Chapter 18: Concurrency Control

Serial Schedule:
In a serial Schedule, a transaction only starts when the other transaction has finished execution.

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Figure 18.3: Serial schedule in which T_1 precedes T_2

T_1	T_2	A	B
		25	25
READ(A,t)			
t := t+100			
WRITE(A,t)		125	
	READ(A,s)		
	s := s*2		
	WRITE(A,s)	250	
READ(B,t)			
t := t+100			
WRITE(B,t)			125
	READ(B,s)		
	s := s*2		
	WRITE(B,s)		250

Figure 18.5: A serializable, but not serial, schedule

Conflicts:

1. Two actions of the same transaction, e.g., **ri(X); Wi(Y),** always conflict.
2. Two writes of the same database element by different transactions conflict. That is, **Wi(X); Wj(X)** is a conflict.
3. A read and a write of the same database element by different transactions also conflict. That is, **ri(X); Wj(X)** is a conflict, and so is Wi(X); rj(X).

Example 18.6: Consider the schedule

$$r_1(A);\ w_1(A);\ r_2(A);\ w_2(A);\ r_1(B);\ w_1(B);\ r_2(B);\ w_2(B);$$

Conflicting pairs:
ri(X); Wi(Y)
Wi(X); Wj(X)
ri(X); Wj(X)

Solution:

serial schedule:
Is me hum ne
transaction 1
waley pehleh
laney 2 waley
baad mein

$$r_1(A);\ w_1(A);\ r_2(A);\ \underline{w_2(A)};\ \underline{r_1(B)};\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ \underline{r_2(A)};\ \underline{r_1(B)};\ w_2(A);\ w_1(B);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ r_2(A);\ \underline{w_2(A)};\ \underline{w_1(B)};\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ \underline{r_2(A)};\ \underline{w_1(B)};\ w_2(A);\ r_2(B);\ w_2(B);$$
$$r_1(A);\ w_1(A);\ r_1(B);\ w_1(B);\ r_2(A);\ w_2(A);\ r_2(B);\ w_2(B);$$

Example 18.7: The following schedule S involves three transactions, T_1, T_2, and T_3.

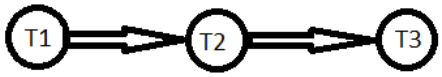
$$S:\ r_2(A);\ r_1(B);\ w_2(A);\ r_3(A);\ w_1(B);\ w_3(A);\ r_2(B);\ w_2(B);$$

Find the conflecting pairs of the same database element by comparing with remaing ones:

r2(A);w2(A) r2(A); r3(A) c.p r2(A); w3(A) c.p	r1(B); w1(B) r1(B); r2(B) r1(B);w2(B) c.p	w2(A);r3(A) c.p w2(A);w3(A) c.p	r3(A); w3(A)	w1(B); r2(B) c.p w1(B);w2(B) c.p	r2(B);w2(B)

Sort the conflecting pairs:

r2(A);r3(A) r2(A);w3(A) w2(A);r3(A) w2(A);W3(A) T2 <s T3	r1(B);w2(B) w1(B);r2(B) w1(B);w2(B) T1 <s T2



$$S:\ \underline{r_2(A)};\ \underline{r_1(B)};\ w_2(A);\ r_3(A);\ w_1(B);\ w_3(A);\ r_2(B);\ w_2(B);$$

r1(B);r2(A);w2(A);<u>r3(A);w1(B)</u>;w3(A);r2(B);w2(B)
r1(B);r2(A);<u>w2(A);w1(B)</u>;r3(A);w3(A);r2(B);w2(B)
r1(B);<u>r2(A);w1(B)</u>;w2(A);r3(A);w3(A);r2(B);w2(B)
r1(B);w1(B);r2(A);w2(A);r3(A);<u>w3(A);r2(B)</u>;w2(B)
r1(B);w1(B);r2(A);w2(A);<u>r3(A);r2(B)</u>;w3(A);w2(B)
r1(B);w1(B);r2(A);w2(A);r2(B);r3(A);<u>w3(A);w2(B)</u>
r1(B);w1(B);r2(A);w2(A);r2(B);<u>r3(A);w2(B)</u>;w3(A)
r1(B);w1(B);r2(A);w2(A);r2(B);w2(B);r3(A);w3(A)

Example 18.9: Consider the schedule

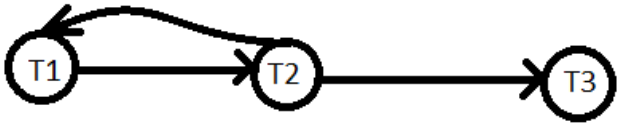
S_1: $r_2(A)$; $r_1(B)$; $w_2(A)$; $r_2(B)$; $r_3(A)$; $w_1(B)$; $w_3(A)$; $w_2(B)$;

r2(A);w2(A) r2(A);r3(A) r2(A);w3(A) c.p	r1(B);r2(B) r1(B);w1(B) r1(B);w2(B) c.p	w2(A);r3(A) c.p w2(A);w3(A) c.p	r2(B);w1(B) c.p r2(B);w2(B)
r3(A);w3(A)	w1(B);w2(B) c.p		

r2(A);w3(A)
w2(A);r3(A)
w2(A);w3(A)
T2 <s T3

r1(B);w2(B)
T1 <s T2

r2(B);w1(B)
T2 <s T1



If **there is a cycle** in the precedence graph, then the schedule is **not conflict-serializable.**

If the precedence **graph has no cycles**, then we can reorder the schedule's actions using legal swaps of adjacent actions, until the **schedule becomes a serial schedule**.

Enforcing Serializability by Locks:
Locks:
Consistency of Transactions: Actions and locks must relate in the expected ways:

- A transaction can only read or write an element if it previously was granted a lock on that element and hasn't yet released the lock.
- If a transaction locks an element, it must later unlock that element.

Legality of Schedules:

- Locks must have their intended meaning: no two transactions may have locked the same element without one having first released the lock.