Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Shop Management System

Course Title: Database System Lab
Course Code: CSE 210
Section: 222 D3

Students Details

Name	ID
Md Syful Islam	222002111

Submission Date: 15 December 2024
Course Teacher's Name: Ms. Farhana Akter Sunny

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The project is designed to create a comprehensive Database Management System (DBMS) for a retail shop. It manages and stores information about products, employees, sales, suppliers, and profits. The system is built to automate key operations and improve efficiency by using SQL queries to retrieve and manipulate data.

1.2 Motivation

Managing a shop's operations manually is time-consuming, error-prone, and inefficient. With a large number of products, employees, and transactions, it becomes difficult to track and update information. By developing a DBMS, this project aims to streamline these tasks, automate processes, and ensure accuracy and efficiency.

1.3 Problem Definition

1.3.1 Problem Statement

Manual shop operations are inefficient and time-consuming, leading to issues like inaccurate stock tracking, missed sales records, and scheduling conflicts. These challenges negatively impact business performance. This project addresses these problems by developing a database management system (DBMS) that automates operations, ensures data integrity, and facilitates efficient data retrieval and reporting.

1.3.2 Complex Engineering Problem

The following Table 1.1 below highlights the complex engineering attributes addressed in this project. These attributes are essential to identifying the challenges and designing effective solutions.

Table 1.1: Summary of the attributes touched in this projects

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	Requires understanding of database normalization, relationships, and SQL for efficient implementation.
P2: Range of conflicting requirements	Balances data redundancy minimization and query performance through normalization and indexing.
P3: Depth of analysis required	Involves analyzing shop operations to create an optimized database schema and relevant queries.
P4: Extent of applicable codes	SQL constraints, such as primary keys, foreign keys, and data integrity rules, ensure database reliability.
P5: Interdependence	Links tables such as inventory and sales using foreign keys to maintain data consistency and accuracy.

1.4 Design Goals/Objectives

1.4.1 Objectives and Design Principles

The primary goal of this project is to develop an efficient database management system to automate and streamline shop operations. Objectives include:

- Designing a schema to handle inventory, sales, employee schedules, and profit tracking.
- Ensuring data integrity and minimizing redundancy through normalization.
- Implementing SQL constraints and relationships for accuracy and consistency.
- Enabling easy data retrieval, reporting, and analysis using queries and aggregation.

1.4.2 Database Schema and Entity Design

The project implements a relational database with multiple table like Users, Products, Sales, Employees, and Profit Records. Each table includes unique attributes, and relationships between table are established using foreign keys to maintain consistency. The Schema diagram of these project are following :

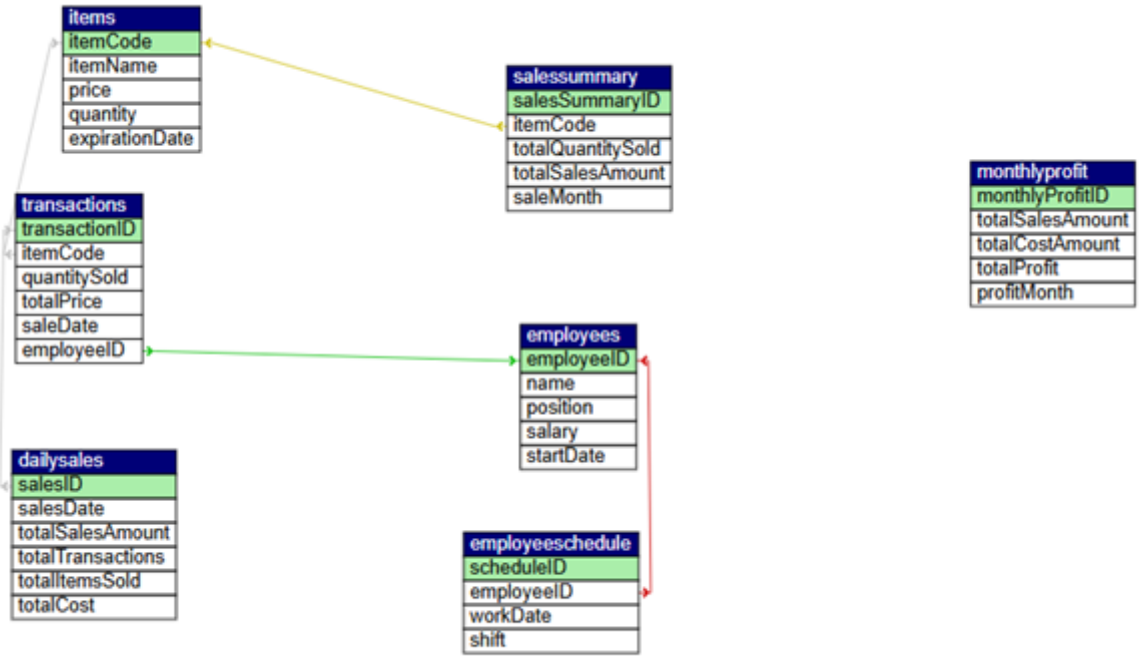

Figure 1.1: E-R Diagram of database

1.5 Application

This database management system can be applied in various domains, including retail stores, e-commerce platforms, and small to medium-sized businesses, to streamline operations such as inventory management, sales tracking, employee scheduling, and profit calculation. By automating these processes, the system ensures data accuracy, integrity, and efficiency, enabling businesses to make informed decisions and enhance overall productivity.

Chapter 2

Implementation and Performance Evaluation

2.1 Introduction

This chapter outlines is about the implementation of the shop management system database, with schema design, data population, and query execution. It demonstrates how tables are structured with relationships, constraints, and integrity rules to ensure accuracy and consistency. Screenshots of queries and outputs validate the system's functionality, validates its ability to handle operations efficiently. The chapter also evaluates system performance, highlighting the reliability of the database in meeting operational requirements.

2.2 System Implementation

2.2.1 Development Environment Setup

Tools and Technologies Used For developing the shop management database system, the following tools and technologies were utilized:

- Database Management System: MySQL was selected as the DBMS for its robust features for constraints, relationships, and aggregate functions.
- Development Interface: MySQL Workbench and phpMyAdmin were used for database schema design, query execution, and data visualization.
- Local Server Environment: XAMPP was used to host the MySQL server and provide a localhost environment for running the database.

These tools collectively created an efficient environment for designing and managing the database, enabling smooth development and testing.

2.2.2 Database Server Configuration

- **XAMPP Setup:** The XAMPP control panel was configured to start the MySQL server. This tool was selected for its simplicity in managing server operations.
- **MySQL Server Running:** MySQL Workbench was connected to the running MySQL server. The database schema was created, and all tables, relationships, and constraints were defined within this environment.

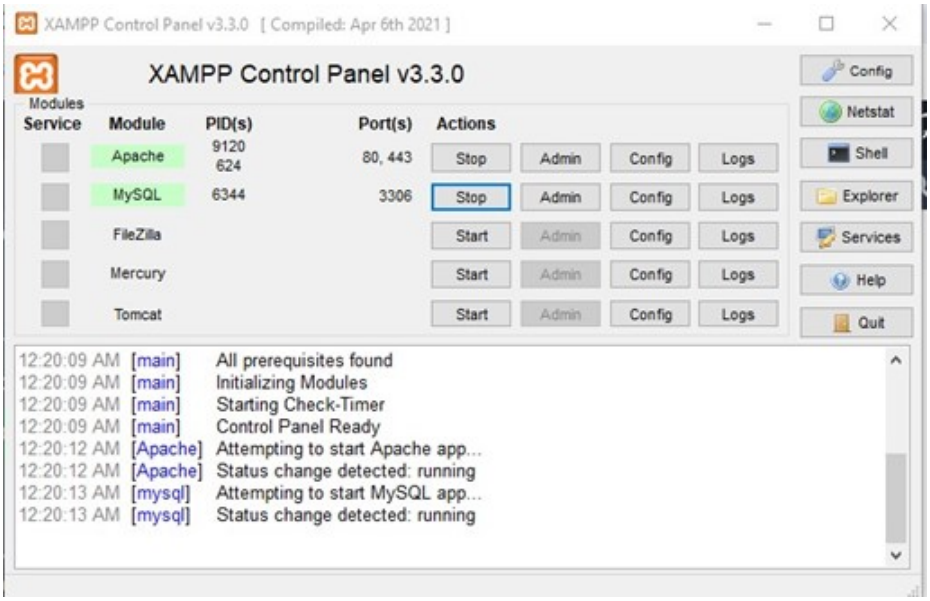

Figure 2.1: XAMPP control panel with MySQL server running

2.3 Implementation Details

Key implementation features include in this project :

1. **Inventory Management:** SQL triggers and constraints ensure stock levels are automatically updated after every transaction.
2. **Sales Tracking:** Daily sales are recorded, with aggregate functions generating weekly, monthly, and yearly summaries.
3. **Profit Calculation:** Arithmetic operations calculate profits by subtracting total expenses from revenue for specific periods.

2.4 Query Implementation, System Testing and Validation

2.4.1 Table Creation and Data Insertion

1.Query for create ‘items’ table

```
CREATE TABLE Items (
itemCode INT PRIMARY KEY,
itemName VARCHAR(50) NOT NULL,
price FLOAT NOT NULL CHECK (price > 0),
quantity INT NOT NULL CHECK (quantity >= 0),
expirationDate DATE NOT NULL );
```

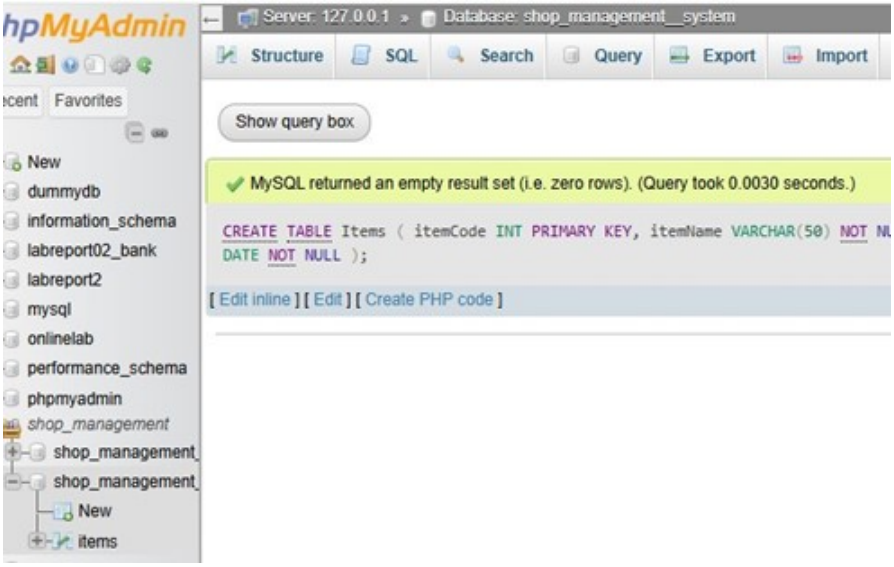

Figure 2.2: Created Items table

2.Insert value in ‘items’ table

```
INSERT INTO Items (itemCode, itemName, price, quantity, expirationDate)
VALUES (101, ’Milk’, 25, 50, ’2024-12-31’),
(102, ’Bread’, 50, 100, ’2024-11-30’),
(103, ’Eggs’, 30, 200, ’2022-12-15’),
(104, ’Butter’, 20, 75, ’2021-12-25’),
(105, ’Cheese’, 40, 120, ’2024-11-28’),
(106, ’Soap’, 65, 70, ’2026-12-31’),
(107, ’Juice’, 35, 80, ’2028-11-30’),
(108, ’Chips’, 10, 200, ’2023-12-15’),
(109, ’Lacchi’, 25, 48, ’2024-12-25’),
(110, ’Ice-Cream’, 100, 45, ’2027-11-28’);
```

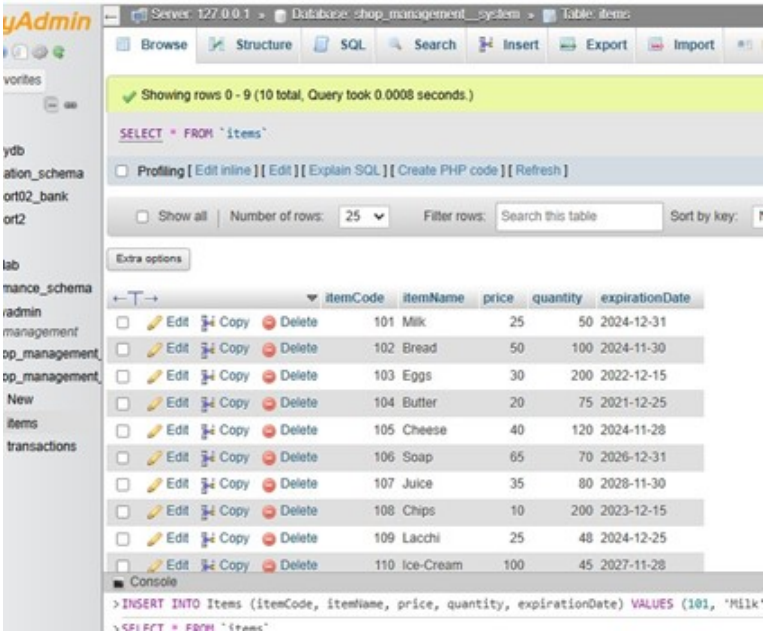

Figure 2.3: Inserted values into items table.

3.Create 'transactions' table

CREATE TABLE Transactions (
transactionID INT AUTO$_INCREMENT PRIMARY KEY$
$itemCode INT NOT NULL, quantitySold INT NOT NULL CHECK(quantitySold >= 0),$
$totalPrice FLOAT NOT NULL CHECK(totalPrice >= 0), saleDate DATE NOT NULL,$
$FOREIGN KEY(itemCode) REFERENCES Items(itemCode)$
$ON DELETE CASCADE$
$ON UPDATE CASCADE$
);

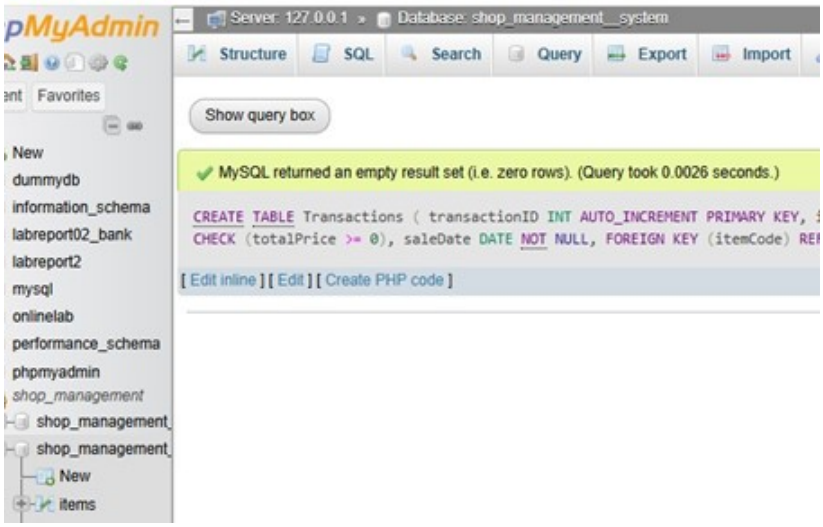

Figure 2.4: Created transactions table

4.Create ‘employees’ table CREATE TABLE Employees (
employeeID INT AUTO$_{I}NCREMENTPRIMARYKEY$,
$nameVARCHAR(50)NOTNULL$,
$positionVARCHAR(50)NOTNULL$,
$salaryFLOATNOTNULLCHECK(salary >= 0)$,
$startDateDATENOTNULL$
);

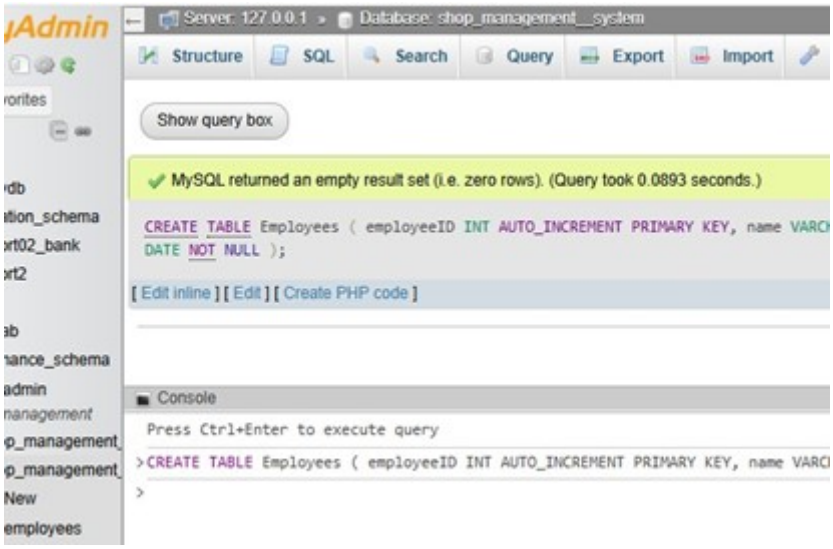

Figure 2.5: Created employees table

5.Insert value into ‘employees’ table
INSERT INTO Employees (name, position, salary, startDate)
VALUES (’Bissosto Kew ekjon’, ’Cashier’, 25000, ’2022-01-01’),
(’Syful’, ’Manager’, 45000, ’2021-05-15’),
(’Bomb smith’, ’Stock Keeper’, 20000, ’2023-02-10’),
(’Kono ekta Nam’, ’Sales Assistant’, 22000, ’2023-07-20’),
(’Adam Smith’, ’Accountant’, 30000, ’2020-11-01’);

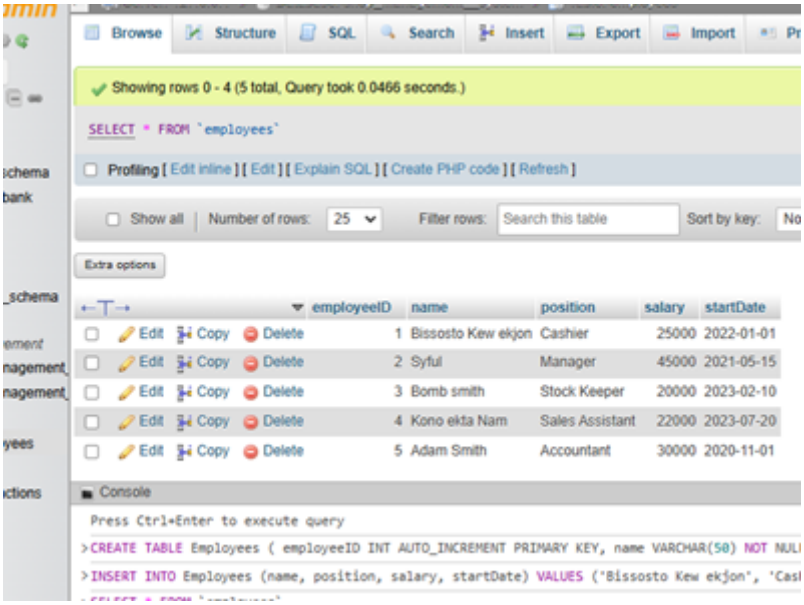

Figure 2.6: Inserted value in employees table

6.Create 'EmployeeSchedule' table

CREATE TABLE EmployeeSchedule (
scheduleID INT AUTO$_I NCREMENT PRIMARY KEY$,
$employeeID INT NOT NULL$,
$workDate DATE NOT NULL$,
$shift ENUM('Morning','Afternoon','Night') NOT NULL$,
$FOREIGN KEY (employeeID) REFERENCES Employees(employeeID)$
$ON DELETE CASCADE$
$ON UPDATE CASCADE$
);

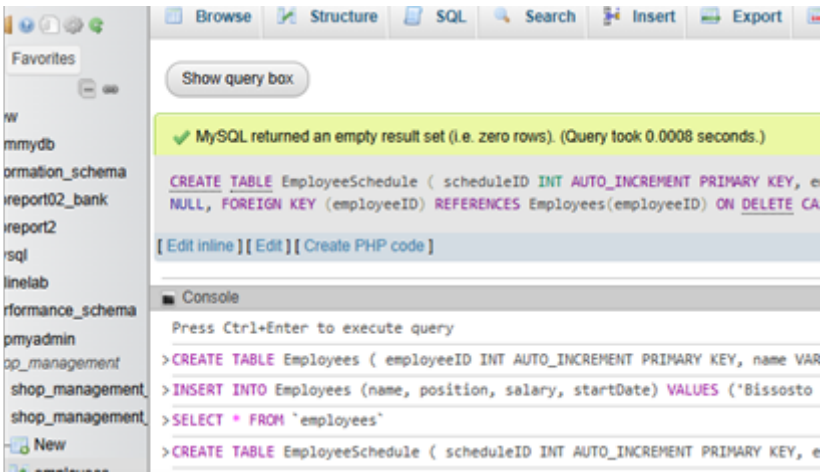

Figure 2.7: Created employeeSchedule table

7.Insert value into ‘employeeSchedule’ table

INSERT INTO EmployeeSchedule (employeeID, workDate, shift)
VALUES (1, ’2024-11-29’, ’Morning’),
(1, ’2024-11-30’, ’Afternoon’),
(2, ’2024-11-29’, ’Morning’),
(3, ’2024-11-29’, ’Night’),
(4, ’2024-11-30’, ’Morning’),
(5, ’2024-11-30’, ’Afternoon’);

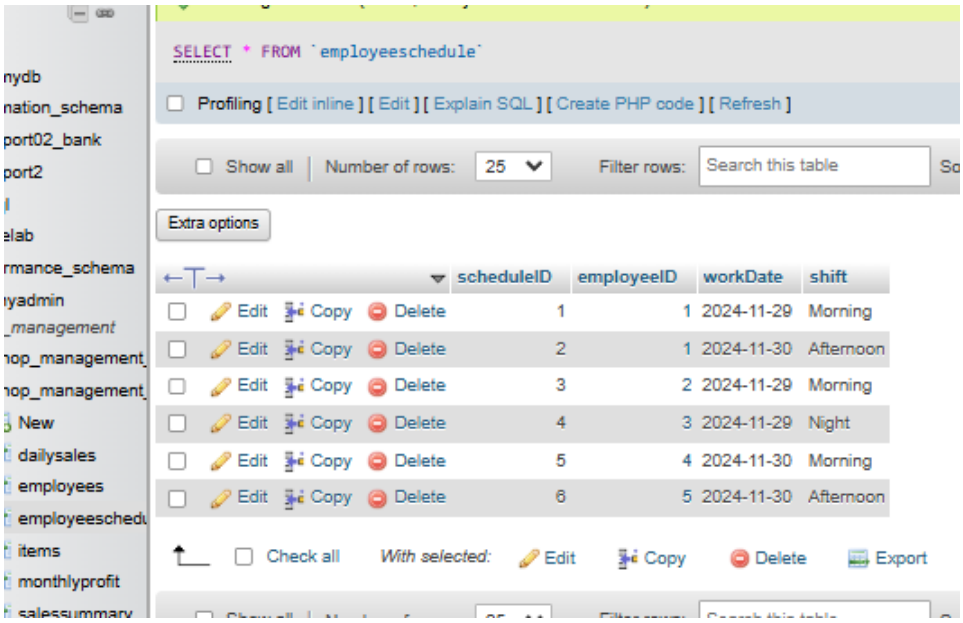

Figure 2.8: Inserted Value in employeeSchedule table

8.Create dailySales table

CREATE TABLE DailySales (
salesID INT AUTO$_I NCREMENT PRIMARY KEY$,
$salesDateDATENOTNULLUNIQUE$,
$totalSalesAmountFLOATNOTNULLCHECK(totalSalesAmount >= 0)$,
$totalTransactionsINTNOTNULLCHECK(totalTransactions >= 0)$,
$totalItemsSoldINTNOTNULLCHECK(totalItemsSold >= 0)$
);

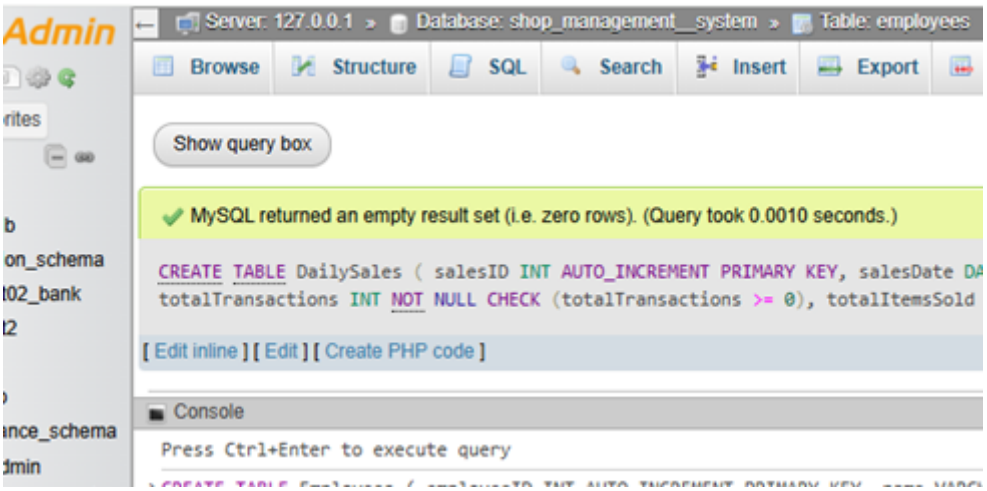

Figure 2.9: Created DailySalse Table

9.Create 'SalesSummary' table

CREATE TABLE SalesSummary (
salesSummaryID INT AUTO$_I NCREMENT PRIMARY KEY$,
$itemCode INT NOT NULL, employeeID INT NOT NULL,$
$totalQuantitySold INT NOT NULL CHECK(totalQuantitySold >= 0),$
$totalSalesAmount FLOAT NOT NULL CHECK(totalSalesAmount >= 0),$
$saleMonth DATE NOT NULL,$
$FOREIGN KEY(itemCode) REFERENCES Items(itemCode)$
$ON DELETE CASCADE ON UPDATE CASCADE,$
$FOREIGN KEY(employeeID) REFERENCES Employees(employeeID)$
$ON DELETE CASCADE ON UPDATE CASCADE$
);

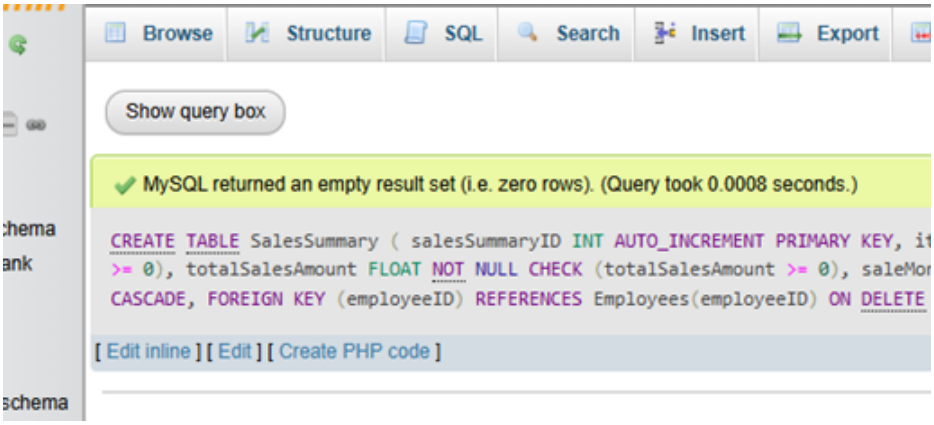

Figure 2.10: Created SalseSummary Table

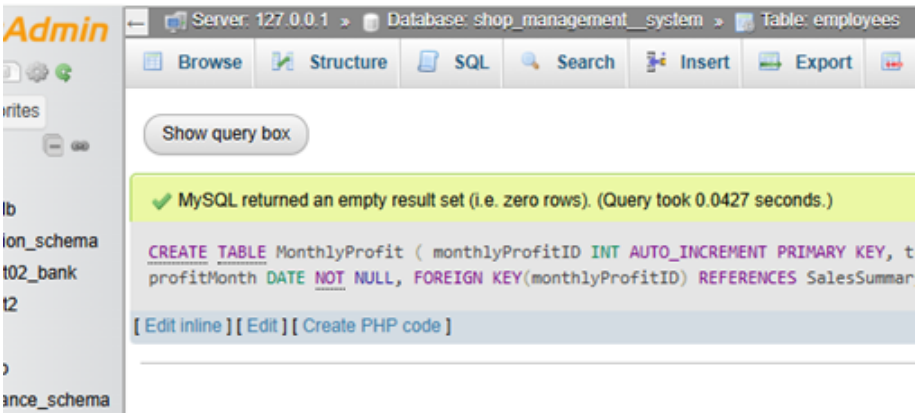

Figure 2.11: Created MonthlyProfit Table

10.Create 'MonthlyProfit' table

CREATE TABLE MonthlyProfit (
monthlyProfitID INT AUTO$_I NCREMENT PRIMARY KEY$,
$totalSalesAmount FLOAT NOT NULL$,
$totalCostAmount FLOAT NOT NULL$,
$totalProfit FLOAT NOT NULL$,
$profitMonth DATE NOT NULL$,
$FOREIGN KEY(monthlyProfitID) REFERENCES SalesSummary(salesSummaryID)$
$ON DELETE CASCADE ON UPDATE CASCADE$
);

2.4.2 Query for data Manipulation

1.Query for Add salseDate in Daily Salse Table

ALTER TABLE DailySales ADD UNIQUE (salesDate);

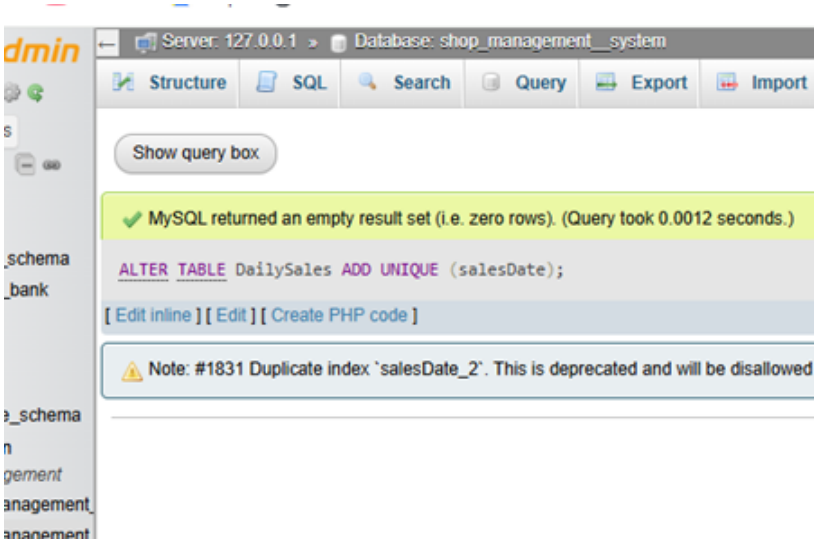

Figure 2.12: Added salseDate in Daily Salse Table

2.Query for drop Foreign key from SalesSummary Table
ALTER TABLE SalesSummary
DROP FOREIGN KEY salessummary$_ibfk_2$;

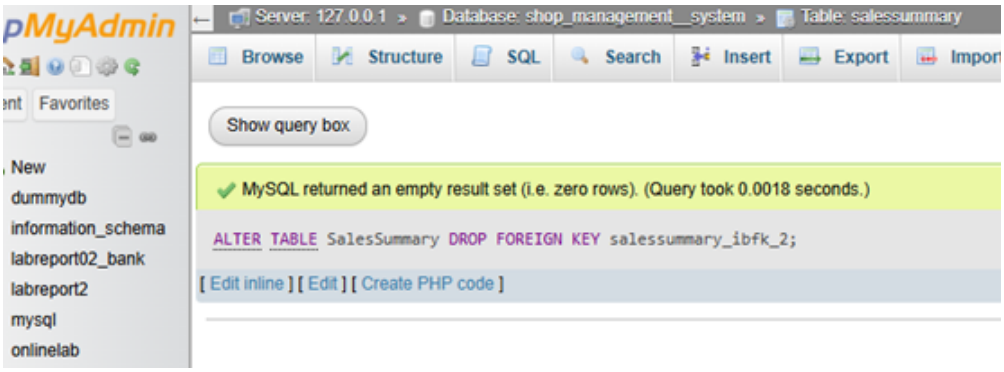

Figure 2.13: Droped Foreign key from SalesSummary Table

3.Query for drop employeeID colum from SalesSummary table

ALTER TABLE SalesSummary
DROP COLUMN employeeID;

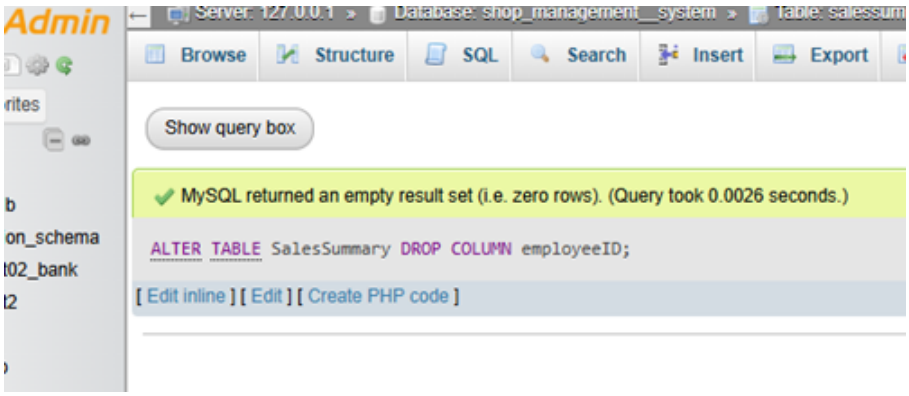

Figure 2.14: Droped employeeID colum from SalesSummary table.

4.Query for Drop foreign key from Monthlyprofit Table

ALTER TABLE MonthlyProfit DROP FOREIGN KEY monthlyprofit$_ibfk_1$;

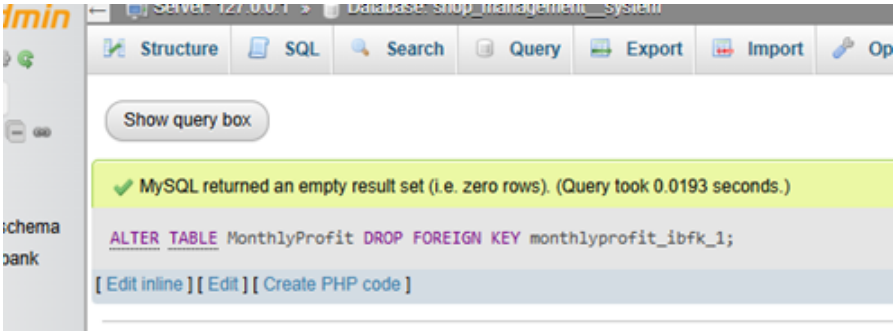

Figure 2.15: Dropped Foreign Key from Monthlyprofit Table

5.Query for change data type of salary in employees
ALTER TABLE Employees
MODIFY salary DECIMAL(10, 2);

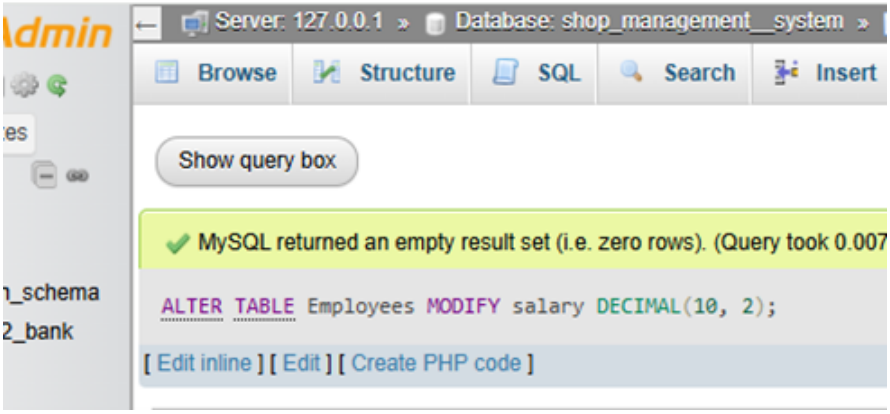

Figure 2.16: Updated employee salary to decimal.

6.Query for drop unique index
ALTER TABLE DailySales DROP INDEX salesDate;

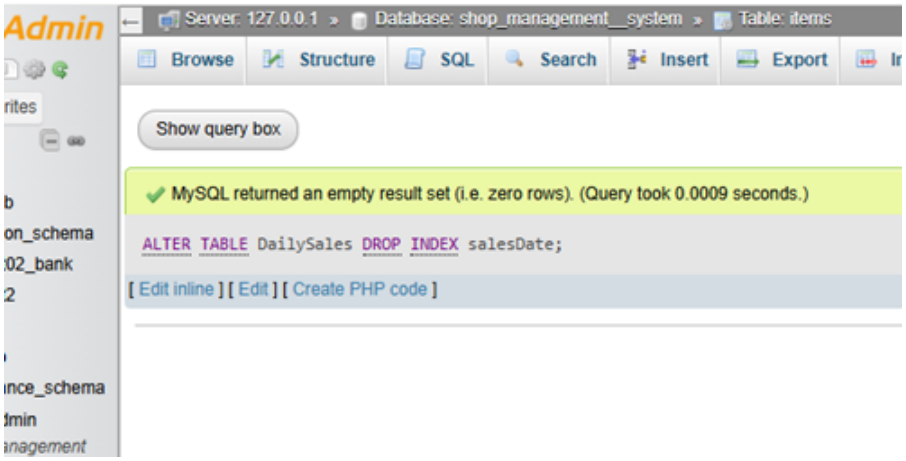

Figure 2.17: Dropped unique index salesDate from Daily salse table.

7.Query for update colum in dailysalse table

UPDATE DailySales
SET salesDate = CURRENT$_{D}ATE$();

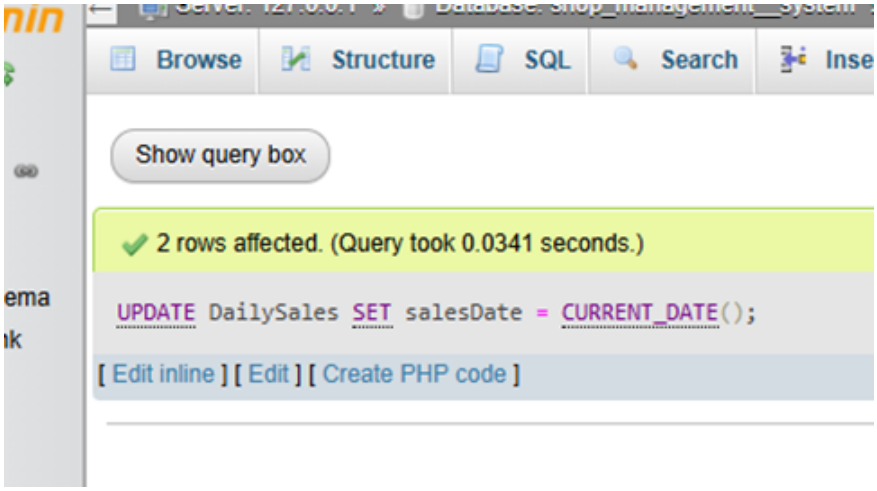

Figure 2.18: Updated colum salseDate to CurrentDate.

8.Query for add a foreign key to link transactions with employees

ALTER TABLE Transactions
ADD COLUMN employeeID INT,
ADD FOREIGN KEY (employeeID) REFERENCES Employees(employeeID)
ON DELETE CASCADE;

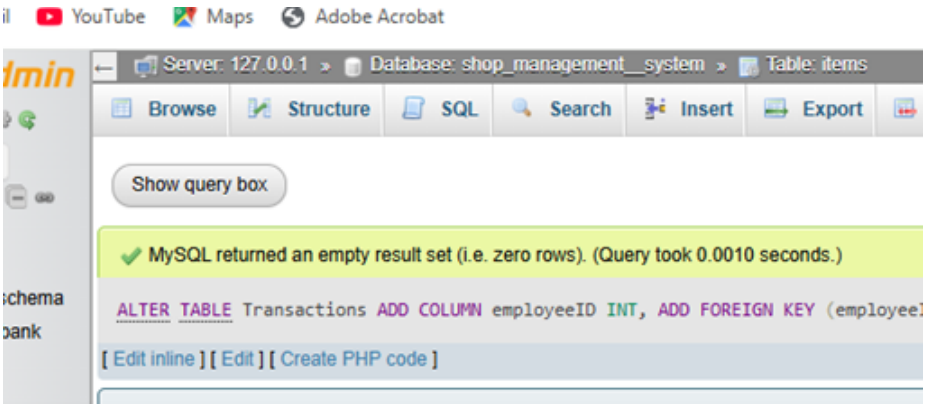

Figure 2.19: Aadded foreign key to link transactions with employees.

9.Query to add a birthDate column to the Employees table

```
ALTER TABLE Employees
ADD COLUMN birthDate DATE AFTER name;
```

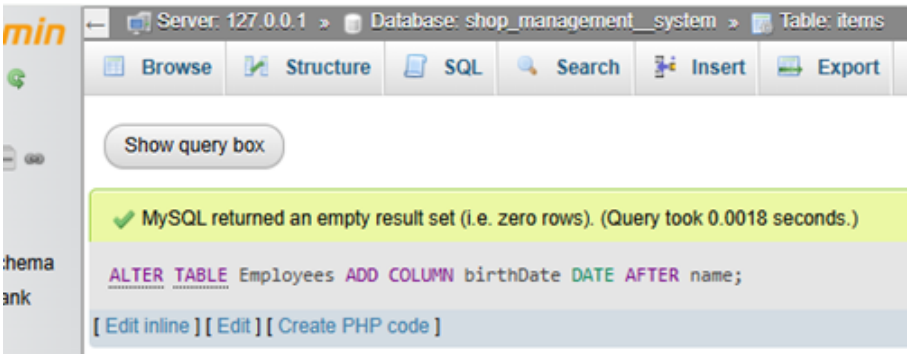

Figure 2.20: Added an birthdate column in employee table.

10.Query for Drop column birthdate column from employees table

```
ALTER TABLE Employees
DROP COLUMN birthDate;
```

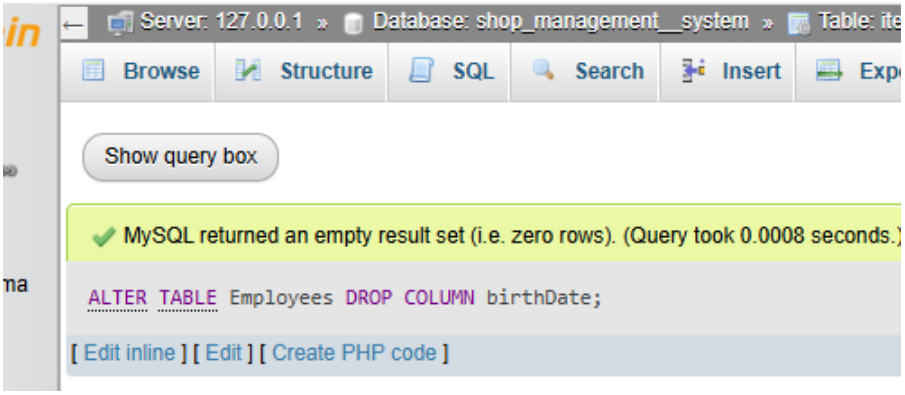

Figure 2.21: Dropped birthdate column.

2.4.3 Relation (Joins, Subqueries, and Aggregate Functions)

1.Query for Find Total Sales Amount for Each Item

```
SELECT i.itemName, SUM(t.totalPrice) AS totalSales FROM Items i
JOIN Transactions t ON i.itemCode = t.itemCode
GROUP BY i.itemCode;
```

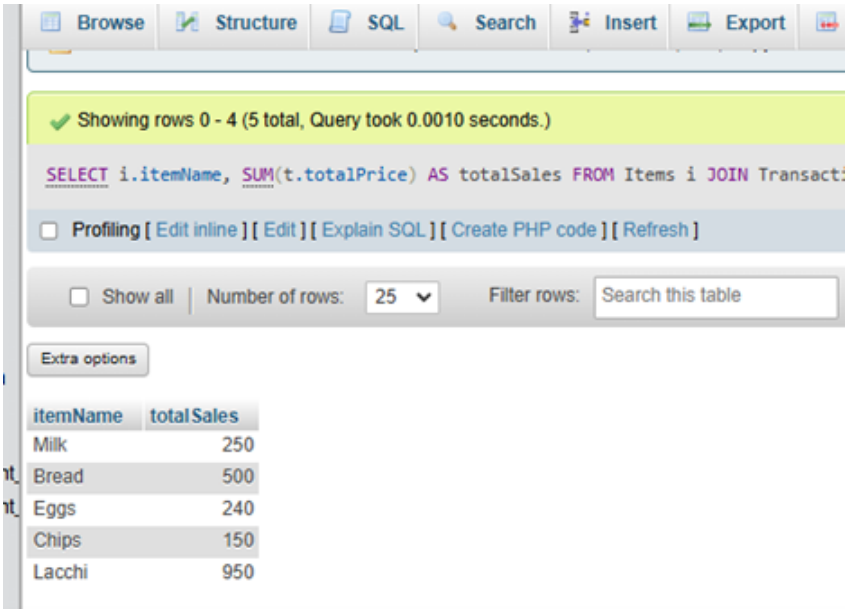

Figure 2.22: Output of query 1.

2.Query for find items that have stock below 50

```
SELECT itemName, quantity
FROM Items
WHERE quantity < 50;
```

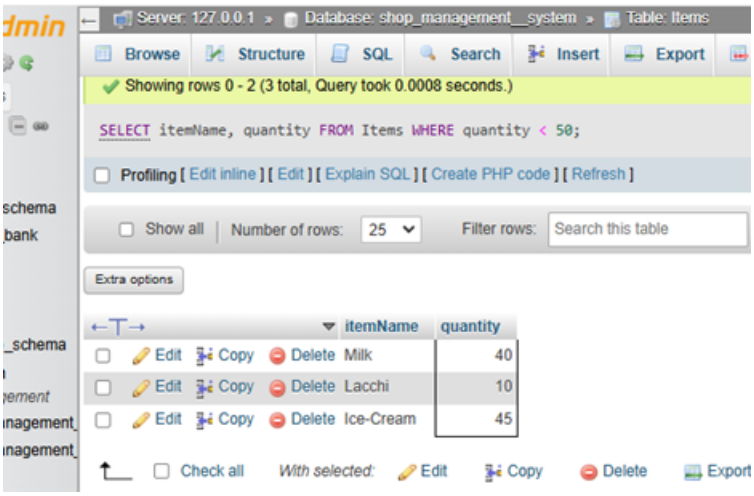

Figure 2.23: Output of query 2.

3.Query for find the highest-selling item based on quantity

```
SELECT i.itemName, SUM(t.quantitySold) AS totalQuantity FROM Items i
JOIN Transactions t ON i.itemCode = t.itemCode
GROUP BY i.itemCode, i.itemName
ORDER BY totalQuantity DESC
LIMIT 1;
```

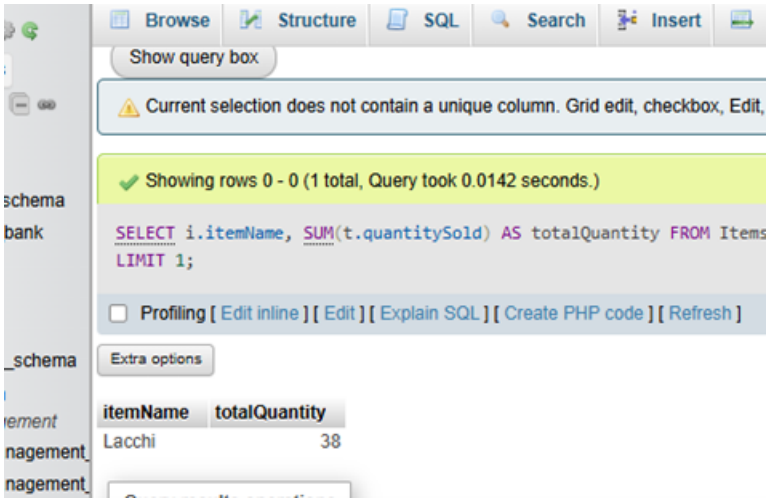

Figure 2.24: Output of query 3.

4.Query for Calculate Average Salary by Employee Position

```
SELECT position, AVG(salary) AS averageSalary FROM Employees
GROUP BY position;
```

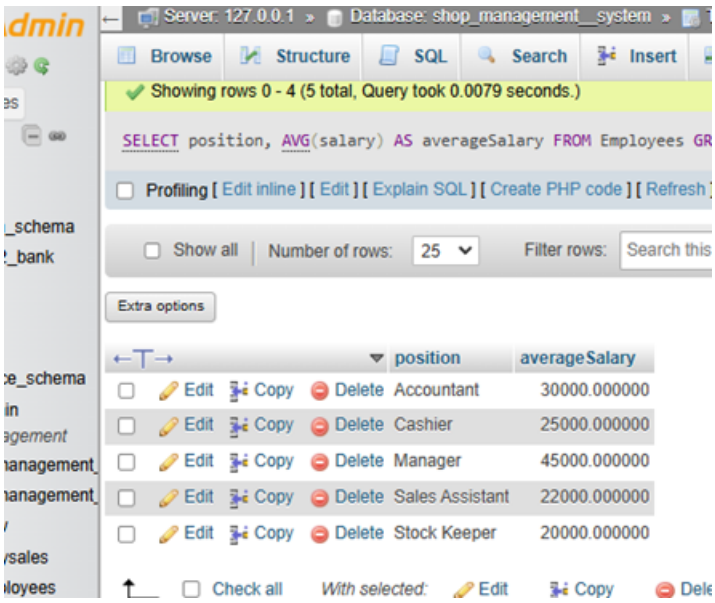

Figure 2.25: Output of query 4.

5.Query for find total profit for the current month

```
SELECT SUM(totalProfit) AS totalMonthlyProfit FROM MonthlyProfit
WHERE MONTH(profitMonth) = MONTH(CURDATE()) AND
YEAR(profitMonth) = YEAR(CURDATE());
```

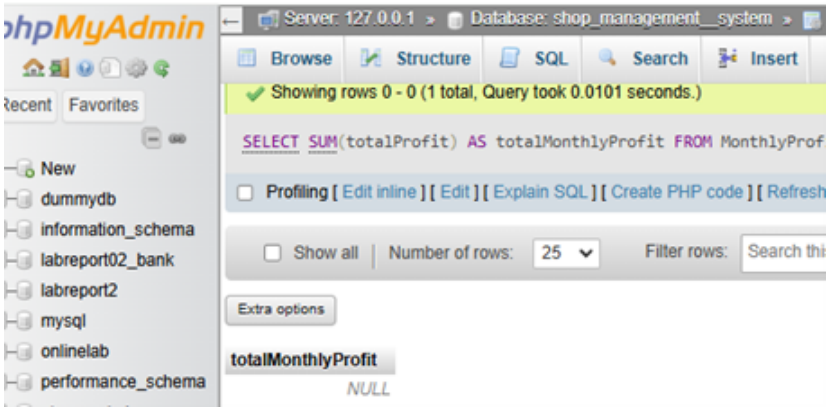

Figure 2.26: Output of query 5.

6.Query for list employees working in both morning and night shifts

```
SELECT DISTINCT e.name
FROM Employees e
JOIN EmployeeSchedule es ON e.employeeID = es.employeeID
WHERE es.shift = 'Morning' UNION SELECT DISTINCT e.name
FROM Employees e
JOIN EmployeeSchedule es ON e.employeeID = es.employeeID
WHERE es.shift = 'Night';
```

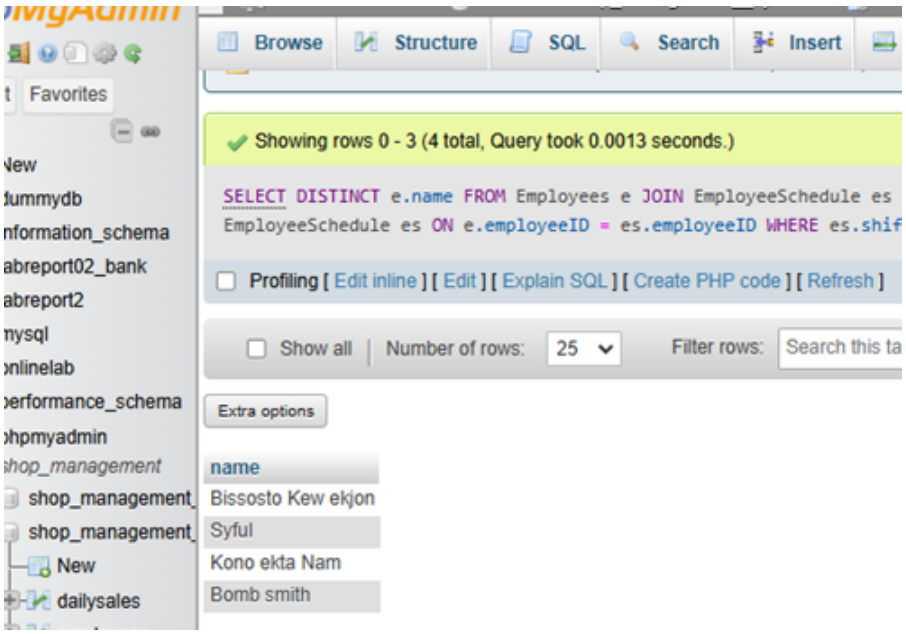

Figure 2.27: Output of query 6.

7.Query for find total items sold per day.

```
SELECT d.salesDate, SUM(d.totalItemsSold) AS totalItemsSold
FROM DailySales d
GROUP BY d.salesDate;
```

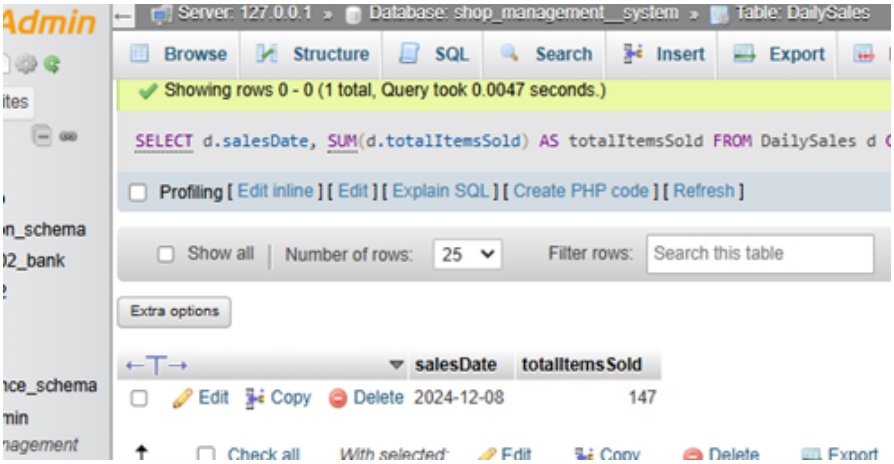

Figure 2.28: Output of query 7.

8.Query for calculate the percentage contribution of each item to total sales.

```
SELECT i.itemName,
(SUM(t.totalPrice) / (SELECT SUM(totalPrice) FROM Transactions) * 100) AS
salesPercentage
FROM Items i
JOIN Transactions t ON i.itemCode = t.itemCode
GROUP BY i.itemCode;
```

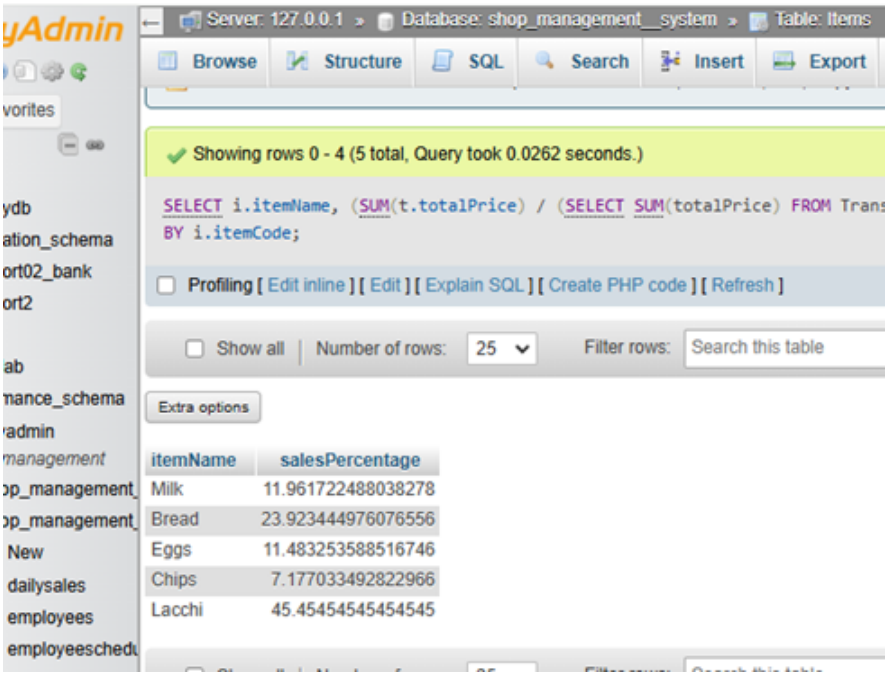

Figure 2.29: Output of query 8.

9. Query to find the list of all items ordered by current quantity in ascending order,

```
SELECT itemName, quantity
FROM Items
ORDER BY quantity ASC;
```

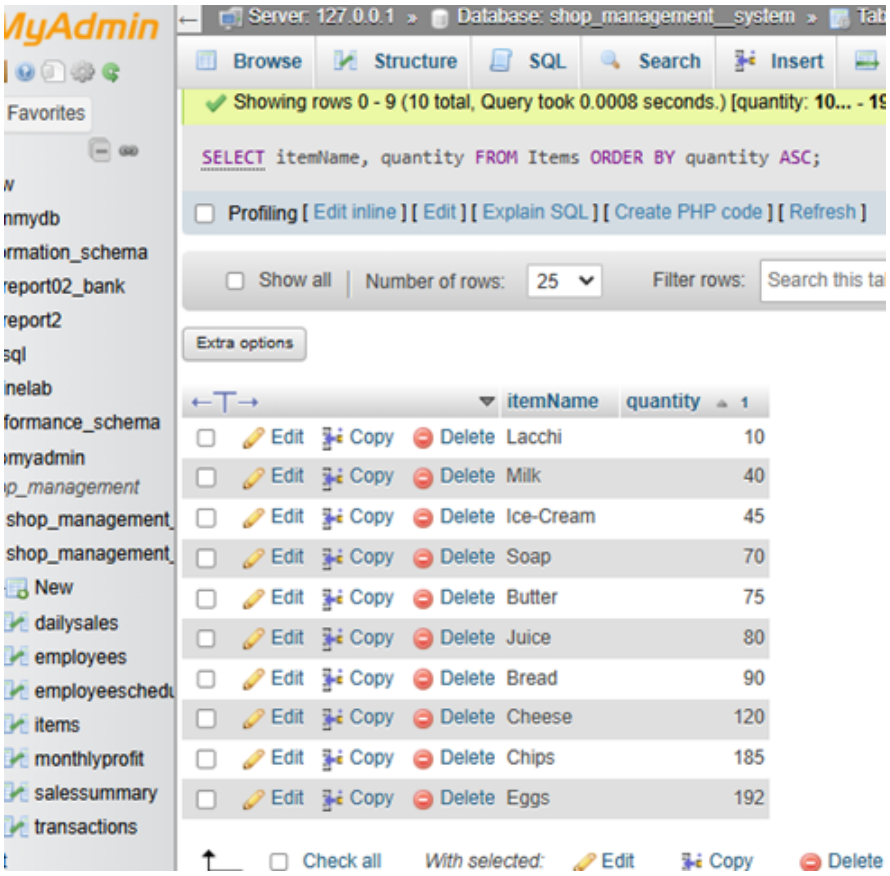

Figure 2.30: Output of query 9.

10. Query for find the expiration date of all products ordered by the nearest expiration date to the current date,

```
SELECT itemName, expirationDate
FROM Items
WHERE expirationDate >= CURDATE()
ORDER BY expirationDate ASC;
```

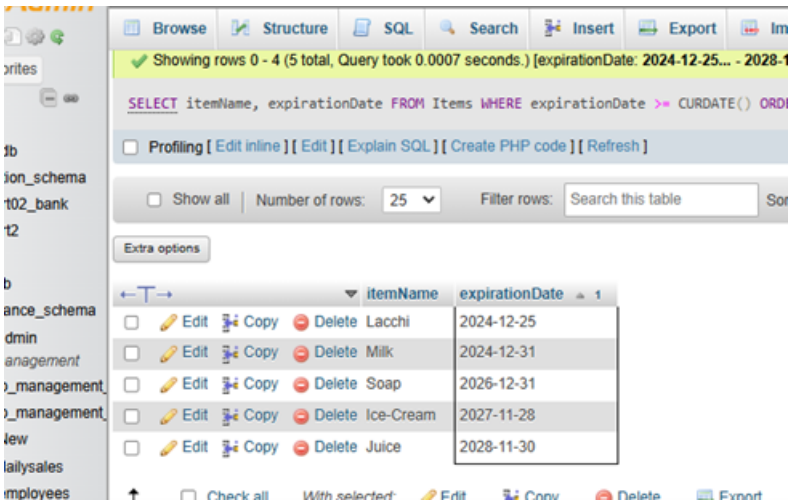

Figure 2.31: Output of query 10.

2.4.4 Triggers

1. Trigger for Stock Update and Daily Sales Record:

DELIMITER

CREATE TRIGGER after$_{t}ransaction_{i}nsert$

$AFTERINSERTONTransactions$

$FOREACHROW$

$BEGIN$

$UPDATEItems$

$SETquantity = quantity - NEW.quantitySoldWHEREitemCode = NEW.itemCode;$

$INSERTINTODailySales(salesDate, totalSalesAmount, totalTransactions, totalItemsSold)$

$VALUES(NEW.saleDate, NEW.totalPrice, 1, NEW.quantitySold)$

$ONDUPLICATEKEYUPDATE$

$totalSalesAmount = totalSalesAmount + NEW.totalPrice,$

$totalTransactions = totalTransactions + 1,$

$totalItemsSold = totalItemsSold + NEW.quantitySold;$

END

$DELIMITER;$

$[DollarSignMissedInThisQueryDueToyntexIssue]$

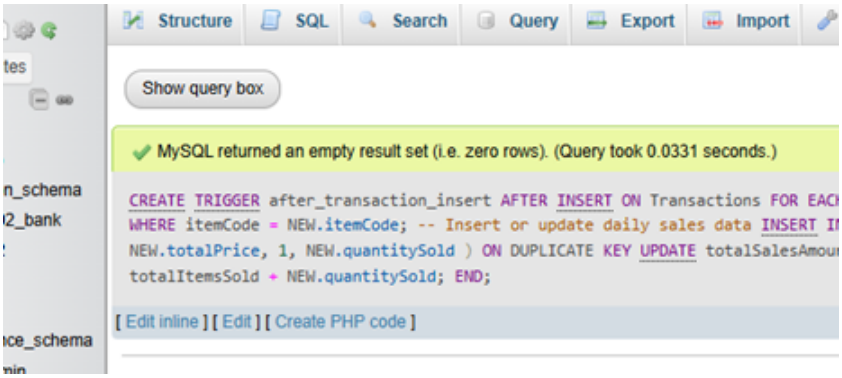

Figure 2.32: Triggered for Stock Update and Daily Sales Record.

2. Query for Sale 5 Milk From Item Table

INSERT INTO Transactions (itemCode, quantitySold, totalPrice, saleDate)
VALUES (101, 5, 125, '2024-12-07');

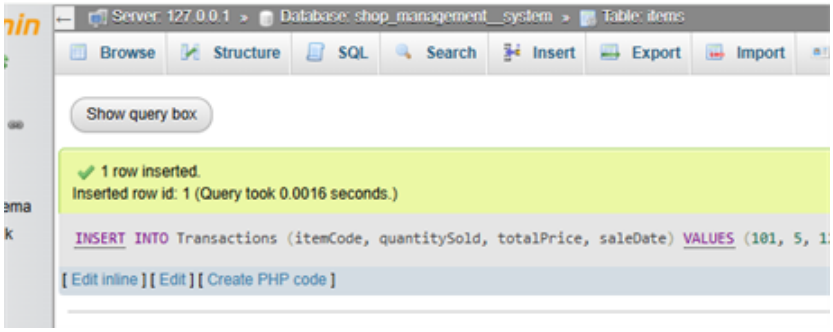

Figure 2.33: Sold 5 Milk From Item Table.

After complete the transaction, the state of Item and Daily Salse Table are following,

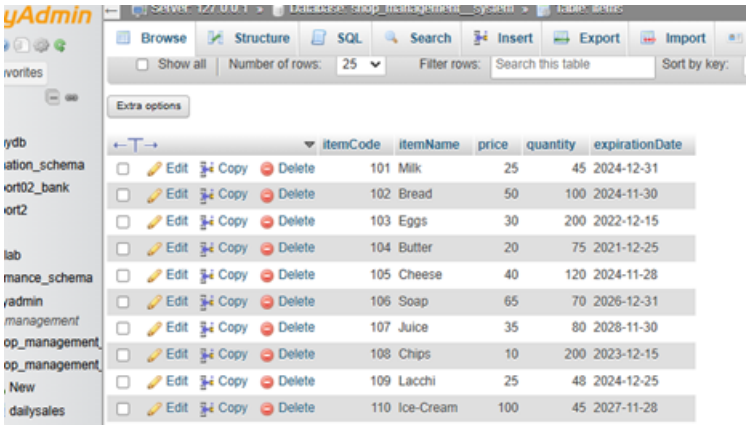

Figure 2.34: State of Items Table.

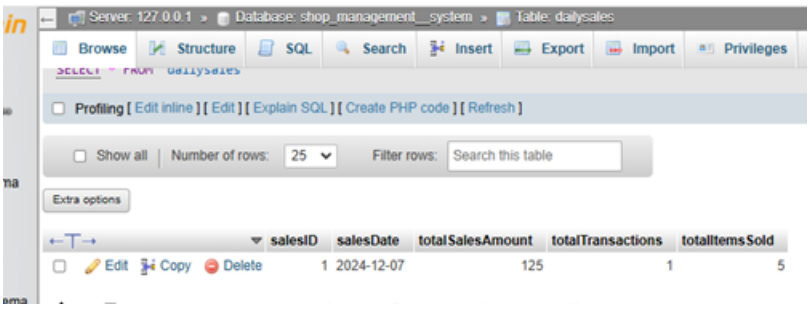

Figure 2.35: State of DailySalse Table.

3.Trigger for Updates daily sales information when a transaction occurs.

CREATE TRIGGER update$_{d}aily_{s}ales$
$AFTERINSERTONTransactions$
$FOREACHROW$
$BEGININSERTINTODailySales(salesDate,$
$totalSalesAmount,totalTransactions,totalItemsSold)$
$VALUES(NEW.saleDate,NEW.totalPrice,1,NEW.quantitySold)$
$ONDUPLICATEKEYUPDATE$
$totalSalesAmount = totalSalesAmount + NEW.totalPrice,$
$totalTransactions = totalTransactions + 1,$
$totalItemsSold = totalItemsSold + NEW.quantitySold;$
END
$DELIMITER;$

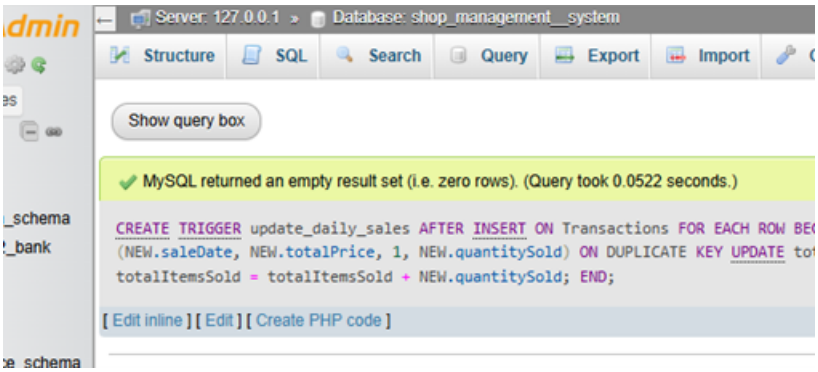

Figure 2.36: Applied Trigger for dailySalse record

4.Trigger for Moves expired items to an archive table.

DELIMITER
CREATE TRIGGER handle$_{e}xpired_{i}tems$
$BEFOREDELETEONItems$
$FOREACHROW$
$BEGIN$
$IFOLD.expirationDate < CURDATE()THEN$
$INSERTINTOExpiredItems(itemCode,itemName,expirationDate)$
$VALUES(OLD.itemCode,OLD.itemName,OLD.expirationDate);$
$ENDIF;$
END
$DELIMITER;$

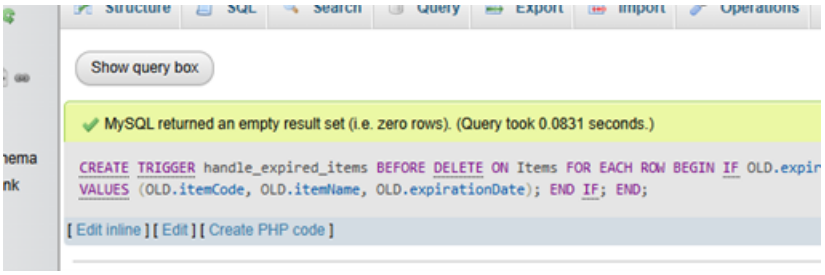

Figure 2.37: Applied Trigger for Moves expired items to an archive table.

5.Trigger for updates monthly profit when daily sales are updated.

DELIMITER

CREATE TRIGGER update$_{m}onthly_{p}rofit$

$AFTERINSERTONDailySales$

$FOREACHROW$

$BEGININSERTINTOMonthlyProfit(profitMonth,totalSalesAmount,totalProfit)$

$VALUES(DATE_FORMAT(NEW.salesDate,'Y-m-01'),$

$NEW.totalSalesAmount,NEW.totalSalesAmount-NEW.totalCost)$

$ONDUPLICATEKEYUPDATE$

$totalSalesAmount=totalSalesAmount+NEW.totalSalesAmount,$

$totalProfit=totalProfit+(NEW.totalSalesAmount-NEW.totalCost);$

END

$DELIMITER;$

$[DollarAndPercentSignAreMissedInThisQueryDueToSyntaxIssue]$

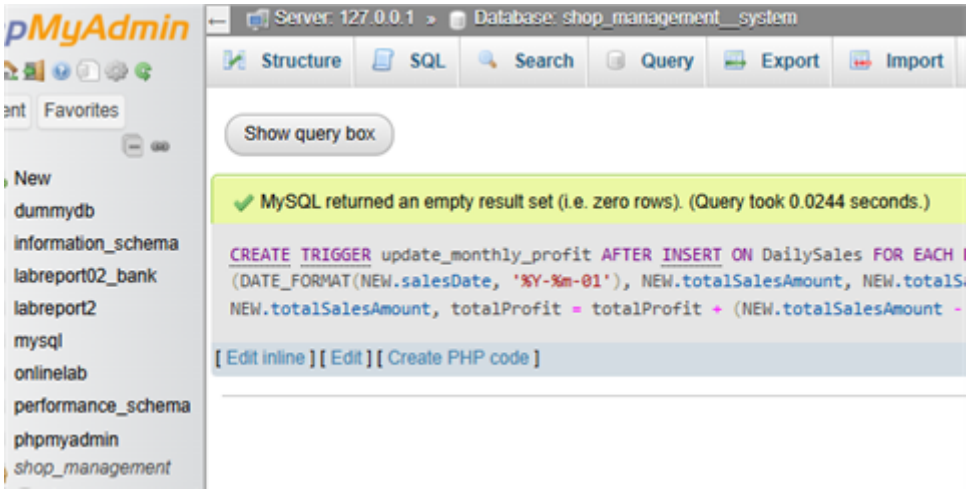

Figure 2.38: Updates monthly profit when daily sales are updated.

6.Updated Trigger for to employee id from Previous trigger

DELIMITER

CREATE TRIGGER track$_{s}ales$

$AFTERINSERTONTransactions$

$FOREACHROW$

$BEGIN$

$INSERTINTOSalesSummary(itemCode,totalQuantitySold,totalSalesAmount,saleMonth)$

$VALUES(NEW.itemCode,NEW.quantitySold,NEW.totalPrice,$

$DATE_FORMAT(NEW.saleDate,'Y-m-01'))$

$ONDUPLICATEKEYUPDATE$

$totalQuantitySold=totalQuantitySold+NEW.quantitySold,$

$totalSalesAmount=totalSalesAmount+NEW.totalPrice;$

END

$DELIMITER;$

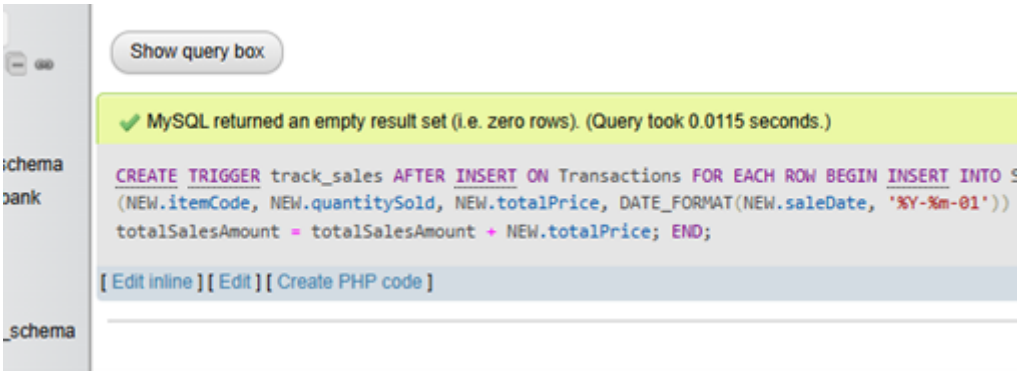

Figure 2.39: Deleted employeeID.

7.Trigger for Logs a warning if stock levels fall below ,
DELIMITER
CREATE TRIGGER monitor$_{i}nventory_{l}evels$
$AFTERUPDATEONItems$
$FOREACHROW$
$BEGIN$
$IFNEW.quantity < 10THEN$
$INSERTINTOLowStockAlerts(itemCode, itemName, quantity)$
$VALUES(NEW.itemCode, NEW.itemName, NEW.quantity)$;
$ENDIF$;
END
$DELIMITER$;

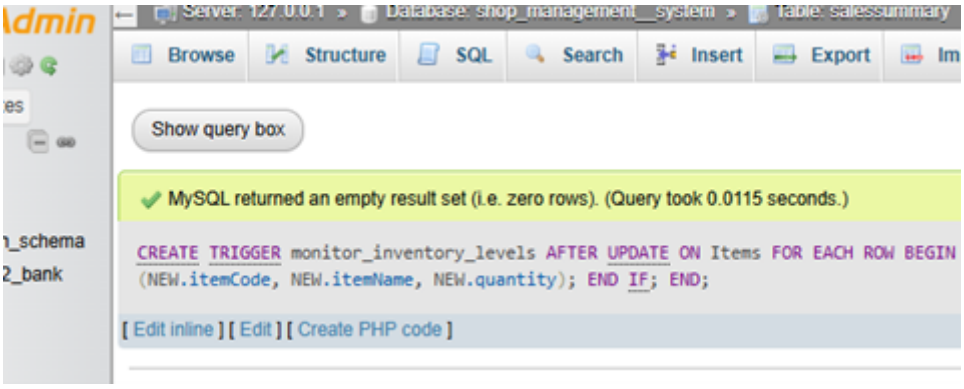

Figure 2.40: Triggered to Logs a warning if stock levels fall below .

8.Trigger for Process automate Sale
DELIMITER
CREATE TRIGGER automated$_{s}ell$
$BEFOREINSERTONTransactions$
$FOREACHROW$
$BEGIN$
$DECLAREitemPriceFLOAT$;

$SELECT\ priceINTOitemPriceFROMItems$
$WHEREitemCode = NEW.itemCode;$
$SETNEW.totalPrice = NEW.quantitySold * itemPrice;$
$IFNEW.saleDateISNULLTHEN$
$SETNEW.saleDate = CURDATE();$
$ENDIF;$
END
$DELIMITER;$

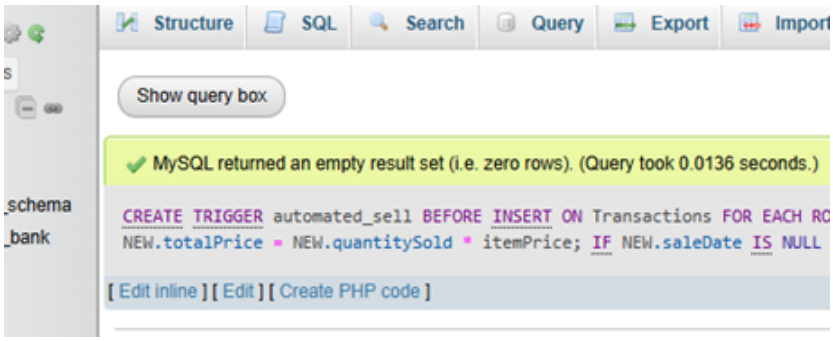

Figure 2.41: Created trigger for automate Sale .

9.Query For sells some item

```
INSERT INTO Transactions (itemCode, quantitySold) VALUES (101, 5);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (102, 5);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (102, 5);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (101, 5);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (103, 8);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (109, 38);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (108, 10);
INSERT INTO Transactions (itemCode, quantitySold) VALUES (108, 5);
```

After complete the transaction the state of transaction are following ,

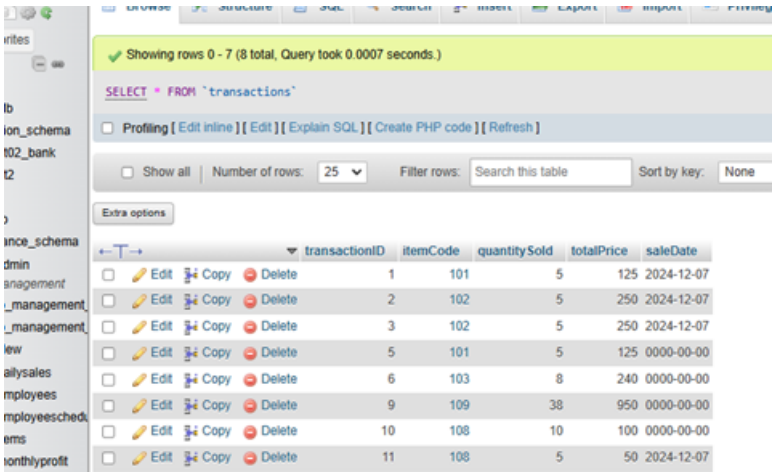

Figure 2.42: State of transaction Table

And after sold the state of items table are ,

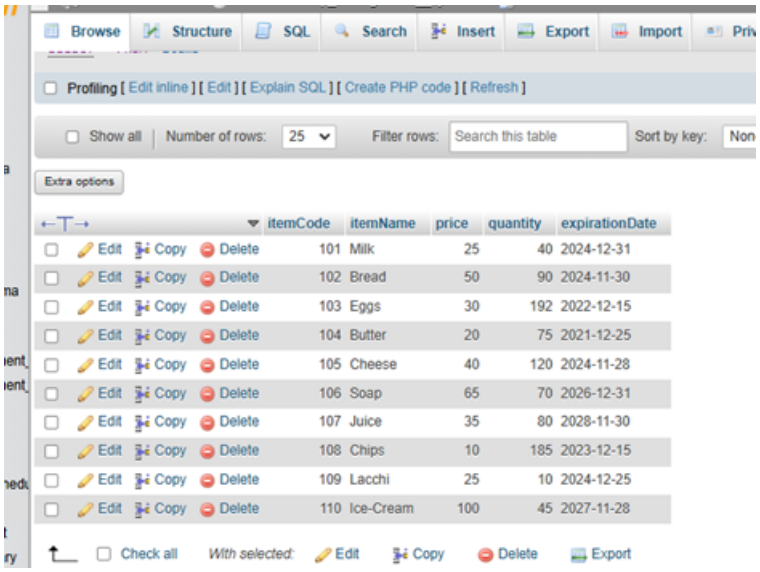

Figure 2.43: The state of items table after complete mentioned transaction

2.4.5 Results Overall Discussion

The results were achieved by designing and implementing different types of sql triggers, queries, and database operations to simulate real-world scenarios in managing sales, inventory, and employee records. The triggers automated critical tasks such as updating stock, tracking profits, and managing employee deletions, while queries provided proper analysis of sales, profits, and inventory status. However, challenges fetches, such as handling foreign key constraints and ensuring data consistency across multiple tables. These issues required careful adjustments to the database structure and logic to achieve accurate and reliable results. Overall, the implementation demonstrates robust database management practices while highlighting areas for optimization.

Chapter 3

Conclusion

3.1 Discussion

This project successfully developed a complete database system for manage a shop operations, including inventory, employee scheduling, sales tracking, and monthly profit analysis. The database focused on maintaining data accuracy, reducing duplication, and making queries easy to execute. By using relationships, constraints, and aggregate functions, the system provided reliable and smooth performance. Observations from the implementation show that the system is effective in improving operations and reducing manual work, making it a valuable system for shop management.

3.2 Limitations

Although the system works well, but it has some limitations. It mainly focuses on database management and does not include any user interface, which might make it hard for non technical users. The system, while functional, has certain limitations thats are :

- It lacks an user-friendly interface,that making it difficult for non-technical users to interact with the system.
- Database performance could decline as the data volume increases, especially in high-activity environments such as large shops.
- There are no integrated backup mechanisms, posing a risk of data loss in the event of server failures.
- The system does not currently support advanced analytics or functionalities for better inventory or sales management.

3.3 Scope of Future Work

Future improvements will aim to solve the current problems and add to more features. Plans include creating an simple user interface for better interaction and adding realtime updates and notifications.To overcome these limitations and enhance the system's utility, following improvements are planned:

- Develop a simple, intuitive user interface to improve accessibility for non-technical users.
- Introduce real-time updates and notifications for immediate transaction and inventory tracking.
- Implement predictive analytics for forecasting future sales and inventory requirements.
- Optimize the database for handling large datasets efficiently to ensure smooth performance in high-demand scenarios.
- Integrate automated backup systems to minimize the risk of data loss during server failures.

These improvements will transform the system into a comprehensive, modern solution tailored to meet the needs of advance technology.

References

1. https://www.upgrad.com/blog/dbms-project-ideas-for-beginners/
2. http://localhost/phpmyadmin/
3. https://www.javatpoint.com/mysql-trigger
4. https://dev.mysql.com/doc/refman/8.4/en/aggregate-functions.html
5. https://www.youtube.com/watch?v=rIi1dvPdTHEpp=ygUII215c3Fsc3A3D
6. https://www.w3schools.com/MySQL/mysql$_{a}utoincrement.asp$