Apna Bazzzar
Make It Easier

Transactions

ADD INTO CART:-

```python
@app.route('/add', methods=['POST'])
def add():
    try:
        data_str = request.form.get("data")
        data = ast.literal_eval(data_str)
        index = data[0][0]

        id = request.form['id']
        quantity = int(request.form['quantity'])

        mycursor = mydb.cursor()
        delete=mydb.cursor()

        # Start transaction
        mycursor.execute("START TRANSACTION")

        # Acquire exclusive lock on inventory table
        mycursor.execute("SELECT * FROM inventory_table WHERE Product_Id = %s FOR UPDATE", (id,))
        inventory_data = mycursor.fetchone()
        inv_qty = inventory_data[1]

        # Check if trigger already exists
        mycursor.execute("SELECT trigger_name FROM information_schema.triggers WHERE trigger_schema = DATABASE() AND trigger_name = 'add_to_cart_trigger'")
        trigger_exists = mycursor.fetchone()

        if trigger_exists:
            # Drop trigger if it exists
            mycursor.execute("DROP TRIGGER add_to_cart_trigger")

        # Create new trigger
        mycursor.execute("""
            CREATE TRIGGER add_to_cart_trigger
            before INSERT ON items_contained_table
            FOR EACH ROW
            BEGIN
                DECLARE inv_qty INT;
                SELECT Quantity INTO inv_qty FROM inventory_table WHERE Product_Id = NEW.Product_Id;
                IF NEW.Quantity >= inv_qty THEN
                    SET NEW.Quantity = inv_qty;
                    UPDATE inventory_table SET Quantity = 0 WHERE Product_Id = NEW.Product_Id;
                ELSE
                    UPDATE inventory_table SET Quantity = inv_qty - NEW.Quantity WHERE Product_Id = NEW.Product_Id;
                END IF;
            END;
        """)

        # Insert new item into items_contained_table
        delete.execute("")
        mycursor.execute("INSERT INTO items_contained_table (Id, Product_Id, Quantity) VALUES (%s, %s, %s)", (index, id, quantity))

        # Update inventory table
        new_qty = inv_qty - quantity
        mycursor.execute("UPDATE inventory_table SET Quantity = %s WHERE Product_Id = %s", (new_qty, id))

        # Commit transaction
        mydb.commit()

        mycursor.close()

        return "Adding successful"

    except (mysql.connector.Error, ValueError, SyntaxError) as error:
        mydb.rollback()  # Rollback transaction in case of error
        mycursor.close()
        return "Error: " + str(error)
```

CONFLICT SERIALIZABLE:-

STEP	T1	T2
1	Read cart	
2	Add product A	
3	Read cart	
4	Add product B	
5	Proceed to check	
6		Read cart
7		Add product C
8		Read cart
9		Add product D
10		Proceed to check

A conflict serializable schedule could be: T1, T2. In this schedule, T1 executes– first, followed by T2. Since the transactions do not conflict with each other, the order in which they are executed does not affect the final result.

NON CONFLICT SERIALIZABLE:-

STEP	T1	T2
1	Read cart	
2	Add product A	
3		Read cart(A)
4		Add product B
5	Read cart(AB)	
6		Read cart(AB)
7	Add product C	
8		Read cart(ABC)
9	Add product D	
10	Proceed to check	
10		Proceed to check

In this schedule, there are conflicts between the two transactions, so it is non-conflict serializable. Hence T1 and T2 are executing concurrently, but they are not accessing the same data items or trying to perform conflicting operations, so there is no need for any locking or serialization.

CHECKOUT:-

CONFLICT SERIALIZABLE:-

STEP	T1	T2
1	Read cart (R: items_contained_table, inventory_table)	
2	Add product A (W: items_contained_table)	
3		Read cart (R: items_contained_table, inventory_table)
4		Add product B (W: items_contained_table)
5	Read cart (R: items_contained_table, inventory_table)	
6		Add product C (W: items_contained_table)
7	Add product B (W: items_contained_table)	
8		Read cart (R: items_contained_table, inventory_table)
10	Proceed to check	
10		Proceed to check

NON-CONFLICT SERIALIZABLE:-

STEP	T1	T2
1	Read cart (R: items_contained_table, inventory_table)	
2	Add product A (W: items_contained_table)	
3	Read cart (R: items_contained_table, inventory_table)	
4	Add product B (W: items_contained_table)	
5	Proceed to check	
6		Read cart (R: items_contained_table, inventory_table)
7		Add product C (W: items_contained_table)
8		Read cart (R: items_contained_table, inventory_table)
9		Add product D (W: items_contained_table)
10		Proceed to check