INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

	Stamp / Signature of the Invigilator

EXAMINATION (End Semester)	**SEMESTER (Spring)**

Roll Number										**Section**		**Name**	
Subject Number	C	S	3	0	2	0	2	**Subject Name**	*Database Management Systems*				
Department / Center of the Student											**Additional sheets**		

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as '**unfair means**'. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

Instructions: Answer all SEVEN questions. Time = 3hrs. Total marks = 100. Write your answers only in the space provided. Show the solution steps. Answers without explanation will be penalised. The question paper has total 28 pages.

ROUGH WORK

1.(a) We have an initially empty buffer pool with 5 buffer frames. We have six data pages U-Z. The access patterns of the pages are: X Y Z X W V U W X Y Z U V W X.

Fill up the table below for the buffer replacement policies mentioned. [3]

Policy	*Number of hits*	*Data pages in buffer pool after completion*
LRU	4	UVWXZ
MRU	8	VWXYZ
CLOCK	4	VWXYZ

(b) Consider a disk with block size B = 2048 bytes. A block pointer is P = 10 bytes long. A file has r = 5,000,000 EMPLOYEE records of fixed size 400 bytes. Suppose that the file is ordered by the key field *emp_id* (15 bytes), and we want to construct a primary index on *emp_id*. Calculate -
(i) The number of first-level index entries and the number of first-level index blocks, in case of single level indexing system. Determine the number of block accesses needed to search for and retrieve a record from the file, given its *emp_id* value using the single level primary index structure. [3]

Single Level Primary Indexing Structure:
Number of records per block = 2048/400 = 5
Number of blocks = 5000000/5 = 1,000,000
1st Level index entries = 1,000,000 **(1M)**
Number of index entries per block = 2048/25 = 81
Number of Index blocks in single level primary indexing system = 1,000,000/81 = 12345 **(1M)**
Number of block accesses required for searching a record from a single level primary indexing system = $\log_2(12345)$ + 1 = 15 **(1M)**

(ii) Suppose, if we want to construct an index structure on employee's Aadhar number whose size is 20 bytes (one of the attributes of the record), on the above mentioned ordered file. Assume employee's Aadhar number is NOT NULL. Determine the number of first-level index entries and the number of first-level index blocks, in case of single level indexing system. Determine the number of block accesses needed to search for and retrieve a record from the file, given its employee's Aadhar number value using the single level index structure. In case of multi-level indexing based Aadhar number, how many block accesses are required to search and retrieve a record from the file. [4]

Secondary Indexing Structure
Number of 1st level index entries = 5,000,000 **(0.5)**
Number index entries per block = 2048/30 = 68
Number of index blocks in single level secondary indexing structure = 5,000,000/68 = 73530 **(1M)**
Number of block accesses required for a single level secondary indexing structure = $\log_2(73530)$ + 1 = 18 **(1M)**
In case of multilevel secondary indexing structure, number of levels = $\log_{68}(73530)$ + 1 = 4 **(1M)**
Number of block accesses required for a multi-level secondary indexing structure = 4 + 1 = 5 **(0.5)**

2.(a) Suppose that we are using extendable hashing on a file that contains records with the following search-key values: 31, 77, 23, 51, 801, 163, 15, 101, 43, 82. Show the extendable hash structure incrementally (with appropriate tables and figures) for every key insertion, if the hash function is h(x) = x mod 16 and buckets can hold three records. Provide all intermediate steps and details for each key insertion. Note: Consider the start of the extendable hashing from the MSB (Most Significant Bit) value of the hash. [10]

31 ---> 31 mod 16 = 15 = 1111
77 ---> 77 mod 16 = 13 = 1101
23 ---> 23 mod 16 = 7 = 0111
GD = Global Depth (Size of a Directory)
LD = Local Depth

GD (0)				**LD**
	31			**0**
	31	77		**0**
	31	77	23	**0**

51 ---> 51 mod 16 = 3 = 0011

GD (1)				**LD**
0	23	51		**1**
1	31	77		**1**

801 ---> 801 mod 16 = 1 = 0001

GD (1)				**LD**
0	23	51	801	**1**
1	31	77		**1**

163 ---> 163 mod 16 = 3 = 0011

GD (2)				**LD**
00	51	801	163	**2**
01	23			**2**
10 11	31	77		**1**

15 ---> 15 mod 16 = 15 = 1111

GD (2)				**LD**
00	51	801	163	**2**
01	23			**2**
10 11	31	77	15	**1**

101 ---> 101 mod 16 = 5 = 0101

GD (2)				**LD**
00	51	801	163	**2**
01	23	101		**2**
10 11	31	77	15	**1**

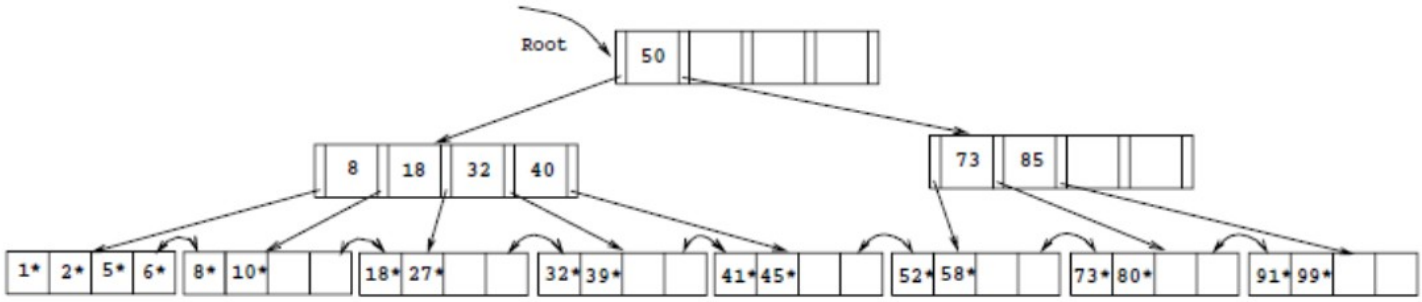
43 ---> 43 mod 16 = 11 = 1011

GD (2)				**LD**
00	51	801	163	**2**
01	23	101		**2**
10	43			**2**
11	31	77	15	**2**

82 ---> 82 mod 16 = 2 = 0010

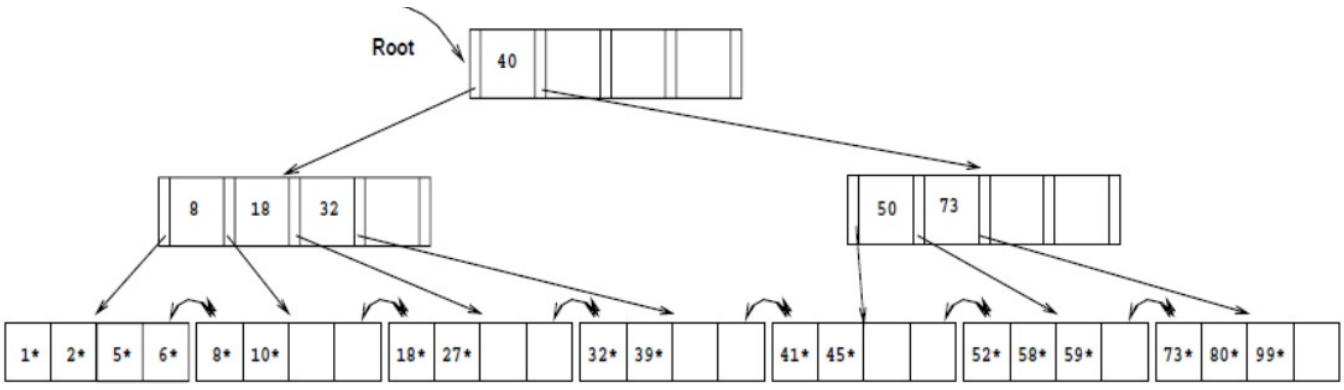
GD (3)				LD
000	801			3
001	51	163	82	3
010 011	23	101		2
100 101	43			2
110 111	31	77	15	2

(b) We created the following B+ tree where each non-root node can store at most 4 keys and at most 5 pointers. When two nodes are merged the right sibling is considered first in case of a tie.



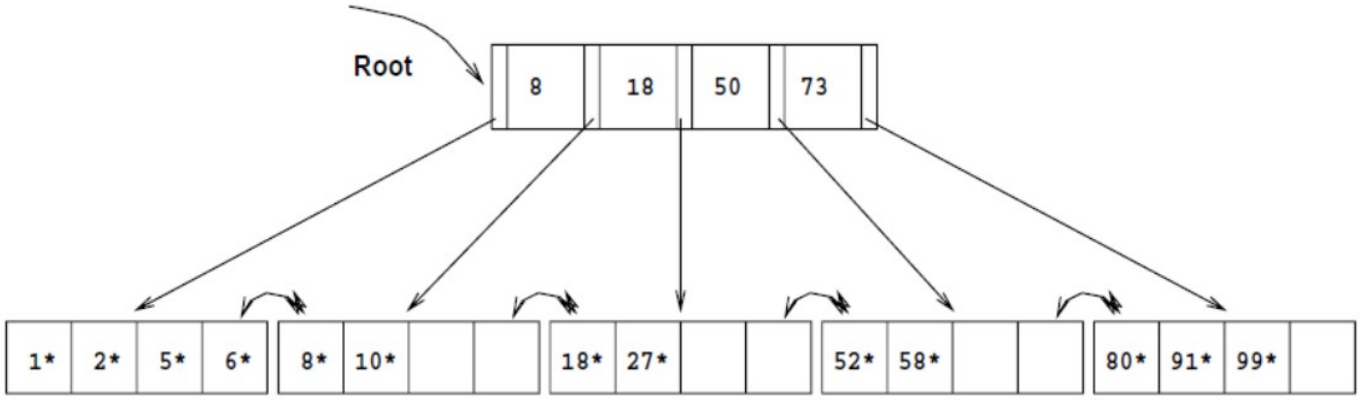
(i) Show the entire tree after inserting the key 59, and then deleting the key 91. [5]

ANS:



(ii) Show the tree after sequentially deleting the keys 32, 39, 41, 45, and 73 from the original tree. [5]

ANS:



3.(a) We want to sort a table which has 1960 pages with 8 available buffer pages using the external merge sort algorithm. [5]

What is the total number of passes required?	4
How many sorted runs will be produced after each pass? Write comma separated values.	245, 35, 5, 1
How many pages will be in each sorted run for each pass?	8, 56, 392, 1960.
How many I/Os does the entire sorting operation take?	4 * 2 * 1960 = 15,680 I/Os.
How many buffer pages are required to sort the table in two passes?	33 buffer pages.

(b) Now we want to sort a table which has 200 pages. Suppose that during Pass 0, you have 10 buffer pages available to you, but for Pass 1 and onwards, you only have 5 buffer pages available. [5]

How many sorted runs will be produced after each pass?	20, 5, 2, 1
How many pages will be in each sorted run for each pass?	10, 40, (160 and 40), 200.
How many I/Os does the entire sorting operation take?	(2 * 200 I/Os per pass) * 4 passes = 1600 I/Os.
Does increasing the number of buffer pages does not affect the number of I/Os performed in Pass 0 of the external sort?	True. Regardless of the number of buffer pages, every pass of an external sort (including Pass 0) performs the same number of IOs.
Does sorting an already sorted table takes fewer IOs than sorting a randomly-arranged table?	False. Regardless of how a table (of fixed length) is sorted, the sorting algorithm always takes the same number of IOs.

(c) Consider relations *Course(roll, cid)*, *Hall(roll, name, hid)*, and *Dept(roll, branch, bid)* to be joined on the common attribute *roll*. Assume that there are no indexes available on the tables to speed up the join algorithms. I/O cost is simply the number of pages read and written. There is an output buffer block and cost of writing the final result may be ignored.

- There are *B = 750* pages in the buffer
- Table *Course* occupies *M = 2,500* pages with *80* tuples per page
- Table *Hall* occupies *N = 600 pages* with *300* tuples per page
- Table *Dept* occupies *O = 1,500* pages with *150* tuples per page
- The join result of *Course* and *Hall* occupies *P = 500* pages

Answer the following questions. Express the answer in terms of B, M, N, O, P. [5]

What is the I/O cost of a simple nested loop join with *Course* as the outer relation and *Hall* as the inner relation?	M + M × N = 2500 + 2500 × 80 × 600 = 120, 002, 500
What is the I/O cost of a block nested loop join with *Hall* as the outer relation and *Dept* as the inner relation?	N + N/(B−2) x O = 600 + ⌈600 748 ⌉ × 1, 500 = 600 + 1, 500 = 2, 100
For a sort-merge join with *Course* as the outer relation and *Dept* as the inner relation: What is the cost of sorting the tuples in Course on attribute *roll*?	2M X passes = 10,000 $passes = 1 + \lceil \log_{B-1}(\lceil \frac{M}{B} \rceil) \rceil$
For a sort-merge join with *Course* as the outer relation and *Dept* as the inner relation: What is the cost of the merge phase in the worst-case scenario?	M × O = 2, 500 × 1, 500 = 3, 750, 000
For a sort-merge join with *Course* as the outer relation and *Dept* as the inner relation: What is the cost of the merge phase assuming there are no duplicates in the join attribute in each table?	M + O = 2, 500 + 1, 500 = 4, 000

4.(a) We have a schema with relation *R(BC)*, *S(ACD)*, *T(EBD)*. Show the sequence of steps (and the corresponding equivalence rule) required to transform the query $\Pi_A\left(\left(R \bowtie_{B=C} S\right) \bowtie_{D=E} T\right)$ to $\Pi_A\left(\left(\Pi_E(T) \bowtie_{E=D} \Pi_{ACD}(S)\right) \bowtie_{C=B} R\right)$. [3]

ANS:

(a) Associativity of the join: $\pi_A(\mathbf{R} \bowtie_{B=C} (\mathbf{S} \bowtie_{D=E} \mathbf{T}))$

(b) Commutativity of the join: $\pi_A((\mathbf{S} \bowtie_{D=E} \mathbf{T}) \bowtie_{C=B} \mathbf{R})$

(c) Commutativity of the join: $\pi_A((\mathbf{T} \bowtie_{E=D} \mathbf{S}) \bowtie_{C=B} \mathbf{R})$

(d) Pushing projection π_A to the first operand of the join: $\pi_A(\pi_{AC}((\mathbf{T} \bowtie_{E=D} \mathbf{S})) \bowtie_{C=B} \mathbf{R})$

(e) Pushing projection to the first operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(\mathbf{T}) \bowtie_{E=D} \mathbf{S})) \bowtie_{C=B} \mathbf{R})$. This is possible if *AC* are the attributes of **S** and not of **T**.

(f) Pushing projection to the second operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(\mathbf{T}) \bowtie_{E=D} \pi_{ACD}(\mathbf{S}))) \bowtie_{C=B} \mathbf{R})$.

(g) Reverse of pushing a projection: $\pi_A((\pi_E(\mathbf{T}) \bowtie_{E=D} \pi_{ACD}(\mathbf{S})) \bowtie_{C=B} \mathbf{R})$

(b) Consider the relation *R(A, B, C)* with the following properties:

- *R* has 5,000 tuples with 5 tuples per page
- Each attribute is an integer in range 1-1000
- Attribute *A* is a candidate key
- Unclustered hash index on attribute *A*
- Clustered B+ tree index on attribute *B*
- Attribute *B* has 1,000 distinct values in *R*
- Attribute *C* has 500 distinct values and an unclustered 3-level B+ tree index

What are the estimated I/O cost of the following relational algebra operations in terms of number of page access? Assume, that the best index choice is used. The indices have appropriate storage structures. [4]

Query	*Cost*	*Index used with explanation*
$\sigma_{A=90}(R)$	Cost is 1.2 (searching the index) + 1(retrieving data).	Since the hash index is on the candidate key, query has at most one tuple.
$\sigma_{B=90}(R)$	Cost is depth of the tree + 1.	B has 1,000 distinct values in R, there are about 5 tuples per value. Query likely to retrieve 5 tuples. The index is clustered and because there are 5 tuples per page, the result fits in 1 page.
$\sigma_{C=90}(R)$	Cost is 4+10 = 14.	Query produce 10 tuples (5000/50). We conservatively estimate that these tuples will occupy 2 pages (index entries are typically much smaller than the data file records). Thus, the cost of retrieving all the pointers is 3 (to search the B+ tree for the first page of pointers in the index) + 1 (to retrieve the second page of the index) = 4. Since the index is unclustered, each of the 10 tuples in the result can be in a separate page of the data file and its retrieval may require a separate I/O.
$\Pi_{AC}(R)$	Since the original file has 5,000/5=1000 blocks, the cost of the operation is 1,000(scan of the file) + 2/3*1,000 (cost of writing out the result).	Since we do not project out the candidate key, the projection will have the same numbeof tuples as the original. In particular, there will be no duplicates and no sorting will be required. The output will be about 2/3 of the original size assuming that all attributes contribute equally to the tuple size.

(c) Consider the following schema:
Sailors(sid, sname, rating, age)
Reserves(sid, did, day)
Boats(bid, bname, size)

Reserves.sid is a foreign key to *Sailors* and *Reserves.bid* is a foreign key to *Boats.bid*.

We are given the following information about the database: Reserves contains 10,000 records with 40 records per page. Sailors contains 1000 records with 20 records per page. Boats contains 100 records with 10 records per page. There are 50 values for *Reserves.bid*. There are 10 values for *Sailors.rating* (1…10). There are 10 values for *Boat.size*. There are 500 values for *Reserves.day*

Consider the following query:
SELECT S.sid, S.sname, B.bname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND
B.size > 5 AND R.day = 'July 4, 2003';

Assuming uniform distribution of values and column independence, estimate the number of tuples returned by this query. Explain. [3]

ANS:
The maximum cardinality this query is 109
Reduction Factors:
1/2 for B.size > 5,
1/500 for R.day = 'July 4, 2003',
1/100 for R.bid = B.bid, and
1/1000 for S.sid = R/sid
So number of tuples returned is (1/2)(1/500)(1/100)(1/1000)(109) = 10

5.(a) Consider the following schedule with one action missing: *r1(A); r2(B); ???; w1(C); w2(A);* Find out what read action could replace the ??? and make the schedule non-serializable. Explain your answer. [1]

ANS:
The only constraint on a serial order so far is that T1 must precede T2, because T1 reads A before T2 writes A. The only way we could get a cycle in the precedence graph is if ??? required T2 to precede T1. A read of anything by T1 will not introduce any constraints, but r2(C) will cause an arc from T2 to T1 and lead to a cycle.

(b) Consider the following two transactions T_1 and T_2. How many schedules involving T_1 and T_2 are recoverable? [5]

T_1: w1(A); w1(B); r1(C); commit1;
T_2: w2(A); r2(B); w2(C); commit2;

ANS:
Each transaction reads only one element, so the question is, does it read from the other? If so, then in order for the schedule to be recoverable, it must commit after the other as well. That is, the two constraints on the schedule can be expressed:

1. If r1(C) follows w2(C) then c1 must follow c2.
2. If r2(B) follows w1(B) then c2 must follow c1.

The partial orders that violate (1) look like:

```
o->o->o->o
  |     ^
  v     |
o->o->o->o
```

With a little effort, we can find that there are 12 orders for the eight nodes involved.

The orders that violate (2) are similar; they have the constraints:

```
o->o->o->o
    |   ^
    v   |
o->o->o->o
```

There are 10 of these. Since there are {8 choose 4} = 70 orders in all, 48 of them are recoverable.

(c) For the schedules shown below mark yes/no if they are conflict serializable, view serializable, recoverable, avoids-cascading-rollbacks. [2]

Schedule	Conflict Serializable	View Serializable	Recoverable	Avoid Cascading Roll Back
T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit	NO	YES	YES	YES
T1:R(X), T2:W(X), T1:W(X), T3:R(X), T1:Commit, T2:Commit, T3:Commit	NO	YES	YES	NO

(d) Consider concurrency control by timestamps. Below are several sequences of events, including start events, where *sti* means that transaction *Ti* starts and *coi* means *Ti* commits. These sequences represent real time, and the timestamp-based scheduler will allocate timestamps to transactions in the order of their starts. In each case below, say what happens with the last request. You have to choose between one of the following four possible answers: accepted/ignored/rolled-back. [2]

Actions	*Last action*
st1; st2; r2(A); co2; r1(A); w1(A);	(rolled back)
st1; st2; st3; r1(A); w2(A); w3(A); r2(A);	(rolled back)
st1; st2; r1(A); r2(A); w1(B); w2(B);	(accepted)
st1; st2; st3; r1(A); w3(A); co3; r2(B); w2(A);	(ignored)

6.(a). Suppose the lock hierarchy for a database consists of database, relations, and tuples. If the database consists of hundreds of relations and millions of tuples under each relation. Determine the locks to be acquired by each of the transaction to carry out it's task. [5]

i. Transaction T1 reads a tuple tup-7 in a relation R2. (IS & S)
ii. Transaction T2 modifies a tuple tup-2 in a relation R2. (IX & X)
iii. Transaction T3 wants to read all tuples of relation R2. (IS & S)
iv. Transaction T4 wants to read the entire database. (S)

In view of the above scenario, mention the various sets of transactions that can be allowed to execute concurrently? (T1, T3, T4), (T1, T2)

(b). Explain, clearly (detailed steps) how a transaction T_i performs *DELETE(Q)* and *INSERT(Q)* using time stamp ordering (TSO) protocol. [4]

DELETE(Q)

If TS(Ti) < R-timestamp(Q), then the value of Q that Ti was to delete has already been read by a transaction Tj with TS(Tj) > TS(Ti). Hence, the delete operation is rejected, and Ti is rolled back.

If TS(Ti) < W-timestamp(Q), then a transaction Tj with TS(Tj) > TS(Ti) has written Q. Hence, this delete operation is rejected, and Ti is rolled back.

Otherwise, the delete is executed.

INSERT(Q)

if Ti performs an insert(Q) operation, the values R-timestamp(Q) and W-timestamp(Q) are set to TS(Ti).

(c). Explain the basic principle of simple locking protocol? What are the issues in simple locking protocol in the context of concurrent execution? [2]

Data items should be locked with appropriate locks before their usage and the locks will be released soon after the usage of data items has been completed. With this simple locking protocol, data base consistency is not ensured, due to release of locks immediate release of locks soon after the usage of data items.

Explain the basic principle of two-phase locking protocol? How it addresses the problems of simple locking protocol? What are the limitations of two-phase locking protocol in view of concurrent execution? [2]

In 2-phase locking, transactions will acquire locks in growing phase only and release locks in shrinking phase only. As data items will be locked during the entire transaction, and the locks will be released only at the end, with that it ensure isolation among the executions of concurrent transactions and database consistency is ensured. As the transactions lock the data items through out its execution, deadlocks may occur, and hence roll-backs will take place.

Explain the basic principle of tree protocol? How it addresses the problems faced by two-phase locking protocol? What are the limitations of tree protocol? [2]

Tree protocols follow some prior ordering to access data items. To access a particular data item, first its parent will be locked and then the actual data item will be locked. It will release the locks soon after the usage of data items, and hence deadlocks can be prevented. In addition to actual data item, additional data items (parents) are also locked, with that overhead of locking is involved. For ensuring the recovery, commit-dependency has to be maintained.

7.(a) Consider a database with two elements, *X* and *Y*. The initial values of *X* and *Y* are both 0. We consider three transactions T_1, T_2, and T_3, that modify these elements concurrently:

- T_1: *X := 42*
- T_2: *Y := 20, X := 10*
- T_3: *X := 100, Z := 101*

The log entries with some missing values (???) are shown below under different logging scenario. [10]

Pure Undo Log	*Pure Redo Log*	*Undo-Redo Log(txn, item, before-val, after-val)*
1. $< START\ \ T_2 >$ 2. $< START\ \ T_3 >$ 3. $< T_2, X, 0 >$ 4. $< T_2, Y, 0 >$ 5. $< COMMIT\ \ T_2 >$ 6. $< START\ \ CKPT(T_3) >$ 7. $< T_3, X, ??? >$ 8. $< START\ \ T_1 >$ 9. $< T_3, Z, 0 >$ 10. $< T_1, X, ??? >$	1. $< START\ \ T_2 >$ 2. $< START\ \ T_3 >$ 3. $< T_2, X, 10 >$ 4. $< T_2, Y, 20 >$ 5. $< COMMIT\ \ T_2 >$ 6. $< START\ \ CKPT(T_3) >$ 7. $< T_3, X, ??? >$ 8. $< START\ \ T_1 >$ 9. $< T_3, Z, ??? >$ 10. $< T_1, X, 42 >$ 11. $< END\ \ CKPT >$ 12. $< COMMIT\ \ T_3 >$ 13. $< START\ \ CKPT(T_1) >$ 14. $< COMMIT\ \ T_1 >$	1. $< START\ \ T_1 >$ 2. $< START\ \ T_2 >$ 3. $< START\ \ T_3 >$ 4. $< T_3, X, 0, 100 >$ 5. $< T_3, Z, 0, 101 >$ 6. $< COMMIT\ \ T_3 >$ 7. $< T_2, X, 100, 10 >$ 8. $< START\ \ CKPT(T_1, T_2) >$ 9. $< T_2, Y, ???, ??? >$ 10. $< END\ \ CKPT >$ 11. $< START\ \ CKPT(T_1, T_2) >$ 12. $< COMMIT\ \ T_2 >$ 13. $< T_1, X, ???, ??? >$

Pure Undo Logging	
What should be the missing entry in line 7. Explain.	10
What should be the missing entry in line 10. Explain.	100
If the database crashes immediately after writing the above log entries, then subsequently performs recovery. What is the state of *X* at the time of crash?	10
Pure Redo Logging	
What should be the missing entry in line 7. Explain.	100
What should be the missing entry in line 9. Explain.	101
If the database crashes immediately after writing the above log entries, then subsequently performs recovery. What is the state of *Y* at the time of crash?	20
Undo-Redo Logging	
What should be the missing entry in line 9. Explain	0, 20
What should be the missing entry in line 13. Explain	10, 42
If the database crashes immediately after writing the above log entries, then subsequently performs recovery. Which of the log entry lines will be ignored. Explain.	Line 7

Which of the log entry lines will be redone? Explain.	
Which of the log entry lines will be undone? Explain.	

(b) Consider three transactions, T_1, T_2 and T_3 that exclusive lock (*L*) and unlock (*U*) database items *X*, *Y* and *Z* in the following order:

- T_1 : *L(Y) L(Z) U(Z) U(Y)*
- T_2 : *L(X) L(Z) U(X) U(Z)*
- T_3 : *L(Z) L(Y) U(Z) U(Y)*

We would like trace the execution of the transactions under the *wait-die* and *wound-wait* deadlock prevention strategies. We assume that these transactions execute in a round-robin fashion (T_1 then T_2 then T_3). When it is a transaction's turn, it executes its next lock or unlock step if it can, and otherwise dies or waits or wounds the holder of the lock, as appropriate. If a transaction is waiting when it is its turn, it skips the step and continues waiting. When a waiting transaction is restarted (due to the action of some other transaction) it becomes eligible to run at its very next turn. If a transaction *A* is wounded by some other transaction *B*, transaction *B*'s next action is 'Die'. Assume that the transactions have timestamps in the order of $T_1 < T_2 < T_3$ (T_1 is the oldest).

Fill in the following tables to show the sequence of steps that the three transactions make. Use *L(X)* to denote locking an object *X*, *U(X)* to denote unlocking *X*, "Die" to denote the transaction dying or being wounded, and "Wait" to denote the transaction waiting.

(i) *Wait-Die* Deadlock Prevention. Provide explanations. [5]

Step	T_1	T_2	T_3
1	*L(Y)*	*L(X)*	*L(Z)*
2	*Wait*	*Wait*	*Die*
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

(ii) *Wound-Wait* Deadlock Prevention. Provide explanations. [5]

Step	T_1	T_2	T_3
1	*L(Y)*	*L(X)*	*L(Z)*
2	*L(C)*	*Wait*	*Die*
3			
4			
5			
6			
7			
8			

ANS:

WAIT-DIE

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	Wait	Wait	Die
3	L(C)	Die	Die
4	U(C)	L(A)	L(C)
5	U(B)	Wait	Die
6		L(C)	Die
7		U(A)	Die
8		U(C)	Die
9			L(C)
10			L(B)
11			U(C)
12			U(B)

WOUND-WAIT

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	L(C)	Wait	Die
3	U(C)	L(C)	Wait
4	U(B)	U(A)	Wait
5		U(C)	L(C)
6			L(B)
7			U(C)
8			U(B)

------- END --------