

**UNIVERITY OF RWANDA**

**HUYE CAMPUS**

**DEPARTMENT OF BIT**

**YEAR 2 (GROUP A)**

**NAMES: TWAHIRWA Egide**

**REG NUMBER: 222007743**

**SECTION I**

**1. Describe all the entities and their corresponding attributes that are in your database.**

**1.Fruit**

Attributes:

Fruit ID (Primary Key)

Name

Description

Category (e.g., citrus, tropical, berries)

Price per Unit

Unit of Measurement (e.g., kilogram, pound, piece)

Stock Quantity

Supplier ID (Foreign Key)

**2.Supplier**

Attributes:

Supplier ID (Primary Key)

Name

Contact Person

Contact Email

Contact Phone

Address

**3.Customer**

Attributes:

Customer ID (Primary Key)

First Name

Last Name

Email

Phone Number

Address

**4.Order**

Attributes:

Order ID (Primary Key)

Customer ID (Foreign Key)

Order Date

Total Price

Status (e.g., Pending, Shipped, Delivered)

**5.OrderItem**

Attributes:

Order Item ID (Primary Key)

Order ID (Foreign Key)

Fruit ID (Foreign Key)

Quantity

Subtotal Price

**6.Payment**

Attributes:

Payment ID (Primary Key)

Order ID (Foreign Key)

Payment Date

Payment Amount

Payment Method (e.g., Credit Card, PayPal)

**2.Create an LDM of your entities**

Fruit (Fruit ID, Name, Description, Category, Price per Unit, Unit of Measurement, Stock Quantity, Supplier ID)

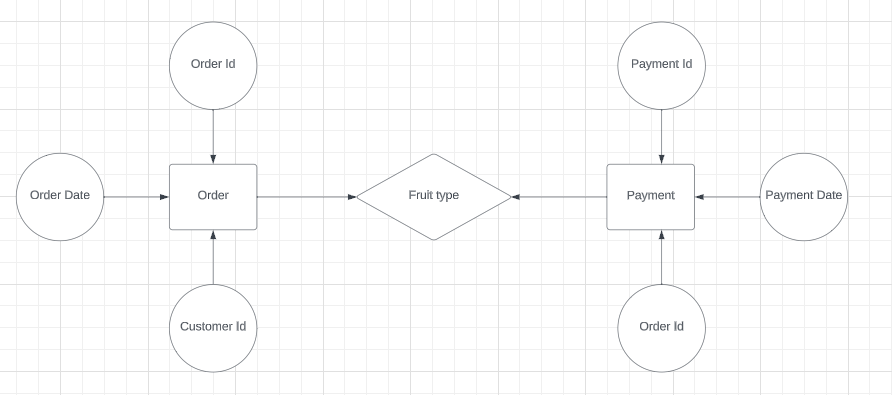
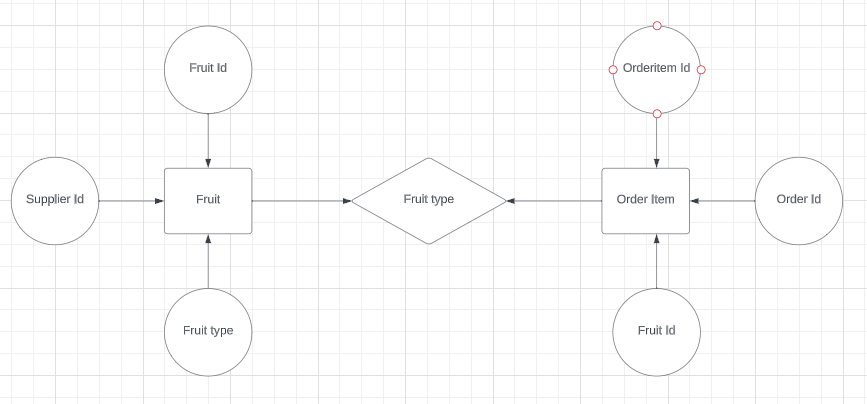
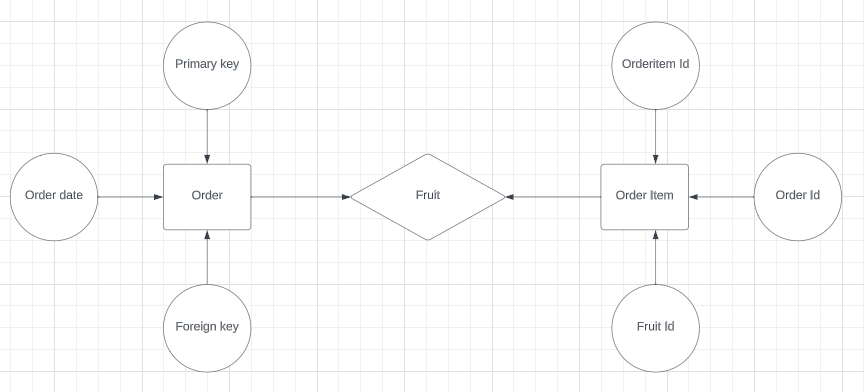
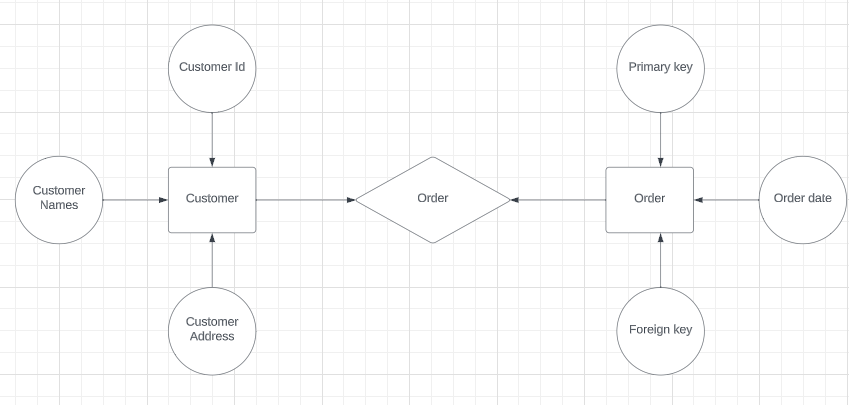
Supplier (Supplier ID, Name, Contact Person, Contact Email, Contact Phone, Address)

Customer (Customer ID, First Name, Last Name, Email, Phone Number, Address)

Order (Order ID, Customer ID, Order Date, Total Price, Status)

OrderItem (Order Item ID, Order ID, Fruit ID, Quantity, Subtotal Price)

Payment (Payment ID, Order ID, Payment Date, Payment Amount, Payment Method

1. **Create anERD**

SECTION II

**1. Create the database of fruit selling system**

-- Create the Customer table

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY AUTO\_INCREMENT,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100),

PhoneNumber VARCHAR(20),

Address VARCHAR(255)

);

-- Create the Supplier table

CREATE TABLE Supplier (

SupplierID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

ContactPerson VARCHAR(50),

ContactEmail VARCHAR(100),

ContactPhone VARCHAR(20),

Address VARCHAR(255)

);

-- Create the Fruit table

CREATE TABLE Fruit (

FruitID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Description TEXT,

Category VARCHAR(50),

PricePerUnit DECIMAL(10, 2) NOT NULL,

UnitOfMeasurement VARCHAR(20) NOT NULL,

StockQuantity INT NOT NULL,

SupplierID INT,

FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)

);

-- Create the Order table

CREATE TABLE Order (

OrderID INT PRIMARY KEY AUTO\_INCREMENT,

CustomerID INT NOT NULL,

OrderDate DATE NOT NULL,

TotalPrice DECIMAL(10, 2) NOT NULL,

OrderStatus VARCHAR(20) NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

-- Create the OrderItem table

CREATE TABLE OrderItem (

OrderItemID INT PRIMARY KEY AUTO\_INCREMENT,

OrderID INT NOT NULL,

FruitID INT NOT NULL,

Quantity INT NOT NULL,

SubtotalPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Order(OrderID),

FOREIGN KEY (FruitID) REFERENCES Fruit(FruitID)

);

-- Create the Payment table

CREATE TABLE Payment (

PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

OrderID INT NOT NULL,

PaymentDate DATE NOT NULL,

PaymentAmount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(50) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Order(OrderID)

);

**2. Write queries to create all the tables and relationships of fruit selling system**

-- Create the Customer table

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100),

PhoneNumber VARCHAR(20),

Address VARCHAR(255)

);

-- Create the Supplier table

CREATE TABLE Supplier (

SupplierID INT PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

ContactPerson VARCHAR(50),

ContactEmail VARCHAR(100),

ContactPhone VARCHAR(20),

Address VARCHAR(255)

);

-- Create the Fruit table

CREATE TABLE Fruit (

FruitID INT PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Description TEXT,

Category VARCHAR(50),

PricePerUnit DECIMAL(10, 2) NOT NULL,

UnitOfMeasurement VARCHAR(20) NOT NULL,

StockQuantity INT NOT NULL,

SupplierID INT,

FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)

);

-- Create the Order table

CREATE TABLE Order (

OrderID INT PRIMARY KEY,

CustomerID INT NOT NULL,

OrderDate DATE NOT NULL,

TotalPrice DECIMAL(10, 2) NOT NULL,

OrderStatus VARCHAR(20) NOT NULL,

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

-- Create the OrderItem table

CREATE TABLE OrderItem (

OrderItemID INT PRIMARY KEY,

OrderID INT NOT NULL,

FruitID INT NOT NULL,

Quantity INT NOT NULL,

SubtotalPrice DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Order(OrderID),

FOREIGN KEY (FruitID) REFERENCES Fruit(FruitID)

);

-- Create the Payment table

CREATE TABLE Payment (

PaymentID INT PRIMARY KEY,

OrderID INT NOT NULL,

PaymentDate DATE NOT NULL,

PaymentAmount DECIMAL(10, 2) NOT NULL,

PaymentMethod VARCHAR(50) NOT NULL,

FOREIGN KEY (OrderID) REFERENCES Order(OrderID)

);

**3. write queries to insert data into your tables in fruit selling system**

-- Insert data into the Supplier table

INSERT INTO Supplier (SupplierID, Name, ContactPerson, ContactEmail, ContactPhone, Address)

VALUES

(1, 'Fresh Fruits Inc.', 'John Smith', 'john@example.com', '123-456-7890', '123 Main St'),

(2, 'Tropical Delights', 'Jane Doe', 'jane@example.com', '987-654-3210', '456 Elm St');

-- Insert data into the Customer table

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, PhoneNumber, Address)

VALUES

(1, 'Alice', 'Johnson', 'alice@gmail.com', '555-123-4567', '789 Oak Ave'),

(2, 'Bob', 'Smith', 'bob@gmail.com', '555-987-6543', '101 Pine St');

-- Insert data into the Fruit table

INSERT INTO Fruit (FruitID, Name, Description, Category, PricePerUnit, UnitOfMeasurement, StockQuantity, SupplierID)

VALUES

(1, 'Apple', 'Red and delicious', 'Fruits', 0.75, 'each', 100, 1),

(2, 'Banana', 'Yellow and ripe', 'Fruits', 0.50, 'each', 150, 2);

-- Insert data into the Order table

INSERT INTO Order (OrderID, CustomerID, OrderDate, TotalPrice, OrderStatus)

VALUES

(1, 1, '2023-09-10', 25.50, 'Shipped'),

(2, 2, '2023-09-11', 15.75, 'Pending');

-- Insert data into the OrderItem table

INSERT INTO OrderItem (OrderItemID, OrderID, FruitID, Quantity, SubtotalPrice)

VALUES

(1, 1, 1, 5, 3.75),

(2, 1, 2, 10, 5.00),

(3, 2, 2, 5, 2.50);

-- Insert data into the Payment table

INSERT INTO Payment (PaymentID, OrderID, PaymentDate, PaymentAmount, PaymentMethod)

VALUES

(1, 1, '2023-09-10', 25.50, 'Credit Card'),

(2, 2, '2023-09-11', 15.75, 'PayPal');

**4. Write queries to display all the information in your tables of fruit selling system**

-- Display all information from the Supplier table

SELECT \* FROM Supplier;

-- Display all information from the Customer table

SELECT \* FROM Customer;

-- Display all information from the Fruit table

SELECT \* FROM Fruit;

-- Display all information from the Order table

SELECT \* FROM Order;

-- Display all information from the OrderItem table

SELECT \* FROM OrderItem;

-- Display all information from the Payment table

SELECT \* FROM Payment;

**5. Write a query to update information in any of the two tables of fruit selling system**

-- Update contact information for a specific supplier (SupplierID = 1)

UPDATE Supplier

SET ContactEmail = 'new\_email@example.com', ContactPhone = '555-555-5555'

WHERE SupplierID = 1;

In this query:

UPDATE Supplier specifies the table you want to update, which is the "Supplier" table.

SET ContactEmail = 'new\_email@example.com', ContactPhone = '555-555-5555' sets the new values for the ContactEmail and ContactPhone columns.

WHERE SupplierID = 1 specifies the condition for which rows to update. In this case, it targets the supplier with SupplierID equal to 1.

**SECTIONS III**

1. **Create a view to insert data into your tables of fruit selling system**

-- Create a stored procedure to insert data into the Customer table

DELIMITER //

CREATE PROCEDURE InsertCustomer(

IN FirstName VARCHAR(50),

IN LastName VARCHAR(50),

IN Email VARCHAR(100),

IN PhoneNumber VARCHAR(20),

IN Address VARCHAR(255)

)

BEGIN

INSERT INTO Customer (FirstName, LastName, Email, PhoneNumber, Address)

VALUES (FirstName, LastName, Email, PhoneNumber, Address);

END;

//

DELIMITER ;

-- Call the stored procedure to insert a new customer

CALL InsertCustomer('John', 'Doe', 'john@example.com', '555-123-4567', '123 Elm St');

1. **Create a view to display all the information in your tables of fruit selling system**

-- Create a view to display all information in the fruit selling system

CREATE VIEW FruitSellingSystemView AS

SELECT

C.CustomerID,

C.FirstName AS CustomerFirstName,

C.LastName AS CustomerLastName,

C.Email AS CustomerEmail,

C.PhoneNumber AS CustomerPhoneNumber,

C.Address AS CustomerAddress,

S.SupplierID,

S.Name AS SupplierName,

S.ContactPerson AS SupplierContactPerson,

S.ContactEmail AS SupplierContactEmail,

S.ContactPhone AS SupplierContactPhone,

S.Address AS SupplierAddress,

F.FruitID,

F.Name AS FruitName,

F.Description AS FruitDescription,

F.Category AS FruitCategory,

F.PricePerUnit AS FruitPricePerUnit,

F.UnitOfMeasurement AS FruitUnitOfMeasurement,

F.StockQuantity AS FruitStockQuantity,

O.OrderID,

O.OrderDate AS OrderDate,

O.TotalPrice AS OrderTotalPrice,

O.OrderStatus AS OrderStatus,

OI.OrderItemID,

OI.Quantity AS OrderItemQuantity,

OI.SubtotalPrice AS OrderItemSubtotalPrice,

P.PaymentID,

P.PaymentDate AS PaymentDate,

P.PaymentAmount AS PaymentAmount,

P.PaymentMethod AS PaymentMethod

FROM

Customer AS C

LEFT JOIN

Order AS O ON C.CustomerID = O.CustomerID

LEFT JOIN

OrderItem AS OI ON O.OrderID = OI.OrderID

LEFT JOIN

Fruit AS F ON OI.FruitID = F.FruitID

LEFT JOIN

Supplier AS S ON F.SupplierID = S.SupplierID

LEFT JOIN

Payment AS P ON O.OrderID = P.OrderID;

1. **Create a view to update information in any of the two tables of fruit selling system**

-- Create a view that combines data from Customer and Supplier tables

CREATE VIEW CustomerSupplierView AS

SELECT

C.CustomerID,

C.FirstName,

C.LastName,

C.Email AS CustomerEmail,

S.SupplierID,

S.Name AS SupplierName,

S.ContactEmail AS SupplierEmail

FROM

Customer AS C

LEFT JOIN

Supplier AS S ON C.CustomerID = S.SupplierID;

-- Create an INSTEAD OF trigger to handle updates through the view

DELIMITER //

CREATE TRIGGER UpdateCustomerSupplierViewInsteadOfUpdate

INSTEAD OF UPDATE ON CustomerSupplierView

FOR EACH ROW

BEGIN

IF OLD.CustomerID IS NOT NULL THEN

-- Update the Customer table

UPDATE Customer

SET Email = NEW.CustomerEmail

WHERE CustomerID = NEW.CustomerID;

ELSE

-- Update the Supplier table

UPDATE Supplier

SET ContactEmail = NEW.SupplierEmail

WHERE SupplierID = NEW.SupplierID;

END IF;

END;

//

DELIMITER ;

-- Update the email address of a customer using the view

UPDATE CustomerSupplierView

SET CustomerEmail = 'new\_customer\_email@example.com'

WHERE CustomerID = 1;

-- Update the email address of a supplier using the view

UPDATE CustomerSupplierView

SET SupplierEmail = 'new\_supplier\_email@example.com'

WHERE SupplierID = 1;

**4. Create a view to delete data in any two of your tables according to any simple**

**condition of your choice in fruit selling system**

-- Create a view to list customers and suppliers meeting the condition

CREATE VIEW CustomersSuppliersToDelete AS

SELECT

C.CustomerID,

S.SupplierID

FROM

Customer AS C

LEFT JOIN

Order AS O ON C.CustomerID = O.CustomerID

LEFT JOIN

Supplier AS S ON C.CustomerID = S.SupplierID

WHERE

O.OrderID IS NULL AND S.SupplierID IS NULL;

-- Create an INSTEAD OF trigger to handle DELETE operations through the view

DELIMITER //

CREATE TRIGGER DeleteCustomersSuppliersViewInsteadOfDelete

INSTEAD OF DELETE ON CustomersSuppliersToDelete

FOR EACH ROW

BEGIN

-- Delete customers and suppliers based on the condition

DELETE FROM Customer WHERE CustomerID = OLD.CustomerID;

DELETE FROM Supplier WHERE SupplierID = OLD.SupplierID;

END;

//

DELIMITER ;

-- Delete customers and suppliers meeting the condition using the view

DELETE FROM CustomersSuppliersToDelete;

**5.In your database of fruit selling system, create one view of your choice that considers sub-query**

-- Create a view to calculate total revenue per customer using a subquery

CREATE VIEW CustomerTotalRevenue AS

SELECT

C.CustomerID,

C.FirstName,

C.LastName,

C.Email,

C.PhoneNumber,

C.Address,

COALESCE(SUM(OrderTotal), 0) AS TotalRevenue

FROM

Customer AS C

LEFT JOIN (

SELECT

O.CustomerID,

SUM(O.TotalPrice) AS OrderTotal

FROM

Order AS O

GROUP BY

O.CustomerID

) AS Subquery ON C.CustomerID = Subquery.CustomerID

GROUP BY

C.CustomerID, C.FirstName, C.LastName, C.Email, C.PhoneNumber, C.Address;

-- Query the CustomerTotalRevenue view to get total revenue per customer

SELECT \* FROM CustomerTotalRevenue;

**SECTION IV**

1. **Create a stored procedure to insert data into your tables of fruit selling system**

-- Create a stored procedure to insert data into the Fruit table

DELIMITER //

CREATE PROCEDURE InsertFruit(

IN p\_Name VARCHAR(100),

IN p\_Description TEXT,

IN p\_Category VARCHAR(50),

IN p\_PricePerUnit DECIMAL(10, 2),

IN p\_UnitOfMeasurement VARCHAR(20),

IN p\_StockQuantity INT,

IN p\_SupplierID INT

)

BEGIN

INSERT INTO Fruit (Name, Description, Category, PricePerUnit, UnitOfMeasurement, StockQuantity, SupplierID)

VALUES (p\_Name, p\_Description, p\_Category, p\_PricePerUnit, p\_UnitOfMeasurement, p\_StockQuantity, p\_SupplierID);

END;

//

DELIMITER ;

-- Call the InsertFruit stored procedure to insert a new fruit

CALL InsertFruit('Orange', 'Juicy and sweet', 'Citrus', 0.75, 'each', 200, 1);

1. **Create a stored procedure to display all the information in your tables of fruit selling system**

-- Create a stored procedure to display all information from all tables

CREATE PROCEDURE DisplayAllInformation

AS

BEGIN

-- Declare variables

DECLARE @TableName NVARCHAR(255)

DECLARE @SqlQuery NVARCHAR(MAX)

-- Create a cursor to iterate through table names

DECLARE TableCursor CURSOR FOR

SELECT TABLE\_NAME

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_SCHEMA = 'dbo'; -- Replace with your schema name if necessary

-- Create a temporary table to store results

CREATE TABLE #Results (Table\_Name NVARCHAR(255), Details NVARCHAR(MAX))

-- Initialize variables

SET @SqlQuery = ''

-- Iterate through table names

OPEN TableCursor

FETCH NEXT FROM TableCursor INTO @TableName

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Build a dynamic SQL query to select all data from the current table

SET @SqlQuery = N'SELECT \* FROM ' + QUOTENAME(@TableName)

-- Insert the table name and query result into the temporary table

INSERT INTO #Results (Table\_Name, Details)

EXEC sp\_executesql @SqlQuery

FETCH NEXT FROM TableCursor INTO @TableName

END

-- Close and deallocate the cursor

CLOSE TableCursor

DEALLOCATE TableCursor

-- Select the results from the temporary table

SELECT \* FROM #Results

-- Clean up the temporary table

DROP TABLE #Results

END

1. **Create a stored procedure to update information in any of the two tables of fruit selling system**

-- Create a stored procedure to update customer and supplier information

DELIMITER //

CREATE PROCEDURE UpdateCustomerSupplierInfo(

IN p\_CustomerID INT,

IN p\_CustomerEmail VARCHAR(100),

IN p\_SupplierID INT,

IN p\_SupplierEmail VARCHAR(100)

)

BEGIN

-- Update the customer's email

IF p\_CustomerID IS NOT NULL AND p\_CustomerEmail IS NOT NULL THEN

UPDATE Customer

SET Email = p\_CustomerEmail

WHERE CustomerID = p\_CustomerID;

END IF;

-- Update the supplier's contact email

IF p\_SupplierID IS NOT NULL AND p\_SupplierEmail IS NOT NULL THEN

UPDATE Supplier

SET ContactEmail = p\_SupplierEmail

WHERE SupplierID = p\_SupplierID;

END IF;

END;

//

DELIMITER ;

-- Call the UpdateCustomerSupplierInfo stored procedure to update customer and supplier information

CALL UpdateCustomerSupplierInfo(1, 'new\_customer\_email@example.com', 2, 'new\_supplier\_email@example.com');

**4. Create a stored procedure to delete data in any two of your tables according to any**

**simple condition of your choice in fruit selling system**

-- Create a stored procedure to delete data from Customer and Supplier tables based on a condition

DELIMITER //

CREATE PROCEDURE DeleteCustomersSuppliersByCondition()

BEGIN

-- Delete customers who haven't placed any orders

DELETE FROM Customer

WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Order);

-- Delete suppliers who don't supply any fruits

DELETE FROM Supplier

WHERE SupplierID NOT IN (SELECT DISTINCT SupplierID FROM Fruit);

END;

//

DELIMITER ;

-- Call the DeleteCustomersSuppliersByCondition stored procedure to delete data

CALL DeleteCustomersSuppliersByCondition();

**5.In your database of fruit selling system, stored the procedure view of your choice that considers sub-query**

-- Create a stored procedure that calculates total revenue per customer using a subquery

DELIMITER //

CREATE PROCEDURE CalculateCustomerTotalRevenue()

BEGIN

SELECT

C.CustomerID,

C.FirstName,

C.LastName,

C.Email,

C.PhoneNumber,

C.Address,

COALESCE(SUM(O.TotalPrice), 0) AS TotalRevenue

FROM

Customer AS C

LEFT JOIN Order AS O ON C.CustomerID = O.CustomerID

GROUP BY

C.CustomerID, C.FirstName, C.LastName, C.Email, C.PhoneNumber, C.Address;

END;

//

DELIMITER ;

-- Call the CalculateCustomerTotalRevenue stored procedure to get total revenue per customer

CALL CalculateCustomerTotalRevenue();

**SECTION V**

1. **Create after inserting triggers for any two tables of your choice in fruit selling system**

AFTER INSERT trigger for the "Order" table: This trigger will update the stock quantity of fruits in the "Fruit" table when a new order is placed. -- Create an AFTER INSERT trigger for the Order table

DELIMITER //

CREATE TRIGGER UpdateStockQuantityAfterOrder

AFTER INSERT ON `Order`

FOR EACH ROW

BEGIN

DECLARE @FruitID INT;

DECLARE @Quantity INT;

-- Get the FruitID and Quantity from the newly inserted order

SET @FruitID = NEW.FruitID;

SET @Quantity = NEW.Quantity;

-- Update the stock quantity in the Fruit table

UPDATE Fruit

SET StockQuantity = StockQuantity - @Quantity

WHERE FruitID = @FruitID;

END;

//

DELIMITER ;

AFTER INSERT trigger for the "Payment" table: This trigger will update the order status in the "Order" table when a new payment is recorded.

-- Create an AFTER INSERT trigger for the Payment table

DELIMITER //

CREATE TRIGGER UpdateOrderStatusAfterPayment

AFTER INSERT ON Payment

FOR EACH ROW

BEGIN

DECLARE @OrderID INT;

-- Get the OrderID from the newly inserted payment

SET @OrderID = NEW.OrderID;

-- Update the order status in the Order table

UPDATE `Order`

SET OrderStatus = 'Paid'

WHERE OrderID = @OrderID;

END;

//

DELIMITER ;

1. **Create after-update triggers for any two tables of your choice in fruit selling system**

AFTER UPDATE trigger for the "Order" table: This trigger will update the order status in the "Payment" table when the order status in the "Order" table is changed.

-- Create an AFTER UPDATE trigger for the Order table

DELIMITER //

CREATE TRIGGER UpdatePaymentStatusAfterOrderUpdate

AFTER UPDATE ON `Order`

FOR EACH ROW

BEGIN

DECLARE @NewStatus VARCHAR(50);

DECLARE @OrderID INT;

-- Get the new order status and OrderID after update

SET @NewStatus = NEW.OrderStatus;

SET @OrderID = NEW.OrderID;

-- Update the payment status in the Payment table

UPDATE Payment

SET PaymentStatus = @NewStatus

WHERE OrderID = @OrderID;

END;

//

DELIMITER ;

AFTER UPDATE trigger for the "Payment" table: This trigger will update the payment date in the "Order" table when the payment date in the "Payment" table is changed.

-- Create an AFTER UPDATE trigger for the Payment table

DELIMITER //

CREATE TRIGGER UpdateOrderPaymentDateAfterPaymentUpdate

AFTER UPDATE ON Payment

FOR EACH ROW

BEGIN

DECLARE @NewPaymentDate DATE;

DECLARE @OrderID INT;

-- Get the new payment date and OrderID after update

SET @NewPaymentDate = NEW.PaymentDate;

SET @OrderID = NEW.OrderID;

-- Update the payment date in the Order table

UPDATE `Order`

SET PaymentDate = @NewPaymentDate

WHERE OrderID = @OrderID;

END;

//

DELIMITER ;

1. **Create after deleting triggers for any two tables of your choice in fruit selling system**

AFTER DELETE trigger for the "Order" table: This trigger will update the stock quantity of fruits in the "Fruit" table when an order is deleted.

-- Create an AFTER DELETE trigger for the Order table

DELIMITER //

CREATE TRIGGER UpdateStockQuantityAfterOrderDelete

AFTER DELETE ON `Order`

FOR EACH ROW

BEGIN

DECLARE @FruitID INT;

DECLARE @Quantity INT;

-- Get the FruitID and Quantity from the deleted order

SET @FruitID = OLD.FruitID;

SET @Quantity = OLD.Quantity;

-- Update the stock quantity in the Fruit table

UPDATE Fruit

SET StockQuantity = StockQuantity + @Quantity

WHERE FruitID = @FruitID;

END;

//

DELIMITER ;

AFTER DELETE trigger for the "Customer" table: This trigger will delete any associated orders when a customer is deleted from the "Customer" table.

-- Create an AFTER DELETE trigger for the Customer table

DELIMITER //

CREATE TRIGGER DeleteOrdersAfterCustomerDelete

AFTER DELETE ON Customer

FOR EACH ROW

BEGIN

DECLARE @CustomerID INT;

-- Get the CustomerID from the deleted customer

SET @CustomerID = OLD.CustomerID;

-- Delete associated orders in the Order table

DELETE FROM `Order`

WHERE CustomerID = @CustomerID;

END;

//

DELIMITER ;

**SECTION VI**

**1. Create a user with your name as username and your student number as password**

**and grant all privileges to the created user.**

CREATE USER 'Twahirwa\_egide'@'localhost' IDENTIFIED BY '222007743';

GRANT ALL PRIVILEGES ON your\_database\_name.\* TO 'Twahirwa\_egide'@'localhost';

**2. Create a user with your "names\_semi" as username and your student number as**

**password and give him insert, update, and delete privileges to the created user.**

CREATE USER 'Twahirwa\_egide'@'localhost' IDENTIFIED BY '222007743';

GRANT INSERT, UPDATE, DELETE ON your\_database\_name.\* TO 'Twahirwa\_egide'@'localhost';

**3.Revoke insert privileges to the last user you created.**

REVOKE INSERT ON your\_database\_name.\* FROM 'Twahirwa\_egide'@'localhost';