SHOP VENDOR-CUSTOMER BASED SYSTEM

Submitted By: Group F

**Submitted To:
Dr. Saima Noreen Khosa**

Contents

Overview

The **Generic Shop Vendor-Customer System** is an integrated solution that streamlines the interactions between vendors (or sellers) and customers in an online shopping environment. The system supports multiple vendors, allowing them to manage their stores and products independently. Customers can browse these products, place orders, and make payments securely.

This system supports various features:

- Vendor management for handling products and store profiles.
- Product categorization to maintain organized inventory.
- Customer management for a personalized shopping experience.
- Secure ordering and payment processing.

In the following sections, we will provide detailed descriptions of the entities, their attributes, and how they interact.

Key Modules

Customer Management

The **Customer Management** module handles customer registration, login, profile management, and order history. **Customers can:**

- Sign up, log in, and manage their personal information.
- Browse products and place orders.
- View and track their order history.
- Manage shipping addresses and payment methods.

Vendor Management

The **Vendor Management** module allows sellers to:

- Create an account and set up their stores.
- List products under appropriate categories.
- Manage stock levels, prices, and discounts.
- View sales statistics and order histories.
- Respond to customer inquiries and process returns.

Product and Category Management

The **Product Management** module organizes products into categories and subcategories. **Vendors can:**

- Add and edit product details, such as name, description, and price.
- Upload product images.
- Set stock levels and reorder points.
- Create special promotions and discounts.

Order Management

The **Order Management** module handles the process of creating, tracking, and completing customer orders. **It includes:**

- Order placement and payment confirmation.
- Shipping and delivery tracking.
- Managing order statuses, such as "Processing," "Shipped," and "Delivered."
- Handling returns and refunds.

Payment Processing

The **Payment Processing** module ensures secure payments through various methods, such as credit cards, PayPal, or bank transfers. **It tracks:**

- Payment statuses for each order.
- Refund transactions.
- Payment history for customers and vendors.

Entities and Attributes

Here we describe the core entities of the system and their respective attributes.

Customer Entity

- **CustomerID:** Unique identifier for each customer.
- **FirstName:** Customer's first name.
- **LastName:** Customer's last name.
- **Email:** Customer's email address.
- **Password:** Hashed password for customer login.

- **Phone:** Customer's contact number.
- **Address:** Shipping and billing addresses.
- **JoinDate:** Date of account creation.
- **LoyaltyPoints:** Points accumulated through purchases.

Customer Relationships:

- **Customer ↔ Orders**: A customer can place many orders, but each order is linked to only one customer.
- **Customer ↔ Payment**: Customers make payments related to their orders.

Vendor (Seller) Entity

- **VendorID:** Unique identifier for each vendor.
- **StoreName:** Name of the vendor’s store.
- **OwnerName:** Vendor’s full name.
- **Email:** Vendor's email address.
- **Phone:** Contact number of the vendor.
- **StoreAddress:** Physical or virtual address of the store.
- **JoinDate:** Date of vendor registration.
- **ProductsCount:** Number of products listed by the vendor.

Vendor Relationships:

- **Vendor ↔ Products**: A vendor can have multiple products listed in the system.
- **Vendor ↔ Orders**: Vendors fulfill orders placed for their products.

Product Entity

- **ProductID:** Unique identifier for each product.
- **VendorID:** Foreign key linked to the vendor who owns the product.
- **CategoryID:** Foreign key linked to the product category.
- **ProductName:** Name of the product.
- **Description:** Detailed information about the product.
- **Price:** Price of the product.
- **StockQuantity:** Number of units available in stock.
- **Discount:** Discount on the product, if any.

- **CreatedAt:** Timestamp for when the product was listed.
- **UpdatedAt:** Timestamp for the last update.

Product Relationships:

- **Product** ↔ **Vendor**: Each product belongs to one vendor.
- **Product** ↔ **Category**: Products are categorized for easier browsing by customers.
- **Product** ↔ **OrderDetails**: Products are linked to orders through order details.

Category Entity

- **CategoryID:** Unique identifier for each category.
- **CategoryName:** Name of the product category.
- **ParentCategoryID:** Self-referencing foreign key for subcategories.

Category Relationships:

- **Category** ↔ **Products**: A category can have many products assigned to it.
- **Category** ↔ **Subcategories**: Categories can have parent-child relationships, where a parent category has multiple subcategories.

Order Entity

- **OrderID:** Unique identifier for each order.
- **CustomerID:** Foreign key linked to the customer placing the order.
- **OrderDate:** Timestamp for when the order was placed.
- **ShippingAddress:** Address to which the order will be shipped.
- **TotalAmount**: Total cost of the order.
- **PaymentStatus:** Status of the payment (e.g., Paid, Pending, Failed).
- **OrderStatus:** Status of the order (e.g., Processing, Shipped, Delivered).
- **TrackingNumber:** Tracking number for the shipment.

Order Relationships:

- **Order** ↔ **Customer**: Each order is placed by a single customer.

- **Order ↔ OrderDetails**: An order can have multiple products associated with it.

OrderDetails Entity

- **OrderDetailID:** Unique identifier for each order detail.
- **OrderID:** Foreign key linked to the order.
- **ProductID:** Foreign key linked to the product in the order.
- **Quantity:** Number of units of the product ordered.
- **Price:** Price of the product at the time of the order.

OrderDetails Relationships:

- **OrderDetails ↔ Order**: Each order can contain multiple order details.
- **OrderDetails ↔ Product**: Each order detail references a product.

Payment Entity

- **PaymentID:** Unique identifier for each payment.
- **OrderID:** Foreign key linked to the order.
- **AmountPaid:** Amount paid for the order.
- **PaymentMethod:** Method used for payment (e.g., Credit Card, PayPal).
- **PaymentDate:** Timestamp of the payment.
- **PaymentStatus:** Status of the payment.

Payment Relationships:

- **Payment ↔ Order**: Each payment is associated with one order.

Relationships Between Entities

The relationships between the system's entities ensure proper data organization and retrieval. **Below are detailed explanations of the relationships:**

1. **Customer ↔ Orders (One-to-Many)**: A customer can place multiple orders, but each order is linked to only one customer.

2. **Vendor ↔ Products (One-to-Many)**: A vendor can list multiple products, but each product belongs to a single vendor.
3. **Product ↔ Category (Many-to-One)**: Products are assigned to a category, and each category can have many products.
4. **Order ↔ OrderDetails (One-to-Many)**: An order can contain multiple products, but each order detail belongs to only one order.
5. **Order ↔ Payment (One-to-One)**: Each order is associated with one payment transaction, and each payment references one order.

Normalization

Normalization is a key part of designing a robust database to reduce redundancy and ensure efficient data storage. We will normalize the schema up to 3NF (Third Normal Form).

First Normal Form (1NF)

In 1NF, all table attributes must contain atomic values, and there should be no repeating groups or arrays.

Example:

Before 1NF:

OrderID	ProductIDs	Quantities
101	P001, P002	2, 1

After 1NF:

OrderID	ProductID	Quantity
101	P001	2
101	P002	1

Second Normal Form (2NF)

In 2NF, all non-key attributes must depend on the entire primary key.

Example:

Before 2NF:

OrderDetailID	OrderID	ProductID	VendorName
1	101	P001	Vendor A

After 2NF:

OrderDetailID	OrderID	ProductID
1	101	P001

ProductID	VendorName
P001	Vendor A

Third Normal Form (3NF)

In 3NF, there should be no transitive dependencies between non-key attributes.

Example:
Before 3NF:

CustomerID	OrderID	TotalAmount	CustomerAddress

After 3NF:

CustomerID	CustomerAddress

OrderID	TotalAmount	CustomerID

Sample Queries

Retrieve All Orders of a Specific Customer

```
SELECT OrderID, OrderDate, TotalAmount, OrderStatus

FROM Orders

WHERE CustomerID = 'CUST123';
```

Get Products under a Specific Category

```
SELECT ProductName, Price, StockQuantity

FROM Products
```

WHERE CategoryID = 'CAT456'**;**

List All Products by a Vendor

SELECT ProductName, Price, StockQuantity

FROM Products

WHERE VendorID = 'VEND789'**;**

Retrieve Payment Details for a Specific Order

SELECT AmountPaid, PaymentMethod, PaymentStatus

FROM Payments

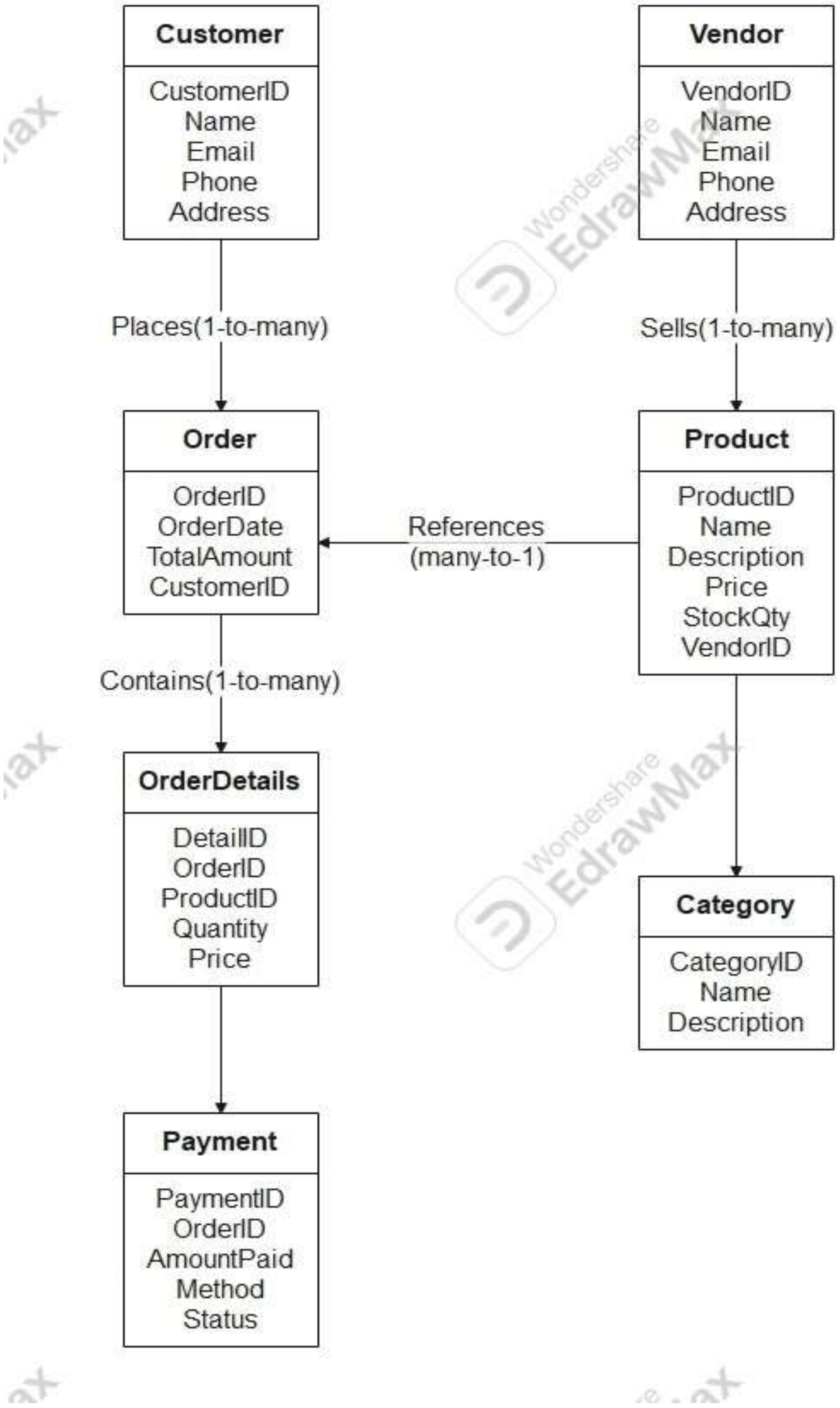
WHERE OrderID = 'ORD321'**;**

Security Considerations

To ensure the security and privacy of data within the system:

- **Customer Data Protection**: All customer passwords must be hashed using secure algorithms (e.g., bcrypt) before storage.
- **Payment Security**: Payment processing must comply with PCI DSS standards to ensure secure handling of payment data.
- **Data Encryption**: Sensitive information such as payment details and addresses should be encrypted both in transit and at rest.
- **Access Control**: Role-based access control (RBAC) should be implemented to restrict access based on user roles (e.g., customers, vendors, administrators).

ER Diagram


Customer
CustomerID
Name
Email
Phone
Address
Vendor
VendorID
Name
Email
Phone
Address
Places(1-to-many)
Sells(1-to-many)
Order
OrderID
OrderDate
TotalAmount
CustomerID
References
(many-to-1)
Product
ProductID
Name
Description
Price
StockQty
VendorID
Contains(1-to-many)
OrderDetails
DetailID
OrderID
ProductID
Quantity
Price
Category
CategoryID
Name
Description
Payment
PaymentID
OrderID
AmountPaid
Method
Status

Scalability Considerations

For large-scale implementations, the system should be designed to handle:

- **Horizontal Scaling**: The ability to add more servers to handle growing customer and vendor traffic.
- **Database Optimization**: Efficient indexing strategies and query optimization should be employed to ensure fast data retrieval.
- **Load Balancing**: The system should distribute traffic across multiple servers to prevent overload on a single server.

Conclusion

The **Generic Shop Vendor-Customer System** is a well-structured solution that facilitates smooth interactions between vendors and customers. By implementing normalization, optimizing relationships between entities, and ensuring secure transactions, the system provides a scalable and secure environment for online shopping. The sample queries and module descriptions demonstrate how the system can be used effectively in real-world scenarios.