Q.4) **Retail Point of Sale (POS) and Inventory System Database**

This database is designed to manage products, sales transactions, inventory, customer details, and supplier information in a retail store. It supports essential retail functions such as processing sales, applying discounts, managing stock levels, and generating reports for decision-making.

### 1. Database Tables Design

**Table: Customers Table**

**This table stores customer details to track sales transactions and customer behavior.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| customer\_id | INT (PK) | Unique identifier for each customer |
| first\_name | VARCHAR(50) | Customer's first name |
| last\_name | VARCHAR(50) | Customer's last name |
| phone\_number | VARCHAR(15) | Customer's phone number |
| email | VARCHAR(100) | Customer's email address |
| address | VARCHAR(255) | Customer's address |

**Table: Products Table**

**This table stores details about the products available in the store.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| product\_id | INT (PK) | Unique identifier for each product |
| product\_name | VARCHAR(100) | Name of the product |
| category | VARCHAR(50) | Product category (e.g., electronics, clothing, etc.) |
| price | DECIMAL(10,2) | Price of the product |
| supplier\_id | INT (FK) | References Suppliers(supplier\_id) |
| stock\_quantity | INT | Quantity of the product in stock |
| reorder\_level | INT | Stock level at which reordering is triggered |

**Table: Transactions Table**

**This table stores details about each sale transaction, including the customer, products, and total amount.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| transaction\_id | INT (PK) | Unique identifier for each transaction |
| customer\_id | INT (FK) | References Customers(customer\_id) |
| transaction\_date | TIMESTAMP | Date and time of the transaction |
| total\_amount | DECIMAL(10,2) | Total amount for the transaction |

**Table: Inventory Table**

**This table tracks the inventory adjustments and movements for products (e.g., restocking, sales, returns).**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| inventory\_id | INT (PK) | Unique identifier for each inventory transaction |
| product\_id | INT (FK) | References Products(product\_id) |
| quantity\_changed | INT | Quantity of products added/removed |
| change\_type | VARCHAR(20) | Type of change: 'sale', 'restock', 'return' |
| change\_date | TIMESTAMP | Date and time of the inventory change |

**Table: Suppliers Table**

**This table contains information about the suppliers who provide the products to the store.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| supplier\_id | INT (PK) | Unique identifier for each supplier |
| supplier\_name | VARCHAR(100) | Name of the supplier |
| contact\_info | VARCHAR(100) | Supplier's contact information |
| address | VARCHAR(255) | Supplier's address |

**Table: Promotions Table**

**This table stores promotional information, such as discounts and offers available for certain products.**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| promotion\_id | INT (PK) | Unique identifier for each promotion |
| promotion\_name | VARCHAR(100) | Name of the promotion |
| discount\_percentage | DECIMAL(5,2) | Discount percentage for the promotion |
| start\_date | DATE | Start date of the promotion |
| end\_date | DATE | End date of the promotion |

### 2. Constraints for Referential Integrity

### ** Foreign Keys:**

### **customer\_id in Transactions references Customers(customer\_id).**

### **supplier\_id in Products references Suppliers(supplier\_id).**

### **product\_id in Inventory references Products(product\_id).**

### ** Primary Keys:**

### **Each table has a primary key for unique identification of rows.**

### ** Check Constraints:**

### **Ensure non-negative values for stock\_quantity and quantity\_changed.**

### **Apply date checks for promotions, ensuring start\_date is before end\_date.**

### 3. Stored Procedures

#### a. Process Sale

#### **This procedure records a sale transaction, updates the product stock, and generates a sales entry in the Transactions table.**

#### CREATE PROCEDURE ProcessSale(IN custId INT, IN productId INT, IN quantity INT)

#### BEGIN

#### DECLARE total DECIMAL(10,2);

#### DECLARE productPrice DECIMAL(10,2);

#### DECLARE currentStock INT;

#### -- Get product price and current stock

#### SELECT price, stock\_quantity INTO productPrice, currentStock FROM Products WHERE product\_id = productId;

#### -- Ensure enough stock is available

#### IF currentStock >= quantity THEN

#### -- Calculate total

#### INSERT INTO Transactions (customer\_id, transaction\_date, total\_amount)

#### VALUES (custId, NOW(), total);

#### -- Update inventory (reduce stock)

#### INSERT INTO Inventory (product\_id, quantity\_changed, change\_type, change\_date)

#### VALUES (productId, -quantity, 'sale', NOW())

#### -- Update Products table stock quantity

#### UPDATE Products SET stock\_quantity = stock\_quantity - quantity WHERE product\_id = productId;

#### ELSE

#### SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient stock for this product.';

#### END IF;

#### END;

#### b. Apply Discount

**This procedure applies a discount to a product during a promotion.**

#### CREATE PROCEDURE ApplyDiscount(IN productId INT, IN promotionId INT)

#### BEGIN

#### DECLARE discount DECIMAL(5,2);

#### DECLARE originalPrice DECIMAL(10,2);

#### DECLARE newPrice DECIMAL(10,2);

#### -- Get discount percentage and original price

#### SELECT discount\_percentage INTO discount FROM Promotions WHERE promotion\_id = promotionId;

#### SELECT price INTO originalPrice FROM Products WHERE product\_id = productId;

#### -- Apply discount

#### SET newPrice = originalPrice - (originalPrice \* discount / 100)

#### -- Update product price

#### UPDATE Products SET price = newPrice WHERE product\_id = productId;

#### END;

#### c. Restock Inventory

### **This procedure restocks a product by adding more stock.**

### CREATE PROCEDURE RestockInventory(IN productId INT, IN quantity INT)

### BEGIN

### -- Update product stock in Products table

### UPDATE Products SET stock\_quantity = stock\_quantity + quantity WHERE product\_id = productId;

### -- Record the inventory change

### INSERT INTO Inventory (product\_id, quantity\_changed, change\_type, change\_date)

### VALUES (productId, quantity, 'restock', NOW());

### END;

### 4. Triggers

#### a. Trigger to Update Inventory upon Sale

**This trigger ensures that stock levels are automatically updated in the inventory when a sale occurs.**

#### CREATE TRIGGER AfterSaleInsert

#### AFTER INSERT ON Transactions

#### FOR EACH ROW

#### BEGIN

#### DECLARE productId INT;

#### DECLARE quantitySold INT;

#### 

#### -- Assume each transaction contains only one product

#### SELECT product\_id, quantity INTO productId, quantitySold FROM Sales WHERE transaction\_id = NEW.transaction\_id;

#### 

#### -- Update inventory (reduce stock)

#### INSERT INTO Inventory (product\_id, quantity\_changed, change\_type, change\_date)

#### VALUES (productId, -quantitySold, 'sale', NOW());

#### -- Update product stock in Products table

#### UPDATE Products SET stock\_quantity = stock\_quantity - quantitySold WHERE product\_id = productId;

#### END;

#### b. Trigger for Low Stock Reordering

**This trigger automatically restocks products when their stock reaches the reorder level.**

CREATE TRIGGER CheckStockLevel

AFTER UPDATE ON Products

FOR EACH ROW

BEGIN

IF NEW.stock\_quantity <= NEW.reorder\_level THEN

-- Call a procedure to reorder the product

CALL RestockInventory(NEW.product\_id, 50); -- Assuming 50 units for reordering

END IF;

END;

### 5. SQL Queries for Reports

#### a. Sales Trends Report

#### SELECT YEAR(transaction\_date) AS year, MONTH(transaction\_date) AS month, SUM(total\_amount) AS total\_sales

#### FROM Transactions

#### GROUP BY year, month

#### ORDER BY year DESC, month DESC;

#### b. Popular Products Report

#### SELECT p.product\_name, SUM(s.quantity) AS total\_quantity\_sold

#### Conclusion

#### The Retail Point of Sale (POS) and Inventory System provides a robust database solution for efficiently managing retail operations. By integrating sales transactions, product inventory, customer data, and supplier management, the system ensures smooth day-to-day retail functions. Key features include automatic stock updates with sales, streamlined order processing, and real-time inventory tracking to avoid stock-outs. Additionally, stored procedures and triggers automate critical tasks like applying discounts, restocking, and updating inventory levels. Comprehensive reporting capabilities offer valuable insights into sales trends, popular products, and supplier performance, empowering store managers to make data-driven decisions for optimal operations and customer satisfaction.