MINI PROJECT

E-Commerce Management System

A report submitted to the

Department of Electrical and Information Engineering

Faculty of Engineering

University of Ruhuna

Sri Lanka

On 13th of September 2025

In completing an assignment for the module

EE4203/EC4203 Database Systems

By Group No: 51

EG/2022/4905: AASIF A.K.M.

EG/2022/5226: NUSAIR M.N.M.

EG/2022/5261: RAJAAEI M.N.I.

EG/2022/5343: SHAROZ R.A.W.

EG/2022/5367: THARUKSHAN G.

TABLE OF CONTENTS

LIST OF FIGURES

PART A – RELATIONAL DATABASES

CHAPTER 1 - REQUIREMENT ANALYSIS

1.1 Introduction

A Database Management System, or DBMS, is a fundamental computer software program for designing, managing, and accessing databases efficiently. A DBMS provides a structured interface to define a database schema, insert and update data, and extract key information useful in making business decisions. E-commerce websites are the finest examples of systems that are completely reliant on high-performance DBMS in today's digital world to handle enormous inventories, customer details, and massive volumes of transactions. It would be unpractical and error-prone to manage this complexity without a DBMS.

This project entails the design and implementation of a database for an E-Commerce Management System. This system will serve as the backbone for an online store, maintaining data about products, customers, orders, and payments. It will enable secure registration of users, efficient product cataloguing, seamless order processing, and guaranteed payment tracking. Therefore, this database is an essential resource for business owners, administrators, and customers to access a single source of truth regarding all e-commerce activity and to make retrieving data-driven insights effortless.

1.2 Functional Requirements

The E-Commerce Management System shall incorporate the following characteristics

- The system shall facilitate new customers to sign up and create a personal account.
- The system shall facilitate registered customers to log in securely to their accounts.
- The system shall facilitate customers to view, edit, and maintain their personal profile and shipping addresses.
- The system shall facilitate administrators to add, edit, and remove product listings from the catalogue.
- The system will allow customers to browse products by product categories and search for products by product name or product description.
- The system will allow customers to place products into an online shopping cart, update item quantities, and remove items.
- The system will allow customers to check out and have the shopping cart contents transferred into an official order.
- The system will debit payments against orders and change the status of the order after successful payment.

- The system will manage product stock, decreasing automatically when a successful order is completed.
- The system will allow customers to view their ordering history and the status of all orders.
- The system will allow user views to generate user-specific reports of sales levels and popularity of products.

1.3 Data Requirements

The system must store and manage the following core data entities and their attributes to support the functional requirements

Entity: Customer - Attributes

- Customer ID
- Email
- Password Hash
- First Name
- Last Name
- Phone
- Registration Date

Entity: Address - Attributes

- Address ID
- Customer ID
- Address Line 1
- Address Line 2
- City
- State
- Zip Code
- Address Type

Entity: Category - Attributes

- Category ID
- Category Name
- Description
- Parent Category ID

Entity: Product - Attributes

- Product ID
- Product Name
- Description
- Price
- Stock Quantity
- Weight
- Date Added
- Status
- Category ID

Entity: ProductImage - Attributes

- Product ID
- Image URL

Entity: ShoppingCart - Attributes

- Cart ID
- Creation Date
- Customer ID

Entity: CartItem - Attributes

- Cart ID
- Product ID
- Quantity

Entity: Order - Attributes

- Order ID
- Order Date
- Total Amount
- Status
- Customer ID

Entity: OrderItem - Attributes

- Order ID
- Product ID
- Quantity
- Unit Price

Entity: Payment - Attributes

- Payment ID
- Payment Method
- Amount
- Transaction Date
- Status
- Order ID

CHAPTER 2 - CONCEPTUAL DESIGN (ER DIAGRAM)

2.1 ER Diagram

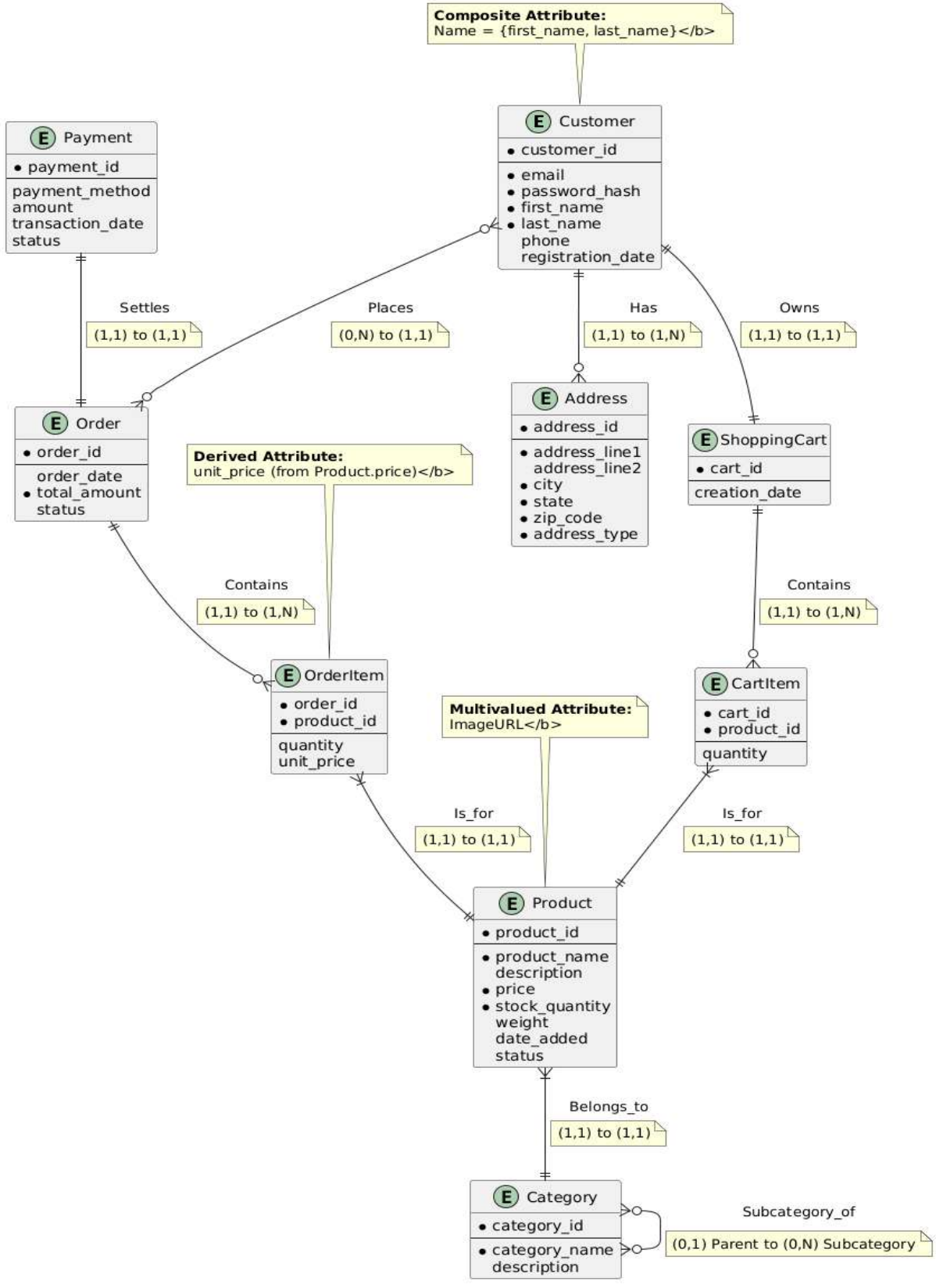

Figure 1 : ER Diagram for E-Commerce Management System

2.2 Relations in the ER Diagram

The Entity-Relationship (ER) diagram created provides a complete conceptual view of the database for the E-Commerce Management System. The key elements and design decisions included in the diagram are explained in this section.

2.2.1 Strong Entities

Strong entities are those that exist independently and do not rely on any other entity for their identification. Their primary key is uniquely defined by their own attributes.

- Customer: Represents a registered user of the platform.
- Category: Represents a classification for products (e.g., Electronics, Clothing).
- Product: Represents an item available for sale.
- ShoppingCart: Represents a virtual cart associated with a single customer.
- Order: Represents a completed purchase transaction.
- Payment: Represents a financial transaction that settles an order.

2.2.2 Weak Entities

Weak entities are existence-dependent on a strong entity. Their primary key is partially or fully derived from the primary key of the strong entity they relate to.

- Address: A weak entity that depends on the Customer entity. An address cannot exist without a customer.
- CartItem: A weak entity that depends on the ShoppingCart entity. An item cannot be in a cart without the cart itself.
- ProductImage: A weak entity that depends on the Product entity. An image cannot exist without a product.

2.2.3 Relationship Cardinalities

One-to-One (1 : 1) Relationships

- Customer OWNS ShoppingCart: (1, 1) to (1, 1)
  One Customer must own exactly one ShoppingCart. One ShoppingCart is owned by exactly one Customer.
- Payment SETTLES Order: (1, 1) to (1, 1)
  One Payment must settle exactly one Order. One Order must be settled by exactly one Payment.

One-to-Many (1 : N) Relationships

- Customer PLACES Order: (0, N) to (1, 1)
  One Customer can place zero or many Orders. One Order must be placed by exactly one Customer.
- Customer HAS Address: (1, 1) to (1, N)
  One Customer must have one or more Addresses. One Address must belong to exactly one Customer.
- ShoppingCart CONTAINS CartItem: (1, 1) to (1, N)
  One ShoppingCart must contain one or more CartItems. One CartItem must be contained in exactly one ShoppingCart.
- Order CONTAINS OrderItem: (1, 1) to (1, N)
  One Order must contain one or more OrderItems. One OrderItem must be part of exactly one Order.
- Product BELONGS_TO Category: (1, 1) to (1, 1)
  One Product must belong to exactly one Category. One Category can have one or many Products.
- Product HAS ProductImage: (1, 1) to (1, N)
  One Product must have one or more ProductImages. One ProductImage must belong to exactly one Product.

Many-to-Many (M : N) Relationships

- The relationship between Product and ShoppingCart is resolved by the weak entity CartItem. One ShoppingCart can contain many Products, and one Product can be in many ShoppingCarts.
- The relationship between Product and Order is resolved by the weak entity OrderItem. One Order can contain many Products, and one Product can be in many Orders.

Recursive Relationship

- Category IS_SUBCATEGORY_OF Category: (0, 1) to (0, N)
  This is a one-to-many relationship within the same entity. One Category (as a parent) can have zero or many subcategories. One Category (as a subcategory) can have zero or one parent Categories.

2.2.4 Special Constructs

Composite Attributes

- Customer Name: Composed of the atomic attributes First Name and Last Name.
- Customer Address: (In the Address entity) Composed of Address Line 1, Address Line 2, City, State, and Zip Code.

Multivalued Attributes

- Product Image: A product can have multiple image URLs. This is modeled as the separate weak entity ProductImage.

Derived Attribute

- OrderItem.UnitPrice: This attribute is derived from the Price attribute in the Product entity at the time the order is placed. It is stored historically in the OrderItem entity to preserve the sale price, making it a derived and stored attribute.

CHAPTER 3 – LOGICAL DESIGN AND NORMALIZATION

3.1 Mapping to the Relational Model (1NF)

The following tables represent the initial mapping of the ER diagram to a relational schema in First Normal Form (1NF). Each table has a primary key, and all attributes are atomic.

1. Customer Table

Attribute	Data Type	Constraints
customer_id	INT	PRIMARY KEY
email	VARCHAR(255)	NOT NULL, UNIQUE
password_hash	VARCHAR(255)	NOT NULL
first_name	VARCHAR(100)	NOT NULL
last_name	VARCHAR(100)	NOT NULL
phone	VARCHAR(20)	
registration_date	DATE	NOT NULL

2. Address Table

Attribute	Data Type	Constraints
address_id	INT	PRIMARY KEY
customer_id	INT	NOT NULL, FOREIGN KEY
address_line1	VARCHAR(255)	NOT NULL
address_line2	VARCHAR(255)	
city	VARCHAR(100)	NOT NULL
state	VARCHAR(100)	NOT NULL
zip_code	VARCHAR(20)	NOT NULL

Attribute	Data Type	Constraints
address_type	ENUM('S','B')	NOT NULL

3. Category Table

Attribute	Data Type	Constraints
category_id	INT	PRIMARY KEY
category_name	VARCHAR(100)	NOT NULL
description	TEXT	
parent_category_id	INT	FOREIGN KEY (self)

4. Product Table

Attribute	Data Type	Constraints
product_id	INT	PRIMARY KEY
product_name	VARCHAR(255)	NOT NULL
description	TEXT	
price	DECIMAL(10,2)	NOT NULL
stock_quantity	INT	NOT NULL
weight	DECIMAL(10,2)	
date_added	DATE	NOT NULL
status	ENUM('A','I')	NOT NULL
category_id	INT	NOT NULL, FOREIGN KEY

5. ProductImage Table

Attribute	Data Type	Constraints
product_id	INT	FOREIGN KEY (PK)
image_url	VARCHAR(500)	NOT NULL (PK)
PRIMARY KEY (product_id, image_url)		

6. ShoppingCart Table

Attribute	Data Type	Constraints
cart_id	INT	PRIMARY KEY
creation_date	DATE	NOT NULL
customer_id	INT	NOT NULL, UNIQUE, FK

7. CartItem Table

Attribute	Data Type	Constraints
cart_id	INT	FOREIGN KEY (PK)
product_id	INT	FOREIGN KEY (PK)
quantity	INT	NOT NULL, CHECK > 0
PRIMARY KEY (cart_id, product_id)		

8. Order Table

Attribute	Data Type	Constraints
order_id	INT	PRIMARY KEY
order_date	DATETIME	NOT NULL
total_amount	DECIMAL(10,2)	NOT NULL
status	ENUM('P','D','C')	NOT NULL

Attribute	Data Type	Constraints
customer_id	INT	NOT NULL, FOREIGN KEY

9. OrderItem Table

Attribute	Data Type	Constraints
order_id	INT	FOREIGN KEY (PK)
product_id	INT	FOREIGN KEY (PK)
quantity	INT	NOT NULL, CHECK > 0
unit_price	DECIMAL(10,2)	NOT NULL
PRIMARY KEY (order_id, product_id)		

10. Payment Table

Attribute	Data Type	Constraints
payment_id	INT	PRIMARY KEY
payment_method	VARCHAR(50)	NOT NULL
amount	DECIMAL(10,2)	NOT NULL
transaction_date	DATETIME	NOT NULL
status	ENUM('S','F','R')	NOT NULL
order_id	INT	NOT NULL, UNIQUE, FK

3.2 Functional Dependencies

For each table, the primary key functionally determines all other attributes.

- **Customer:** customer_id → {email, password_hash, first_name, last_name, phone, registration_date}
- **Address:** address_id → {customer_id, address_line1, address_line2, city, state, zip_code, address_type}
- **Category:** category_id → {category_name, description, parent_category_id}
- **Product:** product_id → {product_name, description, price, stock_quantity, weight, date_added, status, category_id}
- **ProductImage:** {product_id, image_url} → {} (No other attributes)
- **ShoppingCart:** cart_id → {creation_date, customer_id}
- **CartItem:** {cart_id, product_id} → quantity
- **Order:** order_id → {order_date, total_amount, status, customer_id}
- **OrderItem:** {order_id, product_id} → {quantity, unit_price}
- **Payment:** payment_id → {payment_method, amount, transaction_date, status, order_id}

3.3 Normalization to Second Normal Form (2NF)

All tables are already in 2NF because:

1. They are all in 1NF (atomic values, no repeating groups).
2. They have no partial dependencies. In all tables with composite primary keys (ProductImage, CartItem, OrderItem), the non-key attributes depend on the entire composite key, not just a part of it. For example, in OrderItem,
   the quantity and unit_price depend on both the order_id and the product_id together.

No further normalization is required to achieve 2NF. The schema is now optimized to avoid update anomalies and redundancy for this normal form.

CHAPTER 4 – IMPLEMENTATION

4.1 Schema Creation

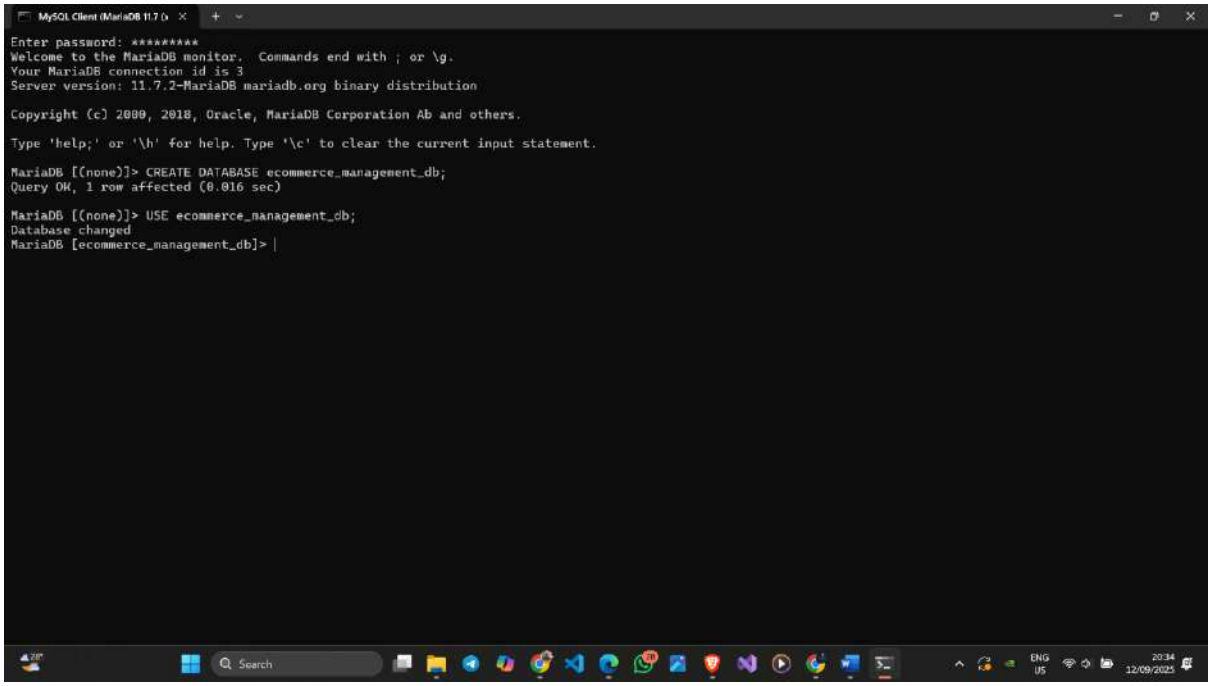

Figure 2 Creating Database

4.2 Table Creation and Definition

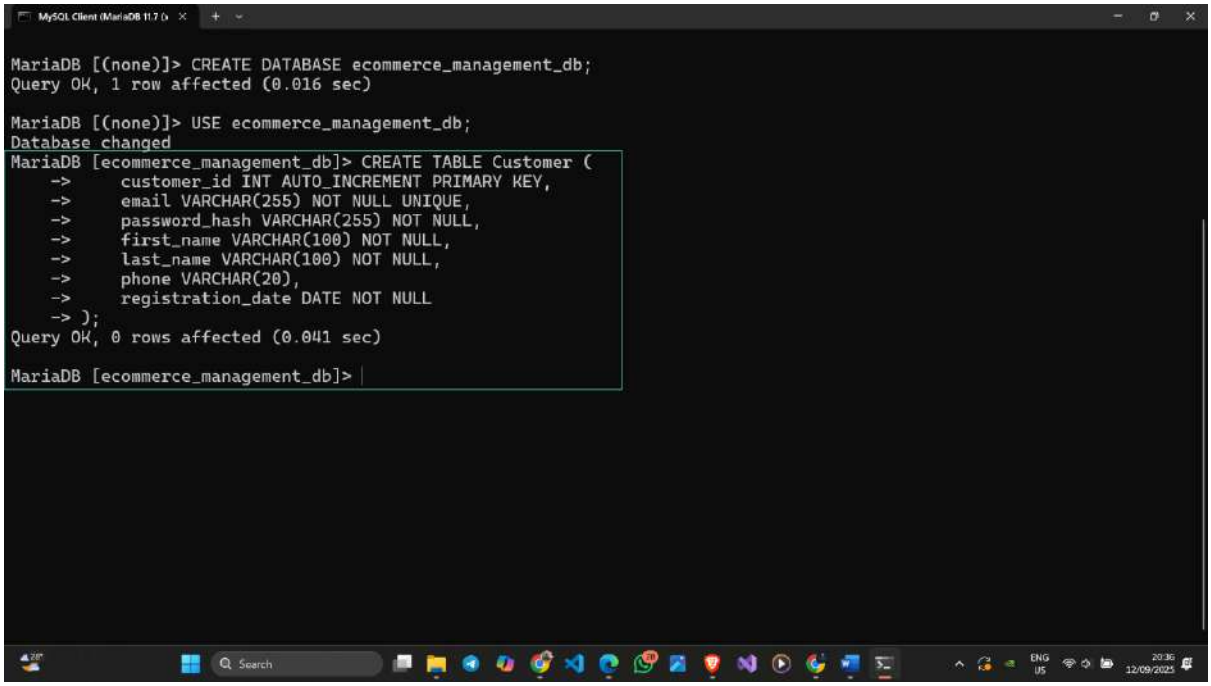

Figure 3 Creating Customer Table

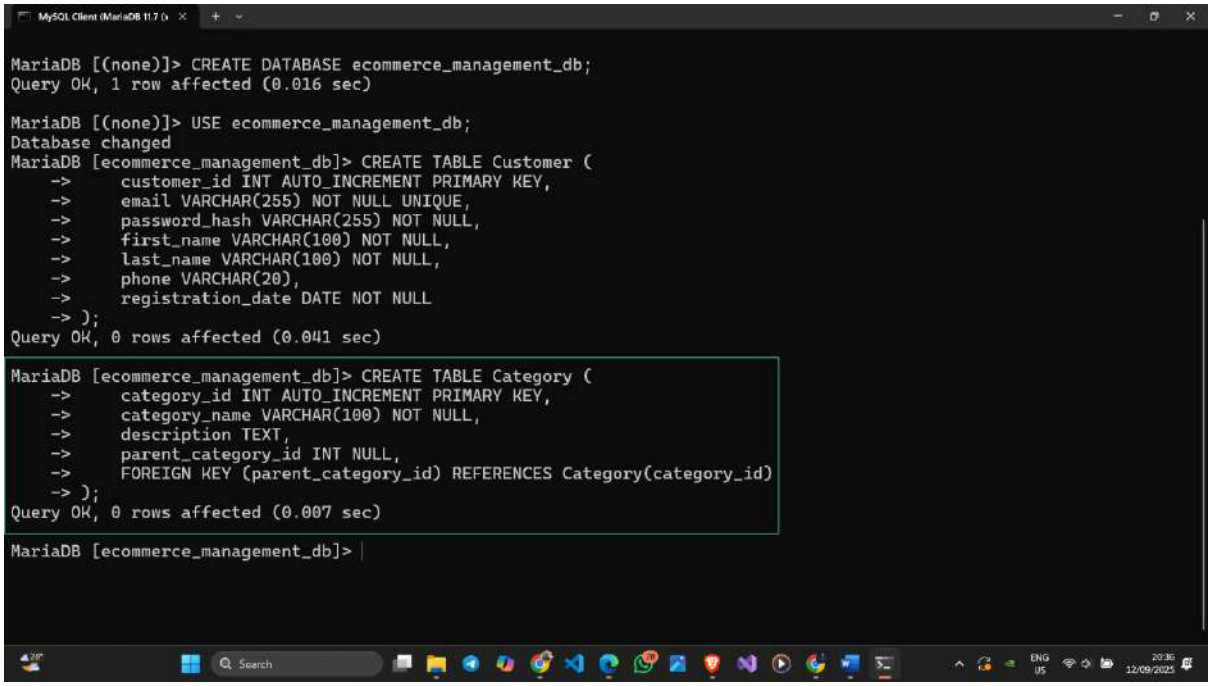

Figure 4 Creating Category Table

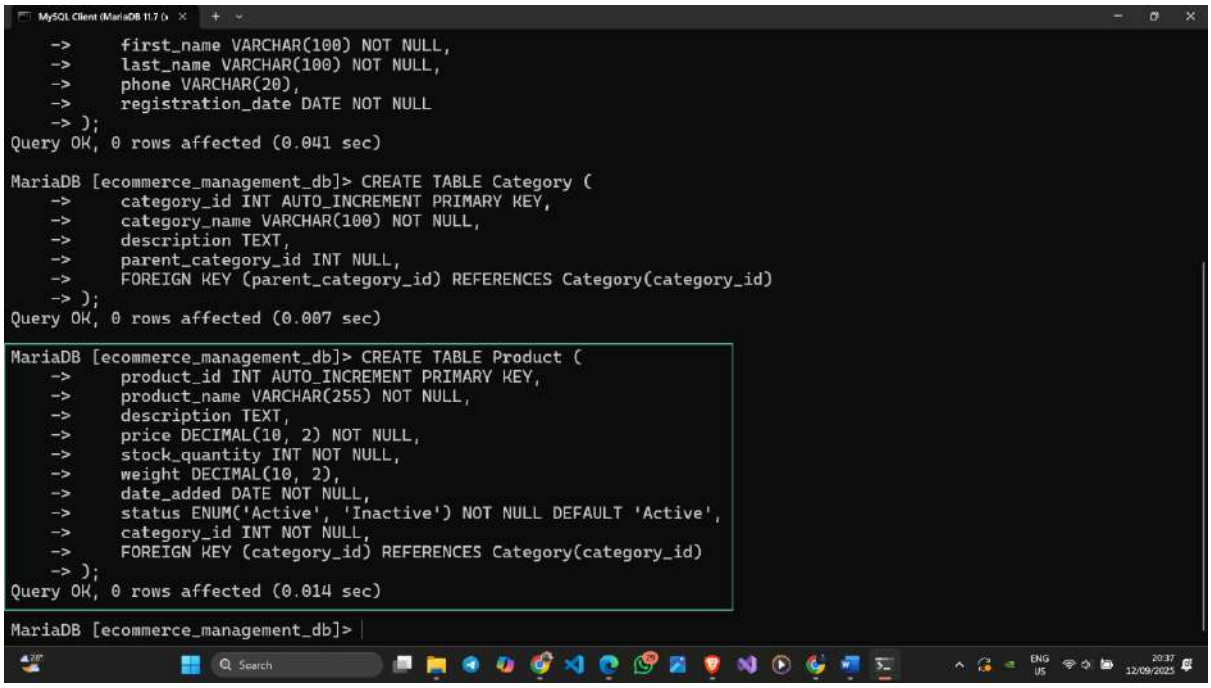

Figure 5 Creating Product Table

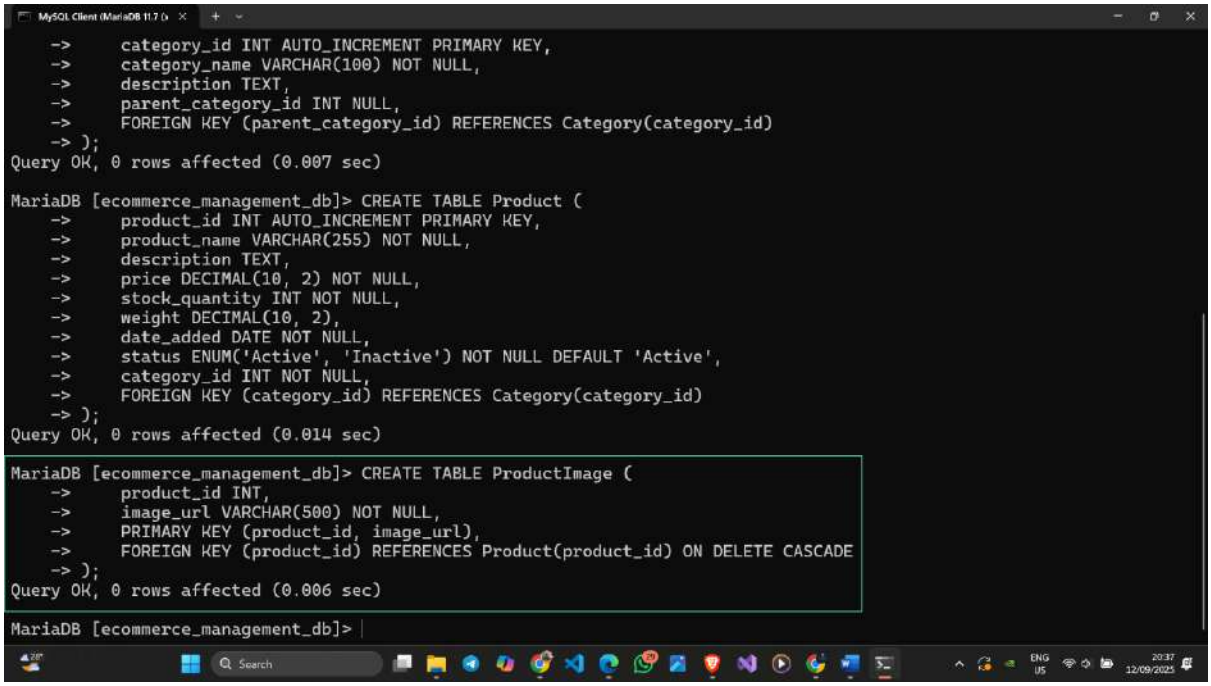

Figure 6 Creating ProductImage Table

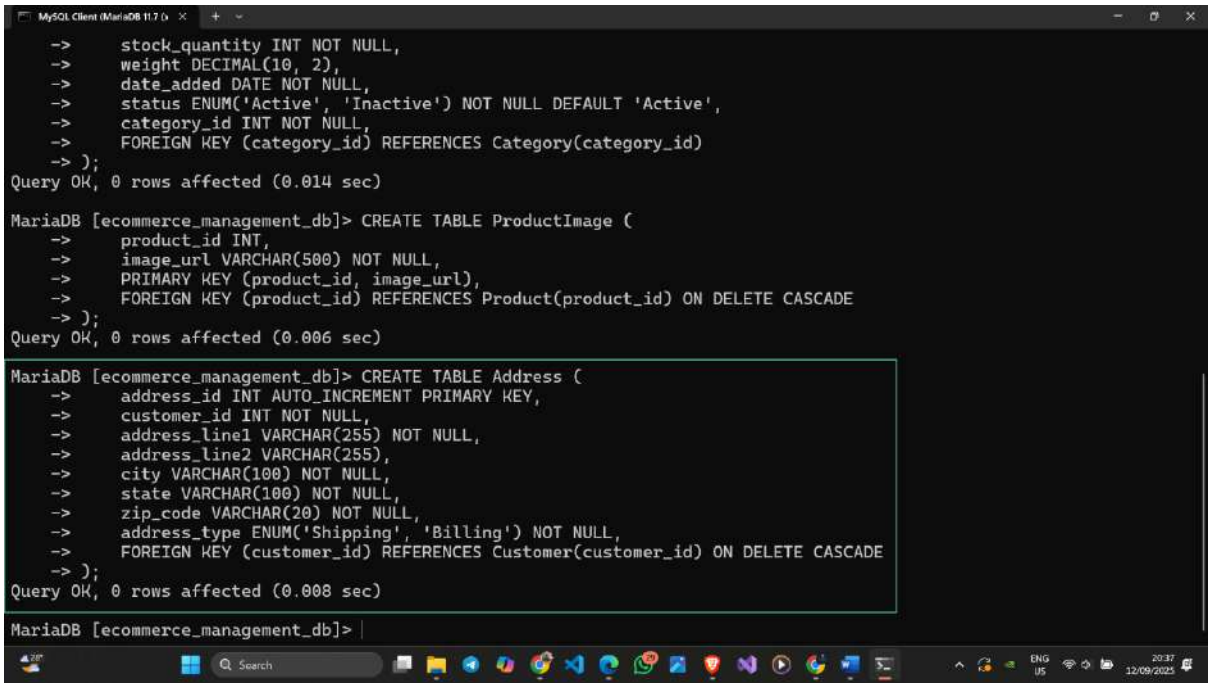

Figure 7 Creating Address Table

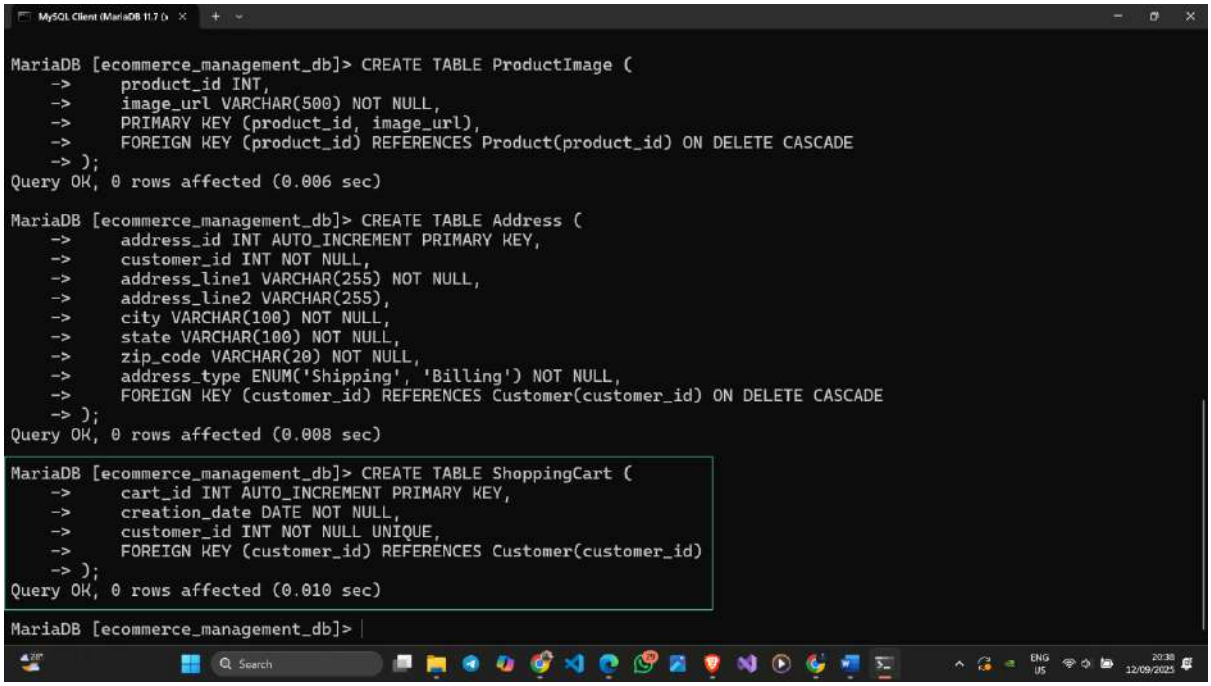

Figure 8 Creating ShoppingCart Table

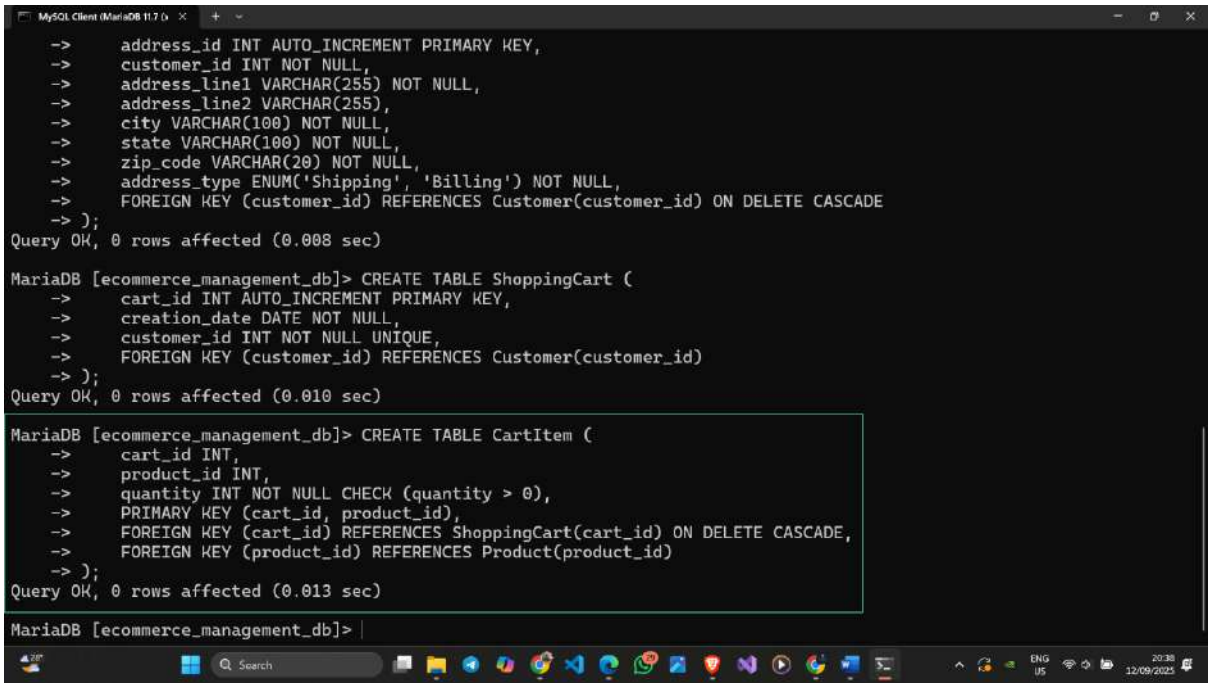

Figure 9 Creating CartItem Table

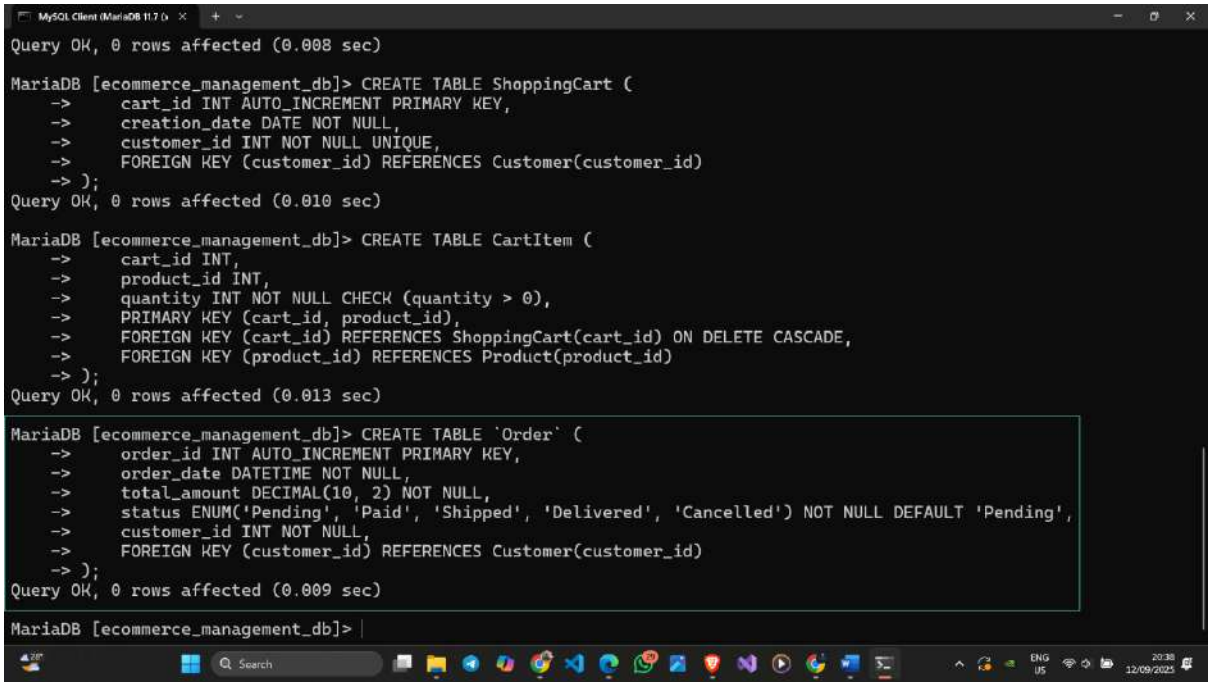

Figure 10 Creating Order Table

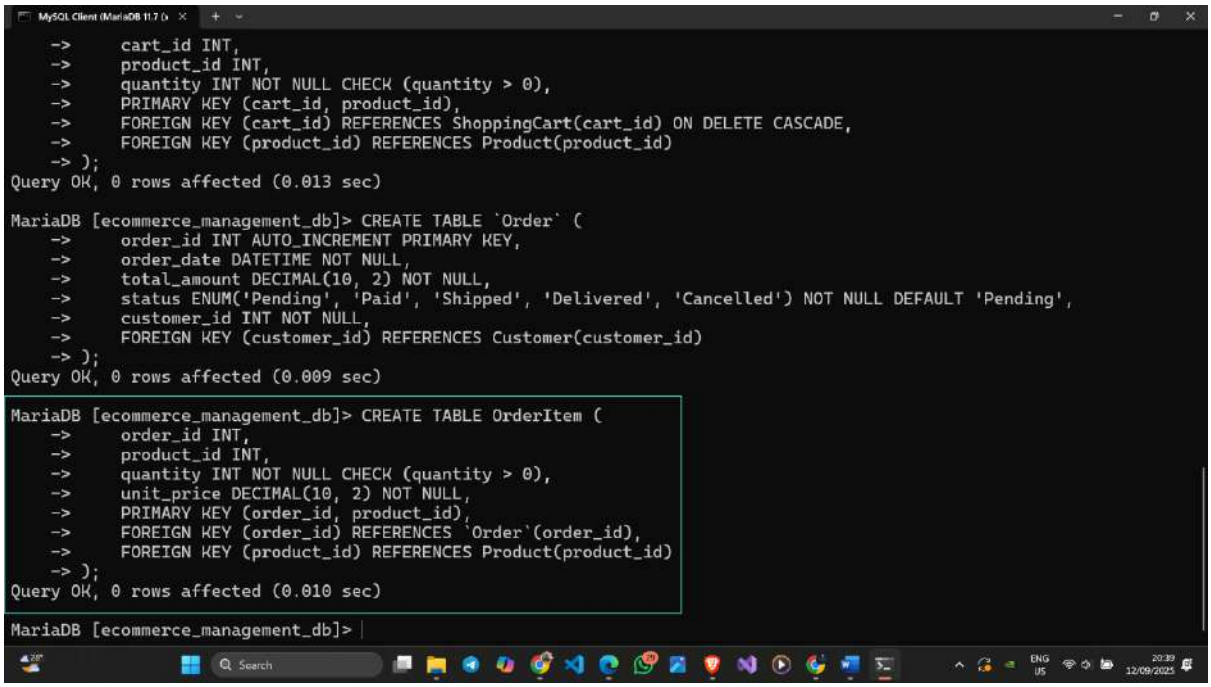

Figure 11 Creating OrderItem Table

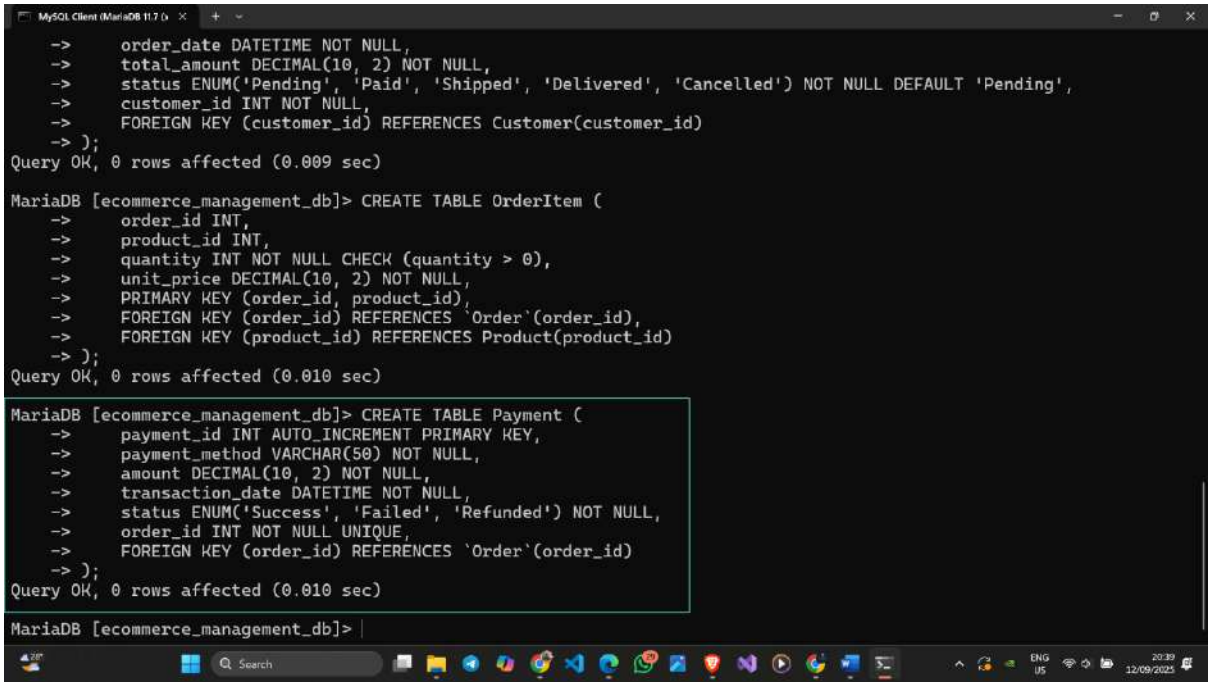

Figure 12 Creating Payment Table

4.3 Data Insertion (INSERT operations)

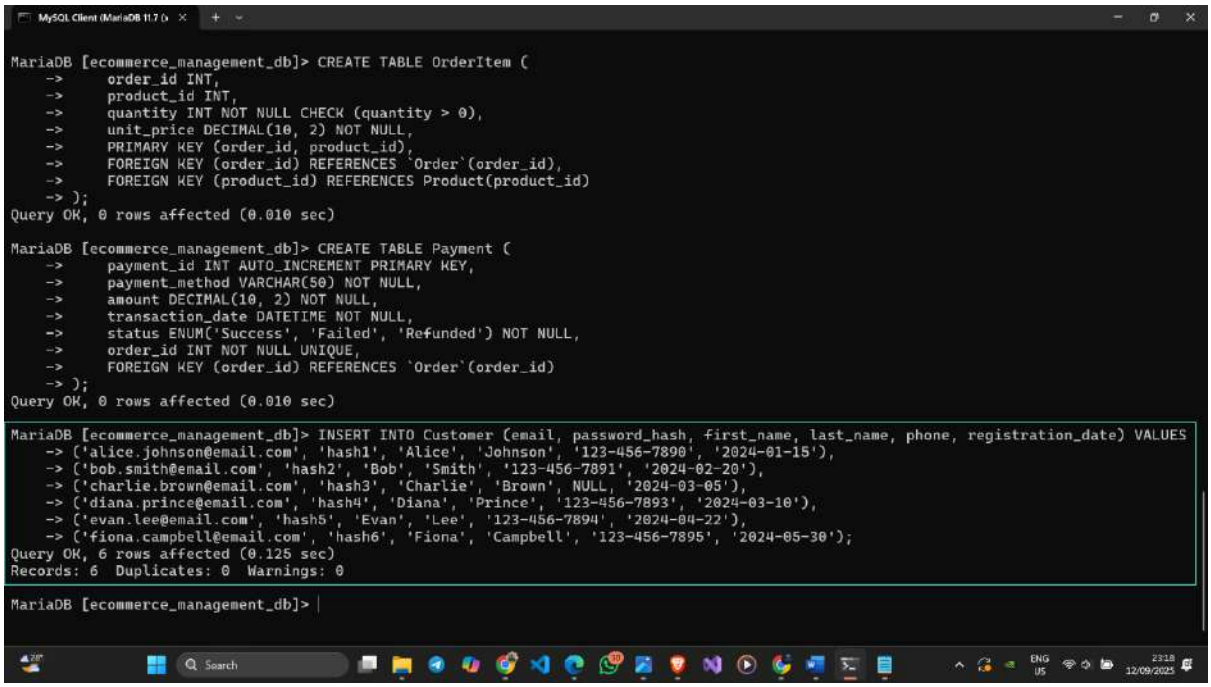

Figure 13 Inserting Data to the Customer Table

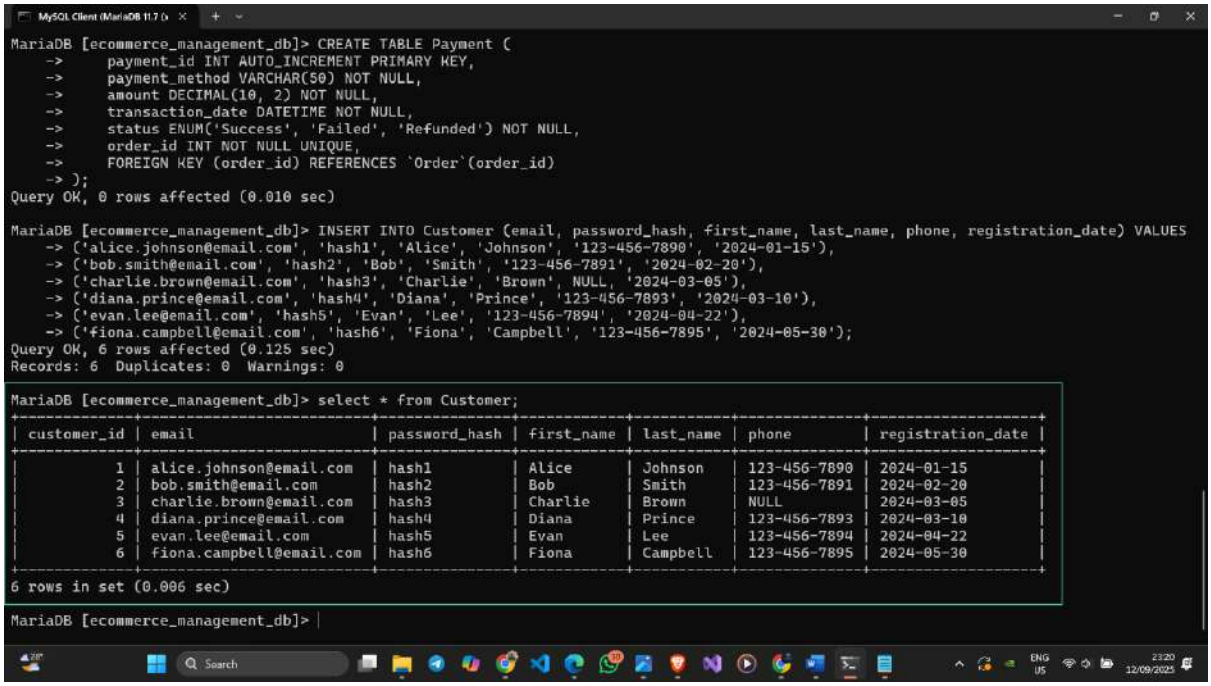

Figure 14 Customer Table with Inserted Data

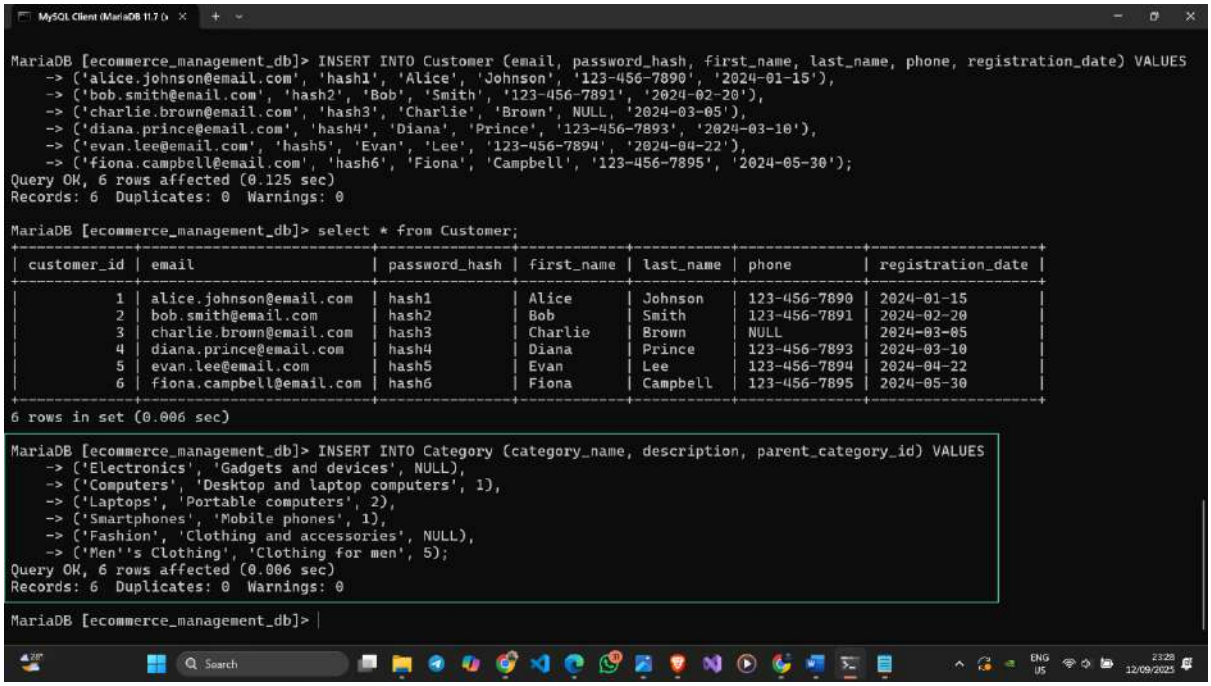

Figure 15 Inserting Data to the Category Table

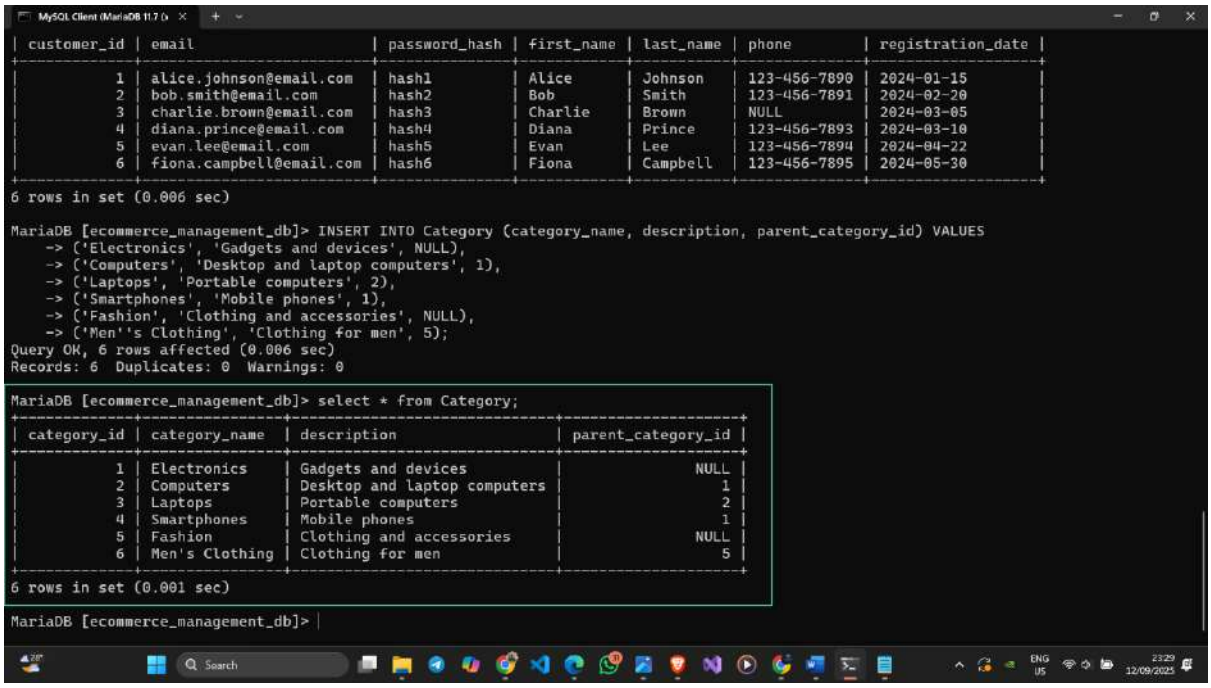

Figure 16 Category Table with Inserted Data

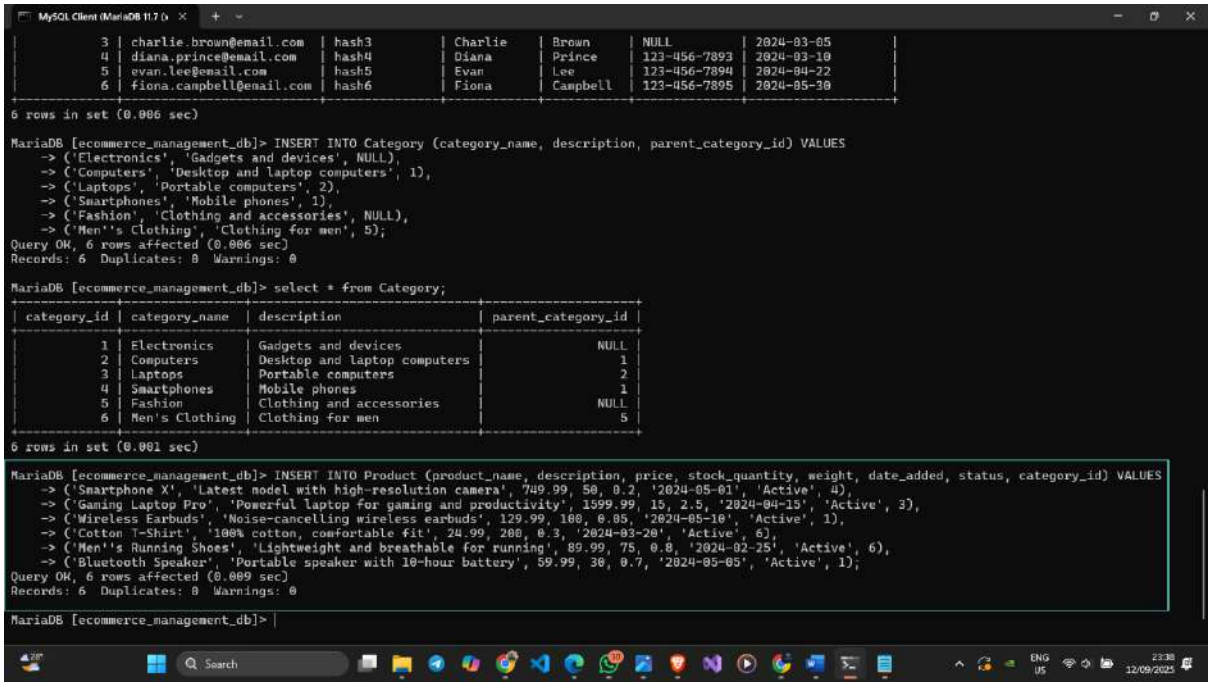

Figure 17 Inserting Data to the Product Table

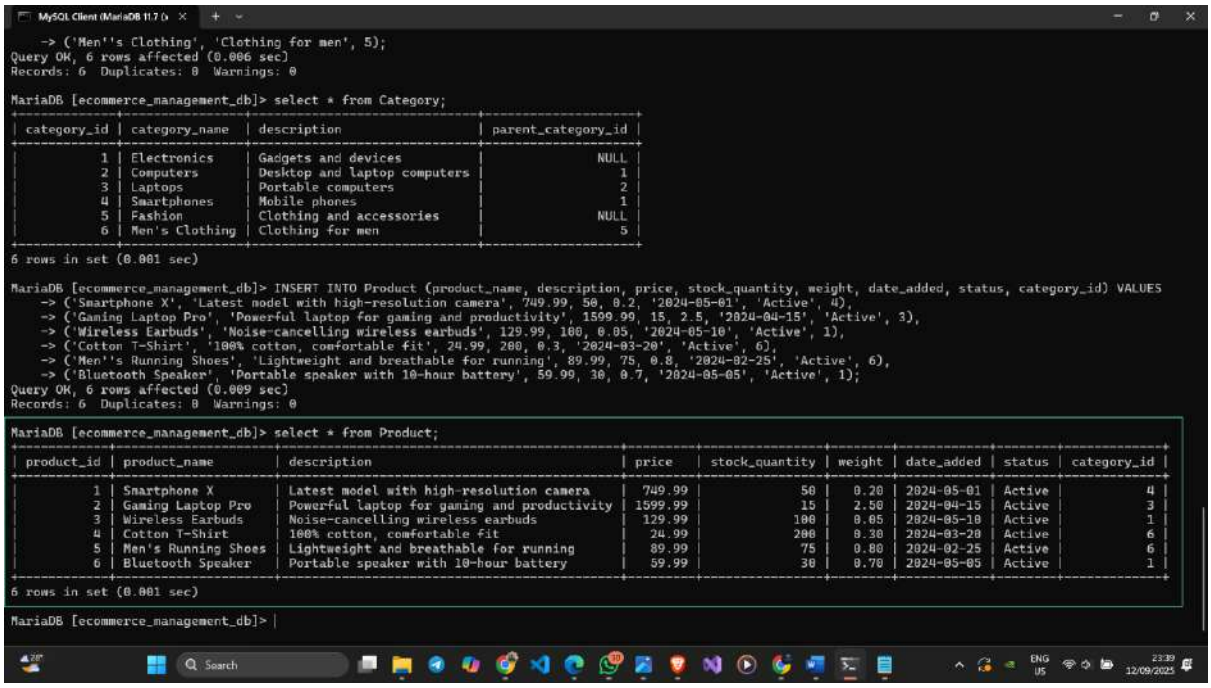

Figure 18 Product Table with Inserted Data

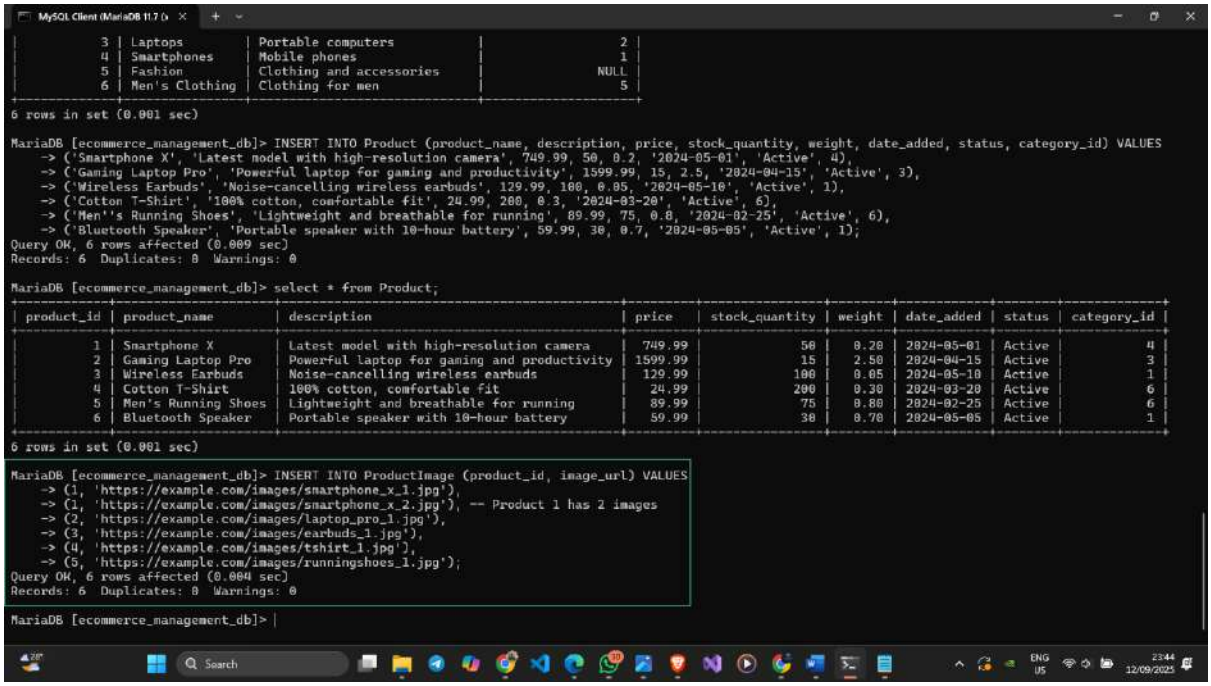

Figure 19 Inserting Data to the ProductImage Table

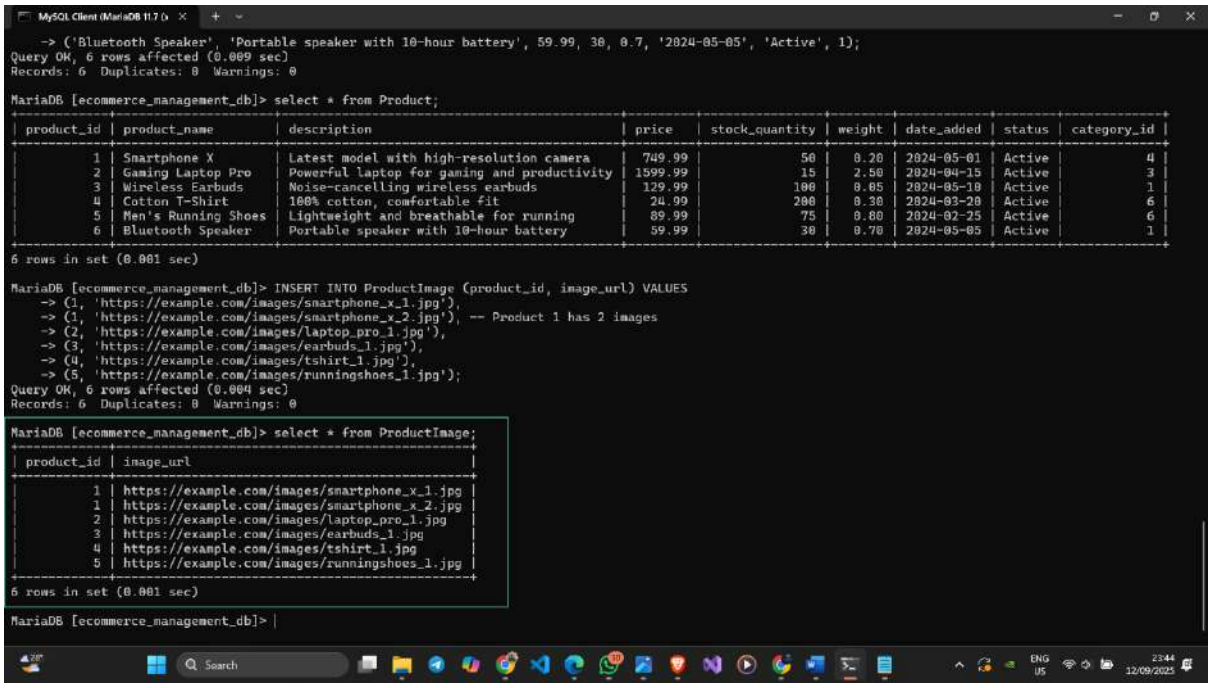

Figure 20 ProductImage Table with Inserted Data

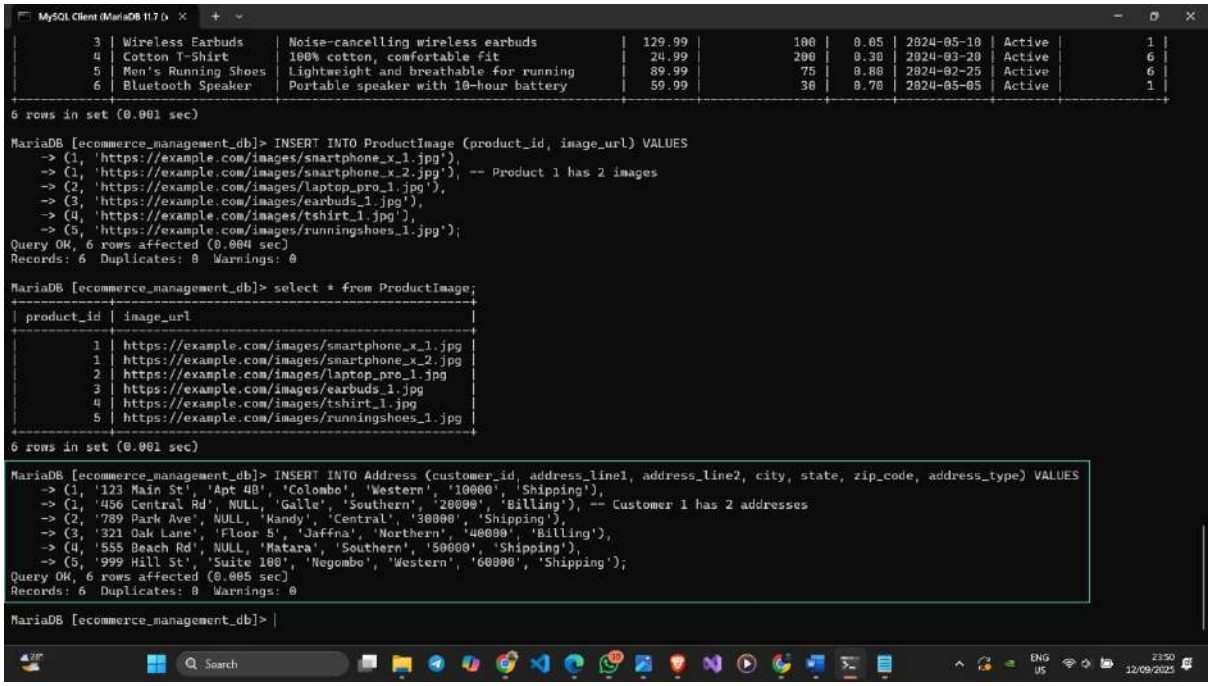

Figure 21 Inserting Data to the Address Table

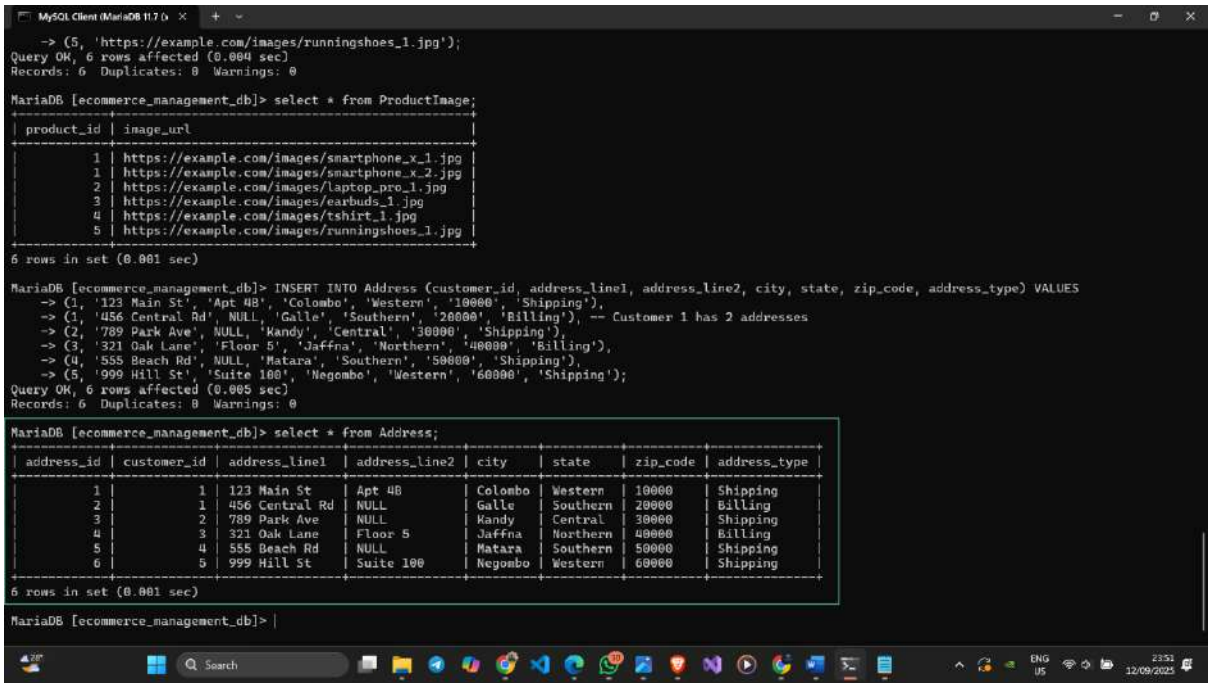

Figure 22 Address Table with Inserted Data

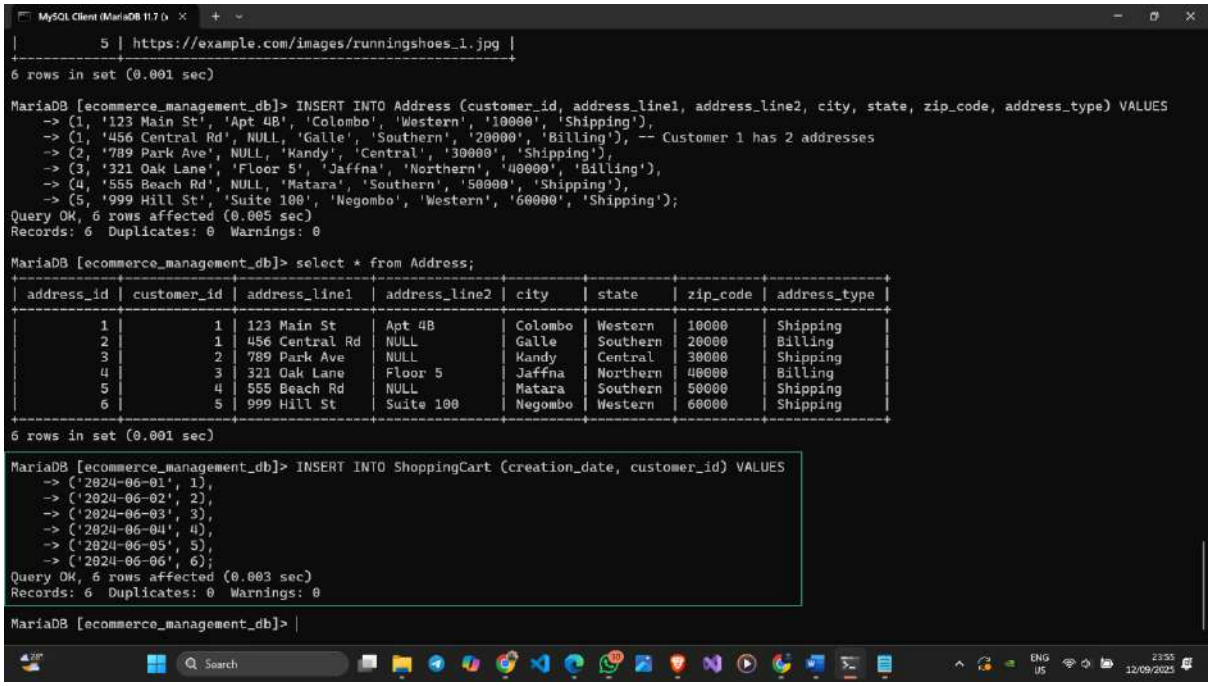

Figure 23 Inserting Data to the ShoppingCart Table

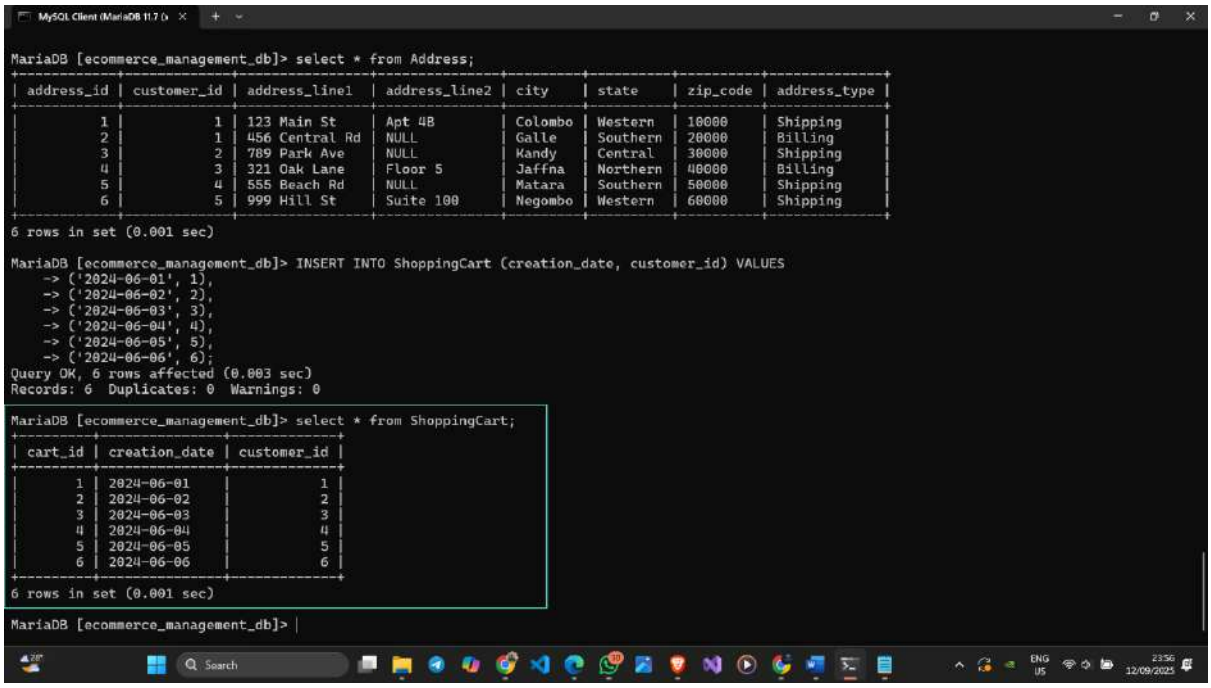

Figure 24 ShoppingCart Table with Inserted Data

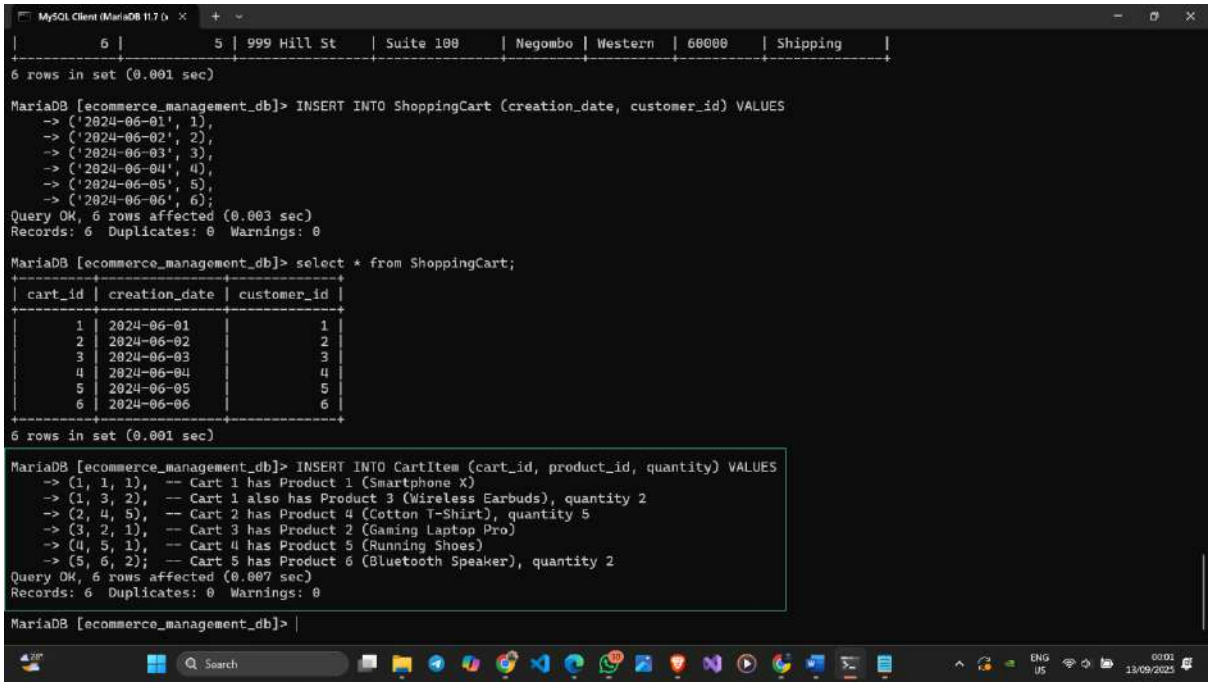

Figure 25 Inserting Data to the CartItem Table

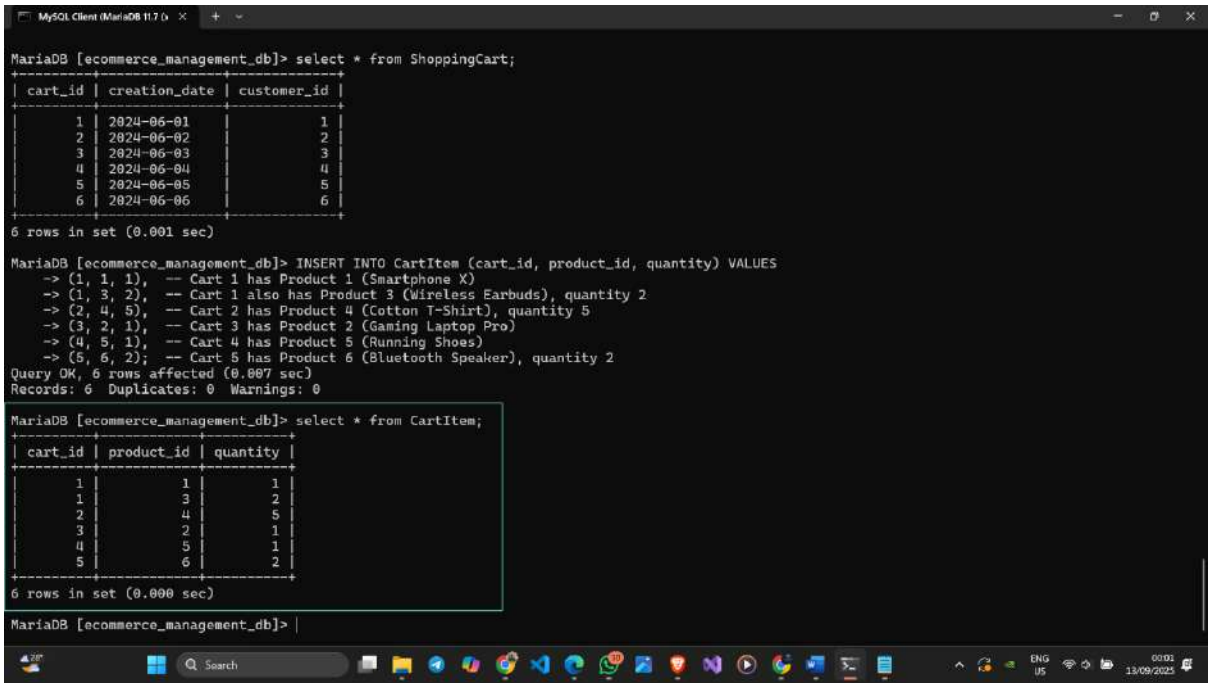

Figure 26 CartItem Table with Inserted Data

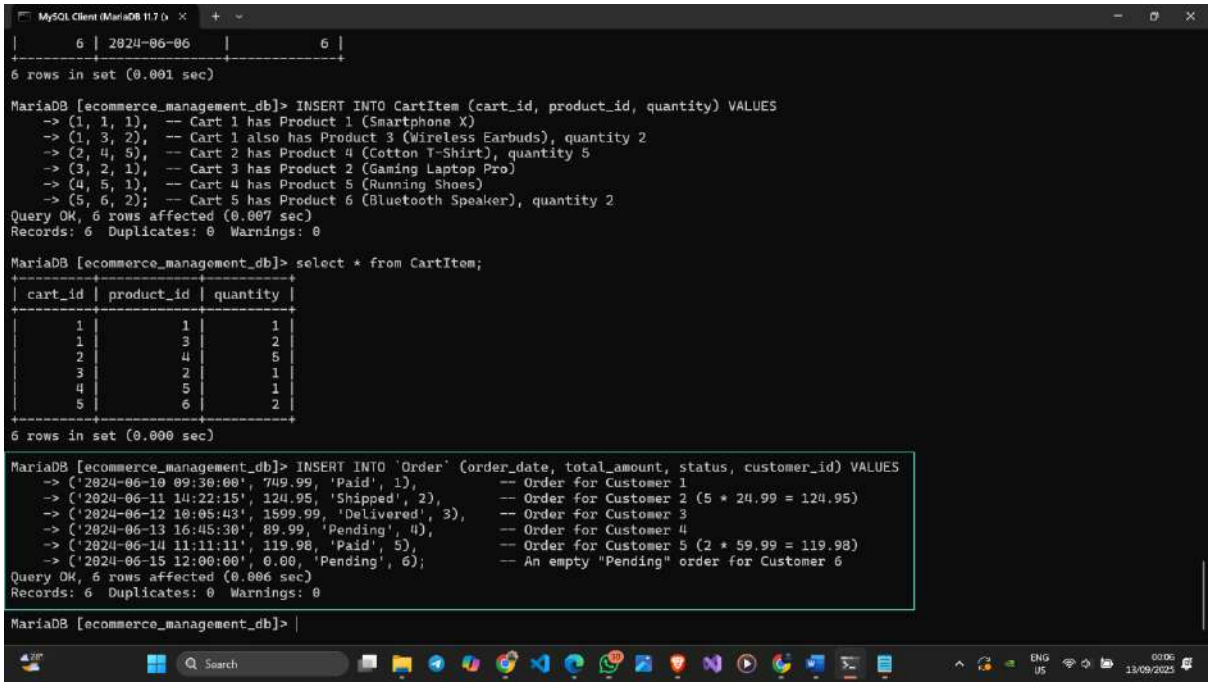

Figure 27 Inserting Data to the Order Table

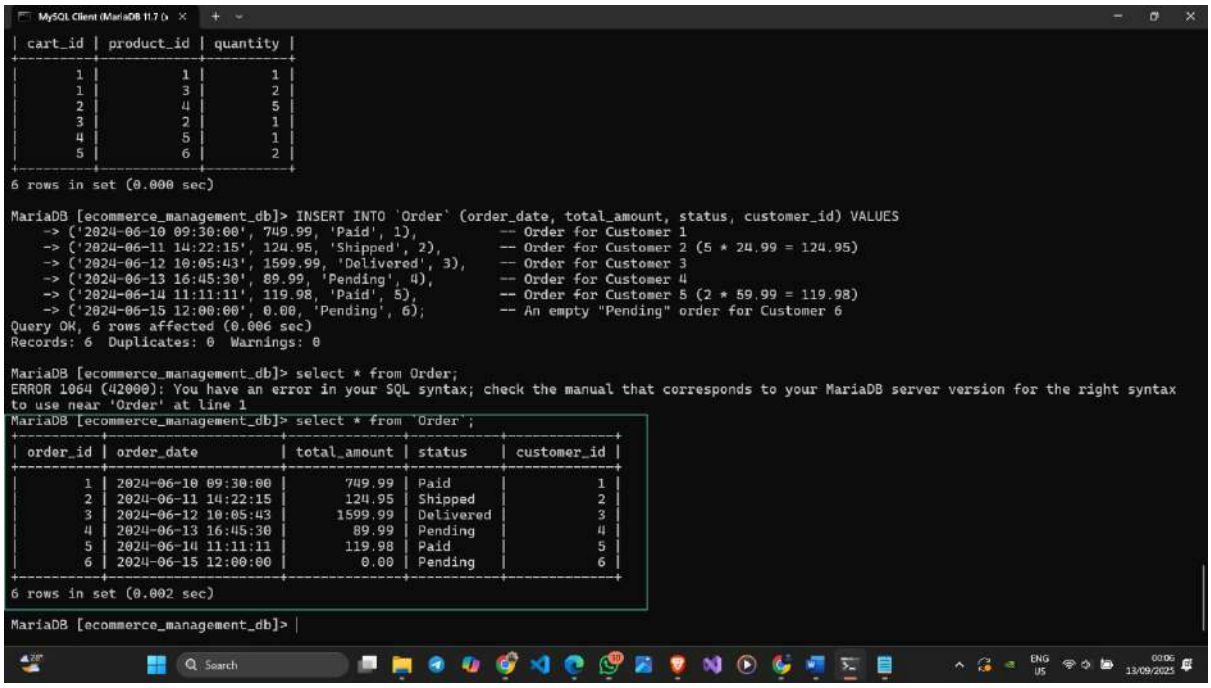

Figure 28 Order Table with Inserted Data

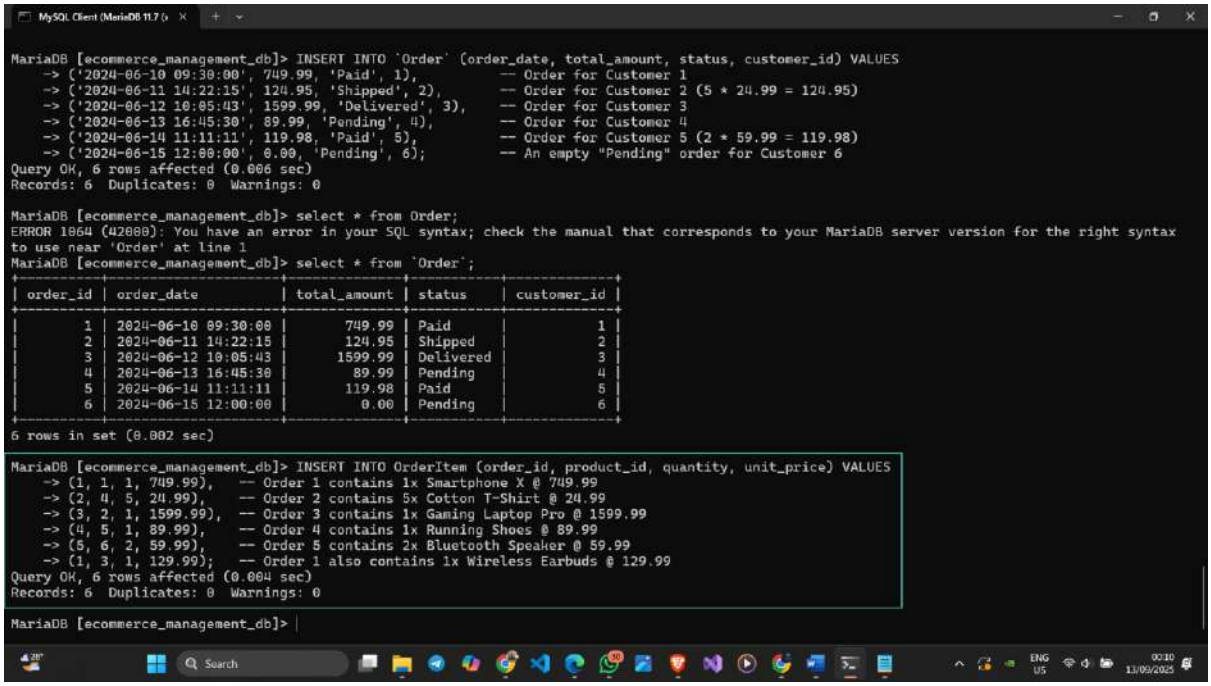

Figure 29 Inserting Data to the OrderItem Table

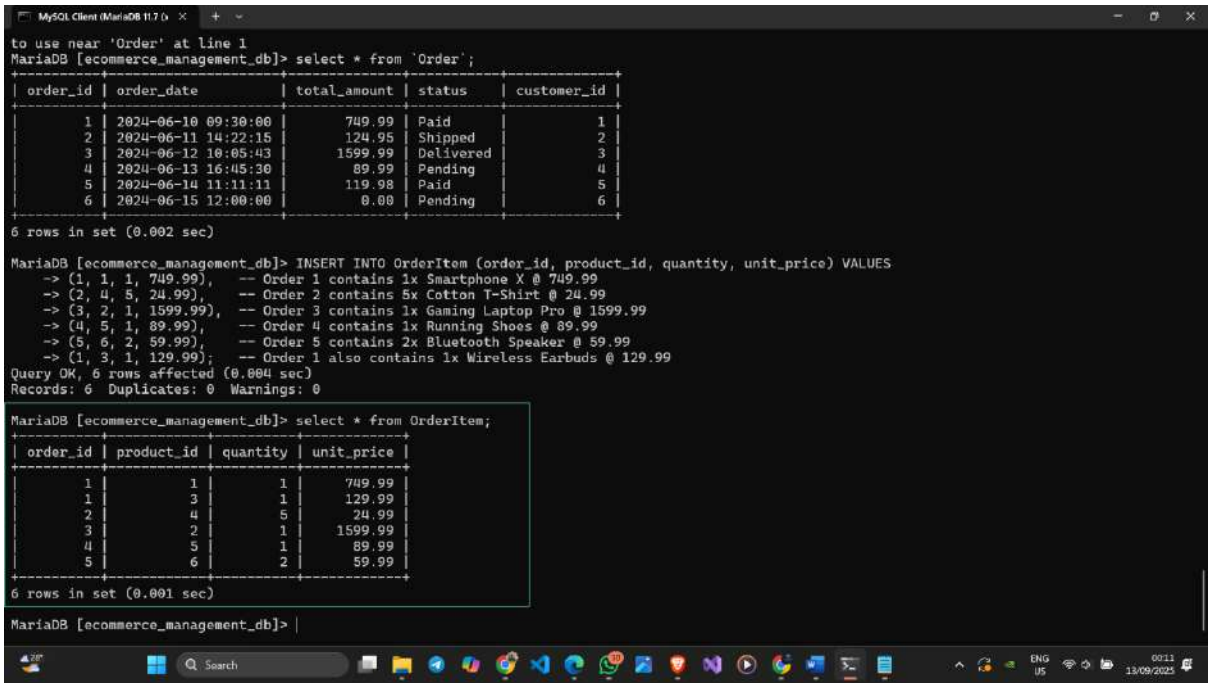

Figure 30 OrderItem Table with Inserted Data

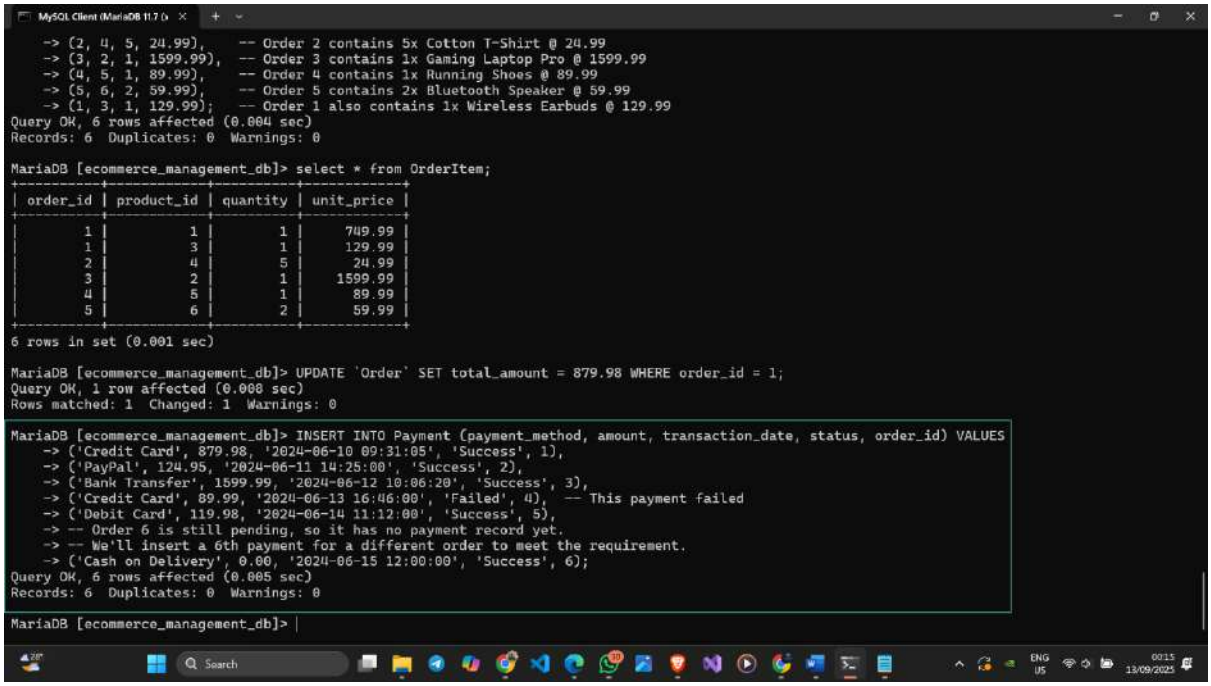

Figure 31 Inserting Data to the Payment Table

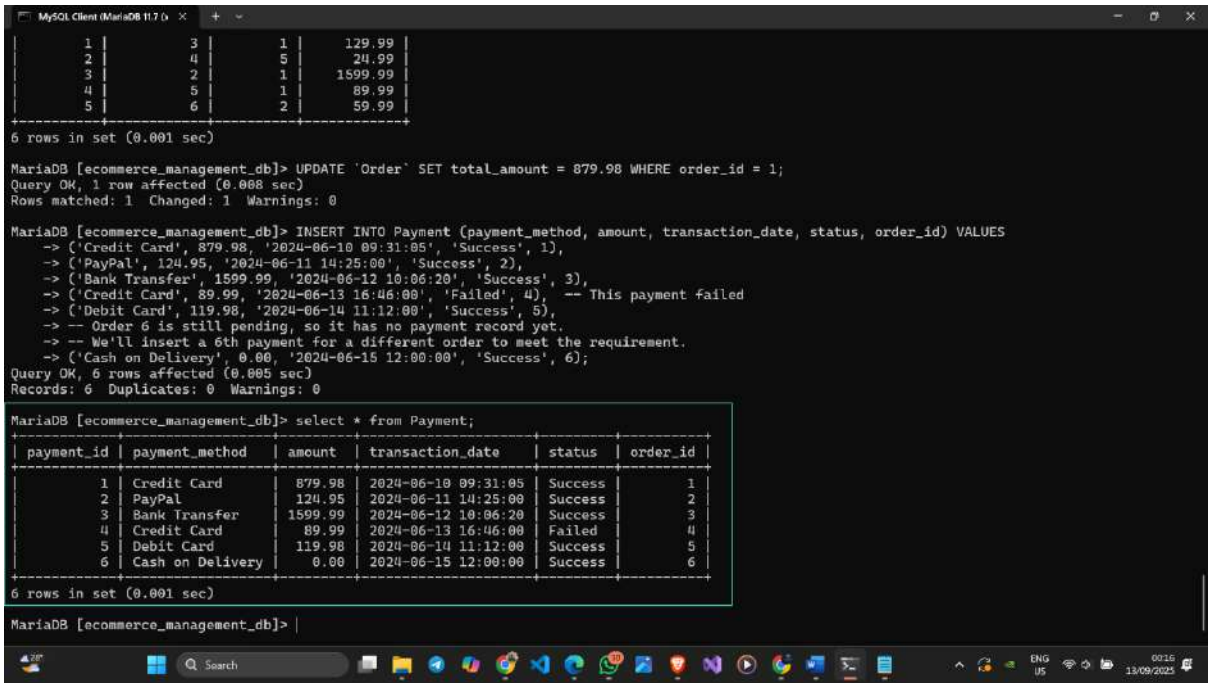

Figure 32 Payment Table with Inserted Data

4.4 Data Modification (UPDATE Operations)

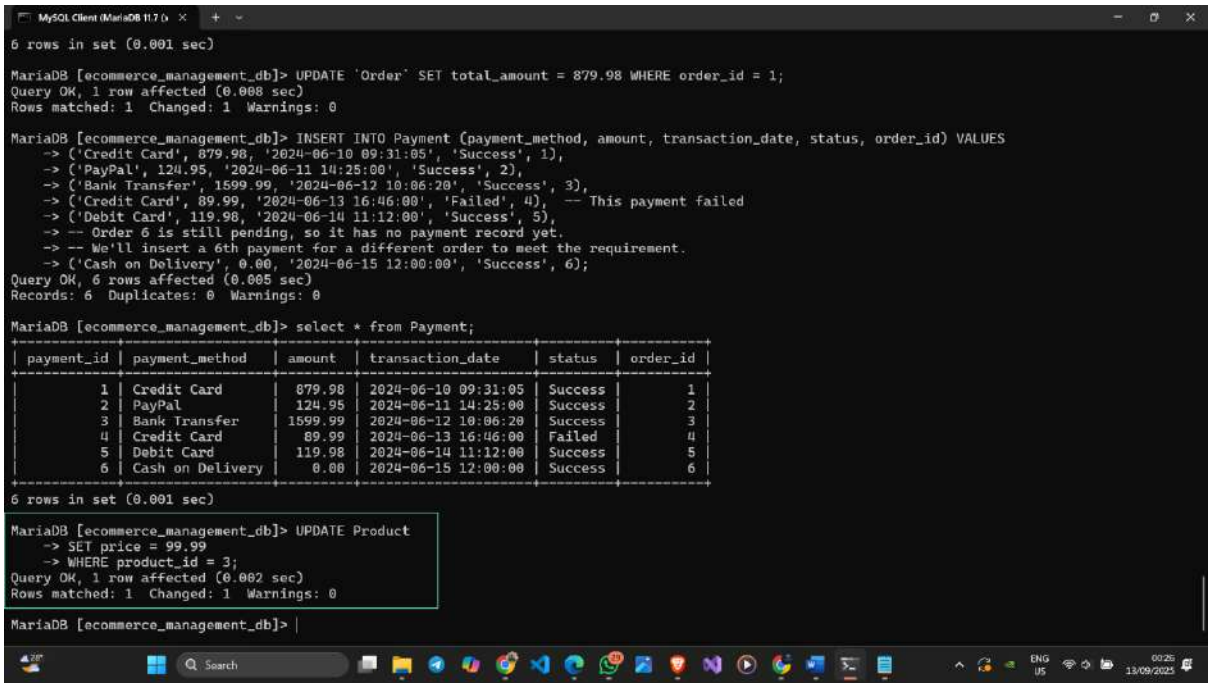

Figure 33 Update a Product's Price

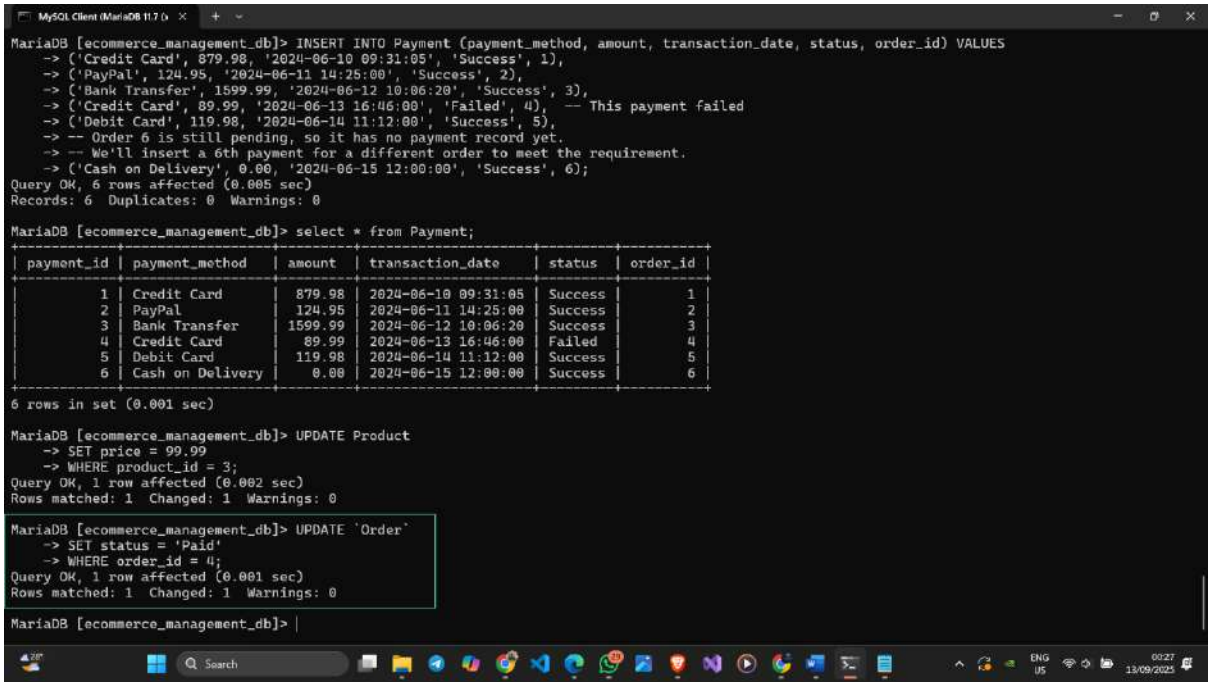

Figure 34 Update an Order's Status

4.5 Data Deletion (DELETE Operations)

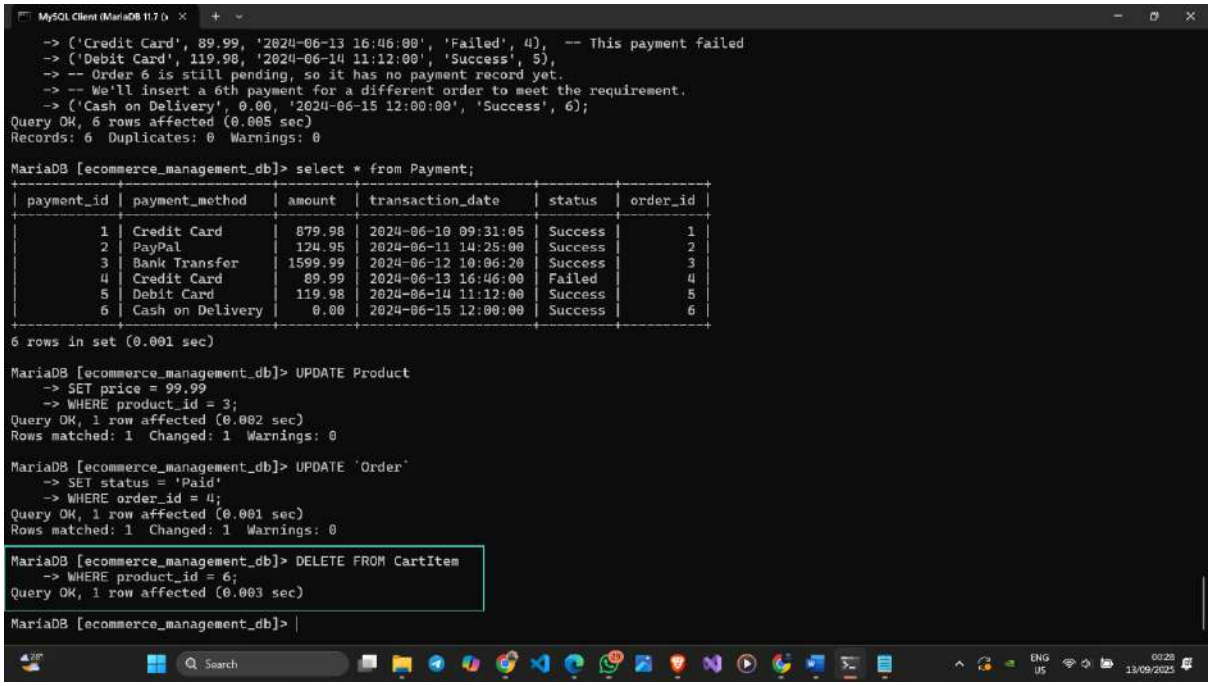

Figure 35 Delete from CartItem Table

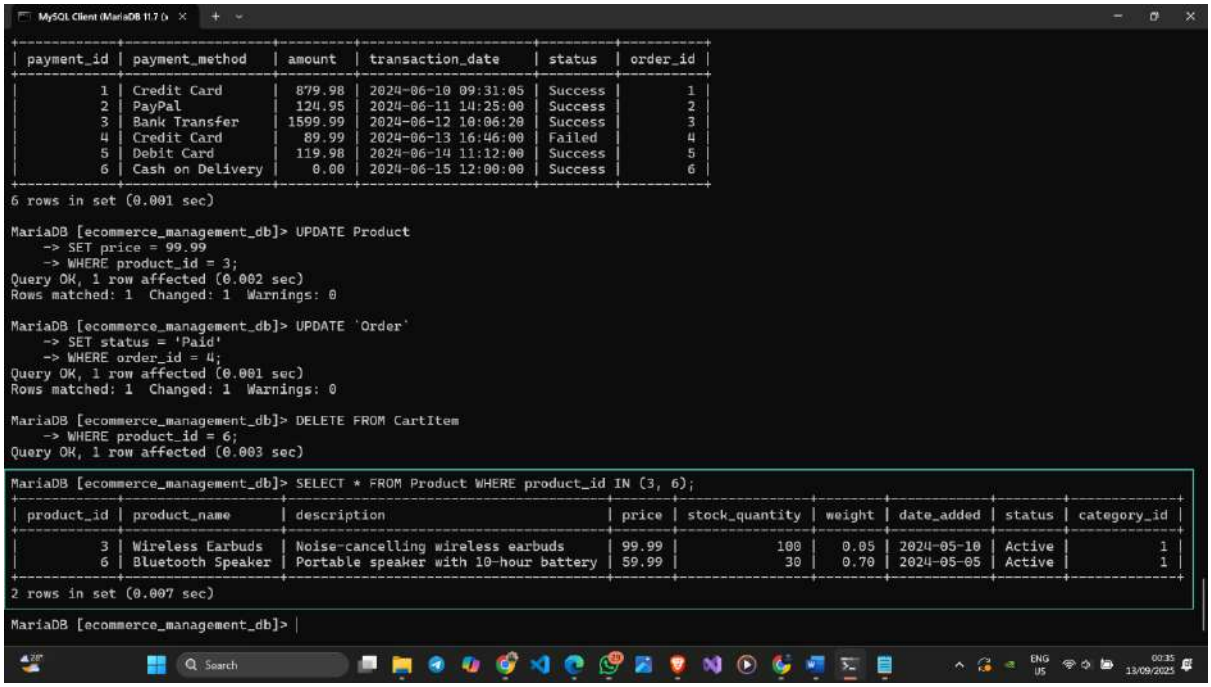

Figure 36 Verify the first Update operation

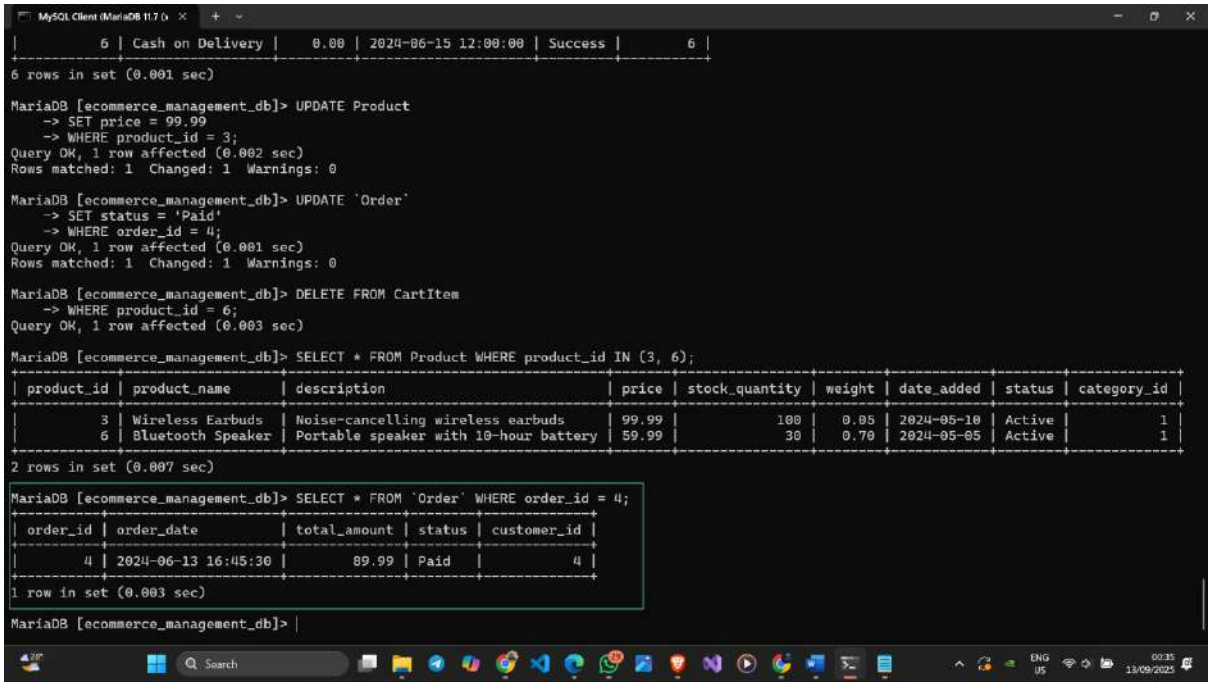

Figure 37 Verify the second Update operation

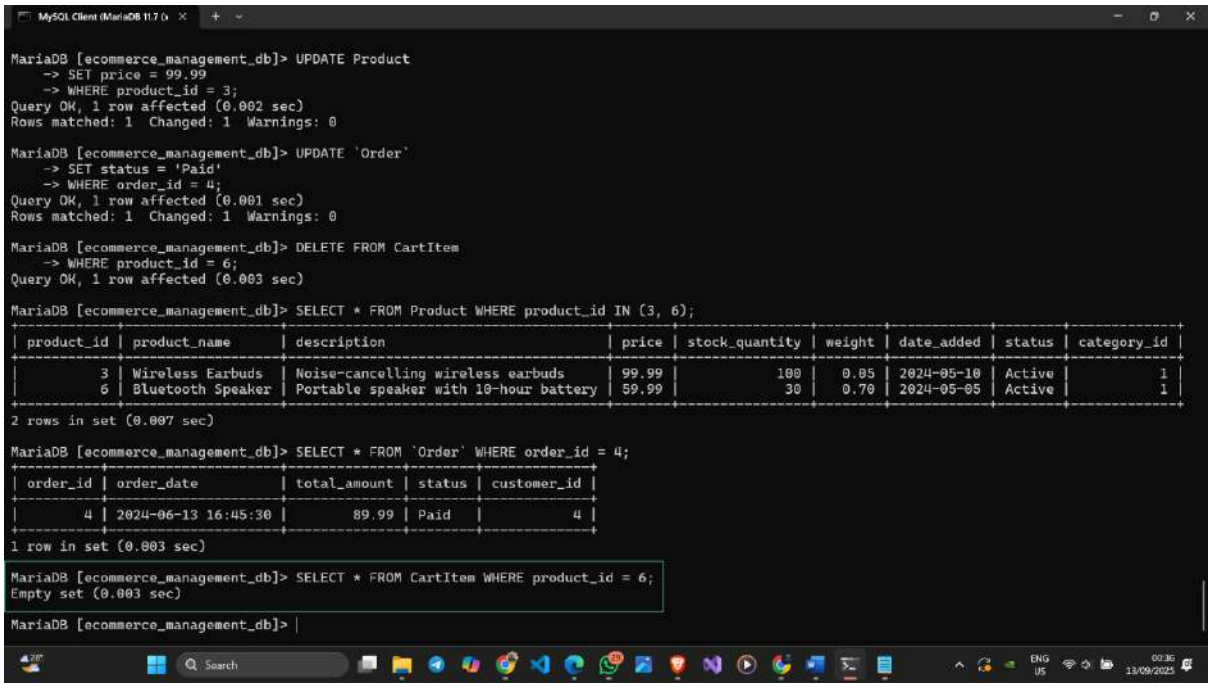

Figure 38 Verify the Delete operation

CHAPTER 5 – TRANSACTIONS

5.1 Simple Queries

5.1.1 Select Operation

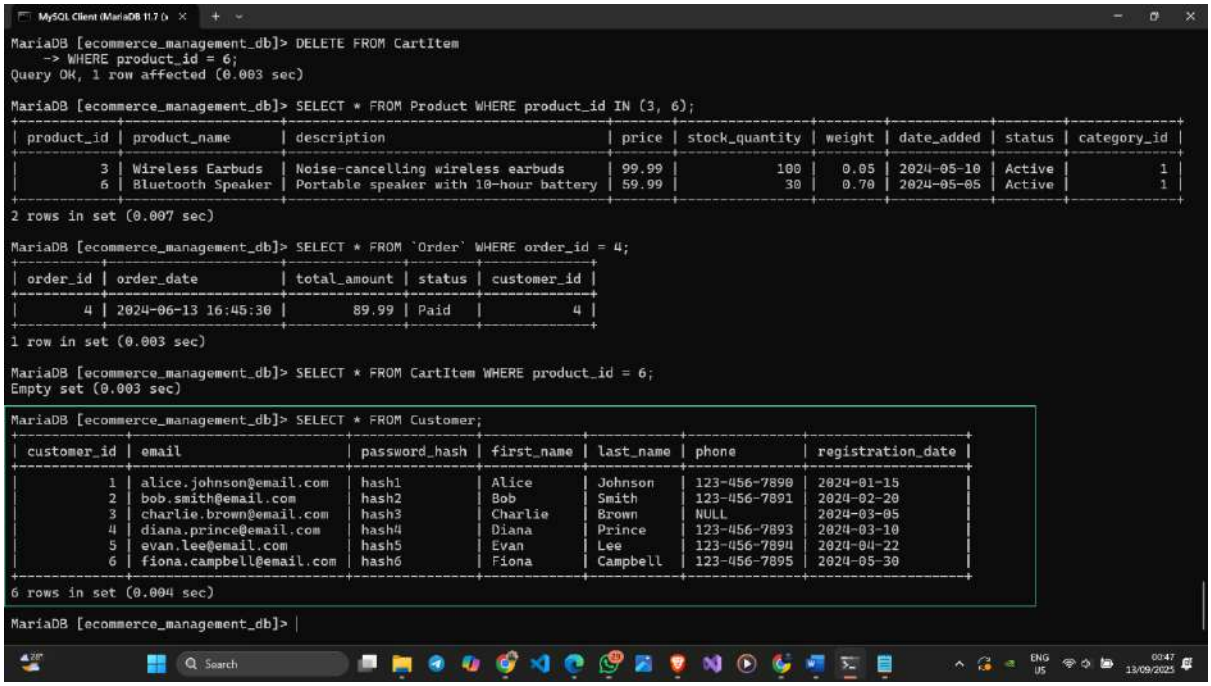

Figure 39 Select all columns from Customer Table

5.1.2 Project Operation

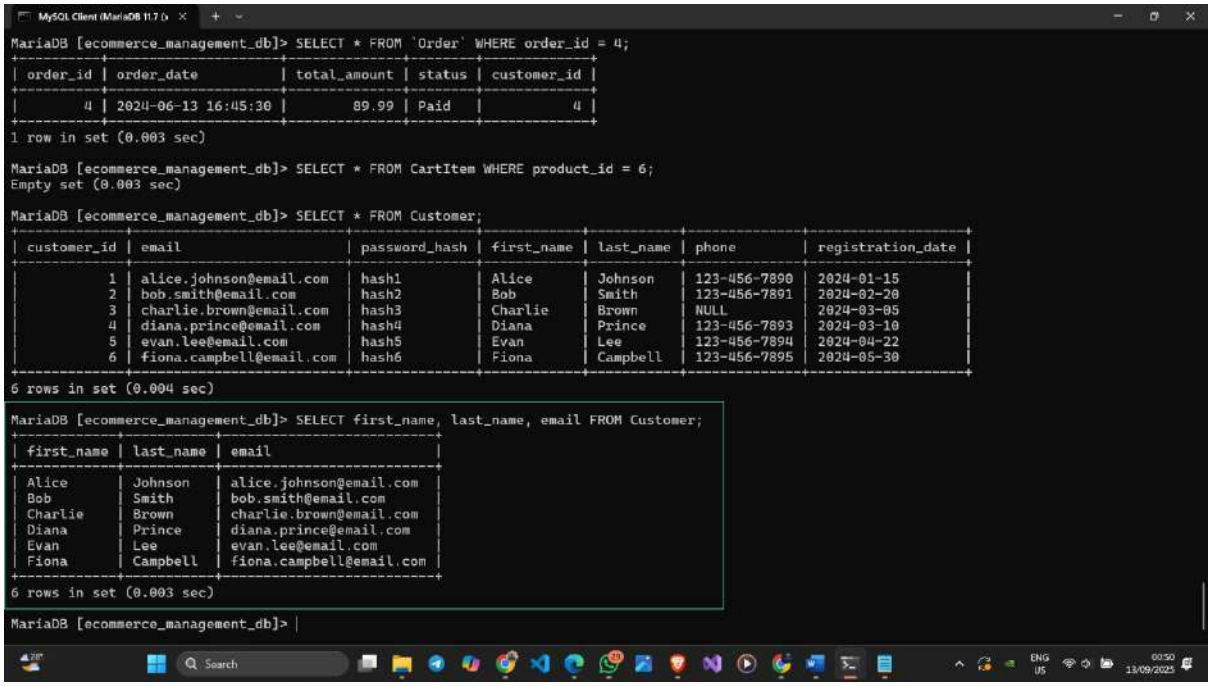

Figure 40 Select specific columns from Customer Table

5.1.3 Cartesian Product

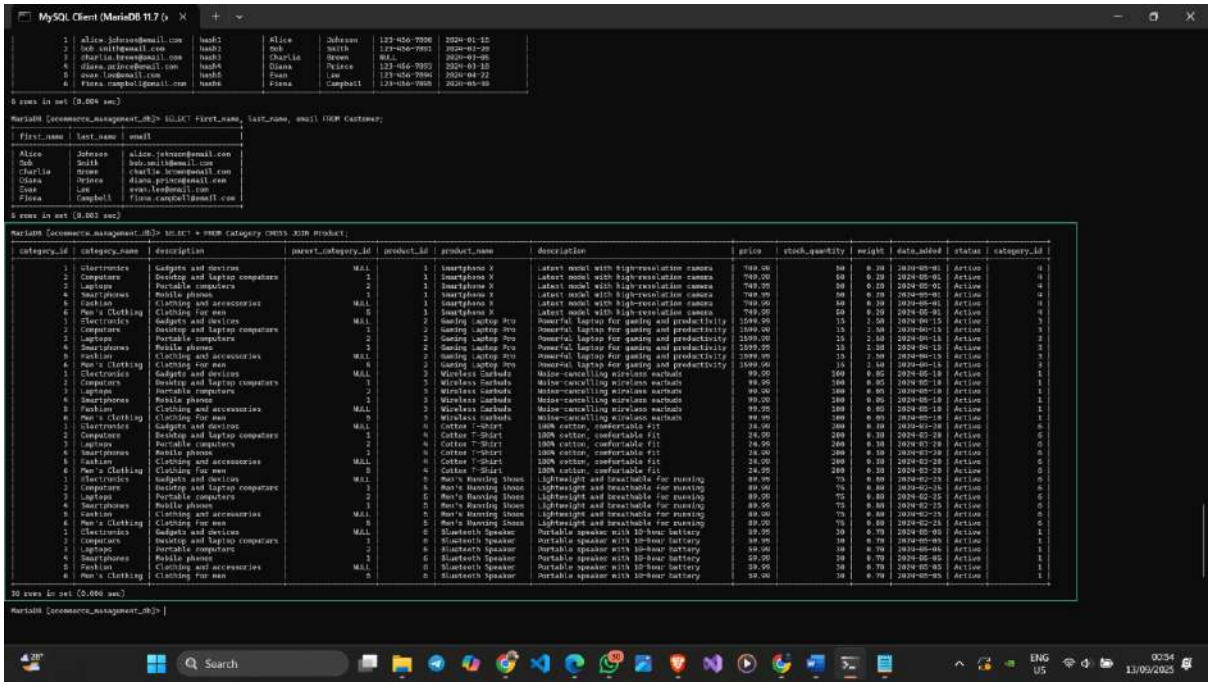

Figure 41Cartesian product between Category and Product Tables

5.1.4 Create View

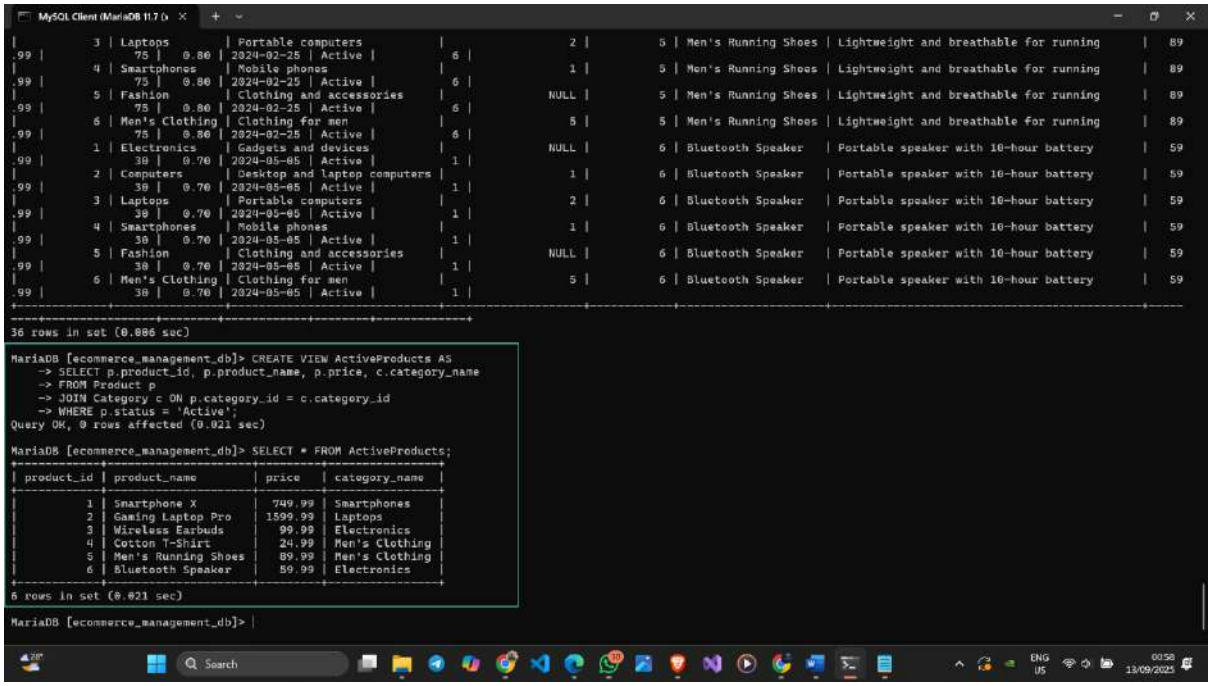

Figure 42 View for active products with their categories

5.1.5 Rename Operation

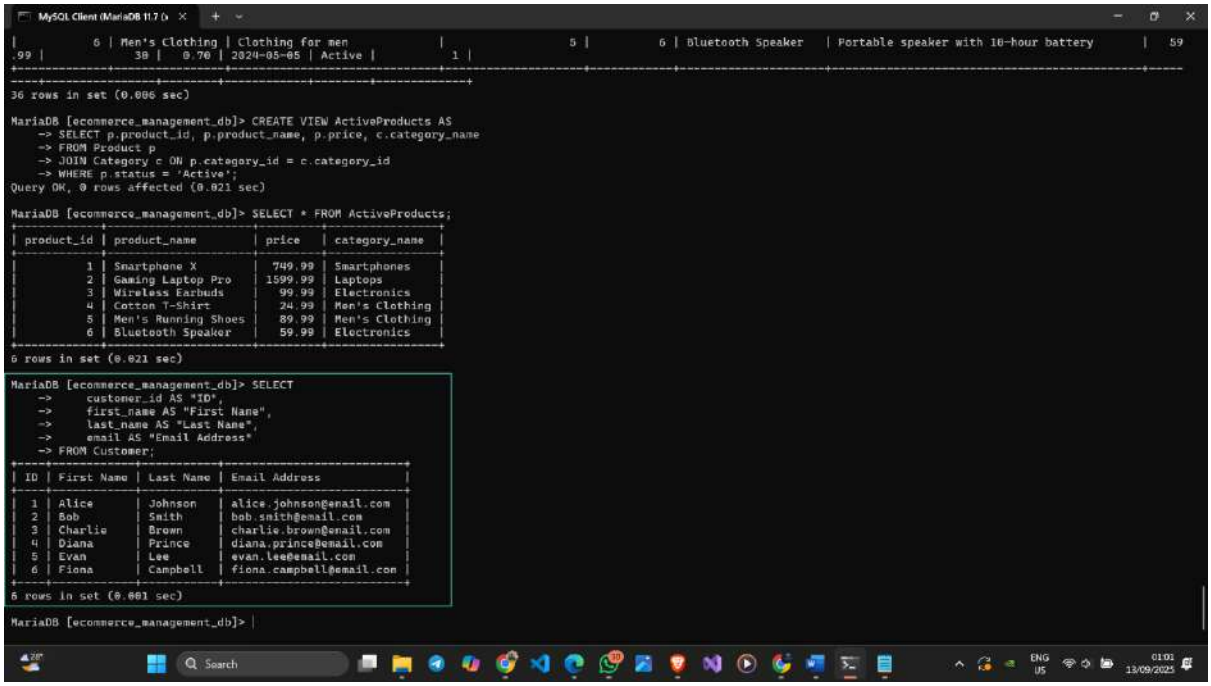

Figure 43 Rename Operation in Customer Table

5.1.6 Aggregate Function

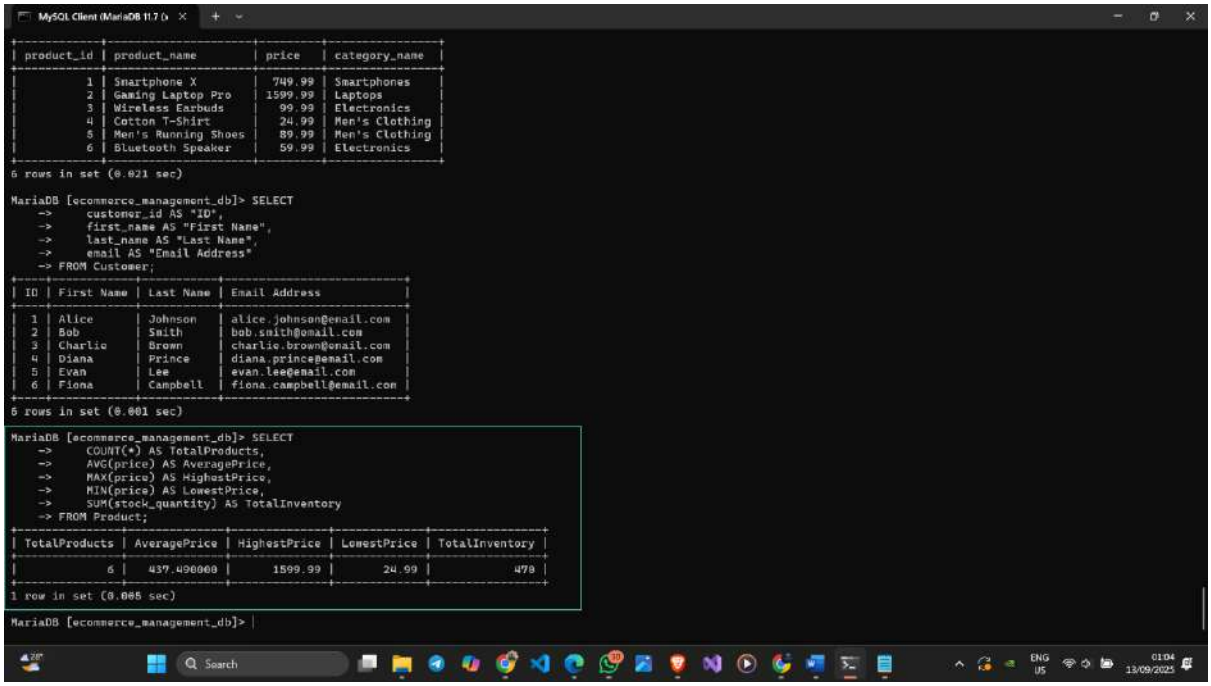

Figure 44 Aggregate Functions in Product Table

5.1.7 LIKE Keyword

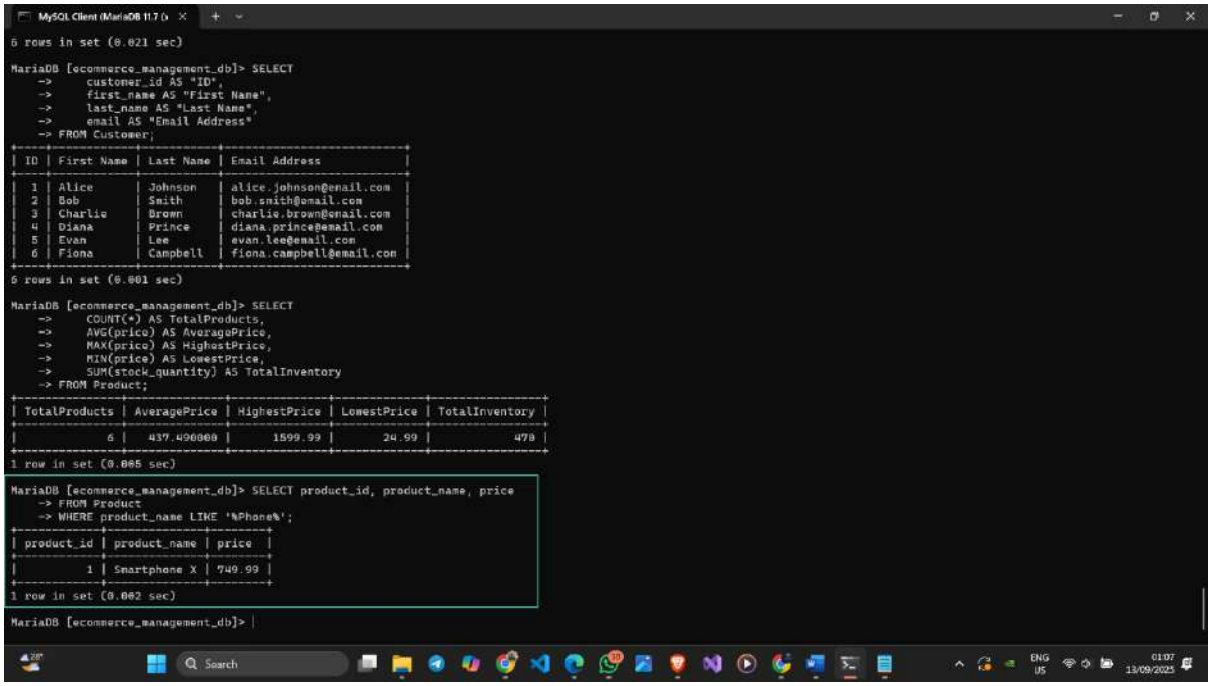

Figure 45 Use of LIKE in Product Table

5.2 Complex Queries

5.2.1 Set Operations (Union, Intersection, Set Difference, Division)

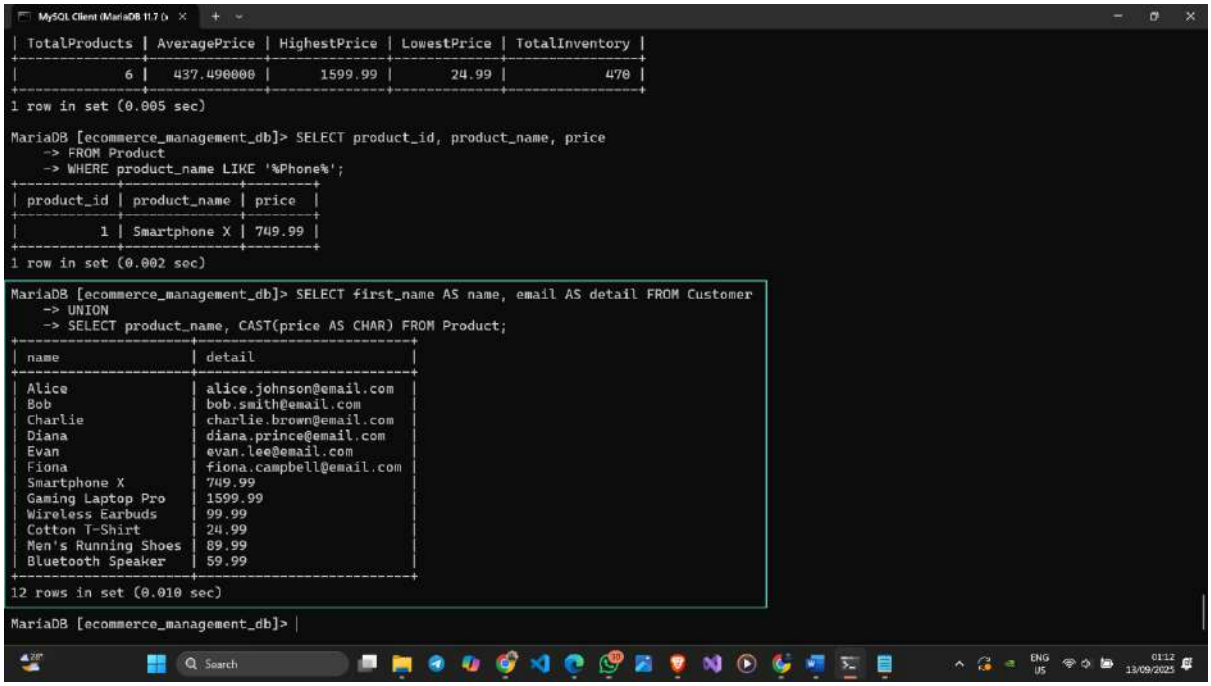

Figure 46 Union Operation

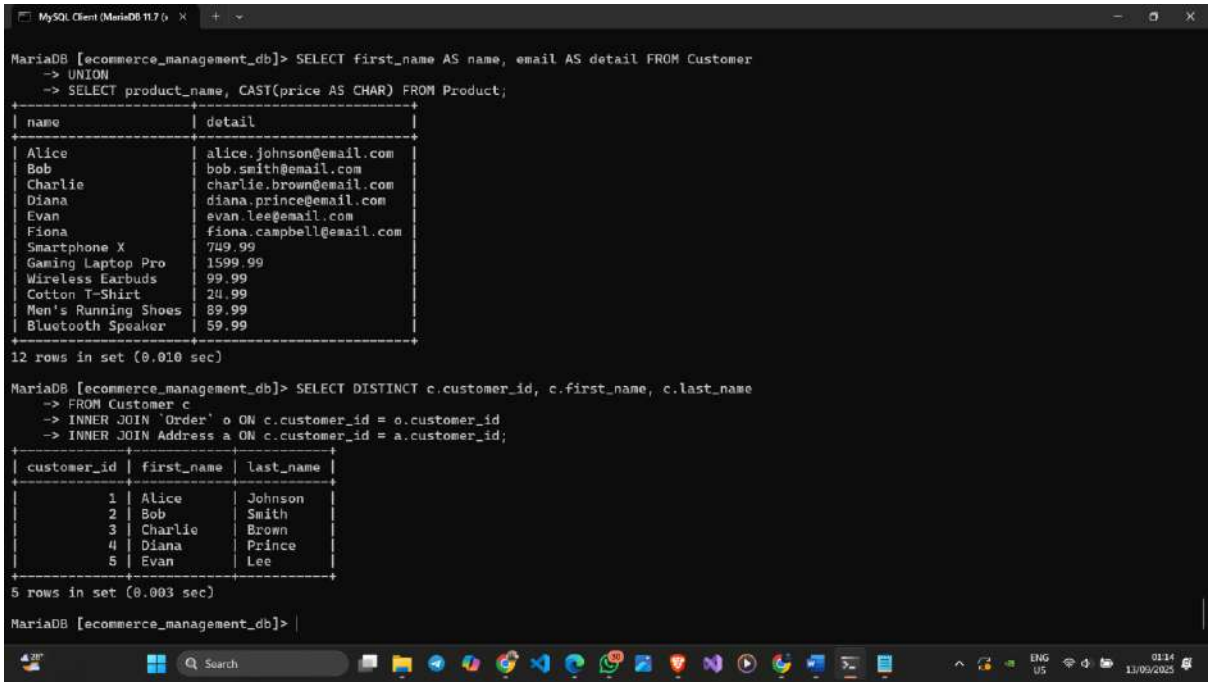

Figure 47 Intersection Operation

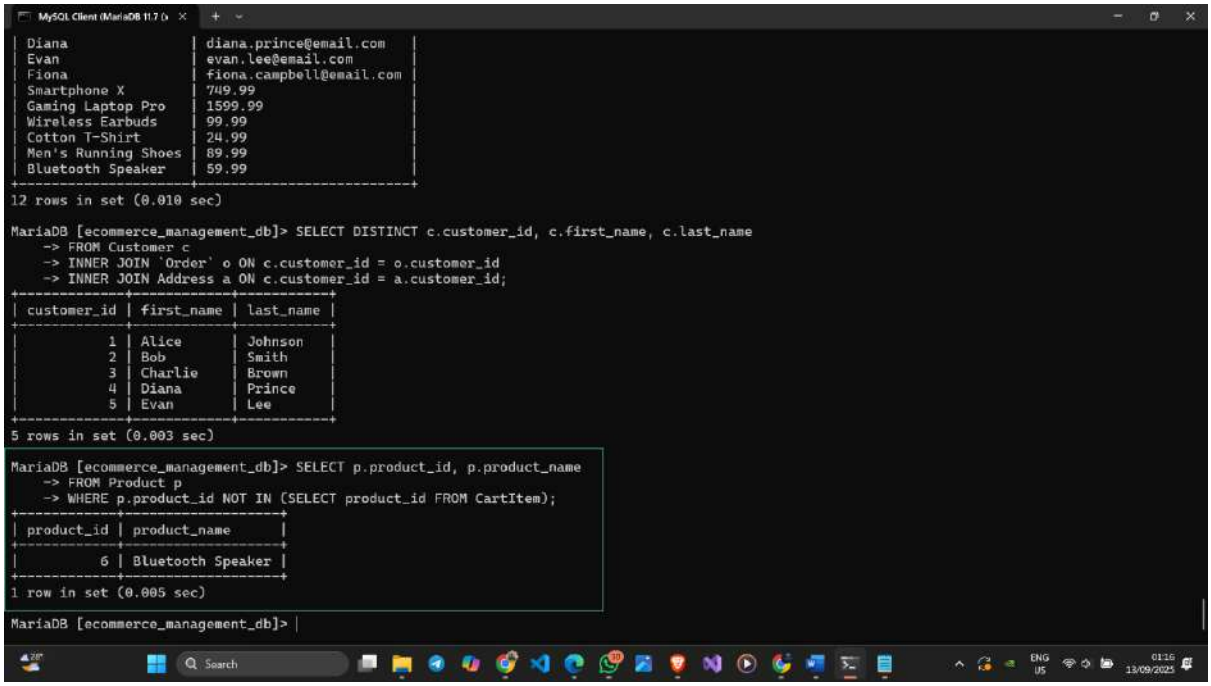

Figure 48 Set Difference Operation

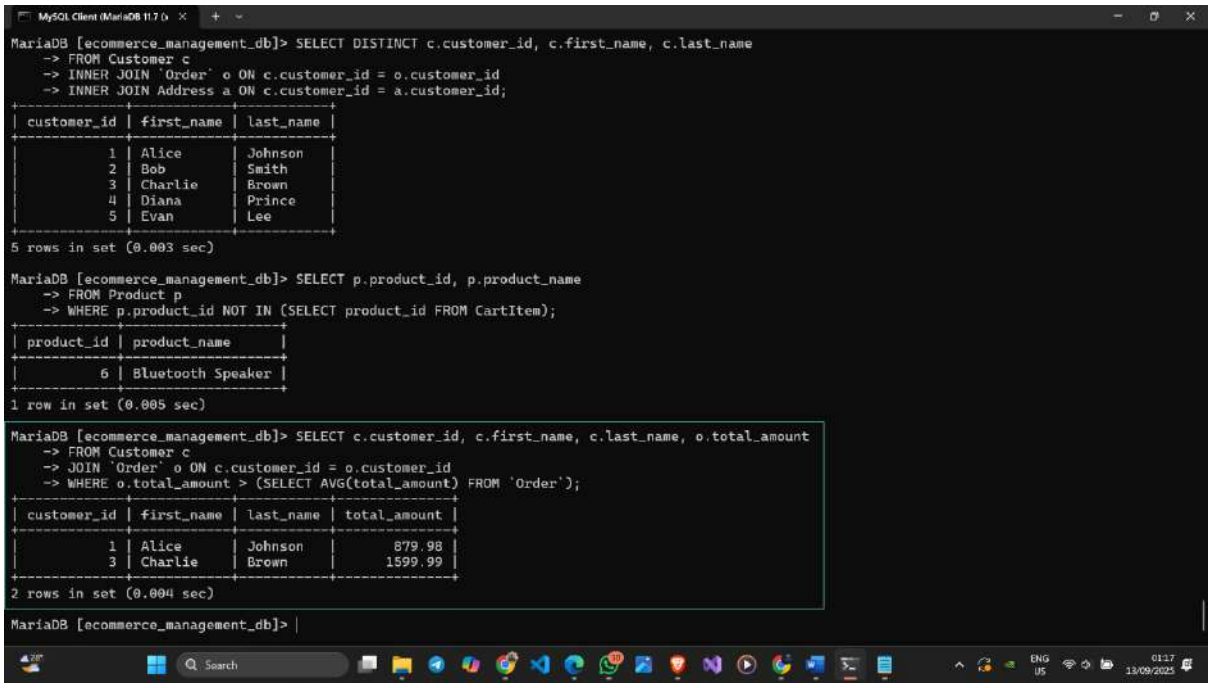

Figure 49 Division Operation

5.2.2 Join Operations (Inner, Natural, Left Outer, Right Outer, Full Outer)

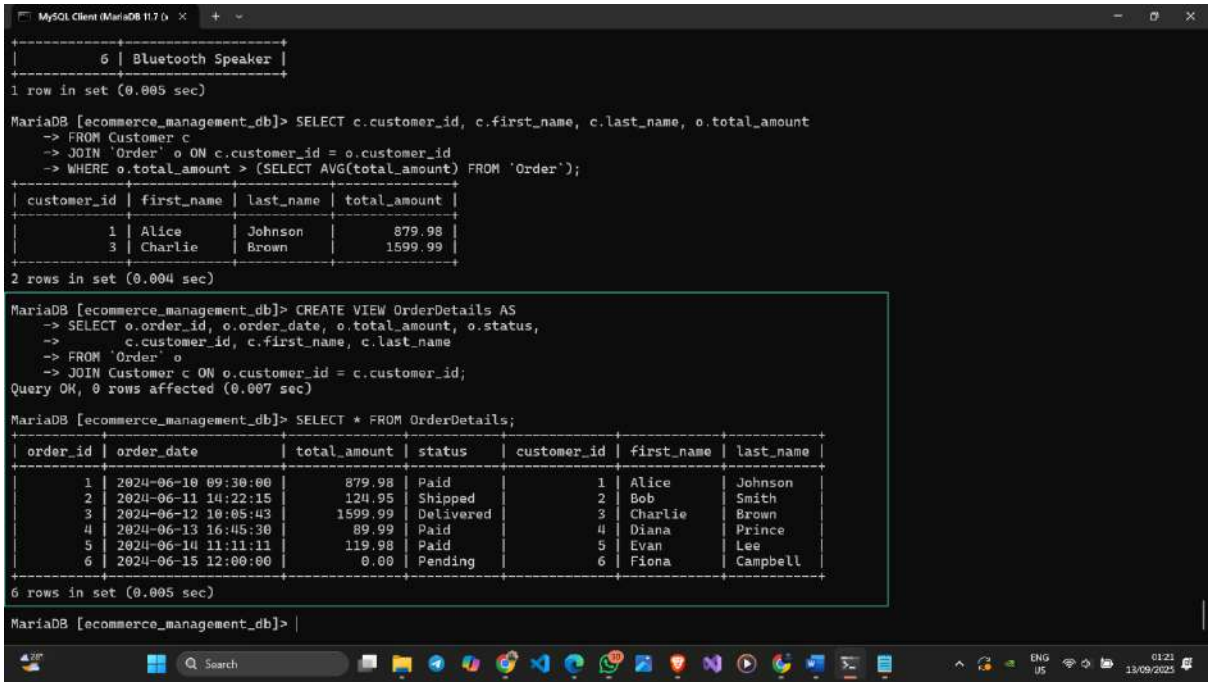

Figure 50 Inner Join Operation

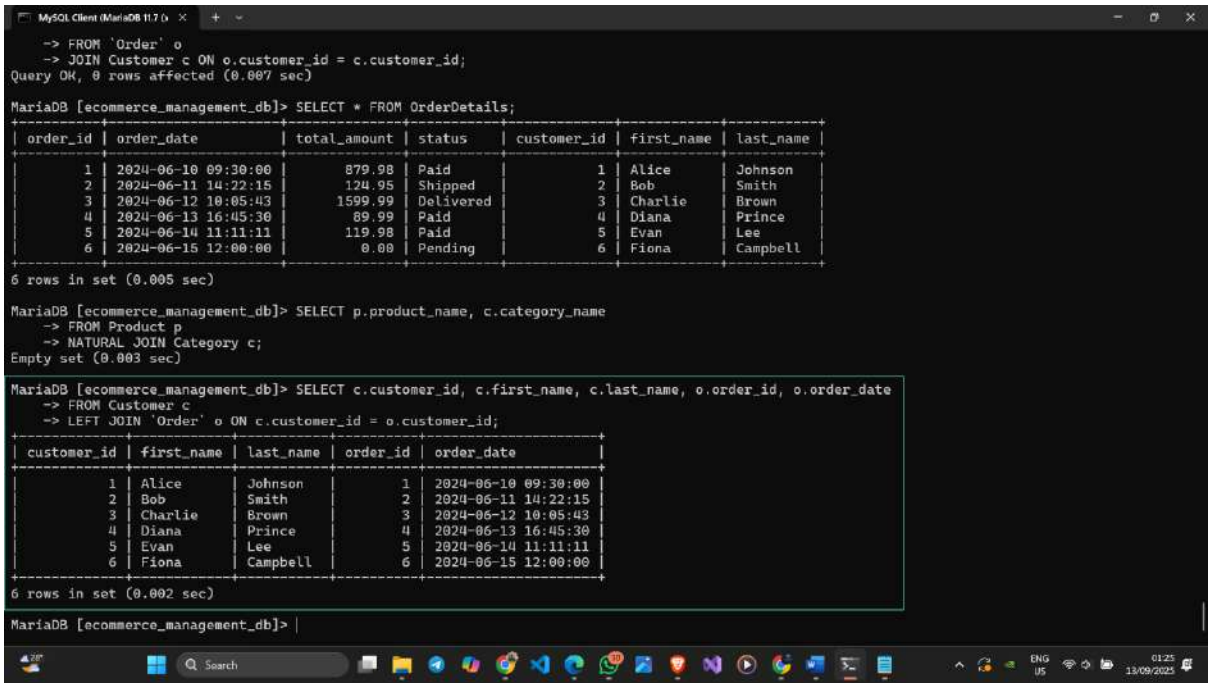

Figure 51 Left Outer Join Operation

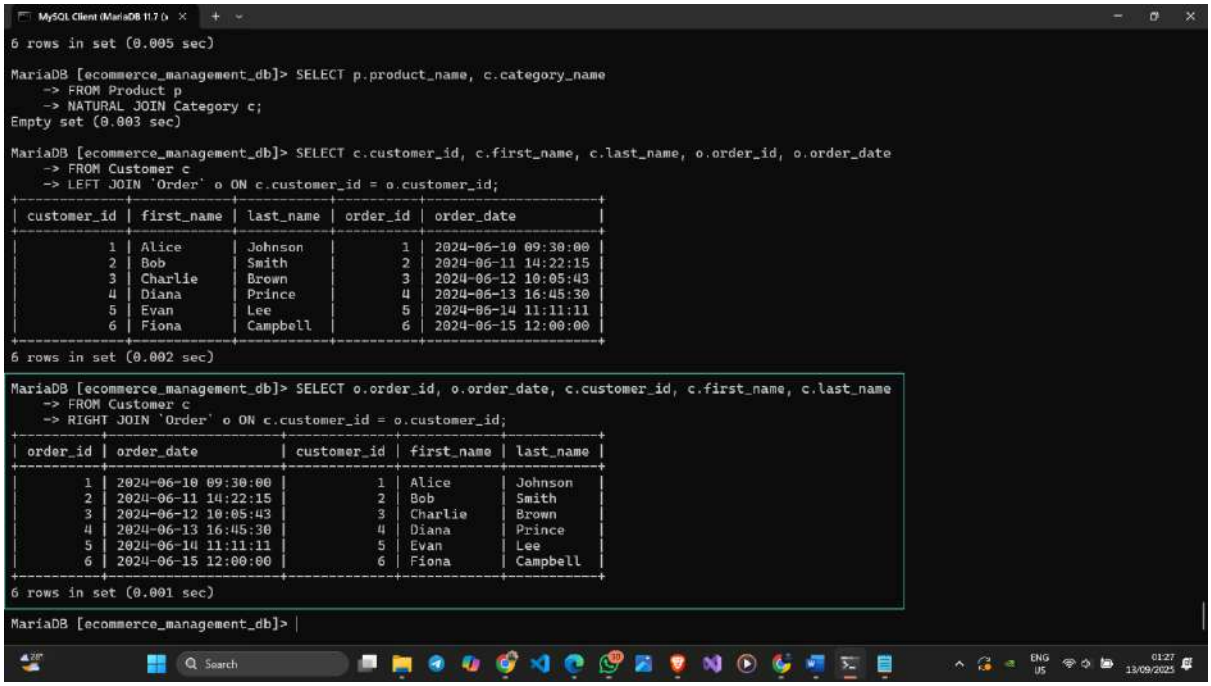

Figure 52 Right Outer Join Operation

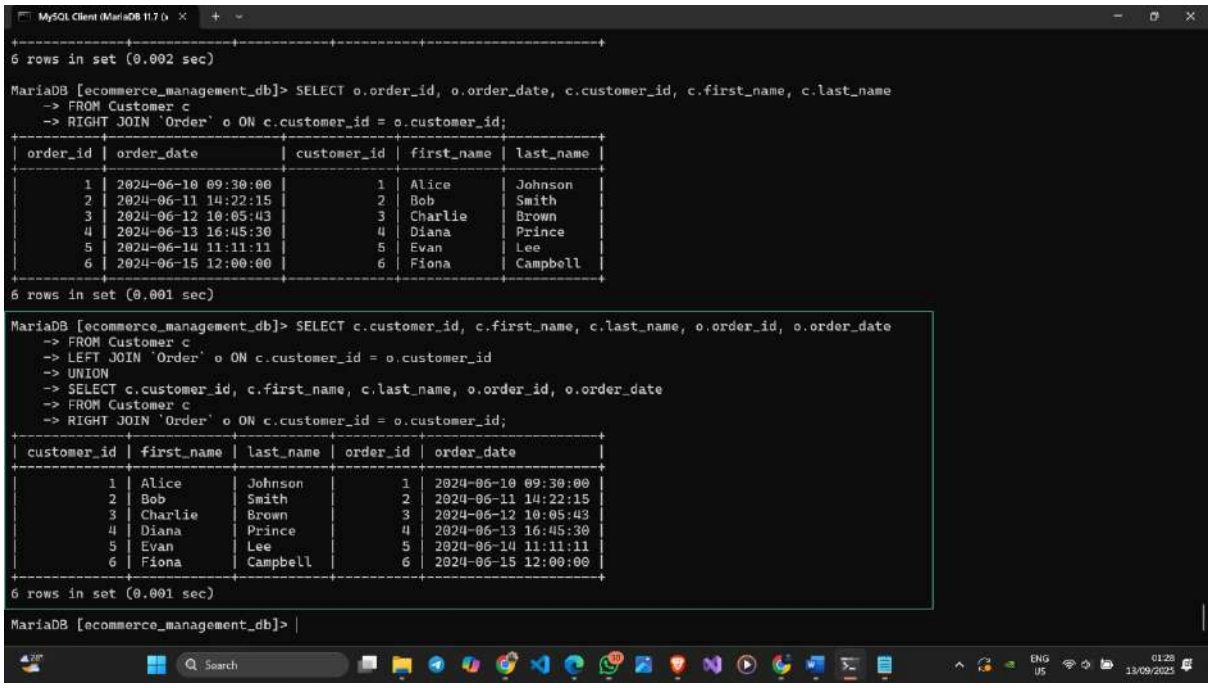

Figure 53 Full Outer Join Operation

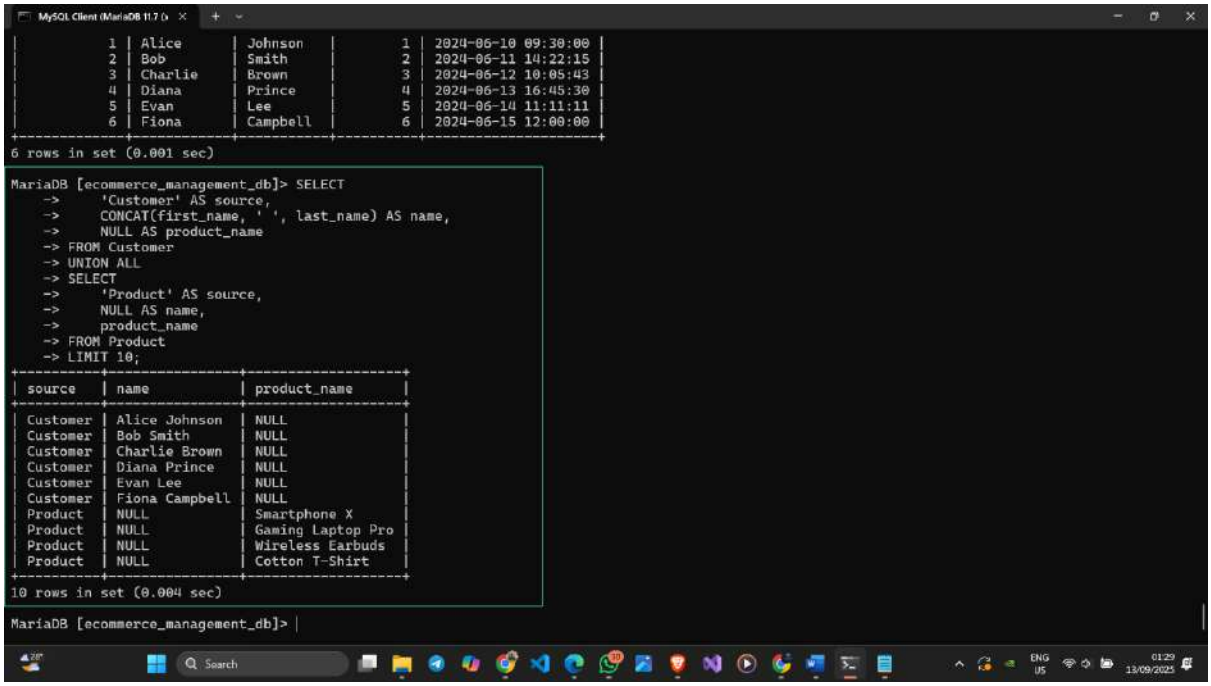

Figure 54 Outer Union Operation

5.2.3 Nested Queries

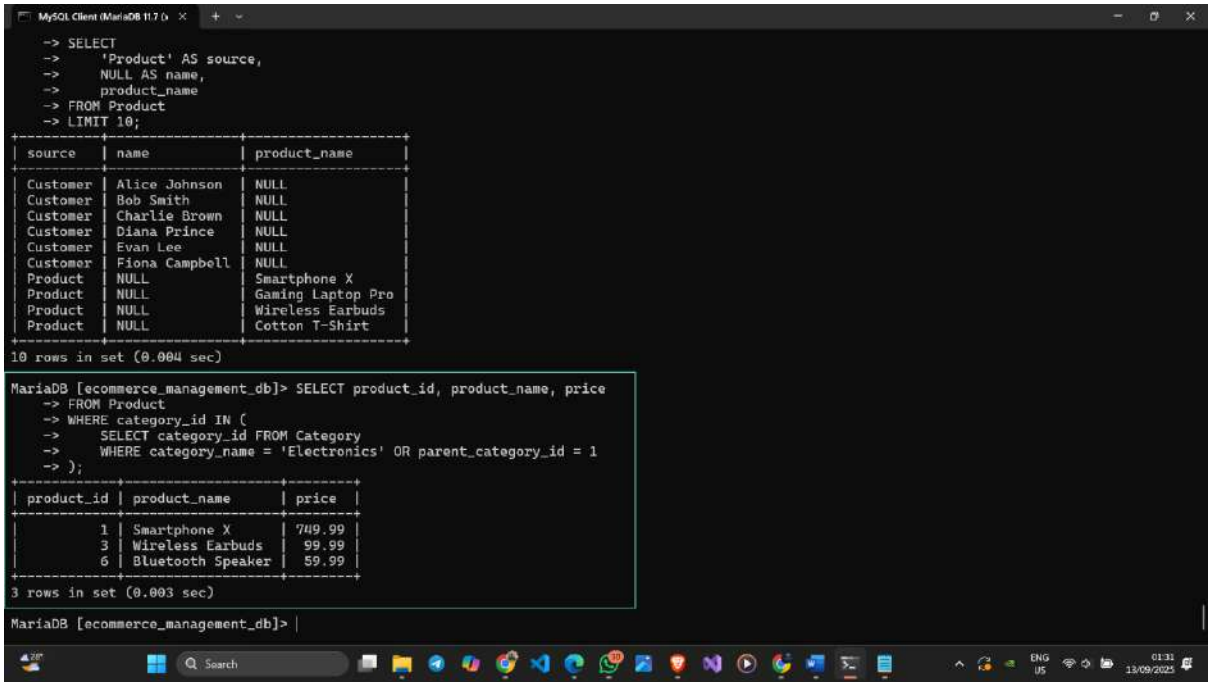

Figure 55 Nested Query with IN

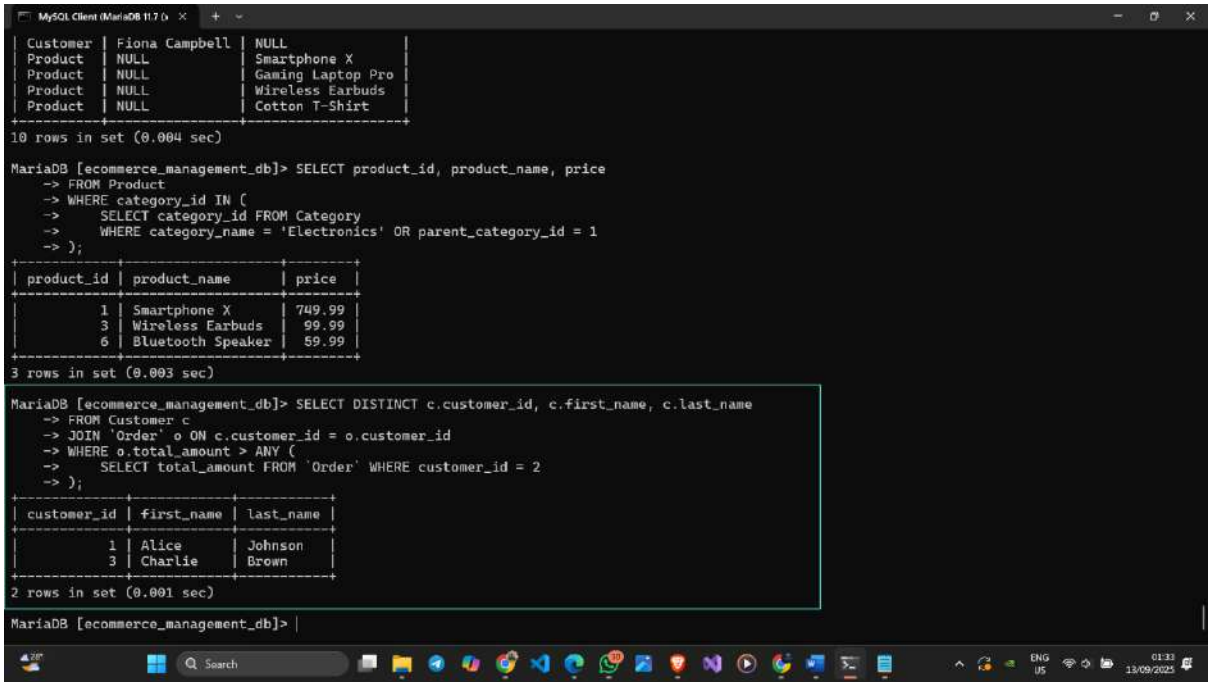

Figure 56 Nested Query with ANY

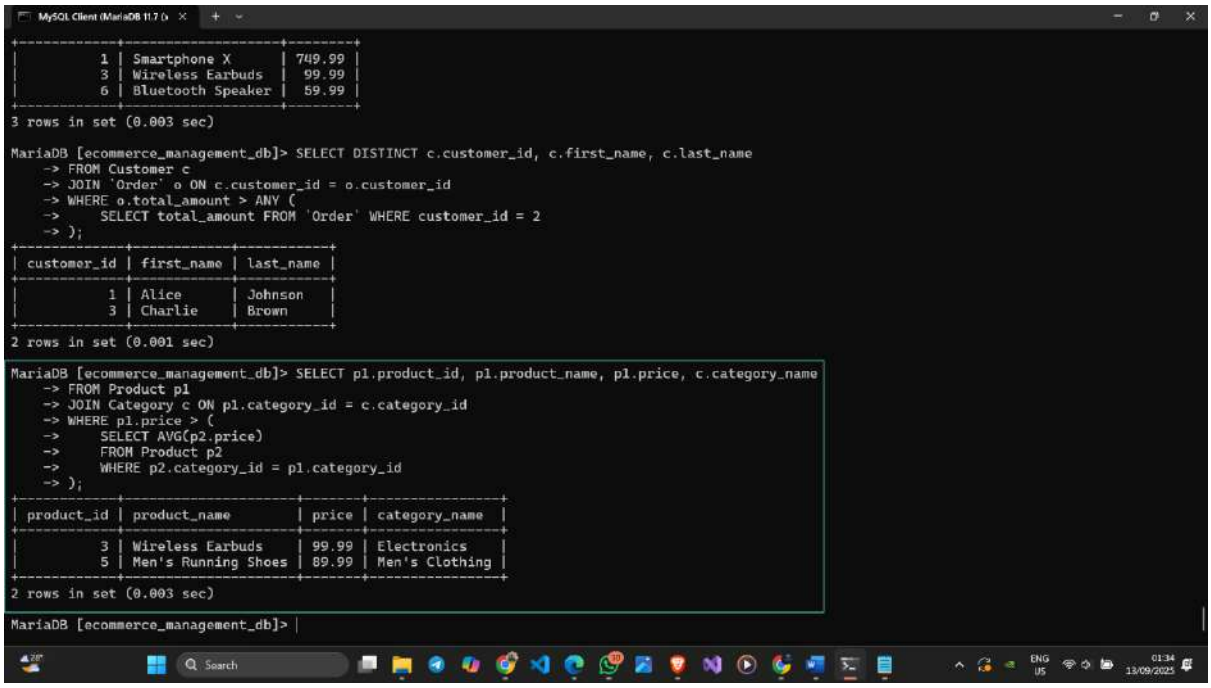

Figure 57 Correlated Nested Query

CHAPTER 6 - DATABASE TUNING

6.1 Query Tuning Methodology

We will select 10 complex queries from Chapter 5 and attempt to tune them. The primary method will be adding **indexes** on columns used in WHERE, JOIN, and ORDER BY clauses. We will use the EXPLAIN command to analyze the query execution plan before and after tuning to prove the improvement. We will compare the number of rows scanned ("rows" in the EXPLAIN output).

6.2 Analysis of Original Queries

First, we need to see how MariaDB currently executes the queries. We do this by putting EXPLAIN before the query.

6.3 Tuned Queries and Index Implementation

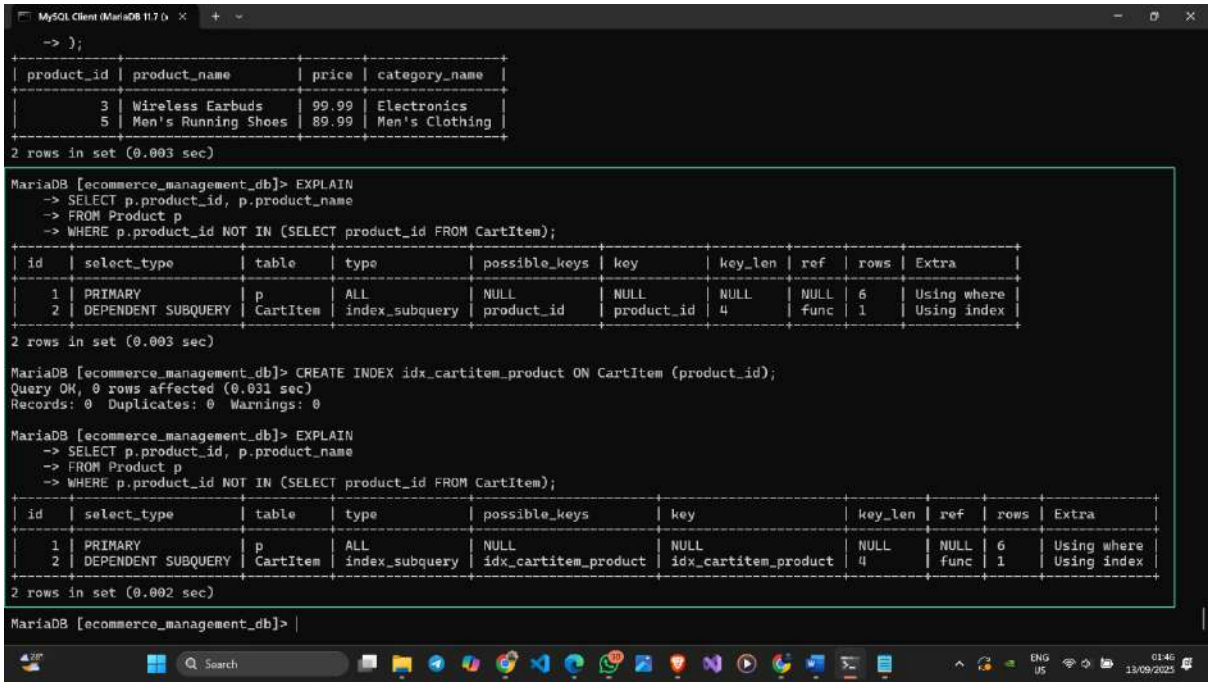

Figure 58 Before tunned and after tuned for query 1

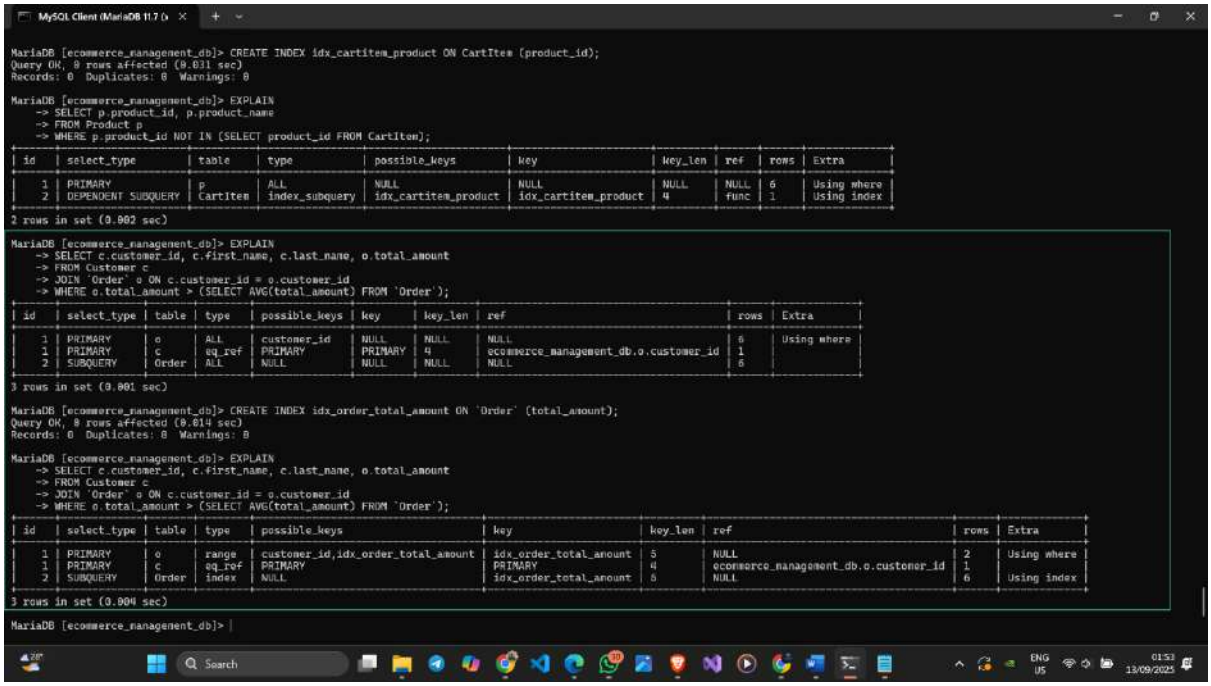

Figure 59 Before tunned and after tuned for query 2

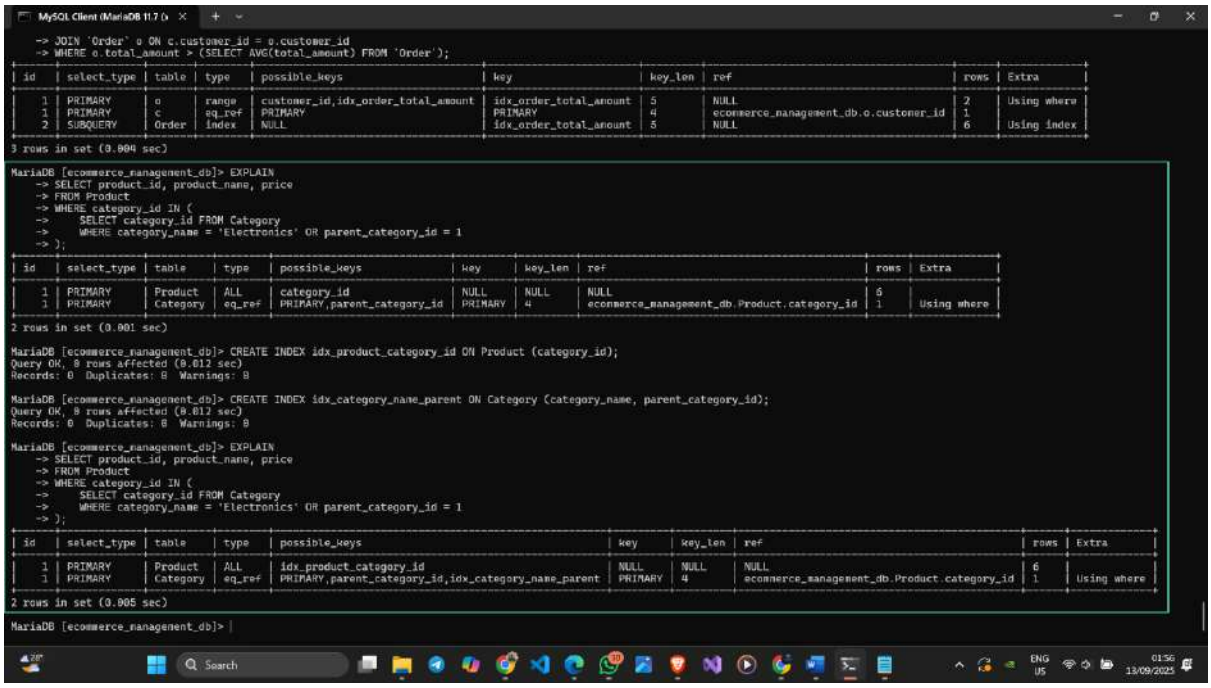

Figure 60 Before tunned and after tuned for query 3

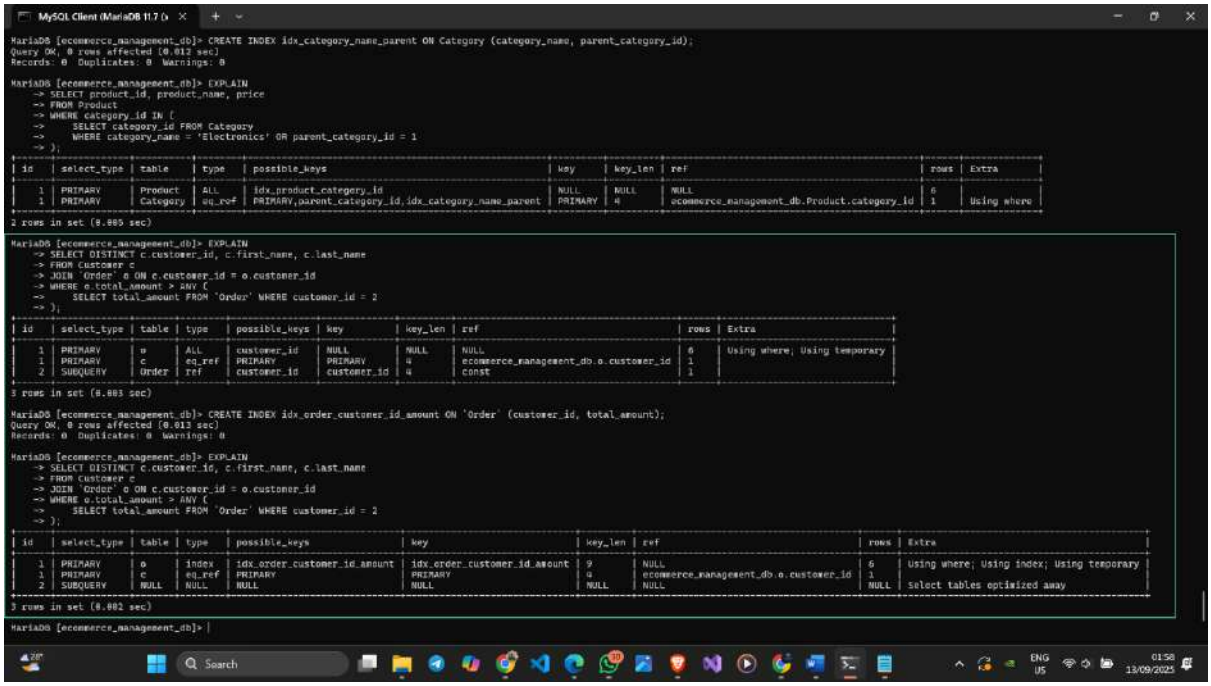

Figure 61 Before tunned and after tuned for query 4

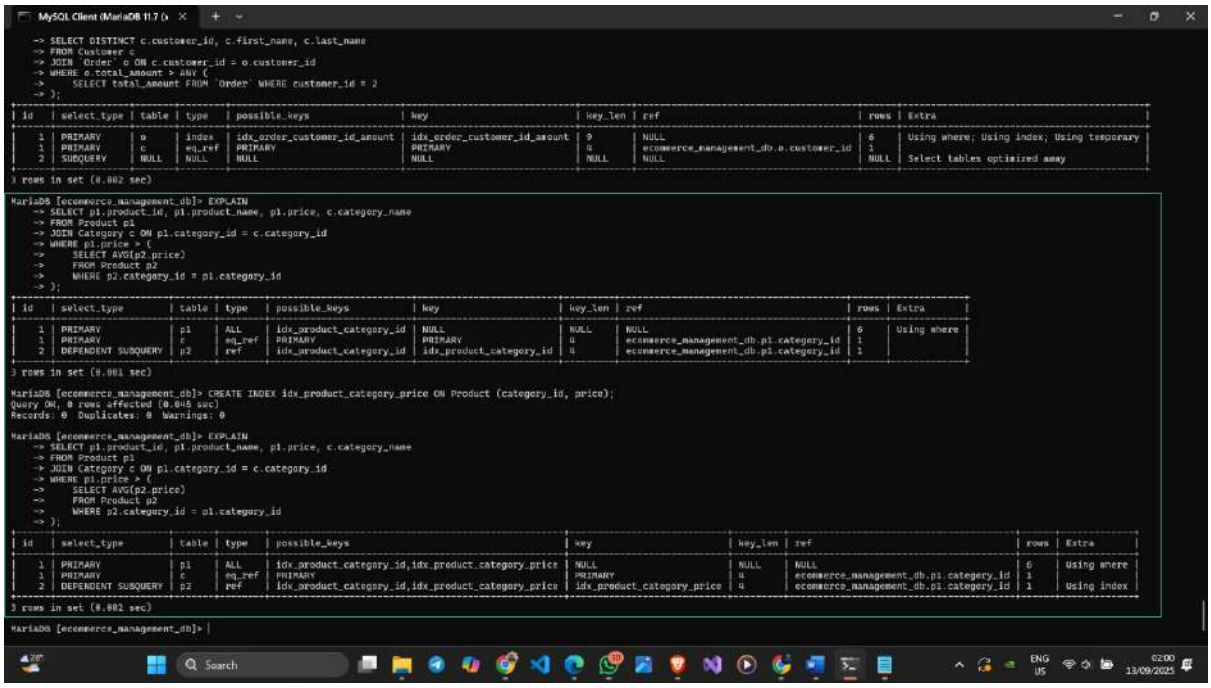

Figure 62 Before tunned and after tuned for query 5

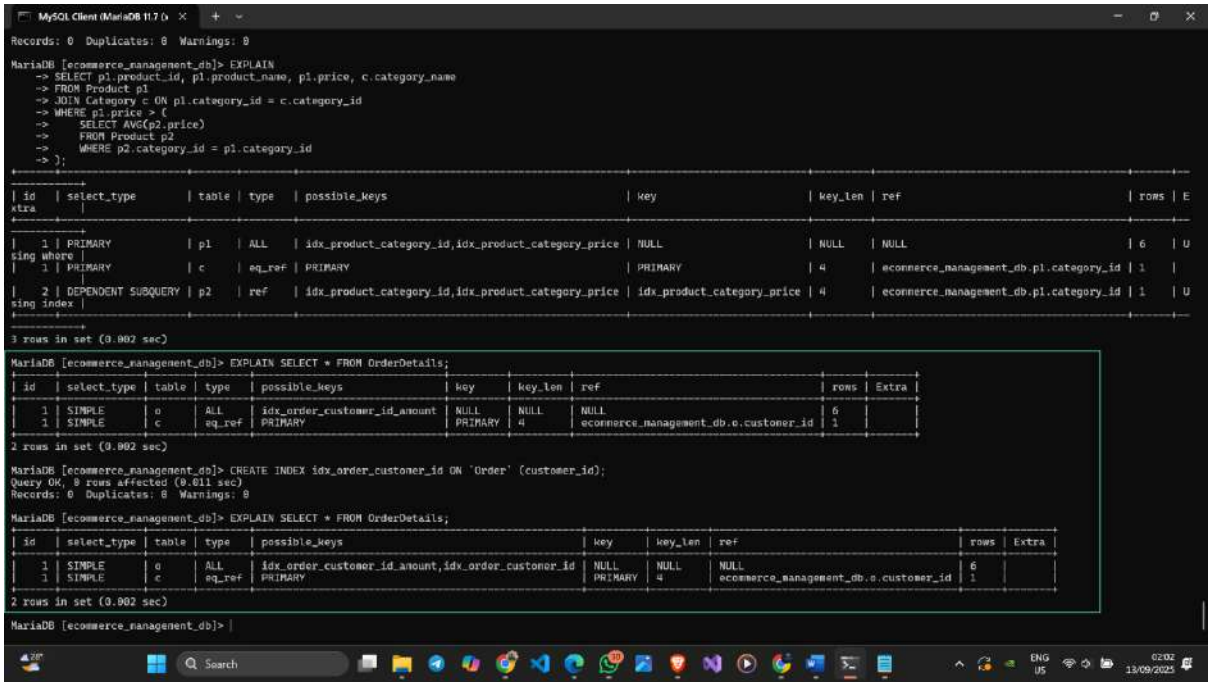

Figure 63 Before tunned and after tuned for query 6

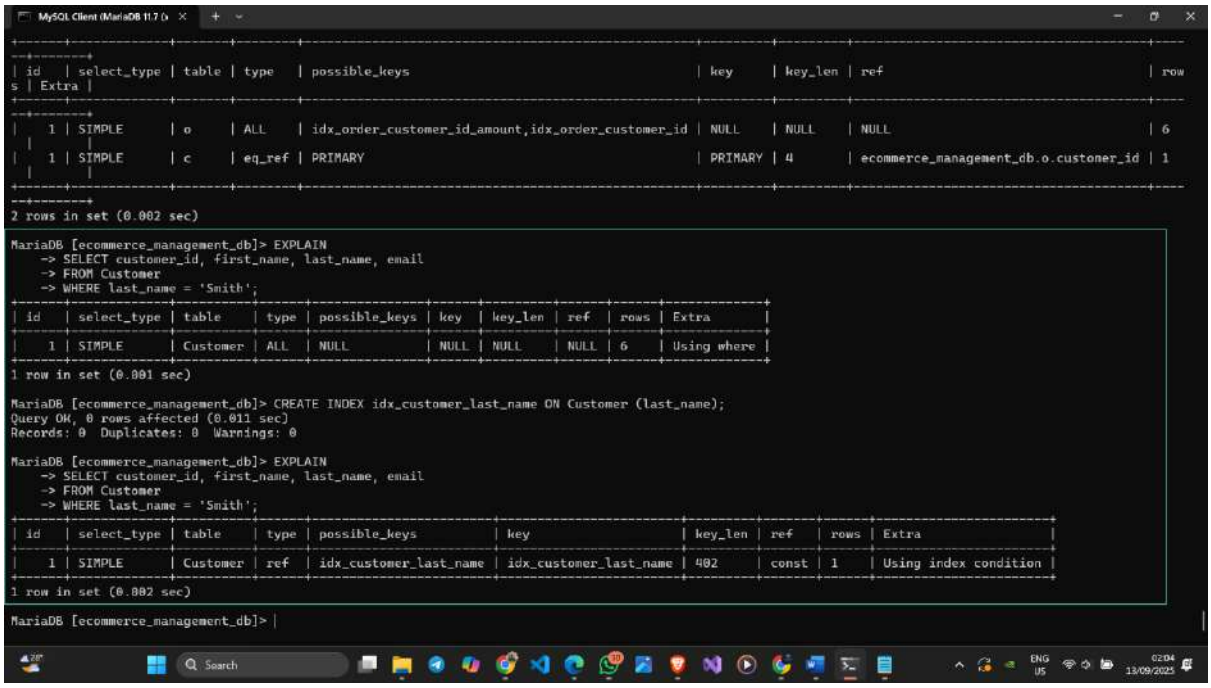

Figure 64 Before tunned and after tuned for query 7

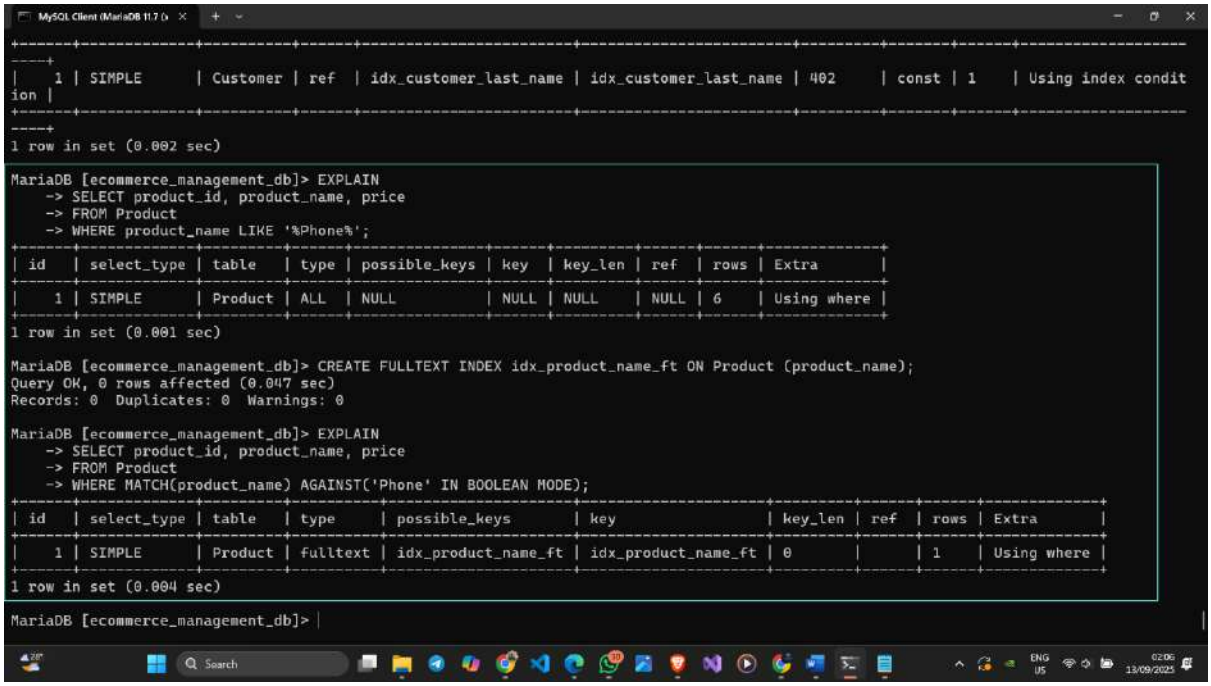

Figure 65 Before tunned and after tuned for query 8

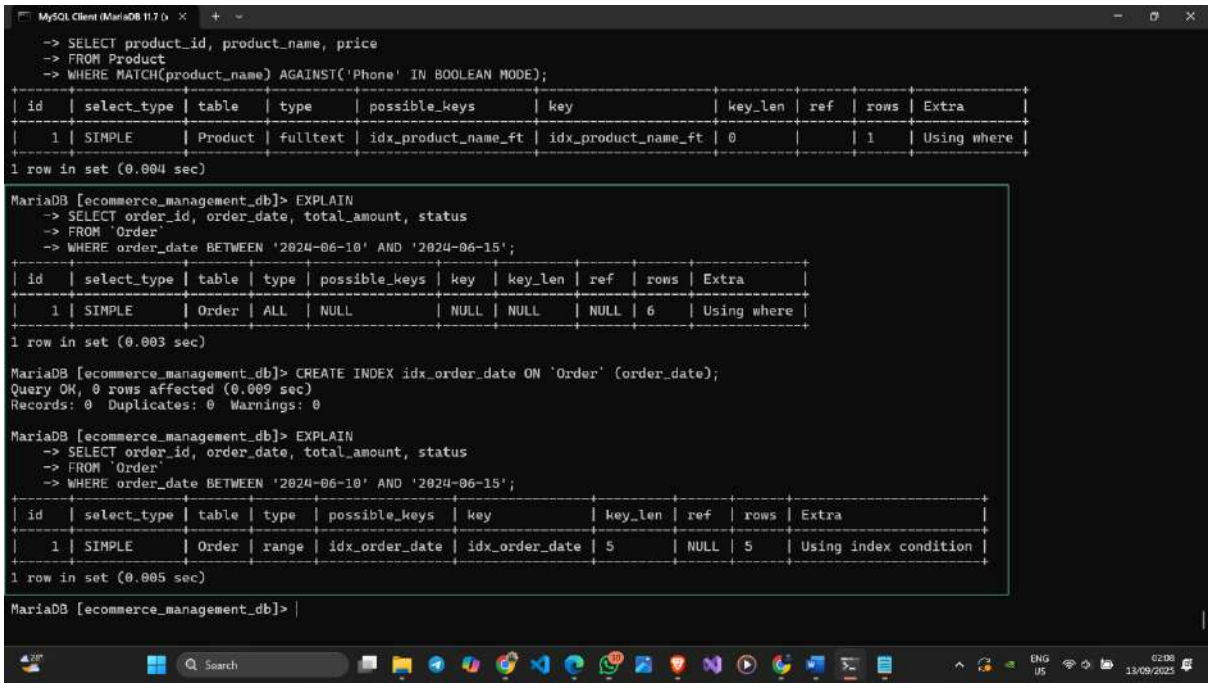

Figure 66 Before tunned and after tuned for query 9

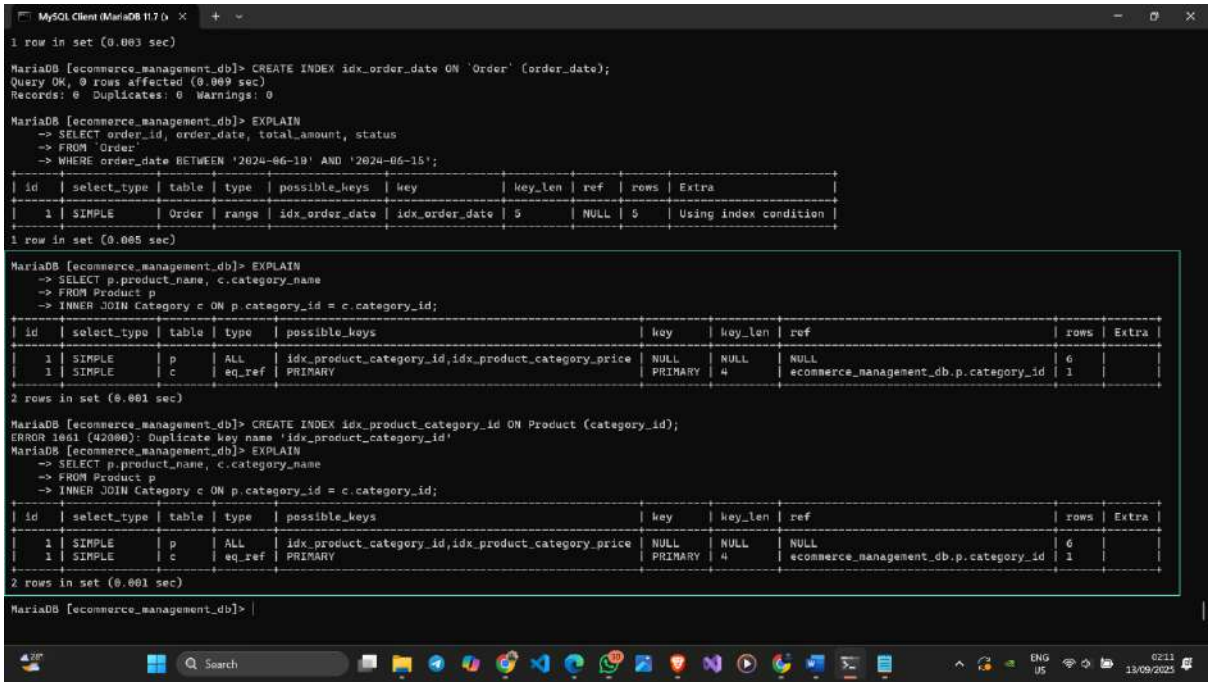

Figure 67 Before tunned and after tuned for query 10

6.4 Performance Comparison

Example: This query finds products never added to a cart. Initially, MariaDB had to perform a full table scan on the CartItem table, examining all 6 rows. After creating an index on CartItem(product_id), the database could use this index to quickly locate the required values, reducing the number of examined rows to 1. This significantly improves the query's performance.

Like this etc…

PART B – NoSQL DATABASES

Chapter 1: Aggregation model

1.1 Bucket Creation and Data Insertion

The aggregation model for this project is implemented as a key-value store using PostgreSQL's JSONB data type. This approach treats a unique integer primary key as the main key and a flexible JSONB document as the value, allowing for a variable number of attributes per record. This schema-less design is well-suited for a NoSQL approach within a relational database context.

The following operations were performed to demonstrate this model:

- Creation of the Customer_EE00 bucket/collection: A table was created to store customer data.
- Data Insertion: Ten rows with diverse customer information were inserted.
- Data Retrieval: Two queries were executed to retrieve specific data aggregates, proving the model's functionality.

The subsequent screenshots in your report will visually document these steps, from table creation and data population to the final query results.

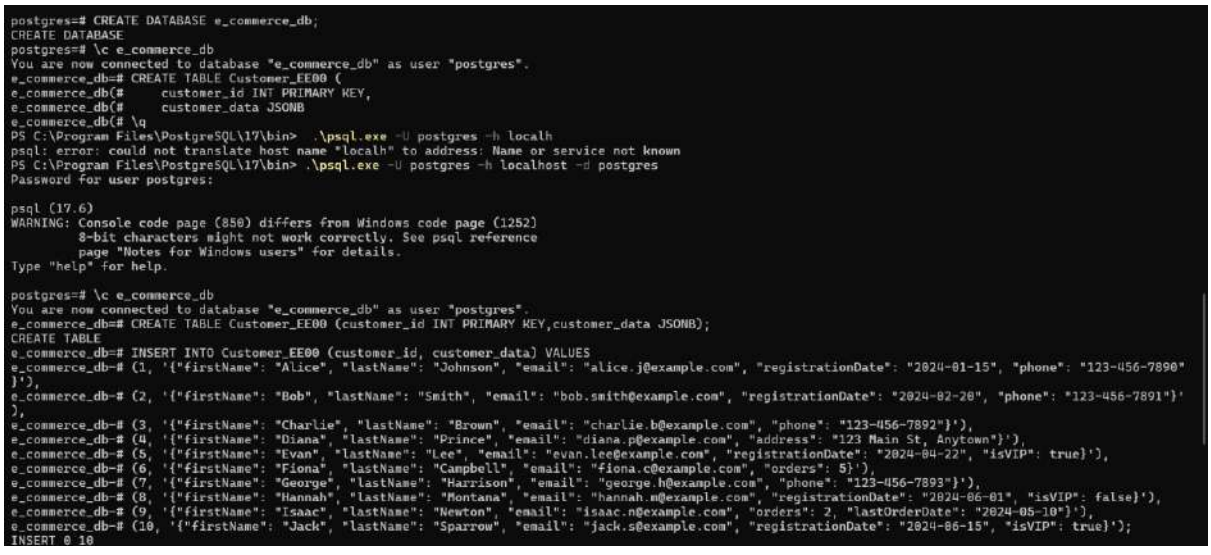

Figure 68 Creation and insertion into table

1.2 Query Operations on the Aggregate

```
e_commerce_db=# SELECT customer_id, customer_data FROM Customer_EE00
e_commerce_db-# WHERE customer_data ->> 'isVIP' = 'true';
 customer_id |                                                  customer_data
-------------+------------------------------------------------------------------------------------------------------------------
           5 | {"email": "evan.lee@example.com", "isVIP": true, "lastName": "Lee", "firstName": "Evan", "registrationDate": "2024-04-22"}
          10 | {"email": "jack.s@example.com", "isVIP": true, "lastName": "Sparrow", "firstName": "Jack", "registrationDate": "2024-06-15"}
(2 rows)
```

Figure 69 First Query

```
e_commerce_db=# SELECT customer_id, customer_data ->> 'firstName' AS first_name, customer_data ->> 'lastName' AS last_name
e_commerce_db-# FROM Customer_EE00
e_commerce_db-# WHERE (customer_data ->> 'orders')::int >= 5;
 customer_id | first_name | last_name
-------------+------------+-----------
           6 | Fiona      | Campbell
(1 row)


e_commerce_db=# SELECT * FROM Customer_EE00
e_commerce_db-#
e_commerce_db-#
e_commerce_db-# SELECT * FROM Customer_EE00;
ERROR:  syntax error at or near "SELECT"
LINE 2: SELECT * FROM Customer_EE00;
        ^
e_commerce_db=# SELECT * FROM Customer_EE00;
 customer_id |                                                  customer_data
-------------+-------------------------------------------------------------------------------------------------------------------------------
           1 | {"email": "alice.j@example.com", "phone": "123-456-7890", "lastName": "Johnson", "firstName": "Alice", "registrationDate": "2024-01-15"}
           2 | {"email": "bob.smith@example.com", "phone": "123-456-7891", "lastName": "Smith", "firstName": "Bob", "registrationDate": "2024-02-20"}
           3 | {"email": "charlie.b@example.com", "phone": "123-456-7892", "lastName": "Brown", "firstName": "Charlie"}
           4 | {"email": "diana.p@example.com", "address": "123 Main St, Anytown", "lastName": "Prince", "firstName": "Diana"}
           5 | {"email": "evan.lee@example.com", "isVIP": true, "lastName": "Lee", "firstName": "Evan", "registrationDate": "2024-04-22"}
           6 | {"email": "fiona.c@example.com", "orders": 5, "lastName": "Campbell", "firstName": "Fiona"}
           7 | {"email": "george.h@example.com", "phone": "123-456-7893", "lastName": "Harrison", "firstName": "George"}
           8 | {"email": "hannah.m@example.com", "isVIP": false, "lastName": "Montana", "firstName": "Hannah", "registrationDate": "2024-06-01"}
           9 | {"email": "isaac.n@example.com", "orders": 2, "lastName": "Newton", "firstName": "Isaac", "lastOrderDate": "2024-05-10"}
          10 | {"email": "jack.s@example.com", "isVIP": true, "lastName": "Sparrow", "firstName": "Jack", "registrationDate": "2024-06-15"}
(10 rows)
```

Figure 70 Second Query

Chapter 2: Graph data Model

2.1 Graph Database Creation

In this project, a graph database was created using ArangoDB to represent relationships between a group of students. The graph consists of 10 student nodes and 15 edges connecting them, representing friendships and interactions.

Each student node in the Person collection contains the following properties:

- _key: a unique identifier for the student
- name: the student's name

Each edge in the FriendssWith collection represents a friendship and contains the following properties:

- _from and _to: references to the connected students
- since: the date when the friendship started
- chat_count: the total number of chats exchanged between the two students
- common_interests: a list of interests shared by both students

This graph model allows us to analyze social connections, common interests, and communication patterns between students. Screenshots were captured during the implementation process and are included below in the order they were created.

1. Created Collections for nodes and edges inside database _system

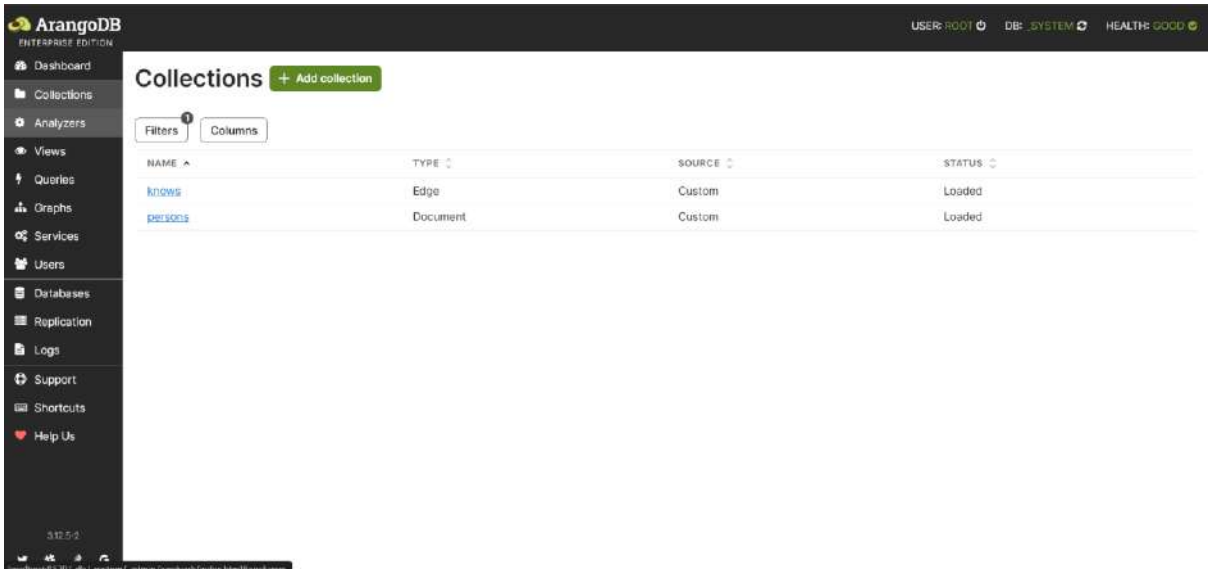

Figure 71 Created Database

2. Creating 10 Nodes

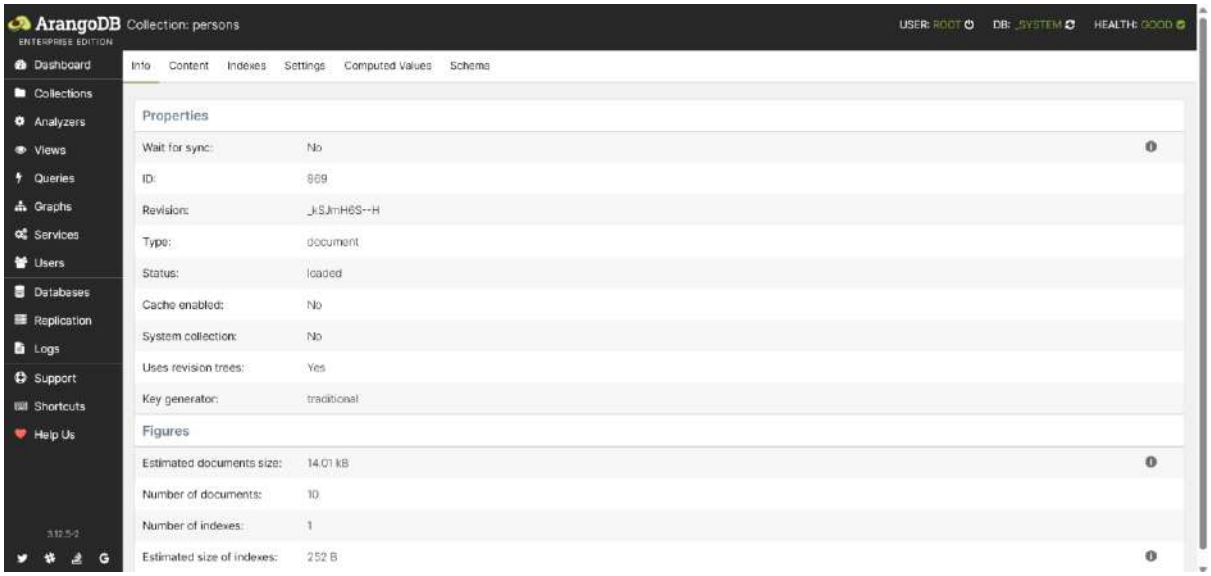

Figure 72 Creating Nodes

3. Created 10 Nodes

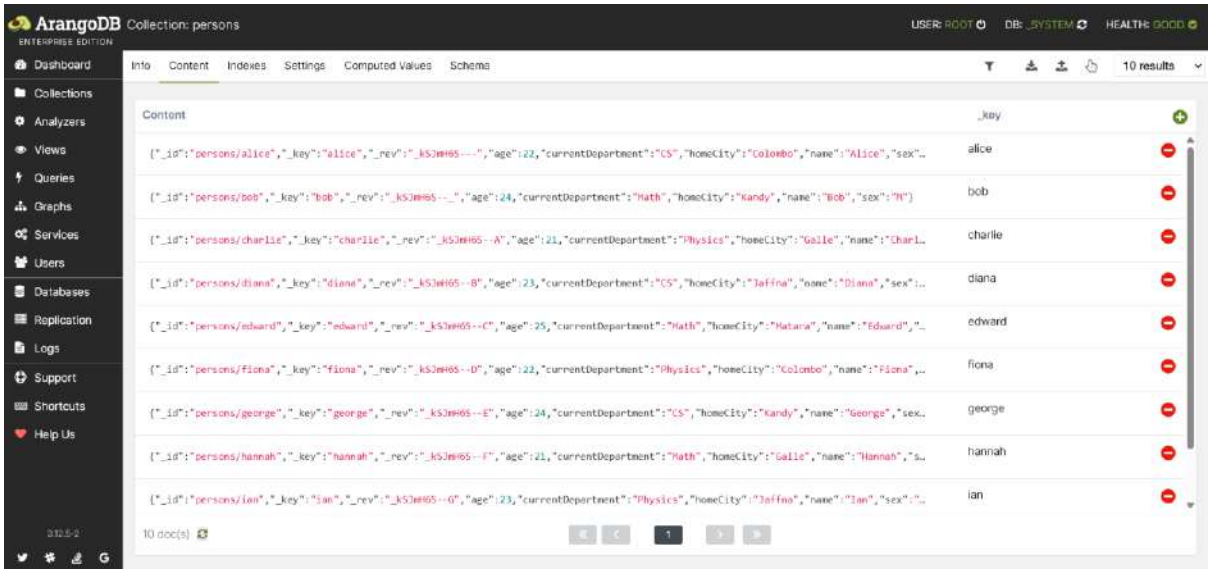

Figure 73 Created 10 nodes

4. Creating 15 edges or relationships

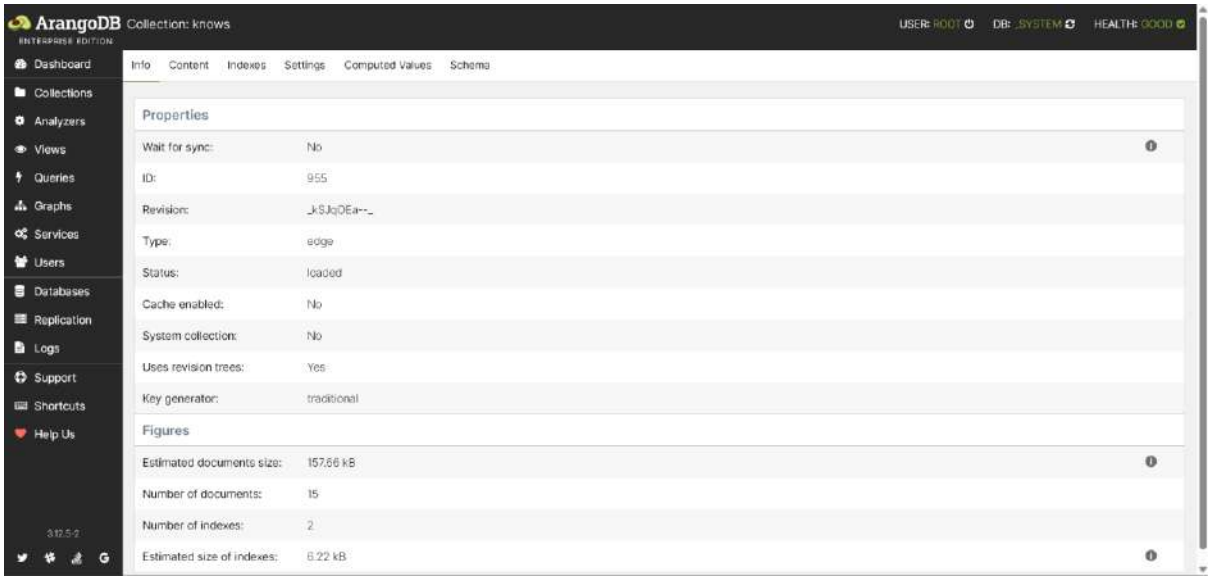

Figure 74 Creating 15 Edges (Relationships)

5. Created 15 edges or relationships

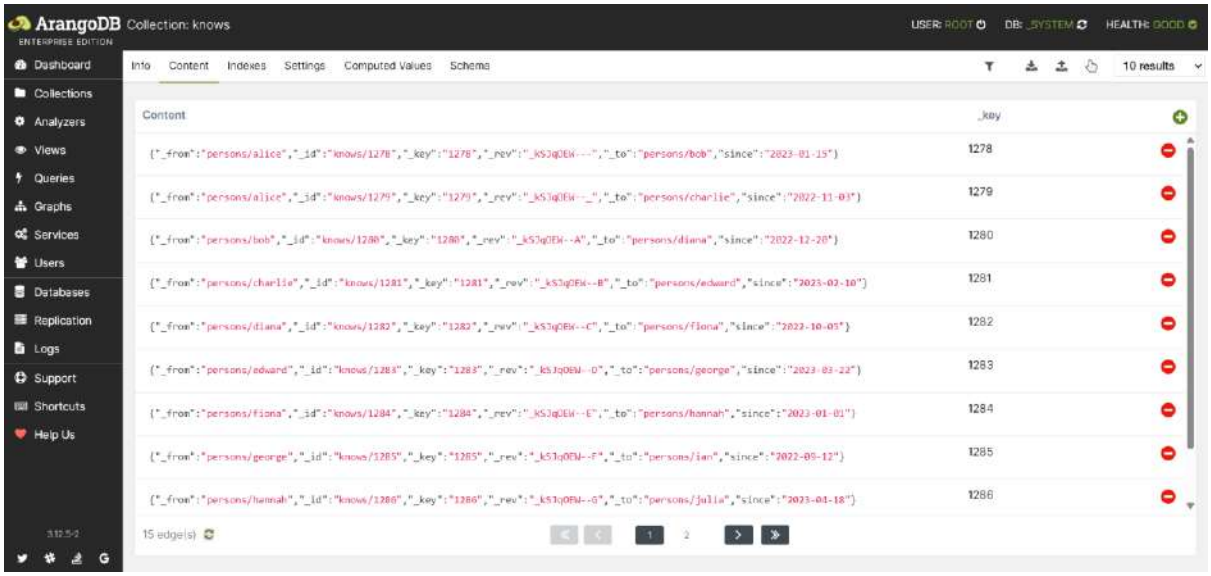

Figure 75 Created 15 Edges (Relationship)

6. Created and Implemented Graph model

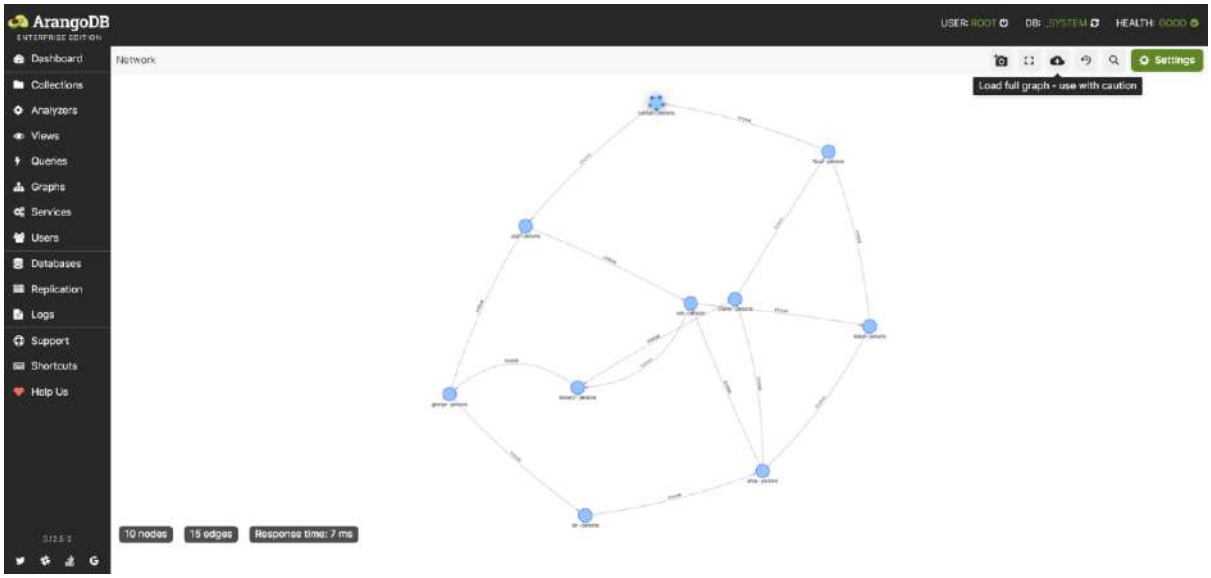

Figure 76 Graph

2.2 Querying the Social Network

7. Performed Query for data retrievals People who are Friends with Alice

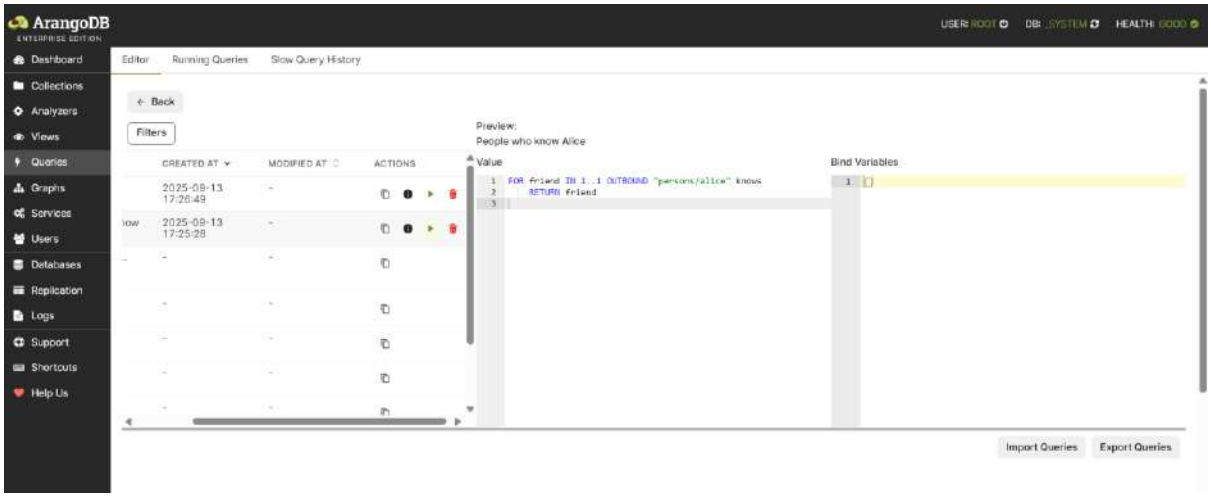

Figure 77 Query to retrieve people who are friends with Alice

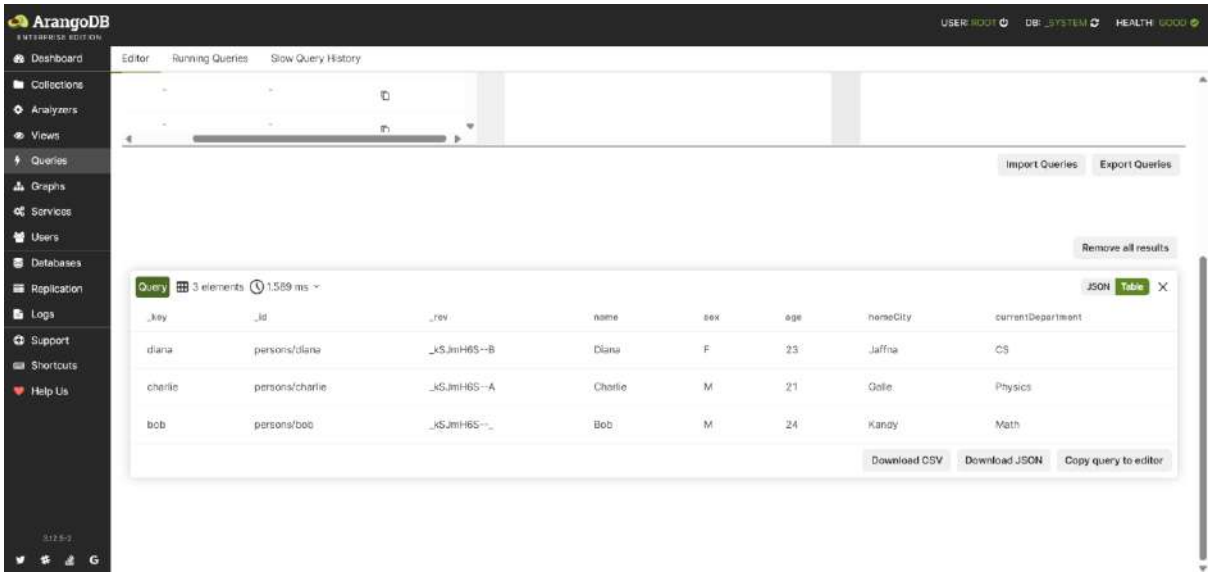

Figure 78 Output of 1st Query

8. Performed Query for retrieve path from Alice to Julia

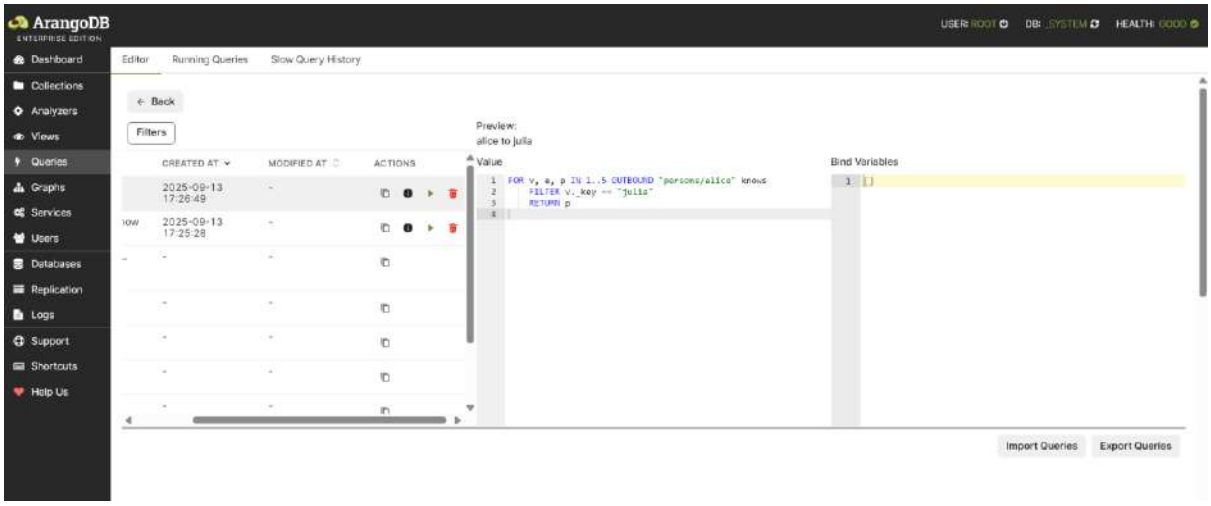

Figure 79 Query to retrieve path from Alice to Julia

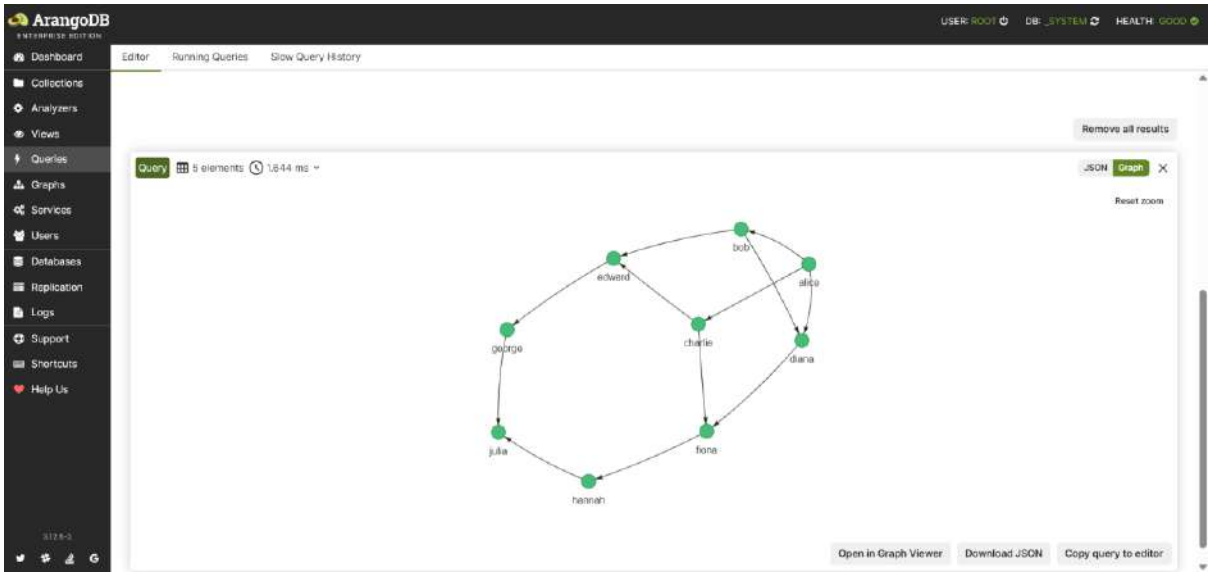

Figure 80 Output of second Query