DBMS Project Deadline - 6

Tikam (2022542)

Non-Conflicting Transactions

Sign up of a new user:
→ When a new user signs up, their username and password are inserted into the Customer_Account table in the database.

SQL Query :
INSERT INTO Users (User_ID, Name, Phone_No, Address, Order_History, Account_Status, Password)

VALUES (11, 'NewUser', 1234567890, 'New Address', 0, 'Unblocked', 'password123');

Code:

```
class SignUpPage(tk.Toplevel):
    def _init_(self, master):
        tk.Toplevel._init_(self, master)
        self.title("Sign Up")
        self.geometry("300x150")

        self.username_label = ttk.Label(self, text="Username:")
        self.username_entry = ttk.Entry(self)
        self.password_label = ttk.Label(self, text="Password:")
        self.password_entry = ttk.Entry(self, show="*")

        self.sign_up_button = ttk.Button(self, text="Sign Up",
command=self.sign_up)

        self.username_label.pack()
        self.username_entry.pack()
        self.password_label.pack()
        self.password_entry.pack()
        self.sign_up_button.pack()
```

```
    def sign_up(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        # Check if username already exists
        cursor.execute("SELECT * FROM Customer_Account WHERE Username =
?", (username,))
        existing_user = cursor.fetchone()
        if existing_user:
            tkMessageBox.showerror("Error", "Username already exists.
Please choose another one.")
        else:
            # Insert new user into the database
            cursor.execute("INSERT INTO Customer_Account (Username,
Password) VALUES (?, ?)", (username, password))
            conn.commit()
            tkMessageBox.showinfo("Success", "Account created
successfully!")
            self.destroy()
```

Payment option comes:
→ When the user proceeds to checkout, a payment page is displayed, and upon successful payment, the order is processed.

Query:
INSERT INTO Payment (Payment_ID, Amount, Order_ID)
VALUES (11, 20000, 11);

Code:

```
class PaymentPage(tk.Toplevel):
    def _init_(self, master, total_cost, order_id):
        tk.Toplevel._init_(self, master)
        self.title("Payment")
        self.geometry("300x150")

        self.total_cost_label = ttk.Label(self, text="Total Cost:
${}".format(total_cost))
        self.pay_button = ttk.Button(self, text="Pay", command=lambda:
self.process_payment(order_id))
```

```
        self.total_cost_label.pack()
        self.pay_button.pack()

    def process_payment(self, order_id):
        # Placeholder function for payment processing
        tkMessageBox.showinfo("Payment", "Payment successful!")
        self.destroy()
        # After payment, display the tracking page
        tracking_window = TrackingPage(self.master, order_id)
```

Tracking details will be listed:

→ After placing an order, the user can view tracking details, such as the tracking ID, delivery address, and delivery time.

Query:

INSERT INTO Tracking (Tracking_ID, Order_ID, Customer_Address, Delivery_Time)

VALUES (11, 11, 'Hostel H1 IIIT delhi', '3 days');

Code:

```
class TrackingPage(tk.Toplevel):
    def _init_(self, master, order_id):
        tk.Toplevel._init_(self, master)
        self.title("Tracking Details")
        self.geometry("400x200")

        print("Order ID:", order_id)  # Debugging statement

        # Retrieve tracking details from the database based on order_id
        cursor.execute("SELECT * FROM Tracking WHERE Order_ID = ?",
(order_id,))
        tracking_details = cursor.fetchone()

        print("Tracking Details:", tracking_details)  # Debugging
statement

        # Display tracking details
        if tracking_details:
            ttk.Label(self, text="Tracking ID:
{}".format(tracking_details[0])).pack()
```

```
            ttk.Label(self, text="Order ID:
{}".format(tracking_details[1])).pack()
            ttk.Label(self, text="Customer Address:
{}".format(tracking_details[2])).pack()
            ttk.Label(self, text="Delivery Time:
{}".format(tracking_details[3])).pack()
        else:
            ttk.Label(self, text="Tracking details not found.").pack()
```

Quantity gets decreased when an item from inventory is added to cart:
→ When a user adds an item to the cart, the quantity of that item in the inventory needs to be decreased.

Query:
UPDATE Inventory
SET Quantity_Left = Quantity_Left - 2
WHERE Product_Name = 'Nike_Shoes';

Code:
```
def display_cart(cart_text):
    cart_text.delete(1.0, tk.END)
    if cart_items:
        cart_text.insert(tk.END, "Cart Details:\n")
        for item in cart_items:
            cart_text.insert(tk.END, "Product ID: {}, Product Name: {},
Quantity: {}, Total Cost: {}\n".format(item['Product_ID'],
item['Product_Name'], item['Quantity'], item['Total_Cost']))
        total_cart_price = sum(item['Total_Cost'] for item in cart_items)
        cart_text.insert(tk.END, "Total Cart Price:
{}\n".format(total_cart_price))
        cart_text.insert(tk.END, "Items added to cart successfully!")
    else:
        cart_text.insert(tk.END, "Cart is empty.")
```

Conflicting Transactions :

Simultaneous login attempts :
→ In a system where multiple users can attempt to log in simultaneously, concurrency issues may arise.
→ For example, if two users try to log in with the same account credentials at the same time, there might be a race condition where one login attempt succeeds while the other fails.

Query :
UPDATE Customer_Account SET Login_Status = 'Success' WHERE Username = 'shared_username' AND Password = 'shared_password';

UPDATE Customer_Account SET Login_Status = 'Success' WHERE Username = 'shared_username' AND Password = 'shared_password';

Code:

```
login_failure_count = 0
def authenticate_user(username, password):
   global login_failure_count
   cursor.execute("SELECT * FROM Customer_Account WHERE Username = ? AND Password =
?", (username, password))
   user = cursor.fetchone()
   if user:
       login_failure_count = 0  # Reset login failure count on successful login
       return True
   else:
       # Introduce a delay to simulate concurrency
       time.sleep(2)
       login_failure_count += 1  # Increment login failure count
       if login_failure_count >= 3:
           tkMessageBox.showerror("Error", "Too many login failures. Your account has
been blocked.")
           # You can implement further actions here like blocking the account in the
database
       else:
           tkMessageBox.showerror("Error", "Invalid username or password. Login
failures: {}".format(login_failure_count))
       return False
```

Simultaneous Updates to Item Description:
→ When multiple users simultaneously add items to their carts in an online shopping system, concurrency issues may arise.
→ For example, if two users try to add the same product to their carts at the same time, there might be a race condition where one user's cart update succeeds while the other's fails.

Query:
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 1001, 1);
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (2, 1001, 1);

Code:

```
# Define a global variable to store cart items
cart_items = []
cart_lock = threading.Lock()  # Lock to control access to cart_items

# Function to add items to the cart
def add_to_cart(product_id, quantity):
   global cart_items
   # Fetch product details from the database based on product_id
   cursor.execute("SELECT * FROM Product_Page WHERE Product_ID = ?", (product_id,))
   product = cursor.fetchone()
   if product:
       # Calculate total cost for the item
       total_cost = product[2] * quantity
       # Add item to cart
       with cart_lock:
           # Introduce a delay to simulate a race condition
           time.sleep(1)
           cart_items.append({
               "Product_ID": product_id,
               "Product_Name": product[1],
               "Quantity": quantity,
               "Total_Cost": total_cost
           })
           tkMessageBox.showinfo("Success", "Added {} {} to cart.".format(quantity,
product[1]))
   else:
       tkMessageBox.showerror("Error", "Product not found.")
```

User Guide for the Database:

User Authentication:
→ Users can log in using their username and password stored in the Customer_Account table.
Viewing and Editing Profile Information:
→ Once logged in, users can view and edit their profile information such as name, phone, email, and address stored in the Customers table.

Registration/Login:
→ New users need to register with a unique username and password.
→ Existing users can log in using their credentials.

Product Categories:
→ Users can explore different product categories available in the Categories table.
→ Each category has a unique ID and name.

Product Listings:
→ The Product_Page table contains information about various products, including their ID, name, price, and category ID.
→ Users can view the products available for purchase.

Browse Products:
→ Users can explore different product categories and view available products.

Inventory Management:
→ The Inventory table tracks the quantity of each product available in stock.
→ Users can see how many items are left in stock for each product.

Adding Items to Cart:
→ Users can add items to their cart using the Cart_Item table.
→ Each item added to the cart is associated with a specific order ID, product ID, quantity, and total cost.

Add to Cart:
→ After selecting desired items, users can add them to the cart.

Proceed to Checkout:
→ Users can review their cart, make any necessary adjustments, and proceed to checkout.

Making Payments:
→ Payments for orders are recorded in the Payment table.
→ Users can see the amount paid for each order.

Order Tracking:
→ The Tracking table allows users to track their orders.
→ It includes information such as order ID, customer address, and expected delivery time.

Track Order:
→ After payment, users can track the status of their orders using the provided tracking information.
