CONFLICTING TRANSACTIONS

1. Consider the case where a particular store adds products to stock (i.e., increases the quantity), and simultaneously, a customer buys the same product (i.e. decreases the quantity).

   T1: Store with Store_ID=1 increases the quantity of product with Prod_ID=3 by 3

   P -> add x to Total Quantity in Product table for Prod_ID=3;

   T2: Customer with Cust_ID=1002 buys product with Prod_ID=3 and reduces the quantity by 4

   AP2 -> add entry in added_product table for cart of Cust_ID = 1002

T1	T2
R1(Quantity)	R2(Quantity)
Quantity=Quantity+x	Quantity=Quantity-y
W1(Quantity)	W2(Quantity)
W1(P)	W2(AP2)
COMMIT	COMMIT

Conflicting Serializable Schedule

T1	T2
R1(Quantity)	
Quantity=Quantity+x	
W1(Quantity)	
	R2(Quantity)
	Quantity=Quantity-y
	W2(Quantity)

W1(P)	
	W2(AP2)

Conflicting Non Serializable Schedule

T1	T2
R1(Quantity)	
	R2(Quantity)
Quantity=Quantity+x	
W1(Quantity)	
	Quantity=Quantity-y
	W2(Quantity)
W1(P)	
	W2(AP2)

Conflicting Serializable Schedule With Locks

T1	T2
Lock-X(Quantity)	
R1(Quantity)	
Quantity=Quantity+x	
W1(Quantity)	
Unlock-X(Quantity)	
	Lock-X(Quantity)
	R2(Quantity)
	Quantity=Quantity-y
	W2(Quantity)

	Unlock-X(Quantity)
Lock-X(P)	
W1(P)	
Unlock-X(P)	
	Lock-X(AP2)
	W2(AP2)
	Unlock-X(AP2)

```sql
-- Transaction 1:

START TRANSACTION;

SELECT Quantity INTO @current_Quantity FROM 'Availability' WHERE Store_ID=1 AND
Prod_ID=3;

UPDATE 'Product' SET Total_Quantity=(SELECT Total_Quantity FROM 'Product' WHERE
Prod_ID=3) + 3 WHERE Prod_ID=3;

UPDATE 'Availability' SET Quantity=@current_Quantity + 3 WHERE Store_ID=1 AND
Prod_ID=3;

COMMIT;

--Transaction 2:

START TRANSACTION;

SELECT Quantity INTO @current_Quantity FROM 'added_products' WHERE Prod_ID=3;

UPDATE 'Product' SET Total_Quantity=(SELECT Total_Quantity FROM 'Product' WHERE
Prod_ID=3) - 4 WHERE Prod_ID=3;

UPDATE 'added_products' SET Quantity=4 WHERE Prod_ID=3 AND Cart_ID=(SELECT
Current_Cart) FROM 'Customer' WHERE Cust_ID=1002;

INSERT INTO cart (SELECT Current_Cart FROM 'Customer' WHERE Cust_ID=1002,1002,NOW());

COMMIT;
```

2. **Two customers try to buy the same item at same time**

T1 -> Customer with Cust_ID = 1001 wants to buy '2' quantity of product with Prod_ID = 13 (R(Qty), Qty = Qty - Q1, W(Qty), W(AP1))

T2 -> Customer with Cust_ID = 1002 wants to buy '5' quantity of product with Prod_ID = 13 (R(Qty), Qty = Qty - Q2, W(Qty), W(AP2))

Qty -> quantity of P1 as in table
AP1 -> write Q1 to added_product table for cart of Cust_ID = 1001
AP2 -> write Q2 to added_product table for cart of Cust_ID = 1002

T1	**T2**
R1(Qty)	R2(Qty)
Qty=Qty-Q1	Qty=Qty-Q2
W1(Qty)	W2(Qty)
W1(AP1)	W2(AP2)
COMMIT	COMMIT

Conflicting Serializable Schedule:

T1	**T2**
R(Qty)	
Qty = Qty - Q1	
W(Qty)	
	R(Qty)
	Qty = Qty - Q2
	W(Qty)
W(AP1)	

	W(AP2)

The above Schedule is non-conflicting because before T2 reads the value of ‘Qty’ it is already updated by T1, there is no dirty read and no conflict arises here

Conflicting non-serializable Schedule:

T1	T2
R(Qty)	
	R(Qty)
Qty = Qty - Q1	
W(Qty)	
	Qty = Qty - Q2
	W(Qty)
W(AP1)	
	W(AP2)

The above schedule can’t be serialized because there are no non-conflicting operations whose order can be changed to transform this into a serializable schedule, this is because T1 writes to Qty after T2 has read Qty already and then T2 writes Qty after it.

Conflicting Serializable Schedule With Locks

T1	T2
Lock-X(Qty)	
R(Qty)	
Qty = Qty - Q1	
W(Qty)	
Unlock-X(Qty)	
	Lock-X(Qty)
	R(Qty)

	Qty = Qty - Q2
	W(Qty)
	Unlock-X(Qty)
Lock-X(AP1)	
W(AP1)	
Unlock-X(AP1)	
	Lock-X(AP2)
	W(AP2)
	Unlock-X(AP2)

```sql
--Transaction 1:

START TRANSACTION;
SELECT Quantity INTO @current_quantity FROM 'Availability' WHERE Prod_ID = 13

UPDATE 'Availability' SET Quantity = @current_quantity - 2 WHERE Prod_ID = 13

INSERT INTO 'added_products' (Prod_ID, Quantity, Cart_ID) VALUES (13, 2, SELECT
Current_Cart FROM 'Customer' WHERE Cust_ID = 1001)
COMMIT;

--Transaction 2 :

START TRANSACTION;
SELECT Quantity INTO @current_quantity FROM 'Availability' WHERE Prod_ID = 13

UPDATE 'Availability' SET Quantity = @current_quantity - 5 WHERE Prod_ID = 13

INSERT INTO 'added_products' (Prod_ID, Quantity, Cart_ID) VALUES (13, 2, SELECT
Current_Cart FROM 'Customer' WHERE Cust_ID = 1002)
COMMIT;
```

NON-CONFLICTING TRANSACTIONS

Example 1:
T1 : Change the password of customer with customer ID = 1001
T2: Change the password of customer with customer ID = 1002

```
START TRANSACTION;
password1 = input("Enter new password: ")
UPDATE Customer SET Password = password1 WHERE Cust_ID = 1001;
COMMIT;

START TRANSACTION;
password2 = input("Enter new password: ")
UPDATE Customer SET Password = password2 WHERE Cust_ID = 1002;
COMMIT;
```

Example 2:
T1: Remove the product with product ID = 13 from added_products table of cart of customer with customer ID = 1001
T2: Remove the product with product ID = 14 from added_products table of cart of customer with customer ID = 1001

```
START TRANSACTION;
DELETE FROM added_products WHERE Prod_ID = 13  AND Cart_ID = (SELECT Current_Cart FROM
customer where Cust_ID = 1001);
COMMIT;

START TRANSACTION;
DELETE FROM added_products WHERE Prod_ID = 13  AND Cart_ID = (SELECT Current_Cart FROM
customer where Cust_ID = 1001);
COMMIT;
```

Example 3:
T1: Sign-In of customer with ID = 1001
T2: Sign-In of customer with ID = 1002

```
START TRANSACTION;
name1 = input("Enter e-mail : ")
pwd1 = input("Enter password : ")
```

```
SELECT * FROM Customer WHERE Email = name1 AND Password = pwd1
INSERT INTO cart (Cart_ID, Cust_ID, date_time) VALUES ((SELECT Current_Cart FROM
customer where Cust_ID = 1001), 1001, NOW())
COMMIT;

START TRANSACTION;
name2 = input("Enter e-mail : ")
pwd2 = input("Enter password : ")
SELECT * FROM Customer WHERE Email = name2 AND Password = pwd2
INSERT INTO cart (Cart_ID, Cust_ID, date_time) VALUES ((SELECT Current_Cart FROM
customer where Cust_ID = 1002), 1002, NOW())
COMMIT;
```

Example 4:
T1: Store with Store_ID = 4, updating the quantity of a product
T2: Store with Store_ID = 5, updating the quantity of a product

```
START TRANSACTION;
prod_id1 = int(input("Enter Product ID: "))
addition1 = int(input("Enter Addition quantity: "))
UPDATE Availability SET Quantity = Quantity + addition1 WHERE Prod_ID = prod_id1 AND
Store_ID = 4"
COMMIT;

START TRANSACTION;
prod_id2 = int(input("Enter Product ID: "))
addition2 = int(input("Enter Addition quantity: "))
UPDATE Availability SET Quantity = Quantity + addition2 WHERE Prod_ID = prod_id2 AND
Store_ID = 5"
COMMIT;
```