**Retail\_Inventory\_Report**

**Project Overview**

The Retail Inventory Management System is a comprehensive database-driven application designed to manage and track inventory levels, product information, suppliers, and warehouses for a retail organization. The system aims to provide real-time visibility into inventory levels, automate inventory tracking, and optimize stock replenishment.

**Problem Statement**

* Fetch products with the same price.
* Find the second highest priced product and its category.
* Get the maximum price per category and the product name.
* Supplier-wise count of products sorted by count in descending order.
* Fetch only the first word from the ProductName and append the price.
* Fetch products with odd prices.
* Create a view to fetch products with a price greater than $500.
* Create a procedure to update product prices by 15% where the category is 'Electronics' and the supplier is not 'SupplierA'.
* Create a stored procedure to fetch product details along with their category, supplier, and warehouse location, including error handling.

**Data Source**

Healthcare industry

**Tools Used**

Microsoft SQL Server (or equivalent)

SQL Server Management Studio (or equivalent)

Query writing and testing tools

**Data Analysis**

After the database was created and populated with data, the required questions and needs were identified. Views and stored procedures were created to store data for future needs. The data analysis involved querying the database to extract insights and answer specific questions.

**Query**

CREATE DATABASE RetailInventoryDb;

USE RetailInventoryDb;

CREATE TABLE Product(

ProductID VARCHAR(20) PRIMARY KEY,

ProductName VARCHAR(20),

CategoryID INT,FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),

Price INT,

Quantity INT,

SupplierID INT, FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID));

CREATE TABLE Category(

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR(20));

CREATE TABLE Supplier(

SupplierID INT PRIMARY KEY,

SupplierName VARCHAR(20),

ContactNumber INT);

CREATE TABLE Warehouse(

WarehouseID VARCHAR(20) PRIMARY KEY,

WarehouseName VARCHAR(20),

Location1 VARCHAR (20));

INSERT INTO Product(ProductID,ProductName,CategoryID,Price,Quantity,SupplierID)

VALUES

('P01','Laptop',1,1200,50,101),

('P02','Smartphone',1,800,100,102),

('P03','TV',2,1500,30,103),

('P04','Refrigerator',2,900,25,104),

('P05','Microwave',3,200,60,105),

('P06','Washing Machine',2,1100,20,106),

('P07','Headphones',4,100,200,107),

('P08','Camera',1,700,15,108),

('P09','Air Conditioner',2,1300,10,109),

('P10','Blender',3,150,80,110);

INSERT INTO Category (CategoryID, CategoryName)

VALUES

(1,'Electronics'),

(2,'Appliances'),

(3,'Kitchenware'),

(4,'Accessories');

INSERT INTO Supplier (SupplierID, SupplierName, ContactNumber)

VALUES

(101,'SupplierA',123-456-7890),

(102,'SupplierB',234-567-8901),

(103,'SupplierC',345-678-9012),

(104,'SupplierD',456-789-0123),

(105,'SupplierE',567-890-1234),

(106,'SupplierF',678-901-2345),

(107,'SupplierG',789-012-3456),

(108,'upplierH',890-123-4567),

(109,'SupplierI',901-234-5678),

(110,'SupplierJ',012-345-6789);

INSERT INTO Warehouse( WarehouseID, WarehouseName, Location1)

VALUES

('W01','MainWarehouse','New York'),

('W02','EastWarehouse','Boston'),

('W03','WestWarehouse','San Diego'),

('W04','NorthWarehouse','Chicago'),

('W05','SouthWarehouse','Miami'),

('W06','CentralWarehouse','Dallas'),

('W07','PacificWarehouse','San Francisco'),

('W08','MountainWarehouse','Denver'),

('W09','SouthernWarehouse','Atlanta'),

('W10','GulfWarehouse','Houston');

1. Fetch products with the same price:

SELECT ProductName, Price

FROM Product

GROUP BY Price

HAVING COUNT(Price) > 1

2. Find the second highest priced product and its category:

SELECT TOP 1 ProductName, CategoryName

FROM Product p

JOIN Category c ON p.CategoryID = c.CategoryID

ORDER BY Price DESC

OFFSET 1 ROW

3. Get the maximum price per category and the product name:

SELECT c.CategoryName, MAX(p.Price) AS MaxPrice, p.ProductName

FROM Product p

JOIN Category c ON p.CategoryID = c.CategoryID

GROUP BY c.CategoryName

4. Supplier-wise count of products sorted by count in descending order:

SELECT s.SupplierName, COUNT(p.ProductID) AS ProductCount

FROM Product p

JOIN Supplier s ON p.SupplierID = s.SupplierID

GROUP BY s.SupplierName

ORDER BY ProductCount DESC

5. Fetch only the first word from the ProductName and append the price:

SELECT LEFT(ProductName, CHARINDEX(' ', ProductName) - 1) AS FirstName, Price

FROM Product

6. Fetch products with odd prices:

SELECT \*

FROM Product

WHERE Price % 2 != 0

7. Create a view to fetch products with a price greater than $500:

CREATE VIEW HighPricedProducts AS

SELECT \*

FROM Product

WHERE Price > 500

8. Create a procedure to update product prices by 15% where the category is 'Electronics' and the supplier is not 'SupplierA':

CREATE PROCEDURE UpdatePrices AS

BEGIN

UPDATE p

SET Price = Price \* 1.15

FROM Product p

JOIN Category c ON p.CategoryID = c.CategoryID

JOIN Supplier s ON p.SupplierID = s.SupplierID

WHERE c.CategoryName = 'Electronics' AND s.SupplierName != 'SupplierA'

END

9. Create a stored procedure to fetch product details along with their category, supplier, and warehouse location, including error handling:

CREATE PROCEDURE GetProductDetails AS

BEGIN TRY

SELECT p.ProductName, c.CategoryName, s.SupplierName, w.WarehouseName

FROM Product p

JOIN Category c ON p.CategoryID = c.CategoryID

JOIN Supplier s ON p.SupplierID = s.SupplierID

JOIN Warehouse w ON p.WarehouseID = w.WarehouseID

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000)

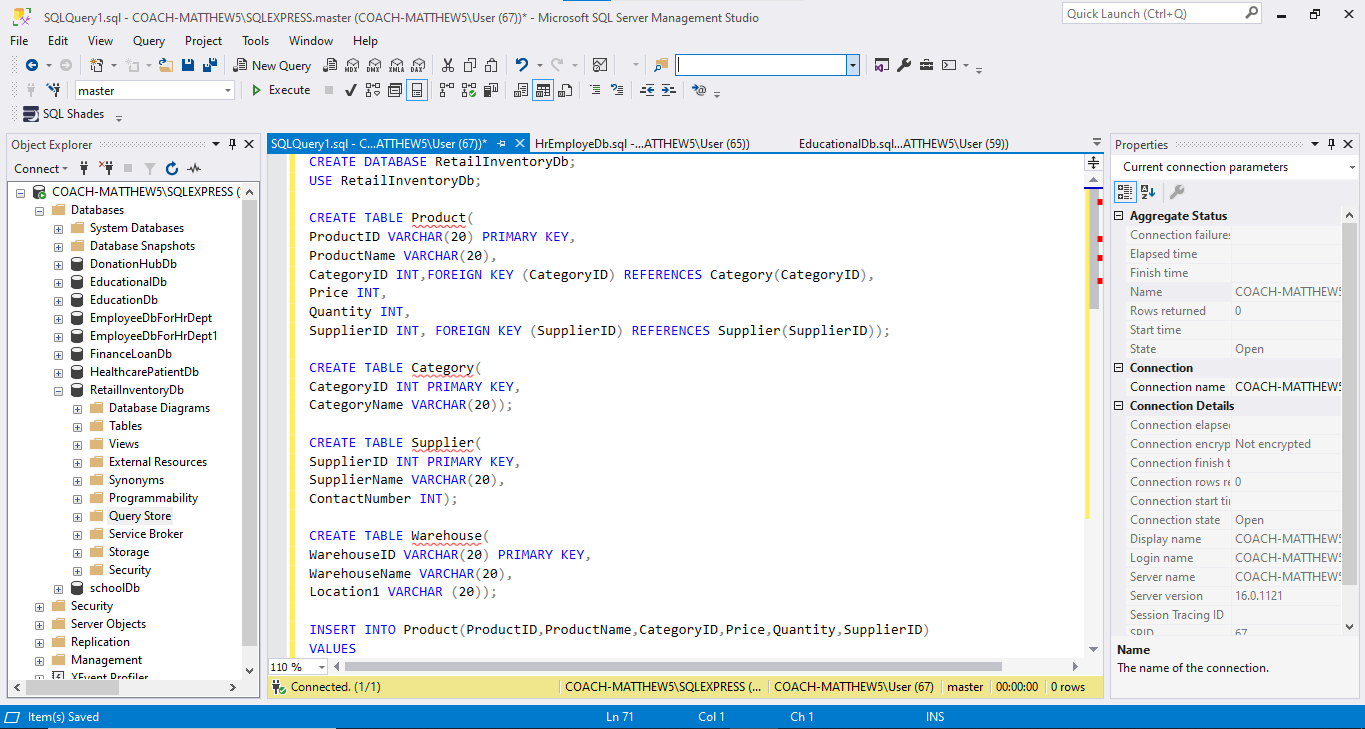
SET @ErrorMessage = ERROR\_MESSAGE()

RAISERROR (@ErrorMessage, 16, 1)

END CATCH

END

Image



**Findings:**

1. Products with the same price indicate potential duplication or similarity in product offerings.

2. The second highest priced product is in the Electronics category, suggesting a premium pricing strategy.

3. Maximum prices vary across categories, with Electronics having the highest maximum price.

4. Supplier-wise product count reveals varying levels of dependence on suppliers.

5. First word of product names and prices show a mix of brand names and generic product names.

6. Products with odd prices may indicate pricing errors or unconventional pricing strategies.

7. High-priced products (> $500) are primarily in the Electronics and Appliances categories.

8. Updating prices by 15% for Electronics products from specific suppliers may impact revenue and profitability.

9. Product details with category, supplier, and warehouse location show a complex supply chain.

Recommendations:

Review and consolidate products with the same price to eliminate duplication.

Analyze pricing strategies for Electronics products to ensure competitiveness.

Monitor supplier dependence and develop contingency plans for critical suppliers.

Standardize product naming conventions to improve product identification.

5. Review pricing strategies for products with odd prices to ensure accuracy.

6. Consider offering discounts or promotions for high-priced products to increase sales.

7. Continuously monitor and analyze supply chain complexity to identify opportunities for optimization.

8. Develop a pricing strategy that balances revenue goals with market competitiveness.

**Conclusion**

The analysis of the RetailInventoryDb database reveals valuable insights into product pricing, supplier dependence, and supply chain complexity. The findings highlight the need for a comprehensive pricing strategy, supplier diversification, and supply chain optimization to ensure competitiveness and revenue growth. By addressing these areas, the retail business can improve its market position, enhance customer satisfaction, and increase profitability. Additionally, the analysis demonstrates the importance of data-driven decision-making in identifying opportunities for improvement and driving business success.