DEADLINE 6

Conflicting Transactions:-

Conflicting transactions in DBMS occur when concurrent processes attempt to modify the same data in incompatible ways, necessitating concurrency control mechanisms for resolution.

Here are the two Conflicting transactions:-

- **first conflicting transaction**

Two users (5 and 6) simultaneously try to order the same product (7). This creates a conflict as both transactions attempt to modify the same data (order for product 7), potentially leading to inconsistency or deadlock.

```
-- Start Transaction
START TRANSACTION;

-- Authentication for User 5
SELECT * FROM users WHERE User_ID = 5 AND RationCard_Number =
'ration_card_number_of_user_5';

-- Add product 7 to cart for User 5
INSERT INTO cart (user_id, items, product_id, price, supplier_id) VALUES
(5, 'Product 7', 7, (SELECT price FROM products WHERE Product_ID = 7), (SELECT supplier_id FROM
products WHERE Product_ID = 7)); -- Assuming user ID 5 and product ID 7

-- Place order from cart for User 5
INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price) VALUES
(5, 7, NOW(), (SELECT supplier_id FROM products WHERE Product_ID = 7), (SELECT price FROM
products WHERE Product_ID = 7)); -- Assuming user ID 5 and product ID 7

-- Commit Transaction
COMMIT;

-- Start Transaction
START TRANSACTION;

-- Authentication for User 6
SELECT * FROM users WHERE User_ID = 6 AND RationCard_Number =
'ration_card_number_of_user_6';

-- Add product 7 to cart for User 6
INSERT INTO cart (user_id, items, product_id, price, supplier_id) VALUES
(6, 'Product 7', 7, (SELECT price FROM products WHERE Product_ID = 7), (SELECT supplier_id FROM
products WHERE Product_ID = 7)); -- Assuming user ID 6 and product ID 7

-- Place order from cart for User 6
INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price) VALUES
```

(6, 7, NOW(), (SELECT supplier_id FROM products WHERE Product_ID = 7), (SELECT price FROM products WHERE Product_ID = 7)); -- Assuming user ID 6 and product ID 7
-- Commit Transaction
COMMIT;

- **second conflicting transaction**

Two users (7 and 8) concurrently attempt to remove and order the same product (8). This creates a conflict as both transactions compete to modify the same data (cart and order for product 8), risking inconsistency or deadlock.

```
-- Start Transaction
START TRANSACTION;

-- User 7: Authentication
SELECT * FROM users WHERE User_ID = 7 AND RationCard_Number =
'ration_card_number_of_user_7';

-- User 7: Add product 8 to cart
INSERT INTO cart (user_id, items, product_id, price, supplier_id)
SELECT 7, 'Product Name', 8, price, supplier_id
FROM products
WHERE Product_ID = 8;

-- User 7: Place order from cart
INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price)
SELECT 7, 8, NOW(), supplier_id, price
FROM products
WHERE Product_ID = 8;

-- User 8: Authentication
SELECT * FROM users WHERE User_ID = 8 AND RationCard_Number =
'ration_card_number_of_user_8';

-- User 8: Remove product 8 from cart
DELETE FROM cart WHERE user_id = 8 AND product_id = 8;

-- User 8: Place order from cart
INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price)
SELECT 8, 8, NOW(), supplier_id, price
FROM products
WHERE Product_ID = 8;

-- Commit Transaction
COMMIT;
```

Non-conflicting transactions:-

Non-conflicting transactions in DBMS involve concurrent processes modifying separate data elements without interfering with each other, ensuring data consistency and integrity.

Here are 4 Non-conflicting transactions:-

- **First Non-conflicting transactions(T1)**

  it checks product list

  ```
  -- Start Transaction
  START TRANSACTION;

  -- Authentication
  SELECT * FROM users WHERE User_ID = 1 AND RationCard_Number =
  'ration_card_number_of_user_1';

  -- View Product Directory
  SELECT Product_ID, PRODUCT_NAME, Description, Price, Stock_Quantity, Availability
  FROM products WHERE Availability = 1;

  -- Commit Transaction
  COMMIT;
  ```

- **Second Non-conflicting transactions (T2 User 2 logs in, adds a product to the cart, places an order, and logs out)**

  ```
  -- Start Transaction
  START TRANSACTION;

  -- Authentication
  SELECT * FROM users WHERE User_ID = 2 AND RationCard_Number =
  'ration_card_number_of_user_2';

  -- Add product to cart
  INSERT INTO cart (user_id, items, product_id, price, supplier_id)
  SELECT 2, 'Product Name', 7, price, supplier_id
  FROM products
  WHERE Product_ID = 7;

  -- Place order from cart
  INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price)
  SELECT 2, 7, NOW(), supplier_id, price
  FROM products
  WHERE Product_ID = 7;

  -- Remove entry from cart
  DELETE FROM cart WHERE user_id = 2 AND product_id = 7;
  ```

```
-- Update product quantity in products table
UPDATE products SET Stock_Quantity = Stock_Quantity - 1 WHERE Product_ID = 7;

-- Commit Transaction
COMMIT;
```

- **Transaction 3: User 3 logs in, views order history, and logs out**

```
-- Start Transaction
START TRANSACTION;

-- Authentication
SELECT * FROM users WHERE User_ID = 3 AND RationCard_Number =
'ration_card_number_of_user_3';

-- View Order History
SELECT oh.ORDER_ID, pd.PRODUCT_NAME, oh.ORDER_PRICE, oh.ORDER_DATE
FROM orders oh
JOIN products pd ON oh.Product_ID = pd.Product_ID
WHERE oh.User_ID = 3;

-- Commit Transaction
COMMIT;
```

- **Transaction 4: User 4 logs in, adds multiple products to the cart, removes one product, places an order, and logs out**

```
-- Start Transaction
START TRANSACTION;

-- Authentication
SELECT * FROM users WHERE User_ID = 4 AND RationCard_Number =
'ration_card_number_of_user_4';

-- Add products to cart
INSERT INTO cart (user_id, items, product_id, price, supplier_id) VALUES
(4, 'Product 7', 7, (SELECT price FROM products WHERE Product_ID = 7), (SELECT supplier_id FROM
products WHERE Product_ID = 7)), -- Assuming user ID 4 and product ID 7
(4, 'Product 8', 8, (SELECT price FROM products WHERE Product_ID = 8), (SELECT supplier_id FROM
products WHERE Product_ID = 8)); -- Assuming user ID 4 and product ID 8

-- Remove product from cart
DELETE FROM cart WHERE user_id = 4 AND product_id = 8; -- Assuming user ID 4 and product ID 8

-- Place order from cart
INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price) VALUES
(4, 7, NOW(), (SELECT supplier_id FROM products WHERE Product_ID = 7), (SELECT price FROM
products WHERE Product_ID = 7)); -- Assuming user ID 4 and product ID 7

-- Commit Transaction
COMMIT;
```

A user interface to showcase the working of your project:-

```
from decimal import Decimal
import tkinter as tk
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import pymysql.cursors

attempts = 0

# Establish connection to MySQL database
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='abcd@1234',
    database='aditya',
    cursorclass=pymysql.cursors.DictCursor
)

def open_login_window(user_type):
    login_window = Toplevel(root)
    login_window.title("Login")

    if user_type == "admin":
        # Create username and password fields for admin login
        username_label = tk.Label(login_window, text="Admin ID:")
        password_label = tk.Label(login_window, text="Admin Password:")
    else:
        # Create fields for user login
        username_label = tk.Label(login_window, text="Username:")
        password_label = tk.Label(login_window, text="Ration Card Number:")

    # Place username and password fields on the window
    username_label.grid(row=0, column=0, padx=5, pady=10, sticky=tk.E)
    username_entry = tk.Entry(login_window)
    username_entry.grid(row=0, column=1, padx=5, pady=10, sticky=tk.W+E+N+S)

    password_label.grid(row=1, column=0, padx=5, pady=10, sticky=tk.E)
    password_entry = tk.Entry(login_window, show="*")
    password_entry.grid(row=1, column=1, padx=5, pady=10, sticky=tk.W+E+N+S)

    # Create authenticate button
    if user_type == "admin":
```

```
        authenticate_button = tk.Button(login_window, text="Login", command=lambda:
authenticate_admin(username_entry.get(), password_entry.get()))
    elif user_type == "user":
        authenticate_button = tk.Button(login_window, text="Login", command=lambda:
authenticate_user(username_entry.get(), password_entry.get()))
    authenticate_button.grid(row=2, column=0, columnspan=2, padx=5, pady=10, sticky=tk.W+E)  #
Center horizontally


def authenticate_admin(admin_id, password):
    try:
        with connection.cursor() as cursor:
            # Fetch admin credentials from the database
            sql = "SELECT * FROM admin WHERE ID = %s AND password = %s"
            cursor.execute(sql, (admin_id, password))
            result = cursor.fetchone()

        if result:
            open_admin_window()
        else:
            messagebox.showerror("Login Failed", "Incorrect admin ID or password.")
    except Exception as e:
        messagebox.showerror("Error", str(e))

def open_admin_window():
    admin_window = Toplevel(root)
    admin_window.title("Admin Panel")

    # Create a label for admin options
    admin_label = tk.Label(admin_window, text="Admin Options")
    admin_label.pack(pady=10)

    # Create buttons for admin options
    view_supplier_id_button = tk.Button(admin_window, text="View Suppliers",
command=view_supplier_id)
    view_supplier_id_button.pack(pady=5)
    view_user_id_button = tk.Button(admin_window, text="View Users", command=view_user_id)
    view_user_id_button.pack(pady=5)

def open_user_options_window(user_id, user_name):
    user_options_window = Toplevel(root)
    user_options_window.title("User Options")

    # Display user's name
    welcome_label = tk.Label(user_options_window, text="Welcome, " + user_name)
    welcome_label.pack()

    # Create buttons for user options
```

```
    check_cart_button = tk.Button(user_options_window, text="Check Cart", command=lambda:
view_cart_items(user_id))
    check_cart_button.pack(pady=10)
    view_order_history_button = tk.Button(user_options_window, text="View Order History",
command=lambda: view_order_history(user_id))
    view_order_history_button.pack(pady=10)
    view_product_directory_button = tk.Button(user_options_window, text="View Product Directory",
command=view_product_directory)
    view_product_directory_button.pack(pady=10)
    order_item_button = tk.Button(user_options_window, text="Order Item", command=lambda:
order_item(user_id))
    order_item_button.pack(pady=10)

def view_supplier_id():
    try:
        # Query the database to retrieve supplier IDs, first names, and addresses
        with connection.cursor() as cursor:
            sql = "SELECT Supplier_ID, First_Name, Address FROM suppliers"
            cursor.execute(sql)
            results = cursor.fetchall()

        if results:
            # Display supplier IDs, first names, and addresses
            supplier_info = "Supplier IDs, Names, and Addresses:\n\n"
            for result in results:
                supplier_info += f"Supplier ID: {result['Supplier_ID']}\nFirst Name:
{result['First_Name']}\nAddress: {result['Address']}\n\n"
            messagebox.showinfo("Supplier IDs, Names, and Addresses", supplier_info)
        else:
            messagebox.showinfo("Supplier IDs", "No suppliers found.")

    except Exception as e:
        # If there's an error with the database connection or query
        messagebox.showerror("Error", str(e))


def view_user_id():
    try:
        # Query the database to retrieve user IDs, names, and order history
        with connection.cursor() as cursor:
            sql = """
                SELECT u.User_ID, u.Name, COUNT(o.ORDER_ID) AS Order_Count
                FROM users u
                LEFT JOIN orders o ON u.User_ID = o.User_ID
                GROUP BY u.User_ID
            """
            cursor.execute(sql)
            results = cursor.fetchall()
```

```
        if results:
            # Display user IDs, names, and order history
            user_info = "User IDs and Order History:\n\n"
            for result in results:
                user_info += f"User ID: {result['User_ID']}\nName: {result['Name']}\nOrder History:
{result['Order_Count']} orders\n\n"
            messagebox.showinfo("User IDs and Order History", user_info)
        else:
            messagebox.showinfo("User IDs", "No users found.")

    except Exception as e:
        # If there's an error with the database connection or query
        messagebox.showerror("Error", str(e))



def authenticate_user(username, password):
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM users WHERE User_name=%s AND RationCard_Number=%s"
            cursor.execute(sql, (username, password))
            result = cursor.fetchone()

        if result:
            welcome_message = "Welcome, " + result['Name']
            messagebox.showinfo("Welcome", welcome_message)
            open_user_options_window(result['User_ID'], result['Name'])
        else:
            messagebox.showerror("Login Failed", "Incorrect username or password.")

    except Exception as e:
        messagebox.showerror("Error", str(e))


def open_user_options_window(user_id, user_name):
    user_options_window = Toplevel(root)
    user_options_window.title("User Options")

    # Display user's name
    welcome_label = tk.Label(user_options_window, text="Welcome, " + user_name)
    welcome_label.pack()

    # Create buttons for user options
    check_cart_button = tk.Button(user_options_window, text="Check Cart", command=lambda:
view_cart_items(user_id))
    check_cart_button.pack(pady=10)
    view_order_history_button = tk.Button(user_options_window, text="View Order History",
command=lambda: view_order_history(user_id))
    view_order_history_button.pack(pady=10)
```

```
    view_product_directory_button = tk.Button(user_options_window, text="View Product Directory",
command=view_product_directory)
    view_product_directory_button.pack(pady=10)
    order_item_button = tk.Button(user_options_window, text="Order Item", command=lambda:
order_item(user_id))
    order_item_button.pack(pady=10)


def view_product_directory():
    try:
        # Query the database to retrieve the product directory
        with connection.cursor() as cursor:
            sql = "SELECT Product_ID, PRODUCT_NAME, Description, Price, Stock_Quantity, Availability
FROM products WHERE Availability=1"
            cursor.execute(sql)
            results = cursor.fetchall()

        if results:
            # Display the product directory
            products = "Product Directory:\n\n"
            for result in results:
                products += f"Product ID: {result['Product_ID']}\nProduct Name:
{result['PRODUCT_NAME']}\nDescription: {result['Description']}\nPrice: {result['Price']}\nStock
Quantity: {result['Stock_Quantity']}\nAvailability: {'Available' if result['Availability'] else 'Not
Available'}\n\n"
            messagebox.showinfo("Product Directory", products)
        else:
            messagebox.showinfo("Product Directory", "No products found.")

    except Exception as e:
        # If there's an error with the database connection or query
        messagebox.showerror("Error", str(e))


def order_item(user_id):
    order_window = Toplevel(root)
    order_window.title("Order Item")

    # Create labels and entry fields for product ID and quantity
    product_id_label = Label(order_window, text="Product ID:")
    product_id_label.pack()
    product_id_entry = Entry(order_window)
    product_id_entry.pack()

    quantity_label = Label(order_window, text="Quantity:")
    quantity_label.pack()
    quantity_entry = Entry(order_window)
    quantity_entry.pack()
```

```
    def add_to_cart_click():
        product_id = int(product_id_entry.get())
        quantity = int(quantity_entry.get())

        try:
            # Query the database to retrieve the product details based on the product ID
            with connection.cursor() as cursor:
                sql = "SELECT PRODUCT_NAME, Stock_Quantity, Price, SUPPLIER_ID FROM products
WHERE Product_ID = %s"
                cursor.execute(sql, (product_id,))
                result = cursor.fetchone()

            if result:
                product_name = result['PRODUCT_NAME']
                stock_quantity = result['Stock_Quantity']
                price = result['Price']
                supplier_id = result['SUPPLIER_ID']

                if stock_quantity >= quantity:
                    # Add the product to the user's cart
                    with connection.cursor() as cursor:
                        cart_sql = "INSERT INTO cart (user_id, items, product_id, price, supplier_id) VALUES
(%s, %s, %s, %s, %s)"
                        item_info = f"{product_name}"
                        cursor.execute(cart_sql, (user_id, item_info, product_id, str(price * quantity),
supplier_id))

                    # Decrease the stock quantity in the products table
                    with connection.cursor() as cursor:
                        update_sql = "UPDATE products SET Stock_Quantity = Stock_Quantity - %s WHERE
Product_ID = %s"
                        cursor.execute(update_sql, (quantity, product_id))

                    connection.commit()
                    messagebox.showinfo("Item Added", f"{product_name} added to cart successfully!")
                    order_window.destroy()
                else:
                    messagebox.showwarning("Insufficient Stock", "The selected product has insufficient
stock.")
            else:
                messagebox.showwarning("Invalid Product", "The entered product ID is invalid.")

        except Exception as e:
            # If there's an error with the database connection or query
            messagebox.showerror("Error", str(e))


    # Create a button to add the item to the cart
    add_to_cart_button = Button(order_window, text="Add to Cart", command=add_to_cart_click)
```

```
    add_to_cart_button.pack()


def place_order_from_cart(user_id):
    try:
        # Query the database to retrieve the cart items of the logged-in user
        with connection.cursor() as cursor:
            sql = """
                SELECT c.product_id, c.price, c.supplier_id
                FROM cart c
                WHERE c.user_id = %s
            """
            cursor.execute(sql, (user_id,))
            cart_items = cursor.fetchall()

        if cart_items:
            # Begin a transaction
            connection.begin()

            # Insert the order details into the orders table
            with connection.cursor() as cursor:
                insert_sql = """
                    INSERT INTO orders (user_id, Product_ID, Order_Date, supplier_id, Order_Price)
                    VALUES (%s, %s, NOW(), %s, %s)
                """
                for item in cart_items:
                    # Convert price to Decimal before inserting
                    price_decimal = Decimal(item['price'])
                    cursor.execute(insert_sql, (user_id, item['product_id'], item['supplier_id'], price_decimal))

                    # Decrease the stock quantity in the products table
                    update_sql = "UPDATE products SET Stock_Quantity = Stock_Quantity - %s WHERE
Product_ID = %s"
                    cursor.execute(update_sql, (1, item['product_id']))  # Assuming each item has a quantity
of 1

            # Clear the user's cart
            with connection.cursor() as cursor:
                delete_sql = "DELETE FROM cart WHERE user_id = %s"
                cursor.execute(delete_sql, (user_id,))

            # Commit the transaction
            connection.commit()

            messagebox.showinfo("Order Placed", "Your order has been placed successfully!")
        else:
            messagebox.showinfo("Empty Cart", "Your cart is empty.")

    except Exception as e:
```

```
        # If there's an error, rollback the transaction
        connection.rollback()
        messagebox.showerror("Error", str(e))


def delete_from_cart(user_id, product_id):
    try:
        # Delete the product from the user's cart
        with connection.cursor() as cursor:
            delete_sql = "DELETE FROM cart WHERE user_id = %s AND product_id = %s"
            cursor.execute(delete_sql, (user_id, product_id))
        connection.commit()
        messagebox.showinfo("Product Removed", "Product removed from cart successfully!")
    except Exception as e:
        # If there's an error with the database connection or query
        messagebox.showerror("Error", str(e))

def view_cart_items(user_id):
    cart_window = Toplevel(root)
    cart_window.title("Cart")

    try:
        # Query the database to retrieve the cart items of the logged-in user
        with connection.cursor() as cursor:
            sql = """
                SELECT pd.PRODUCT_NAME, c.items AS Quantity, c.price AS Price, c.product_id
                FROM cart c
                JOIN products pd ON c.product_id = pd.Product_ID
                WHERE c.user_id = %s
            """
            cursor.execute(sql, (user_id,))
            results = cursor.fetchall()

        if results:
            # Display the cart items
            cart_tree = ttk.Treeview(cart_window, columns=("Product Name", "Quantity", "Price"),
show="headings")
            cart_tree.heading("Product Name", text="Product Name")
            cart_tree.heading("Quantity", text="Quantity")
            cart_tree.heading("Price", text="Price")
            cart_tree.pack()

            for result in results:
                cart_tree.insert("", tk.END, values=(result['PRODUCT_NAME'], result['Quantity'],
result['Price']))

            def place_order_click():
                # Place the order
                place_order_from_cart(user_id)
```

```
            cart_window.destroy()

        def remove_from_cart():
            if not cart_tree.selection():
                messagebox.showerror("Error", "Please select an item to remove.")
                return
            selected_item = cart_tree.selection()[0]
            product_id = cart_tree.item(selected_item, 'values')[2]  # Get the product ID from the
selected item
            print("Removing product:", product_id)
            delete_from_cart(user_id, product_id)
            connection.commit()
            print("Product removed from database.")
            cart_window.destroy()
            print("Reopening cart window...")
            view_cart_items(user_id)


        # Create a button to place the order
        place_order_button = Button(cart_window, text="Place Order", command=place_order_click)
        place_order_button.pack(pady=10)

        # Create a button to remove item from cart
        remove_from_cart_button = Button(cart_window, text="Remove from Cart",
command=remove_from_cart)
        remove_from_cart_button.pack(pady=10)

    else:
        messagebox.showinfo("Cart", "Your cart is empty.")
        cart_window.destroy()

except Exception as e:
    # If there's an error with the database connection or query
    messagebox.showerror("Error", str(e))


def view_order_history(user_id):
    order_history_window = Toplevel(root)
    order_history_window.title("Order History")

    try:
        # Query the database to retrieve the order history of the logged-in user
        with connection.cursor() as cursor:
            sql = """
                SELECT oh.ORDER_ID, pd.PRODUCT_NAME, oh.ORDER_PRICE, oh.ORDER_DATE
                FROM orders oh
                JOIN products pd ON oh.Product_ID = pd.Product_ID
                WHERE oh.User_ID = %s
            """
```

```
            cursor.execute(sql, (user_id,))
            results = cursor.fetchall()

        if results:
            # Display the order history
            order_history_tree = ttk.Treeview(order_history_window, columns=("Order ID", "Product Name",
"Price", "Order Date"), show="headings")
            order_history_tree.heading("Order ID", text="Order ID")
            order_history_tree.heading("Product Name", text="Product Name")
            order_history_tree.heading("Price", text="Price")
            order_history_tree.heading("Order Date", text="Order Date")
            order_history_tree.pack()

            for result in results:
                order_history_tree.insert("", tk.END, values=(result['ORDER_ID'], result['PRODUCT_NAME'],
result['ORDER_PRICE'], result['ORDER_DATE']))
        else:
            messagebox.showinfo("Order History", "No order history found.")

    except Exception as e:
        # If there's an error with the database connection or query
        messagebox.showerror("Error", str(e))


# Create main window
root = tk.Tk()
root.title("Rapid Ration")
root.geometry("300x200")

# Create a label for the selection
select_label = tk.Label(root, text="Select User Type")
select_label.pack(pady=10)

# Create buttons for selecting user type
admin_button = tk.Button(root, text="Admin", command=lambda: open_login_window("admin"))
admin_button.pack(pady=5)
user_button = tk.Button(root, text="User", command=lambda: open_login_window("user"))
user_button.pack(pady=5)

# Run the main event loop
root.mainloop()

# Close the database connection when the program ends
connection.close()
```