THE CATHOLIC UNIVERSITY OF EASTERN AFRICA

FACULTY OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

GROUP 11

PROJECT TITLE:

ONLINE SHOPPING SYSTEM

GROUP MEMBERS:

HARRIET MUMBUA-1049499

SHERYL ROLANDA-1048855

NATALIE MONDA-1049082

LAURA NASIMIYU-1049510

TEDDY GITONGA-1049409

SUBMITTED ON 19TH OCTOBER 2024

INTRODUCTION

Overview

The Online Shopping System aims to provide a platform that manages product catalogs, orders, and payments. This system will allow customers to search for products, place orders, and make payments online.

Rationale

Over the recent years, e-commerce has shown a tremendous growth. Many businesses have migrated online while others run both physically and online this creates a pressing need for the business to have efficient and reliable database systems so as to provide real-time data that is accurate. This Online Shopping System project aims to come up with a solution as it focuses on managing the product catalogs, orders made by customers and payments. The focus is on ensuring seamless transactions, and data integrity.

Objectives

i. Design and develop an efficient database for an online shopping system.
ii. To effectively implement CRUD Operations-Create, Read, Update, Delete.
iii. Develop triggers and stored procedures to automate specific operations
iv. To test the database system through various operations.

SYSTEM DESIGN

ER Diagrams

Entities in the ER Diagram include:

i. Customer- stores information about the customers such as Customer ID, Name, Email, Address and Contact.
ii. Order- represents the orders placed by a customer
iii. OrderItem- Contains the details of the products that are ordered
iv. Shoe- This is the product chosen for this project it represents the available products for sale
v. Payment- Represents the payments made for the orders by the customers.

Relationships within the Diagram:

i. Customer and Order:

- Relationship is One-to-Many
- A single customer can place multiple orders, but each order is associated with one customer.

ii. Order and OrderItem:

- Relationship is One-to-Many
- Each order can have multiple items in it, but each OrderItem belongs to a single order.

iii. OrderItem and Shoe:

- Relationship is Many-to-One

- Each order item is linked to a specific shoe, but a shoe can appear in multiple order items.

iv. Order and Payment:

- Relationship is One-to-One
- Each order can have one payments associated with it.

v. Shoe and OrderItem:

- Relationship is One-to-Many
- Each shoe can be part of multiple order items in different orders.

The Entity Relationship Diagram:

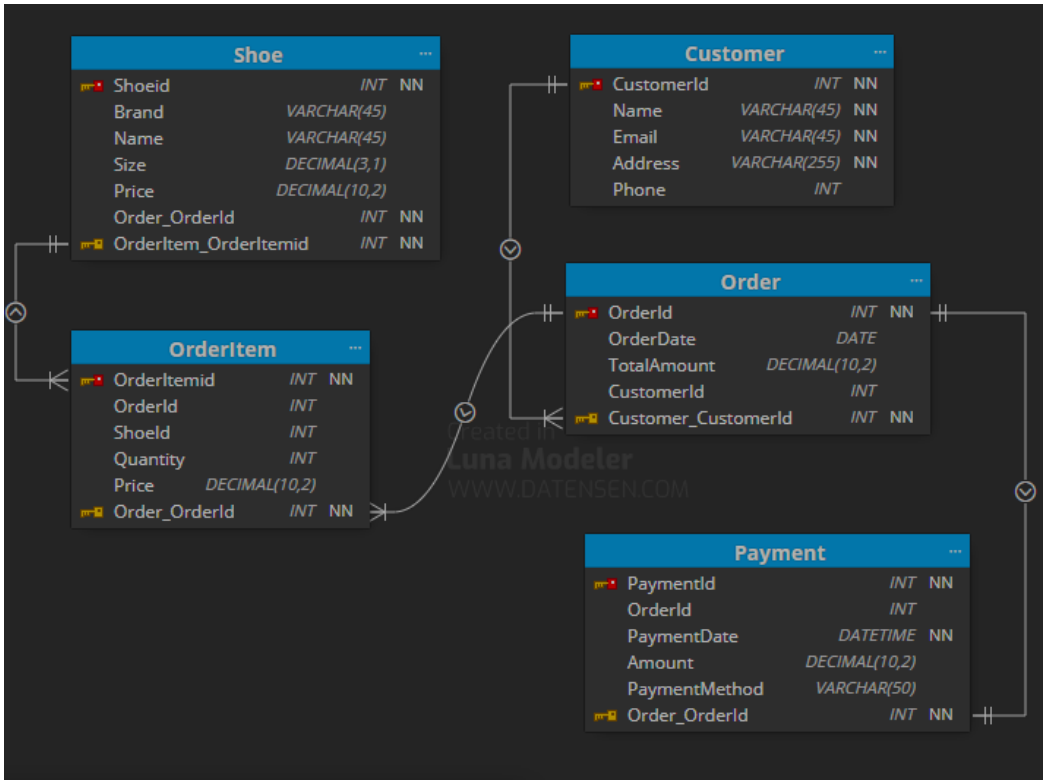


Table structures

1. Table shoe- stores information about the product such as the Brand, name, size, price

```
CREATE TABLE `shoe` (
  `Shoeid` int NOT NULL,
  `Brand` varchar(45) DEFAULT NULL,
  `Name` varchar(45) DEFAULT NULL,
  `Size` decimal(3,1) DEFAULT NULL,
  `Price` decimal(10,2) DEFAULT NULL,
  `Orders_OrderId` int DEFAULT NULL,
  `OrderItem_OrderItemid` int NOT NULL,
  PRIMARY KEY (`Shoeid`),
  KEY `OrderItem_Shoe` (`OrderItem_OrderItemid`),
  CONSTRAINT `OrderItem_Shoe` FOREIGN KEY (`OrderItem_OrderItemid`)
REFERENCES `orderitem` (`OrderItemid`)
```

2. Table Customer- stores information about the customer

```
CREATE TABLE `customer` (
  `CustomerId` int NOT NULL,
  `Name` varchar(45) NOT NULL,
  `Email` varchar(45) NOT NULL,
  `Address` varchar(255) NOT NULL,
  `Phone` int DEFAULT NULL,
  PRIMARY KEY (`CustomerId`)
```

3. Table OrderItem- stores information about a product ordered by a customer

```
CREATE TABLE `orderitem` (
  `OrderItemid` int NOT NULL,
  `OrderId` int DEFAULT NULL,
  `ShoeId` int DEFAULT NULL,
  `Quantity` int DEFAULT NULL,
  `Price` decimal(10,2) DEFAULT NULL,
  `Orders_OrderId` int DEFAULT NULL,
  PRIMARY KEY (`OrderItemid`),
  KEY `Order_OrderItem` (`Orders_OrderId`),
  CONSTRAINT `Order_OrderItem` FOREIGN KEY (`Orders_OrderId`)
REFERENCES `orders` (`OrderId`)
```

4. Table Orders- this stores information about the entire order that a customer makes

```
CREATE TABLE `orders` (
  `OrderId` int NOT NULL,
  `OrderDate` date DEFAULT NULL,
  `TotalAmount` decimal(10,2) DEFAULT NULL,
  `CustomerId` int DEFAULT NULL,
  `Customer_CustomerId` int NOT NULL,
  PRIMARY KEY (`OrderId`),
  KEY `Customer_Order` (`Customer_CustomerId`),
  CONSTRAINT `Customer_Order` FOREIGN KEY (`Customer_CustomerId`)
REFERENCES `customer` (`CustomerId`)
```

5. Table Payment- stores information of the payment details of a product

```
CREATE TABLE `payment` (
  `PaymentId` int NOT NULL,
  `OrderId` int DEFAULT NULL,
  `PaymentDate` datetime NOT NULL,
  `Amount` decimal(10,2) DEFAULT NULL,
  `PaymentMethod` varchar(50) DEFAULT NULL,
  `Orders_OrderId` int DEFAULT NULL,
  PRIMARY KEY (`PaymentId`),
  KEY `Order_Payment` (`Orders_OrderId`),
  CONSTRAINT `Order_Payment` FOREIGN KEY (`Orders_OrderId`) REFERENCES
`orders` (`OrderId`)
```

IMPLEMENTATION

CRUD Operations

This includes Create, Read, Update and Delete Operations. Below are sample code snippets:

1. **CREATE**

a) Adding a new customer:

```
INSERT INTO
Customer(CustomerId,Name,Email,Address,Phone)VALUES(006,'James','yeh','Eldoret',0757
58435);
```

b) Adding a new product-shoe:

INSERT INTO
Shoe(ShoeId,Brand,Name,Size,Price,Orders_OrderId,OrderItem_OrderItemId)VALUES(006,'Nike','Red',27.2,2345.99,003,002);

2. **READ**

a) Retrieving an order:

SELECT * FROM Orders WHERE OrderId=1;

b) Retrieving order details for a specific customer:

SELECT * FROM Orders WHERE Customer_CustomerId = 1;

3. **UPDATE**

a) Updating the Amount column in the payment table:

UPDATE payment SET Amount=6473.88 WHERE PaymentId=003;

b) Updating the Name column in the customer table

UPDATE Customer SET Name='Ken' WHERE CustomerId=005;

4. **DELETE**

a) Removing a specific order:

DELETE FROM Orders Where Orderid=006;

b) Removing a specific Customer:

DELETE FROM Customer Where Customerid=006;

Advanced SQL queries

1. Customers with Highest Number of Products Ordered*/

SELECT c.CustomerId, c.Name AS CustomerName, SUM(oi.Quantity) AS TotalQuantityOrdered FROM Customer c JOIN Orders o ON c.CustomerId = o.Customer_CustomerId JOIN OrderItem oi ON o.OrderId = oi.OrderId GROUP BY c.CustomerId ORDER BY TotalQuantityOrdered DESC LIMIT 10;

2. Monthly revenue per shoe brand and its trend, including a running total for each shoe.

WITH ShoeRevenue AS (SELECT s.ShoeId, s.Brand, DATE_FORMAT(o.OrderDate, '%Y-%m') AS Month, SUM(oi.Quantity * oi.Price) AS Revenue FROM Shoe s JOIN OrderItem oi ON s.ShoeId = oi.ShoeId JOIN Orders o ON oi.OrderId = o.OrderId GROUP BY s.ShoeId, s.Brand, Month) SELECT ShoeId,Brand,Month, Revenue, SUM(Revenue) OVER (PARTITION BY ShoeId ORDER BY Month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal FROM ShoeRevenue ORDER BY ShoeId, Month DESC;

TESTING AND VALIDATION

1. **Testing Methodology**

The online shopping system was thoroughly tested to validate its functionality and ensure all features worked as intended. The testing process involved:

- Functional Testing: Each feature the CRUD operations, advanced SQL queries, views and reports were tested to ensure they performed their intended tasks.

- Integration Testing: Relationships between the tables were validated to ensure data integrity.
- User Acceptance Testing: The system was evaluated using sample data to mimic real-world use cases.

2. **Testing Scenarios:**

i. **Adding a New Customer:**
   - A new customer was added using the INSERT operation in the customer table.
   - The customer details were successfully saved in the database and verified through a SELECT query.

ii. **Placing an Order:**
   - A new order was created by inserting a record into the orders table with a valid CustomerId.
   - The order was successfully saved, and its details were linked to the customer in the database.
   - Verified the relationship between orders and customer using a JOIN query.

iii. **Adding Items to an Order:**
   - Items were added to an order by inserting records into the Orderitem table, linking them to the respective order and shoe.
   - Order items were correctly linked to the corresponding order and shoe records.

iv. **Processing a Payment:**
   - A payment record was added to the payment table, linked to an existing order.
   - The payment details were saved successfully, with the correct OrderId.
   - Checked that each order had only one payment associated with it.

v. **Fetching Customer Orders:**
   - A query was executed to retrieve all orders for a specific customer using their CustomerId.
   - The query returned accurate results, displaying all orders along with the order date and total amount.

vi. **Calculating Total Order Amount:**
   - The total order amount was calculated using the SUM function on the Price field in the orderitem table.
   - The total was calculated accurately and matched manual calculations.
   - Confirmed that the result included all relevant order items and accounted for their quantities.

3. **Testing Tools and Sample Data**
   Tool used to execute the SQL queries was the MySQL Workbench CommandLine
   A predefined dataset was created with shoe (Product), customer, order, order item, and payment details for testing purposes.

CONCLUSION AND RECOMMENDATIONS

This project successfully designed and implemented a database for an online shopping system, addressing essential operations like customer management, order processing, and payment tracking. The database ensures data integrity and supports advanced queries for analytics.

Future Improvements

- Enhance security features for payment processing.
- Expand database scalability for larger datasets.

References

Soegoto, E. S., & Suripto, A. (2018, August). Design of E-commerce Information System on Web-based Online Shopping. In *IOP Conference Series: Materials Science and Engineering* (Vol. 407, No. 1, p. 012008). IOP Publishing.

Sultana, A. (2017). Online shopping management system.

Lee, Z. J., Su, Z. Y., Cheng, X., Chen, Z. Z., Lian, Z. X., Wu, J. Y., & Qiu, R. F. (2020). Design an online shopping store based on opencart. *Artificial Intelligence Evolution*, 1-7.

mysql workbench documentation - Bing. (n.d.). Bing. https://www.bing.com/search?q=mysql+workbench+documentation&filters=dtbk:%22MCFjZ192NV9kb2N1bWVudGF0aW9uIWNnX3Y1X2RvY3VtZW50YXRpb24hNjcyYWE1MGItMWMwMy1jNjQwLTQ2ZjEtNWI3OTU0OTg0Yjg2%22+sid:%22672aa50b-1c03-c640-46f1-5b7954984b86%22&FORM=DEPNA

APPENDICES

APPENDIX A: CODE SNIPPETS

REPORTS:

i. Report to show the total revenue generated for each month

```
SELECT DATE_FORMAT(OrderDate, '%Y-%m') AS Month, SUM(TotalAmount)
AS MonthlyRevenue FROM Orders GROUP BY Month ORDER BY Month DESC;
```

ii. Report to show the total revenue collected by each payment method and the transactions made by each payment method

```
SELECT p.PaymentMethod,COUNT(p.PaymentId) AS TransactionsCount,
SUM(p.Amount) AS TotalRevenue FROM Payment p GROUP BY
p.PaymentMethod ORDER BY TotalRevenue DESC;
```

VIEWS:

i. View for customers with pending payment:

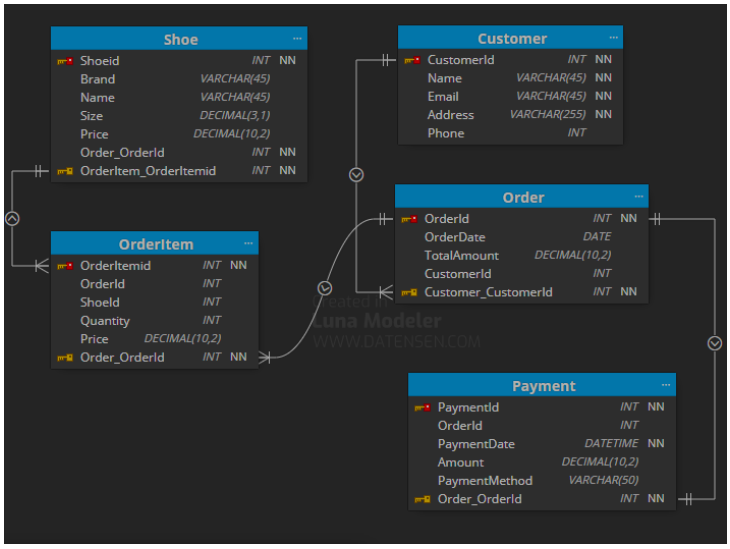
```
CREATE VIEW CustomersWithPendingPayments AS SELECT c.CustomerId,
c.Name AS CustomerName,c.Email,o.OrderId,o.TotalAmount,
IFNULL(SUM(p.Amount), 0) AS PaidAmount,(o.TotalAmount -
IFNULL(SUM(p.Amount), 0)) AS PendingAmount FROM customer c JOIN orders
o ON c.CustomerId = o.Customer_CustomerId LEFT JOIN payment p ON
o.OrderId = p.Orders_OrderId GROUP BY o.OrderId HAVING PendingAmount >
0;
```

ii. View for the payment summary:

```
CREATE VIEW PaymentSummary AS SELECT p.PaymentId,
p.OrderId,o.TotalAmount AS OrderTotal, p.PaymentDate, p.Amount AS
PaymentAmount,p.PaymentMethod FROM payment p JOIN orders o ON
p.Orders_OrderId = o.OrderId;
```

APPENDIX 2: DIAGRAMS AND EXECUTION SNIPPETS

1. ENTITY RELATIONSHIP DIAGRAM:



2. EXECUTION:

i. Inserting a product-shoe

```
mysql> INSERT INTO Shoe(ShoeId,Brand,Name,Size,Price,Orders_OrderId,OrderItem_OrderItemId)VALUES(007,'Nike','Shox R4',28.0,3890.00,003,002);
Query OK, 1 row affected (0.18 sec)

mysql> SELECT * FROM Shoe;
+--------+--------+---------+------+---------+---------------+----------------------+
| Shoeid | Brand  | Name    | Size | Price   | Orders_OrderId | OrderItem_OrderItemid |
+--------+--------+---------+------+---------+---------------+----------------------+
|      1 | Nike   | Dunks   | 33.2 | 1762.99 |             2 |                    1 |
|      2 | Nike   | Blazers | 27.5 |  326.33 |             3 |                    4 |
|      3 | Puma   | Deviate | 32.5 | 2353.99 |             4 |                    3 |
|      4 | Adidas | Samba   | 24.7 | 2356.44 |             5 |                    2 |
|      5 | Adidas | Gazelle | 35.2 | 3764.66 |             2 |                    4 |
|      6 | Nike   | Red     | 27.2 | 2345.99 |             3 |                    2 |
|      7 | Nike   | Shox R4 | 28.0 | 3890.00 |             3 |                    2 |
+--------+--------+---------+------+---------+---------------+----------------------+
7 rows in set (0.01 sec)

mysql>
```

ii. Customers with Highest Number of Products Ordered:

```
mysql> SELECT c.CustomerId, c.Name AS CustomerName, SUM(oi.Quantity) AS TotalQuantityOrdered FROM Customer c JOIN Orders o ON c.CustomerId = o.Custo
mer_CustomerId JOIN OrderItem oi ON o.OrderId = oi.OrderId GROUP BY c.CustomerId ORDER BY TotalQuantityOrdered DESC LIMIT 10;
+------------+--------------+---------------------+
| CustomerId | CustomerName | TotalQuantityOrdered |
+------------+--------------+---------------------+
|          3 | Jane         |                  10 |
|          4 | Mary         |                   6 |
|          2 | Peter        |                   5 |
|          1 | John         |                   4 |
+------------+--------------+---------------------+
4 rows in set (0.00 sec)

mysql>
```

iii. monthly revenue per shoe brand and its trend, including a running total for each shoe.

```
mysql> WITH ShoeRevenue AS ( SELECT s.ShoeId, s.Brand, DATE_FORMAT(o.OrderDate, '%Y-%m') AS Month, SUM(oi.Quantity * oi.Price) AS Revenue FROM Shoe
s JOIN OrderItem oi ON s.ShoeId = oi.ShoeId JOIN Orders o ON oi.OrderId = o.OrderId GROUP BY s.ShoeId, s.Brand, Month ) SELECT ShoeId,Brand,Month, R
evenue, SUM(Revenue) OVER (PARTITION BY ShoeId ORDER BY Month ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal FROM ShoeRevenue ORD
ER BY ShoeId, Month DESC;
+-------+--------+---------+-----------+-------------+
| ShoeId | Brand  | Month   | Revenue   | RunningTotal |
+-------+--------+---------+-----------+-------------+
|      1 | Nike   | 2024-04 |  271836.10 |    271836.10 |
|      2 | Nike   | 2023-11 | 1101868.32 |   1101868.32 |
|      3 | Puma   | 2024-04 |   21637.75 |     21637.75 |
|      4 | Adidas | 2024-09 |  219367.75 |    314741.95 |
|      4 | Adidas | 2024-07 |   95374.20 |     95374.20 |
|      5 | Adidas | 2023-11 |   10500.00 |     10500.00 |
+-------+--------+---------+-----------+-------------+
6 rows in set (0.00 sec)

mysql>
```

iv. Show the most profitable shoes by brand, categorized by customer type- first-time customers or returning customers

```
mysql> WITH ShoeProfitability AS (
    ->     SELECT
    ->         s.Brand,
    ->         s.ShoeId,
    ->         SUM(oi.Quantity * oi.Price) AS Revenue,
    ->         c.CustomerId,
    ->         CASE
    ->             WHEN COUNT(o.OrderId) = 1 THEN 'First-Time Customer'
    ->             ELSE 'Returning Customer'
    ->         END AS CustomerType
    ->     FROM Shoe s
    ->     JOIN OrderItem oi ON s.ShoeId = oi.ShoeId
    ->     JOIN Orders o ON oi.OrderId = o.OrderId
    ->     JOIN Customer c ON o.Customer_CustomerId = c.CustomerId
    ->     GROUP BY s.Brand, s.ShoeId, c.CustomerId
    -> )
    -> SELECT
    ->     Brand,
    ->     ShoeId,
    ->     CustomerType,
    ->     SUM(Revenue) AS TotalRevenue
    -> FROM ShoeProfitability
    -> GROUP BY Brand, ShoeId, CustomerType
    -> ORDER BY TotalRevenue DESC;
+--------+--------+---------------------+--------------+
| Brand  | ShoeId | CustomerType        | TotalRevenue |
+--------+--------+---------------------+--------------+
| Nike   |      2 | First-Time Customer |   1101868.32 |
| Adidas |      4 | First-Time Customer |    314741.95 |
| Nike   |      1 | First-Time Customer |    271836.10 |
| Puma   |      3 | First-Time Customer |     21637.75 |
| Adidas |      5 | First-Time Customer |     10500.00 |
+--------+--------+---------------------+--------------+
5 rows in set (0.00 sec)
```