CS145 Transaction Homework

Instructions

- ANSWERS are included below the question, in white text.
    - E.g. Highlight 42 this text.
- Try to solve each problem yourself before looking at its solution.

1. Disk v main memory

You are working as the administrator for a database, and you are responsible for performance tuning. Your database server has a disk which can perform sequential I/O at 170MB/s, but when writing random 4KB chunks it only operates at 1MB/s.

Assume your database receives transactions that modify 4KB of data (using the convention 1000KB = 1MB)

a) How many transactions can you process per second if you don't use a transaction log? (assume transactions modify data in random locations on disk)

b) How many transactions can you process per second if you set your database to only write to the sequential log without ever updating the tables disk?

c) A typical memory access takes approximately 100ns, with read/write speeds of 25GB/s. How many transactions can you process per second operating only in memory? (assume each transaction requires 1 memory access + the time to write 4KB to memory)

d) Imagine you can perfectly scale your database workload across servers (so 100 servers will process transactions 100x faster -- in reality they would be somewhat slower). If you had a cluster of servers durably processing transactions as in part b), how many would it take to match the performance of a single server processing transactions in non-durable memory as in part c)?

Solution: a) 250 transactions per sec. b) 42.5 thousand transactions per sec. c) ~3.84 million transactions per sec. d) about 90 servers

---

2. The problem with crashes/aborts

Examine the following transaction:

1. Read AccountA
2. Read AccountB
3. AccountB = AccountB + AccountA
4. AccountA = 0

For every step, explain how the output of the program would change if the computer crashes right before that step is executed.

Solution: Before steps 1, 2, and 3, nothing happens since we just performed reads. Before step 4, AccountB will be updated but AccountA will not be set to 0. In the case of a financial transaction, that would clearly be undesirable (for the bank, at least!). This is why we will care about Atomicity.

---

3. Atomicity

What is NOT a possible result from an atomic transaction?

a. All changes are made
b. Some changes are made

c. No changes are made

Solution: b) Atomicity guarantees that transactions are “all or nothing,” meaning that either all changes occur or no changes occur. This guarantees that it’s not possible for just “some” changes to be made.

---

4. Consistency

Which of the following example is related to consistency in transactions?

A. A teller looking up a balance at one time should not be allowed to see a concurrent transaction involving a deposit or withdraw from the same account. Only when the withdraw transaction commits successfully and the teller looks at the balance again will the new balance be reported.
B. When processing medical tests, there’s a whole set of rules that medical professionals have to follow. Doctors and their staff have to fill in a requisition properly. The specimen collection center has do verify that information, take samples and pass everything on to a lab. The lab that performs the tests must ensure that everything is valid before performing the test.
C. To transfer funds from one account to another involves making a withdraw operation from the first account and a deposit operation on the second. If the deposit operation failed, you don’t want the withdraw operation to happen either.
D. A data warehouse can still keep the contents of its database even facing a system crash or other failure.

Solution:
B is the correct answer.

A: This is an example of isolation
B: This examples shows that the lab has to follow some specified rules and it has to make sure the test goes from one valid state to another.
C: This is an example of atomicity
D: This is an example of durability

---

5. Isolation

Which of the following statements about isolation is correct?

A. To maintain the isolation of transaction, the transactions must be executed serially.
B. Isolation guarantees that once a transaction has committed, its effects remain in the database.

C. Isolation can be achieved by wisely scheduling the transactions as if they are executed serially.
D. Isolation of the transactions can be achieved by Write-Ahead Logging

Solution: C
A: the transactions can be executed concurrently, as described in C.
B: This is the description of durability.
D: Isolation of the transactions is ensured by concurrency control. WAL provides atomicity and durability.

---

6. Durability

Given the following transaction T:
A = 1 and B = 2 at the start of the execution

T	R(A)	R(B)	A := A+B	B := A+B	COMMIT

We run this transaction on a database that does not guarantee **Durability**. If our computer crashes after committing T, what are the possible values of A and B after it restarts?

Solution:
If both writes persist, A = 3, B = 5
If W(A) does not persist, A = 1, B = 5
If W(B) does not persist, A = 3, B = 2
If neither of them persist, A = 1, B = 2

---

7. Write-Ahead Logging (WAL)

Transaction T performs the following operations:

1. R(A)
2. R(B)
3. B := B+1, e.g. W(B)
4. A := A+1, e.g. W(A)

Questions:

A) Initially, A = 0, B = 1, and each write operation increases the record's value by 1. Write out the step-by-step changes to both the data and the log on main memory and disk, before T, during each step of T, and for each of the 2 steps in which WAL writes the log and data to disk (7 steps total).

B) Imagine we alter the WAL protocol so that we first commit T, then write the data and log records to disk. Describe which ACID property we lose and give an example why.

C) Now, say we alter our WAL protocol so instead it writes the data to disk before the corresponding log record. Describe what ACID property we lose, and give an example why.

Solution:

(A)

step	MM data	MM log	Disk data	Disk log
0 (before T)				
1	A=0			
2	A=0, B=1			
3	A=0, B=2	B: 1->2		
4	A=1, B=2	B: 1->2, A: 0->1		
5 (Write log)	A=1, B=2	B: 1->2, A: 0->1		B: 1->2, A: 0->1
6 (Write data)	A=1, B=2	B: 1->2, A: 0->1	A=1, B=2	B: 1->2, A: 0->1

(B) Durability. If we crash after writing “T committed”, then when we restart, we see that T completed, but the values of A and B are still 0 and 1.

(C) Atomicity. Say we write the data to disk then crash and restart before the log is saved to disk. We don’t know that T completed, so according to Atomicity (all or nothing) we should undo its actions and try again. However, without the log, we don’t know which actions to undo.

---

8. Scheduling

How can transaction scheduling affect the ACID properties of a DBMS? Give an example why.

Solution: Scheduling affects **isolation** and **consistency**.

Initially, A=1, B=1 (integrity constraint that A,B > 0)

T1: A*=2, B*=2

T2: B-=1, A-=1

If we schedule T2 before T1, then we break consistency (A=B=0). If we interleave the steps of T1 and T2 like so:
A*=2, B-=1, A-=1, B*=2
Then T1 is not isolated from T2 (We expect A=B, but in that case A=1 and B=0 at the end)

---

9. Serializability

Consider the following two schedules for transactions T_1 and T_2:

<u>Schedule 1</u>

T_1:		R(B)	W(C)			R(C)
T_2:	W(B)			R(C)	W(A)	
timestamp:	0	1	2	3	4	5

<u>Schedule 2</u>

T_1:	R(B)	W(C)				R(C)
T_2:			W(B)	R(C)	W(A)	
timestamp:	0	1	2	3	4	5

For each schedule, specify whether or not the schedule is conflict serializable and why. If the schedule is conflict serializable, give the equivalent serial schedule.

Solution: Schedule 1 is not conflict serializable since the conflict graph creates a cycle due to the WR conflicts at t = 0,1 and t = 2, 3. Schedule 2 is conflict serializable since the conflict graph forms a DAG. The equivalent serial schedule would be T_1 before T_2.

---

10. Conflict Types

Let's say you are buying a $2.00 cup of coffee at Coupa Cafe. But *just* after you've paid and your debit card transaction is being processed, Stanford decides to levy a draconian tax of 2% of both your account and Coupa Cafe's account. Consider the following simplified interleaving of transactions:

T1:			[Your acct -= $2.00]		[Coupa's acct += $2.00]			
T2:	[Coupa's acct*= .98]						[Your acct*= .98]	
timestamp:	0	1	2	3	4	5	6	7

You can assume that each transaction above consists of a read and a write, in that order, with each read and each write taking up a unit of time.

What are all the pairs of actions that conflict and what type of conflict do they form? Identify them by their timestamp. (Hint: the read in “Coupa’s acct*= .98” occurs at timestamp 0, and the write occurs at timestamp 1).

Solution: 0-5 (RW), 1-4 (WR), 1-5 (WW), 2-7 (RW), 3-6 (WR), 3-7 (WW)

---

11. Conflict Serializability

Are the following schedules conflict serializable? If not, explain why. If yes, provide the serial schedule.

Schedule 1:

T1	**R(A)**		**R(B)**			**W(B)**		
T2		**W(A)**		**R(C)**	**W(C)**		**R(B)**	**W(B)**

Solution: Yes. We see that T1 needs to do R(A) before T2 does W(A), T1 needs to R(B) before T2 W(B), and T1 needs to W(B) before T2 either does R(B) or W(B). We can have all of T1 go before all of T2. The serial schedule is T1 first, then T2. Another approach to solve this problem is to draw the conflict graph and notice that it’s a DAG.

Schedule 2:

T1	**R(A)**		**R(B)**			**W(A)**		
T2		**R(A)**		**R(C)**	**W(C)**		**R(B)**	**W(B)**

Solution: Schedule 2 is not conflict serializable. T2 has to do R(A) before T1 does W(A), which implies T2 must go before T1 in a serial schedule. However, T1 has to do R(B) before T2 does W(B), which implies T1 must go before T2 in a serial schedule. These cannot both be true, so there is no equivalent serial schedule.

---

12. Two-Phase Locking (Strict 2PL)

Which of the following schedules is conflict serializable, but would not be possible under strict 2PL? Remember that the 2PL protocol grabs an X (exclusive) lock for writing and an S (shared) lock for reading.

Schedule 1

T_1:	R(A)	W(A)		R(B)	
T_2:			W(A)		W(B)
timestamp:	0	1	2	3	4

Schedule 2

T_1:	R(A)		R(B)	W(B)	
T_2:		R(A)			R(A)
timestamp:	0	1	2	3	4

Schedule 3

T_1:	R(A)		R(A)		
T_2:		W(A)		W(B)	R(B)
timestamp:	0	1	2	3	4

Solution:
Answer: Schedule 1
Schedule 1 is conflict serializable but not possible under strict 2PL.

- Serial schedule: T_1 before T_2.
- Not possible under strict 2PL: T_1 obtains an S and then X lock on A at time 0/1, but has not yet committed (still needs to R(B) at time 3), so T_2 cannot obtain an X lock to W(A) at time 2.

Schedule 2 is conflict serializable and possible under 2PL.

- Serial schedule: T_1 before T_2.
- Possible under strict 2PL: T_1 obtains an S lock on A, but T_2 can still also obtain an S lock to read A after. T_1 then obtains an S and X lock on B, but T_2 doesn't need either.

Schedule 3 is not conflict serializable (and hence also not possible under strict 2PL). Note that T_1 performs a read before T_2 performs a write, but T_2 performs the write before T_1 performs another read.

---

13. Deadlocks

In lecture, we saw that 2PL is susceptible to deadlocks. Provide a series of lock requests made by transactions T1, T2, and T3 over shared resources A, B and C such that a deadlock occurs. Draw the corresponding waits-for graph for your solution.

Solution: Possible answer below

T_1:	S(A)		S(B) ….			
T_2:		X(B)			X(C) ….	
T_3:				S(C)		X(A) ….

Waits-for graph has 3 nodes: T1, T2, T3, and 3 edges: T1->T2, T2->T3, T3->T1 (a cycle)