# Transaction Pair II

Let's suppose that the Admin adds a particular product to the stock, which means that the product quantity increases. At the same time, a customer buys the same product whose quantity was increased, meaning that its quantity should be decreased after the successful purchase of that product.

Variable Notations used:-
Q represents the quantity (in stock) of Product 1.
q1 represents the quantity of Product 1, which the admin increased.
q2 represents the quantity of Product 1, which the customer purchased.
B represents the balance of the customer.
B' represents the new balance of the customer.
O represents the customer's order.

| Transaction-1 (T-1) | Transaction-2 (T-2) |
|---|---|
| Read(Q)<br>Q = Q + q1<br>Write(Q)<br>Commit | Read(Q)<br>Q = Q − q2<br>Write(Q)<br>Read(B)<br>Write(B')<br>Write(Order O)<br>Commit |

## A Serial Schedule of the two transactions

| Transaction-1 (T-1) | Transaction-2 (T-2) |
|---|---|
| Read(Q)<br>Q = Q + q1<br>Write(Q)<br>Commit | Read(Q)<br>Q = Q − q2<br>Write(Q)<br>Read(B)<br>Write(B')<br>Write(Order O)<br>Commit |

## Conflict Serializable Schedule

| Transaction-1 (T-1) | Transaction-2 (T-2) |
|---|---|
| Read(Q)<br>Q = Q + q1<br>Write(Q)<br>Commit | Read(Q)<br>Q = Q − q2<br>Write(Q)<br>Read(B)<br>Write(B')<br>Write(Order O)<br>Commit |

This Schedule is conflict serializable as we can easily move down the three statements of T-1, which results in a serial schedule (T-2 =>T-1).

**Non-Conflict Serializable Schedule**

| Transaction-1 (T-1) | Transaction-2 (T-2) |
|---|---|
| | Read(Q) |
| Read(Q)<br>Q = Q + q1<br>Write(Q)<br>Commit | |
| | Q = Q − q2<br>Write(Q)<br>Read(B)<br>Write(B')<br>Write(Order O)<br>Commit |

This schedule is non-conflict serializable as there exists a loop in the precedence graph between T-2 and T-1 on data item Q.
This is because T1 writes to Q before and after a read and write, respectively, which are executed by T2.

**Non-Conflict Serializable Schedule With Locks**

| Transaction-1 (T-1) | Transaction-2 (T-2) |
|---|---|
| | Lock-S(Q)<br>Read(Q)<br>Unlock(Q) |
| Lock-X(Q)<br>Read(Q)<br>Q = Q + q1<br>Write(Q)<br>Unlock(Q)<br>Commit | |
| | Lock-X(Q)<br>Q = Q − q2<br>Write(Q)<br>Unlock(Q)<br>Lock-X(B)<br>Read(B)<br>Write(B')<br>Unlock(B)<br>Lock-X(Order O)<br>Write(Order O)<br>Unlock(Order O)<br>Commit |

In this schedule, we add locks to each of the data items to prevent conflicts and dirty reads. Adding locks makes sure that no two transactions simultaneously manipulate the same data item.