card', orderID: 'order1' },
  { paymentID: 'pay2', amount: 299.99, paymentDate: new Date(), paymentMethod: 'Debit Card', orderID: 'order2' },
  { paymentID: 'pay3', amount: 47.97, paymentDate: new Date(), paymentMethod: 'PayPal', orderID: 'order3' },
  { paymentID: 'pay4', amount: 49.99, paymentDate: new Date(), paymentMethod: 'Credit Card', orderID: 'order4' },
  { paymentID: 'pay5', amount: 159.96, paymentDate: new Date(), paymentMethod: 'Cash', orderID: 'order5' },
  { paymentID: 'pay6', amount: 179.98, paymentDate: new Date(), paymentMethod: 'Credit Card', orderID: 'order6' },
  { paymentID: 'pay7', amount: 149.95, paymentDate: new Date(), paymentMethod: 'Debit Card', orderID: 'order7' },
  { paymentID: 'pay8', amount: 79.99, paymentDate: new Date(), paymentMethod: 'Credit Card', orderID: 'order8' },
  { paymentID: 'pay9', amount: 74.97, paymentDate: new Date(), paymentMethod: 'PayPal', orderID: 'order9' },
  { paymentID: 'pay10', amount: 999.99, paymentDate: new Date(), paymentMethod: 'Credit Card', orderID: 'order10' }
]);
```

Inserting data in the Stores collection that have been created using mongo shell is as follows, and we can see the output below that all collections have been created and the sample data has been inserted in the collections.



```
// Insert sample data for Stores
db.Stores.insertMany([
  { storeID: 'store1', names: 'Main Store', address: '789 Elm St, Maincity, NY 10101', inventory: ['prod1', 'prod2'], staff: 'emp1' },
  { storeID: 'store2', names: 'Branch Store1', address: '789 Br1 St, Br1city, NY 10231', inventory: ['prod3', 'prod4'], staff: 'emp2' },
  { storeID: 'store3', names: 'Branch Store2', address: '987 Br2 St, Br2town, BR 54231', inventory: ['prod5', 'prod6'], staff: 'emp3' },
  { storeID: 'store4', names: 'Branch Store3', address: '789 Br3 St, Br4city, BR 10321', inventory: ['prod7', 'prod8'], staff: 'emp4' },
  { storeID: 'store5', names: 'Branch Store4', address: '987 Br4 St, Br4town, BR 53541', inventory: ['prod9', 'prod10'], staff: 'emp5' }
]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('6659bddbad2ffcffe94c701d'),
      '1': ObjectId('6659bddbad2ffcffe94c701e'),
      '2': ObjectId('6659bddbad2ffcffe94c701f'),
      '3': ObjectId('6659bddbad2ffcffe94c7020'),
      '4': ObjectId('6659bddbad2ffcffe94c7021')
    }
  }
STORE >
```

Interface of the created database of Store is as follows with all its collections



Categories
Storage size: 20.48 kB
Documents: 5
Avg. document size: 70.00 B
Indexes: 1
Total index size: 20.48 kB

Customers
Storage size: 20.48 kB
Documents: 10
Avg. document size: 193.00 B
Indexes: 1
Total index size: 20.48 kB

Orders
Storage size: 20.48 kB
Documents: 10
Avg. document size: 457.00 B
Indexes: 1
Total index size: 20.48 kB

Payments
Storage size: 20.48 kB
Documents: 10
Avg. document size: 128.00 B
Indexes: 1
Total index size: 20.48 kB

Products

TASK 4

In the Oracle implementation of the online store system, object features were incorporated through the use of object types and user-defined types (UDTs) to model complex entities such as customers, orders, and products. For example, the `CustomerType` UDT encapsulates attributes related to a customer, including personal information and contact details. Similarly, the `OrderItemType` UDT represents individual items within an order, containing information such as product ID, quantity, and price. These UDTs enhance data modeling by allowing for structured storage and retrieval of information.

Additionally, database integrity rules were enforced using constraints such as primary keys, foreign keys, and check constraints. For instance, primary keys were defined to ensure uniqueness within tables, while foreign keys maintained referential integrity between related tables. This ensures data consistency and accuracy, preventing inconsistencies or errors in the database.

In the MongoDB implementation, document store features were leveraged to model the data as flexible, schema-less documents. Complex entities were represented as nested documents or arrays within documents. For example, a store document contains nested arrays for inventory and staff, allowing for easy representation of hierarchical data structures. This document-oriented approach provides flexibility in data representation, accommodating varying structures and fields across documents.

Furthermore, MongoDB's document store features such as indexes, sharding, and replication enhance scalability and performance (Selvaraj et al., 2020). Indexes can be created to optimize query performance, while sharding allows for horizontal scaling across multiple nodes (Sim et al., 2020). Replication ensures high availability and fault tolerance by maintaining multiple copies of data across nodes.

Comparing the two implementations, Oracle's relational database excels in enforcing structured data models and complex relationships through its support for transactions, ACID compliance, and robust query capabilities. It is well-suited for scenarios where data integrity and consistency are paramount, such as financial transactions or enterprise applications. Additionally, Oracle's support for SQL provides a standardized language for data manipulation and retrieval, facilitating ease of development and integration with existing systems.

On the other hand, MongoDB's document-oriented approach offers flexibility and scalability, making it suitable for scenarios with evolving or unstructured data requirements. Its distributed architecture and horizontal scalability make it ideal for handling large volumes of data and high throughput workloads, such as real-time analytics or content management systems. MongoDB's JSON-like document model also simplifies development by eliminating the need for complex joins or schema migrations, allowing for agile development and iteration.

Ultimately, the choice between Oracle and MongoDB depends on the specific requirements and constraints of the online store system. If the system prioritizes data integrity, transactional consistency, and complex relationships, Oracle's relational database may be more appropriate. However, if the system values flexibility, scalability, and rapid development, MongoDB's document store may offer better alignment with the project's goals (Giamas , 2022). It is essential to carefully evaluate factors such as data structure, scalability requirements, and development complexity before making a decision.

TASK 5 AND 6

Query a: A join of three or more tables – you should consider various types of join in this query (e.g. inner join, left/right/full outer joins, etc.) and the query must include a restriction on the rows selected

These queries will retrieve orders made by the customer named "John Doe" along with the corresponding product names, using inner join, left join, and full outer join operations in both SQL and MongoDB.

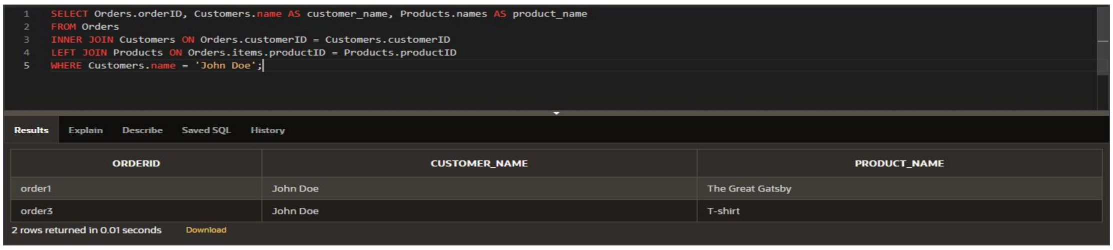
SQL code

```
SELECT Orders.orderID, Customers.name AS customer_name, Products.names AS product_name
FROM Orders
INNER JOIN Customers ON Orders.customerID = Customers.customerID
LEFT JOIN Products ON Orders.items.productID = Products.productID
WHERE Customers.name = 'John Doe';
```

MongoDB code

```
db.Orders.aggregate([ { $match: { "customerID": "cust1" } }, { $lookup: { from: "Customers", localField: "customerID", foreignField: "customerID", as: "customer" } }, { $unwind: "$customer" }, { $lookup: { from: "Products", localField: "items.productID", foreignField: "productID", as: "products" } }, { $unwind: { path: "$products", preserveNullAndEmptyArrays: true } }, { $project: { "_id": 0, "orderID": 1, "customer_name": "$customer.name", "product_name": "$products.names" } } ]);
```

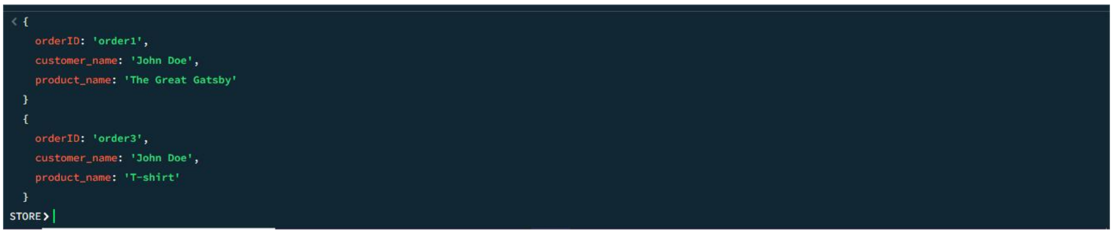
Screenshots



```
SELECT Orders.orderID, Customers.name AS customer_name, Products.names AS product_name
FROM Orders
INNER JOIN Customers ON Orders.customerID = Customers.customerID
LEFT JOIN Products ON Orders.items.productID = Products.productID
WHERE Customers.name = 'John Doe';
```

ORDERID	CUSTOMER_NAME	PRODUCT_NAME
order1	John Doe	The Great Gatsby
order3	John Doe	T-shirt

2 rows returned in 0.01 seconds



```
< {
    orderID: 'order1',
    customer_name: 'John Doe',
    product_name: 'The Great Gatsby'
  }
  {
    orderID: 'order3',
    customer_name: 'John Doe',
    product_name: 'T-shirt'
  }
STORE >
```

Query b: A query which uses one (or more) of the UNION, DIFFERENCE or INTERSECT operators.

This query retrieves a unified list of items, where each item can be either a product from the Products table or a customer from the Customers table. This query provides a unified view of both products and customers, allowing you to analyze them together if needed.
Combines the results of the two SELECT statements into a single result set.

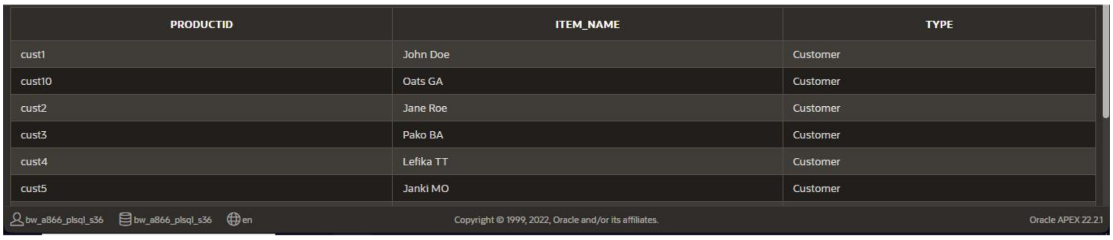
SQL code

```
SELECT productID, names AS
item_name, 'Product' AS type
FROM Products
UNION
SELECT customerID, name AS
item_name, 'Customer' AS type
FROM Customers;
```

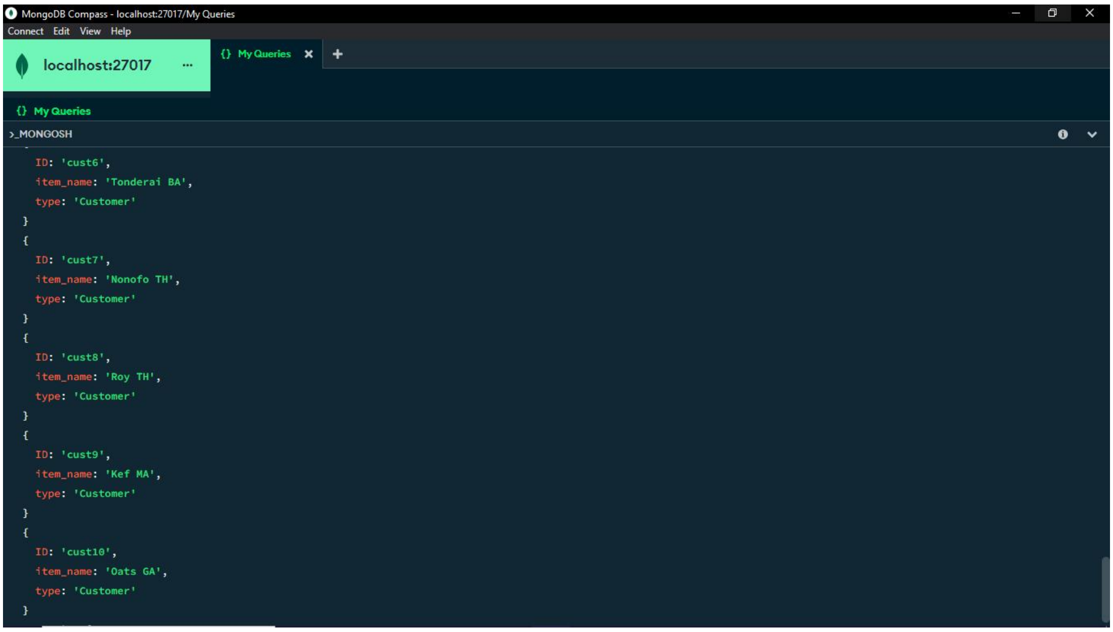
MongoDB code

```
db.Products.aggregate([ { $project:
{ "_id": 0, "ID": "$productID",
"item_name": "$names", "type":
"Product" } }, { $unionWith: { coll:
"Customers", pipeline: [ { $project:
{ "_id": 0, "ID": "$customerID",
"item_name": "$name", "type":
"Customer" } } ] } } ]);
```

Screenshots



PRODUCTID	ITEM_NAME	TYPE
cust1	John Doe	Customer
cust10	Oats GA	Customer
cust2	Jane Roe	Customer
cust3	Pako BA	Customer
cust4	Lefika TT	Customer
cust5	Janki MO	Customer



```
    ID: 'cust6',
    item_name: 'Tonderai BA',
    type: 'Customer'
  }
  {
    ID: 'cust7',
    item_name: 'Nonofo TH',
    type: 'Customer'
  }
  {
    ID: 'cust8',
    item_name: 'Roy TH',
    type: 'Customer'
  }
  {
    ID: 'cust9',
    item_name: 'Kef MA',
    type: 'Customer'
  }
  {
    ID: 'cust10',
    item_name: 'Oats GA',
    type: 'Customer'
  }
```

Query c: A query which requires use of either a nested table or subtypes

We select customerID, name from the Customers table (aliased as customer_name), orderID, and totalAmount from the Orders table.
We join the Customers table with the Orders table using the customerID column as the join condition. This retrieves orders for each customer.
Both queries achieve the same result, fetching details of customers along with their orders. The MongoDB aggregation query utilizes the $lookup stage to perform a similar operation to SQL's JOIN.

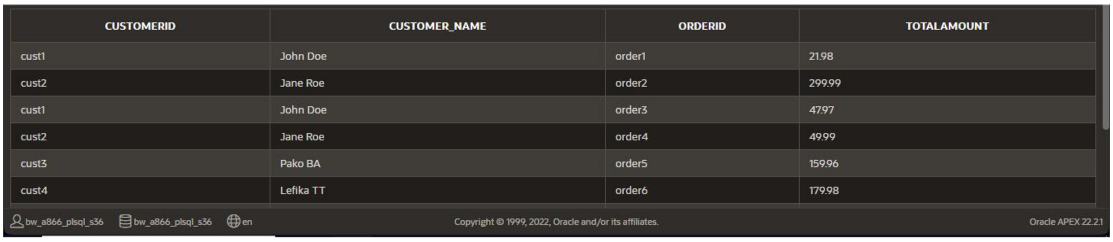
SQL code

```
SELECT
   Customers.customerID,
   Customers.name AS
customer_name,
   Orders.orderID,
   Orders.totalAmount
FROM
   Customers
JOIN
   Orders ON Customers.customerID
= Orders.customerID;
```

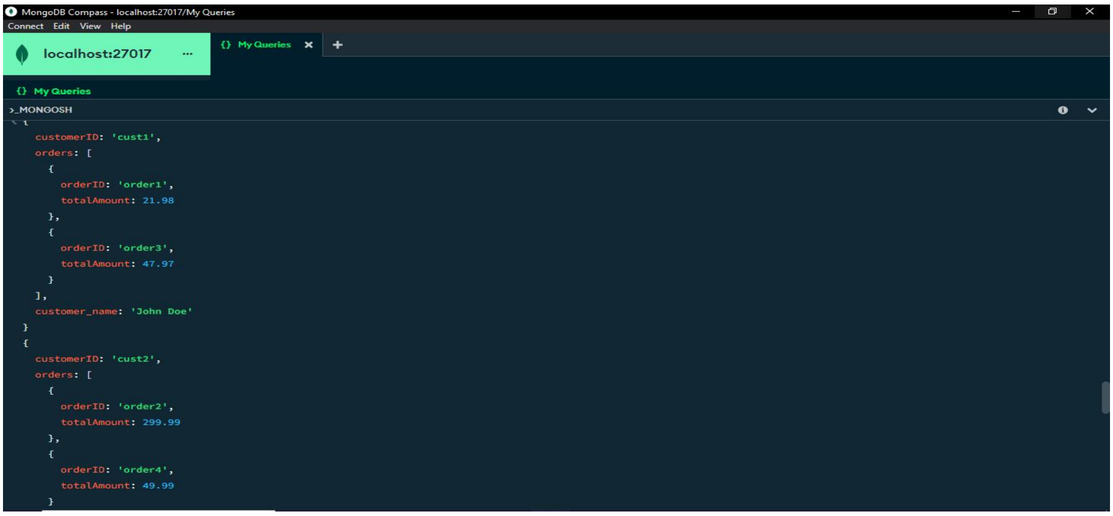
MongoDB code

```
db.Customers.aggregate([ { $lookup:
{ from: "Orders", localField:
"customerID", foreignField:
"customerID", as: "orders" } },
{ $project: { "_id": 0, "customerID": 1,
"customer_name": "$name",
"orders.orderID": 1,
"orders.totalAmount": 1 } } ]);
```

Screenshots



CUSTOMERID	CUSTOMER_NAME	ORDERID	TOTALAMOUNT
cust1	John Doe	order1	21.98
cust2	Jane Roe	order2	299.99
cust1	John Doe	order3	47.97
cust2	Jane Roe	order4	49.99
cust3	Pako BA	order5	159.96
cust4	Lefika TT	order6	179.98



```
    customerID: 'cust1',
    orders: [
      {
        orderID: 'order1',
        totalAmount: 21.98
      },
      {
        orderID: 'order3',
        totalAmount: 47.97
      }
    ],
    customer_name: 'John Doe'
  }
  {
    customerID: 'cust2',
    orders: [
      {
        orderID: 'order2',
        totalAmount: 299.99
      },
      {
        orderID: 'order4',
        totalAmount: 49.99
      }
```

Query d: A query using temporal features (e.g., timestamps, intervals, etc.) of Oracle SQL

Suppose we want to retrieve orders that were placed within the last 7 days, but we want to display the order date along with the difference between the order date and the current timestamp.This query demonstrates the usage of temporal features like timestamps, intervals, and date arithmetic in to perform complex temporal operations.Calculates the difference between the current timestamp and the order date, giving us the number of days since the order was placed.

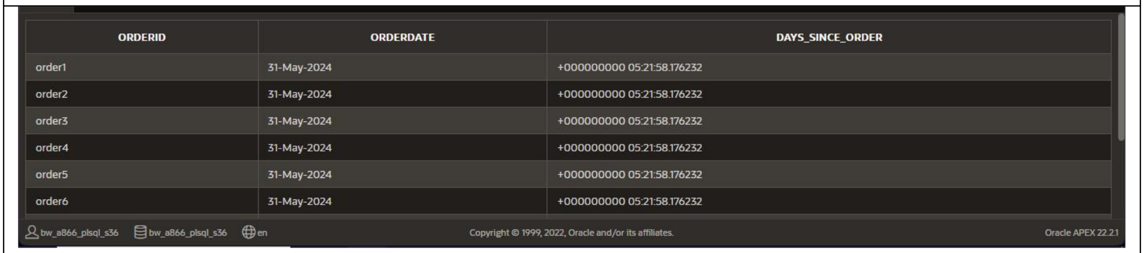
SQL code

```
SELECT
   orderID,
   orderDate,
   CURRENT_TIMESTAMP -
orderDate AS days_since_order
FROM
   Orders
WHERE
   orderDate >=
CURRENT_TIMESTAMP -
INTERVAL '7' DAY;
```

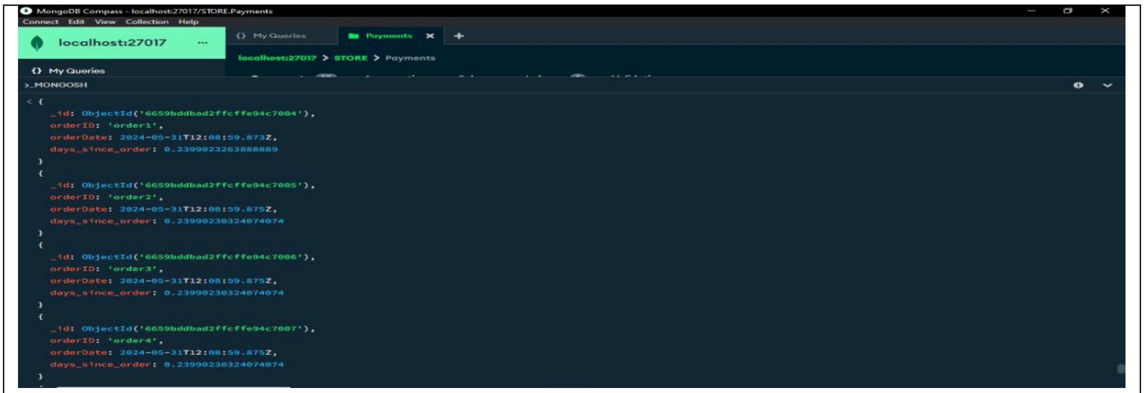
MongoDB code

```
db.Orders.aggregate([
  {
    $match: {
      orderDate: { $gte: new
Date(Date.now() - 7 * 24 * 60 * 60 *
1000) }
    }
  },
  {
    $project: {
      orderID: 1,
      orderDate: 1,
      days_since_order: {
        $divide: [
          { $subtract: [new Date(),
"$orderDate"] },
          1000 * 60 * 60 * 24
        ]
      }
    }
  }
]);
```

Screenshots



ORDERID	ORDERDATE	DAYS_SINCE_ORDER
order1	31-May-2024	+000000000 05:21:58.176232
order2	31-May-2024	+000000000 05:21:58.176232
order3	31-May-2024	+000000000 05:21:58.176232
order4	31-May-2024	+000000000 05:21:58.176232
order5	31-May-2024	+000000000 05:21:58.176232
order6	31-May-2024	+000000000 05:21:58.176232



MongoDB Compass - localhost:27017/STORE.Payments
Connect Edit View Collection Help
localhost:27017
My Queries
Payments
localhost:27017 > STORE > Payments
My Queries
>_MONGOSH

```
< {
    _id: ObjectId('6659bddbad2ffcffe94c7004'),
    orderID: 'order1',
    orderDate: 2024-05-31T12:08:59.873Z,
    days_since_order: 0.2399023263888889
  }
  {
    _id: ObjectId('6659bddbad2ffcffe94c7005'),
    orderID: 'order2',
    orderDate: 2024-05-31T12:08:59.875Z,
    days_since_order: 0.23990230324074074
  }
  {
    _id: ObjectId('6659bddbad2ffcffe94c7006'),
    orderID: 'order3',
    orderDate: 2024-05-31T12:08:59.875Z,
    days_since_order: 0.23990230324074074
  }
  {
    _id: ObjectId('6659bddbad2ffcffe94c7007'),
    orderID: 'order4',
    orderDate: 2024-05-31T12:08:59.875Z,
    days_since_order: 0.23990230324074074
  }
```

Query e: A query using OLAP (e.g., ROLLUP, CUBE, PARTITION) features of Oracle SQL

This query retrieves data from the Orders and Payments tables, calculating total orders and total payment amounts for each combination of customerID, orderDate, and paymentMethod.In MongoDB, we don't have direct support for the CUBE operator as in SQL. However, we can achieve similar results using the aggregation framework with multiple grouping stages

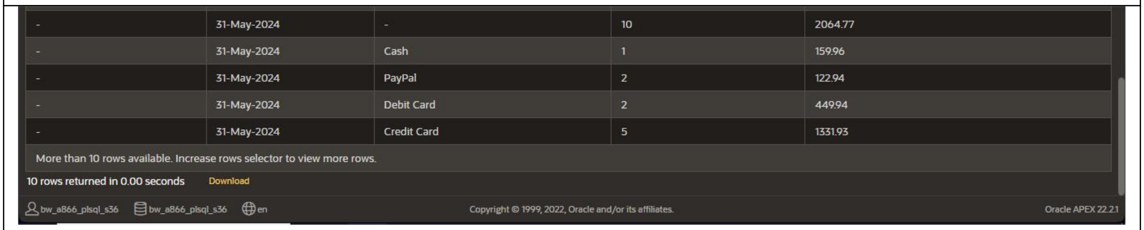
SQL code

```
SELECT
   o.customerID,
   TRUNC(o.orderDate) AS orderDate,
   p.paymentMethod,
   COUNT(o.orderID) AS totalOrders,
   SUM(p.amount) AS totalPaymentAmount
FROM
   Orders o
LEFT JOIN
   Payments p ON o.orderID = p.orderID
GROUP BY
   CUBE (o.customerID, TRUNC(o.orderDate), p.paymentMethod)
HAVING
   COUNT(o.orderID) > 0 OR SUM(p.amount) > 0;
```

MongoDB code

```
db.Orders.aggregate([
 {
  $lookup: {
   from: "Payments",
   localField: "orderID",
   foreignField: "orderID",
   as: "payments"
  }
 },
 {
  $unwind: {
   path: "$payments",
   preserveNullAndEmptyArrays: true
  }
 },
 {
  $group: {
   _id: {
    customerID: "$customerID",
    orderDate: { $dateToString: { format: "%Y-%m-%d", date: "$orderDate" } },
    paymentMethod: "$payments.paymentMethod"
   },
   totalOrders: { $sum: { $cond: [{ $ifNull: ["$orderID", false] }, 1, 0] } },
      totalPaymentAmount: { $sum: "$payments.amount" }
    }
  },
  {
    $match: {
      $or: [
        { "totalOrders": { $gt: 0 } },
        { "totalPaymentAmount": { $gt: 0 } }
      ]
    }
  },
  {
    $project: {
      customerID: "$_id.customerID",
      orderDate: "$_id.orderDate",
      paymentMethod: "$_id.paymentMethod",
      totalOrders: 1,
      totalPaymentAmount: 1,
      _id: 0
    }
  }
]);
```

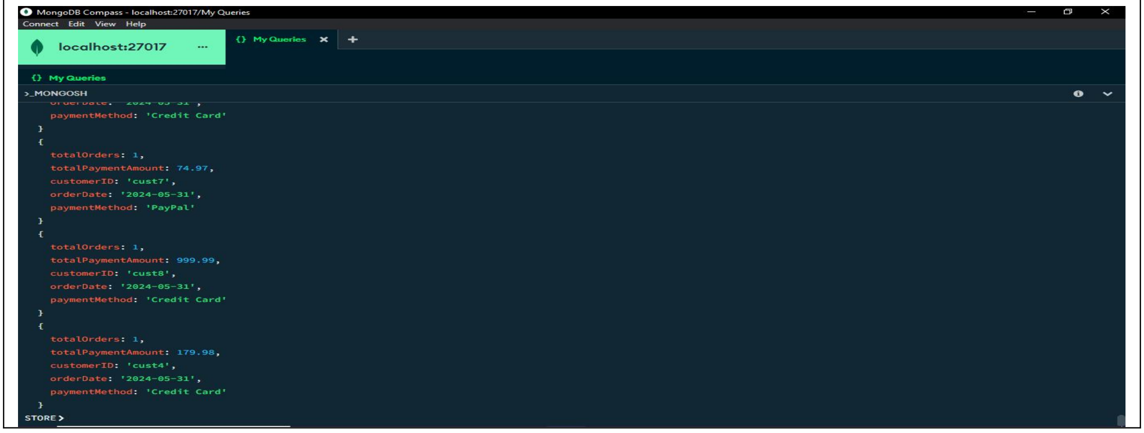
Screenshots



-	31-May-2024	-	10	2064.77
-	31-May-2024	Cash	1	159.96
-	31-May-2024	PayPal	2	122.94
-	31-May-2024	Debit Card	2	449.94
-	31-May-2024	Credit Card	5	1331.93

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds



```
    orderDate: '2024-05-31',
    paymentMethod: 'Credit Card'
  }
  {
    totalOrders: 1,
    totalPaymentAmount: 74.97,
    customerID: 'cust7',
    orderDate: '2024-05-31',
    paymentMethod: 'PayPal'
  }
  {
    totalOrders: 1,
    totalPaymentAmount: 999.99,
    customerID: 'cust8',
    orderDate: '2024-05-31',
    paymentMethod: 'Credit Card'
  }
  {
    totalOrders: 1,
    totalPaymentAmount: 179.98,
    customerID: 'cust4',
    orderDate: '2024-05-31',
    paymentMethod: 'Credit Card'
  }
```

REFERENCE

Selvaraj, P., Kannan, V. and Voisin, B., 2020. *Modified Data Storage and Replication Mechanism with Frequent Use-Case Based Indexing. Journal of Computational and Theoretical Nanoscience,* 17(12), pp.5229-5237.

Sim, H., Khan, A., Vazhkudai, S.S., Lim, S.H., Butt, A.R. and Kim, Y., 2020. *An integrated indexing and search service for distributed file systems. IEEE Transactions on Parallel and Distributed Systems,* 31(10), pp.2375-2391.

Giamas, A., 2022. Mastering MongoDB 6. *x: Expert techniques to run high-volume and fault-tolerant database solutions using MongoDB 6. x.* Packt Publishing Ltd.