American University of Beirut
CMPS 377
Internal of Database Systems
Spring 2003

Final Exam

Date: June. 11th, 2003, 3:00 – 5:00pm.
Instructor: Jihad Boulos

Name: .....

ID #: .....

This is an open-book, closed-notes exam. Your exam should have 16 pages, and there are 7 questions totaling 130 points. You may use the main course textbook. You are **NOT** allowed to use any other notes. Your answers should be concise, and when possible should be a list of important points rather than prose. Solve as much problems as you can. I advise each one of you to pick the problems that he/she thinks are easiest for him/her and work on them. I also advise you to spend time on understanding the problem and budget your time for solving each problem, or else you will be wasting a lot of time on one problem and will run out of time for other problems.

Exercise 1 (8 points):

Imagine that you have a data warehouse with the following relations:

- Customers (Name, Age, Gender, CustID): 100,000 disk blocks.
- Purchases (CustID, Product, Date, Location, Amount): 2,000,000 disk block.
- SalesCalls(CustID, Salesperson, Date, Result): 300,000 disk blocks.

You have observed the following query mix over these relations:

- 10% queries selecting on Customers.CustID
- 30% queries selecting on Customers.Name.
- 35% queries selecting on Purchases.Product.
- 10% queries selecting on SalesCalls.Salesperson.
- 15% queries selecting on SalesCalls.Date.

You want to create indexes over these relations to speed up queries over these relations. You decide that you have enough resources (disk space, etc.) to build two indexes. You may assume that the index allows you to retrieve the answer to the query with significantly less cost than doing a table scan.

Which attributes should you build indexes over? Please explain briefly.

Exercise 2 (16 points):

Say you have a relation with 100,000 records. You want to hash the relation into a hash table with 1000 buckets. A disk block can store at most 100 records (along with an optional pointer to an overflow block). Assume that a disk block cannot store records from two different buckets.

a. What is the maximum number of disk blocks you would need for the relation?

b. What is the minimum number of disk blocks you would need?

Next, suppose that you want to store the relation with 100,000 records in a B+-tree. However, unlike a standard B+-tree, here the leaf nodes store the actual records, as opposed to just pointers to the records. Thus, a leaf node can store a maximum of 100 records. That is, for leafs the order is 100. For non-leaf, the order is 1000, i.e., a maximum of 1000 keys can be stored.

c. What is the maximum number of disk blocks you would need to store the relation in this B+-tree?

d. What is the minimum number of disk blocks you would need?

Exercise 3 (16 points):

Consider the following database stored on disk:

Element	Value
A	13
B	40
C	35
D	4
E	18

For each of the following logs state (i) whether that log could be an undo log for actions that resulted in the above database, and (ii) whether that log could be a redo log for actions that resulted in the above database. For each of (i) and (ii), if not, explain why not.

(a)
<START T1>
<T1, C 35>
<T1, D, 450>
<START T2>
<T2, C, 18>
<T2, B, 40>
<COMMIT T1>
<START CKPT (T2)>
<END CKPT>
<T2, D, 18>
<START T3>
<T3, C, 35>
<T3, E, 18>
<T2, A, 13>
<COMMIT T3>
<COMMIT T2>

Could be UNDO LOG?

Could be REDO LOG?

(b)
<START T1>
<T1, D, 4>
<START T2>
<T2, E, 6>
<T1, A, 5>
<START CKPT (T1,T2)>
<T1, E, 18>
<START T3>
<T3, C, 35>
<T3, A, 13>
<COMMIT T2>
<T3, B, 40>
<COMMIT T3>
<END CKPT>
<T1, A, 11>
<COMMIT T1>

Could be UNDO LOG?

Could be REDO LOG?

Exercise 4 (30 points):

Consider the following sequence of actions, listed in the order they are submitted to the DBMS:

Sequence S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction T*i* is *i*. For lock-based concurrency control mechanisms, add lock and unlock requests to the above sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all of its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. (Show the waits-for graph if a deadlock cycle develops.)
3. Conservative (and strict, i.e., with locks held until end-of-transaction) 2PL.
4. Optimistic concurrency control.
5. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.

Exercise 5 (16 points):

Consider a database that uses a validation mechanism for concurrency control. There are only 5 kinds of transactions in the system. The read, write actions of the 5 kinds of transactions are given as follows:

(1) r(A) r(B) w(A) w(B)
(2) r(B) w(D) w(E)
(3) r(B) r(D) w(F)
(4) r(B) r(E)
(5) r(B) r(C) w(F)

Note that (1)-(5) are transaction types and not transactions. In particular we can have two different transactions of the same type. Two transactions are said to execute concurrently if at some point in time, both the transactions have started, but neither has finished.

(a) Determine for each pair of transaction types (m, n) (including the case $m = n$), if the transaction of type n can by aborted due to a transaction of type m when executing concurrently (in the absence of any other transaction). Please indicate your answer in the table below by placing YES in column n and row m if a transaction of type n can be aborted due to a transaction of type m, and NO otherwise.

	Transaction of this type (n)				
Can be aborted by type below (m)	(1)	(2)	(3)	(4)	(5)
(1)					
(2)					
(3)					
(4)					
(5)					

(b) We now want to determine if two transactions of types n and m can by run without any concurrency control. The answer will by true if under every possible schedule, neither transaction can cause the other to abort (in the absence of any transactions of types different from n and m).
Write an expression $E(n, m)$ that tells us when transactions can be run without concurrency control. In particular, expression $E(n, m)$ should evaluate to true if transactions of type n and m can be run without concurrency control. Write your answer using the table of part (a), where TABLE(n, m) is true if entry (n, m) in

the table is YES (true). (For example, your answer can by an expression like "TABLE(n, m) AND NOT TABLE(n, m+1)," but this is of course ***not*** the right answer.)

Exercise 6 (24 points):

Consider a relational database with tuples $A, \ldots, L$, stored in disk blocks 1,...,4 as shown:

- Block 1: A, B, C
- Block 2: D, E, F
- Block 3: G, H, I
- Block 4: J, K, L

Assume we only have one kind of locks: exclusive locks. Define a "convoy" as a point in time in which one transaction T holds a lock on an object O, and at least two other transactions are waiting for the lock on object O. Define a "deadlock" as a point in time in which there is a sequence of transactions $T_1, \ldots, T_n$, such that for all i such that $i < n$, T_i waits for T_{i+1}, and also T_n waits for T_1.

(a) Assume that transactions can acquire locks on individual tuples ("tuple level locking"), and consider the following three transaction:

- T_1: $L(E)\ R(E)\ L(H)\ R(H)\ W(E)\ UL(E)\ UL(H)$
- T_2: $L(A)\ R(A)\ L(E)\ R(E)\ W(A)\ UL(A)\ UL(E)$
- T_3: $L(G)\ R(G)\ L(D)\ R(D)\ W(D)\ UL(D)\ UL(G)$

Note: L = lock, R = read, W = write, UL = unlock.

Is there some schedule where a convoy occurs? If so, draw the waits-for graph that shows the convoy. If not, explain why not.

(b) For the same scenario as part (a), is there some schedule where a deadlock occurs? If so, draw the waits-for graph that shows the deadlock. If not, explain why not. Label the area in a wait-for graph with the object (tuple) that is being waited for.

(c) Now assume that transactions can acquire locks only on whole blocks. For example, to lock tuple A, a transaction must actually acquire a lock on block 1. For the same set of transactions as part (a), is there a *new* convoy (i.e., a new set of transactions) that could occur now that could not occur under tuple-level locking? If so, draw the waits-for graph that illustrates the convoy.

(d) With block-level locks and the transactions of part (a), is there a *new* deadlock (i.e., a new set of transactions) that could occur not that could not occur under tuple-level locking? If so, draw the waits-for graph that illustrates the deadlock. Again, label the arcs in a waits-for graph with the object (block) that is being waited for, if any such deadlock exists.

(e) Next we consider a different set of transactions and the same questions posed in part (a) through (d). Consider the transactions

- T_1: $L(E)$ $R(E)$ $L(H)$ $R(H)$ $W(E)$ $UL(E)$ $UL(H)$
- T_2: $L(J)$ $R(J)$ $L(E)$ $R(E)$ $W(E)$ $UL(E)$ $UL(J)$
- T_3: $L(C)$ $R(C)$ $L(H)$ $R(H)$ $L(J)$ $R(J)$ $W(H)$ $UL(H)$ $UL(J)$ $UL(C)$
- T_4: $L(K)$ $R(K)$ $L(E)$ $R(E)$ $W(E)UL(E)$ $UL(K)$

With record-level locking, is there some schedule where a convoy occurs? If so, draw the waits-for graph that shows the convoy. If not, explain why not.

(f) For the same scenario as part (e), is there some schedule where a deadlock occurs? If so, draw the waits-for graph that shows the deadlock. If not, explain why not. Label the arcs in a waits-for graph with the object (tuple) that is being waited for.

(g) Again assume that transactions can acquire locks only on whole blocks. For the same set of transactions as part (e), is there a *new* convoy (i.e., a new set of transactions) that could occur now that could not occur under tuple-level locking? If so, draw the waits-for graph that illustrates the convoy.

(h) With block-level locks and the transactions of part (e), is there a *new* deadlock (i.e., a new set of transactions) that could occur not that could not occur under tuple-level locking? If so, draw the waits-for graph that illustrates the deadlock. Again, label the arcs in a waits-for graph with the object (block) that is being waited for, if any such deadlock exists.

Exercise 7 (20 points):

Consider the three transactions:

- T_1: $R(a)$ $R(b)$ $W(a)$ $W(b)$ $W(c)$
- T_2: $R(c)$ $R(b)$ $W(b)$
- T_3: $R(c)$ $R(a)$ $W(c)$

Each part below asks for a schedule of a particular type, for transactions T1, T2, and T3 (and no other transactions). Please represent you schedule as a 2-D grid, with time flowing down the vertical axis and a separate column for each data value. Also, please show the validation or commit points in each schedule. *Careful: in some cases, there may be no schedule satisfying the requested properties. If there is no schedule, simple write* "NO SCHEDULE EXISTS".

Also for this problem, please use the following definitions:

- Schedule S is recoverable if each transaction commits only after all transactions from which it read have committed.
- Schedule S avoids cascading rollback if each transaction may read only those values written by committed transactions.
- Schedule S is strict if each transaction may read and write only items previously written by committed transactions.

a. A serializable, but not recoverable schedule generated by a locking based scheduler.

a	*b*	*c*	commit point

b. A serializable, but not recoverable schedule generated by a validation scheduler.

a	*b*	*c*	commit point

c. A serialazible and recoverable schedule, that does not avoid cascading rollback, generated by a validation scheduler.

a	*b*	*c*	commit point

d. A schedule that *cannot* by generated by a validation-based scheduler, but can be generated by a strict, locking-based scheduler.

a	*b*	*c*	commit point