Gisma University
of Applied Sciences

Assessment Submission Form

Student Number (If this is group work, please include the student numbers of all group participants)	GH1027620
Assessment Title	E-Commerce Database
Module Code	M605A
Module Title	Advanced Database
Module Tutor	Dr. Mazhar Hameed
Date Submitted	25.03.2025

Declaration of Authorship

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

I fully understand that the unacknowledged inclusion of another person's writings or ideas or works in this work may be considered plagiarism and that, should a formal investigation process confirms the allegation, I would be subject to the penalties associated with plagiarism, as per GISMA Business School, University of Applied Sciences' regulations for academic misconduct.

Signed Parisa Khosravi Date 25.03.2025

Book Store Database System

Parisa Khosravi

Student No. GH1027620

Parisa.Khosravi@gisma-student.com

Abstract

The project discussed in this report is a book store database system using SQL and NoSQL technologies, allowing the business owner to manage and review order processes and analyze the products and sales. These goals are reached by creating functional queries that support CURD operations and more advanced queries including stored procedures specifically transactions and triggers in SQL that forms the order processing and logging, as well as managing the inventory and customer information. In the NoSQL part the product reviews are handled to give the management more flexibility with the data.

The Link to the Github repository

https://github.com/parisakh4/database

The link to the video demonstration

https://youtu.be/YNEN1Dv5ZJk?si=RvTxbVY_B8iyoNTA

Introduction

An online bookstore management like any other E-commerce platform needs a database system that allows for both operational tasks and business analysis. This project implements a system that uses both SQL and NoSQL technologies to meet these needs with the following objectives:

- Storing Customer Information including addresses, orders and transactions to maintain data consistency and for marketing analysis, using SQL.
- Order Processing and Tracking, using stored procedures and transactions in SQL to handle orders, and tracking the process using triggers to log the changes at every step.
- Inventory Management, by updating stock after orders are completed and restock after they are cancelled using procedures and transactions.
- Sales And Product Performance analysis with the use of queries to retrieve sales and product ratings in SQL, and analyzing product reviews in NoSQL.

In the following sections the system design and schema, implementation and project results and challenges are discussed.

System Design

The data in this system has been organized as a relational (SQL) and non-relational database (NoSQL). SQL database is used for managing structural data with a fixed schema to handle necessary operations and functionalities such as managing product information, inventory, customer data and handling orders. NoSQL database has been used to handle semi-structured data with a flexible schema which in this project is the user generated product reviews. This allows for easier management of evolving data models which can also grow in scale. The admin user can incorporate new fields, retrieve and update data more efficiently.

SQL Database

The first step of the relational database creation was the design of ER diagram to get a clear picture of the SQL database; to identify the needed tables, the relations between the tables and their schema, the attributes, primary key and foreign key in each table and the data type of each column in tables.

As shown in the ER diagram below, the SQL database comprise of eight tables: Customers, Addresses, Suppliers, Products, Orders, Order Details, Transactions and Order Logs.

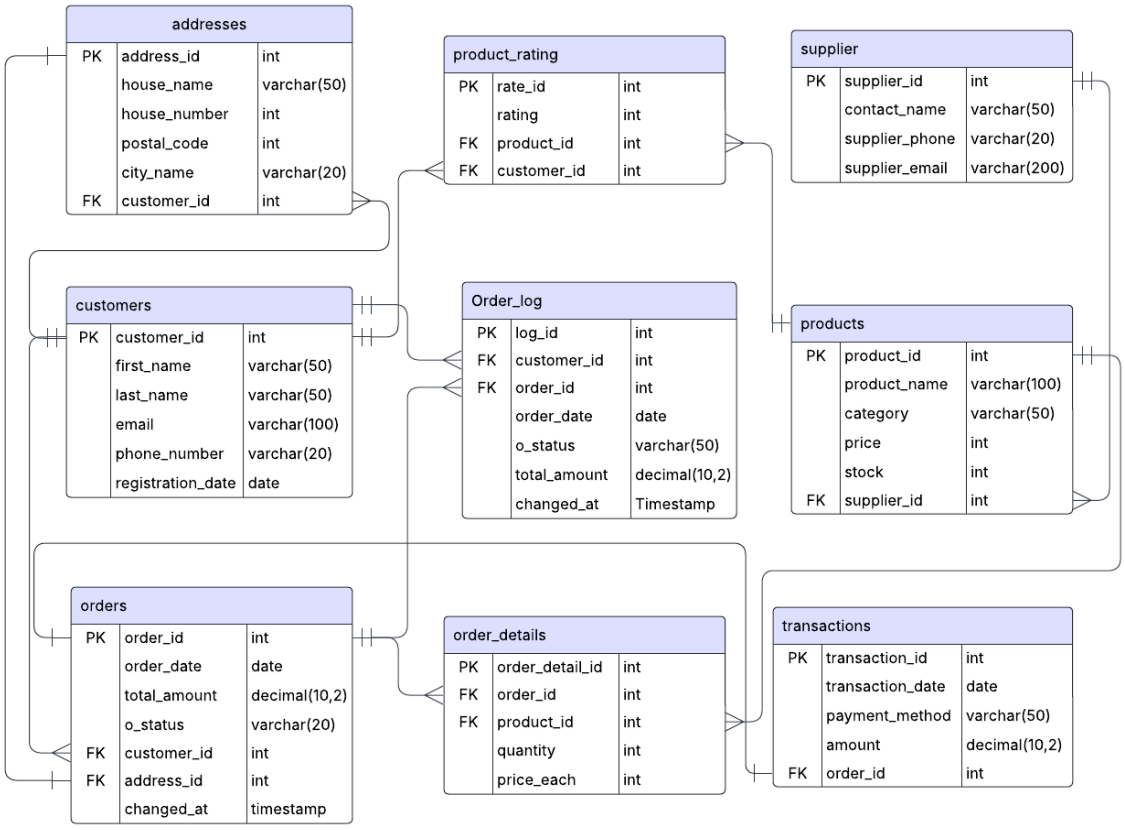

Figure 1. ER Diagram

As shown in the diagram, the customer information has a separate table from addresses. This allows the customer to have multiple addresses, indicating a one-to-many relation. The relation

between the addresses and orders table are one to one because each order can be delivered to only one address. Orders are in a table separate from the order log table, so that different changes in order status can be logged with their time for keeping record and tracking orders. Orders table is also in a one-to-many relationship with the order details table to keep track of each product and their quantity to manage the inventory. There is a transactions table in a one-to-one relation with orders, so that for each order there is a one-time payment. The relation between the customer and orders table is one-to-many because each customer can have multiple orders. There is a table for the suppliers in a one-to-many relation with the products table because books are purchased from different publications, and each publication can supply multiple books. Products table are in a one-to-many relation with the product rating because each product can have multiple ratings. Moreover, each customer can rate multiple products, leading to another one-to-many relation in the database.

NoSQL Database

The NoSQL data is used for product reviews. In this part, there is a review collection in which each document represents a product with an id, name and an embedded array of reviews with customer id and name, rating of product, the review and review date. This method was chosen to handle one to many relationships; each product can have multiple reviews. The review section is relatively small and frequently used together and not used across various collections, which makes embedding an appropriate solution.

Implementation

Following the system design, the implementation of the database had multiple stages. After creating the SQL database, the tables were created and populated via bulk inserts. Then Select, Update and Delete queries were used for carrying out simple CURD operations. In the next step different queries, Stored Procedures, Triggers, Transactions were created to handle more complex operations. In the end Test Queries were created to test these complex functions. All of the above are organized in separate files and are explained bellow in detail.

The queries for different selecting (Read) operations allow the admin to review, sort and analyze the information on the business. These operations include:

- Listing products based on ratings (Figure 2) and sale to identify popular and unpopular products and make correct decision for restocking them.

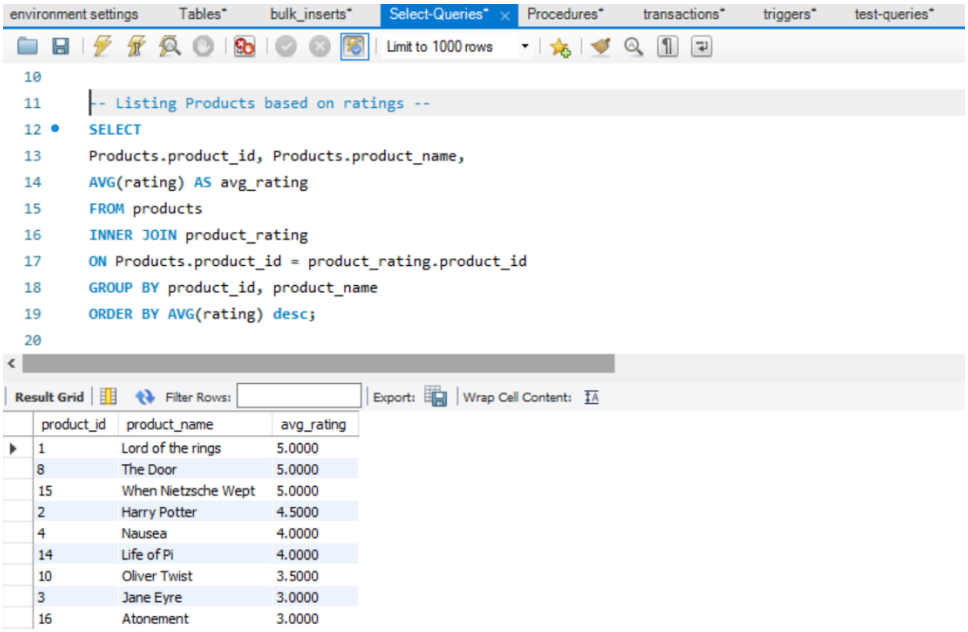

Figure 2. Listing products based on ratings

- Finding a products specific rating with the similar logic mentioned above.
- Listing customers with most transactions and customers with no transactions, to be able to give incentives like discounts to both groups, the first group for their loyalty and to keep them engaged with the business, and the second group to encourage them to make a purchase and preventing them from leaving.

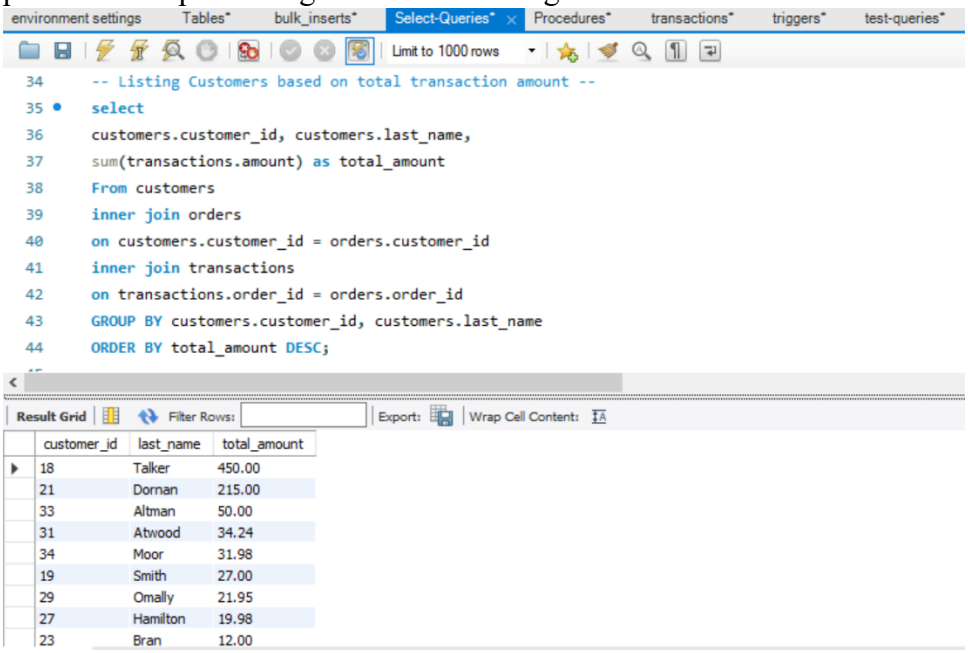

Figure 3. Listing customers based on transactions

- Listing customers with all their orders, ordered by the status of their order.

- Listing suppliers with and without their products to have a clear picture which supplier had the most interaction with the business.

```
60    -- Listing suppliers with and without their products --
61    select suppliers.supplier_id, suppliers.supplier_name, products.product_id, products.product_name
62    from suppliers
63    left join products
64    on suppliers.supplier_id = products.supplier_id;
65
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

supplier_id	supplier_name	product_id	product_name
1	Bloomsbury	13	Anne of Green Gables
1	Bloomsbury	16	Atonement
2	ML Press	10	Oliver Twist
3	Chef Press	8	The Door
3	Chef Press	11	War and Peace
4	Edmund Bollifant	6	American Psycho

Result 5

Figure 4. Listing suppliers

The update and delete operations include the following:

- Updating product table; allowing admin to update certain attribute such as price or stuck.

```
1    -- Update product stock and price--
2    UPDATE Products
3    SET stock = stock + 100, price = 60
4    WHERE product_id = 1;
```

Figure 5. Updating stock for a product

- Updating customer information; this operation is written as a simple query, for simplicity and keeping the options open to adjust the code based on the use case, for instance, once a customer email might change, but another time they address might change.
- Updating suppliers; which can be a simple query whenever there are changes in a supplier info.
- Deleting from customers, products and supplier tables

One important point to consider is deleting operation for the orders. In this project an order is cancelled by being deleted from order details, transactions and order table, which are done sequentially in a transaction to keep the atomicity and consistency and making the process easier. The process sets off a trigger to change to order status to “Cancelled” and log the changes.

```
97    -- Transaction for deleting an order and related info on tables --
98    DELIMITER //
99    CREATE PROCEDURE DeleteOrderAndRelatedData(
100       IN p_order_id INT
101   )
102   BEGIN
103       -- Exit handler to rollback if any error occurs
104       DECLARE EXIT HANDLER FOR SQLEXCEPTION
105       BEGIN
106           ROLLBACK;
107           SELECT 'Error occurred, transaction rolled back.' AS message;
108       END;
109
110       START TRANSACTION;
111
112       DELETE FROM Order_Details
113       WHERE order_id = p_order_id;
114
115       DELETE FROM Transactions
116       WHERE order_id = p_order_id;
117
118       DELETE FROM Orders
119       WHERE order_id = p_order_id;
120
121       COMMIT;
122       SELECT 'Deletion successful.' AS message;
123   END//
124   DELIMITER ;
```

Figure 6. Delete transaction

Next group of queries are the stored procedures:

- Getting the total sail in a certain time period, to be able to analyze the sales each month and make the right financial decisions.

```
1   -- total sale calculation in a time period
2   DELIMITER //
3   Create procedure GetSale(IN begin_date Date, IN end_date date)
4   Begin
5   select sum(transactions.amount) as total_sale
6   from transactions
7   where transaction_date between begin_date and end_date;
8   end //
9   DELIMITER ;
```

Figure 7. Getting total sale procedure

- Tracking the customers order based on customer and order id to be able to inform and communicate with the customers if necessary, and to take actions in case of problems with a specific order.
- Getting all of a customers order to gather information for suggesting products in the next phases of developing the e-commerce application on the user level.
- Restocking books; this operation already exists as an update query and the procedure is only added in case it was needed in future development.

```
33    -- restock books--
34    DELIMITER //
35    CREATE PROCEDURE Restock(IN p_id INT, IN p_restock INT)
36    BEGIN
37    START TRANSACTION;
38    UPDATE Products
39    SET stock = stock + p_restock
40    WHERE product_id = p_id;
41    COMMIT;
42    END //
43    DELIMITER ;
```

Figure 8. Restocking Procedure

Other procedures are written as transactions and triggers. The first three transactions consecutively with an application layer can form a complete order process. Each off these transactions pull a trigger that is stored in a separate file but explained here in relation to the transaction:

- **Create New Order**; which adds a new order to the orders table, setting off the trigger to log the order creation into the order_log table.

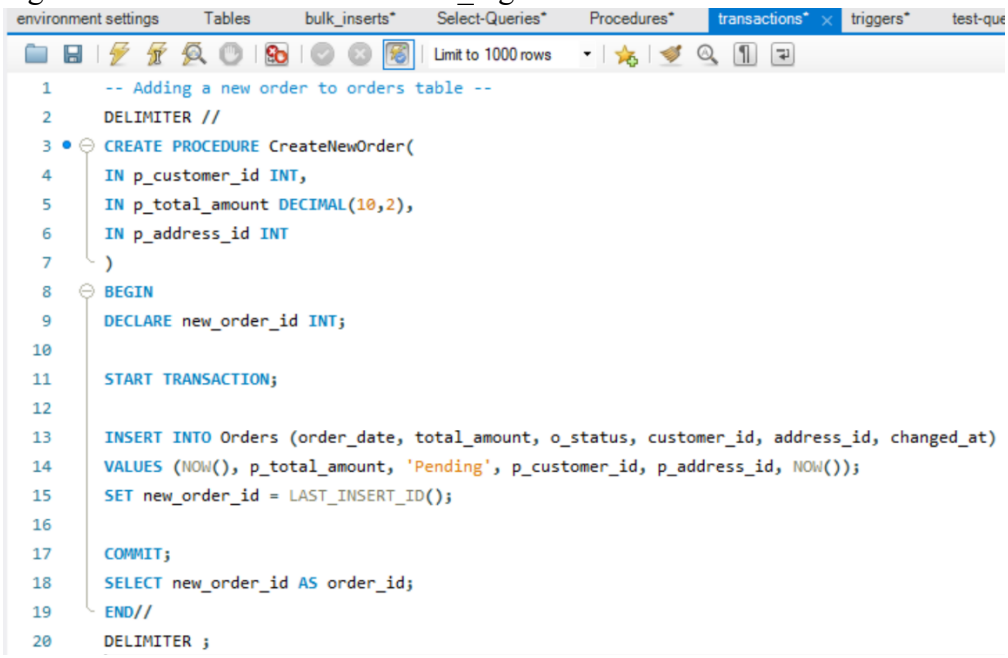

Figure 9. Create New Order Transaction

- **Handle Order**; which gets the order id from the previous transaction, checks the stock and then if the product is available, goes through with the order and then updates the stock. If there is not enough stock, the procedure rolls back and change the order status in the orders table to “Failed”, setting off another trigger that logs the status change of each order in the order_log table. In this transaction when checking the availability if the stock, the product row is locked to handle possible simultaneous orders of the same product.

```
START TRANSACTION;
-- Check stock availability
SELECT stock INTO available_stock
FROM Products
WHERE product_id = p_prod_id
FOR UPDATE; -- Lociking the product row to handle simultaneous order --

-- If insufficient stock, rollback and set the flag and change the order status to "Failed" --
IF available_stock < p_quant THEN
UPDATE Orders
SET o_status = 'Failed', changed_at = NOW()
WHERE order_id = p_ord_id;
ROLLBACK;
SET transaction_successful = FALSE;
SELECT CONCAT('Transaction failed: Insufficient stock. Available stock: ', available_stock) AS message;
END IF;

-- Only proceed if the flag indicates success
IF transaction_successful THEN
-- Deduct stock
UPDATE Products
SET stock = stock - p_quant WHERE product_id = p_prod_id;
-- Insert into Order_Details
INSERT INTO Order_Details (order_id, product_id, quantity, price_each)
VALUES (p_ord_id, p_prod_id, p_quant, p_price);
-- Commit the transaction
COMMIT;
```

Figure 10. Handle Order Transaction

- **Process Order Transaction**; which checks if the order exists and if it is in the pending state, gets the order id and payment information, then goes through and adds a new row into the Transactions table and changes the order status to “Completed”, which again sets off the trigger for the status change.

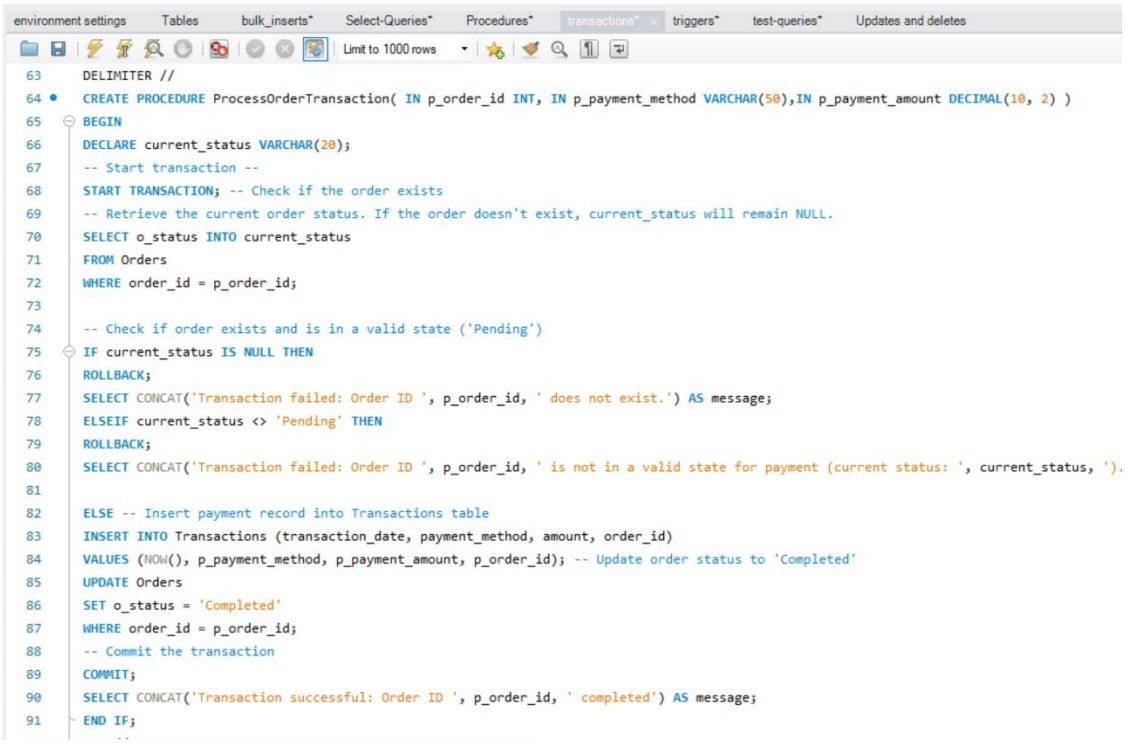

Figure 11. Process Order Transaction

- **Delete Order and Related Data**; this transaction was already explained in update and delete section. It also sets off the trigger for restocking the product table.

As an example, one of the triggers is shown below.

```
-- Log order cancellation after deleting an order --
DELIMITER //
CREATE TRIGGER log_order_cancellation
AFTER DELETE ON Orders
FOR EACH ROW
BEGIN
    INSERT INTO Order_Log (order_id, customer_id, order_date, o_status, total_amount, changed_at)
    VALUES (OLD.order_id, OLD.customer_id, OLD.order_date, 'Cancelled', OLD.total_amount, NOW());
END//
DELIMITER ;
```

Figure 12. Trigger Example

For the NoSQL part, the data is added using insertMany command, and then other CURD operation was carried out using update, delete, find and aggregate functions. These queries allow the manager to view the reviews for a specific product, filter them based on the existence of a certain word, edit the reviews, sort the products based on the number of reviews and more. The code examples with their results are showed in the result section of this paper.

Challenges and Solutions

One of the biggest challenges of this project was at the decision-making level, to decide whether to write an operation as a query or a stored procedure. Trying to keep the system clear and simple, for the cases that could have been handled by a query I wrote them as such, and for cases more complicated, which needed an input or had multiple and sequential operations I decided on procedures. In cases all the operations that needed to be done as a single unit, they were wrapped in a transaction layer, the function of which were explained in the previous section.

Another challenge was the late creation of the order log triggers that was designed to write the time of change into the order log table, while no matching attribute existed in the orders table. This challenge was result by altering the order table to add the new attribute “changed_at” with a default null value and then updating them after bulk insert based on order date.

Results

In this part the results of the project are explained. In the SQL database, other than simple queries, i.e. selecting all the rows from the tables, examples for the results of more complex operations are as follows:

- GetSale store procedure

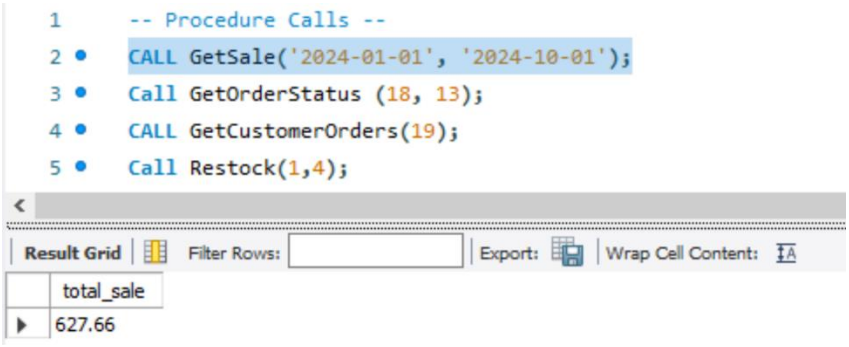

Figure 13. GetSale store procedure result

- GetOrderStatus procedure

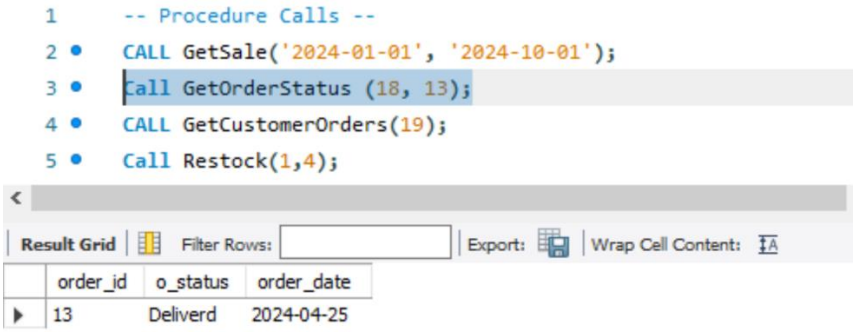

Figure 14. GetOrderStatus procedure result

- CreateNewOrder transaction

```
7    -- Transactions --
8 •  CALL CreateNewOrder(18, 100, 23);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id
27

Figure 15. CreateNewOrder transaction result

- Trigger Logging the New Order (row 3)

Result Grid | Filter Rows: | Edit: | Export/Import:

log_id	order_id	customer_id	order_date	o_status	total_amount	changed_at
1	25	18	2025-03-23	Cancelled	100.00	2025-03-23 15:24:25
2	26	18	2025-03-23	Pending	100.00	NULL
3	27	18	2025-03-23	Pending	100.00	2025-03-23 15:47:51
NULL	NULL	NULL	NULL	NULL	NULL	

Figure 16. Trigger Logging the New Order result

- ProcessOrder Transaction

```
7    -- Transactions --
8 •  CALL CreateNewOrder(18, 100, 23);
9 •  CALL HandleOrder( 27, 1, 2, 50);
10 • CALL ProcessOrderTransaction( 27, 'Credit Card' ,100 );
11 • CALL DeleteOrderAndRelatedData(27);
12
13 • select *from order_log;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

message
Transaction successful: Order ID 27 completed

Figure 17. ProcessOrder Transaction result

- Trigger for adding new transaction to the table

```
13 • select *from transactions;
```

Result Grid | Filter Rows: | Edit: | Export/Impor

transaction_id	transaction_date	payment_method	amount	order_id
6	2024-07-21	PayPal	7.44	18
7	2024-07-20	Credit Card	215.00	19
8	2024-09-13	Stripe	14.00	20
9	2024-09-24	Credit Card	20.24	21
10	2024-10-11	Debit Card	19.98	22
11	2024-11-28	Bank Transfer	21.95	23
12	2024-09-15	Apple Pay	50.00	24
14	2025-03-23	Credit Card	100.00	27
NULL	NULL	NULL	NULL	NULL

Figure 18. Adding new transaction to the table using trigger

- Trigger for changing the order status after processing transaction

```
13 •  select *from orders;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

order_id	order_date	total_amount	o_status	customer_id	address_id	changed_at
19	2024-07-20	215.00	Deliverd	21	29	2024-07-20 00:00:00
20	2024-09-13	14.00	Deliverd	31	26	2024-09-13 00:00:00
21	2024-09-24	20.24	Deliverd	31	26	2024-09-24 00:00:00
22	2024-10-11	19.98	Pending	27	27	2024-10-11 00:00:00
23	2024-11-28	21.95	Pending	29	32	2024-11-28 00:00:00
24	2024-09-15	50.00	Pending	33	31	2024-09-15 00:00:00
26	2025-03-23	100.00	Pending	18	23	NULL
27	2025-03-23	100.00	Completed	18	23	2025-03-23 15:47:51
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 19. Changing the order status after processing transaction using trigger

- Logging the status change in order log table

Result Grid | Filter Rows: | Edit: | Export/Import:

log_id	order_id	customer_id	order_date	o_status	total_amount	changed_at
1	25	18	2025-03-23	Cancelled	100.00	2025-03-23 15:24:25
2	26	18	2025-03-23	Pending	100.00	NULL
3	27	18	2025-03-23	Pending	100.00	2025-03-23 15:47:51
4	27	18	2025-03-23	Completed	100.00	2025-03-23 15:47:51
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 20. Logging the status change in order log table

- Deleting order and related data

```
10 •  CALL DeleteOrderAndRelatedData(25);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

message
Deletion successful.

Figure 21. Deleting order and related data using Transaction

In the NoSQL database, the results of the queries are shown below.

- Find and read a product name and reviews

```
> //Find and read Only a Product Name and Reviews:
  db.BookReviews.find(
    { product_id: 2 },
    { product_name: 1, reviews: 1, _id: 0 }
  );
< {
    product_name: 'Harry Potter',
    reviews: [
      {
        customer_id: 18,
        customer_name: 'John',
        rating: 4,
        review_text: 'Magical and engaging, though slightly predictable.',
        review_date: 2025-03-22T12:05:00.000Z
      },
      {
        customer_id: 21,
        customer_name: 'Rees',
        rating: 5,
        review_text: 'Magical, enchanting, and a true delight for all ages.',
        review_date: 2025-03-22T12:05:00.000Z
      }
    ]
  }
```

Figure 22. Find and read a product name and reviews; code and result

- Adding a review to a product

```
> //Add a review to an existing product (updating its reviews)
  db.BookReviews.updateOne(
    { product_id: 1 },
    { $push: { reviews: {
        customer_id: 25,
        customer_name: "Chris",
        rating: 4,
        review_text: "Enjoyable and adventurous."
    } } }
  );
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Figure 23. Adding a review to a product; code and result

- Sorting the products based on review count

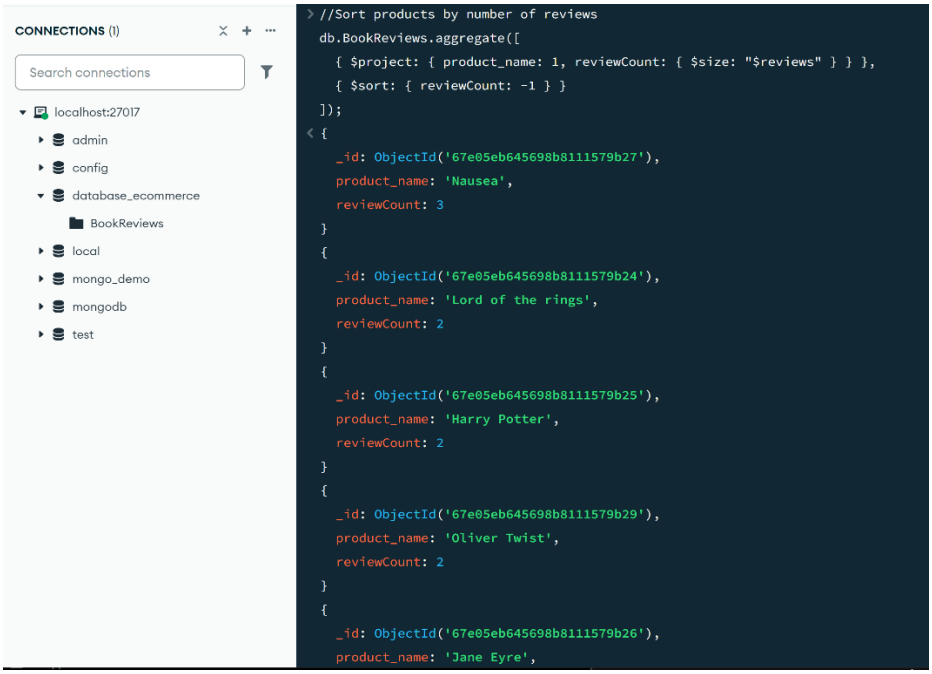

Figure 24. Sorting the products based on review count; code and result

Conclusion and Future Projects

As explained in this project, there are many operations that can be implemented in an E-commerce database using both SQL and NoSQL technologies, allowing for efficient management and data analysis. For the future development of this project, a few more tables can also be added to have a more comprehensive system. For instance, a cart table can be added to temporarily store the products chosen by user before they are transferred into the orders table. There can also be log tables for products to better manage the inventory. Moreover, triggers can be created before deleting or updating data to keep the data secure and prevent them from being accidentally deleted. For the NoSQL database, more types of data such as pictures can be added to the reviews Finally, more advanced queries can be used for more complex functions and an advanced hybrid database system.