CS411 Database Systems

Fall 2007

Department of Computer Science
University of Illinois at Urbana-Champaign

Final Examination

December 12, 2007
Time Limit: 180 minutes

- Print your name and NetID below. In addition, print your NetID in the upper right corner of every page.

  Name: ______________________ **NetID:** ______________________

- Including this cover page, this exam booklet contains **17** pages. Check if you have missing pages.
- The exam is closed book and closed notes. You are allowed to use scratch papers. No calculators or other electronic devices are permitted. Any form of cheating on the examination will result in a zero grade.
- Please write your solutions in the spaces provided on the exam. You may use the blank areas and backs of the exam pages for scratch work.
- Please make your answers clear and succinct; you will lose credit for verbose, convoluted, or confusing answers. *Simplicity does count!*
- Each problem has different weight, as listed below– So, plan your time accordingly. *You should look through the entire exam before getting started, to plan your strategy.*

Problem	1	2	3	4	5	6	7	8			Total
Points	15	10	10	14	17	12	12	10			100
Score											
Grader											

Problem 1 (*15 points*) Basics

For each of the following statements, indicate whether it is TRUE or FALSE by circling your choice. If you change your mind, cross out both responses and write "True" or "False." You will get 1 point for each correct answer, 0 point for each incorrect answer.

(1) *True False*
Logical and physical addresses are both representations for the database address.

(2) *True False*
When a client requests a record that contains a BLOB (binary, large object), the database server that receives the request should return the entire record at a time.

(3) *True False*
When an index covers many blocks, we may want to put a second-level index on the first-level index that contains pointers to records. When we use multiple levels of indexes, the first-level index must be sparse.

(4) *True False*
Secondary indexes are always dense.

(5) *True False*
In B-trees, we sometimes need to have overflow blocks.

(6) *True False*
Dynamic hash tables support range queries.

(7) *True False*
In linear hash tables, we sometimes have overflow blocks.

(8) *True False*
One-pass algorithms for set union of two relations R and S requires M main memory buffers such that $B(R) + B(S) \leq M$.

(9) *True False*
In algebraic laws, $\sigma_C(R - S) = R - \sigma_C(S)$ holds.

(10) *True False*
Dynamic programming is a bottom-up method, where we consider only the best plan for each subexpression of the logical-query plan.

(11) *True False*
In undo logging, it is not always possible to recover some consistent state of a database system if the system crashes during recovery.

(12) *True False*
A scheduler based on the two-phase locking scheme always executes multiple transactions with some serial schedule.

(13) *True False*
Two-phase locking prevents deadlocks.

(14) *True False*
Concurrency control by timestamps is superior to that by locks if most transactions are read-only.

(15) *True False*
ACR (avoids cascading rollback) schedules are always serializable.

Problem 2 (*10 points*) Pointer Swizzling

Suppose that the important actions related to data storage take the following times, in some arbitrary time units:

- On-demand swizzling of a pointer: 30;
- Automatic swizzling of pointers: 20 per pointer;
- Following a swizzled pointer: 1;
- Following an unswizzled pointer: 10.

(i) Suppose we design a pointer-swizzling control scheme like the following. At the beginning, we automatically swizzle 30% of the pointers and leave the rest unswizzled. Once a pointer is followed, we swizzle it by probability 0.5. If an unswizzled pointer has been followed twice, we swizzle it. Suppose there are 200 pointers in our data. The number of times that they are followed by a program is distributed according to the following histogram.

times of being followed	0	1	2	3
number of pointers	20	80	60	40

What's the **expected cost** of this program in terms of pointer following? (6 points)

(ii) A block in memory is said to be *pinned* if it cannot at the moment be written back to disk safely. Explain in what situation a block could be pinned because of swizzled pointers and describe how we should extend a translation table that maps database addresses to memory addresses in order to unpin such blocks. (4 points)

Problem 3 (*10 points*) Indexing

Consider the B-tree of $d = 2$ (*i.e.*, each index node can hold at least $d = 2$ keys and at most $2d = 4$ keys) shown in Figure 1.

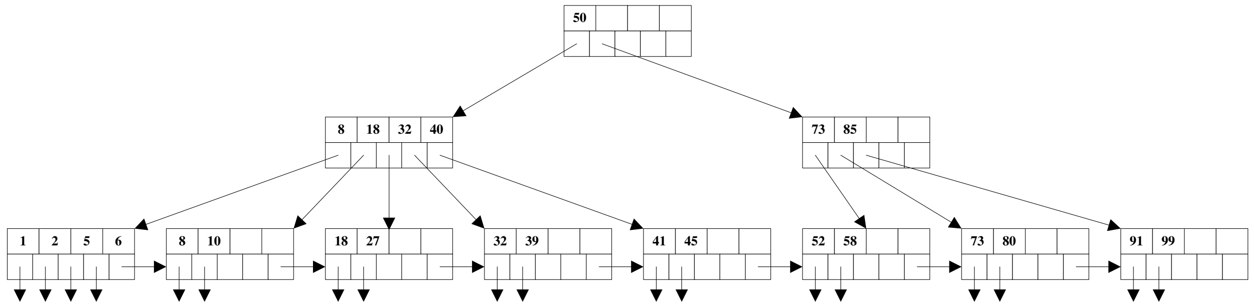


Figure 1: B-tree of Problem 3

(a) What is the largest number of records that can be inserted into this tree (more nodes may be created), while the height of the tree does not change. (3 points)

(b) show the steps in executing the following operation: Lookup all records in the range from 10 to 58 (including 10 and 58). (3 points)

(c) Show the B+ tree that would result from deleting the data entry with key 91 from the original tree. (4 points)

Problem 4 (*14 points*) Query Execution

Consider two relations $R(a, b)$ and $S(b, c)$ with the following statistics:

$T(R) = 10,000$, $B(R) = 1,000$ (each block contains 10 tuples),
$V(R, b) = 200$ (number of distinct values of attribute b in R),

$T(S) = 6,000$, $B(S) = 1,500$ (each block contains 4 tuples),
$V(S, b) = 100$ (number of distinct values of attribute b in S),
$V(S, c) = 20$ (number of distinct values of attribute c in S) and $\underline{c > 100}$.

Also, we assume the number of available memory blocks is $M = 101$.

Please answer the following questions:

(i) Estimate the number of tuples in $\sigma_{c=150}(S)$ (2 points)

(ii) Estimate the number of tuples in $R \bowtie \sigma_{c>25}(S)$. (2 points)

(iii) Suppose we have a B-tree index available for attribute b of S. The tree has 3 levels, and each node contains 4 keys– that is, there would be totally 100 keys in the 5×5 leaf nodes, where each key would correspond to a distinct value of b in S. Assume each node of the index occupies one block.

For simplicity, we assume the tuples with the same b value are stored consecutively in the disk but may spread in different blocks. Please estimate the worst cost of executing the following query: (the cost is measured by disk I/Os for accessing the index and the tuples). (3 points)

SELECT *
FROM S
WHERE $b = 100$ OR $b = 1000$

(iv) Consider joining R and S using a block-based nested loop join (without using any index). Suppose R is used as the outer loop. Estimate the cost. (3 points)

(v) Now suppose we want to use sort-join. Assuming that the number of tuples with the same b value is not large, we aim to use the more efficient sort-based join approach:

1. Created sorted sublists of size M, using b as the sort key, for both R and S.
2. Bring the first block of each sublist into the memory buffer.
3. Repeatedly find the least b-value, and output the join of all tuples from R with all tuples from S that share this common b-value. If the buffer for one of the sublist is exhausted, then replenish it from disk.

Please estimate the cost (total disk I/Os) of applying this sort-join on R and S. Also, state the requirement on the number of memory blocks M considering the maximum number of blocks K for holding tuples of R and S with the same b value.

(4 points)

Problem 5 (*17 points*) Query Optimization

Suppose we want to compute the following:

$$\tau_b(R(a,b) \bowtie S(b,c) \bowtie T(c,d)), \quad \text{where } \tau_b \text{ specifies sorting on } b.$$

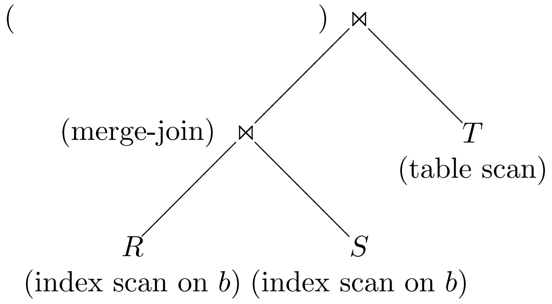
That is, we want to join three relations R, S, and T, and sort the results on attribute b. Let us make the following assumptions:

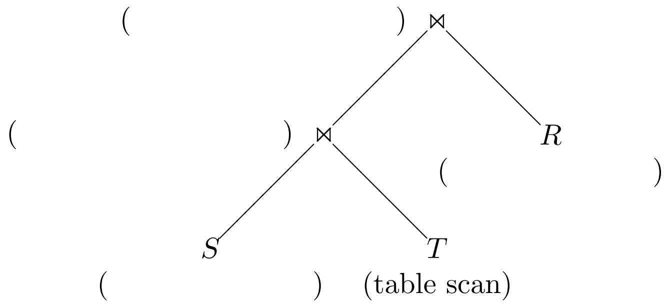
- First, for accessing each relation:
  - R can be *index scanned* on attribute a or b, or *table scanned.*
  - S can be *index scanned* only on b, or *table scanned.*
  - T can only be *table scanned.*

  We assume the index is based on B-tree. Accessing a relation using *index scan* on an attribute will produce records sorted by that attribute, while *table scan* does not give any guaranteed output order for the records.
- Second, for joining two relations, we have the following two choices:
  - *merge-join* can be used if the two relations are already sorted on the join attribute. It will simply join the two sorted relations using the merge part of the simple sort-based join algorithm, and the output would be sorted.
  - *nested-loop join* can be used in any case. For simplicity, we assume the block-based nested-loop join is used and the relation produced in the *left* sub-tree of a query plan will be used as the outer loop.

(i) Please draw *two* logical query plans (without concerning the physical access and join methods) that are *not* "left-deep." (3 points)

(ii) Now, consider the following two logical query plans (regardless of what your answer is for (i)), where we also fill in some physical access and join methods. Note we show only the join part (and not sorting) of the query.





Suppose we would like to avoid applying an additional sort on the final output relation, while still achieve the desired sorted result (i.e., τ_b). Please help fill appropriate access and join methods into the brackets in the above two query plans. (5 points)

(iii) An *interesting order* holds for a join result if it is sorted in an order that will be useful for the operations in the higher part of the expression tree– for example, sorted on the attribute(s) specified in a sort (τ) operator at the root, or the join attribute(s) of a later join.

For our problem, we want the final result sorted on attribute b. Consider the following three plans for $R \bowtie S$. Please circle "yes" if the plan produces an interesting order, and "no" if not. (3 points)

Plan	$R \bowtie S$	interesting order?
Plan A	(table-scan R) nested-loop-join (table-scan S)	yes / no
Plan B	(index-scan R on b) merge-join (index-scan S on b)	yes / no
Plan C	(index-scan R on b) nested-loop-join (table-scan S)	yes / no

Please briefly explain your choices. (3 points)

(iv) For query optimization, we use an enhanced method that improves upon the dynamic programming approach: It will keep for each subexpression the plan of lowest cost (as dynamic programming does). In addition, it wll also keep the one with lowest cost from those plans that produce an interesting order.

The table below lists the estimated costs for the three plans (see the table in (iii)). Which plans would be kept when considering the subexpression $R \bowtie S$ using this enhanced method? Please circle "yes" if the plan will be kept, and "no" if it will be pruned. (You may need to refer to the results, the interesting order of each plan, in (iii) to make the decisions)

Plan	estimated cost	keep?
Plan A	2000	yes / no
Plan B	3000	yes / no
Plan C	4000	yes / no

Please briefly explain your choices. (3 points)

Problem 6 (*12 points*) Recovery

Consider a database with data items {A, B, C, D}. The system uses an undo/redo scheme and has the following logs. Note that an entry <T, X, old, new> means transaction T changes the value of X from old to new. We consider recovery using this undo/redo log.

1. <T2 start>
2. <T2, B, 10, 11>
3. <T1 start>
4. <T2 commit>
5. <T1, A, 20, 21>
6. <Checkpoint start; Active= {T1}>
7. <T3 start>
8. <T3, C, 30, 31>
9. <T4 start>
10. <T4, B, 40, 41>
11. <T4 commit>
12. <Checkpoint end>
13. <T3, D, 50, 51>
14. <Checkpoint start; Active= {T1, T3}>
15. <T1, C, 31, 32>
16. <T5 start>
17. <T5, D, 51, 52>
18. <T3 commit>
19. <T6 start>
20. <T6, C, 32, 33>
21. <T5 commit>
22. System failed

(i) List all possible values of A, B, C and D. That is, what are the possible data values on the disk at the point of failure (after action 21)? (3 points)

(ii) During recovery, what are the transactions that need to be undone? (3 points)

(iii) During recovery, what are the transactions that need to be redone? (3 points)

(iv) What are the values of A, B, C, D after recovery? Explain why? (3 points)

Problem 7 (*12 points*) Concurrency control

Answer the following questions:

(i) Briefly explain one advantage of nonquiescent checkpointing compared to quiescent checkpointing. (3 points)

(ii) Consider the schedule S: w1(X); w3(X); w2(X); w4(X); r4(Y); w1(Y). Draw the precedence graph for the schedule S. Is it conflict-serializable? Explain why. (3 points)

(iii) Consider the schedule S: r1(X); r2(Y); w2(X); w1(Y). Does deadlock occur using the two-phase locking rule? Explain why. (3 points)

(iv) In the following sequences of events, $R_i(X)$ means that transaction T_i starts and its read set is the list of database elements X. Also, V_i means T_i attempts to validate and $W_i(X)$ means that T_i finishes, and its write set was X. Tell what happens when the following sequence of events is processed by a validation-based scheduler. (3 points)

$$R_1(A,B); R_2(B,C); V_1; R_3(C,D); V_3; W_1(A); V_2; W_2(A); W_3(D)$$

Problem 8 (*10 points*) Deadlocks

Answer the following questions:

(i) For the sequence of actions below, assume that locks are requested immediately before each read and write action. However, if a transaction is already holding a lock on a database element, it does not need to obtain that lock again to execute another action on that element. Also, unlocks occur immediately after the final action that a transaction executes. Tell which locking actions are denied, and whether a deadlock occurs, by drawing a wait-for graph. (5 points)

$$r_1(A); r_2(B); w_1(C); w_2(D); r_3(C); w_1(B); w_4(D); w_2(A);$$

(ii) We observed in our study of lock-based schedules that there are several reasons why transactions that obtain locks could deadlock. Can a timestamp-based scheduler using the commit bit C(X) have a deadlock? If you beleive that the scheduler does not have any deadlock, prove that. Otherwise, describe a situation where a deadlock occurs. (Hint: consider the case where two transactions T_1 and T_2 run while accessing two database elements A and B.) (5 points)