Advanced Database Concept

Assignment # 1 Solution

Question 1.

a. S1: r1(X), w3(X), c3, w1(Y), c1, r2(Y), w2(Z), c2

Strict schedule because no transaction is reading / updating a data item which is updated by another transaction.

b. S2: r1(X), w2(X), w1(X), c2, c1

Cascadeless schedule because although there is no dirty read but T2 updated X and then T1 updated X before commit of T2.

c. S3: r2(X), w3(X), c3, w1(Y), r2(Y), r2(Z), c2, r1(Z), c1

Non-recoverable schedule because T2 dirty reads Y updated by T1 and committed before T1.

d. S4: r1(X), w2(X), w1(X), r3(X), c1, c2, c3

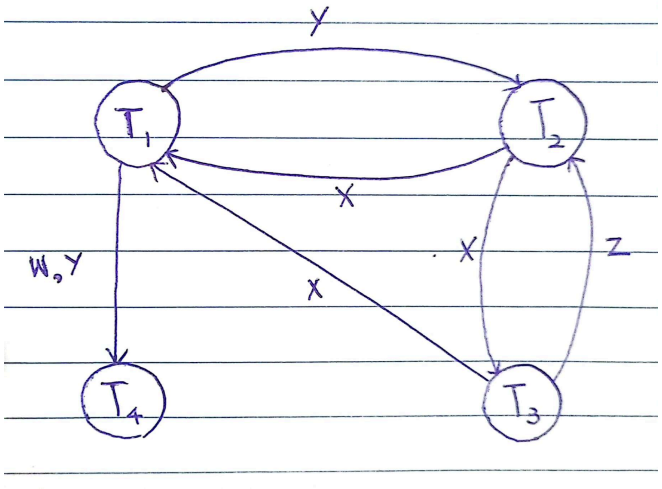
Recoverable schedule because T3 dirty reads X updated by T1 and T2 but T1 and T2 commits before T3.

e. S5: r1(X), r2(Z), r3(X), r1(Z), r2(Y), r3(Y), w1(X), c1, w2(Z), w3(Y), w2(Y), c3, c2

Cascadeless schedule because although there is no dirty read but T3 updated Y and then T2 updated Y before commit of T3.

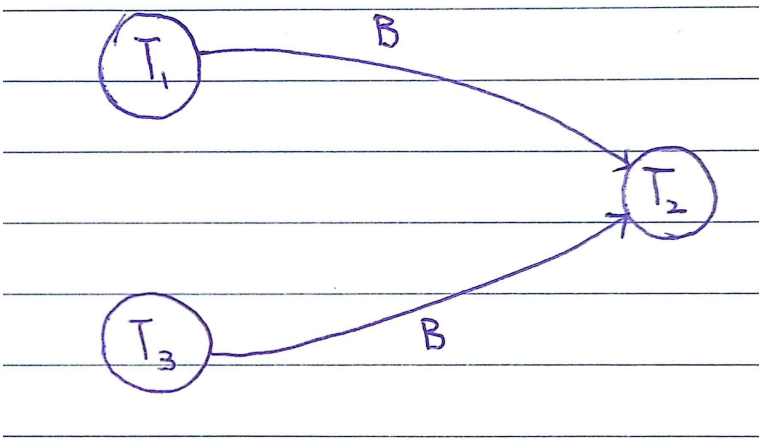
Question 2.

a. S1: r1(W), r2(X), w1(Y), r3(Z), r2(Y), w4(W), w3(X), r4(Y), w2(Z), w1(X)



There exists a cycle in the precedence graph. Therefore, the given schedule S1 is not conflict-serializable.

b. S2: r1(A), r2(C), r3(A), w2(C), r3(B), r1(B), w2(B)



There exists no cycle in the precedence graph. Therefore, the given schedule S2 is conflict-serializable. Equivalent serial schedules are:

i. T1 → T3 → T2
ii. T3 → T1 → T2

Question 3.

a. S1: r2(X), w3(X), w1(Y), r2(Y), r2(Z), r3(Y), c3, c2, r1(Z), c1.

i. Basic 2PL with protocol based on a timestamp for deadlock avoidance (use wait-die policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
	s2-lock(X) r2(X)	
		x3-lock(X)…**abort** due to T2 & restart with same TS
x1-lock(Y) w1(Y)		
	s2-lock(Y)…**abort** due to T1 & restart with same TS unlock(X)	
s1-lock(Z) r1(Z) unlock(Y) unlock(Z) **c1**		
	T2 can now restart here.	
		T3 can now restart here.

ii. Strict 2PL with protocol based on a timestamp for deadlock avoidance (use wound-wait policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
	s2-lock(X)	
	r2(X)	
		x3-lock(X)…**wait** for T2 on X
x1-lock(Y)		
w1(Y)		
	s2-lock(Y)…**wait** for T1 on Y	
s1-lock(Z)		
r1(Z)		
unlock(Z)		
c1, releases all locks		
unlock(Y)		
	r2(Y)…**wake-up**	
	s2-lock(Z)	
	r2(Z)	
	unlock(X)	
	unlock(Y)	
	unlock(Z)	
	c2	
		w3(X)…**wake-up**
		s3-lock(Y)
		r3(Y)
		unlock(Y)
		c3, releases all locks
		unlock(X)

iii. Rigorous 2PL with protocol based on a timestamp for deadlock avoidance (use wait-die policy)

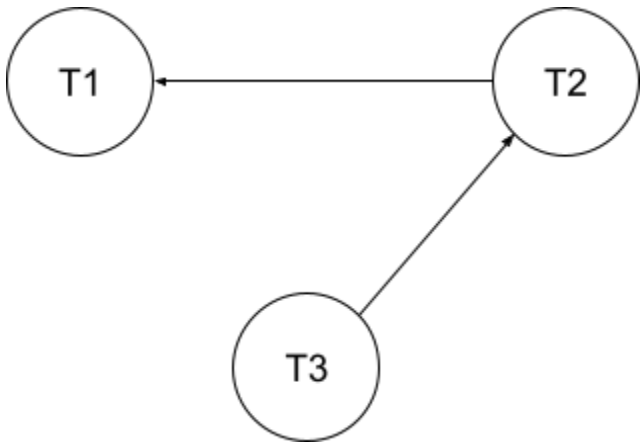
Transaction T_1	**Transaction T_2**	**Transaction T_3**
	s2-lock(X) r2(X)	
		x3-lock(X)…**abort** due to T2 & restart with same TS
x1-lock(Y) w1(Y)		
	s2-lock(Y)…**abort** due to T1 & restart with same TS unlock(X)	
s1-lock(Z) r1(Z) **c1**, releases all locks unlock(Y) unlock(Z)		
	T2 can now restart here.	
		T3 can now restart here.

iv. Rigorous 2PL with protocol based on a timestamp for deadlock avoidance (use wound-wait policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
	s2-lock(X)	
	r2(X)	
		x3-lock(X)…**wait** for T2 on X
x1-lock(Y)		
w1(Y)		
	s2-lock(Y)…**wait** for T1 on Y	
s1-lock(Z)		
r1(Z)		
c1, releases all locks		
unlock(Y)		
unlock(Z)		
	r2(Y)…**wake-up**	
	s2-lock(Z)	
	r2(Z)	
	c2, releases all locks	
	unlock(X)	
	unlock(Y)	
	unlock(Z)	
		w3(X)…**wake-up**
		s3-lock(Y)
		r3(Y)
		c3, releases all locks
		unlock(X)
		unlock(Y)

v. Rigorous 2PL with protocol based on a deadlock detection (Use wait-for-graph to deal with deadlock)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
	s2-lock(X) r2(X)	
		x3-lock(X)…**wait** for T2 on X
x1-lock(Y) w1(Y)		
	s2-lock(Y)…**wait** for T1 on Y	
s1-lock(Z) r1(Z) **c1**, releases all locks unlock(Z) unlock(Y)		
	r2(Y)…**wake-up** s2-lock(Z) r2(Z) **c2,** releases all locks unlock(X) unlock(Y) unlock(Z)	
		w3(X)…**wake-up** s3-lock(Y) r3(Y) **c3**, releases all locks unlock(X) unlock(Y)



vi. Basic timestamp ordering (TO) protocol

T1	**T2**	**T3**
	r2(X)	
		w3(X)
w1(Y)		
	r2(Y)	
	r2(Z)	
		r3(Y)
		c3
	c2	
r1(Z)		
c1		

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
{T2}					
	T3				
			T1		
		{T2}			
				{T2}	
		{T3}			
				{T2}	

vii. Strict timestamp ordering protocol

T1	**T2**	**T3**
	r2(X)	
		w3(X)
w1(Y)		
	r2(Y) **delay T2** until c1 or a1, as WTS(Y)<TS(T2)	
		r3(Y) **delay T3** until c1 or a1, as WTS(Y)<TS(T3)
r1(Z)		
c1		
	r2(Y), exe here	
	r2(Z)	
		r3(Y), exe here
		c3
	c2	

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
{T2}					
	T3				
			T1		
				{T1}	
		{T2}			
				{T2}	
		{T3}			

viii. Timestamp ordering using Thomas's write rule (TWR)

T1	**T2**	**T3**
	r2(X)	
		w3(X)
w1(Y)		
	r2(Y)	
	r2(Z)	
		r3(Y)
		c3
	c2	
r1(Z)		
c1		

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
{T2}					
	T3				
			T1		
		{T2}			
				{T2}	
		{T3}			
				{T2}	

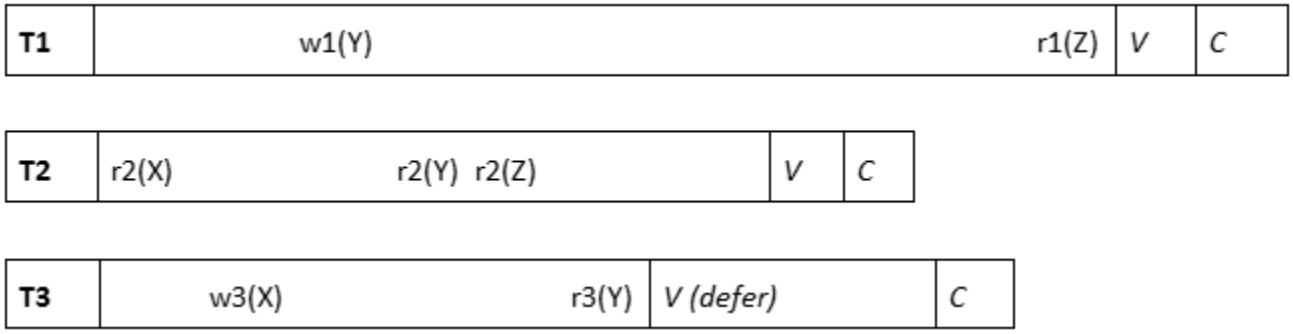
ix. Multi-version timestamp ordering protocol

T1	**T2**	**T3**
	r2(X)	
		w3(X)
w1(Y)		
	r2(Y)	
	r2(Z)	
		r3(Y)
		c3
	c2	
r1(Z)		
c1		

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
{T2}					
	T3				
			T1		
		{T2}			
				{T2}	
		{T3}			
				{T2}	

x. Validation (Optimistic) concurrency control technique (use defer the validation until a later time when the conflicting transactions have finished)



When T3 tries to commit

Backward: passed because no committed transactions

Forward: WS(T3) ∩ RS(T1, T2)

{X} ∩ {X, Y, Z} = {X} **failed**

So, T3 will defer.

When T2 tries to commit

Backward: passed because no committed transactions

Forward: WS(T2) ∩ RS(T1, T3)

{ } ∩ {Y, Z} = { } **passed**

So, T2 will commit.

T3 again tries to commit

Backward: RS(T3) ∩ WS(T2)

{Y} ∩ { } = { } **passed**

Forward: WS(T3) ∩ RS(T1)

{X} ∩ { } = { } **passed**

So, T3 will commit.

T1 tries to commit

Backward: RS(T1) ∩ WS(T2, T3)

{Z} ∩ {X} = { } **passed**

Forward: passed because no active transactions

So, T1 will commit.

b. S2: r1(Z), r1(Y), w1(Y), w2(Y), r2(Z), r3(X), w3(X), w1(X), c1, w2(Z), r3(Y), c2, c3.

i. Basic 2PL with protocol based on a timestamp for deadlock avoidance (use wait-die policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(Z)		
r1(Z)		
x1-lock(Y)		
r1(Y)		
w1(Y)		
	x2-lock(Y)…**abort** due to T1 & restart with same TS	
		x3-lock(X)
		r3(X)
		w3(X)
x1-lock(X)…**wait** for T3 on X		
		s3-lock(Y)…**abort** due to T1 & restart with same TS
		unlock(X)
w1(X)…**wake-up**		
unlock(Z)		
unlock(Y)		
unlock(X)		
c1		
	T2 can now restart here.	**T3** can now restart here.
	x2-lock(Y)	
	w2(Y)	
	x2-lock(Z)	
	r2(Z)	
		x3-lock(X)
		r3(X)
		w3(X)
	w2(Z)	
	unlock(Y)	
	unlock(Z)	
		s3-lock(Y)
		r3(Y)
		unlock(X)
		unlock(Y)
	c2	
		c3

ii. Strict 2PL with protocol based on a timestamp for deadlock avoidance (use wound-wait policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(Z)		
r1(Z)		
x1-lock(Y)		
r1(Y)		
w1(Y)		
	x2-lock(Y)…**wait** for T1 on Y	
		x3-lock(X)
		r3(X)
		w3(X)
x1-lock(X)…**wound T3**		
		abort due to T1 & restart with same TS & release all locks
		unlock(X)
w1(X)		
unlock(Z)		
c1		
unlock(Y)		
unlock(X)		
	w2(Y)…**wake-up**	**T3** can now restart here.
	x2-lock(Z)	
	r2(Z)	
		x3-lock(X)
		r3(X)
		w3(X)
	w2(Z)	
		s3-lock(Y)…**abort** due to T2 & restart with same TS
		unlock(X)
	c2	
	unlock(Y)	
	unlock(Z)	
		T3 can now restart here.

iii. Rigorous 2PL with protocol based on a timestamp for deadlock avoidance (use wait-die policy)

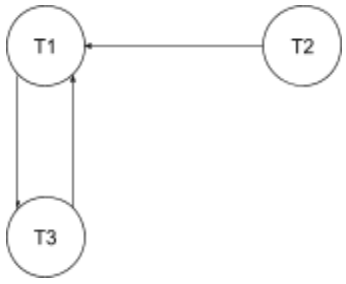
Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(Z)		
r1(Z)		
x1-lock(Y)		
r1(Y)		
w1(Y)		
	x2-lock(Y)…**abort** due to T1 & restart with same TS	
		x3-lock(X)
		r3(X)
		w3(X)
x1-lock(X)…**wait** for T3 on X		
		s3-lock(Y)…**abort** due to T1 & restart with same TS
		unlock(X)
w1(X)…**wake-up**		
c1		
unlock(Z)		
unlock(Y)		
unlock(X)		
	T2 can now restart here.	**T3** can now restart here.
	x2-lock(Y)	
	w2(Y)	
	x2-lock(Z)	
	r2(Z)	
		x3-lock(X)
		r3(X)
		w3(X)
	w2(Z)	
		s3-lock(Y)…**abort** due to T2 & restart with same TS
		unlock(X)
	c2	
	unlock(Y)	
	unlock(Z)	
		T3 can now restart here.

iv. Rigorous 2PL with protocol based on a timestamp for deadlock avoidance (use wound-wait policy)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(Z)		
r1(Z)		
x1-lock(Y)		
r1(Y)		
w1(Y)		
	x2-lock(Y)…**wait** for T1 on Y	
		x3-lock(X)
		r3(X)
		w3(X)
x1-lock(X)…**wound T3**		
		abort due to T1 & restart with same TS & release all locks
		unlock(X)
w1(X)		
c1		
unlock(Z)		
unlock(Y)		
unlock(X)		
	w2(Y)…**wake-up**	**T3** can now restart here.
	x2-lock(Z)	
	r2(Z)	
		x3-lock(X)
		r3(X)
		w3(X)
	w2(Z)	
		s3-lock(Y)…**wait** for T2 on Y
	c2	
	unlock(Y)	
	unlock(Z)	
		r3(Y)…**wake-up**
		c3
		unlock(X)
		unlock(Y)

v. Rigorous 2PL with protocol based on a deadlock detection (Use wait-for-graph to deal with deadlock)

Transaction T_1	**Transaction T_2**	**Transaction T_3**
s1-lock(Z)		
r1(Z)		
x1-lock(Y)		
r1(Y)		
w1(Y)		
	x2-lock(Y)…**wait** for T1 on Y	
		x3-lock(X)
		r3(X)
		w3(X)
x1-lock(X)…**wait** for T3 on X		
		s3-lock(Y)…**wait** for T1 on Y
		abort due to cycle in the graph and release locks
		unlock(X)
w1(X)…**wake-up**		
c1		
unlock(Z)		
unlock(Y)		
unlock(X)	w2(Y)…**wake-up**	**T3** can now restart here.
	x2-lock(Z)	
	r2(Z)	
		x3-lock(X)
		r3(X)
		w3(X)
	w2(Z)	
		s3-lock(Y)…**wait** for T2 on Y
	c2	
	unlock(Y)	
	unlock(Z)	
		r3(Y)…**wake-up**
		c3
		unlock(X)
		unlock(Y)



vi. Basic timestamp ordering (TO) protocol

T1	**T2**	**T3**
r1(Z)		
r1(Y)		
w1(Y)		
	w2(Y)	
	r2(Z)	
		r3(X)
		w3(X)
w1(X) **abort T1** and restart with new TS as WTS(X)>TS(T1)		
	w2(Z)	
		r3(Y)
	c2	
		c3

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
				{T1}	
		{T1}			
			T1		
			T2		
				{T2}	
{T3}					
	T3				
					T2
		{T3}			

vii. Strict timestamp ordering protocol

T1	**T2**	**T3**
r1(Z)		
r1(Y)		
w1(Y)		
	w2(Y) **delay T2** until c1 or a1, as WTS(Y)<TS(T2)	
		r3(X)
		w3(X)
w1(X) **abort T1** and restart with new TS as WTS(X)>TS(T1)		
	w2(Y), exe here	
	r2(Z)	
	w2(Z)	
		r3(Y) **delay T3** until c2 or a2, as WTS(Y)>TS(T3)
	c2	
		r3(Y), exe here
		c3

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
				{T1}	
		{T1}			
			T1		
{T3}					
	T3				
			T2		
				{T2}	
					T2
		{T3}			

viii. Timestamp ordering using Thomas's write rule (TWR)

T1	**T2**	**T3**	*Timestamps and versions of Objects*					
			X		**Y**		**Z**	
			RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
			{ }	**T0**	{ }	**T0**	{ }	**T0**
r1(Z)							{T1}	
r1(Y)					{T1}			
w1(Y)						T1		
	w2(Y)					T2		
	r2(Z)						{T2}	
		r3(X)	{T3}					
		w3(X)		T3				
w1(X) **abort T1** and restart with new TS as RTS(X)>TS(T1)								
	w2(Z)							T2
		r3(Y)			{T3}			
	c2							
		c3						

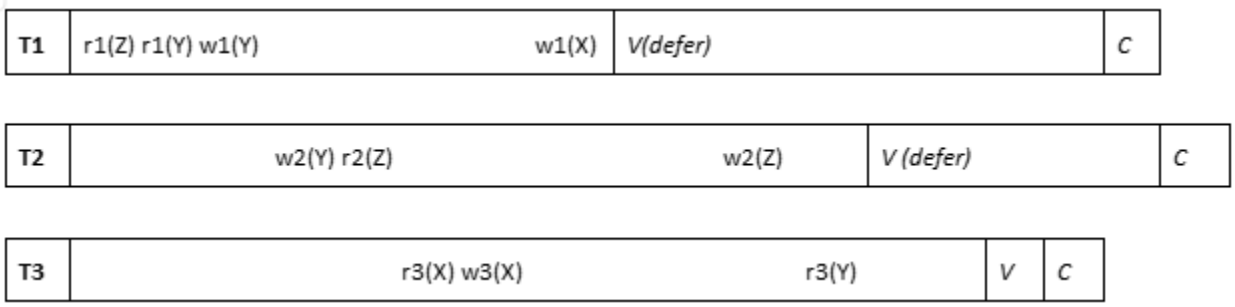
ix. Multi-version timestamp ordering protocol

T1	**T2**	**T3**
r1(Z)		
r1(Y)		
w1(Y)		
	w2(Y)	
	r2(Z)	
		r3(X)
		w3(X)
w1(X) **abort T1** and restart with new TS as RTS(X)>TS(T1)		
	w2(Z)	
		r3(Y)
	c2	
		c3

Timestamps and versions of Objects

X		**Y**		**Z**	
RTS	*WTS*	*RTS*	*WTS*	*RTS*	*WTS*
{ }	**T0**	{ }	**T0**	{ }	**T0**
				{T1}	
		{T1}			
			T1		
		{T2}	T1 T2		
				{T2}	
{T3}					
	T3				
					T2
		{T3}			

x. Validation (Optimistic) concurrency control technique (use defer the validation until a later time when the conflicting transactions have finished)



When T1 tries to commit

Backward: passed because no committed transactions

Forward: WS(T1) ∩ RS(T2, T3)

{X, Y} ∩ {X, Z} = {X} **failed**

So, T1 will defer.

When T2 tries to commit

Backward: passed because no committed transactions

Forward: WS(T2) ∩ RS(T1, T3)

{Y, Z} ∩ {X, Y, Z} = {Y, Z} **failed**

So, T2 will defer.

When T3 tries to commit

Backward: passed because no committed transactions

Forward: WS(T3) ∩ RS(T1, T2)

{X} ∩ {Y, Z} = { } **passed**

So, T3 will commit.

T1 again tries to commit

Backward: RS(T1) ∩ WS(T3)

{Y, Z} ∩ {X} = { } **passed**

Forward: WS(T1) ∩ RS(T2)

{X, Y} ∩ {Z} = { } **passed**

So, T1 will commit.

T2 again tries to commit

Backward: RS(T2) ∩ WS(T1, T3)

{Z} ∩ {X, Y} = { } **passed**

Forward: passed because no active transactions

So, T2 will commit.