Project: Online Store System

Name	Id	Group
هناء شريف صلاح عبد الغني ادريس	2401249723	G2
منة محمد محمود محمد حسين	2401246474	G2
مريم محمد عزت محمد	2401242374	G2
ندى احمد احمد ابراهيم	2401241250	G2

Design Phase

1. Entity-Relationship Diagram (ERD) Identification

1.1 Overview

The database structure was identified using a conceptual model (Chen Notation) focusing on the relationship between customers, their shopping carts, and the final orders.

1.2 Entities & Attributes

Based on the visual design, the following entities and specific attributes have been defined:

- **Customer**
    - customer_id (PK): Unique identifier.
    - email: User login.
    - password: Security credential.
    - name: Composite attribute broken down into f_name (First Name) and l_name (Last Name).
    - address: Shipping/Billing address.
    - phone_num: Contact number.

- **Products**
  - product_id (PK): Unique identifier.
  - sku: Stock Keeping Unit code.
  - name: Product title.
  - stock: Current inventory level.
  - cost: The cost to the store (for profit calculation).
  - retail_price: The selling price to the customer.
- **Cart & Cart_Item (Transient Data)**
  - Cart: cart_id (PK), quantity, created_at.
  - Cart_Item: cart_item_id (PK), quantity.
  - *Relationship:* A Cart *contains* many Cart_Items. Products are *included in* Cart_Items.
- **Order & Order_Item (Historical Data)**
  - Order: order_id (PK), created_at, total_price, status (e.g., 'confirmed', 'cancelled').
  - Order_Item: order_item_id (PK), quantity, price.
  - *Relationship:* An Order *includes* many Order_Items.
- **Payment**
  - payment_id (PK): Unique identifier.
  - amount: value paid.
  - method: Payment mode (Credit Card, PayPal, etc.).
  - date: Transaction timestamp.

2. Transaction Pathways (Business Logic)

This section documents the specific data flow required for the core business operations: placing an order and canceling it.

2.1 Pathway: Placing an Order

1. **Authentication Phase:**
    - User attempts Sign in/Sign up.
    - Check: Verify if user exists. If False, loop back to creation. If True, proceed.
2. **Selection Phase (Cart Management):**
    - Action: Product Selection.
    - DB Operation: INSERT, UPDATE, or DELETE operations performed on Cart and Cart_Item tables as the user modifies their basket.
3. **Checkout & Payment Phase:**
    - Action: User initiates Checkout.
    - DB Operation: INSERT INTO Payment (Records the payment method and details).
4. **Finalization Phase (The Order Commit):**
    - Step 1 (Create Order): INSERT INTO Order with status set to "confirmed".
    - Step 2 (Migrate Data): Copy data from Cart_Item table to Order_Item table to create a permanent record of the items purchased.
    - Step 3 (Inventory Update):
        - Logic: New Quantity = Old Quantity - Ordered Quantity
        - DB Operation: UPDATE Products SET stock = ...

2.2 Pathway: Canceling an Order

1. **Validation:**

    - Check: System verifies if the order status is currently "complete".

2. **Inventory Restoration:**
    - Action: Return items to inventory.
    - DB Operation: UPDATE Products by adding the ordered items back to the stock count.
3. **Status Update:**
    - Action: Mark order as void.
    - DB Operation: UPDATE Order SET status = 'cancelled'.
4. **Completion:**
    - Process ends successfully.

Mapping & create Schema & Data Dictionary

This part outlines the logical and physical database design for an Online Store application. The design process transitions from conceptual mapping to a detailed data dictionary, ensuring data integrity, normalization, and scalability. The system comprises 8 core tables designed to handle customers, products, shopping carts, orders, and payments.

1. Entity-to-Table Mapping

The mapping phase converted logical entities into physical database tables. The design identifies the following key structures:

User Management:

Customer: Stores core profile information including name, email, password, and address.

Customer_phone: A dedicated table for phone numbers, adhering to normalization rules to allow multiple contacts per customer.

Inventory & Shopping:

Products: Manages inventory details, including SKU, stock levels, cost, and retail price.

Cart & Cart_item: A decomposed structure where Cart represents the session/container and Cart_item stores specific products and quantities linked to that cart.

Order Processing:

Order: Represents the header information of a transaction (total price, status, date).

Order item: Represents the line items of an order, linking specific products to the order header.

Financials:

Payment: Records transaction details, methods, and amounts linked to customers.

2. Relational Schema

The relational schema defines the structural blueprint, establishing Primary Keys (PK) and Foreign Keys (FK) to ensure referential integrity:

Primary Keys: Every table utilizes a unique identifier (e.g., customer_id, order_id, product_id) as a Primary Key.

Relationships:

The Customer is linked to Order, Cart, Payment, and Customer_phone via the customer_id Foreign Key.

Order item and Cart item act as junction tables to handle the many-to-many relationship between orders/carts and products.

3. Data Dictionary & Technical Specifications

The data dictionary provides granular details regarding data types and constraints to ensure data quality.

A. Data Types

Integers (INT): Used for IDs (PK/FK) and quantities (e.g., stock, quantity).

Strings (VARCHAR): Used for names, emails, passwords, and addresses with varying lengths (e.g., VARCHAR(50) for names, VARCHAR(100) for passwords).

Financials (DECIMAL): Prices (cost, retail_price, total_price) are stored as DECIMAL to ensure mathematical precision for currency.

Dates (DATETIME): Used for tracking event timestamps such as created_at for carts/orders and date for payments.

B. Key Constraints

AUTO INCREMENT: Applied to all Primary Keys to automate ID generation.

NOT NULL: Enforced on essential fields (e.g., email, prices, status) to prevent incomplete data entries.

UNIQUE: Applied to the email field in the Customer table and sku in the Products table to prevent duplication.

DEFAULT 0: Used for quantity in the Cart table and stock in Products to initialize values safely.

FK, CASCADE: Implemented on relationships (like Customer_phone) to ensure that if a parent record (Customer) is deleted, the related child records are automatically removed.

4. Design Highlights

Price History Accuracy: The Order item table includes a price column described as a "Snapshot of the product price at the time of purchase". This is a critical design feature that preserves historical data accuracy even if the main product price changes later.

Normalization: Separating Customer_phone and decomposing orders into Order and Order item demonstrates a normalized design (likely 3NF), reducing redundancy.

Creation Tables

In this part of the project, I designed a relational database for an **Online Store system**. The goal was to create a structured and organized database model that handles customers, products, carts, orders, and payments. Below is a detailed explanation of the tables I created and the purpose of each one.

1. **Creating the Database**
   I started by creating a new database called Online_Store, which contains all the tables related to the system.
2. **Customers Table**
   This table stores the main information about each customer.
   It includes:
   - First and last name
   - Email (unique)
   - Password
   - Address

   The customer_id field is the primary key and is set to auto-increment.
3. **Customer Phones**
   Table Since a customer can have more than one phone number, I created a separate table for phone numbers.
   It contains:
   - A unique phone ID
   - The customer ID it belongs to (Foreign Key)
   - The phone number

   I used ON DELETE CASCADE to ensure that when a customer is deleted, their phone numbers are deleted automatically.
4. **Cart Table**
   Each customer can have a cart.
   This table stores:
   - Cart ID

- Customer ID
- Quantity of items in the cart
- Creation date

The customer ID is a Foreign Key with cascade delete, so removing a customer also removes their cart.

5. Products Table

This table stores information about all products available in the store.
It includes:

- Product name
- SKU (unique)
- Stock quantity
- Cost and retail price

Each product has a unique product_id.

6. CartItem Table

Because each cart can contain multiple products, this table links products to the cart.
It stores:

- Cart ID
- Product ID
- Quantity of each product

Both IDs are set as foreign keys, and cart_id uses ON DELETE CASCADE to remove cart items if the cart is deleted.

7. Orders Table

This table stores all customer orders.
It includes:

- Order ID
- Customer ID
- Date of creation
- Total price

- Order status

This creates a connection between customers and the orders they place.

8. **OrderItem Table**

   Each order can contain multiple products.
   This table stores:

   - Order ID
   - Product ID
   - Quantity
   - Price at the time of purchase

   The order_id key uses **ON DELETE CASCADE** so that if an order is removed, all of its items are automatically deleted.

9. **Payment Table**

   This table records payment information for customers.
   It includes:

   - Payment amount
   - Payment method
   - Date of payment
   - Customer ID

   This links each payment to the customer who made it.

Final Output

The final database design creates a fully connected structure that supports customers, products, carts, orders, and payments. The use of primary keys, foreign keys, and cascade delete ensures data integrity and smooth relationships between tables. This design supports an online store's basic operations efficiently and clearly.

Database Queries and Analysis

1. **Total Number of Customers**

Query:

```
SELECT
  COUNT(DISTINCT customer_id) AS total_customers
FROM
  Customers;
```

Explanation:

This query calculates the total number of unique customers registered in the system by counting distinct customer IDs from the Customers table.

2. Low Stock Products

Query:

```
SELECT
   product_id, name, stock
FROM
   Products
WHERE stock < 10;
```

Explanation:

This query retrieves the list of products whose available stock is less than 10 units. It helps in identifying products that are close to being out of stock and require restocking.

3. Number of Orders per Customer

Query:

```
SELECT
   c.customer_id ID,
   CONCAT(c.f_name, ' ', c.l_name) AS Name,
   COUNT(o.order_id) AS Total_Orders
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id;
```

Explanation:

This query displays each customer's full name along with the total number of orders they have placed. A LEFT JOIN is used to ensure that customers with no orders are also included in the results.

4. Best-Selling Products

Query:

```
SELECT
   p.product_id Product_ID,
   p.name AS Name,
   SUM(oi.quantity) AS Total_Sold
FROM Products p
JOIN OrderItem oi ON p.product_id = oi.product_id
GROUP BY p.product_id
ORDER BY total_sold DESC;
```

Explanation:

This query calculates the total quantity sold for each product by summing the quantities from the OrderItem table. The results are sorted in descending order to identify the best-selling products.

5. Daily Sales Report

Query:

```
SELECT
   DATE(created_at) AS Order_Date,
   SUM(total_price) AS Daily_Sales
FROM Orders
GROUP BY DATE(created_at)
ORDER BY order_date;
```

Explanation:

This query calculates the total sales for each day by grouping orders according to their creation date. It provides insights into the daily performance of the business.

6. Top Spending Customers

Query:

```
SELECT
   c.customer_id ID,
   CONCAT(c.f_name, ' ', c.l_name) AS Name,
   SUM(o.total_price) AS Total_Spent
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_id
ORDER BY total_spent DESC;
```

Explanation:

This query determines the total amount of money spent by each customer by summing the total price of their orders. The results are ordered in descending order to identify the highest-spending customers.

7. Shopping Cart Details

Query:

```
SELECT
   CONCAT(cu.f_name, ' ', cu.l_name) AS Name,
   p.name AS Product_Name,
   ci.quantity Quantity
FROM CartItem ci
JOIN Cart ca ON ci.cart_id = ca.cart_id
JOIN Customers cu ON ca.customer_id = cu.customer_id
JOIN Products p ON ci.product_id = p.product_id;
```

Explanation:

This query displays the contents of each customer’s shopping cart, including the customer’s name, product name, and selected quantity.

8. Total Profit per Product

Query:

```
SELECT
   p.product_id ID,
   p.name Product_Name,
   SUM(oi.quantity * (p.retail_price - p.cost)) AS Total_Profit
FROM Products p
JOIN OrderItem oi ON p.product_id = oi.product_id
GROUP BY p.product_id
ORDER BY total_profit DESC;
```

Explanation:

This query calculates the total profit generated from each product by computing the difference between the retail price and cost, multiplied by the total quantity sold. The results are sorted to show the most profitable products first.