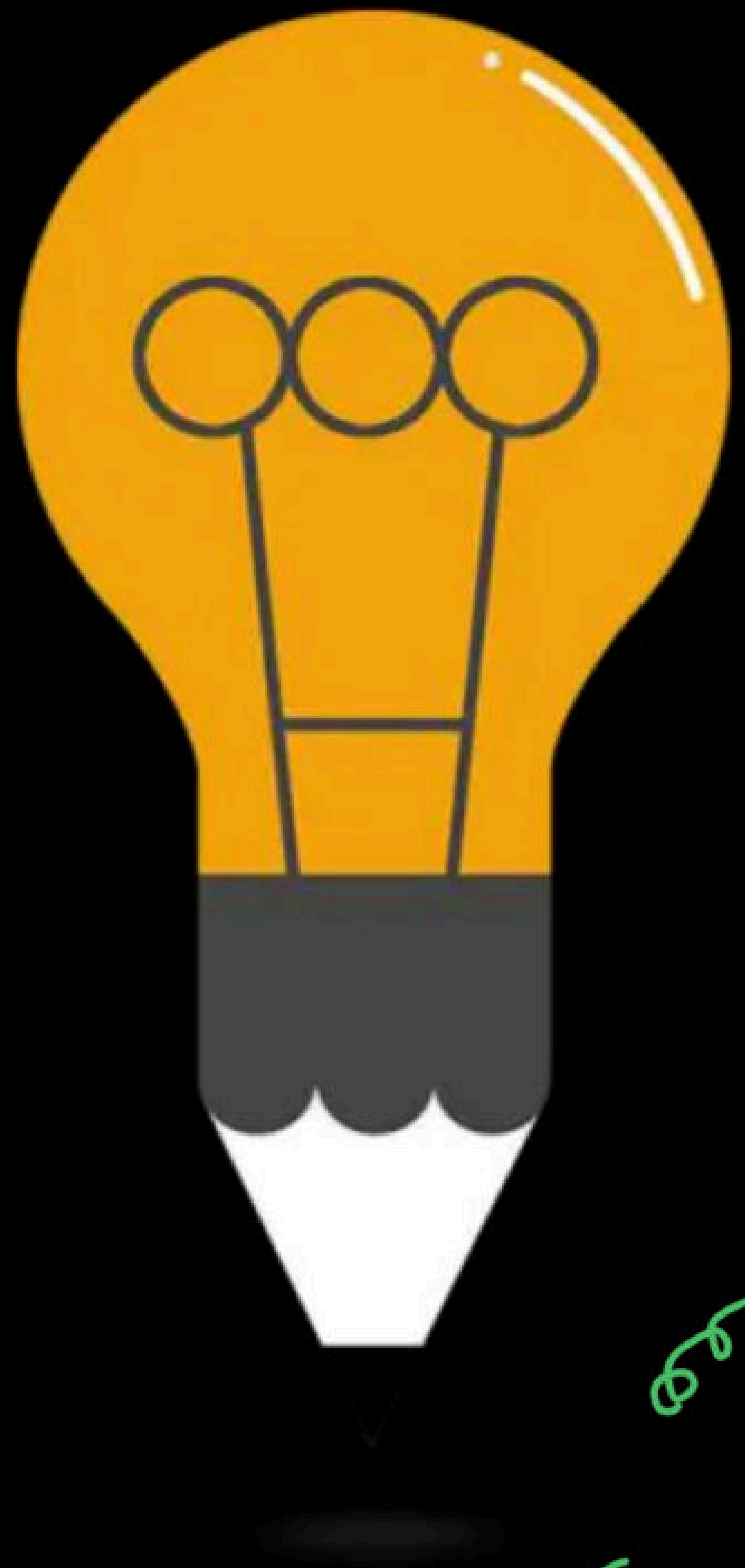
unacademy

Transaction & Concurrency Control: Part V

Complete Course on Database Management System

Vishvadeep Gothi • Lesson 27 • July 22, 2023

Growing & shrinking



DBMS

Locking Protocols & Timestamp based protocols

By: Vishvadeep Gothi

Basic 2 Phase Locking Protocol

Systematic Locking mechanism

Once unlock done, a transaction is not allowed to lock any database item.

Basic 2 Phase Locking Protocol

T1	T2
R(X)	
	W(X)
R(X)	

Basic 2 Phase Locking Protocol

T1	T2
R(X)	
	W(X)
	R(Y)
W(Y)	

Basic 2 Phase Locking Protocol

Every schedule which is allowed under basic 2PL, is conflict serializable also.

T1	T2	T3
lock(x)		
10:00 am		
lock(z)		
unlock(x)		
	lock(z)	
	10:10 am	
	lock(y)	
	10:15 am	
		lock(y) 10:20 am

T1	T2	T3
		lock(z) 10:25 am
w(z)		
10:30 am		

won't be allowed under basic 2PL

Basic 2 Phase Locking Protocol

T1	T2
lock_S(X)	
R(X)	
	lock_Ex(Y)
	W(Y)
	lock_Ex(X)
	W(X)
lock_S(Y)	
R(Y)	

blocked

blocked

not allowed in 2PL.

Deadlock

Basic 2 Phase Locking Protocol

Suffers from deadlock

Basic 2 Phase Locking Protocol

Starvation of small transactions due to large transaction

↳ indefinite waiting

T1	T2	T3	T4
lock_Ex(x)			
lock_Ex(y)			
lock_Ex(z)			
	lock_S(x)		
		lock_S(y)	
			lock_S(z)

Basic 2 Phase Locking Protocol

Starvation of large transaction due to small transactions

T1	T2	T3	T4
lock(x)			
	lock(y)		
		lock(z)	
			lock(x)
			lock(y)
			lock(z)

Strict Schedule

Every Write operation should be ended with commit before other transaction performs read or write on the same DB item.

Strict Schedule

Every Write operation should be ended with commit before other transaction performs read or write on the same DB item.

T1	T2	T3
lock_Ex(x)		
W(X)		
commit		
unlock_(x)	lock_S(x)	
	R(X)	
	unlock_S(x)	
		lock_Ex(x)
		W(X)
		commit
		unlock(x)

Strict Schedule

Every Write operation should be ended with commit before other transaction performs read or write on the same DB item.

T1	T2	T3
lock_s(x)		
R(X)		
unlock(x)	lock_Ex(x)	
	W(X)	
Commit		
	Commit	
	unlock(x)	lock_Ex(x)
		W(X)
		commit
		unlock(x)

allowed under strict 2PL

Strict Schedule

Every Write operation should be ended with commit before other transaction performs read or write on the same DB item.

T1	T2	T3	T4
R(X)			
	W(X)		
Commit			
	Commit	lock_exc(x)	
		W(X)	
			W(X)
			commit
		commit	
		Unlock(x)	

not allowed under strict 2PL

Strict 2PL

Exclusive lock should not be release until commit

Strict 2PL

Exclusive lock should be released after commit

T1
Lock_X(X)
W(X)
Commit
Unlock(X)

Strict 2PL

So Strict 2PL allows only strict schedules

T1
Lock_X(X)
W(X)
Commit
Unlock(X)

Strict 2PL

T1	T2
lock-EX(X)	
W(X)	
	lock-S(X)
	R(X)
R(Y)	
	W(Y)
Commit	
unlock(X)	
	Commit

blocked

not allowed under strict 2PL

Strict 2PL

T1	T2
lock_Ex(x)	
W(X)	
	lock_Ex(Y)
	W(Y)
blocked lock_S(Y)	
R(Y)	
	lock_Ex(x) blocked
	W(X)
unlock(x)	
	unlock(y)

not allowed under strict 2PL

Deadlock

Rigorous 2PL

Every lock should be released after commit

Rigorous 2PL

Every lock should be released after commit

deadlock

T1

Lock(X)

W(X) or R(X)

Commit

Unlock(X)

T1	T2
lock_S(x)	
R(x)	
	lock_Ex(y)
	w(y)
	lock_Ex(x) → blocked
	w(x)
blocked ← lock_S(y)	
R(y)	

Rigorous 2PL

Every lock should be released after commit

T1	T2	T3	T4
R(X)			
Commit			
	W(X)		
		R(X)	
	Commit		
		Commit	
			W(Y)
			Commit

blocked

not allowed

Rigorous 2PL

Is it allowed under Basic 2PL?

T1	T2	T3	T4
lock_s(x)			
R(X)			
Commit	lock_Ex(x)		
unlock(X)			
	W(X)	lock_S(x)	
	unlock(x)		
		R(X)	
		unlock(x)	
	Commit		
		Commit	lock_Ex(y)
			W(Y)
			Commit
			unlock(y)

allowed under Basic 2PL

Rigorous 2PL

Strict & Rigorous 2PL don’t have dirty read

Rigorous 2PL

Is it allowed under Basic 2PL?

T1	T2
W(X)	
	R(X)
failed	Commit
Commit	

basic 2PL allows non-recoverable schedules also.

Basic 2PL allows dirty read

T1	T2
W(x)	
	R(x)
failed → Commit	
	Commit

Cascaded recoverable

if T1 failed then T2 also rolled back.

2PL

- Basic 2PL does not ensure recoverability
- Basic 2PL does not ensure a cascadeless schedule
- Strict & Rigorous 2PL have only recoverable schedules (cascadeless)

- Basics 2PL, strict 2PL and Rigorous 2PL may suffer from deadlock

Conservative 2PL (Static 2PL)

Lock all the items before the Transaction begins execution by predeclaring its read-set and write-set

no deadlock

shared lock

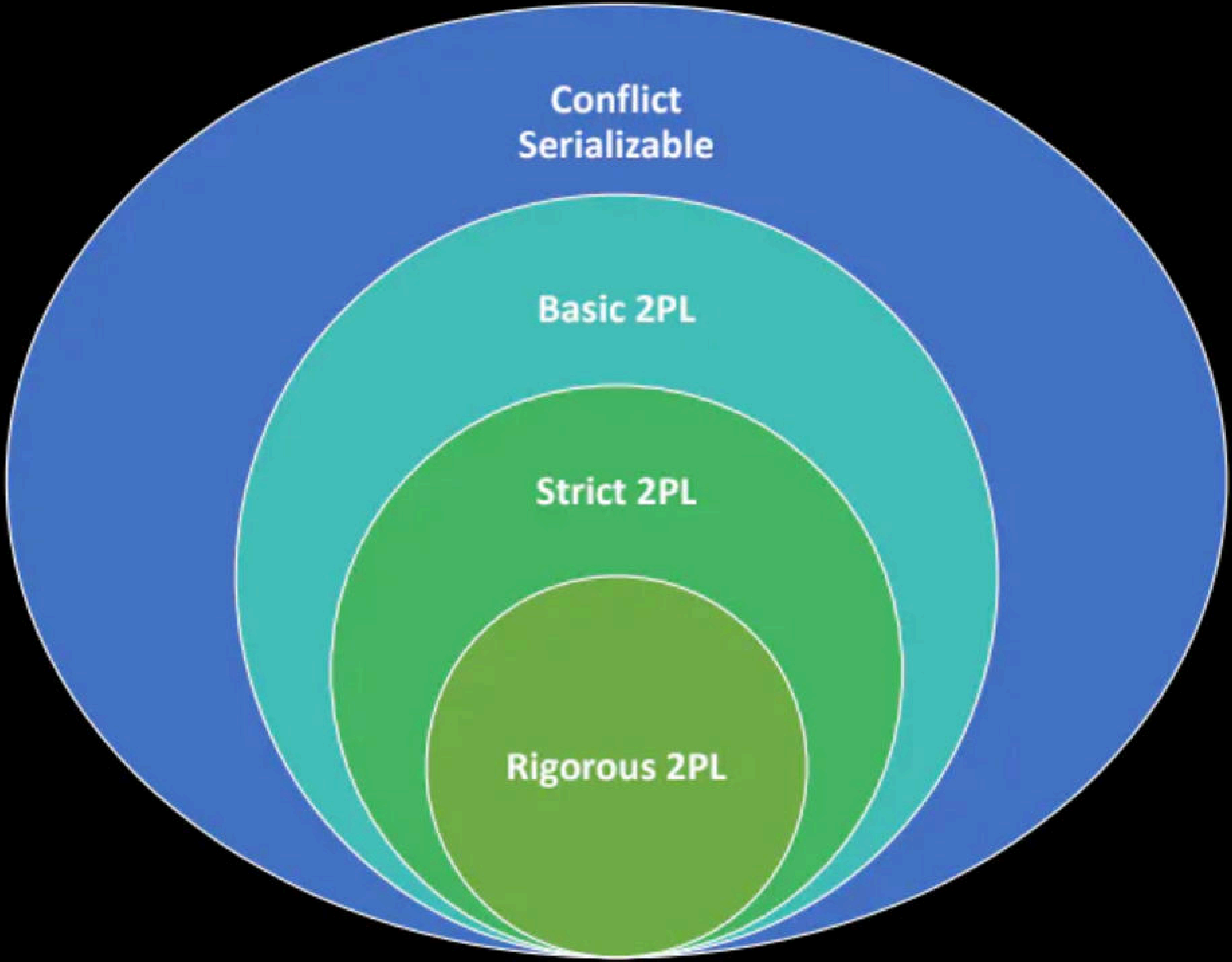
Read-set (x, y, B)

write-set (z, A)

exclusive

T_1	
R(x)	
R(y)	
W(z)	
W(A)	
R(B)	

2PL



Allowed under basic 2PL ?

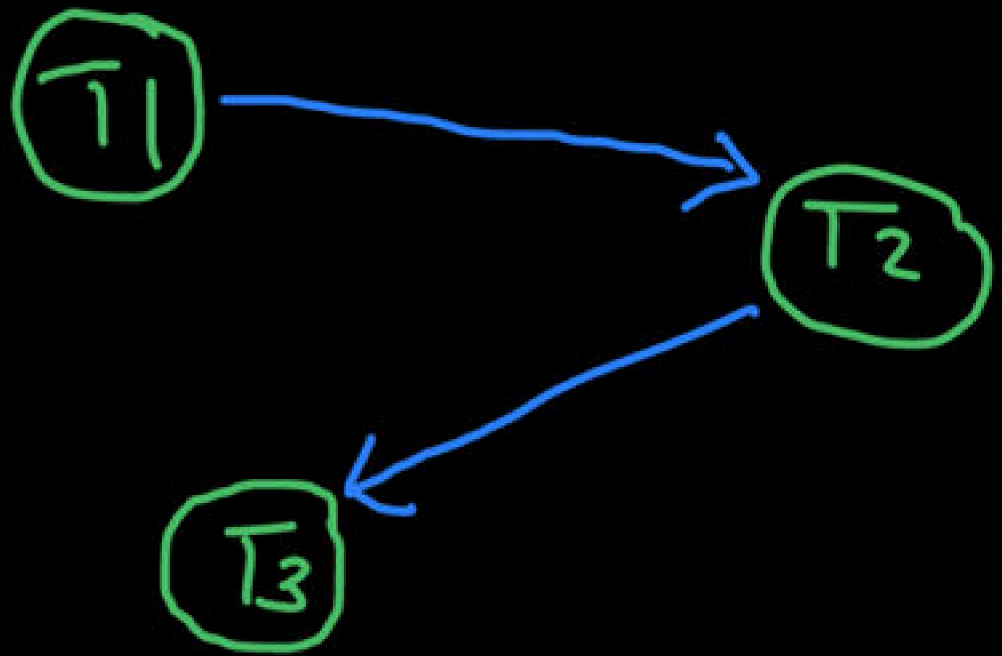
T1	T2	T3
	lock_S(A)	
	lock_S(B), lock_Ex(C)	
	R(A)	
	unlock(A)	lock_Ex(A)
		W(A)
lock_Ex(B)		
W(B)		
		Unlock(A)
	R(B)	
	W(C)	

blocked

not allowed under 2PL

2PL

T1	T2	T3
	R(A)	
		W(A)
W(B)		
	R(B)	
	W(C)	



conflict serializable

T1 ⟶ T2 ⟶ T3

Timestamp

↳ At what time transaction started

Younger vs Older transaction

Assume 2 transactions T1 and T2 and their respective timestamps TS(T1) and TS(T2)

TS(T1) < TS(T2)

⇓

T1 ⇒ older transaction

T2 ⇒ younger transaction

Deadlock Prevention

1. Wait_Die
2. Wait_Wound

Deadlock Prevention

Assume 2 transactions T_i and T_j. T_i tries to acquire lock on a database item x, which is already locked by T_j.

unacademy

Deadlock Prevention

Assume 2 transactions T_i and T_j. T_i tries to acquire lock on a database item x, which is already locked by T_j.

Wait_Die: An older transaction is allowed to wait for a younger transaction, whereas a younger transaction requesting an item held by an older transaction is aborted and restarted with same timestamp.

Deadlock Prevention

Assume 2 transactions T_i and T_j. T_i tries to acquire lock on a database item x, which is already locked by T_j.

Wait_Wound: A younger transaction is allowed to wait for an older one, whereas if an older transaction requests an item held by the younger transaction, we preempt the younger transaction by aborting it.

Starvation

Question

Assume that T1 requests a lock held by t2. Consider the following table which shows the actions taken for wait_die and wait_wount schemes:

	Wait_Die	Wait_Wound
T1 is younger than T2	W	X
T2 is older than T2	Y	Z

What will be the correct status of T1 andT2 at W, X, Y, and Z respectively?

Question GATE-2017

In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let $TS(T_1)$ and $TS(T_2)$ be the timestamps of transactions T_1 and T_2 respectively. Besides, T_1 holds a lock on the resource R, and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp.

if $TS(T_2) < TS(T_1)$ then

T_1 is killed

else T_2 waits.

Assume any transaction that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks?

A. The database system is both deadlock-free and starvation-free.
B. The database system is deadlock-free, but not starvation-free.
C. The database system is starvation-free, but not deadlock-free.
D. The database system is neither deadlock-free nor starvation-free.

Happy Learning.!

easiest → Toughest

$\boxed{\text{Revision order}}$

CoA → OS

ToC → CD

C → DS → algo.