DB Transactions

Schedule 1:

Customer-1 (Transaction):

1. Customer A places an order for a prescription refill.
2. The system checks the inventory for the prescribed medicine.
3. The system updates the inventory and reserves the medicine for the order.
4. The system updates the customer's order status to "in progress".

```
BEGIN TRANSACTION;

-- read inventory value
SELECT total FROM medicines_in_pharmacy WHERE medicine_name = 'item1' FOR UPDATE;

-- calculate new inventory value
UPDATE medicines_in_pharmacy  SET total = total - 1 WHERE medicine_name = 'item1';

-- commit the transaction
COMMIT;
```

Customer-2 (Transaction):

1. Customer B adds several items to their shopping cart.
2. The system checks the inventory for each item and updates the quantities accordingly.
3. Customer B removes one of the items from the shopping cart.
4. The system updates the inventory for the removed item and updates the shopping cart.

```
BEGIN TRANSACTION;

-- read inventory value
SELECT total FROM medicines_in_pharmacy  WHERE medicine_name = 'item1' FOR UPDATE;

-- calculate new inventory value
UPDATE medicines_in_pharmacy SET total = total - 1 WHERE medicine_name = 'item1';

-- commit the transaction
COMMIT;
```

Conflict Serializable:

In this schedule, both transactions are executed sequentially, but in a different order than they were originally submitted. The schedule is conflict serializable because there is no conflict between the

operations of the two transactions. Specifically, the two transactions do not access the same data items in conflicting ways.

Transaction/Step	Action	1	2	3	4
T1	Read inventory		Read inventory	Calculate new inventory value	Write new inventory value
T2	Read inventory	Read inventory	Calculate new inventory value	Write new inventory value and commit	

In this schedule, T2 waits until T1 has completed its first three steps before it starts. This ensures that there is no conflict between the write operation (step 3) in T1 and the write operation (step 4) in T2.

Non-Conflict Serializable:

In this schedule, the first two operations of each transaction are executed in the same order as in Schedule 1. However, there is a conflict between the third operation of T1 and the fourth operation of T2, as both involve updating the inventory. Therefore, the schedule is non-conflict serializable.
The non-conflict serializable schedule would result in a different final value of the inventory depending on the order of the transactions. This is indicated by the conflicting operations (4 in T1 and 4 in T2) that cannot be reordered without affecting the final result.

Transaction/Step	Action	1	2	3	4

T1	Read inventory		Read inventory	Calculate new inventory value	Write new inventory value and commit
T2	Read inventory	Read inventory	Calculate new inventory value	Write new inventory value	Commit

The conflict that occurred in the non-conflict serializable schedule is a write-write conflict. This means that both transactions T1 and T2 updated the same data item (the inventory) in conflicting ways. Specifically, T1 updated the inventory after reserving the medicine for an order, while T2 updated the inventory after removing an item from the shopping cart. As a result, the order of the two transactions in the schedule affects the final value of the inventory, leading to a potential inconsistency.

Schedule 2:

Customer-1 (Transaction):

1. Customer A places an order for a prescription refill.
2. The system checks the inventory for the prescribed medicine.
3. The system updates the inventory and reserves the medicine for the order.
4. The system calculates the total price for the order.
5. The system updates the customer's order status to "in progress".

```
BEGIN TRANSACTION;

-- read shopping cart value
SELECT SUM(quantity * price) AS total_price FROM cart WHERE patient_id = '1';

-- commit the transaction
COMMIT;
```

Customer-2 (Transaction):

1. Customer B searches for a non-prescription medicine and adds it to their shopping cart.
2. The system updates the shopping cart with the added item.
3. The system calculates the total price for the shopping cart.
4. Customer B enters a coupon code for a discount.
5. The system applies the coupon and recalculates the total price for the shopping cart.

BEGIN TRANSACTION;

-- update shopping cart value
UPDATE cart SET quantity = 2 WHERE patient_id = '1' AND medicine_name = 'item1';

-- commit the transaction
COMMIT;

Conflict Serializable:

In this schedule, both transactions are executed sequentially, but in a different order than they were originally submitted. The schedule is conflict serializable because there is no conflict between the operations of the two transactions. Specifically, the two transactions do not access the same data items in conflicting ways.

Transaction/Step	**Action**	**1**	**2**	**3**	**4**	**5**
T1	Read shopping cart, calculate total		Read Coupon code	Apply Coupon-code, calculate new total	Write new total to cart	
T2	Read shopping cart, calculate total	Read shopping cart, calculate total	Read Coupon code	Apply Coupon-code,	Write new total to	

				calculate new total	cart and commit	

In this schedule, T2 waits until T1 has completed its first three steps before it starts. This ensures that there is no conflict between the read operation (step 4) in T1 and the write operation (step 4, 5) in T2.

Non-Conflict Serializable:

Transaction/Step	Action	1	2	3	4	5
T1	Read shopping cart, calculate total		Read Coupon code	Apply Coupon-code, calculate new total	Write new total to cart and commit	
T2	Read shopping cart, calculate total	Read shopping cart, calculate total	Read Coupon code	Apply Coupon-code, calculate new total	Write new total to cart and commit	

In this schedule, the first two operations of each transaction are executed in the same order as in Schedule 1. However, there is a conflict between the fourth operation of T2 and the fifth operation of T1, as both involve updating the total price for the order or shopping cart. Therefore, the schedule is not conflict serializable.

The conflict that occurred in the non-conflict serializable schedule of my example is a write-read conflict. This means that one transaction (T2) updated a data item (the total price of the shopping cart) after another transaction (T1) had already read that data item. Specifically, T2 updated the total price of the shopping cart after applying a coupon code, while T1 had already read the original total price and

calculated the total price for the order. As a result, the order of the two transactions in the schedule affects the final value of the total price, leading to a potential inconsistency.