Non-Conflicting Transactions

1.Updating Customer Subscription

TRANSACTION T1

```
START TRANSACTION;

UPDATE customer
SET Subscription = 'Premium'
WHERE Customer_id=2;
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF;
```

TRANSACTION T2

```
START TRANSACTION

UPDATE customer
SET Subscription = ‘Premium’
WHERE Customer_id=4;
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF;
```

SCHEDULE

T1	T2
READ(customer)	
WRITE(customer)	
COMMIT	

	READ(customer)
	WRITE(customer)
	COMMIT

Explanation: The above transaction deals with updation of subscription of two different users from Basic to Premium.This is a non conflicting transaction as there is involvement of two different users and their attribute.If due to some hardware issues or any other failures the system is unable to update the value,then in order to main consistency, we use ROW_COUNT() function,if it returns 0 then the entire transaction is rolled back else the transition is committed.

2.Updating inventory's location

TRANSACTION T1

```
START TRANSACTION;

UPDATE inventory
SET location = 'Ghaziabad'
WHERE inventory_id=2;
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF;
```

TRANSACTION T2

```
START TRANSACTION

UPDATE inventory
SET location = 'Jaipur'
WHERE inventory_id=7;
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF;
```

SCHEDULE

T1	T2
READ(inventory)	
WRITE(inventory)	
COMMIT	
	READ(inventory)
	WRITE(inventory)
	COMMIT

Explanation: The above transaction deals with updation of inventory’s location.We are updating two different Inventory’s location here.If due to some hardware issues or any other failures the system is unable to update the value,then in order to main consistency, we use ROW_COUNT() function,if it returns 0 then the entire transaction is rolled back else the transition is committed.

3. Updating product and orders

TRANSACTION 1

```
START TRANSACTION;
UPDATE product
SET price = price * 1.1
WHERE category = 'Grocery';
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF
```

TRANSACTION 2

```
START TRANSACTION;
UPDATE orders
SET discount = discount * 0.9
WHERE total_price > 1000;
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;
```

```
ENDIF
```

T1	T2
READ(product)	
WRITE(product)	
COMMIT	
	READ(order)
	WRITE(order)
	COMMIT

Explanation: The above transaction deals with updating products and orders. This is a nonconflicting transaction as there is involvement of two different entites and their attributes.If due to some hardware issues or any other failures the system is unable to update the value,then in order to main consistency, we use ROW_COUNT() function,if it returns 0 then the entire transaction is rolled back else the transition is committed.

4. Inserting into customer and inventory

TRANSACTION 1

```
START TRANSACTION;
INSERT INTO customer (First_name, Second_name, Email, Password, Subscription, Address)
VALUES ('John', 'Doe', 'john@example.com', 'password', 'Premium', '123 Main St, Anytown');
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;

ENDIF
```

TRANSACTION 2

```
START TRANSACTION;
INSERT INTO inventory (vendor_id, location)
VALUES (10, 'New Location');
IF ROW_COUNT()=0 THEN
        ROLLBACK;
ELSE
        COMMIT;
```

```
ENDIF
```

T1	T2
READ(customer)	
WRITE(customer)	
COMMIT	
	READ(inventory)
	WRITE(inventory)
	COMMIT

Explanation: The above transaction deals with inserting values into customer and inventory. This is a nonconflicting transaction, as two entities are involved. If, due to some hardware issues or any other failures, the system is unable to update the value,then in order to main consistency, we use ROW_COUNT() function,if it returns 0 then the entire transaction is rolled back else the transition is committed.

Conflicting Transactions

1.Product out of stock conflict

```
TRANSACTION T1

START TRANSACTION

UPDATE product
SET quantity = quantity - 1
WHERE product_id = 12

COMMIT

TRANSACTION T2

START TRANSACTION

UPDATE product
SET quantity = quantity - 1
WHERE product_id = 12
```

COMMIT

SCHEDULE

T1	T2
READ(product)	
	READ(product)
WRITE(product)	
COMMIT	
	WRITE(product)
	COMMIT

Explanation: When multi users are using the application concurrently and a Read-Write conflict occurs.Assume there was only 1 quantity available of product id 12 . If T1 comes first to order that last quantity and updates the quantity to 0, but before it commits, T2 comes and reads the old value 1 and also tries to buy the same product. This leads to a conflict.

2. Delivery agent unavailability conflict

TRANSACTION 1

START TRANSACTION

```
UPDATE orders
SET orders.DeliveryAgent_ID = (select DeliveryAgent_ID from Delivery_Agent where
Availability = ’YES’)
WHERE order_id = 4
```

COMMIT

Assuming DeliveryAgent_ID = 13 was selected for order_id = 4

TRANSACTION 2

START TRANSACTION

```
UPDATE Delivery_Agent
SET Availability = ‘NO’
WHERE DeliveryAgent_ID = 13
```

COMMIT

T1	T2
READ(Orders)	
WRITE(Orders)	
	READ(Delivery_Agent)
	WRITE(Delivery_Agent)
COMMIT	
	COMMIT

Explanation: When the Admin and client use the application concurrently, a Read-Write conflict occurs. We have transaction one as the Admin and transaction two as the Delivery Agent. If both T1 and T2 execute simultaneously, and if T1 assigns delivery agent=13 for order_id=4 before it commits, T2 assigns the availability of delivery agent id 13 as ‘NO’ (We have assumed the delivery agent has already received another request for another order). It leads to a conflict as the T1 assigns the delivery agent to an order despite being unavailable.