## **Step 1: User Signup (Account Creation)**

### ****How It Works:****

1. A new user **signs up** by providing:
   * Name, Email, Password, Phone, and Address.
2. The system **inserts** this data into the **User Table**.
3. A corresponding **Address ID** is generated and stored in the **Address Table**.
4. A **record** is added in the **FirstOrderDiscount Table** with:
   * IsDiscountUsed = 0 (eligible for discount).
5. User logs in and can now **browse products**.

### ****Database Updates:****

* **User Table:** New entry for the user.
* **Address Table:** Stores the user’s address.
* **FirstOrderDiscount Table:** Tracks first-order discount eligibility.

## **Step 2: Admin Manages Products (Add, Update, Remove)**

### ****How It Works:****

1. **Adding Products:**
   * The admin **adds a new product** with Name, Description, Price, Stock, and Category.
   * Product details are inserted into the **Product Table**.
   * A log is created in the **ProductAudit Table** (ChangeType = ‘Added’).
2. **Updating Products:**
   * Admin **updates** product details (e.g., price, stock).
   * The **Product Table** is updated.
   * The old value is logged in **ProductAudit Table** (ChangeType = ‘Updated’).
3. **Removing Products:**
   * Instead of deleting, IsActive = 0 is updated in the **Product Table**.
   * Logs removal in **ProductAudit Table** (ChangeType = ‘Removed’).

### ****Database Updates:****

* **Product Table:** Stores active products.
* **ProductAudit Table:** Tracks all changes made by admins.

## **Step 3: User Adds Items to Cart**

### ****How It Works:****

1. User selects products and adds them to their cart.
2. A **CartID** is assigned if the user does not have an existing cart.
3. The **Cart\_Items Table** records:
   * CartID
   * ProductID
   * Quantity
4. Stock levels are **not updated** yet (only when order is placed).

### ****Database Updates:****

* **Cart Table:** Stores user’s active cart.
* **Cart\_Items Table:** Stores products added to the cart.

## **Step 4: User Places an Order**

### ****How It Works:****

1. The user proceeds to checkout.
2. The system **checks if the user qualifies** for a first-order discount.
   * If IsDiscountUsed == 0, **applies 10% off**.
   * The system updates IsDiscountUsed = 1 in **FirstOrderDiscount Table**.
3. If the user **applies a Gift Card**, the system:
   * **Checks if it's used before** (UsedStatus == 0).
   * Deducts the gift card amount.
   * Updates UsedStatus = 1 in **GiftCard Table**.
4. The **final order amount** is calculated.
5. The system creates a new **OrderID** in **Orders Table**.
6. The **Payment Table** records:
   * Payment Method (Card, Gift Card, Cash, etc.).
   * Payment Status.
7. The ordered products **reduce stock** in the **Product Table**.

### ****Database Updates:****

* **Orders Table:** New order created.
* **FirstOrderDiscount Table:** IsDiscountUsed = 1 if discount applied.
* **GiftCard Table:** UsedStatus = 1 if used.
* **GiftCardTransaction Table:** Logs gift card use.
* **Payment Table:** Stores payment details.
* **Product Table:** Updates stock for ordered items.

## **Step 5: Delivery System & Order Tracking**

### ****How It Works:****

1. A **Courier is assigned** from the **Courier Table**.
2. A new **DeliveryID** is created in **Delivery Table** with:
   * OrderID
   * DeliveryStatus = ‘Pending’
   * TrackingNumber
   * EstimatedDeliveryDate
3. The **Admin updates** the order:
   * When shipped: DeliveryStatus = 'Shipped'
   * When delivered: DeliveryStatus = 'Delivered', and ActualDeliveryDate is recorded.

### ****Database Updates:****

* **Delivery Table:** Tracks the status of the order.
* **Courier Table:** Links the order to a courier.

## **Step 6: Order Completion & Review**

### ****How It Works:****

1. Once delivered, the user can **leave a review** for products.
2. The **Admin can analyze** sales, update inventory, and track order history.

### ****Final Database Updates:****

* **Orders Table:** Final status updates.
* **Product Table:** Stock adjustments.
* **Delivery Table:** Marks order as delivered.

## **1️⃣ First-Order Discount System**

Yes, this feature should be in the **database schema**, as we need to track whether a user has placed their first order.

### ****New Table: FirstOrderDiscount****

* UserID (Primary Key, FK referencing User Table)
* IsDiscountUsed (Boolean: 0 = Not Used, 1 = Used)
* DiscountAmount (Stores 10% of their first order total)
* AppliedOnOrderID (FK referencing Orders Table, if used)

### ****How It Works (Real-Time Process)****

1. **User Signs Up & Places First Order**
   * Before placing an order, system checks if UserID exists in FirstOrderDiscount.
   * If not, a **10% discount** is calculated & stored in DiscountAmount.
2. **Applying Discount**
   * If IsDiscountUsed == 0, system applies DiscountAmount at checkout.
   * Order total updates accordingly.
   * IsDiscountUsed updates to **1**, preventing reuse.

## **2️⃣ Admin Product Management System**

Yes, this feature needs to be in the **database schema** to allow **adding, updating, and deleting products**.

### ****Modifications to Product Table****

* ProductID (Primary Key)
* Name
* Description
* Price
* StockQuantity (Ensures stock tracking)
* CategoryID (FK referencing Category Table)
* IsActive (Boolean: 1 = Available, 0 = Removed by Admin)

### ****New Table: ProductAudit**** (Tracks Changes by Admins)

* AuditID (Primary Key)
* AdminID (FK referencing Admin Table)
* ProductID (FK referencing Product Table)
* ChangeType (‘Added’, ‘Updated’, ‘Removed’)
* ChangeDetails (JSON format for details like price change, description update, etc.)
* Timestamp

### ****How It Works (Real-Time Process)****

1. **Admin Adds a New Product**
   * Inserts into Product Table.
   * Logs action in ProductAudit.
2. **Admin Updates a Product**
   * Updates Product Table fields.
   * Logs old & new values in ProductAudit.
3. **Admin Removes a Product**
   * Instead of deleting, IsActive is set to **0**.
   * This keeps data for reporting but hides the product from users.

**1️⃣ GiftCard Table (Updated)**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| GiftCardID (PK) | INT | Unique ID for the gift card |
| SenderUserID (FK) | INT | References UserTable (Sender) |
| RecipientEmail | VARCHAR(255) | Email of the recipient |
| GiftAmount | DECIMAL(10,2) | Fixed balance |
| UsedStatus | BOOLEAN | If used (0 = Unused, 1 = Used) |
| PersonalNote | TEXT | Message from sender |
| GeneratedAt | TIMESTAMP | When created |
| ExpirationDate | DATE | Expiry date (optional) |
| **PaymentID (FK)** | INT | **References PaymentTable (NEW Addition)** |

✅ **New Addition:**

* **PaymentID (FK)** is added to track the payment for each gift card purchase.

**2️⃣ Payment Table (Updated)**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| PaymentID (PK) | INT | Unique ID for payment |
| OrderID (FK, Nullable) | INT | References OrdersTable (NULL if not an order) |
| **GiftCardID (FK, Nullable)** | INT | **References GiftCardTable (If payment is for a gift card)** |
| PaymentMethod | ENUM('Credit Card', 'Bkash', 'Nagad') | Payment type |
| TransactionID | VARCHAR(255) | Transaction reference |
| AmountPaid | DECIMAL(10,2) | Payment amount |

✅ **New Addition:**

* **GiftCardID (FK)** is added to **track gift card payments** separately.

**3️⃣ GiftCardTransaction Table (Updated)**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| TransactionID (PK) | INT | Unique transaction ID |
| GiftCardID (FK) | INT | References GiftCardTable |
| OrderID (FK, Nullable)` | INT | If used for an order |
| AmountUsed | DECIMAL(10,2) | Amount deducted |
| TransactionDate | TIMESTAMP | Date used |

✅ **No major changes needed here.**

## **🔹 How the System Will Work (Step-by-Step)**

### ****🔸 Step 1: User Clicks "Proceed to Buy" in Buy Gift Card Section****

* A **payment form** appears with **amount, recipient email, personal note, and payment options**.
* User enters details & submits payment.

### ****🔸 Step 2: Payment Processing****

1️⃣ **GiftCard & Payment Data is Stored**

* The **gift card details** (amount, recipient email, sender’s note) are stored in the GiftCard table.
* A **payment entry** is stored in the Payment table, linking to the GiftCardID.

### ****🔸 Step 3: Trigger Emails****

1️⃣ **Sender Gets a Payment Confirmation Email**

* **"Your gift card purchase is successful!"**
* Includes **transaction details** and **GiftCard ID**.

2️⃣ **Recipient Gets a Gift Card Email**

* **"You’ve received a gift card from [Sender Name]!"**
* Includes **Gift Amount, Sender’s Note, and Instructions** on how to use it.

### ****🔸 Step 4: Using the Gift Card****

1️⃣ **Recipient places an order** and chooses "Gift Card" as payment.  
2️⃣ If **order total ≤ gift amount** → No extra payment needed.  
3️⃣ If **order total > gift amount** → Recipient pays the remaining balance.  
4️⃣ **Gift card is marked as used (UsedStatus = 1).**  
5️⃣ Entry added to GiftCardTransaction table.

## **📌 How the Full Process Works – Step by Step**

Now, let’s go through the **workflow of the entire system**, including the rider’s role.

### ****🛒 Step 1: User Signup & Places Order****

📌 **User Signup**

1. User **signs up** → Stored in **User Table**.
2. Address is added to **Address Table**.
3. If it’s their **first order**, entry is made in **FirstOrderDiscount Table**.

📌 **User Places an Order**

1. Adds products to the **Cart Table**.
2. System **checks first-order discount** and applies it if applicable.
3. Order details go into **Orders Table**.
4. Default DeliveryStatus = ‘Pending’ in **Delivery Table**.

🛑 **No Rider Assigned Yet!**

### ****🛠 Step 2: Admin Assigns Order as “Shipped”****

📌 **Admin Updates Order**

1. Admin updates DeliveryStatus = ‘Shipped’.
2. System **finds available riders** for the **customer's Upazila** in **RiderPreferredArea Table**.
3. Selects **a random rider** who has:
   * **Matching area** in RiderPreferredArea Table
   * **Not exceeded max deliveries**
4. **RiderID is updated** in **Delivery Table**.
5. The assigned rider **can now see the order in their dashboard**.

✅ **Now the order is assigned to a Rider!**

### ****🚴 Step 3: Rider Logs In & Sees Assigned Deliveries****

📌 **Rider Dashboard**

1. Rider logs in → System shows **Pending Deliveries** for their area.
2. Rider can **track assigned orders**.

📌 **Delivery Process**

1. Once Rider **delivers the order**, they **update status to “Delivered”**.
2. ActualDeliveryDate is recorded in **Delivery Table**.

✅ **Order is now completed!**

## **📌 How This Helps in Rider Matching**

Now, we can easily find available riders for an order using:

sql

CopyEdit

SELECT r.RiderID

FROM RiderPreferredArea r

JOIN Address a ON r.CityCorporation = a.CityCorporation AND r.Upazila = a.Upazila

WHERE a.UserID = (SELECT UserID FROM Orders WHERE OrderID = ?)

AND r.RiderID NOT IN (SELECT RiderID FROM Delivery WHERE DeliveryStatus = 'Shipped');

🔹 **This query finds a rider who:**

* **Delivers to the user’s area** (matching City Corporation + Upazila).
* **Has not exceeded delivery limits**.
* **Hasn’t already been assigned an order that’s still pending delivery**.

## **📌 Final Workflow for Automatic Rider Assignment**

1. **User places an order** → Address is stored with structured **City Corporation & Upazila**.
2. **Admin updates order to "Shipped"** → System looks for available riders in **RiderPreferredArea Table**.
3. **Matching riders are fetched** → A **random rider** is assigned from the available pool.
4. **Rider sees order in their dashboard** and updates delivery status accordingly.

✅ **Now, order-rider matching is fully automated & optimized!**

## **📌 Step-by-Step Process for Assigning a Rider**

### ****1️⃣ User Places an Order****

* Order is inserted into the **Orders Table**.
* DeliveryStatus = "Pending" is set in **Delivery Table**.
* No rider is assigned yet (RiderID = NULL).

### ****2️⃣ Admin Updates Order to "Shipped"****

* The system needs to **find a rider** who:  
  ✅ **Delivers to the same Upazila & City Corporation as the customer**.  
  ✅ **Has less than 4 active orders** (Pending + Shipped).  
  ✅ **Has not exceeded their maximum deliveries (4)**.

## **📌 SQL Query to Assign a Rider**

We can use this SQL query to **find an available rider**:

sql

CopyEdit

SELECT r.RiderID

FROM RiderPreferredArea r

JOIN Address a ON r.CityCorporation = a.CityCorporation AND r.Upazila = a.Upazila

LEFT JOIN Delivery d ON r.RiderID = d.RiderID AND d.DeliveryStatus IN ('Pending', 'Shipped')

WHERE a.UserID = (SELECT UserID FROM Orders WHERE OrderID = ?)

GROUP BY r.RiderID

HAVING COUNT(d.DeliveryID) < 4

ORDER BY RAND()

LIMIT 1;

### ****🛠 Explanation:****

1. Joins **RiderPreferredArea** to **Address** to find riders covering the same City Corporation & Upazila.
2. Left joins with **Delivery Table** to count how many active orders (Pending + Shipped) a rider already has.
3. Filters only **riders who have fewer than 4 active orders**.
4. Randomly picks **one available rider** (ORDER BY RAND() ensures fair assignment).
5. Assigns the **selected rider’s RiderID** to the order in the **Delivery Table**.

## **📌 Step 3️⃣: Update the Delivery Table with Assigned Rider**

Once a rider is selected, update the **Delivery Table**:

sql

CopyEdit

UPDATE Delivery

SET RiderID = ?, DeliveryStatus = 'Shipped'

WHERE OrderID = ?;

📌 **This ensures:**

* A **rider is automatically assigned** if they meet the conditions.
* The **system prevents overloading a rider with more than 4 deliveries**.

## **📌 Step 4️⃣: Rider Dashboard Shows Assigned Deliveries**

* When the rider **logs in**, they see all **Pending + Shipped** orders assigned to them.
* Once a delivery is completed, the rider updates **DeliveryStatus = "Delivered"**, freeing up space for new deliveries.

## **🖥️ Admin Dashboard Layout & Features**

The admin dashboard will have a **navigation bar** with the following sections:

### ****1️⃣ Orders Management (Order Tracking & Status Update)****

🔹 **Page Overview**:

* Admin can **view all orders** (Order ID, Customer Details, Ordered Products, Total Amount, Delivery Status).
* A **checkbox** allows updating the **Delivery Status** from **Pending → Shipped**.
* A **"Save Changes" button** commits updates to the database.

🔹 **🔄 Database Considerations**:

* **Orders Table** already exists, but add an **"UpdatedBy" field** in the **Delivery Table** to track who updated the status.

sql

CopyEdit

ALTER TABLE Delivery ADD COLUMN UpdatedBy INT; -- Stores AdminID who updated the delivery

* **Query to Fetch Orders for Admin View**

sql

CopyEdit

SELECT o.OrderID, o.UserID, u.Name, u.Email, p.ProductName, od.Quantity, od.TotalPrice, d.DeliveryStatus

FROM Orders o

JOIN Users u ON o.UserID = u.UserID

JOIN OrderDetails od ON o.OrderID = od.OrderID

JOIN Products p ON od.ProductID = p.ProductID

JOIN Delivery d ON o.OrderID = d.OrderID;

* **Query to Update Delivery Status**

sql

CopyEdit

UPDATE Delivery

SET DeliveryStatus = 'Shipped', UpdatedBy = ?

WHERE OrderID = ?;

### ****2️⃣ Customer Search (Search by Customer Name or ID)****

🔹 **Page Overview**:

* A **search bar** allows admin to search for customers by **Customer ID, Name, or both**.
* Displays customer details (Name, Email, Address, Total Orders).

🔹 **🔄 Database Considerations**:

* No schema changes required, but an **index** on UserID and Name will improve search speed.

sql

CopyEdit

CREATE INDEX idx\_user\_search ON Users (UserID, Name);

* **Query to Fetch Customer Data**

sql

CopyEdit

SELECT UserID, Name, Email, PhoneNumber, COUNT(o.OrderID) AS TotalOrders

FROM Users u

LEFT JOIN Orders o ON u.UserID = o.UserID

WHERE u.Name LIKE ? OR u.UserID = ?

GROUP BY u.UserID;

### ****3️⃣ Product Management (Add, Update, Remove Products)****

🔹 **Page Overview**:

* Admin can **add new products**, **update existing ones**, or **remove products** from the system.
* Product details include: **Name, Price, Stock, Category, Description, Image, Status (Active/Inactive)**.

🔹 **🔄 Database Considerations**:

* Add a Status column (Active/Inactive) to handle **soft deletion** instead of permanently deleting products.

sql

CopyEdit

ALTER TABLE Products ADD COLUMN Status ENUM('Active', 'Inactive') DEFAULT 'Active';

* **Query to Add a Product**

sql

CopyEdit

INSERT INTO Products (ProductName, Description, Price, Stock, Category, Image, Status)

VALUES (?, ?, ?, ?, ?, ?, 'Active');

* **Query to Update Product**

sql

CopyEdit

UPDATE Products

SET ProductName = ?, Price = ?, Stock = ?, Description = ?, Category = ?, Image = ?, Status = ?

WHERE ProductID = ?;

* **Query to Remove a Product (Soft Delete)**

sql

CopyEdit

UPDATE Products SET Status = 'Inactive' WHERE ProductID = ?;

* Designing the **Admin Dashboard UI**.
* Creating **Navigation Bar & Buttons**.
* Adding **Search Bars, Checkboxes, and Forms**.
* Handling **AJAX calls** to update orders, products, and customers without page reload.

**✅ Back-End & Database Responsibilities:**

* **Fetching order, customer, and product data** (SQL queries).
* **Updating the delivery status** in the database.
* **Ensuring a product is removed safely** (Soft Delete mechanism).
* **Optimizing the database structure** for faster queries.

### 📌 How Everything Works Together (Step-by-Step)

1. **User Signups & Places Order**
   * If it’s the **first order**, system **applies a 10% discount**.
2. **Admin Views Orders**
   * Can **update status from "Pending" → "Shipped"**.
3. **Rider Assignment**
   * System **auto-matches rider** with a **max of 4 deliveries**.
   * Rider **logs in & sees pending orders**.
4. **Product Management by Admin**
   * Admin **adds, updates, or marks products as inactive**.
5. **Gift Cards**
   * User **buys & gifts a card**.
   * Recipient **redeems it for a purchase**.
   * System **automatically calculates the remaining amount**.

**1️⃣ Admin Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| AdminID (PK) | INT | Unique identifier for admin |
| Name | VARCHAR(100) | Admin’s name |
| Email | VARCHAR(255) | Unique email for login |
| Password | VARCHAR(255) | Hashed password |

**✅ Improvements:**

* Added **VARCHAR(255)** for Email to handle different email lengths.

**2️⃣ User Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| UserID (PK) | INT | Unique identifier for users |
| Name | VARCHAR(100) | User’s name |
| Email | VARCHAR(255) | Unique email for login |
| Password | VARCHAR(255) | Hashed password |
| Phone | VARCHAR(15) | Contact number |

**✅ Improvements:**

* Used **VARCHAR(15)** for Phone instead of INT to handle international numbers.

**3️⃣ Address Table *(Snowflake for User Table)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| AddressID (PK) | INT | Unique identifier for addresses |
| UserID (FK) | INT | References UserTable |
| FullAddress | TEXT | Complete address |
| CityCorporation | VARCHAR(50) | City corporation name |
| Upazila | VARCHAR(50) | Upazila name |
| PostalCode | VARCHAR(10) | Postal code |

**✅ Improvements:**

* Indexed **CityCorporation** & **Upazila** for **faster matching** with riders.
* Used VARCHAR(10) for PostalCode to accommodate different formats.

**4️⃣ Orders Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| OrderID (PK) | INT | Unique order ID |
| UserID (FK) | INT | References UserTable |
| TotalAmount | DECIMAL(10,2) | Total order value |
| OrderDate | TIMESTAMP | When the order was placed |
| Status | ENUM('Pending', 'Shipped', 'Delivered') | Order status |
| PaymentID (FK) | INT | References PaymentTable |

**✅ Improvements:**

* Used **ENUM** for Status instead of VARCHAR to save space.
* Indexed OrderDate for faster order history retrieval.

**5️⃣ Payment Table *(Snowflake for Orders Table)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| PaymentID (PK) | INT | Unique ID for payment |
| OrderID (FK) | INT | References OrdersTable |
| PaymentMethod | ENUM('Credit Card', 'COD', 'Gift Card', 'Bkash', 'Nagad') | Payment type |
| TransactionID | VARCHAR(255) | Transaction reference |
| AmountPaid | DECIMAL(10,2) | Payment amount |

**✅ Improvements:**

* Used **ENUM** for PaymentMethod to limit valid options.

**6️⃣ Cart Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| CartID (PK) | INT | Unique cart ID |
| UserID (FK) | INT | References UserTable |

**7️⃣ Cart\_Items Table *(Snowflake for Cart Table)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| CartItemID (PK) | INT | Unique item ID |
| CartID (FK) | INT | References CartTable |
| ProductID (FK) | INT | References ProductTable |
| Quantity | INT | Number of units |

**8️⃣ Product Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| ProductID (PK) | INT | Unique product ID |
| Name | VARCHAR(255) | Product name |
| Price | DECIMAL(10,2) | Product price |
| CategoryID (FK) | INT | References CategoryTable |
| Description | TEXT | Product details |
| StockQuantity | INT | Available stock |
| IsActive | BOOLEAN | If the product is available |

**9️⃣ Category Table *(Snowflake for Product Table)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| CategoryID (PK) | INT | Unique category ID |
| CategoryName | VARCHAR(255) | Category name |

**🔟 ProductAudit Table *(Tracks Changes by Admin)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| AuditID (PK) | INT | Unique audit ID |
| AdminID (FK) | INT | References AdminTable |
| ProductID (FK) | INT | References ProductTable |
| ChangeType | ENUM('Added', 'Updated', 'Removed') | Type of change |
| ChangeDetails | JSON | Stores details like price change |
| Timestamp | TIMESTAMP | Change timestamp |

**1️⃣1️⃣ Delivery Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| DeliveryID (PK) | INT | Unique delivery ID |
| OrderID (FK) | INT | References OrdersTable |
| DeliveryStatus | ENUM('Pending', 'Shipped', 'Delivered') | Status |
| RiderID (FK) | INT | References RiderTable |
| EstimatedDeliveryDate | DATE | Expected delivery date |
| ActualDeliveryDate | DATE | Actual delivery date |
| TrackingNumber | VARCHAR(255) | Tracking number |

**1️⃣2️⃣ Rider Table *(Snowflake for Delivery Table)***

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| RiderID (PK) | INT | Unique rider ID |
| Name | VARCHAR(100) | Rider’s name |
| Email | VARCHAR(255) | Login email |
| ContactNumber | VARCHAR(15) | Phone number |
| MaxDeliveries | INT | Max deliveries allowed (4) |
| ActiveStatus | BOOLEAN | If available |

**1️⃣3️⃣ RiderPreferredArea Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| RiderID (FK) | INT | References RiderTable |
| CityCorporation | VARCHAR(50) | Chosen city corporation |
| Upazila | VARCHAR(50) | Chosen upazila |

**1️⃣4️⃣ GiftCard Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| GiftCardID (PK) | INT | Unique ID |
| SenderUserID (FK) | INT | User sending the card |
| RecipientEmail | VARCHAR(255) | Email of recipient |
| GiftAmount | DECIMAL(10,2) | Fixed balance |
| UsedStatus | BOOLEAN | If used |
| PersonalNote | TEXT | Message from sender |
| GeneratedAt | TIMESTAMP | When created |
| ExpirationDate | DATE | Expiry date (if applicable) |

**1️⃣5️⃣ GiftCardTransaction Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| TransactionID (PK) | INT | Unique transaction ID |
| GiftCardID (FK) | INT | References GiftCardTable |
| OrderID (FK) | INT | References OrdersTable |
| AmountUsed | DECIMAL(10,2) | Amount deducted |
| TransactionDate | TIMESTAMP | Date used |

**1️⃣6️⃣ FirstOrderDiscount Table**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| UserID (PK, FK) | INT | References UserTable |
| IsDiscountUsed | BOOLEAN | If used |
| DiscountAmount | DECIMAL(10,2) | 10% of first order |
| AppliedOnOrderID (FK) | INT | References OrdersTable |