CS315A Final Examination

Indian Institute of Technology, Kanpur

April 20, 2016

Time: 3 hrs. Total Marks: 200

INSTRUCTIONS. Support your answers with clear arguments and/or examples.

Problem 1. *SQL.* (30)

Consider a movies database, with the following abbreviated schema.

Movies(*Mname*, *Date*, *Budget*, *GrossEarnings*)
Rolein(*Mname*, *Name*, *Role*)

Mname is the name of the movie and *Name* is the name of a person playing some role in the movie. Role takes values as "Lead Actor", "Supporting Actor", "Producer", "Director", etc.. Express the following queries in SQL.

1. Find Mnames M and Names N such that N worked both as a "producer" and a "lead actor" in the movie named M. (10)

```
select    r.mname, r.name
from      rolein r, rolein s
where     r.mname = s.mname and r.name = s.name
          and r.role LIKE 'producer' and s.role LIKE 'lead actor'
```

2. Find Mnames M such that M grossed more than Rs. 10 Crore in earnings and had at least two distinct persons each of whom played at least two distinct roles in M, none of the roles being a "Director" role. (20)

```
with DualRole as
          ( select r.mname, r.name
          from      Rolein r
          where     r.mname = v.mname
          and       r.name not in (select s.name
                                   from Rolein s
                                   where s.mname = r.mname and s.role LIKE "director")
          group by r.mname, r.name
          having    count(distinct r.role) >=2
)
select    d.mname
from      DualRole d
where     GrossEarnings >= 100000000
group by  d.mname
having    count (distinct d.name) >= 2
```

Problem 2. *Concurrency Control* (30)

1. Give an example of a conflict-serializable schedule involving three or more transactions that cannot be produced if all the transactions followed 2PL protocol. (That is, there is no possible placement of lock/unlock instructions in accordance with the 2PL rule in the schedule). Argue your answer. (10)

	T_1	T_2	T_3
10	$r(A)$		
20		$w(A)$	
30			$r(A)$
40	$r(B)$		
50		$w(B)$	

Let $t_1, \ldots, t_4$ be the following time instants.

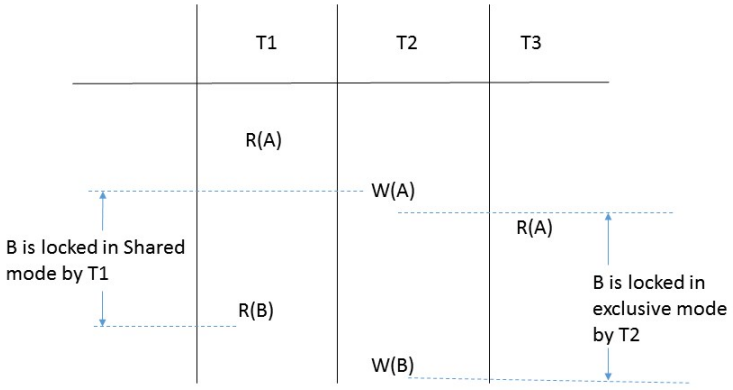
(a) t_1: time when T_1 acquires lock on B.

(b) t_2: T_2 acquires lock on A.

(c) t_3: T_2 acquires lock on B.

(d) t_4: T_3 acquires lock on A.

T_2 can obtain exclusive lock on A only after T_1 completes read (A) and releases shared lock. Since T_1 follows 2PL, T_1 must acquire shared lock on B before releasing shared lock on A. So, $t_1 < t_2 < 20$. Similarly, $t_3 < t_4 < 30$. So, T_1 has a shared lock on B in the region $[t_1, 40] \supset [20, 40]$ and T_2 has an exclusive lock on B in the time region $[t_3, 50] \supset [30, 50]$. The intervals $[20, 40]$ and $[30, 50]$ have a non-trivial intersection, which means that T_1 and T_2 both have locks on B in conflicting mode in the common time period. This is a contradiction, implying that the schedule cannot be produced if all transactions follow 2PL protocol.



Contradiction: B is locked in exclusive and shared mode by different transactions for overlapping time intervals.

2. In the implementation of the lock table in a lock manager, explain where and how semaphores are used during lock/unlock operations? (5)

Lock manager is implemented as a hash table on data items that are locked. For every data item, up to two queues are maintained, one each for transactions waiting on each lock mode. For a lock operation, a sempahore *mutex.down* operation is invoked, and transactions are checked and lock is granted (if possible). After this the mutex is released (*up* is invoked). For unlock operation, the same operations are done, (a) *down* is invoked, transactions and checked and lock is granted (where possible), and (b) mutex is released.

A semaphore protects the integrity of the waiting queues for each data item by regulating access to them as a critical section.

For concurrency, the hash table is partitioned into $k > 1$ parts, and each part is protected by a different semaphore. This avoids a single semaphore to be a bottleneck in a high throughput system.

3. Consider a locking-based system for concurrency control with the following change. Each transaction T_i upon entry gets a priority number $T_i.P$. If a transaction T_i requests a lock, and it has higher priority than the transaction T_j holding the lock, then T_j is rolled back. The waiting queue for a lock at the data item is always maintained in descending order by the priority value of the transactions. If the lock is free, the transaction with the highest priority among all waiting transactions is given the lock. At any time, a lower priority transaction is not given a lock if a higher priority transaction is waiting for a lock. Additionally, 2PL protocol is followed. Explain with reasons, one advantage and one disadvantage of this scheme over a pure 2PL locking scheme. (7.5+7.5)

   Assume that the priorities of all transactions are distinct. Then, in this scheme, there is no deadlock. This is because, a transaction T_i waits for a lock granted to T_j only if $T_i.P < T_j.P$. The waits-for graph is therefore acyclic. This is an advantage over 2PL scheme, which can have deadlock and would require periodic cycle detection algorithm to run over the waits-for graph created from the lock table.

   The disadvantage of this scheme is the extent of rollbacks possible. The system trades more rollbacks to avoid deadlock. Suppose that a system has very low probability that a transaction falls into a deadlock. Then, 2PL would be a good system, whereas, the proposed scheme would still have a considerable probability that a transaction is rolled back.

   Another disadvantage is the long waiting and possible starvation of low priority transactions.

Problem 3. *Phantoms* (20)
Consider the database schema: STUDENT(<u>NAME</u>, DID, CPI). Suppose the following two transactions are executed concurrently:

T_1: update STUDENT set CPI = 10.0 where DID = 'CSE'

T_2: insert into STUDENT values ('Bhuwan', 'CSE', 9.0)
insert into STUDENT values ('Gauri', 'CSE', 9.0)

Suppose that Bhuwan and Gauri were not in the STUDENT table before the start of T_1 or T_2. Suppose also that strict-2PL protocol is being followed.

1. Suppose that all transactions uses 2PL locking at the record level. Give a schedule to show how it may be possible that after both T_1 and T_2 have committed, that Bhuwan and Gauri have different CPI values. How did this anomaly arise? (10)

Inconsistency may arise only if record-level locking is used by both, the phantom phenomenon. Using MGL or index locking, this cannot happen.

T_1	T_2
write("Abhinav",10)	
write ("Faldu",10)	
	write("Bhuwan",9.0)
	write ("Gauri",9.0)
	commit
write("Gauri",10.0)	
commit	

Since locking is done at record level and strict 2PL is used, there is no conflict at the record level. Transaction T_1 proceeds in alphabetic order, locking records as needed. T_2 inserts records in alphabetic order, locking them until commit. T_2 precedes T_1. T_1 misses the inserted record for "Bhuwan" but sees the record for "Gauri" and updates the CPI for this record to 10.0. This gives an inconsistent state.

2. Argue how multi-granularity locking would solve this issue. Alternatively, show how index locking solves this problem. (10)

   MGL would solve this issue because, T_1 would need to obtain an SIX lock on the STUDENT table and T_2 would obtain IX lock on the table and X locks on the individual inserted records. These conflict, and so 2PL rules will ensure serializability.

Problem 4. *Basic Recovery* (20)
Explain with reasons.

1. A database system wants to steal a page P sitting in frame F from the buffer and some transaction T currently has an exclusive lock on a record inside P. Would it be correct to steal P? Assuming WAL rule is being enforced. (5)

   As long as WAL rule is enforced, there is no problem in stealing a page that has a record with an exclusive lock on it by some transaction. Even if the page is not in the buffer, the lock table is in memory, and the lock continues to be held!

2. Suppose there is a database page on disk with PAGELSN > FLUSHLSN . What is the problem in the system, and why? (5)

   WAL has been violated. Atomicity may be compromised.

3. A transaction T has completed and a commit record has been created in the log in memory with LSN 100. Suppose FLUSHLSN is 0. The system has a heuristic that it will flush only when the LSN of the current record is at least a multiple of 4096. Is this heuristic consistent with ACID properties? When does the transaction T commit? (5)

   Heuristic is consistent, as long as WAL rule is followed. Commits are delayed leading to increased waiting times for transactions. T commits when LSN L, with L at least 4096 is created and this log page is forced to disk.

4. During fuzzy checkpointing (ARIES algorithm), the dirty page table is written to log as part of the checkpoint record. Concurrently, the buffer manager may force some dirty pages to disk, causing the dirty page table to change. Argue why this does not affect the correctness of the redo/undo procedures for recovery. (5)

   There are two concurrent threads running, namely, writing the dirty page table in units of pages, onto the log. The second thread is the usual buffer manager that may

force some dirty pages to disk to create space. Consider a page table entry E with fields $E.Pageid$ and $E.\text{RECLSN}$. Suppose the buffer manager forces this page to disk. This would remove the page from DPT, and subsequently, the checkpoint procedure will not find the DPT entry E. This is good, since the RECLSN for this page is at least the current LSN. On the other hand, if the checkpoint procedure had already written this entry to the log, and then the buffer manager forces this page to the disk, then, the recovery procedure will start from $E.\text{RECLSN}$ or before. This is not necessary since the page has been written to disk, but is not incorrect. Some extra disk reads and checks get done during the recovery process, but does not make it incorrect.

Problem 5. ARIES *Recovery.* (40)

Consider the log below. Some of the information for the log records is not shown for sake of brevity. The checkpoint record is shown in detail below. The notation $P1.2$ indicates record 2 in the page $P1$. The log record $T1$ writes $P1.2, 40, 60$ indicates that the old value of $P1.2$ was 40 and the updated value is 60. Assume that all *PageLSN*'s are NIL. The system *crashes* after the last log record is written.

LSN	**Record**	**prevLSN**	**UndoNextLSN**
0	Update: $T1$ writes $P1.5$, 20, 40	NIL	—
10	Update: $T1$ writes $P1.2$, 40 , 60	0	–
20	Checkpoint		
30	Update: $T2$ writes $P2$, 80, 10	NIL	–
40	$T1$ writes $P3.5$, 20, 90	10	–
50	$T2$ writes $P4.1$, 30, 50	30	–
60	ABORT: $T1$	40	—
70	CLR: $T1$, $P3.5$, 20	60	10
80	???	???	???
90	END : $T1$	80	–
100	Update: $T3$ writes $P3.4$, 50, 60	NIL	–
110	ABORT: $T2$	50	—
120	CLR: $T2$, $P4.1$, 30	110	30

The checkpoint record has the following entries.

Transactions Table

Txn	lastLSN
T1	10

Dirty Page Table

PageID	PageLSN	RecLSN
P1	10	0

1. Write down the missing record in the log with LSN 80 using the same format. (5)

LSN	**Record**	**prevLSN**	**UndoNextLSN**
80	CLR $T1$ $P1.2$ 40	70	0
85	CLR $T1$ $P1.1$ 20	80	—

2. Show the transaction table at the end of the ANALYSIS phase. (7.5)

Transactions Table

Txn	lastLSN
T2	120
T3	100

3. Show the dirty pages table at the end of the ANALYSIS phase. (7.5)

Dirty Page Table

PageID	PageLSN	RecLSN
P1	10	0
P2	30	30
P3	100	40
P4	120	50

4. What actions will be done in the REDO phase? List the LSN's in the order that they are redone. (10)

   Redo phase starts from the log record whose LSN equals the smallest of the RECLSN field of the Dirty page table constructed in the Analysis phase. In this case, this is 0. So all log records with LSNs 0,10,30,40, 50, 60, 70, 80, 85, 90, 100, 110 120 will be redone.

5. What are the log records that are written in the UNDO phase. Start from LSN 200. Keep the difference between adjacent log records to be 10. (10)

LSN	**Record**	**prevLSN**	**UndoNextLSN**
200	ABORT T3	120	—
210	CLR: T3, P3.4, 50	200	—
220	END T3	210	
230	CLR: T2, P2, 80	120	—
240	END T2	230	

Problem 6. *Query Processing.* (60)
The questions use the following tables:

1. *EMP*(*eno*: integer, *ename*: string, *salary*: integer, *did*:integer), and
2. *DEPT*(*did*: integer, *dname*: string, *location*: string, *mgrno*: integer).

EMP.did is a foreign key referencing *DEPT* and *DEPT.mgrno* is a foreign key referencing *EMP*. *EMP* table has 100,000 tuples, with 100 bytes per tuple, and 40 tuples per page. *DEPT* table has 1000 tuples, with 100 bytes per tuple and 40 tuples per page. There are 50 distinct salary values and 100 distinct location values. You may use the assumption that values are distributed uniformly in a column or in a set of columns.

1. Suppose *EMP* has a B^+-tree clustered index on (*Deptid, Salary*). It is needed to scan *EMP* on *Deptid* in non-descending sorted order. What is the best algorithm for doing this and calculate the I/O cost for doing the same. Assume that you are provided with memory buffer of 100 pages. (10)

   Say that each index record of (*did*, *salary*, *) takes 20 bytes, so there are 200 index tuples in a page. Scanning the leaf pages and following the pointers will give *EMP* records sorted by (*did*, *salary*) and therefore on (*did*). The cost is as follows.

   (a) IndexScan cost. Index has three levels, so to get to the left most tuple requires 3 I/Os. Next, scanning the index requires 100,000/200 = 500 pages.
   (b) FileScan cost. 100,000/40 = 2500 pages.

   Total = 3003 pages. Memory consumption is 2 buffer pages.

2. Suppose *EMP* has an extendible hash index on (*Deptid*). A scan of *EMP* on *Deptid* in non-descending sorted order is needed. What is the best algorithm for doing this and calculate the I/O cost for doing the same. Assume that you are provided with memory buffer of 100 pages. (10)

EMP has 2500 pages, and a two-pass sort would require $3 \cdot 2500 = 7,500$ I/Os with 100 pages of memory buffer, assuming the final write does not happen and the sorted stream is in memory. The first pass will take an additional 503 page I/Os to access EMP via the index on *ENo* or any clustered index on EMP.

Since it is not mentioned that *EMP* is clustered by *did*, we cannot take advantage of it. Extendible hash indices are always non-clustered.

3. Consider the query that finds all employee ids and names working in a department located in "Kanpur".

```
select   E.eno, E.ename
from     EMP E, DEPT D
where    E.did = D.did and D.location LIKE '%KANPUR%'
order byE.eno
```

Suppose EMP table has a clustered B^+-tree index on E.eno and DEPT table has an extendible hash index on DEPT.did and a non-clustered B^+-tree index on DEPT.location. (Note that the predicate Like '%Kanpur%' does not match the index). Suppose that the number of memory pages available is 100. Give an efficient query processing plan for executing this query and estimate the I/O cost of executing the query. (20)

Dept has 1000 tuples and $1000/40 = 25$ pages, so it will fit in memory. One access plan would be to load *Dept* into memory and filter only those tuples satisfying D.location LIKE '%KANPUR%'. The resulting size is no more than 25 pages. A nested-loop join or a hash join of E with D (D is in memory) can now be done. E is scanned ordered by $E.no$ using clustered index scan on $E.no$. Assume there are 400 index records for EMP.eno to a page . Cost of clustered index scan is $3 + 100,000/400 + 2500 = 2753$ pages. Cost of join $= 25 + 2753 = 2778$ pages.

The access plan $D \bowtie E$, where, D is outer and E is inner relation would expensive since E does not have a matching index for *did*.

Given the hash index on Dept.did, it is possible to have a nested-loop join between EMP and DEPT with DEPT as the inner relation and EMP as the outer relation. However this will require at least 1 scan of all the tuples of EMP and thereafter finding all the matching tuples of DEPT. The latter will require a minimum of 1 access of the index and 1 access of the tuple, leading to significant expense (at least 2500×3 pages). A less expensive method would be to do a hash join as described above.

4. Suppose that the hash-index on DEPT.did has been dropped. Evaluate the I/O cost of executing the query in the best possible manner using a hash-join algorithm. You are given 100 pages in buffer. Detail the algorithm and then estimate the cost. (20)

This has been done as the first alternative of the previous sub-question.