[A Mix of Yes/No Questions]
Indicate whether each of the following is true or false. Explain why (unless it is already explicitly explained in the notes.)

1. Consider the sort merge join of R and S. The minimum memory requirement for achieving two-pass processing of the sort merge join is that the memory should be able to contain at least the square root of max(|R|,|S|) blocks where |R| and |S| stand for the sizes of the relations in blocks.

2. Consider the hash join of R and S. The minimum memory requirement for achieving two-pass processing is the square root of max(|R|,|S|) blocks.

3. Checkpointing can significantly reduce recovery time when using UNDO logging.

4. If a schedule is not conflict serializable then it will definitely violate the consistency constraints of the database.

5. Two schedules with identical precedence graphs must be conflict equivalent.

6. In query optimization it is always better to perform projections as early as possible.

7. Sequential IOs on hard disk are more expensive than random IOs.

8. When the sizes of join relations are big (i.e., no relation fits in main memory) and we have indexes on the join attributes, the join index technique will outperform every other join technique we have seen.

9. A second level index for an indexed relation is usually dense.

10. Natural join is associative and commutative.

11. For queries of the form "SELECT$_{A>a}$R" extensible hashing is better than B+ tree.

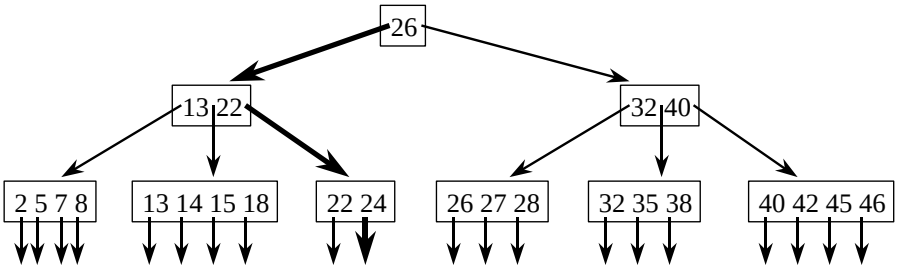
[Data organization]
Consider a table *R* that is stored in consecutive pages of the disk. We know that reading pages of the disk in the sequential order in which they are stored is at least ten times faster than reading pages in some random order. Give two non-trivially different queries where we may still prefer to read in random order instead of reading sequentially.

[Indexing 1]
Consider B+ trees where the maximum number of values in a node is 2 (and the minimum is 1). Give an example of 3-level B+ tree that can be reorganized into a 2-level tree with exactly the same values. Display both the 3-level and the 2-level tree. Use letters or numbers for the values.

[Indexing 2]
Consider the table R(A,B,…). Suggest ways in which a B+ Tree can be used for multi-key indexing of the pair of attributes (A,B).

[Indexing 3]
Draw an example of a valid 3-level B+ Tree of order n=2 where inserting an entry with key 20 causes a split to a non-leaf node but does not increase the height of the tree.

[Indexing 4]
Consider the following B+ Tree of order n=4.



Show "snapshots" of the B+ Tree as it evolves as a result of the following actions:

1. Insert an entry with key 29.
2. Insert an entry with key 30.
3. Insert 44
4. Insert 43
5. Insert 41

[Indexing 5]

The C+ trees are a variation of B+ trees (btw, I just invented them ☺). They have the following properties:

- Each page/node of the C+ tree has *m* to *3m/2* elements (*m* is even). The root may have any number of elements.
- *m > 2*
- has more than one levels
- All other properties of the C+ tree are identical with the B+ tree.

Write an *O(*log*n)* procedure for insertion of a new element *e* into a C+ tree.

[Indexing 6]

1. Consider an extensible hash table where 200 buckets are actually allocated at this point. What is the size of the smallest possible directory at this point?
2. Consider an extensible hash table whose directory has 1024 entries. What is the largest number of entries that could point to the same bucket?
3. Consider a linear hash table with max used bucket 110 (in binary). Which bucket should a key with hash value 111 (binary) be sent to?

[Indexing 7]

Consider an extensible hash structure where buckets can hold up to three entries. Initially the structure is empty. The keys and their hash values are the following (don't ask what function produces such values☺)

A → 00001
B → 00011
C → 00110
D → 01110
E → 01111
F → 10001
G → 10101
H → 10111
I → 11000
K → 11001
L → 11101
M → 11111

1. Suppose we insert the values in the order given above. Show the structure after all the values have been inserted.
2. Now assume we insert in the order F,E,D,C,B,A,M,L,K,I,H,G. Show the structure after all values have been inserted.

[Indexing 8]

Consider a linear hash structure where buckets can hold up to two records. Initially the structure is empty. Assume that the utilization threshold value is 100%. The keys and their hash values are given above. Show the structure after all values have been inserted.

No More Indexing Please…

Provide reasons why we should **not** keep an index on an attribute A of some relation R.

Query Processing 1
Can it be worthwhile for a query optimizer/query processor to temporarily create an index (B-tree index, hash table index, ...) to speed up the cost of executing a query ? That is, does it make sense during the query execution to create an index that is discarded at the end of the query ? If your answer is "no" discuss why other alternatives (nested loops, merge join, or hash join) will *always* be better. If your answer is "yes", give an example, including statistics and rough cost estimates, that shows that the index join (with index creation) is cheaper than nested loops or hash join. You may assume that merge join is too expensive if the relations are not sorted.

Query Processing 2
Consider the following query:

```
SELECT *
FROM R, S, T
WHERE R.A = S.A and R.B = T.B AND T.C = S.C
      AND R.D = 16 AND S.F = 17
```

1. Give six algebraic expressions that (I) contain no cartesian product, (II) have pushed selections down, (III) each join is associated with an atomic condition (not a conjunctive condition). They should be non-trivially different. For example, expressions that are equivalent upon applying the commutativity of join do not count as different.
2. Find the optimum plan (in terms of #block accesses), i.e., an algebraic expression where selections and joins are annotated with execution primitives. The execution primitives are *SCAN* and *INDEX* for the selection operator and *ONE-PASS*, *RIGHT-INDEX*, and *SORT-MERGE* for join. Compute an estimation of the optimal plan's cost. There may be more than one optimum plans, i.e., different optimum plans that have the same cost.

Assume the following schemas and statistics for the relations.
- R has 10^9 tuples, an index on D, V(D,R)=10^3, V(A,R)=10^9, V(B,R)=10^9
- S has 10^9 tuples, an index on F, V(F,S)=10^3, V(A,S)=10^9, and V(C,S)=10^9
- T has 10^9 tuples, an index on B, with V(B,T)=10^9, V(C,T)=10^9.

Assume the block size is 4096 Bytes, the block header size is 96 bytes, each indexed attribute occupies 5 bytes and the tuple size of each relation is 80 bytes. Assume the available memory buffer is 10^9 bytes. Assume SSD (i.e., random and sequential accesses cost the same.)

[Query Processing 3]
Construct the hypergraph for the following expression.

$SELECT_{R.A=W.A}$ *((((((R(A,B) JOIN S(B,C,D)) JOIN T(B,E,F)) JOIN U(F,G,H)) JOIN V(G,I)) JOIN W(I,J,A))*

Provide 4 left-deep algebraic expressions that do not involve cartesian products and compute the above.

[Query Processing 4]
Suppose we have relations R(A,B), with 1000000 tuples, and S(B,C), with 100000 tuples. Also assume that 20 of R's tuples fit in one block and 100 records of S fit in one block. Assume that there are 500 possible values for B. Each of the 500 values appears with equal probability in R and with equal probability in S. Assume the main memory is 1500 blocks. Compute the following, first assuming pipelining and then without assuming pipelining.

(a) $SELECT_{A=a}(R \times S)$. Do not optimize the expression (say be pushing the selection down. Just estimate the cost of computing it.)
(b) Sort-merge join for $SELECT_{A=a}(R$ JOIN $S)$ assuming that the relations are not sorted.

[Query Processing 5]
Is the following equation correct in the general case?
Is it correct if we know that $PROJECT_{attr}R$ has no duplicates and $PROJECT_{attr}S$ also has no duplicates.
Prove why or why not for each case.
DELTA $PROJECT_{attr}(R-S)$ = DELTA($PROJECT_{attr}R$) - DELTA($PROJECT_{attr}S$)

[Recovery 1]
Consider a transaction T that performs the following two actions:

A:= A + 5

B:= B + 5

.UNDO Logging

Assume that UNDO logging is in use and initially A=5 and B=5. For each hypothetical disk state below, state whether it is a possible (legal) state for UNDO logging. If it is not, explain why not.

1. **DB(A:5, B:5), LOG(<T,start>)** , where the notation means that the persistent part of the DB has the value 5 in A and 5 in B.
2. **DB(A:10, B:5), LOG(<T,start>)**
3. **DB(A:10, B:5), LOG(<T,start>, <T,A,5>, <T,B,5>, <T,commit>)**
4. **DB(A:5, B:10), LOG(<T,start>, <T,A,5>, <T,B,5>)**

.REDO Logging

Assume that REDO logging is in use and initially A=5 and B=5. For each hypothetical disk state below, state whether it is a possible (legal) state for UNDO logging. If it is not, explain why not.

1. **DB(A:10, B:5), LOG(<T,start>, <T, A, 10>)**
2. **DB(A:10, B:5), LOG(<T,start>, <T, A, 10>, <T, B, 10>)**
3. **DB(A:10, B:5), LOG(<T,start>, <T, A, 10>, <T, B, 10>, <T, commit>)**

.UNDO/REDO Logging

Assume that UNDO/REDO logging is in use and initially A=5 and B=5. For each hypothetical disk state below, state whether it is a possible (legal) state for UNDO/REDO logging. If it is not, explain why not.

1. **DB(A:10, B:5), LOG(<T,start>, <T, A, 5, 10>)**
2. **DB(A:10, B:5), LOG(<T,start>)**
3. **DB(A:5, B:10), LOG(<T,start>, <T, A, 5, 10>, <T, B, 5, 10>)**
4. **DB(A:5, B:10), LOG(<T,start>, <T, A, 5, 10>, <T, B, 5, 10>, <T, commit>)**

[Recovery 2]
A system reboots after a crash and finds the following database and UNDO log state:
DB(A:10, B:10)
LOG(<T1,start>, <T2,start>, <T1, A, 5>, <T1, commit>, <T2, B, 5>, <T2, A, 15> *CRASH)*

1. What was the initial state of the system before T1 and T2 began executing?
2. What will be the state after the recovery?
3. Repeat questions (1) and (2) assuming it is a REDO log.

[Recovery 3]

1. **Undo logging** requires that before an item *X* is modified on disk the log records pertaining to *X* are in the disk. (This is the WAL rule.) Show using an example that an inconsistent database may result if log records for *X* are not output to the disk before *X*.
2. **Redo logging**: Show using an example that an inconsistent database may result if some items are written on the disk before the commit is written on the log, even if WAL holds.
3. **Undo logging**: Show using an example that an inconsistent database may result if some items are not written in the disk by the time the commit is written. Assume that WAL holds.

Your examples should involve a crash and should clearly show (I) the write actions of the transaction, (II) the state of the log and the database at the time of the crash, and (III) why successful recovery cannot be accomplished.

[Recovery 4]

Consider a transaction that performs the following actions in the given order

1. Write object A
2. Write object B
3. Commit

Decide whether each one of the following snapshots of the database and the log are **possible** or **impossible** at any point during or after the end of the transaction. The snapshots refer to data that are actually in the disk. Assume that if we do not mention explicitly that something is stored on the disk then it is not stored on the disk. For each item give an answer for both Undo and Redo logging.

1. Log has entry for write(A) and the database has the new value of A
2. The database has the new value for A
3. The log has entries for write(A) and write(B) and also has the commit entry
4. The log has entries for write(A) and write(B). The database has the new values for A and B
5. The log has entries for write(A), write(B), and the commit entry. The database has the new value for A
6. Log has entry for write(A) and commit and the database has the new value of A

.Complete Solution:

	UNDO	**LOGGING**
	possible	**impossible**
1		
2		
3		
4		
5		
6		

	REDO	**LOGGING**
	possible	**impossible**
1		
2		
3		
4		
5		
6		

[Concurrency Control 1]

Consider the following schedule:

S=$w_3(E)$ $r_1(D)$ $w_2(C)$ $w_3(A)$ $r_1(E)$ $w_1(B)$ $r_1(B)$ $w_2(E)$ $r_4(A)$ $w_4(C)$

1. Draw the precedence graph.
2. Is this schedule conflict serializable? Why? Why not? If yes, provide the equivalent serial schedule.

[Concurrency Control 2]

Consider the following two transactions:

T1= $w_1(C)$ $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$
T2= $r_2(B)$ $w_2(B)$ $r_2(A)$ $w_2(A)$

Assume that the scheduler uses exclusive locks only. For each of the following instances involving T1 and T2, annotated with lock and unlock actions, complete the following table

.Instance A

T1= **$L_1(C)$** w1(C) **$L_1(A)$** $r_1(A)$ $w_1(A)$ **$U_1(A)$ $L_1(B)$** $r_1(B)$ $w_1(B)$ **$U_1(C)$ $U_1(B)$**
T2= **$L_2(B)$** $r_2(B)$ $w_2(B)$ **$U_2(B)$ $L_2(A)$** $r_2(A)$ $w_2(A)$ **$U_2(A)$**

.Instance B

T1= **$L_1(C)$** w1(C) **$L_1(A)$** $r_1(A)$ $w_1(A)$ **$L_1(B)$** $r_1(B)$ $w_1(B)$ **COMMIT $U_1(A)$ $U_1(C)$ $U_1(B)$**
T2= **$L_2(B)$** $r_2(B)$ $w_2(B)$ **$L_2(A)$** $r_2(A)$ $w_2(A)$ **COMMIT $U_2(A)$ $U_2(B)$**

.Instance C

T1= **$L_1(C)$ $L_1(A)$** w1(C) $r_1(A)$ $w_1(A)$ **$L_1(B)$** $r_1(B)$ $w_1(B)$ **COMMIT $U_1(A)$ $U_1(C)$ $U_1(B)$**
T2= **$L_2(B)$** $r_2(B)$ $w_2(B)$ **$L_2(A)$** $r_2(A)$ $w_2(A)$ **COMMIT $U_2(A)$ $U_2(B)$**

.Instance D

T1= **$L_1(B)$ $L_1(C)$** w1(C) **$L_1(A)$** $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$ **COMMIT $U_1(A)$ $U_1(C)$ $U_1(B)$**
T2= **$L_2(B)$** $r_2(B)$ $w_2(B)$ **$L_2(A)$** $r_2(A)$ $w_2(A)$ **COMMIT $U_2(A)$ $U_2(B)$**

a) Question	(1) A	b) B	C D
Is 2PL			
Is strict 2PL			
'ill always result in conflict serializable schedul			
ill always result in schedule that avoids cascadi rollback			
Will result in a strict schedule			
Will result in a serial schedule			
May result in a deadlock			

[1] If no deadlock happens

[Concurrency Control 3]
Consider the following two transactions:
T0:
read(A);
read(B);
if (A = 0) then B = B + 1;
write(B);

T1:
read(B);
read(A);
if (B = 0) then A = A + 1;
write(A);

Let the consistency requirement be (A = 0) OR (B = 0),with A = B = 0 the initial values
(a) Show that every serial execution involving these two transactions preserves the consistency of the database.
(b) Show a concurrent execution of T0 and T1 which produces a nonserializable schedule
(c) Is there a concurrent execution of T0 and T1 which produces a serializable schedule? Prove why or why not.

[Concurrency Control 4]
For each schedule below indicate whether the schedule is conflict serializable or not. If yes, provide the equivalent serial schedule.
1. *w1(A),w2(B),w3(C),,r3(A),,r3(B)*
2. *w1(A),w2(B),w3(C),r3(A)r3(B)*
3. *r1(A),w2(A),w2(B),r1(B)*
4. *w1(A),w1(B),r2(A),w2(B),w1(C),r3(B),w3(C)*

[Concurrency Control 5]
Consider the following schedule.
Is it conflict serializable?
Prove why or why not.

w3(B),w3(A),w1(B),r2(B),r2(A),w1(A)

[Concurrency Control 6]
Consider the following schedule *S*, consisting of transactions T_1, T_2 and T_3

T_1	T_2	T_3
w(A)		
	r(A)	
		w(B)
w(B)		
		w(B)
	w(A)	
		r(B)
	r(B)	

- Give the precedence graph for *S*
- Is *S* conflict serializable *?* Justify your answer.

[Concurrency Control 7]
Two transactions are *not interleaved* in a schedule *S* if every operation of one transaction precedes every operation of the other. (Note, the schedule may not be serial.) Give an example of a *serializable* schedule *S* that has all of the following properties:

- transactions T_1 and T_2 are not interleaved in S
- T_1 precedes T_2 in S
- in any serial schedule equivalent to S, T_2 precedes T_1

Hint: The schedule may include more than two transactions.

[Concurrency Control 8]

Consider two non-identical schedules S and S' consisting of transactions $T_1, \ldots, T_n$ where $n>1$. For each of the following set of conditions decide whether S and S'

- have to be **equivalent**,
- have to be **nonequivalent**,
- the described conditions **cannot hold**
- **none** of the above

The conditions are

1. S only reads, S and S' are serial and $n=2$
2. All transactions $T_1, \ldots, T_n$ write a specific item A exactly once and S and S' are serial.
3. The precedence graphs of S and S' are identical and acyclic.
4. S is serializable but S' is not serializable.
5. No two serial schedules (of $T_1, \ldots, T_n$) are equivalent and S and S' are serializable.

	equivalent	**nonequivalent**	**none**	**cannot hold**
1				
2				
3				
4				
5				

[Concurrency Control 9]

For each one of the following schedules decide whether

- they are serializable
- they can be produced by a Two Phase Lock (2PL) scheduler
- they can be produced by a strict 2PL scheduler

and check in the table below all entries that apply. Justify your answers in a concrete way. For example, if a schedule is 2PL show a series of lock/unlock operations that are compatible with the 2PL rules (you can use the empty lines of the schedules to place lock/unlock operations.) Answers of the form "it is serializable because I can see that I can swap the operations" would not deserve full credit. Assume that shared locks can be used if in this way you can make a schedule 2PL or strict 2PL.

T_1	T_2	T_3
		r(D)
w(A)		
	r(B)	
		w(D)
w(B)		
	r(D)	

T_1	T_2	T_3
w(A)		
	r(B)	
		w(D)
w(B)		
	r(D)	
		w(D)

T_1	T_2	T_3
		r(D)
w(A)		
	r(B)	
w(B)		
	r(D)	
		w(D)

	serializable	**2PL**	**Strict 2PL**
1			
2			
3			