(1)

(START T)
(T, X, 3)
(START S)
(S, Y, 1)
(T, Y, 4)
(COMMIT T)

- Both X and Y are written to disk from Transaction T
- undo (S, Y, 1) set Y to 1 and then (ABORT S)

② How far back to look when doing recovery using checkpointing

A) UNDO LOGGING:

When scanning the log up from the bottom,

i) If we find an <END CKPT> first:
Do nothing to all transactions active from the start of the CKPT that have been committed

ii) If we find a <START CKPT $T_1 \dots T_K$> first:
Scan upward until the earliest of $T_1 \dots T_K$

B) REDO LOGGING:

i) If we find an <END CKPT> first:
Apply redo logging recovery for all transactions started after the start of the CKPT or was active in the previous CKPT

ii) If we find a <START CKPT (...)> first:
Scan upwards, executing recovery until start of previous CKPT

C) UNDO/REDO LOGGING:

i) If we find an <END CKPT> first:
Go back as high as the earliest transaction which was active at the start of the CKPT

ii) If we find a <START CKPT (...)> first:
Go back as high as the earliest transaction which was active at the start of the previous CKPT

<START T_1>
<START T_2>

#3. <START DUMP>
<START CKPT (T_1, T_2)>
<T_1, A, 1, 5>
<T_2, C, 3, 6>
<COMMIT T_2>
<T_1, B, 2, 7>
<END CKPT>
Dump completes
<END DUMP>

Steps:

1. Write a log record <START DUMP>
2. Perform a checkpoint according to the logging method being used.
3. Perform a full or incremental dump of the data, making sure data is copied to the remote site.
4. Make sure that the log has been copied to the secure, remote site. The log must include all entries up to the dump of the previous checkpoint.
5. Write a log record <END DUMP>

#4

a)

T_1	T_2	T_3
$w_1 Y$		
	$w_2 Y$	
	$w_2 X$	
$w_1 X$		
		$w_3 X$

This is T_1, T_2, T_3 serial.
But it is not conflict serializable

b)

T_1	T_2
$w(A)$	
	$w(A)$
$r(A)$	

c)

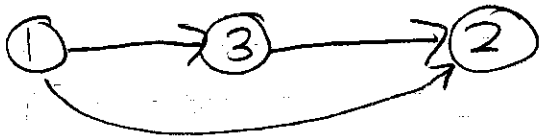
Transaction order: T_1, T_2

T_1	T_2
	$w_2 A$
$R_1 A$	
$w_1 B$	
	$w_2 B$
$R_1 B$	

// This is not realizable using 2PL, but it is realizable if using multi-version timestamp.

(5) Draw the precedence graph for the following schedule:

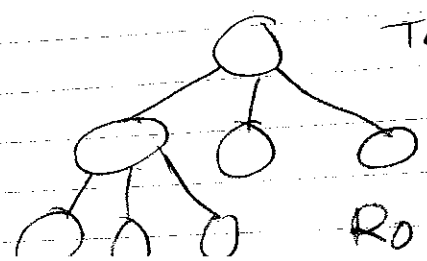
$w_1(x), r_2(y), w_3(x), r_2(x)$



(6) Explain how intensional share and intensional exclusive locks are used.

1) To place an ordinary S or X lock on any element, we must begin at the root of the hierarchy. (table level)
2) If we are at the element that we want to lock, we need lock no further. We request an S or X lock on that element.
3) If the element we wish to lock is further down the hierarchy, then we place a warning at this node; that is, if we want to get a shared lock on a subelement we request an IS lock at this node, and if we want an exclusive lock on a subelement, we request an IX lock on this node. When the lock on the current node is granted, we proceed to the appropriate child (the one whose subtree contains the node we wish to lock). We then repeat step (2) or step (3), as appropriate, until we reach the desired node.

Intensional share locks and intensional exclusive locks are used when we have different size locks. For instance, locks on tables, blocks, ~~or~~ rows.



Question #7.

Update lock - is type of a lock which helps to avoid deadlock situations.
An update lock can be used to read and can be upgraded to an exclusive lock, but only one transaction at a time is allowed to hold an update lock.

Matrix:

		Lock Requested		
		S	X	U
lock held in mode	S	Y	N	Y
	X	N	N	N
	U	N	N	N

CS157B - Practice Final

(8) **Increment Locks**

- multiple transactions can hold an increment lock, increment operations are commutative → which means that:

  inc(x, 40) = inc(x, 2)
  inc(x, 2) = inc(x, 40)

- increment locks are used when a transaction wants to increment a database value

- compatibility matrix

		Lock Requested by Someone Else		
		S	X	I
Lock Held by You	S	Y	N	N
	X	N	N	N
	I	N	N	Y

Q: 9

A phantom tuple occurs when a table is changed and a row is inserted while performing a query. Since all isolation levels we've considered involve only row level locking, phantom reads could still occur. Repeatable Read isolation stops Phantoms

A dirty read occurs when reading from data in memory that has not yet been committed. Only "read uncommitted" isolation level can result in dirty reads.

10

① When transaction T requests $R_T(X)$

$TS(T) \geq WT(X)$

and $C(X) = \text{false}$

② When transaction T request $W_T(X)$

$TS(T) \geq RT(X)$, $TS(T) < WT(X)$

if $C(X) = \text{false}$ (other transaction might abort)

5.