Group 7

Name: Hyuntaek Oh, Woonki Kim

Email: ohhyun@oregonstate.edu, kimwoon@oregonstate.edu

ONID: ohhyun, kimwoon

CS 540_001

14 Mar 2024

Group Assignment 6

1: Serializability and 2PL (4 points)

(a) Consider the following classes of schedules: serializable and 2PL. For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly. Also, for each 2PL schedule, identify whether a cascading rollback (abort) may happen. A cascading rollback will happen in a schedule if a given transaction aborts at some point in the schedule, and at least one other transaction must be aborted by the system to keep the database consistent.

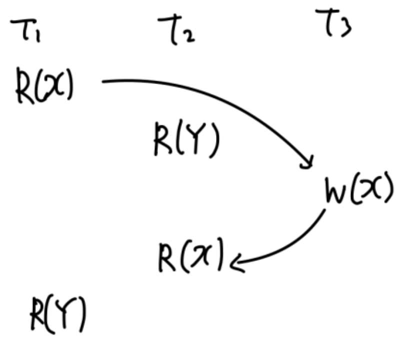
The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
2. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)
3. T1:W(X), T2:R(X), T1:W(X)
4. T1:R(X), T2:W(X), T1:W(X), T3:R(X)

Answer:

1. The schedule is serializable, since there is no cycle in the graph. This schedule is 2PL and cascading rollback (abort) may happen in this schedule.

Schedule



Graph

$$\begin{tikzcd}T_1 \arrow[r] & T_3 \arrow[r] & T_2\end{tikzcd}$$

The drawing below proves that this schedule is 2PL by showing how shared and exclusive locks are achieved and released.

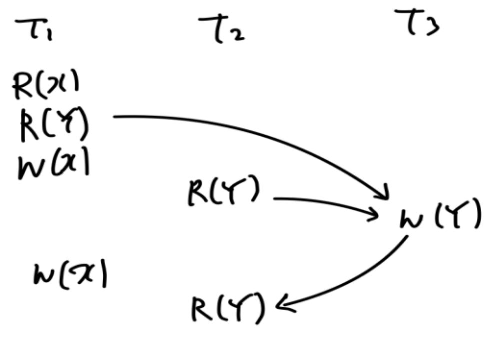
Let's say that T3 has been aborted right after T2 reads X, it is possible since T3 hasn't been committed yet.

When this happens, system will abort T2 as well. So, which means a cascading rollback (abort) may happen in this schedule.

T1	T2	T3
S.lock(X)		
S.lock(Y)		
R(X)		
S. release-lock(X)		
	S.lock(Y)	
	R(Y)	
		X.lock(X)
		W(X)
		X. release-lock(X)
	S.lock(X)	
	R(X)	Abort!
	S. release-lock(X)	
R(Y)		
S. release-lock(Y)		

2. The schedule is not serializable and not a 2PL schedule, since there is a cycle in the graph.

Schedule



Graph

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

3. The schedule is not serializable and not a 2PL schedule, since there is a cycle in the graph.

Schedule

$$\begin{tikzcd}
T_1 & T_2 & T_3 \\
W(x) \arrow[dr] & & \\
& R(x) \arrow[dl] & \\
W(x) & &
\end{tikzcd}$$

Graph

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

4. The schedule is not serializable and not a 2PL schedule, since there is a cycle in the graph

Schedule

$$\begin{tikzcd}
T_1 & T_2 & T_3 \\
R(x) \arrow[dr] & & \\
& W(x) \arrow[dl] \arrow[ddr] & \\
W(x) \arrow[drr] & & \\
& & R(x)
\end{tikzcd}$$

Graph

$$\begin{tikzcd}
T_1 \arrow[r, bend left] \arrow[rr, bend left=50] & T_2 \arrow[l, bend left] \arrow[r] & T_3
\end{tikzcd}$$

.

2: Degrees of Consistency (1 points)

(a) Consider the schedule shown in Table 1.

	T1	T2
0	Start	
1	Read X	
2	Write X	
3		Start
4		Read X
5		Write X
6		Commit
7	Read Y	
8	Write Y	
9	Commit	

Table 1: Transaction schedule

What are the maximum degrees of consistency for T1 and T2 in this schedule? You must find the maximum degrees of consistency for T1 and T2 that makes this schedule possible.

Answer:

maximum degrees of consistency for T1: degree 0.

maximum degrees of consistency for T2: degree 3.

In this schedule transaction T1 begins by reading and writing X. Given that transaction T2 reads from X and writes to X before T1 commits, T1 can only have short exclusive lock which makes T1's maximum degree of consistency to 0.

Since the maximum consistency level for T1 is 0. The schedule does not have any other transactions operating on X before T2 completes. Which makes T2 to have long shared exclusive lock on X, resulting in T2's maximum degree consistency to 3.

3: Degrees of Consistency (2 points)

The degrees of consistencies in MySQL are called isolation levels: https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html

Describe each degree of consistency in MySQL briefly and explain its equivalent degree of consistency explained in the class if such a degree exists.

- **Repeatable Read**
  - Description: This is default isolation level for InnoDB. This level of isolation ensures that if reads the same data multiple times within the same transaction (nonlocking), it will get the same result each time and also ensures not to dirty other transaction data. It prevents both dirty reads and non-repeatable reads.

  - It is equivalent to Degree of consistency 3, since both guarantees not to read dirty data from other transactions and not to dirty any data of other transaction.
- **READ COMMITTED**
  - Description: In this isolation level, each transaction cannot read data that is not yet committed. So, there is no dirty read in this isolation level, but there is possibility of non-repeatable read.

  - Equivalent degree of consistency: Degree 2(Does not read dirty data from other xact). Since both degree 2 and read committed does not read dirty data from other transaction and has possibility of non-repeatable read.
- **READ UNCOMMITTED**
  - This level of isolation is the lowest level which can have “Dirty Read”, since it does not prevent reading data that has been modified by other transactions.
  - Equivalent degree of consistency: There is no equivalent degree of consistency. Since even the lowest level of consistency introduced in class, which is degree 0 prevents not to overwrite dirty data of other transactions.

- **SERIALIZABLE**
  - Description: This level of isolation ensures transactions that are running at the same time to run serially, which means transactions do not influence other transactions and do not get influenced either.

  - Equivalent degree of consistency: There is no equivalent degree of consistency. Since the highest degree of consistency introduced in class, which is degree 3, does not prevent phantom reads.

4: Recovery and ARIES (2 points)

In this problem, you need to simulate the actions taken by the ARIES algorithm. Consider the following log records and buffer actions:

time	LSN	Log	Buffer actions
0	00	update: T1 updates P7	P7 brought into the buffer
1	10	update: T0 updates P9	P9 brought into the buffer; P9 flushed to disk
2	20	update: T1 updates P8	P8 brought into the buffer; P8 flushed to disk
3	30	begin_checkpoint	
4	40	end_checkpoint	
5	50	update: T1 updates P9	P9 brought into the buffer
6	60	update: T2 updates P6	P6 brought into the buffer
7	70	update: T1 updates P5	P5 brought into the buffer
8	80	update: T1 updates P7	P6 flushed to disk
9		CRASH RESTART	

(a) For the actions listed above, show Transaction Table (XT) and Dirty Page Table (DPT) after each action. Assume that DPT holds pageID and recLSN, and XT contains transID and lastLSN.

Transaction Table (XT) | **Dirty Page Table(DPT)**

0

transID	lastLSN
T1	0

pageID	recLSN
P7	0

1

transID	lastLSN
T1	0
T0	10

pageID	recLSN
P7	0
P9	10

P9 flushed to disk

2

transID	lastLSN
T1	20
T0	10

pageID	recLSN
P7	0
P8	20

P8 flushed to disk

3 begin_checkpoint

4 end_checkpoint

5

transID	lastLSN
T1	50
T0	10

pageID	recLSN
P7	0
P9	50

6

transID	lastLSN
T1	50
T0	10
T2	60

pageID	recLSN
P7	0
P9	50
P6	60

7

transID	lastLSN
T1	70
T0	10
T2	60

pageID	recLSN
P7	0
P9	50
P6	60
P5	70

8

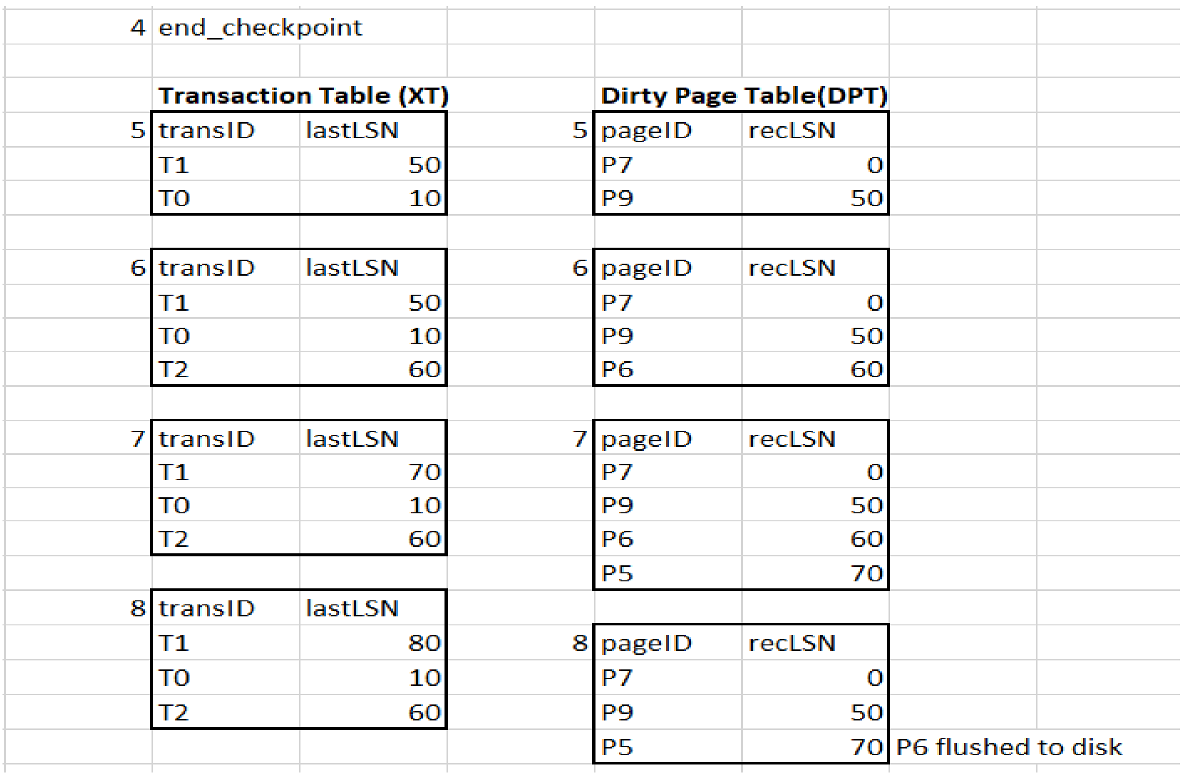
transID	lastLSN
T1	80
T0	10
T2	60

pageID	recLSN
P7	0
P9	50
P5	70

P6 flushed to disk

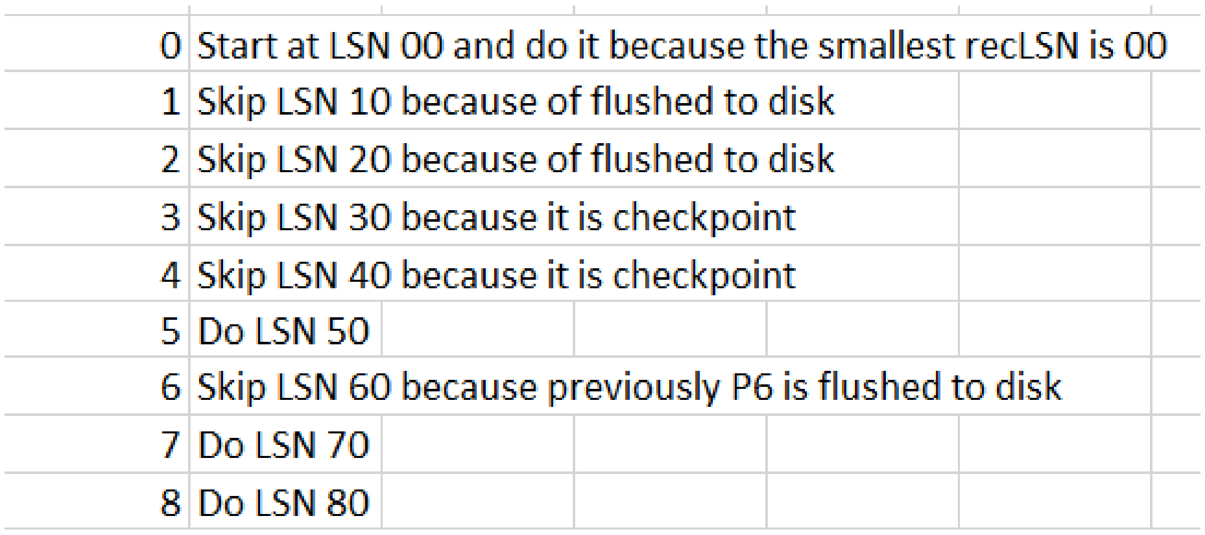
9 crash restart

(b) Simulate Analysis phase to reconstruct XT and DPT after the crash. Identify the point where the Analysis phase starts scanning log records and show XT and DPT after each action.



4	end_checkpoint					
	Transaction Table (XT)			**Dirty Page Table(DPT)**		
5	transID	lastLSN	5	pageID	recLSN	
	T1	50		P7	0	
	T0	10		P9	50	
6	transID	lastLSN	6	pageID	recLSN	
	T1	50		P7	0	
	T0	10		P9	50	
	T2	60		P6	60	
7	transID	lastLSN	7	pageID	recLSN	
	T1	70		P7	0	
	T0	10		P9	50	
	T2	60		P6	60	
				P5	70	
8	transID	lastLSN				
	T1	80	8	pageID	recLSN	
	T0	10		P7	0	
	T2	60		P9	50	
				P5	70	P6 flushed to disk

(c) Simulate Redo phase: first identify where the Redo phase starts scanning the log records. Then, for each action identify whether it needs to be redone or not.



0	Start at LSN 00 and do it because the smallest recLSN is 00
1	Skip LSN 10 because of flushed to disk
2	Skip LSN 20 because of flushed to disk
3	Skip LSN 30 because it is checkpoint
4	Skip LSN 40 because it is checkpoint
5	Do LSN 50
6	Skip LSN 60 because previously P6 is flushed to disk
7	Do LSN 70
8	Do LSN 80

5: Recovery and ARIES (1 points)

The algorithm used in MySQL for crash recovery is explained at https://dev.mysql.com/doc/refman/8.0/en/innodb-recovery.html#innodb-crash-recovery

Explain the similarities and differences between this algorithm and ARIES algorithm.

Answer:

1. **Tablespace discovery**: System Identifies tablespace and, the redo logs, which is written since the last checkpoint, must be applied to affected tablespace. Tablespace discovery is similar to the analysis phase in ARIES algorithm. The tablespace recovery fills out the tablespace, which is a datafile, so it can be used redo logs. Likewise, the analysis phase in the ARIES algorithm fills out dirty page table and transaction table that can be cleaned during crash. While activating recovery system, both algorithms analyze which transaction to be redo and undo, and thus they are similar.

2. **Redo log application:** Redo log is A disk-based data structure used during crash recovery, to correct data written by incomplete transactions. It encodes requests to change InnoDB table data, which result from SQL statements. In the redo log, the current maximum auto-increment counter value is used each time the value changes, which makes it crash-safe. And the redo phase in ARIES algorithm repeats previous transactions to reconstruct at crash and re-apply all updates. Both InnoDB and ARIES algorithms do not redo the data that already flushed to the disk. Moreover, they redo or reconstruct all updates that happen during the crash recovery process.

3. **Rollback of incomplete transaction:** Rollback in InnoDB algorithm revert any changes that is derived from incomplete transaction are abandoned, going back to its state before the transaction began. Similarly, undo phase in ARIES algorithm set back transactions and return to the position to reverse their changes effectively. Both algorithms undo incomplete transactions, so they are similar.

4. **Change buffer merge:** There is no similar function in ARIES. The change buffer merge helps speed up data writing to the database by organizing changes before they are applied. It reduces the workload required to update indexes, especially when there is a large amount of new data being added. Ultimately, the change buffer merge ensures that index pages accurately reflect the updated data.

5. **Purge:** There is no similar function in ARIES. During the purge phase, It scans the undo log to identify and delete records that are delete-marked, since it is no longer needed for maintaining transactions consistency or recovery. By doing so, the database frees up disk space and ensures efficiency.