CS127 Quiz 3 Prep

Quiz 3: December 4th, 2019 3:00 P.M.

This ungraded handout is to help you prepare for Quiz 3. Answers will be posted later.

HW5 and its solutions are also relevant to the quiz and should be reviewed with this quiz prep.

In answering these questions, if you don't understand the slides, the answers can be found in the textbook (except for the questions about the recovery algorithm, which follows the fourth edition of the textbook).

Topic #1 ACID Properties

1. Transactions
   (a) What is a transaction?

   (b) How is it indicated in SQL?

2. Atomicity and Durability
   (a) How does the recovery system create *atomicity*?

   (b) How does the recovery system create *durability*?

3. Isolation and Consistency
   (a) What is the definition of *isolation* (in the context of databases)?

   (b) When must isolation be enforced?

   (c) The concurrency control system creates [the appearance of] isolation for concurrent transactions. How does it do this?

   (d) What is the definition of *consistency* (in the context of databases) and who is responsible for it?

Topic #2 Serial Equivalence

1. Concurrency's Benefits
   How is using concurrency in transaction processing beneficial? Give two reasons.

2. Serializability
   The schedule in Figure 1 is serializable even though it meets neither the definition of conflict serializable nor the definition of view serializable.

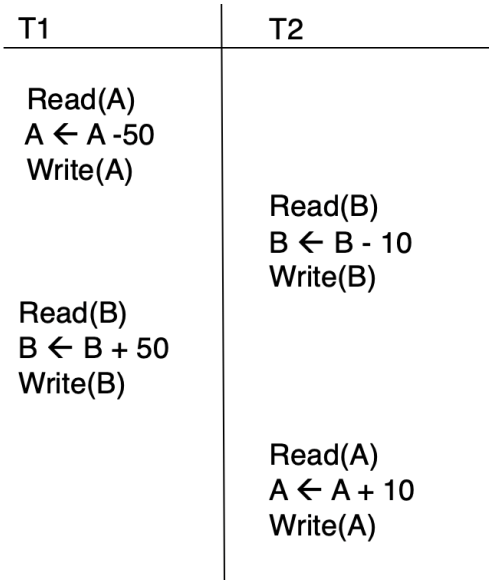

Figure 1: Serializable Schedule

(a) Why is it not conflict serializable?

(b) Why is this not view serializable?

(c) Why, then, is the schedule serializable?

(d) What property of addition and subtraction does the fact that this is a serializable schedlue even though it is neither view nor conflict serializable demonstrate?

(e) What other arithmetic operations have this same property?

3. View Serializability
The schedule in Figure 2 is view serializable.

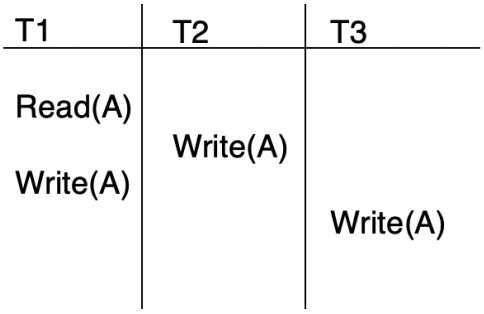

Figure 2: View Serializable Schedule

(a) What serial schedule is it equivalent to?

(b) Why is it not conflict serializable?

(c) In the above schedule, add a Read(A) instruction to T3 before T3's Write(A) instruction such that it is still view serializable.

(d) In the above schedule, can a read(A) instruction be added to T2 before T2 write(A) without breaking view serializability?

(e) Explain how the 3 rules of view serializability ensure that the scheudle is serializable.

(f) How is a conflict serializable schedule more similar to a serial schedule than a view serializable schedule?

(g) Are the schedules in figures 3 and 4 conflict serializable? If so, which serial schedule are they conflict equivalent to?

T1	T2	T3
(1) Read(A)		
	(a) Write(A)	
	(b) Read(B)	
		(x) Write(B)
		(y) Read(S)
(2) Write(S)		

Figure 3: Problem 2.4.c, Schedule 1

T1	T2
1. Read(A)	
2. Write(A)	
	a. Read(A)
	b. Write(A)
3. Read(B)	
4. Write(B)	
	c. Read(B)
	d. Write(B)

Figure 4: Problem 2.4.c, Schedule 2

(h) Create a precedence graph for the above two schedules. Explain how this graph can help you to determine conflict serializability.

Topic #3 Recoverability and Cascadelessness

1. Which of the below schedules is recoverable?

2. Modify whichever schedule is recoverable to make it cascadeless.

Topic #4 Lock Based Protocols

1. Motivation and basics

(a) Why are lock-based protocols necessary?

T1	T2
Write(A)	
	Read(A)
	Write(A)
	Commit
Commit	

Figure 5: Problem 3.1, Schedule 1

T1	T2
Write(A)	
	Read(A)
	Write(A)
Commit	
	Commit

Figure 6: Problem 3.1, Schedule 2

(b) What are the two types of locks, and what do they allow?

2. Two-phase locking

(a) What does two-phase locking guarantee?

(b) Does the partial schedule in Figure 7 adhere to two phase locking? If it does not adhere to two phase locking, then change it to adhere to two phase locking, and say whether the result is conflict serializable.

T_1	T_2
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)
	lock-S(B)
	read(B)
	unlock(B)
	display($A + B$)
lock-X(A)	
read(A)	
$A := A - 50$	
write(A)	
unlock(A)	

Figure 7: Problem 4.2.b: Two-phase locking

(c) What does strict two phase locking guarantee?

(d) Adjust the result of whatever you answered to part (c) to adhere to strict two phase locking.

3. Starvation

   What protocol must be adhered to to prevent starvation of a transaction?

4. Deadlock avoidance protocols

   (a) The following two partial schedules attempt to use two phase locking but run into a lock conflict.

T1	T2
Lock-S(A)	
Read(A)	
	Lock-X(B)
	Read(B)
Lock-S(B)	
Unlock(A)	
Read(B)	
	Write(B)
	Unlock(B)
Unlock(B)	

Figure 8: Problem 4.4.a, Schedule 1

T1	T2
Lock-S(A)	
Read(A)	
	Lock-X(B)
	Read(B)
Lock-S(B)	
Unlock(A)	
Read(B)	
	Write(B)
	Lock-X(A)
	Unlock(B)
	Read(A)
	Write(A)
	Unlock(A)
Unlock(B)	

Figure 9: Problem 4.4.a, Schedule 2

   i. If using the wait die protocol, how is this resolved?

   ii. If using the wound-wait protocol, how is it resolved?

   iii. The above protocols have the disadvantage of potentially causing unnecessary rollbacks. To avoid this, you might want to only issue a rollback in the case of deadlock. To determine when you have reached deadlock, you draw a dynamic wait-for graph and run cycle detection on it.

   Draw the wait for graphs of the Schedule 1 at the time of the instruction T2 write(B), and the wait-for graph of Schedule 2 at the time of the instruction T2 lock-X(A). Determine from these graphs which of the above schedules has deadlock. (Note: In a wait-for graph, T1 points to T2 if T1 requires a lock that T2 holds).

Topic #5 Timestamp Based Protocols

Timestamp ordering is another way of guaranteeing serializability. The serial equivalent of a given schedule is a serial schedule that is in the order of the timestamps assigned to each transaction. Transactions that enter the system are assigned a unique timestamp, based on a logical counter or the system clock. Transactions

that enter the system after the schedule has begun are issued timestamps when they enter which are greater than the timestamps of transactions already in the system.

In the following schedules, T1 starts with an initial timestamp smaller than T2. Attempt to implement the following schedules following the timestamp protocol outlined in the slides and in the textbook (p. 683). If a transaction gets rolled back and restarted at a later timestamp, retain its original identifier (i.e. "T1" is still called "T1" even after it's been rolled back and restarted). When relevant, use Thomas's write rule.

What serial schedule do each of the following partial schedules end up being equivalent to (assuming that the only instructions not shown here are commits)?

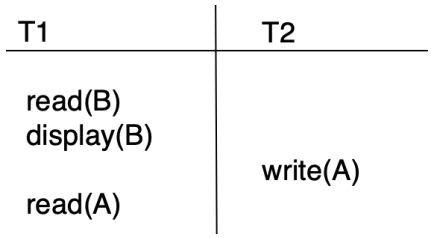

Figure 10: Problem 5, Schedule 1

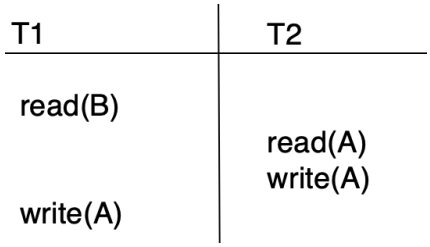

Figure 11: Problem 5, Schedule 2

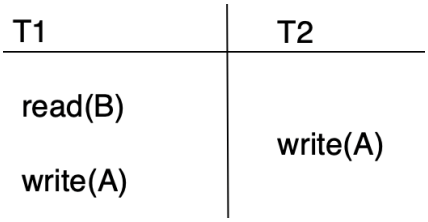

Figure 12: Problem 5, Schedule 3

Topic #6 Recovery

1. Commits
   What defines a commit?

2. Immediate and Deferred Database Modification

   (a) In immediate database modification, what happens when a write command is issued?

   (b) How does this explain why it is necessary to undo or redo writes when there is a system failure?

   (c) If we use deferred database modification, is it necessary to undo writes?

3. Logging
   There are 4 types of log entries used in a database log that we studied in this course:

   1. $< T_i$ start$>$
   2. $< T_i, X_j, V_1, V_2 >$ (this is an update entry)
   3. $< T_i$ commit$>$

4. $< T_i$ checkpoint>

Why is V_1 in an update entry?

4. Recovery algorithm:
The lecture slides slides give the following algorithm for recovering from system failures.

(1) Initialize an undo list and a redo list

(2) Scan the log from the end backward until the first encountered checkpoint, adding a transaction's ID to the redo list when a transaction commit is encountered, and adding a transaction ID to the undo list when a transaction start is encountered and the transaction is not in the redo list.

(3) Pass backward from the end of the log undoing updates by transactions in the undo list. When a start entry for a transaction in the undo list is encountered, remove the transaction ID from the undo list. Continue until the undo list is empty.

(4) Pass forward from the last checkpoint until the end of the log and redo (in memory) each log entry from a transaction in the redo list.

(a) Given the following log:

Beginning of log
<T_0 start>
<T_0, B, 2000, 2050>
<T_1 start>
<checkpoint {T_0, T_1}>
<T_1, C, 700, 600>
<T_1 commit>
<T_2 start>
<T_2, A, 500, 400>

Figure 13: Problem 6.4.a, Database Log

i. After the analysis pass, which transaction IDs are in the undo list and which are in the redo list?

ii. What are the values of A, B, and C after the algorithm has finished?

iii. Does it meet the requirements of 2 phase locking?

iv. Now suppose that the system fails at the end of the above schedule. T1 must be undone, and T2 redone.
If step 4 in the above recovery algorithm came before step 3, what error would result from this and how is this avoided by undoing before redoing?

v. Now, undoing before redoing is in accordance with the 4^{th} edition of the textbook (from 2004), p 659, however, according to the $5th$ and 6^{th} editions of the textbook, ($6th$ edition, pp. 753-755), redoing comes before undoing. If so, then to avoid the above problem, a recovery algorithm must require that if a data item has been modified by a transaction, no other transaction may modify that data item until the first transaction either is undone or is committed. What locking protocol may be used to ensure this?

vi. [Note: The following questions further examine the discrepancies between the textbook's recovery algorithm and the slides' recovery algorithm, and will afford you a broader understanding of recovery, but will not be tested on in quiz 3.]

Even if the above requirement is met, if redo comes before undo, an error may arise. Consider the schedule shown in figure 15.

$<T_i,\ A,\ 10,\ 20>$
$<T_j,\ A,\ 10,\ 30>$
$<T_j$ commit$>$

Figure 14: Problem 6.4.b.v, A Schedule

This schedule implies that T_i was rolled back before T_j modified A. If the system now fails, then T_i must be undone a second time, and T_j redone.

If redo comes before undo, what problem does this cause?

One way to solve this is to keep track of transactions that have already been undone, so that they will not be undone a second time. The later versions of the textbook implement this by logging a $< T_i$ abort $>$ entry when a transaction has been aborted. During the analysis phase, transactions that have a $< T_i$ abort $>$ entry are omitted from the undo list.

vii. Another key difference between the recovery algorithm given in the later versions of the textbook and the recovery algorithm given in the slides is that the algorithm in the later editions redoes everything in the log from the last checkpoint forward so as to simplify the recovery process, rather than implementing a redo list. This is called "redoing history." However, this would seemingly cause an error when transactions in the log have been undone. Explain that error.

viii. To resolve this, the later editions say to log a "redo-only" log entry of the form $< T_i,\ X_j,\ V >$, indicating that data item X_j has been restored to value V, after undoing an update. This is called a "redo-only" entry because it cannot be undone, since no original value is given. For example, in the schedule in figure 15, $< T_i,\ A,\ 10 >$ is entered into the log when T_i is rolled back. The log at the time of the system crash looks like this:

$< T_i,\ A,\ 10,\ 20 >$
$< T_i,\ A,\ 10 >$
$< T_i$ abort$>$
$< T_j,\ A,\ 10,\ 30 >$
$< T_j$ commit$>$

Explain how this resolves the problem mentioned above.

A complete outline of the $6th$ edition's recovery algorithm can be found on pp. 736-37.

Further Resources for Quiz 3 Preparation

Once you have completed the above questions, the following textbook exercises will help you to absorb the material even better. The solutions can be found at https://www.db-book.com/db6/practice-exer-dir/

index.html. If you have limited time, it is recommended that you look at each question, understand what it is asking and its relevance to the material on the quiz, and then review the correct answer, rather than spending the time to come up with the correct answer by yourself.

1. Ch. 14: 1, 5, 6, 7, 11
2. Ch 15: 1, 2, 11
3. Ch. 16: 1, 2, 5, 7