CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2021)
PROF. LIN MA AND ANDREW CROTTY

Homework #4 (by Sophie Qiu)
Due: **Wednesday Nov 10, 2021 @ 11:59pm**

IMPORTANT:

- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Wednesday Nov 10, 2021**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: ≈ 1 - 2 hours (0.5 - 1 hours for each question)

Revision : 2021/11/09 01:21

Question	Points	Score
Serializability and 2PL	18	
Deadlock Detection and Prevention	42	
Hierarchical Locking	20	
Optimistic Concurrency Control	20	
Total:	100	

Question 1: Serializability and 2PL ....[18 points]

(a) Yes/No questions:

i. **[2 points]** A conflict serializable schedule need not always be view serializable.
☐ Yes ☐ No

ii. **[2 points]** There could be schedules under 2PL (not rigorous) that are not serializable.
☐ Yes ☐ No

iii. **[2 points]** A view serializable schedule may contain a cycle in its precedence graph.
☐ Yes ☐ No

iv. **[2 points]** It is not possible to have a deadlock in rigorous 2PL.
☐ Yes ☐ No

v. **[2 points]** You will never have unrepeatable reads in rigorous 2PL.
☐ Yes ☐ No

(b) Serializability:
Consider the schedule given below in Table 1. R(·) and W(·) stand for 'Read' and 'Write', respectively.

事务

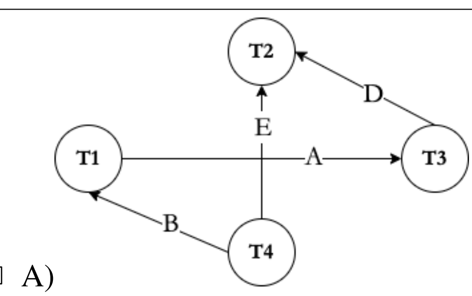
time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
T_1	R(A)					R(C)	R(B)		W(C)		
T_2				R(C)						W(D)	W(E)
T_3					W(A)			R(D)			
T_4		R(E)	W(B)								

Table 1: A schedule with 4 transactions

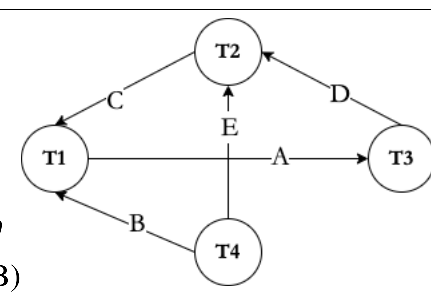
i. **[1 point]** Is this schedule serial?
☐ Yes ☐ No

ii. **[2 points]** Choose the correct dependency graph of the schedule given above. Each edge in the dependency graph looks like this: '$T_x \rightarrow T_y$ with Z on the arrow indicating that there is a conflict on Z where T_x read/wrote on Z before T_y'.

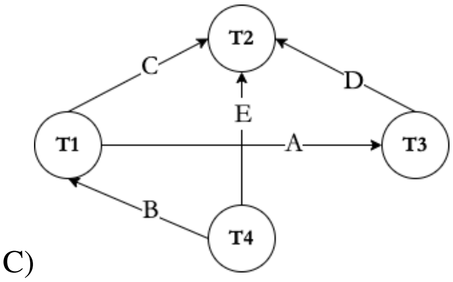
有环



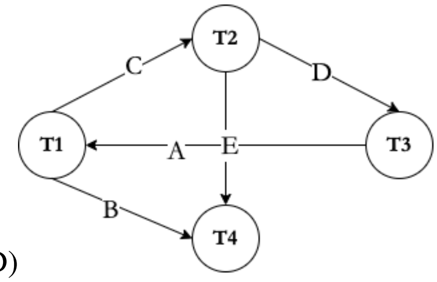
☐ A)



☐ B)



☐ C)



☐ D)

iii. **[1 point]** Is this schedule conflict serializable?
☐ Yes ☐ No

iv. **[3 points]** Mark all the transactions that can be removed from the schedule that can make it serializable.
☐ T1 ☐ T2 ☐ T3 ☐ T4 ☐ Original schedule is also serializable

v. **[1 point]** Is this schedule possible under 2PL?
☐ Yes ☐ No

Question 2: Deadlock Detection and Prevention . . . . [42 points]

(a) **Deadlock Detection:**
Consider the following two transactions and note that

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1 and T_2 represent two transactions.
- LM stands for 'lock manager'.
- Transactions will never release a granted lock. 事务结束时才会放锁

T_1: (a) read(A); (b) read(B); (c) write(B);

T_2: (d) write(A); (e) read(B); (f) read(A);

i. For each position, what lock should be requested:

α) **[1 point]** At (a): ☑ S(A) ☐ S(B) ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

β) **[1 point]** At (b): ☐ S(A) ☑ S(B) ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

γ) **[1 point]** At (c): ☐ S(A) ☐ S(B) ☐ X(A) ☑ X(B) ☐ No lock needs to be requested

δ) **[1 point]** At (d): ☐ S(A) ☐ S(B) ☑ X(A) ☐ X(B) ☐ No lock needs to be requested

ϵ) **[1 point]** At (e): ☐ S(A) ☑ S(B) ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

ζ) **[1 point]** At (f): ☑ S(A) ☐ S(B) ☐ X(A) ☐ X(B) ☐ No lock needs to be requested

ii. **[4 points]** Which of the following schedule can cause a deadlock?

☑

T_1	S(A)			read(A)	S(B)	
T_2		S(B)	read(B)			X(A)

☐

T_1	S(A)			read(A)	S(B)	
T_2		S(B)	read(B)			S(A)

☑

T_1	X(A)			read(A)	S(B)	
T_2		X(B)	read(B)			S(A)

☑

T_1	S(A)			read(A)	X(B)	
T_2		S(B)	read(B)			X(A)

(b) Consider the following lock requests in Table 2. And note that

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, and T_3 represent three transactions.

- LM stands for 'lock manager'.
- Transactions will never release a granted lock.

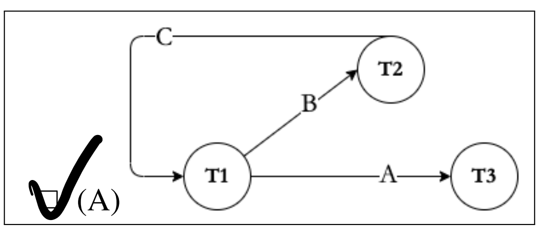
time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
T_1	S(C)				S(B)	X(B)	S(A)
T_2		S(B)		X(C)			
T_3			X(A)				
LM	g						

Table 2: Lock requests of three transactions

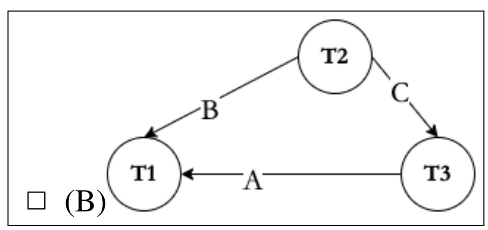
i. For the lock requests in Table 2, determine which lock will be granted or blocked by the lock manager. Please write 'g' in the LM row to indicate the lock is granted and 'b' to indicate the lock is blocked or the transaction has already been blocked by a former lock request. For example, in the table, the first lock (S(A) at time t_1) is marked as granted.

α) **[1 point]** At t_2: ☐ g ☐ b
β) **[1 point]** At t_3: ☐ g ☐ b
γ) **[1 point]** At t_4: ☐ g ☐ b
δ) **[1 point]** At t_5: ☐ g ☐ b
ϵ) **[1 point]** At t_6: ☐ g ☐ b
ζ) **[1 point]** At t_7: ☐ g ☐ b

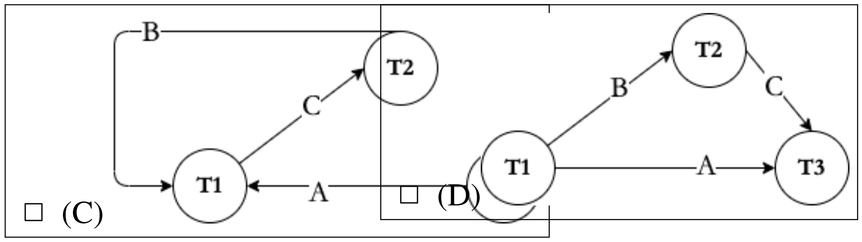
ii. **[2 points]** Mark the correct wait-for graph for the lock requests in Table 2. Each edge in the wait-for graph looks like this: $T_x \rightarrow T_y$ because of Z. Z is denoted in the arrow in the figure. (i.e., T_x is waiting for T_y to release its lock on resource Z).



☐ (A)



☐ (B)



☐ (C) ☐ (D)

iii. **[2 points]** Determine whether there exists a deadlock in the lock requests in Table 2. Mark all that apply.

☐ There is no deadlock

☐ Cycle ($T_3 \rightarrow T_1 \rightarrow T_3$) exists and schedule deadlocks

☐ There is no cycle

☐ ✓ Cycle ($T_1 \rightarrow T_2 \rightarrow T_1$) exists and schedule deadlocks

(c) **Deadlock Prevention:**
Consider the following lock requests in Table 3.
Like before,

- S(·) and X(·) stand for 'shared lock' and 'exclusive lock', respectively.
- T_1, T_2, T_3, T_4, and T_5 represent five transactions.
- LM represents a 'lock manager'.
- Transactions will never release a granted lock.

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	S(B)						S(A)	
T_2		S(D)			X(B)			
T_3			X(A)	X(D)				S(C)
T_4						X(C)		
LM	g	g						

Table 3: Lock requests of four transactions

i. For the lock requests in Table 3, determine which lock request will be granted, blocked or aborted by the lock manager (LM), if it has no deadlock prevention policy. *Please mark 'g' for grant, 'b' for block (or the transaction is already blocked), 'a' for abort, and '–' if the transaction has already died.*

α) **[1 point]** At t_3: ☐ g ☐ b ☐ a ☐ –

β) **[1 point]** At t_4: ☐ g ☐ b ☐ a ☐ –

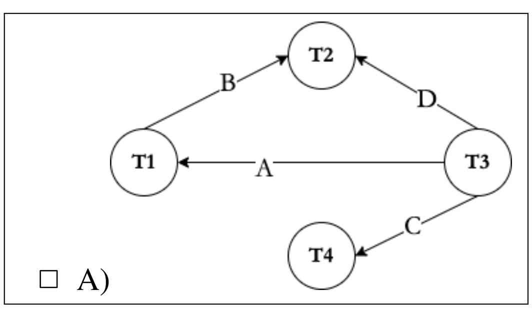
γ) **[1 point]** At t_5: ☐ g ☐ b ☐ a ☐ –

δ) **[1 point]** At t_6: ☐ g ☐ b ☐ a ☐ –

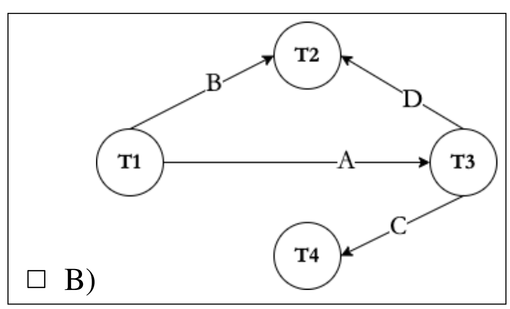
ϵ) **[1 point]** At t_7: ☐ g ☐ b ☐ a ☐ –

ζ) **[1 point]** At t_8: ☐ g ☐ b ☐ a ☐ –

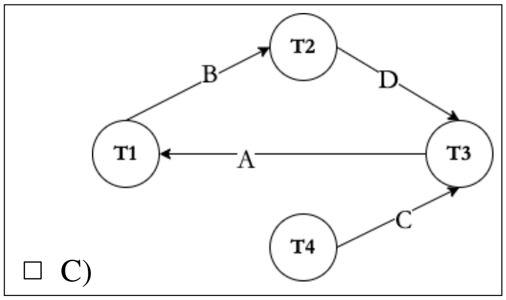
ii. **[2 points]** Mark the correct wait-for graph for the lock requests in Table 3. Each edge in the wait-for graph looks like this: $T_x \rightarrow T_y$ because of Z. Z is denoted in the arrow in the figure. (i.e., T_x is waiting for T_y to release its lock on resource Z).



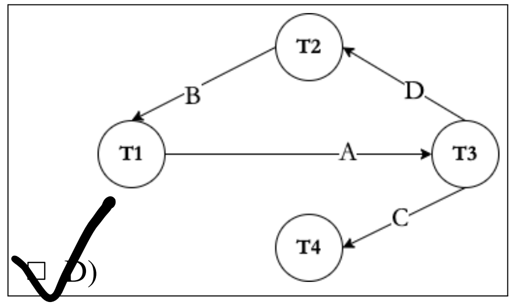
☐ A)



☐ B)



☐ C)



☐ D)

iii. **[2 points]** Determine whether there exists a deadlock in the lock requests in Table 3. Mark all that apply.

☐ There is no deadlock

☐ Cycle ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) exists and schedule deadlocks

☐ There is no cycle

☐ Cycle ($T_3 \rightarrow T_2 \rightarrow T_1 \rightarrow T_3$) exists and schedule deadlocks

等待死亡：非剥夺

iv. To prevent deadlock, we use the lock manager 高优先级 (LM) that adopts the Wait-Die policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Here, $T_1 > T_2$ because T_1 is older than T_2 (i.e., older transactions have higher priority). *Determine whether the lock request is granted ('g'), blocked ('b'), aborted ('a'), or already dead('–').* Follow the same format as the previous question.

α) **[1 point]** At t_3: ☐ g ☐ b ☐ a ☐ –

β) **[1 point]** At t_4: ☐ g ☐ b ☐ a ☐ –

γ) **[1 point]** At t_5: ☐ g ☐ b ☐ a ☐ –

δ) **[1 point]** At t_6: ☐ g ☐ b ☐ a ☐ –

ϵ) **[1 point]** At t_7: ☐ g ☐ b ☐ a ☐ –

ζ) **[1 point]** At t_8: ☐ g ☐ b ☐ a ☐ –

v. Now we use the lock manager (LM) that adopts the Wound-Wait policy. We assume that in terms of priority: $T_1 > T_2 > T_3 > T_4$. Here, $T_1 > T_2$ because T_1 is older than T_2 (i.e., older transactions have higher priority). *Determine whether the lock request is granted ('g'), blocked ('b'), granted by aborting another transaction ('a'), or the requester is already dead('–').* Follow the same format as the previous question.

α) **[1 point]** At t_3: ☐ g ☐ b ☐ a ☐ –

β) **[1 point]** At t_4: ☐ g ☐ b ☐ a ☐ –

γ) **[1 point]** At t_5: ☐ g ☐ b ☐ a ☐ –

δ) **[1 point]** At t_6: ☐ g ☐ b ☐ a ☐ –

ϵ) **[1 point]** At t_7: ☐ g ☐ b ☐ a ☐ –

ζ) **[1 point]** At t_8: ☐ g ☐ b ☐ a ☐ –

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	S(B)						S(A)	
T_2		S(D)			X(B)			
T_3			X(A)	X(D)				S(C)
T_4						X(C)		
LM	g	g						

Table 3: Lock requests of four transactions

Question 3: Hierarchical Locking .... [20 points]

Consider a database (D) consisting of two tables, Release (R) and Artists (A). Specifically,

- `Release(rid, name, artist_credit, language, status, genre, year, number_sold)`, spans 1000 pages, namely R_1 to R_{1000}
- `Artists(id, name, type, area, gender, begin_date_year)`, spans 50 pages, namely A_1 to A_{50}

页 条 一页100条

Further, **each page contains 100 records**, and we use the notation $R_3 : 20$ to represent the 20^{th} record on the third page of the Release table. Similarly, $A_5 : 10$ represents the 10^{th} record on the fifth page of the Artists table.

We use Multiple-granularity locking, with **S, X, IS, IX** and **SIX** locks, and **four levels of granularity**: (1) *database-level (D)*, (2) *table-level (R, A)*, (3) *page-level* ($R_1 - R_{1000}$, $A_1 - A_{50}$), (4) *record-level* ($R_1 : 1 - R_{1000} : 100$, $A_1 : 1 - A_{50} : 100$).

For each of the following operations on the database, check all the sequence of lock requests based on intention locks that should be generated by a transaction that wants to efficiently carry out these operations by maximizing concurrency. Please take care of efficiency for e.g., share vs. exclusive lock and granularity.

Please follow the format of the examples listed below:

- mark **"IS(D)"** for a request of **database-level IS lock**
- mark **"X(A_2 : 30)"** for a request of **record-level X lock for the 30^{th} record on the second page of the Artists table**
- mark **"S($A_2 : 30 - A_3 : 100$)"** for a request of **record-level S lock from the 30^{th} record on the second page of the Artists table to the 100^{th} record on the third page of the Artists table**.

(a) **[4 points]** Fetch the 70^{th} record on page R_{450}.

☐ `S(`$R_{450} : 70$`)`
☐ `IS(D), IS(`R_{450}`), S(`$R_{450} : 70$`)`
☑ `IS(D), IS(R), IS(`R_{450}`), S(`$R_{450} : 70$`)`
☐ `SIX(D), SIX(R), SIX(`R_{450}`), X(`$R_{450} : 70$`)`

(b) **[4 points]** Scan all the records on pages R_1 through R_{10}, and modify the record $R_{10} : 33$.

☐ `IX(D), IX(R), IX(`$R_1 - R_{10}$`), X(`$R_{10} : 33$`)`
☑ `IX(D), IX(R), S(`$R_1 - R_9$`), SIX(`R_{10}`), X(`$R_{10} : 33$`)`
☐ `IX(D), SIX(R), IX(`R_{10}`), X(`$R_{10} : 33$`)`
☐ `IX(D), SIX(R), IS(`R_{10}`), X(`$R_{10} : 33$`)`

(c) **[4 points]** Count the number of releases with 'year' > 2011.
- ☐ `IS(D), S(R)`
- ☐ `S(D), S(R)`
- ☐ `X(R)`
- ☐ `IS(D), X(R)`

(d) **[4 points]** Increase the number_sold of all release by 2021.
- ☐ `IX(D), IS(R),` X(R_{100})
- ☐ `IX(R)`
- ☐ `X(R), S(A)`
- ☐ `IX(D), X(R)`

(e) **[4 points]** Increase the artist_credit in release and id in artist by 1 for all the tuples in the respective tables.
- ☐ `X(D)`
- ☐ `S(D), IS(R), X(A)`
- ☐ `IX(D), X(R), X(A)`
- ☐ `IX(D), X(R), S(A)`

Question 4: Optimistic Concurrency Control . . . . [20 points]

Consider the following set of transactions accessing a database with object *A, B, C, D*. The questions below assume that the transaction manager is using **optimistic concurrency control** (OCC). Assume that a transaction switches from the READ phase immediately into the VALIDATION phase after its last operation executes.

Note: VALIDATION may or may not succeed for each transaction. If validation fails, the transaction will get immediately aborted.

You can assume that the DBMS is using the serial validation protocol discussed in class where only one transaction can be in the validation phase at a time, and each transaction is doing forward validation (i.e. Each transaction, when validating, checks whether it intersects its read/write sets with any active transactions that have not yet committed.)

前向{自己解或冲突解,

time	T_1	T_2	T_3
1	READ(A)		
2	READ(C)		
3		READ(B)	
4	WRITE(A)		
5			READ(B)
6	WRITE(C)		
7	VALIDATE?		
8		READ(D)	
9	WRITE?		
10		WRITE(D)	
11		WRITE(B)	
12		VALIDATE?	
13			READ(A)
14		WRITE?	
15			WRITE(A)
16			WRITE(B)
17			VALIDATE?
18			WRITE?

Figure 1: An execution schedule

(a) **[6 points]** Will T1 abort?
- ☐ Yes
- ☑ No

(b) **[6 points]** Will T2 abort?
- ☑ Yes
- ☐ No

(c) **[6 points]** Will T3 abort?
- ☐ Yes
- ☑ No

(d) **[2 points]** OCC is good to use when there are few conflicts.
- ☐ True
- ☐ False

CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2021)
PROF. LIN MA

Homework #5 (by Preetansh Goyal and Joseph Koshakow)
Due: **Thursday Dec 2, 2021 @ 11:59pm**

IMPORTANT:

- **Upload this PDF** with your answers to **Gradescope by 11:59pm on Thursday Dec 2, 2021**.
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.
- **You have to use this PDF for all of your answers.**

For your information:

- Graded out of **120** points; **4** questions total

Revision : 2021/11/26 16:04

Question	Points	Score
Write-Ahead Logging	35	
Replication	33	
Two-Phase Commit	40	
Miscellaneous	12	
Total:	120	

刷脏时未commit的刷一起

Question 1: Write-Ahead Logging..............................[35 points]

Consider a DBMS using write-ahead logging with physical log records with the STEAL and NO-FORCE buffer pool management policy. Assume the DBMS executes a non-fuzzy checkpoint where all dirty pages are written to disk.

Its transaction recovery log contains log records of the following form:

`<txnId, objectId, beforeValue, afterValue>`

The log also contains checkpoint, transaction begin, and transaction commit records.

The database contains three objects (i.e., A, B, and C).

The DBMS sees records as in Figure 1 in the WAL on disk after a crash.

Assume the DBMS uses ARIES as described in class to recover from failures.

LSN	WAL Record
1	<T1 BEGIN>
2	<T1, A, 6, 7>
3	<T1, B, 42, 43>
4	<T2 BEGIN>
5	<T2, C, 33, 71>
6	<T1 COMMIT>
7	<T2, B, 43, 100>
8	<T3 BEGIN>
9	<T3, A, 7, 20>
10	<T2, B, 100, 67>
11	<**CHECKPOINT**>
12	<T3, A, 20, 42>
13	<T2, C, 71, 13>
14	<T2 COMMIT>
15	<T3, A, 42, 66>

故障点

A 7
T2 redo
T3 undo

A 7
B 43 100 67
C 71 13

Figure 1: WAL

1. **[10 points]** What are the values of A, B, and C in the database stored on disk before the DBMS recovers the state of the database?

   ☐ A=6, B=100, C=71
   ☐ A=66, B=67, C=13
   ☐ A=7, B:Not possible to determine, C=43
   ☐ A=42, B=42, C=71
   ☐ A=20, B:43, C=Not possible to determine
   ☐ A=20, B:Not possible to determine, C=43
   ☐ A=20, B,C:Not possible to determine
   ☐ A:Not possible to determine, B=42 C=71
   ☐ A:Not possible to determine, B=67, C:Not possible to determine
   ☐ A,B,C:Not possible to determine

(b) **[5 points]** What should be the correct action on T1 when recovering the database from WAL?

- ☐ do nothing to T1
- ☐ redo all of T1's changes
- ☐ undo all of T1's changes

(c) **[5 points]** What should be the correct action on T2 when recovering the database from WAL?

- ☐ do nothing to T2
- ☐ redo all of T2's changes
- ☐ undo all of T2's changes

(d) **[5 points]** What should be the correct action on T3 when recovering the database from WAL?

- ☐ do nothing to T3
- ☐ redo all of T3's changes
- ☐ undo all of T3's changes

(e) **[10 points]** Assume that the DBMS flushes all dirty pages when the recovery process finishes. What are the values of A, B, and C after the DBMS recovers the state of the database from the WAL in Figure 1?

- ☐ A=6, B=42, C=33
- ☐ A=66, B=67, C=13
- ☐ A=6, B=100, C=13
- ☐ A=7, B=67, C=13
- ☐ A=20, B=42, C=71
- ☐ A=42, B=100, C=33
- ☐ A=7, B=100, C=71
- ☐ A=42, B=67, C=13
- ☐ A=20, B=43, C=33
- ☐ A=66, B=43, C=71
- ☐ A=42, B=42, C=13
- ☐ Not possible to determine

Question 2: Replication .... [33 points]

Consider a DBMS using active-passive, master-replica replication with multi-versioned concurrency control. All read-write transactions go to the master node (NODE A), while read-only transactions are routed to the replica (NODE B). You can assume that the DBMS has "instant" fail-over and master elections. That is, there is no time gap between when the master goes down and when the replica gets promoted as the new master. For example, if NODE A goes down at timestamp ① then NODE B will be elected the new master at ②.

The database has a single table `foo(id,val)` with the following tuples:

id	val
1	x
2	y
3	z

zz

yyy

Table 1: **`foo(id,val)`**

For each questions listed below, assume that the following transactions shown in Figure 2 are executing in the DBMS: (1) Transaction #1 on NODE A and (2) Transaction #2 on NODE B. You can assume that the timestamps for each operation is the real physical time of when it was invoked at the DBMS and that the clocks on both nodes are perfectly synchronized.

time	operation
①	`BEGIN;`
②	`UPDATE foo SET val = 'xx';`
③	`UPDATE foo SET val = 'yyy' WHERE id = 3;`
④	`UPDATE foo SET val = 'zz' WHERE id = 1;`
⑤	`COMMIT;`

(a) Transaction #1 – NODE A

time	operation
②	`BEGIN READ ONLY;`
③	`SELECT val FROM foo WHERE id = 3;`
④	`SELECT val FROM foo WHERE id = 1;`
⑤	`SELECT val FROM foo WHERE id = 1;`
⑥	`COMMIT;`

(b) Transaction #2 – NODE B

Figure 2: Transactions executing in the DBMS.

(a) Assume that the DBMS is using *asynchronous* replication with *continuous* log streaming (i.e., the master node sends log records to the replica in the background after the transaction executes them). Suppose that NODE A crashes at timestamp ⑤ before it executes the `COMMIT` operation.

1. **[10 points]** If Transaction #2 is running under `READ COMMITTED`, what is the return result of the `val` attribute for its `SELECT` query at timestamp ⑤? Select all that are possible.

   ☐ zz
   ☐ x
   ☐ y
   ☐ yyy
   ☐ z

☐ xx
☐ None of the above

i. **[10 points]** If Transaction #2 is running under the `READ UNCOMMITTED` isolation level, what is the return result of the `val` attribute for its `SELECT` query at timestamp ⑤? Select all that are possible.

☐ zz
☐ x
☐ `y
☐ yyy
☐ z
☐ xx
☐ None of the above

(b) **[13 points]** Assume that the DBMS is using *synchronous* replication with *on commit* propagation. Suppose that both NODE A and NODE B crash at exactly the same time at timestamp ⑥ after executing Transaction #1's `COMMIT` operation. You can assume that the application was notified that the Transaction #1 was committed successfully.

After the crash, you find that NODE A had a major hardware failure and cannot boot. NODE B is able to recover and is elected the new master.

What are the values of the tuples in the database when the system comes back online? Select all that are possible.

☐ { (1,x), (2,y), (3,z) }
☐ { (1,xx), (2,xx), (3,xx) }
☐ { (1,xx), (2,xx), (3,yyy) }
☐ { (1,zz), (2,xx), (3,yyy) }
☐ { (1,x), (2,xx), (3,z) }
☐ { (1,x), (2,xx), (3,xx) }
☐ None of the above

Question 3: Two-Phase Commit .... [40 points]

Consider a distributed transaction T operating under the two-phase commit protocol. Let N_0 be the *coordinator* node, and N_1, N_2, N_3 be the *participant* nodes.

The following messages have been sent:

time	message
1	N_0 to N_1: "**Phase1:PREPARE**"
2	N_1 to N_0: "**OK**"
3	N_0 to N_2: "**Phase1:PREPARE**"
4	N_0 to N_3: "**Phase1:PREPARE**"

Figure 3: Two-Phase Commit messages for transaction T

(a) **[10 points]** Who should send a message next at time 5 in Figure 3? Select *all* the possible answers.

- ☐ N_0
- ☐ N_1
- ☑ N_2
- ☑ N_3
- ☐ It is not possible to determine

(b) **[10 points]** To whom? Again, select *all* the possible answers.

- ☑ N_0
- ☐ N_1
- ☐ N_2
- ☐ N_3
- ☐ It is not possible to determine

(c) **[10 points]** Suppose that N_0 received the "**ABORT**" response from N_2 at time 5 in Figure 3. What should happen under the two-phase commit protocol in this scenario?

- ☐ N_0 resends "**Phase1:PREPARE**" to N_2
- ☐ N_2 resends "**OK**" to N_0
- ☐ N_0 sends "**Phase2:COMMIT**" all of the participant nodes
- ☑ N_0 sends "**ABORT**" all of the participant nodes
- ☐ N_0 resends "**Phase1:PREPARE**" to all of the participant nodes
- ☐ It is not possible to determine

(d) **[10 points]** Suppose that N_0 <u>successfully</u> receives all of the "**OK**" messages from the participants from the first phase. It then sends the "**Phase2:COMMIT**" message to all of the participants but N_1 and N_3 crash before they receives this message. What is the status of the transaction T when N_1 comes back on-line?

☐ T's status is *aborted*
☐ T's status is *committed*
☐ It is not possible to determine

Question 4: Miscellaneous .... [12 points]

(a) **[4 points]** With consistent hashing, if a node fails then all keys must be reshuffled among the remaining nodes.

- ☑ True
- ☐ False

(b) **[4 points]** For a DBMS that uses ARIES, all updated pages must be flushed to disk for a transaction to commit.

- ☐ True
- ☑ False

(c) **[4 points]** During the undo phase of ARIES, all transactions that committed after the last checkpoint are undone.

- ☐ True
- ☑ False