Conflicting and Non-Conflicting Transactions:

Schedule 1:

Consider the case where a supplier adds products to stock (i.e., increases the quantity of stock), and concurrently, a customer buys the same product (i.e., decreases the quantity of the stock). Here, Q represents the quantity (in stock) of Product 1. B2 represents the total cost of the customer in T2, while O2 represents their order.

| T1 | T2 |
| --- | --- |
| Read(Q)  Q = Q+q1  Write(Q)  Commit | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  B2 = B2+X  Write(B2)  Write(Order O2)  Commit |

Conflict-Serializable Schedule

| T1 | T2 |
| --- | --- |
| Read(Q)  Q = Q+Q1  Write(Q) | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  B2 = B2+X  Write(B2)  Write(Order O2) |

This schedule is a conflict serializable as we can repeatedly move the three statements of T1 upward, such that both transactions get executed serially(T2→T1).

We can do it better by using Locks that haven’t taught yet.

Non-Conflict Serializable Schedule

| T1 | T2 |
| --- | --- |
| Read(Q)  Q=Q+q1  Write(Q) | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  B2 = B2+X  Write(Order O2) |

The above schedule is non-conflict-serializable as there does not exist any sequence of non-conflicting switches to make the schedule serial. This is because T1 writes to Q before and after a read and write, respectively, which are executed by T2.

We can do it better by using Locks that haven’t taught yet.

Schedule = 2

Consider the case of two customers buying the same product simultaneously. Here, Q represents the quantity (in stock) of Product 1. Similarly, B1 and B2 represent the balance of customers 1 and 2, whereas their orders are represented by O1 and O2, respectively.

| T1 | T2 |
| --- | --- |
| Read(Q)  Q = Q-q1  Write(Q)  Read(B1)  Write(B1)  Commit | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  Write(B2)  Commit |

Conflict-Serializable Schedule

| T1 | T2 |
| --- | --- |
| Read(Q)  Q = Q-q1  Write(Q)  Read(B1)  Write(B1’)  Write(Order O1) | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  Read(B2’)  Write(Order O2) |

This schedule is conflict serializable since we can repeatedly switch the last command in T2 up, such that both the transactions get executed serially (T1→T2).

Non-Conflict-Serializable Schedule

| T1 | T2 |
| --- | --- |
| Read(Q)  Q = Q-q1  Write(Q)  Read(B1)  Write(B1’)  Write(Order O1) | Read(Q)  Q = Q-q2  Write(Q)  Read(B2)  Write(B2’)  Write(Order O2) |

The above schedule is non-conflict-serializable as there does not exist any sequence of non-conflicting switches to make the schedule serial. This is because T1 write to Q

before and after a read and write, respectively, which are executed by T2

Similarly there are many possible transactions:

If the same user logs in to their account in two terminals simultaneously,concurrently and add some product to his/her cart and then in the cart he/she has two option 1. Remove from the cart 2. Proceed to checkout.

Assume he choose option 1 in 1st terminal(T1) and 2nd on 2nd terminal (T2)

Many more are there…..

Schedule 3:

-- T1

START TRANSACTION;

UPDATE customercontact SET contact= 98765432 where CustomerID= 1;

COMMIT;

-- T2

START TRANSACTION;

SELECT \* FROM customer WHERE CustomerID = 1;

UPDATE customer SET city='DELHI' where CustomerID=1;

COMMIT;

-- T3

START TRANSACTION;

INSERT INTO customer(CustomerID,fname,mname,lname,customerDOB,streetNo, landmark, city, state, pincode, email) values('1','Prince','','Kumar','2004-06-13','Gali No. 30 B','Bharat Mata Mandir','Narela','Delhi','110040','prince22378@gmail,com');

COMMIT;

-- T4

START TRANSACTION;

DELETE FROM customer WHERE CustomerID = 5;

COMMIT;

Conflicting Serializable:

| T1 | T2 | T3 | T4 |
| --- | --- | --- | --- |
| Update cus….=1;  Commit; | Select \*........=1;  Update cus…=1;  Commit; | Insert into cus.(……);  Commit; | Delete form….=5;  Commit; |

Operational

| T1 | T2 | T3 | T4 |
| --- | --- | --- | --- |
| Read(Y)  Write(Y)  commit; | Read(X)  Read(X)  Write(X)  commit; | Read(X)  Write(X)  Commit; | Read(X1)  Write(X2)  commit; |

The precedence graph for this has no loops so it’s conflicting serializable.

In conflicting schedule, T1 and T2 perform operation on same data items of different tables.

Non-Conflicting Serializable:

| T1 | T2 | T3 | T4 |
| --- | --- | --- | --- |
| Update cus….=1;  commit; | Select \*........=1;  Update cus…=1;  commit; | Insert into cus.(……);  Commit; | Delete form….=5;  commit; |

Operational

| T1 | T2 | T3 | T4 |
| --- | --- | --- | --- |
| Read(Y)  Write(Y)  Commit | Read(X)  Read(X)  Write(X)  commit | Read(X)  Write(X)  Commit | Read(Z)  Write(Z)  commit |

In non-conflicting schedule, transactions T3 and T4 execute independently of each other and don’t interfere with each other’s operations. Because T3 and T4 acts on different data items of same table.