TRANSACTIONS

DBMS Project

SwiftCart

KANISHK KUMAR MEENA
2022233

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

1. Customer Purchase

```
-- T1: Customer adds items to cart and places order
START TRANSACTION;
INSERT INTO Cart (Customer_ID, Item_List_ID, Cart_Price) VALUES ('cus0000000', 1, 50);
INSERT INTO Orders (Cart_ID, Order_Date) VALUES (1, CURDATE());
INSERT INTO Places (Order_ID, Customer_ID, Transaction_Mode) VALUES (1, 'cus0000000',
'Online');
COMMIT;

-- T2: Deduct item quantities from inventory and update customer's purchased items
START TRANSACTION;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity - 1 WHERE Item_ID =
'itm0000001' AND Supplier_ID = 'sup0000001';
INSERT INTO Is_Purchased_By (Item_ID, Customer_ID) VALUES ('itm0000001', 'cus0000000');
COMMIT;
```

Transaction T1	Transaction T2
Read(Cart, Orders, Places)	
Write(Cart, Orders, Places)	
	Read(Item_Quantity)
Commit	
	Write(Item_Quantity, Is_Purchased_By)
	Commit

Explanation:
In Transaction T1, a customer adds items to the cart, places an order, and the order details are committed. Meanwhile, in Transaction T2, the item quantities are updated in the inventory based on the purchase made by the customer. These transactions are non-conflicting as they involve different tables and operations.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

2. Supplier Restock

```
-- T1: Supplier updates item quantities
START TRANSACTION;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 50 WHERE Item_ID =
'itm0000001' AND Supplier_ID = 'sup0000001';
COMMIT;

-- T2: Update item quantity in inventory
START TRANSACTION;
UPDATE Item SET Item_Quantity = Item_Quantity + 50 WHERE Item_ID = 'itm0000001';
COMMIT;
```

Transaction T1	Transaction T2
	Read(Supplier_Phone_Number)
Read(Item_Quantity)	
Write(Item_Quantity)	
Commit	
	Write(Supplier_Phone_Number)
	Commit

Explanation:
Transaction T1 involves updating the item quantities by a supplier, committing the changes. Concurrently, Transaction T2 adds a supplier's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not affect each other's data.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

3. Customer Registration

```
-- T1: Insert new customer details
START TRANSACTION;
INSERT INTO Customer (Customer_ID, Cus_Password, First_Name, Last_Name, Email,
Address, Age)
VALUES ('C001', 'pswrd0', 'John', 'Doe', 'john@example.com', '123 Main St', 25);
COMMIT;

-- T2: Add customer's phone number
START TRANSACTION;
INSERT INTO Customer_Phone_Number (Customer_ID, Phone_Number) VALUES ('C001',
'8234567890');
COMMIT;
```

Transaction T1	Transaction T2
Read(Customer)	
Write(Customer)	
Commit	
	Read(Customer_Phone_Number)
	Write(Customer_Phone_Number)
	Commit

Explanation:
Transaction T1 inserts new customer details into the database and commits the changes. Concurrently, Transaction T2 adds the customer's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not interfere with each other's data.

TRANSACTIONS

NON-CONFLICTING TRANSACTIONS

4. Supplier Registration

```
-- T1: Insert new supplier details
START TRANSACTION;
INSERT INTO Supplier (Supplier_ID, Store_ID, Sup_Password, First_Name, Last_Name,
Email)
VALUES ('sup0000000', 'str0000000', 'pas123', 'Jane', 'Smith', 'jane@example.com');
COMMIT;

-- T2: Add supplier's phone number
START TRANSACTION;
INSERT INTO Supplier_Phone_Number (Supplier_ID, Phone_Number) VALUES ('sup0000000',
'9777777710');
COMMIT;
```

Transaction T1	Transaction T2
Read(Supplier)	
Write(Supplier)	
Commit	
	Read (Supplier_Phone_Number)
	Write (Supplier_Phone_Number)
	Commit

Explanation:
Transaction T1 inserts new supplier details into the database and commits the changes. Concurrently, Transaction T2 adds the supplier's phone number to the database. These transactions are non-conflicting as they operate on different tables and do not interfere with each other's data.

TRANSACTIONS

CONFLICTING TRANSACTIONS

1. Customer Purchase Conflict

```
-- T1: Updating Cart for Customer 1
START TRANSACTION;
SELECT * FROM Cart WHERE Customer_ID = 'cus0000000' FOR UPDATE;
UPDATE Cart SET Cart_Price = 100 WHERE Customer_ID = 'cus0000000';
COMMIT;

-- T2: Updating Cart for Customer 2
START TRANSACTION;
SELECT * FROM Cart WHERE Customer_ID = 'cus0000001' FOR UPDATE;
UPDATE Cart SET Cart_Price = 150 WHERE Customer_ID = 'cus0000001';
COMMIT;
```

Transaction T1	Transaction T2
Read(Cart) {Exclusive Lock}	
	Read(Cart) {Exclusive Lock}
Write(Cart) {Exclusive Lock}	
Commit {Release Locks}	
	Write(Cart) {Exclusive Lock}
	Commit {Release Locks}

Explanation:
Transaction T1 and T2 both aim to update the cart table concurrently. T1 might be updating the cart for one customer, while T2 could be doing the same for another. However, both transactions may access and modify the same data simultaneously, potentially leading to inconsistencies or data corruption if not properly managed.
To prevent conflicts, T1 acquires an exclusive lock on the Cart table before updating it, ensuring that no other transaction can access or modify the data until T1 commits. T2, attempting to acquire its own exclusive lock, is blocked until T1 releases the lock. This mechanism ensures that only one transaction can modify the Cart table at a time, maintaining data integrity.

TRANSACTIONS

CONFLICTING TRANSACTIONS

2. Supplier Restock Conflict

```
-- T1: Restocking Items for Supplier 1
START TRANSACTION;
SELECT * FROM Item_Quantity WHERE Supplier_ID = 'sup0000001' FOR UPDATE;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 10 WHERE Supplier_ID =
'sup0000001';
COMMIT;
```

```
-- T2: Restocking Items for Supplier 2
START TRANSACTION;
SELECT * FROM Item_Quantity WHERE Supplier_ID = 'sup0000002' FOR UPDATE;
UPDATE Item_Quantity SET Item_Quantity = Item_Quantity + 20 WHERE Supplier_ID =
'sup0000002';
COMMIT;
```

Transaction T1	Transaction T2
Read(Item_Quantity) {Exclusive Lock}	
	Read (Item_Quantity) {Exclusive Lock}
Write(Item_Quantity) {Exclusive Lock}	
Commit {Release Locks}	
	Write (Item_Quantity) {Exclusive Lock}
	Commit {Release Locks}

Explanation:
Transactions T1 and T2 both involve updating the Item_Quantity table to restock items from different suppliers. However, as they operate concurrently, both transactions may attempt to modify the same item quantities simultaneously, posing a risk of data inconsistencies or incorrect updates.
To manage conflicts, T1 acquires an exclusive lock on the Item_Quantity table before proceeding with its update. This lock prevents T2 from accessing or modifying the data until T1 completes its operation. T2, waiting to acquire its exclusive lock, is blocked until T1 releases the lock. This locking mechanism ensures that only one transaction can modify the Item_Quantity table at a time, preserving data consistency.