**CSA0593**

**DATABASE MANAGEMENT SYSTEM**

SWETHA.K

192311404

ASSIGNMENT-1

**Product Inventory and Sales Forecasting System:-**

1). Design a database to support inventory management and sales forecasting for a retail store.

Requirements:

Model tables for products, suppliers, sales, and inventory levels.

2). Write SQL queries to analyse historical sales data and predict future inventory needs.

3). Implement a stored procedure to reorder stock based on sales trends and low inventory levels.

4). Write views for management to see forecasted sales by product category.

1). To build a database for a Product Inventory and Sales Forecasting System, here’s a structured approach with tables, sample SQL queries, stored procedure, and views for management reporting:

**Products Table**: Stores details about each product, including pricing and reorder information.

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100) NOT NULL,

Category VARCHAR(50),

UnitPrice DECIMAL(10, 2),

SupplierID INT,

ReorderLevel INT NOT NULL, -- Quantity level that triggers a reorder

FOREIGN KEY (SupplierID) REFERENCES Suppliers(SupplierID)

);

**Suppliers Table**: Contains information about suppliers that provide products.

CREATE TABLE Suppliers (

SupplierID INT PRIMARY KEY,

SupplierName VARCHAR(100) NOT NULL,

ContactInfo VARCHAR(100),

Address VARCHAR(200),

Phone VARCHAR(20)

);

**Sales Table**: Tracks each sale transaction, including the product sold, the date, and quantity.

CREATE TABLE Sales (

SaleID INT PRIMARY KEY,

ProductID INT,

SaleDate DATE NOT NULL,

QuantitySold INT NOT NULL,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

**Inventory Table**: Records the current inventory level for each product.

CREATE TABLE Inventory (

ProductID INT PRIMARY KEY,

QuantityInStock INT NOT NULL,

LastUpdated DATE,

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

2). To analyze historical sales data and predict future inventory needs, you can use SQL queries to calculate monthly or weekly sales trends, estimate future demand, and determine optimal reorder quantities. Here are some essential queries:

**Monthly Sales Trends for Each Product:**

SELECT

ProductID,

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(QuantitySold) AS TotalQuantitySold

FROM

Sales

GROUP BY

ProductID, SaleYear, SaleMonth

ORDER BY

ProductID, SaleYear, SaleMonth;

Estimate Future Inventory Needs:

SELECT

i.ProductID,

p.ProductName,

i.QuantityInStock,

COALESCE(ams.AverageMonthlySales, 0) AS ForecastedMonthlyDemand,

CASE

WHEN i.QuantityInStock < COALESCE(ams.AverageMonthlySales, 0)

THEN COALESCE(ams.AverageMonthlySales, 0) - i.QuantityInStock

ELSE 0

END AS RecommendedReorderQuantity

FROM

Inventory i

JOIN

Products p ON i.ProductID = p.ProductID

LEFT JOIN (

SELECT

ProductID,

AVG(TotalQuantitySold) AS AverageMonthlySales

FROM (

SELECT

ProductID,

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(QuantitySold) AS TotalQuantitySold

FROM

Sales

GROUP BY

ProductID, SaleYear, SaleMonth

) AS MonthlySales

GROUP BY

ProductID

) AS ams ON i.ProductID = ams.ProductID;

* These queries collectively offer a thorough analysis of historical sales and inventory needs. You can use them to forecast demand, identify products that require restocking, and manage inventory levels efficiently.

3). To implement a stored procedure that reorders stock based on sales trends and low inventory levels, the procedure should:

1. Identify products with inventory levels below the reorder level.
2. Calculate reorder quantities based on sales trends, current stock levels, and reorder levels.
3. Generate reorder records or update inventory to reflect pending reorders.

CREATE PROCEDURE ReorderStock()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE reorderQty INT;

DECLARE productID INT; CREATE PROCEDURE ReorderStock()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE reorderQty INT;

DECLARE productID INT;

DECLARE productName VARCHAR(100);

DECLARE quantityInStock INT;

DECLARE reorderLevel INT;

DECLARE avgMonthlySales INT;

-- Declare a cursor to iterate over products needing reorder

DECLARE reorder\_cursor CURSOR FOR

SELECT p.ProductID, p.ProductName, i.QuantityInStock, p.ReorderLevel,

COALESCE(ams.AverageMonthlySales, 0) AS ForecastedMonthlyDemand

FROM Products p

JOIN Inventory i ON p.ProductID = i.ProductID

LEFT JOIN (

SELECT

ProductID,

AVG(TotalQuantitySold) AS AverageMonthlySales

FROM (

SELECT

ProductID,

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(QuantitySold) AS TotalQuantitySold

FROM Sales

GROUP BY ProductID, SaleYear, SaleMonth

) AS MonthlySales

GROUP BY ProductID

) AS ams ON p.ProductID = ams.ProductID

WHERE i.QuantityInStock < p.ReorderLevel;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open cursor

OPEN reorder\_cursor;

-- Begin looping over products

read\_loop: LOOP

FETCH reorder\_cursor INTO productID, productName, quantityInStock, reorderLevel, avgMonthlySales;

IF done THEN

LEAVE read\_loop;

END IF;

-- Calculate reorder quantity (minimum of forecasted monthly demand or reorder level)

SET reorderQty = GREATEST(reorderLevel - quantityInStock, avgMonthlySales);

-- Insert reorder record into an example Reorders table

INSERT INTO Reorders (ProductID, ReorderDate, QuantityOrdered)

VALUES (productID, CURDATE(), reorderQty);

-- Log for confirmation (depends on specific implementation)

SELECT CONCAT("Reordered ", reorderQty, " units of ", productName, " (ProductID: ", productID, ")") AS ReorderLog;

END LOOP;

-- Close cursor

CLOSE reorder\_cursor;

END;

DECLARE productName VARCHAR(100);

DECLARE quantityInStock INT;

DECLARE reorderLevel INT;

DECLARE avgMonthlySales INT;

-- Declare a cursor to iterate over products needing reorder

DECLARE reorder\_cursor CURSOR FOR

SELECT p.ProductID, p.ProductName, i.QuantityInStock, p.ReorderLevel,

COALESCE(ams.AverageMonthlySales, 0) AS ForecastedMonthlyDemand

FROM Products p

JOIN Inventory i ON p.ProductID = i.ProductID

LEFT JOIN (

SELECT

ProductID,

AVG(TotalQuantitySold) AS AverageMonthlySales

FROM (

SELECT

ProductID,

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(QuantitySold) AS TotalQuantitySold

FROM Sales

GROUP BY ProductID, SaleYear, SaleMonth

) AS MonthlySales

GROUP BY ProductID

) AS ams ON p.ProductID = ams.ProductID

WHERE i.QuantityInStock < p.ReorderLevel;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

-- Open cursor

OPEN reorder\_cursor;

-- Begin looping over products

read\_loop: LOOP

FETCH reorder\_cursor INTO productID, productName, quantityInStock, reorderLevel, avgMonthlySales;

IF done THEN

LEAVE read\_loop;

END IF;

-- Calculate reorder quantity (minimum of forecasted monthly demand or reorder level)

SET reorderQty = GREATEST(reorderLevel - quantityInStock, avgMonthlySales);

-- Insert reorder record into an example Reorders table

INSERT INTO Reorders (ProductID, ReorderDate, QuantityOrdered)

VALUES (productID, CURDATE(), reorderQty);

-- Log for confirmation (depends on specific implementation)

SELECT CONCAT("Reordered ", reorderQty, " units of ", productName, " (ProductID: ", productID, ")") AS ReorderLog;

END LOOP;

-- Close cursor

CLOSE reorder\_cursor;

END;

4). To create views for management that show forecasted sales by product category, you can structure SQL views to display:

1. **Average monthly sales by category**, based on historical sales data.
2. **Forecasted sales for each product in a category** based on calculated monthly averages.

CREATE VIEW CategorySalesForecast AS

SELECT

p.Category,

YEAR(s.SaleDate) AS SaleYear,

MONTH(s.SaleDate) AS SaleMonth,

SUM(s.QuantitySold) AS TotalQuantitySold,

AVG(s.QuantitySold) AS AvgMonthlySales

FROM

Sales s

JOIN

Products p ON s.ProductID = p.ProductID

GROUP BY

p.Category, SaleYear, SaleMonth

ORDER BY

p.Category, SaleYear, SaleMonth;

CREATE VIEW ProductForecastByCategory AS

SELECT

p.Category,

p.ProductName,

p.ProductID,

AVG(monthly\_sales.TotalQuantitySold) AS ForecastedMonthlySales

FROM

Products p

LEFT JOIN (

-- Calculate total quantity sold per product per month

SELECT

ProductID,

YEAR(SaleDate) AS SaleYear,

MONTH(SaleDate) AS SaleMonth,

SUM(QuantitySold) AS TotalQuantitySold

FROM

Sales

GROUP BY

ProductID, SaleYear, SaleMonth

) AS monthly\_sales ON p.ProductID = monthly\_sales.ProductID

GROUP BY

p.Category, p.ProductName, p.ProductID

ORDER BY

p.Category, p.ProductName;

 **CategorySalesForecast**: Aggregates monthly sales data by category, providing a high-level forecast for management.

 **ProductForecastByCategory**: Displays forecasted sales for each product within each category, averaging monthly sales over historical data to project future demand.

**Conclusion:-**

In summary, the **Product Inventory and Sales Forecasting System** provides a robust framework for managing inventory and predicting future demand in a retail environment. By designing structured tables for **Products**, **Suppliers**, **Sales**, and **Inventory**, the system organizes essential data, laying a foundation for comprehensive inventory management. SQL queries offer tools to analyse historical sales trends and calculate average monthly demand, helping forecast future inventory needs accurately.

The stored procedure for **reordering stock** automates restocking by calculating reorder quantities based on sales trends and current inventory levels, ensuring timely replenishment to prevent stockouts. Additionally, the **Category Sales Forecast** and **Product Forecast By Category** views provide management with clear insights into forecasted sales by product category and individual products. These views empower decision-makers to track demand trends, make informed purchasing decisions, and optimize inventory levels.

Overall, this system allows for data-driven management of inventory and resources, which can lead to improved stock availability, minimized excess inventory, and more efficient operations in retail management.