CS 245 Final Test
Winter 2021

- Please read all instructions carefully. In the case of any ambiguity, state any assumptions you made in your answer. You may also ask clarifying questions in a **private** question on Piazza, but note that the course staff may take several hours to answer these.

- There are five problems, some with multiple parts, for a total of **77** points. You have until 11:59 PM Pacific on Thursday, March 18th to work on the test. **Late submissions will not be accepted past 11:59 PM Pacific on March 20th.**

- The test is open-book, but you are not allowed to communicate with other people to do it. This includes asking public questions on Piazza -- please only ask private questions about the test or any content covered in it. You may also use the Internet during the test, but keep in mind that many online resources might use terms differently from our course, and that directly copying an online resource is considered plagiarism and is not allowed. We don't think you will need resources other than the course materials.

- You may write answers digitally on the test PDF, or print it, handwrite your answers legibly, and scan it using a scanner or a mobile app such as GeniusScan. We'll also provide a Word version of the test that you can fill if you are willing to mark your answer locations when you upload to GradeScope. In either case, ensure that the file you upload to Gradescope matches the format of this document and is sharp and aligned.

- Solutions will be graded on correctness and clarity. For the long-answer problems, please show your intermediate work. Each problem has a relatively simple to explain solution, and we may deduct points if your solution is much more complex than necessary. Partial solutions will be graded for partial credit.

NAME: _____

SUID: _____

In accordance with both the letter and spirit of the Stanford Honor Code, I have neither given nor received assistance on this test. A typed signature is fine.

SIGNATURE: _____

Problem 1: Recovery and Logging (12 points)

a) (6 points) Answer the following assuming standard-case database operation. *Explain your answers as well, using 1-2 sentences.*

i) Is undo logging or redo logging more memory efficient?

Undo logging, as redo logging requires keeping values in memory until commit.

ii) Which logging scheme is more disk efficient when many transactions affect the same value repeatedly: undo logging or redo logging?

Redo logging, as only the latest value needs to be written.

iii) Name one **drawback** of using undo/redo logging over the other logging schemes we studied (and explain why it is a drawback).

Undo/redo logging requires storing two values in the log (as opposed to one), which takes more disk space for the log.

Undo/redo logging requires two passes for recovery.

etc.

b) (2 points) Assume the database crashes immediately after line 8 for the undo log below (and there are no checkpoints). How many values must be modified when recovering from the undo log? List them all below the log.

Entry number	**Undo log**
1	<T1, start>
2	<T1, A, 0>
3	<T2, start>
4	<T3, start>
5	<T2, B, 0>
6	<T1, D, 0>
7	<T3, C, 0>
8	<T3, commit>

Three values, D, B and A.

c) (4 points) Consider the undo log below. List all the row numbers after which truncating the log is **valid** with naive checkpointing. In other words, select all the entry numbers where the system could perform a naive checkpoint and truncate the log immediately after that log entry. Feel free to describe your reasoning below the table for partial credit.

Entry number	**Undo log**	**Valid to checkpoint & truncate log?**
1	<T1, start>	
2	<T1, A, 0>	
3	<T2, start>	
4	<T3, start>	
5	<T2, B, 0>	
6	<T1, D, 0>	
7	<T3, C, 0>	
8	<T3, commit>	
9	<T2, commit>	
10	<T1, commit>	x
11	<T4, start>	
12	<T4, C, 0>	
13	<T4, commit>	x
14	<T5, start>	
15	<T5, A, 1>	
16	<T5, commit>	x

Problem 2: Transaction Schedules (14 points)

a) (2 points) For each statement below, indicate whether it's true or false.

i. All conflict serializable schedules are ACR. TRUE FALSE

ii. Recoverable schedules can result in dirty reads. TRUE FALSE

False, True

In the following parts, consider the transactions T_1 and T_2:

T_1: $r_1(B)$; $w_1(A)$; $r_1(A)$ T_2: $w_2(B)$; $r_2(A)$

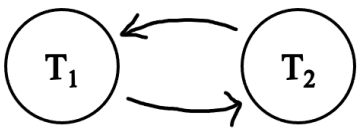
We use the notation from lecture, where $r_i(A)$ and $w_i(A)$ mean transaction T_i reads and writes A. Let A=5 and B=11 before either transaction starts. $w_1(A)$ writes A=14 and $w_2(B)$ writes B=20.

b) (2 points) Given the serial schedules below, fill in the values that each operation reads:

S_1	$r_1(B)$	$w_1(A)$	$r_1(A)$	$w_2(B)$	$r_2(A)$
	____	14	____	20	____
S_2	$w_2(B)$	$r_2(A)$	$r_1(B)$	$w_1(A)$	$r_1(A)$
	20	____	____	14	____

S_1: 11, 14, 14. S_2: 5, 20, 14.

c) (2 points) Write a schedule S_3 using transactions T_1 and T_2 with the precedence graph below. Label the conflicting operations in the schedule.



$r_1(B)$ $w_2(B)$ $r_2(A)$ $w_1(A)$ $r_1(A)$
$T_1 \rightarrow T_2$ conflict: $r_1(B) \rightarrow w_2(B)$
$T_2 \rightarrow T_1$ conflict: $r_2(A) \rightarrow w_1(A)$

d) (6 points) We introduce additional notation, c_i, which indicates that transaction T_i commits. For each part, use the notation for reads, writes, and commits to fill in a schedule for T_1 and T_2 that has the given property, as indicated below. Both transactions must commit. There may be multiple solutions. Write ***1-2 sentences*** explaining why each schedule has the property.

i. Recoverable but not serializable.

$r_1(B)$ _____ _____ _____ _____ _____ c_2

$r_1(B)$ $w_2(B)$ $r_2(A)$ $w_1(A)$ $r_1(A)$ c_1 c_2
The schedule is recoverable because neither transaction reads from the other. It is not serializable because $r_1(B) = 10$ while $r_2(A) = 5$, which is equivalent to neither serial schedule from part b).

ii. Serializable but not recoverable.

$r_1(B)$ _____ $r_1(A)$ _____ _____ _____ _____

$r_1(B)$ $w_1(A)$ $r_1(A)$ $w_2(B)$ $r_2(A)$ c_2 c_1
The schedule is serial and therefore serializable. It is not recoverable because T_2 reads from T_1, but commits before T_1.

iii. Recoverable but not ACR.

$r_1(B)$ _____ _____ _____ _____ _____ _____

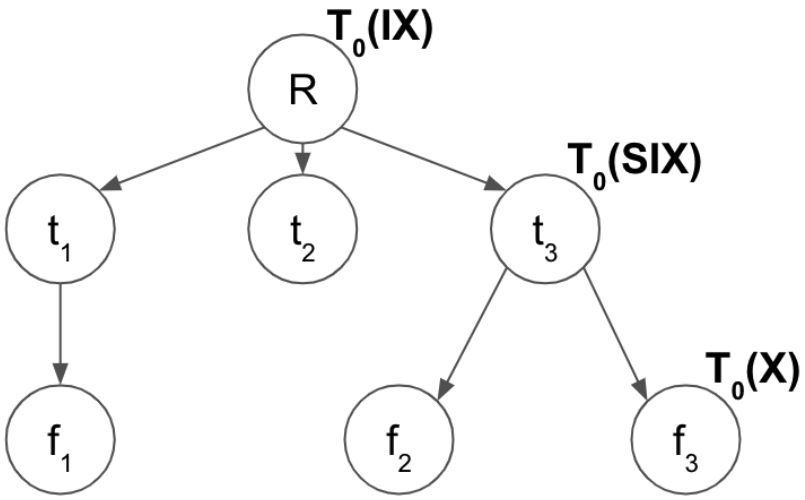
$r_1(B)$ $w_1(A)$ $r_1(A)$ $w_2(B)$ $r_2(A)$ c_1 c_2 (multiple solutions)
The schedule is recoverable because T_2 reads A from T_1 and commits after T_1, and T_1 does not read from T_2. It is not ACR because $r_2(A)$ occurs before c_1.

e) (2 points) If a schedule consisting of two transactions T_1 and T_2 is ACR, does it also have to be strict? If yes, briefly explain why ***(1-2 sentences)***. Otherwise, give a counterexample.

Yes, strict has the additional requirement that transactions can only write items that have been previously written by committed transactions. T_1 and T_2 write different keys, so all ACR schedules must be strict.
In the general case, no, since strict is a subset of ACR e.g. $w_1(A)$ $w_2(A)$ c_1 c_2.

Problem 3: Locking (15 points)

a) (6 points) Consider a database system holding the records shown below that implements hierarchical locking:



Suppose that this database is currently running only one transaction T_0. **This transaction has already acquired the locks it needs as shown in the figure.** For each of the other transactions below, indicate whether it can successfully acquire the locks it needs and run. If yes, provide the locks it acquires; otherwise explain the reason. *Note: Consider each transaction below separately, all based on the locks T_0 is holding at the start.*

T_1: Read f_2

Yes. IS(R), IS(t_3), S(f_2)

T_2: Insert f_4 under t_3

No.

T_3: Delete t_2

Yes. IX(R), X(t_2)

b) (3 points) Which of these statements about locking are true? *(Mark all that are true.)*

__X__ Reads and writes can be performed at any point during 2-Phase locking as long as the transaction is well-formed.

____ Coarse grain locks improve concurrency more than fine grain locks.

____ Multiple transactions can simultaneously hold an update lock for the same record.

c) (6 points) Complete the compatibility matrix below to add intention increment (II) and increment (I) modes to the hierarchical locking scheme discussed in lecture. Assume that incrementing a value is an atomic action, meaning that transactions can read, increment, and write the value in one operation without conflicting. For each cell marked as True in the II and I holder rows, explain why the holder and requester modes are compatible. Please use the following format for your explanations:

(Holder, Requester): Explanation for why the cell is marked as True.

		Requester						
		IS	**IX**	**S**	**SIX**	**X**	**II**	**I**
Holder	**IS**	**True**	**True**	**True**	**True**	**False**	**True**	**False**
	IX	**True**	**True**	**False**	**False**	**False**	**True**	**False**
	S	**True**	**False**	**True**	**False**	**False**	**False**	**False**
	SIX	**True**	**False**	**False**	**False**	**False**	**False**	**False**
	X	**False**	**False**	**False**	**False**	**False**	**False**	**False**
	II	**True**	**True**	**False**	**False**	**False**	**True**	**True**
	I	**False**	**False**	**False**	**False**	**False**	**True**	**True**

(II, IS) and (II, IX): intentions don't conflict because locking is handled at a lower level
(II, II), (I, I), (II, I), and (I, II): Multiple transactions can increment the same value because addition is commutative.

Problem 4: Optimistic Concurrency (8 points)

Consider the operation sequence of transactions in a database on the next page. R(X) means read X and W(X) means write X. The database uses validation (optimistic concurrency) for concurrency control. ***Note:*** *A transaction may succeed or fail at the validation phase, so ignore the "Finish" events in the table for transactions that you don't think will validate.*

Circle the correct answer of the following questions and briefly explain if the transaction aborts.

a) (2 points) T1 will

Succeed Abort

b) (2 points) T2 will

Succeed Abort

c) (2 points) T3 will

Succeed Abort (It will conflict with T2.)

d) (2 points) T4 will

Succeed Abort

Time	T1	T2	T3	T4
1	R(B)			
2	W(B)			
3		W(C)		
4		R(A)		
5	R(C)			
6	Validate			
7			R(C)	
8		W(A)		
9		Validate		
10			W(C)	
11			R(D)	
12			W(D)	
13	Finish			
14				R(D)
15			Validate	
16			Finish	
17		Finish		
18				W(C)
19				Validate
20				Finish

Problem 5: Distributed Databases (12 points)

All parts of this question will consider the 2PC protocol. Let us make the following assumptions about our system:

1. All failures involve hosts halting with their disks and logs intact.
2. Failed hosts reboot.
3. There is no network message loss (if a server sends a message and the target crashes before responding, the server will resend the message after a reboot).

Suppose there is a coordinator C and two participants P1 and P2.

a) (3 points) Let us attempt a transaction with 2PC. The following sequence of events happens:

C sends *Prepare Transaction T1* to P1
C sends *Prepare Transaction T1* to P2
P1 sends *Prepared* to C.
P2 sends *Prepared* to C.
C sends *Commit Transaction T1* to P1
C sends *Commit Transaction T1* to P2.
P2 *crashes*.
P2 *recovers*.

Will T1 commit on P1? What about P2? Carefully read the assumptions above!

Yes and yes.

b) (3 points) Let us attempt the same transaction *without* 2PC. The following sequence of events happens:

C sends *Execute Transaction T1* to P1
C sends *Execute Transaction T1* to P2
P1 *prepares* the transaction
P1 *commits* the transaction

Write down one additional event that would guarantee that P1 and P2 have an inconsistent state, given the above assumptions.

P2 aborts. "P2 crashes" was only accepted if the answer described a scenario that would lead to an abort--such as a crash before the transaction was written to the log--and explicitly said so.

c) (6 points) Let us say T1 commits successfully on both P1 and P2. Let us try another transaction, again *with* 2PC:

C sends *Prepare Transaction T2* to P1
C sends *Prepare Transaction T2* to P2
P1 sends *Prepared* to C
P2 sends *Prepared* to C
C sends *Commit Transaction T2* to P1
C *crashes*
P1 *commits*

Assume C does not recover.

i. What transactions have committed on P1? On P2? Are they consistent?

T2 has committed on P1 but not P2. They are not consistent.

ii. Which of the assumptions outlined above does this scenario violate?

C never recovers.

iii. What is one potential solution to scenarios such as this?
Elect a new coordinator.

Problem 6: Real-World Data Systems (16 points)

a) (2 points) Amazon Aurora performed significantly better than replicated MySQL over Elastic Block Storage (EBS) disks for write operations, even though it is based on the MySQL engine. How does Aurora reduce **write amplification** compared to running a primary-backup database over replicated disks, and how does this improve write performance?

Aurora only replicates the redo log over the network, instead of also replicating disk blocks over the network as in the primary-backup system.

b) (6 points) Cloud object stores such as Amazon S3 provide limited consistency guarantees, but bolt-on systems like Delta Lake can implement stronger guarantees on top. Consider the following sequence of transactions against a Delta Lake table that is initially empty:

1. Transaction T1 writes object a.parquet in the table's data directory and then adds log entry 1.json in the table's log directory to say that a.parquet was added to the table.
2. After T1 ends, transaction T2 writes object b.parquet in the table's data directory and then adds log entry 2.json in the log directory to say that b.parquet was added to the table.
3. After T2 ends, transaction T3 attempts to read the entire table.

i. Suppose that T3 begins by running an S3 LIST operation against the entire table. Which of the following results could that operation return? *Circle all options that S3 might return.*

Option 1:	Option 2:	Option 3:	Option 4:
a.parquet b.parquet _delta_log/1.json _delta_log/2.json	a.parquet b.parquet _delta_log/1.json	b.parquet _delta_log/1.json _delta_log/2.json	a.parquet _delta_log/2.json

ii. Now, suppose that T3 proceeds to read the table via Delta Lake's read protocol, using the snapshot isolation level. Which of these sets of files might Delta return? *(Check all that apply.)*

- ☐ a.parquet and b.parquet
- ☐ a.parquet only
- ☐ b.parquet only
- ☐ an empty result (no data)

iii. When clients write a log checkpoint in Delta Lake, such as 2.parquet, they only update the _last_checkpoint file after writing the checkpoint. This means that the client might crash before updating _last_checkpoint. Why is it OK for this file to sometimes point to an old checkpoint?

If this file is not updated, clients will still start reading the log at an older checkpoint and discover new log entries after it when they run a LIST operation.

c) (4 points) All distributed systems have to contend with the CAP theorem. Which of the C, A and P properties do each of the following systems provide? *(Check the boxes for each one.)*

System Name	**C**	**A**	**P**
Amazon S3		X	X
Amazon Aurora	X		X
Delta Lake	X		X
Spark Structured Streaming	X		X

d) (2 points) Which of the following statements about time and watermarks in Google Dataflow are true? *(Check all that apply.)*

- □ For any data record, its processing time is always ≤ its event time.
- □ For any data record, its processing time is always ≥ its event time.
- □ Once the watermark passes a given event time T, records with event time less than T will be ignored by the system.
- ☒ A query can have both processing-time-based and event-time-based triggers.

e) (2 points) Most modern database systems can use SQL views to implement access control, similar to System R. Which of the following security policies can be achieved by creating a SQL view and granting access to it just to specific users? *(Check all that apply.)*

- ☒ Alicia can only read the date and amount_sold fields in the "sales" table.
- □ Bob can only insert records in the sales table with amount_sold values less than 1000.
- ☒ Chen can only read the amount field in sales records where the country field is "USA".
- □ Dan can read the "sales" and "returns" tables separately but is not allowed to join them.

NOTE: some systems allow inserts on views so we accepted the second item too.