Conflicting Transactions:

1)

T1:
```
BEGIN;
INSERT INTO Product (Name, Price, Quantity, Discount, Description, Image,
Category_ID, Retailer_ID)
VALUES ('Product A', 10.99, 50, 0.1, 'Product A description', 'productA.jpg', 1, 1);
COMMIT;
```

T2:
```
BEGIN;
DELETE FROM Category WHERE Category_ID = 1;
COMMIT;
```

The above transactions are conflicting as they both involve modifying the same Category with ID 1. One transaction adds a product to this category while the other transaction deletes the category entirely.

2)

T1:
```
BEGIN;
SELECT Quantity FROM Product WHERE Product_ID = product_id;
IF Quantity >= quantity_requested THEN
  UPDATE Product SET Quantity = Quantity - quantity_requested WHERE
Product_ID = product_id;
  INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (user_id,
product_id, quantity_requested);
  COMMIT;
ELSE
  ROLLBACK;
END IF;
```

T2:
```
BEGIN;
SELECT * FROM Cart WHERE Product_ID = product_id;
IF NOT EXISTS(SELECT * FROM Cart WHERE Product_ID = product_id) THEN
  DELETE FROM Product WHERE Product_ID = product_id;
  COMMIT;
ELSE
  ROLLBACK;
END IF;
```

The two transactions are conflicting as they are accessing and modifying the same table "Product" but with different actions (one is adding while the other is deleting). This

means that there could be a possibility of a conflict, leading to an inconsistent state of the database.

Non Conflicting Transactions:

Transaction 1:

```
UPDATE Product SET Price = Price * 1.1 WHERE Category_ID = 1;
UPDATE Product SET Price = Price * 1.15 WHERE Category_ID = 2;
```

This transaction updates the price of products belonging to different categories. Since these updates do not overlap and affect different rows in the Product table, this transaction is non-conflicting.

Transaction 2:

```
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (3, 12, 2);
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (4, 8, 1);
```

This transaction adds new rows to the Cart table for different users and different products. These inserts do not conflict with each other, so this transaction is non-conflicting.

Transaction 3:

```
UPDATE Retailer SET Name =’Harsh Gujral' WHERE Retailer_ID = 2;
DELETE FROM Category WHERE Retailer_ID = 2;
```

This transaction updates the name of a retailer and deletes some categories that belong to that retailer. Since the update and delete do not overlap and affect different tables, this transaction is non-conflicting.

Transaction 4:

```
UPDATE DeliveryAgent SET Availability = 0 WHERE DeliveryAgent_ID = 8;
INSERT INTO Orders (User_ID, DeliveryAgent_ID, Order_Amount, Cart_ID) VALUES (7, 8, 123.45, 27);
```

This transaction updates the availability of a delivery agent and inserts a new order. These operations do not overlap, so this transaction is non-conflicting.

These transactions are non-conflicting because they do not affect the same rows or tables. Each transaction operates on different sets of rows or tables, and they can execute concurrently

without interfering with each other. Therefore, these transactions can be executed in any order without any conflict.

Schedules for 2 transactions:

T1:
```
BEGIN TRANSACTION;
UPDATE Product SET Quantity = Quantity - 5 WHERE Product_ID = 1;
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 1, 5);
COMMIT;
```

T2:
```
BEGIN TRANSACTION;
UPDATE Product SET Quantity = Quantity - 2 WHERE Product_ID = 2;
INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 2, 2);
COMMIT;
```

Conflict Serializable Schedule:

```
T1: BEGIN TRANSACTION;
T1: UPDATE Product SET Quantity = Quantity - 5 WHERE Product_ID = 1;
T1: INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 1, 5);
T1: COMMIT;
T2: BEGIN TRANSACTION;
T2: UPDATE Product SET Quantity = Quantity - 2 WHERE Product_ID = 2;
T2: INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 2, 2);
T2: COMMIT;
```

Non-Conflict Serializable Schedule:

```
T1: BEGIN TRANSACTION;
T1: UPDATE Product SET Quantity = Quantity - 5 WHERE Product_ID = 1;
T2: INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 2, 2);
T2: BEGIN TRANSACTION;
T2: UPDATE Product SET Quantity = Quantity - 2 WHERE Product_ID = 2;
T1: INSERT INTO Cart (User_ID, Product_ID, Quantity) VALUES (1, 1, 5);
T1: COMMIT;
T2: COMMIT;
```

In the given transactions, both Transaction 1 and Transaction 2 involve updating the "Product" table and inserting a record into the "Cart" table. Since both transactions access the same tables and rows, there is a possibility of a conflict, which can result in different outcomes depending on the order in which the transactions are executed.

In the conflicting serializable schedule, T1 updates Product_ID 1 and inserts a record into the Cart table with Product_ID 1, while T2 updates Product_ID 2 and inserts a record into the Cart table with Product_ID 2. Since both transactions involve updates to the same tables and rows, there is a possibility of a conflict, and the outcome depends on the order in which the transactions are executed.

On the other hand, the non-conflicting serializable schedule orders the transactions such that T1 updates the Product table first, followed by T2. Then, both transactions insert their respective records into the Cart table. Since there is no overlapping access to the same rows, there is no possibility of conflict, and the outcome of the transactions is the same regardless of their order of execution.

Therefore, the transactions are conflict serializable and non-conflict serializable depending on whether or not they involve conflicting operations on the same tables and rows.