Transactions

CONFLICTING TRANSACTION

1) T1
START TRANSACTION;
UPDATE orders SET order_total = order_total + 500 WHERE order_id = 5838;
COMMIT;
T2
START TRANSACTION;
UPDATE orders SET order_total + 50 WHERE order_id = 5838;
COMMIT;

Explanation : These two transactions can cause a conflict if they are executed concurrently, as both transactions are updating 'order_total'. To avoid conflicts the database management system may use concurrency control mechanisms such as locking to ensure that the transactions are executed in a conflict-free manner

=) To make a conflict serializable schedule, we can use locking to ensure that the transactions do not conflict with each other.

- T1:
  - LOCK orders WHERE order_id = 5838
  - UPDATE orders SET order_total = order_total + 500 WHERE order_id = 5838
  - UNLOCK orders

- T2:
  - LOCK ORDERS WHERE order_id = 5838
  - UPDATE orders SET order_total = order_total + 50 WHERE

order_id = 5838
UNLOCK orders

In this schedule, T1 & T2 are both acquiring a lock on the 'orders' table before making any updates, which ensures that only one transaction can access & update the data at a time. This ensures conflict serializability.

2) T1:
START TRANSACTION;
UPDATE prescription SET prescription_date = '2022-04-20'
WHERE Prescription_id = 80171;
COMMIT;
T2:
START TRANSACTION;
UPDATE prescription SET Prescription_date = '2022-05-24'
WHERE Prescription_id = 80171;

Explanation: Both T1 and T2 are updating the same row, so it will conflict if both transactions try to update the row simultaneously. The conflict can result in the loss of data consistency. To prevent this, we can do a serial execution where transactions are executed one after the other.

A non conflict serializable schedule:

⇒ To make this transaction non conflict serializable, we will execute T1 & T2 one after the other. This way the final state of the database is equivalent to serial execution of transactions.

3) -START TRANSACTION;
INSET INTO orders (order_total, customer_id, order_status)
VALUES (100, 1010, 'pending');
DELETE FROM product WHERE product_id = 4779;
COMMIT;

-START TRANSACTION;
INSERT INTO ORDES (order_total, customer_id, order_status)
VALUES (150, 1020 'pending').
DELETE FROM product WHERE product_id = 4779;
COMMIT;

Explanation: Both the transactions are modifying the same product record in the 'product' table which will lead to a conflicting transaction.

NON CONFICTING TRANSACTION

1) -BEGIN TRANSACTION
START TRANSACTION
INSERT INTO prescription (prescription_id, customer_id, prescription_date, prescription_time) VALUES (67129, 5952, '2022-05-26', '18:23:07');
UPDATE product SET product_quantity = 50 WHERE product_id = 7648;

Explanation: The transaction above does not conflict with other transactions as it only updating one table and adding values into another table.

Conflicting Serializable Schedule

T1 : START TRANSACTION
T2 : START TRANSACTION
T1 : INSERT INTO prescription (prescription_id, customer_id, prescription_date, prescription_time) VALUES (67129, 5952, '2022-05-26' '18:23:07')
T2 : UPDATE product SET product_quality = 50 WHERE product_id = 7648
T1 : UPDATE product SET product_quantity = 100 WHERE product_id = 7648
T2 : COMMIT
T1 : COMMIT

NON CONFLICTING Serializable schedule

T1 : START TRANSACTION
T1 : INSERT INTO prescription (prescription_id, customer_id, prescription_date,

prescription_time) VALUES (67129, 5952, '2022-05-26', '18:23:07');

T1: COMMIT

T2: START TRANSACTION

T2: UPDATE product SET product_quality = 50 WHERE product_id = 7648

T2: COMMIT

2) - BEGIN TRANSACTION
START TRANSACTION;
INSERT INTO orders (order_id, order_total, customer_id, order_status, order_channel, store_name, store_location)
VALUES ('28643', '8811', '8593', 'Un-fulfilled', 'Online', 'Null', 'NULL');
UPDATE product SET product_quantity = 6760 WHERE product_id = 8332;
COMMIT;

Explanation: The transaction is non conflicting since its only inserting a new order and updating the product. There are no conflicts between them.

3) T1
BEGIN TRANSACTION;
UPATE STOCK
SET qty = qty - 10
WHERE product_id = '8332' AND EXPIRY > GETDATE();

T2 → START TRANSACTION
UPDATE CART
SET card_qty = card_qty + 10
card_total = card_total + (10 * product_price)

WHERE customer_id = '1438' AND product_id = '8332'
COMMIT;

Explanation: The transaction is non-conflicting transaction, so it updates 2 different tables and the updates do not depend on each other.

Serializable conflicting schedule

T1: START TRANSACTION;
UPDATE STOCK SET qty = qty - 10 WHERE product_id = '8332' AND EXPIRY > GETDATE();

T2: START TRANSACTION;
UPDATE CART SET cart_qty + 10 cart_total + 10, cart_total = cart_total + (10 * product_price)
WHERE customer_id = '1438' AND product_id = '8332';

T1 COMMIT;
T2 COMMIT;

Serializable non-conflicting schedule

T1: START TRANSACTION;
UPDATE STOCK SET qty = qty - 10 WHERE product_id = '8332' AND EXPIRY > GETDATE();
COMMIT;

T2: START TRANSACTION;
UPDATE CART SET cart_qty + 10, cart_total + 10, cart_total = cart_total + (10 * product_price)
WHERE customer_id = '1438' AND product_id = '8332';
COMMIT;

Date
Page

1 In the non-conflicting schedule, both transactions acquire locks on the same data before updating it, which ensures that they do not conflict with each other.

4) START TRANSACTION;
INSERT INTO customer (customer_name, customer_age, customer_gender, customer_email, customer_address, customer_contact, customer_states, customer_password) VALUES ('9662' 'Jacab Marane' 80, 'Male' 'Jacob19@example.com' '33112 Wilford Islands' Apt. 474 \n North Aidamouth, VA '90405' 4374, 'Elite', 'fbe65012e');

INSERT INTO CART (customer_id card_products card_qty, card_total) VALUES ('3540', 'eep', 25, 2237');

UPDATE stock levels after purchase
UPATE Stock SET qty = qty - 5 WHERE Product_id IN (1271, 2379);
COMMIT;

Explanation: The transaction is non conflicting. The transaction contains 3 seperate statement: inserting into customer inserting into cart and updating stock. These statements operate on different tables and do not have any dependencies or conflicts with each other.