Deadline 6: Conflicting and Non-Conflicting Database Transactions

Transaction pair 1

Consider the case where a supplier adds products to stock (i.e., increases the quantity), and simultaneously, a customer buys the same product (i.e. decreases the quantity). Here, Q represents the quantity (in stock) of the product that is in interest. B1 represents the balance of the customer in T1, while O1 represents their order. B'1 is the new balance.

Conflict-Serializable Schedule: This schedule is conflict serializable as we can repeatedly move the three statements from T2 upwards, such that both the transactions get executed serially (T1 after T2).

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
READ(Q1)	
Q = Q - Q1	
WRITE(Q)	
	READ(Q)
	Q = Q + Q2
	WRITE(Q)
READ(B1)	
CHECK(B1)	
WRITE(B'1)	
WRITE(ORDER O1)	

Conflict-Serializable Schedule with Locks : We add locks before each read, and unlock after each write. This ensures that no two transactions are accessing/altering the same data point at the same time, which could lead to conflicts.

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
LOCK-X(Q)	
READ(Q)	
Q = Q - Q1	
WRITE(Q)	
UNLOCK(Q)	

	LOCK-X(Q)
	READ(Q)
	Q = Q + Q2
	WRITE(Q)
	UNLOCK(Q)
LOCK-X(B1)	
READ(B1)	
CHECK(B1)	
WRITE(B1)	
UNLOCK(B1)	
LOCK-X(ORDER O1)	
WRITE(ORDER O1)	
UNLOCK(ORDER O1)	

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

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
READ(Q1)	
	READ(Q)
	Q = Q + Q1
	WRITE(Q)
Q = Q - Q2	
WRITE(Q)	
READ(B1)	
CHECK(B1)	
WRITE(B1)	
WRITE(ORDER O1)	

Non-Conflict-Serializable Schedule with Locks : We add shared locks before and after each solitary read, and exclusive locks around a read-write pair. This makes sure that no two transactions are accessing/altering the same data point at the same time, which could lead to conflicts.

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
	Lock-S(Q)
	Read(Q)
	unlock(Q)
Lock-X(Q)	
Read(Q)	
Q = Q + Q1	
Write(Q)	
Unlock(Q)	
	Lock-X(Q)
	Q = Q – Q2
	Write(Q)
	Unlock(Q)
	Lock-X(B2)
	Read(B2)
	Check(B2)
	Write(B2)
	unlock(B2)
	Lock-X(Order O2)
	Write(Order O2)
	Unlock(Order O2)

Transaction Pair 2

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

Conflict-Serializable Schedule : First, the quantity bought by B1 is deducted from the stock quantity Q. B1's balance is also updated. Then we switch to T2 and deduct the quantity bought by B2 from Q. Now, we switch back to T1 and write the order details. Finally, we switch back to T2 and execute the remaining commands and commit the transactions. This schedule is conflict serializable since we can repeatedly switch the last command in T1 up, such that both the transactions get executed serially (T2 after T1).

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
Read(Q)	
Q = Q − Q1	
Write(Q)	
Read(B1)	
Write(B′ 1)	
	Read(Q)
	Q = Q − Q2
	Write(Q)
Write(Order O1)	
	Read(B2)
	Check(B2)
	Write(B′ 2)
	Write(Order O2)

Conflict-Serializable Schedule with Locks : We add locks before each read, and unlock after each write. This ensures that no two transactions are accessing/altering the same data point at the same time, which could lead to conflicts.

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
LOCK- X(Q)	
READ(Q)	
Q = Q - Q1	
WRITE(Q)	
UNLOCK(Q)	
LOCK - X(B1)	
READ(B1)	
WRITE(B’1)	
UNLOCK(B1)	

	LOCK - X(Q)
	READ(Q)
	Q = Q - Q2
	WRITE(Q)
	UNLOCK(Q)
LOCK - C(ORDER O1)	
WRITE(ORDER O1)	
UNLOCK(ORDER O1)	
	LOCK - X(B2)
	READ(B2)
	CHECK(B2)
	WRITE(B’2)
	UNLOCK(B2)
	LOCK - X(ORDER O2)
	WRITE(ORDER O2)
	UNLOCK(ORDER O2)

Non-Conflict-Serializable Schedule: The below 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.

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
READ(Q)	
	READ(Q)
Q = Q - Q1	
WRITE(Q)	
	Q = Q - Q2
	WRITE(Q)
	READ(B2)
	WRITE(B’2)
	WRITE(ORDER O2)

READ(B1)	
CHECK(B1)	
WRITE(B’1)	
WRITE(ORDER O1)	

Non-Conflict-Serializable Schedule with Locks: We add shared locks before and after each solitary read, and exclusive locks around a read-write pair. This makes sure that no two transactions are accessing/altering the same data point at the same time, which could lead to conflicts.

TRANSACTION- 1 (T1)	**TRANSACTION- 2 (T2)**
	LOCK-S(Q)
	READ(Q)
	UNLOCK(Q)
LOCK - S(Q)	
READ(Q)	
UNLOCK(Q)	
	LOCK- X(Q)
	Q = Q -Q2
	WRITE(Q)
	UNLOCK(Q)
LOCK - X(Q)	
Q = Q - Q1	
WRITE(Q)	
UNLOCK(Q)	
LOCK - X(B1)	
READ(B1)	
CHECK(B1)	
WRITE(B’1)	
UNLOCK(B1)	
LOCK - X(ORDER O1)	
WRITE(ORDER O1)	

UNLOCK(ORDER O1)	
	LOCK - X(B2)
	READ(B2)
	CHECK(B2)
	WRITE(B’2)
	UNLOCK(B2)
	LOCK - X(ORDER O2)
	WRITE(ORDER O2)
	UNLOCK(ORDER O2)