CPS 196.3 Fall 2002
Optional Problem Set #2
Assigned: Friday, December 6
Due: Wednesday, December 11

Problem 1.

Consider the join $R \bowtie_{R.A = S.B} S$, given the following information about the tables to be joined. The cost metric is the number of disk I/O's and the cost of writing out the result should be uniformly ignored.

- R contains 10,000 rows and has 10 rows per page.
- S contains 2,000 rows and also has 10 rows per page.
- $S.B$ is a key of S.
- Both tables are stored compactly on disk in no particular order.
- No indexes are available.
- 52 memory blocks are available for query processing.

(a) What is the expected cost of joining R and S using a block-based nested-loop join, with R as the outer table?
(b) What is the expected cost of joining R and S using a block-based nested-loop join, with S as the outer table?
(c) What is the expected cost of joining R and S using a sort-merge join? What is the minimum number of memory blocks required for this cost to remain unchanged?
(d) What is the expected cost of joining R and S using a hash join? What is the minimum number of memory blocks required for this cost to remain unchanged?

Problem 2.

Consider the following schedule.

T_1	T_2	T_3	T_4
		w(E)	
r(D)			
	w(C)		
		w(A)	
r(E)			
w(B)			
r(B)			
	w(E)		
			r(A)
			w(C)

(a) Draw the precedence graph.
(b) Is this schedule conflict-serializable? If not, explain why not. Otherwise, give the equivalent serial schedule.

Problem 3.

For each schedule below, tell whether it is conflict-serializable. If yes, also tell:

- Whether it is recoverable;
- Whether it avoids cascading rollbacks;
- Whether it is possible under *strict* 2PL.

(a) T_1.write(B), T_2.read(A), T_2.write(A), T_1.read(A), T_1.write(A), T_1.commit, T_2.commit
(b) T_1.write(B), T_2.read(A), T_2.write(A), T_1.read(A), T_1.write(A), T_2.commit, T_1.commit
(c) T_1.write(B), T_2.read(A), T_2.write(A), T_2.commit, T_1.read(A), T_1.write(A), T_1.commit
(d) T_1.write(B), T_2.read(A), T_1.read(A), T_2.write(A), T_1.write(A), T_2.commit, T_1.commit
(e) T_2.write(B), T_2.read(A), T_2.write(A), T_1.write(B), T_2.commit, T_1.read(A), T_1.commit

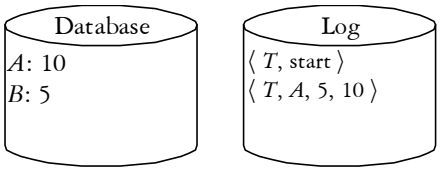
Problem 4.

Suppose we have a transaction T that performs the following two actions:

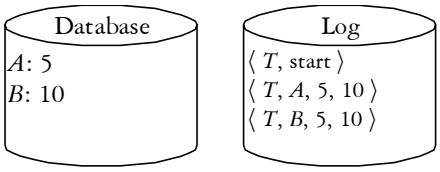
$A := A + 5; B := B + 5.$

Say that undo/redo logging is in use, and that initially, A = 5 and B = 5. For each hypothetical disk state shown below, state whether it is a legal (possible) state for undo/redo logging. If it is not a legal state, explain why not. Recall that the log entries are in the format ⟨ *transaction_id*, *variable_id*, *old_value*, *new_value* ⟩.

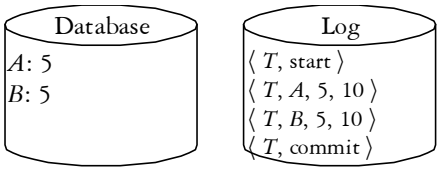
(a)



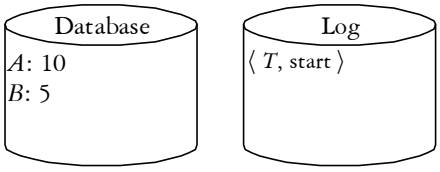
(b)



(c)



(d)



Problem 5.

Consider the following transaction log from the start of the run of a database system that uses undo/redo logging with fuzzy checkpointing:

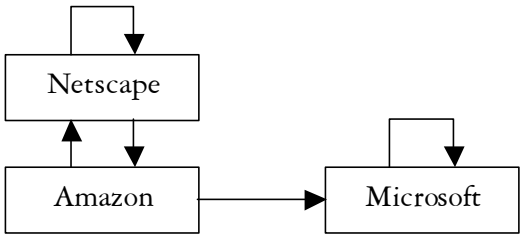
1. ⟨ T1, start ⟩
2. ⟨ T1, *A*, 45, 10 ⟩
3. ⟨ T2, start ⟩
4. ⟨ T2, *B*, 5, 15 ⟩
5. ⟨ T2, *C*, 35, 10 ⟩
6. ⟨ T1, *D*, 15, 5 ⟩
7. ⟨ T1, commit ⟩
8. ⟨ T3, start ⟩
9. ⟨ T3, *A*, 10, 15 ⟩
10. ⟨ begin-checkpoint {T2, T3} ⟩
11. ⟨ T2, *D*, 5, 20 ⟩
12. ⟨ T2, commit ⟩
13. ⟨ end-checkpoint ⟩
14. ⟨ T4, start ⟩
15. ⟨ T4, *D*, 20, 30 ⟩
16. ⟨ T3, *C*, 10, 15 ⟩
17. ⟨ T3, commit ⟩
18. ⟨ T4, commit ⟩

What is the value of the data items *A, B, C,* and *D* on disk after recovery:

(a) if the system crashes just before line 6 is written to disk?
(b) if the system crashes just before line 10 is written to disk?
(c) if the system crashes just before line 12 is written to disk?
(d) if the system crashes just before line 13 is written to disk?
(e) if the system crashes just before line 16 is written to disk?
(f) if the system crashes just before line 18 is written to disk?

Problem 6.

Consider the following mini-Web consisting of three pages.



(a) Compute the naïve PageRank values for these three pages using the fixed-point iteration. Start with all PageRank values equal to 1 and carry out at least 10 iterations. Show your intermediate results.
(b) Compute the practical PageRank values for these three pages (using a delay factor of 0.8). You may solve for the PageRank values directly instead of using the fixed-point iteration.