CS 541 Spring 12
Assignment 4
DUONG NGUYEN

1. (Ex 16.2)

16.2.1. T_2 : $R(Y)$, $W(Y)$

without concurrency control, T_2 could interfere with T_1

T_1 :	$R(X)$	$W(X)$		$R(Y)$		$W(Y)$
T_2 :			$R(Y)$		$W(Y)$	

} lost update.

16.2.2. with strict 2PL locking, either T_1 or T_2 will obtain exclusive lock on Y first.

If T_1 obtains first

T_1:	T_2
$X(X)$	
$R(X)$	
$X(Y)$	
$R(Y)$	
$W(Y)$	
Commit	
	$X(Y)$
	$R(Y)$
	$W(Y)$
	Commit

If T_2 obtains first

T_1	T_2
$X(X)$	
$R(X)$	
$W(X)$	
	$X(Y)$
	$R(Y)$
	$W(Y)$
	Commit
$X(Y)$	
$R(Y)$	
$W(Y)$	
commit	

2. (Ex. 17.2)

1,

T_1	T_2
R(x)	
	R(x)
W(x)	
	W(x)

Serializable: unknown.
If both T_1 and T_2 commit ⇒ No
If T_1 aborts, T_2 commit ⇒ Yes (T_1(abort) → T_2)

Conflict serializable: unknown.
If both commit ⇒ precedence graph is

$T_1 \rightleftarrows T_2$

⇒ cycle ⇒ No
If either transaction aborts ⇒ YES.

View-serializable: unknown
If both T_1 and T_2 commit ⇒ No.
If T_1 abort, T_2 commit ⇒ YES (T_1(abort) → T_2)

recoverable: YES
Avoid-cascading abort: YES
Strict: No.
Since X written by T_1 is overwritten by T_2 before T_1 commits / aborts

2,

T_1	T_2
W(x)	
	R(Y)
R(Y)	
	R(X)

Serializable: YES
(T_1 → T_2)

Conflict serializable: YES
Even both commit, graph is acyclic $T_1 \rightarrow T_2$

View serializable: YES
(T_1 → T_2)

Recoverable: Unknown
If T_2 commits after T_1 commits ⇒ YES
If T_2 commits before T_1 ⇒ No

Avoid cascading Abort: No.
Since T_2 read X written by T_1 before T_1 commit / abort.

Strict: No
Since X written by T_1 is read by T_2 before T_1 commits.

5,

T_1	T_2
R(x)	
	w(x)
w(x)	
	Abort.
Com.	

Serializable: YES
(T_1(abort) → T_2 ≡ T_2)

Conflict - serializable: YES
graph has only 1 node ⇒ acyclic

View - serializable: YES
(T_1 → T_2 ≡ T_2)

recoverable: YES

Avoid cascading abort: YES

Strict: No
since x written by T_2 is over written by T_1
before T_2 aborts.

9,

T_1	T_2
w(x)	
	R(x)
w(x)	
	Com.
Abort	

serializable: YES
(T_1(abort) → T_2 ≡ T_2).

Conflict - serializable: YES
graph has only one node ⇒ acyclic

View serializable: YES
(T_1(abort) → T_2 ≡ T_2)

recoverable: No
Since T_2 read x ~~fro~~ written by T_1 while
T_2 commits before T_1 abort.

avoid cascading abort: No
Since T_2 commits before T_1 abort while
T_2 read x written by T_1

Strict: No
since T_2 read x from T_1 before T_1 commits.

11,

T_1	T_2	T_3
R(x)		
	W(x)	
	Com.	
W(x)		
Com.		
		R(x)
		Com.

Serializable: No.
Due to conflict between T_1 and T_2

Conflict serializable: No
Graph

$$\begin{tikzcd} T_1 \arrow[r, bend left] \arrow[dr] & T_2 \arrow[l] \arrow[d] \\ & T_3 \end{tikzcd}$$

$\Rightarrow$ cyclic

View serializable: No.
Since T_3 read x written by T_1 $\Rightarrow$
T_3 must follow T_1 in serial schedule $T_1 \rightarrow T_3$.
T_2 can't be at the middle since
T_3 will read x from T_2
T_2 can't be at the end since
T_2 will be last write of x
T_2 can't be at front since
T_1 will not read x from initial value
$\Rightarrow$ can't find equivalent serial schedule.

recoverable
avoid cascading abort
Strict

} YES

12,

T_1	T_2	T_3
R(x)		
	W(x)	
W(x)		
		R(x)
Com		
	Com	
		Com

Serializable: No
Due to conflict between T_1 and T_2

Conflict serializable: No
graph

$$\begin{tikzcd} T_1 \arrow[r, bend left] \arrow[dr] & T_2 \arrow[l] \arrow[d] \\ & T_3 \end{tikzcd}$$

$\Rightarrow$ cyclic

View serializable: No
Explain as the case of (11)

Recoverable: YES
T_3 read x from T_1, T_2 and commits after T_1, T_2 commits.

Avoid cascading abort: No
Since T_3 read x written by T_1 before T_1 commit.

Strict: No
Since x written by T_2 is overwritten by T_1 before T_2 commits.

3. (Ex. 17.8.1)

If the second query is

Replace (salary = 100 + salary) where EMP.ename == 'Santa';

Then it could conflict with the provided query.

For example, suppose table EMP has 1 'Santa' tuple

eid	ename	age	salary	did
15	'Santa'	30	20	50

If we execute query 1 and query 2 serially, the result will be

Q_1	Q_2
R(EMP)	
W(EMP)	
	R(EMP)
	W(EMP)

$\Rightarrow$

ename	salary
'Santa'	122

Q_1	Q_2
	R(EMP)
	W(EMP)
R(EMP)	
W(EMP)	

$\Rightarrow$

ename	salary
'Santa'	132

If we execute the 2 queries concurrently, it may be

Q_1	Q_2
R(EMP)	
	R(EMP)
W(EMP)	
	W(EMP)

$\Rightarrow$

ename	salary
'Santa'	120

$\Rightarrow$ different from both serial schedule!

Using locking tuples, then Q_2 can't read until Q_1 releases the exclusive lock or Q_2 will hold the exclusive lock until it is done and releases to Q_1. Thus the result will be similar to one of the 2 serial schedule.