LinkedIn of Abhay Badyal

Database Design and Implementation

WEBTECH: E-commerce Case Study

Table of Content

Introduction

WEBTECH is a growing e-commerce store that specializes in selling affordable, quirky, and practical tech gadgets. Designed for tech enthusiasts and everyday users alike, the platform aims to simplify digital living by offering unique accessories. This case study outlines the complete database design and implementation strategy used to support core business operations such as customer transactions, order processing, inventory management, and sales tracking also presents the database design and implementation, covering table structures, relationships, complex queries, and views.

Mission and Objectives

Mission

To deliver unique, fun, and practical tech gadgets that enhance the digital lifestyle — combining affordability, quality, and convenience.

Objectives

- Provide smooth user experience and secure online checkout.
- Maintain real-time inventory to avoid stockouts.
- Analyze sales data to improve customer service and marketing.
- Ensure fast delivery and transparent order tracking.
- Offer high-quality gadgets and accessories that stand out in the market.

Database Design

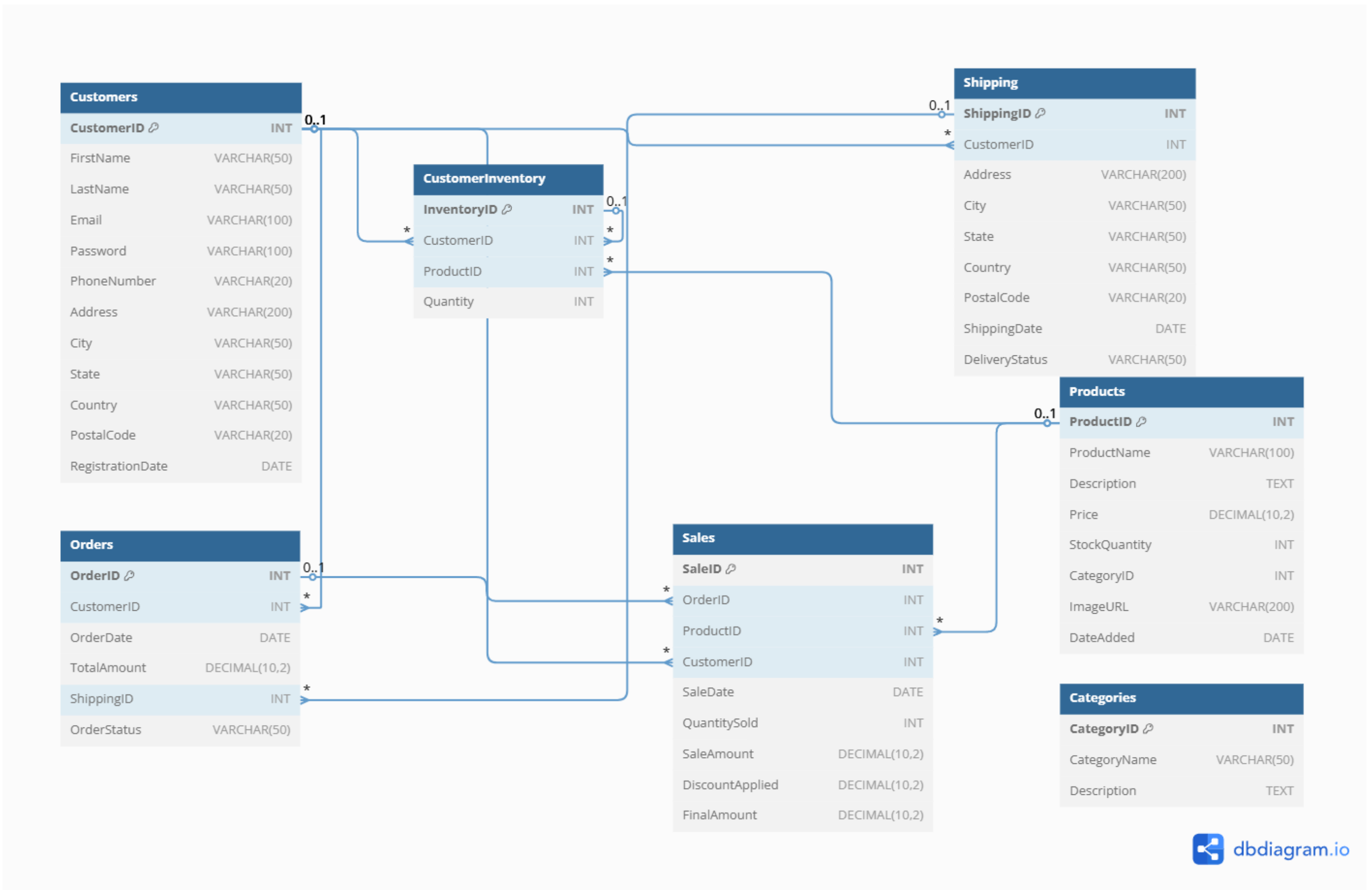
Entities

Entities used in this database are as follows:

- **Customers** – Stores user data including contact and shipping information.
- **Products** – Information about each gadget such as name, category, price, and stock level.
- **Orders** – Records each purchase made by a customer.
- **Sales (Fact Table)** – Central table for analysis of transactions and revenue.
- **Inventory** – Tracks stock availability and warehouse data.
- **Shipping** – Stores details about shipping methods, delivery status, and courier info.
- **Categories** – Defines product categories like “Mobile Accessories,” “Wearables,” etc.

Entity-Relationship Diagram (ERD)

The **ER Diagram** represents the key entities of the system and their relationships.



Relationships

1. Customers and Orders (1:M)

- **Relationship**: One customer can place multiple orders, but each order belongs to only one customer.
- **Reason**: A customer may return to the website to make purchases on multiple occasions**.**

2. Orders and Sales (1:M)

- **Relationship**: One order can include multiple sales records (one per product), but each sales record belongs to only one order.
- **Reason**: Popular products may be purchased by many different customers in separate orders.

3. Products and Sales (1:M)

- **Relationship:** One product can appear in multiple sales records, but each sales record is linked to a single product.
- **Reason:** Faculty members may be assigned to multiple students in various courses.

4. Products and Inventory (1:1)

- **Relationship:** Each product has one corresponding inventory record, and each inventory record refers to only one product.
- **Reason:** Inventory is tracked uniquely for every product to monitor stock availability and restocking needs.

5. Products and Categories (M:1)

- **Relationship:** Multiple products can belong to the same category, but each product is assigned to only one category.
- **Reason:** Products are grouped into categories (e.g., Accessories, Smart Gadgets) for browsing and classification.

6. Orders and Shipping (1:1)

- **Relationship:** Each order has one corresponding shipping record, and each shipping record belongs to one order.
- **Reason:** Shipping details like method, status, and delivery date are tracked per order to ensure accurate delivery.

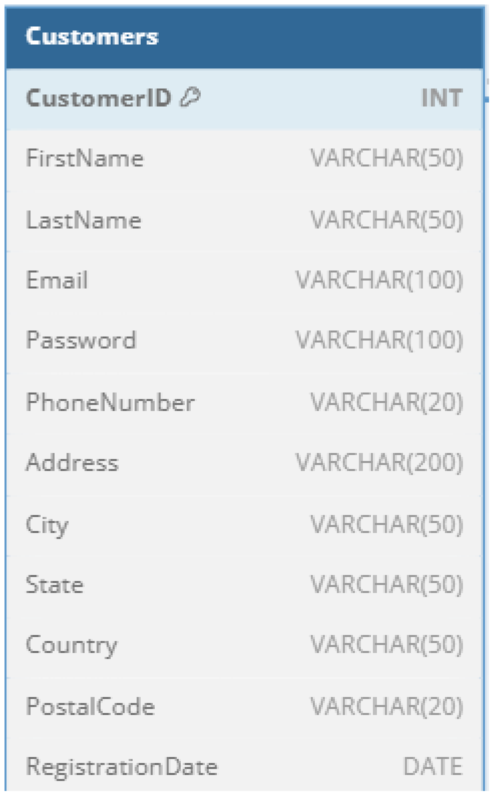
Data Dictionary

The **relational model** of the College Enrollment System consists of multiple tables, each designed to store structured information and maintain referential integrity. Below is a description of the **key tables** and their relationships:

1. Customers Table

Stores customer details, including contact and address information for order and delivery processing.

- **Primary Key:** CustomerId
- **Not null Constraints:** FirstName, LastName, email, phone, address
- **Unique Constraint:** email

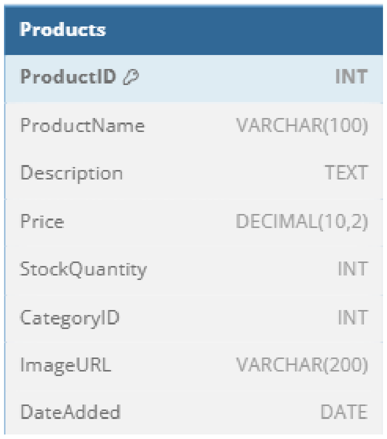


Customers	
CustomerID	INT
FirstName	VARCHAR(50)
LastName	VARCHAR(50)
Email	VARCHAR(100)
Password	VARCHAR(100)
PhoneNumber	VARCHAR(20)
Address	VARCHAR(200)
City	VARCHAR(50)
State	VARCHAR(50)
Country	VARCHAR(50)
PostalCode	VARCHAR(20)
RegistrationDate	DATE

2. Products Table

Stores product-related details, including name, price, stock, and category linkage.

- **Primary Key**: ProductId
- **Foreign Key**: CategoryId references Categories table CategoryId
- **Not null Constraints**: product_name, price, CategoryId
- **Unique Constraint**: product_name

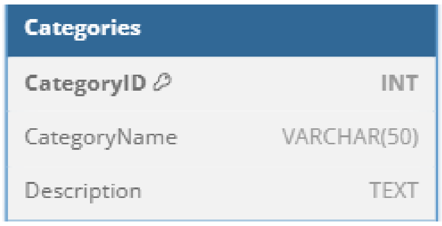


Products	
ProductID	INT
ProductName	VARCHAR(100)
Description	TEXT
Price	DECIMAL(10,2)
StockQuantity	INT
CategoryID	INT
ImageURL	VARCHAR(200)
DateAdded	DATE

3. Categories Table

Defines product classifications (e.g., Mobile Accessories, Smart Devices).

- **Primary Key**: CategoryId
- **Not null Constraint**: CategoryName
- **Unique Constraint**: CategoryName

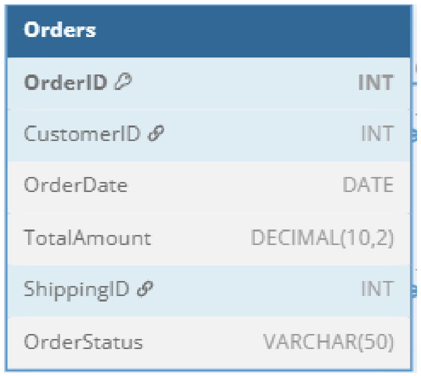


Categories	
CategoryID	INT
CategoryName	VARCHAR(50)
Description	TEXT

4. Orders Table

Stores details of orders placed by customers including date and amount.

- **Primary Key:** OrderId
- **Foreign Key:** CustomerId references Customers table CustomerId
- **Not null Constraints:** CustomerId, order_date, total_amount

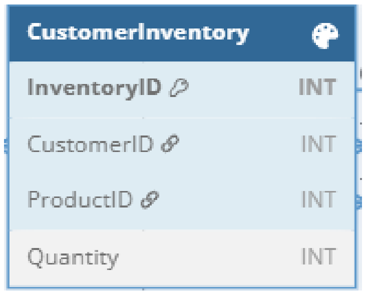


Orders	
OrderID	INT
CustomerID	INT
OrderDate	DATE
TotalAmount	DECIMAL(10,2)
ShippingID	INT
OrderStatus	VARCHAR(50)

5. **Inventory Table**

Tracks stock level and restock thresholds for each product.

Primary Key: InventoryId
Foreign Key: ProductId references Products table ProductId
Not null Constraints: ProductId, stock_level, reorder_level
Unique Constraint: ProductId (one-to-one relationship)

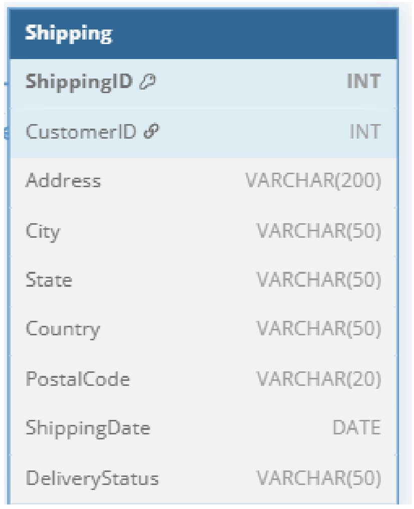


CustomerInventory	
InventoryID	INT
CustomerID	INT
ProductID	INT
Quantity	INT

6. **Shipping Table**

Captures shipment data for each order, including delivery status and method.

- **Primary Key:** ShippingId
- **Foreign Key:** OrderId references Orders table OrderId
- **Not null Constraints:** OrderId, shipping_method, status
- **Unique Constraint:** OrderId (one-to-one relationship)

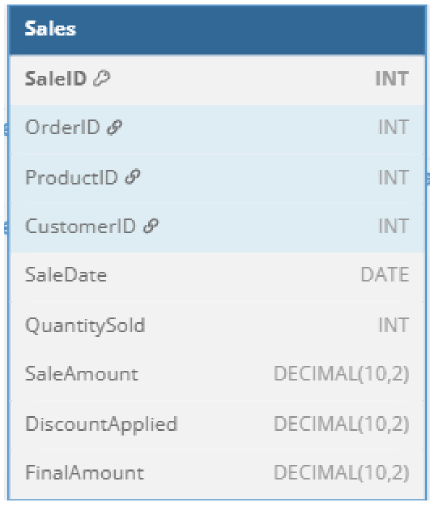


Shipping	
ShippingID	INT
CustomerID	INT
Address	VARCHAR(200)
City	VARCHAR(50)
State	VARCHAR(50)
Country	VARCHAR(50)
PostalCode	VARCHAR(20)
ShippingDate	DATE
DeliveryStatus	VARCHAR(50)

7. **Sales Table**

Central fact table capturing product sales linked to orders.

- **Primary Key**: SaleId
- **Foreign Keys**: ProductId references Products table ProductId,
  OrderId references Orders table OrderId
- **Not null Constraints**: ProductId, OrderId, quantity_sold, total_amount

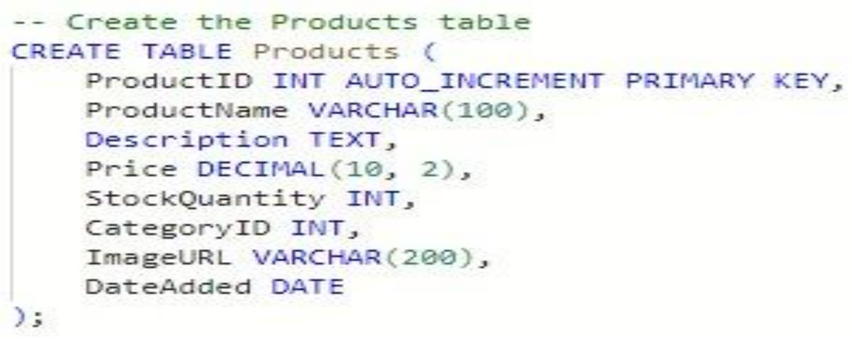


Sales	
SaleID	INT
OrderID	INT
ProductID	INT
CustomerID	INT
SaleDate	DATE
QuantitySold	INT
SaleAmount	DECIMAL(10,2)
DiscountApplied	DECIMAL(10,2)
FinalAmount	DECIMAL(10,2)

Queries to create Tables

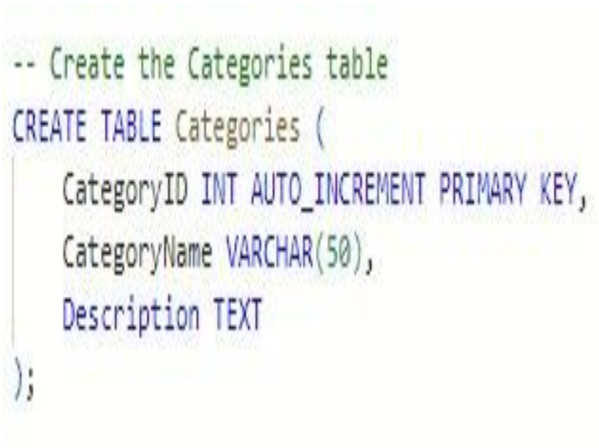
Queries to create tables are as follows :

1. **Products table**:



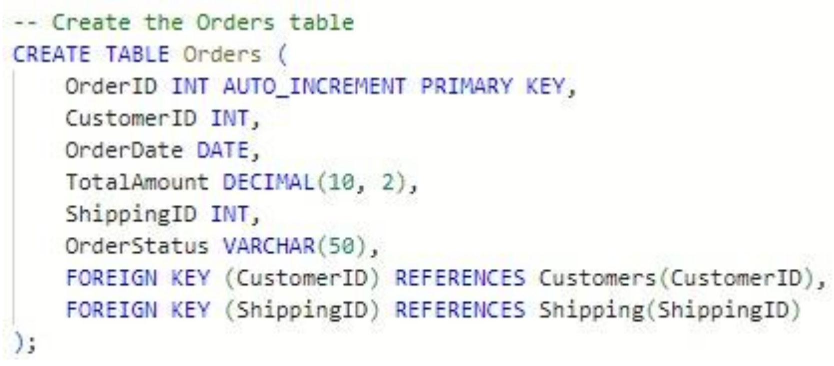
```
-- Create the Products table
CREATE TABLE Products (
    ProductID INT AUTO_INCREMENT PRIMARY KEY,
    ProductName VARCHAR(100),
    Description TEXT,
    Price DECIMAL(10, 2),
    StockQuantity INT,
    CategoryID INT,
    ImageURL VARCHAR(200),
    DateAdded DATE
);
```

2. **Categories table**:



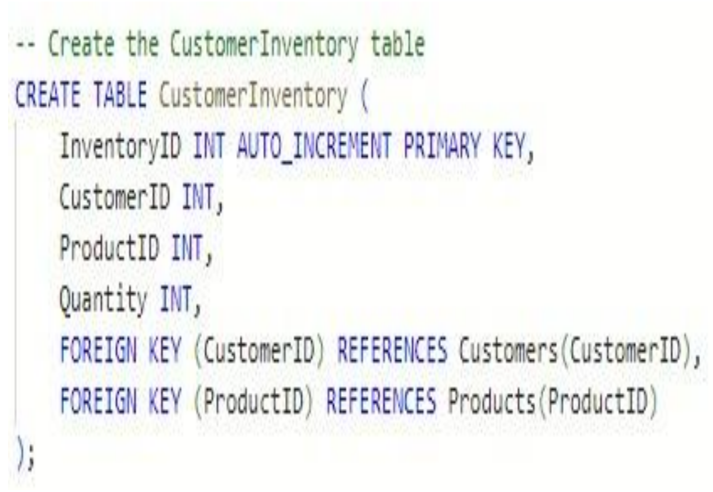
```
-- Create the Categories table
CREATE TABLE Categories (
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,
    CategoryName VARCHAR(50),
    Description TEXT
);
```

3. **Orders table**:



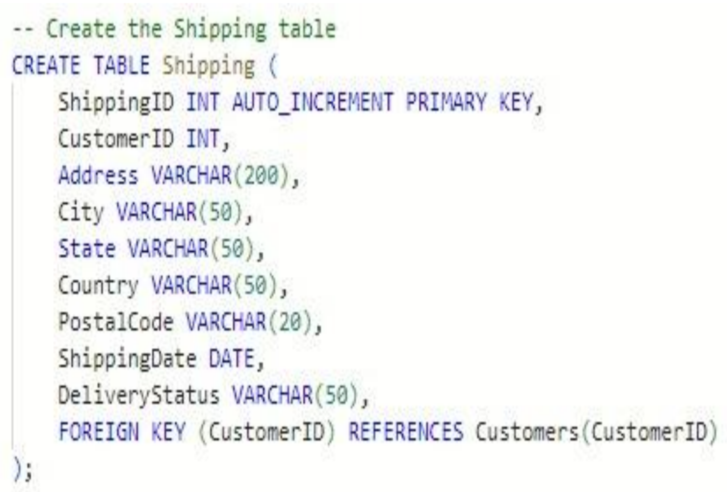
```
-- Create the Orders table
CREATE TABLE Orders (
    OrderID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10, 2),
    ShippingID INT,
    OrderStatus VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (ShippingID) REFERENCES Shipping(ShippingID)
);
```

4. **CustomerInventory table :**



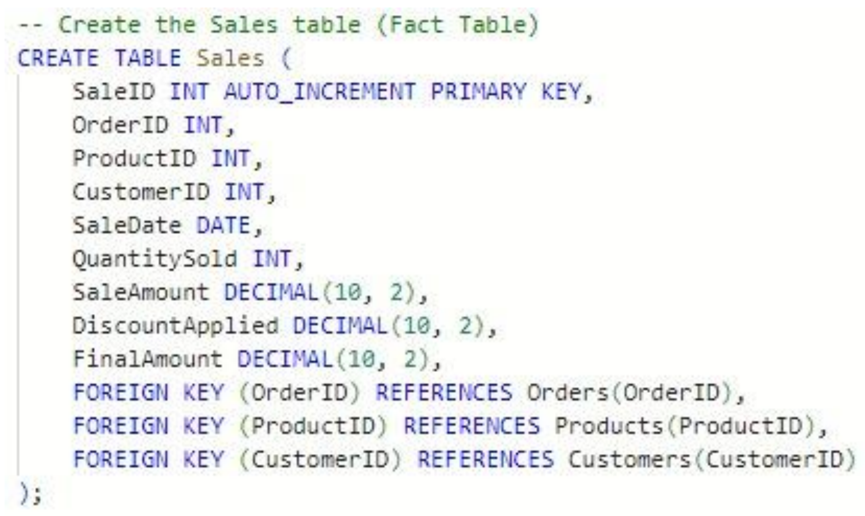
```
-- Create the CustomerInventory table
CREATE TABLE CustomerInventory (
    InventoryID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    Quantity INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

5. **Shipping table :**



```
-- Create the Shipping table
CREATE TABLE Shipping (
    ShippingID INT AUTO_INCREMENT PRIMARY KEY,
    CustomerID INT,
    Address VARCHAR(200),
    City VARCHAR(50),
    State VARCHAR(50),
    Country VARCHAR(50),
    PostalCode VARCHAR(20),
    ShippingDate DATE,
    DeliveryStatus VARCHAR(50),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

6. Sales table :



```
-- Create the Sales table (Fact Table)
CREATE TABLE Sales (
    SaleID INT AUTO_INCREMENT PRIMARY KEY,
    OrderID INT,
    ProductID INT,
    CustomerID INT,
    SaleDate DATE,
    QuantitySold INT,
    SaleAmount DECIMAL(10, 2),
    DiscountApplied DECIMAL(10, 2),
    FinalAmount DECIMAL(10, 2),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Views

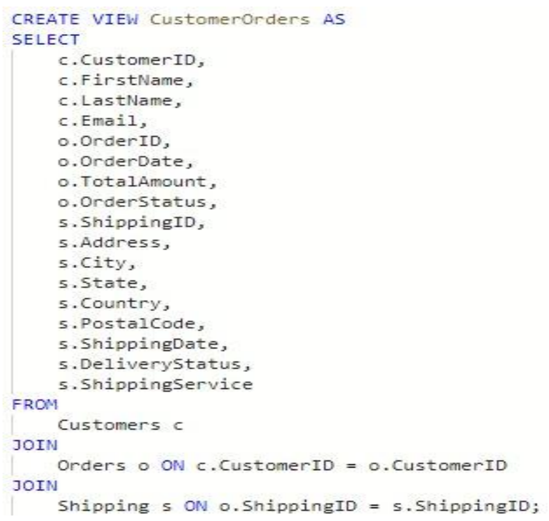
1. CustomerOrders VIEW

Purpose:
To display a summary of customer orders including customer names, order IDs, dates, and total amounts. This helps in tracking who placed what order and when.

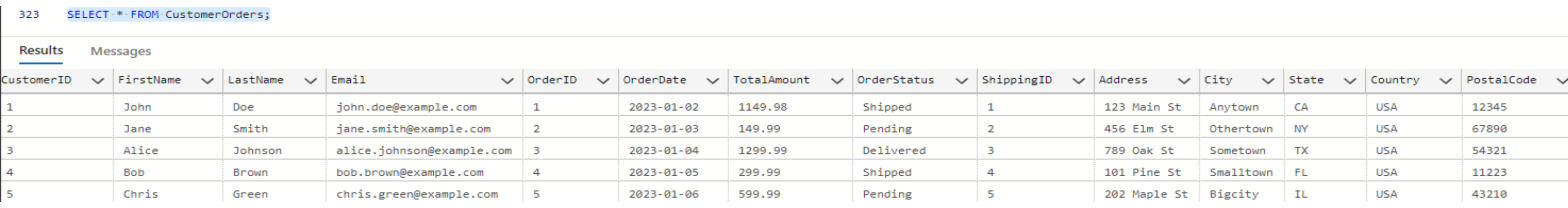
Tables Used:

- Customers
- Orders

Explanation:
This view joins the Orders and Customers tables to display user-friendly order history. It concatenates the first and last names to generate full customer names and includes basic order information for reporting or customer service use



```
CREATE VIEW CustomerOrders AS
SELECT
    c.CustomerID,
    c.FirstName,
    c.LastName,
    c.Email,
    o.OrderID,
    o.OrderDate,
    o.TotalAmount,
    o.OrderStatus,
    s.ShippingID,
    s.Address,
    s.City,
    s.State,
    s.Country,
    s.PostalCode,
    s.ShippingDate,
    s.DeliveryStatus,
    s.ShippingService
FROM
    Customers c
JOIN
    Orders o ON c.CustomerID = o.CustomerID
JOIN
    Shipping s ON o.ShippingID = s.ShippingID;
```



```sql
SELECT * FROM CustomerOrders;
```

Results Messages

CustomerID	FirstName	LastName	Email	OrderID	OrderDate	TotalAmount	OrderStatus	ShippingID	Address	City	State	Country	PostalCode
1	John	Doe	john.doe@example.com	1	2023-01-02	1149.98	Shipped	1	123 Main St	Anytown	CA	USA	12345
2	Jane	Smith	jane.smith@example.com	2	2023-01-03	149.99	Pending	2	456 Elm St	Othertown	NY	USA	67890
3	Alice	Johnson	alice.johnson@example.com	3	2023-01-04	1299.99	Delivered	3	789 Oak St	Sometown	TX	USA	54321
4	Bob	Brown	bob.brown@example.com	4	2023-01-05	299.99	Shipped	4	101 Pine St	Smalltown	FL	USA	11223
5	Chris	Green	chris.green@example.com	5	2023-01-06	599.99	Pending	5	202 Maple St	Bigcity	IL	USA	43210

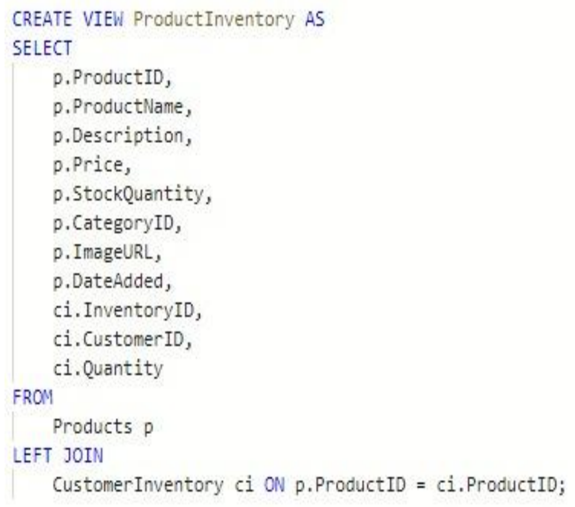
2. ProductInventory VIEW

Purpose:
To show the current stock status of each product along with its category, making inventory reviews easier for restocking and sales readiness.

Tables Used:

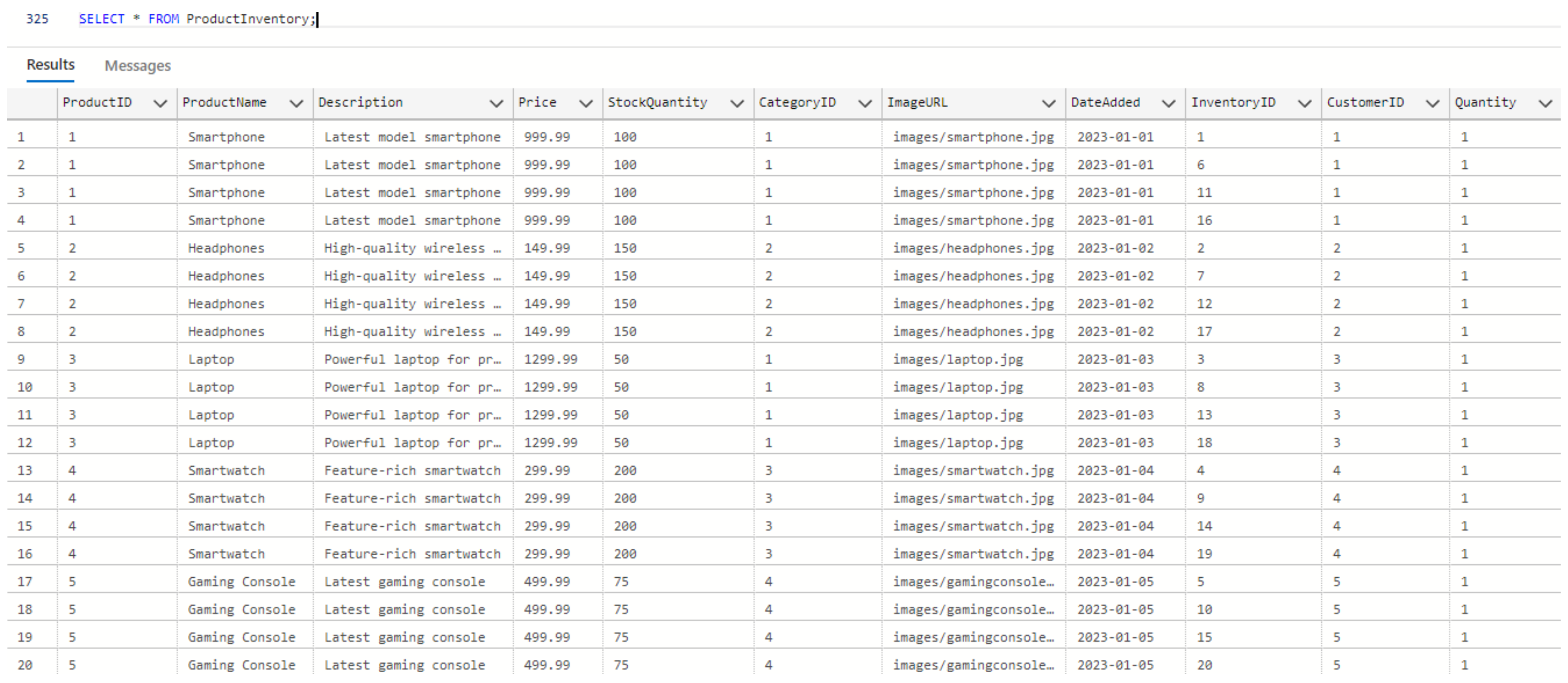
- Products
- Inventory
- Categories

Explanation:
This view brings together product information, current stock quantity, and the category it belongs to. It supports inventory audits, stock-level monitoring, and helps identify which categories may need replenishment.



```sql
CREATE VIEW ProductInventory AS
SELECT
    p.ProductID,
    p.ProductName,
    p.Description,
    p.Price,
    p.StockQuantity,
    p.CategoryID,
    p.ImageURL,
    p.DateAdded,
    ci.InventoryID,
    ci.CustomerID,
    ci.Quantity
FROM
    Products p
LEFT JOIN
    CustomerInventory ci ON p.ProductID = ci.ProductID;
```

LinkedIn of Abhay Badyal



```
SELECT * FROM ProductInventory;
```

	ProductID	ProductName	Description	Price	StockQuantity	CategoryID	ImageURL	DateAdded	InventoryID	CustomerID	Quantity
1	1	Smartphone	Latest model smartphone	999.99	100	1	images/smartphone.jpg	2023-01-01	1	1	1
2	1	Smartphone	Latest model smartphone	999.99	100	1	images/smartphone.jpg	2023-01-01	6	1	1
3	1	Smartphone	Latest model smartphone	999.99	100	1	images/smartphone.jpg	2023-01-01	11	1	1
4	1	Smartphone	Latest model smartphone	999.99	100	1	images/smartphone.jpg	2023-01-01	16	1	1
5	2	Headphones	High-quality wireless …	149.99	150	2	images/headphones.jpg	2023-01-02	2	2	1
6	2	Headphones	High-quality wireless …	149.99	150	2	images/headphones.jpg	2023-01-02	7	2	1
7	2	Headphones	High-quality wireless …	149.99	150	2	images/headphones.jpg	2023-01-02	12	2	1
8	2	Headphones	High-quality wireless …	149.99	150	2	images/headphones.jpg	2023-01-02	17	2	1
9	3	Laptop	Powerful laptop for pr…	1299.99	50	1	images/laptop.jpg	2023-01-03	3	3	1
10	3	Laptop	Powerful laptop for pr…	1299.99	50	1	images/laptop.jpg	2023-01-03	8	3	1
11	3	Laptop	Powerful laptop for pr…	1299.99	50	1	images/laptop.jpg	2023-01-03	13	3	1
12	3	Laptop	Powerful laptop for pr…	1299.99	50	1	images/laptop.jpg	2023-01-03	18	3	1
13	4	Smartwatch	Feature-rich smartwatch	299.99	200	3	images/smartwatch.jpg	2023-01-04	4	4	1
14	4	Smartwatch	Feature-rich smartwatch	299.99	200	3	images/smartwatch.jpg	2023-01-04	9	4	1
15	4	Smartwatch	Feature-rich smartwatch	299.99	200	3	images/smartwatch.jpg	2023-01-04	14	4	1
16	4	Smartwatch	Feature-rich smartwatch	299.99	200	3	images/smartwatch.jpg	2023-01-04	19	4	1
17	5	Gaming Console	Latest gaming console	499.99	75	4	images/gamingconsole…	2023-01-05	5	5	1
18	5	Gaming Console	Latest gaming console	499.99	75	4	images/gamingconsole…	2023-01-05	10	5	1
19	5	Gaming Console	Latest gaming console	499.99	75	4	images/gamingconsole…	2023-01-05	15	5	1
20	5	Gaming Console	Latest gaming console	499.99	75	4	images/gamingconsole…	2023-01-05	20	5	1

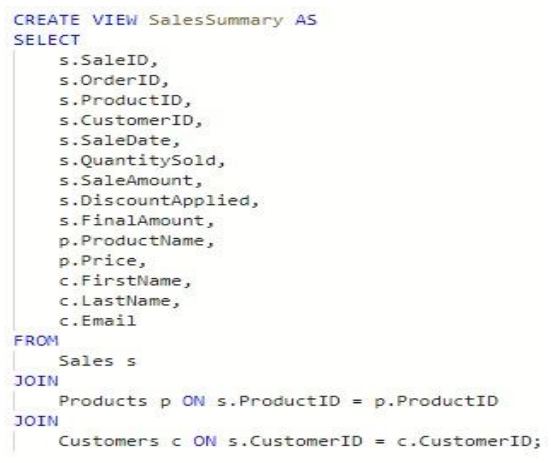
3. SalesSummary View

Purpose:
To provide an aggregated summary of total revenue generated per product, useful for analyzing top-selling items and making strategic decisions.

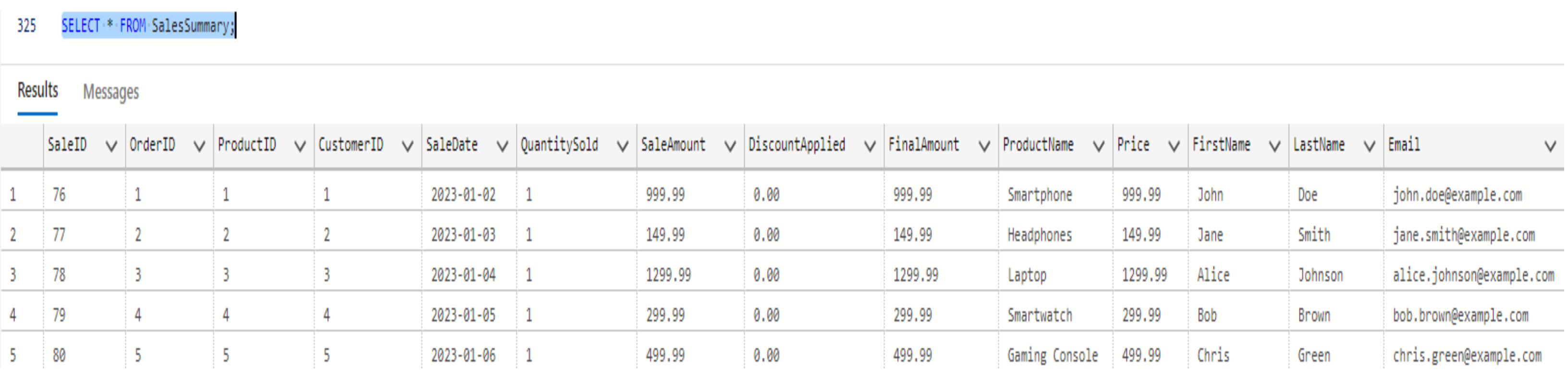
Tables Used:

- Sales
- Products

Explanation:
This view calculates the total revenue per product using SUM() grouped by product_name. It simplifies revenue analysis and can support dashboards or periodic performance reports.



```
CREATE VIEW SalesSummary AS
SELECT
    s.SaleID,
    s.OrderID,
    s.ProductID,
    s.CustomerID,
    s.SaleDate,
    s.QuantitySold,
    s.SaleAmount,
    s.DiscountApplied,
    s.FinalAmount,
    p.ProductName,
    p.Price,
    c.FirstName,
    c.LastName,
    c.Email
FROM
    Sales s
JOIN
    Products p ON s.ProductID = p.ProductID
JOIN
    Customers c ON s.CustomerID = c.CustomerID;
```



```sql
SELECT * FROM SalesSummary;
```

Results Messages

	SaleID	OrderID	ProductID	CustomerID	SaleDate	QuantitySold	SaleAmount	DiscountApplied	FinalAmount	ProductName	Price	FirstName	LastName	Email
1	76	1	1	1	2023-01-02	1	999.99	0.00	999.99	Smartphone	999.99	John	Doe	john.doe@example.com
2	77	2	2	2	2023-01-03	1	149.99	0.00	149.99	Headphones	149.99	Jane	Smith	jane.smith@example.com
3	78	3	3	3	2023-01-04	1	1299.99	0.00	1299.99	Laptop	1299.99	Alice	Johnson	alice.johnson@example.com
4	79	4	4	4	2023-01-05	1	299.99	0.00	299.99	Smartwatch	299.99	Bob	Brown	bob.brown@example.com
5	80	5	5	5	2023-01-06	1	499.99	0.00	499.99	Gaming Console	499.99	Chris	Green	chris.green@example.com

Conclusion

The WEBTECH e-commerce database is a well-structured, scalable solution built on core relational principles. It ensures robust management of customer data, product listings, transactions, and inventory while supporting detailed sales reporting and insights. With efficient queries and reusable views, the system enhances operational visibility and data-driven decision-making — all while ensuring a smooth shopping experience for users.