



ABES Engineering College, Ghaziabad
B. Tech Odd Semester Sessional Test-2
DBMS ST-2 Solution

Course Code: KCS501

Course Name: Database Management System

Maximum Marks: 75

Date of Exam: 06-12-22

Time: 2 Hrs

Q.No	Question	Marks	CO	KL	PI
Section-A		Total Marks : 5*2 =10			
1	Attempt ALL Parts				
a)	<ul style="list-style-type: none"> The table “Collection” is in BCNF as there is only one functional dependency “Title Author → Catalog_no” and {Author, Title} is key for collection. Book is not in BCNF because Catalog_no is not a key and there is a functional dependency “Catalog_no → Title Author Publisher Year”. Book is not in 3NF because non-prime attributes (Publisher Year) are transitively dependent on key [Title, Author]. Book is in 2NF because every non-prime attribute of the table is either dependent on the whole of a candidate key [Title, Author], or on another non prime attribute. In table book, candidate keys are {Title, Author} and {Catalog_no}. In table Book, non-prime attributes (attributes that do not occur in any candidate key) are Publisher, Year and Place. 			1	2
b)	<p>Two different deadlock prevention schemes using timestamps have been proposed:</p> <p>1. The wait–die scheme is a nonpreemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (that is, T_i is older than T_j). Otherwise, T_i is rolled back (dies).</p> <p>For example, suppose that transactions T_{22}, T_{23}, and T_{24} have timestamps 5, 10, and 15, respectively. If T_{22} requests a data item held by T_{23}, then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will be rolled back.</p> <p>2. The wound–wait scheme is a preemptive technique. It is a counterpart to the wait–die scheme. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (that is, T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is wounded by T_i).</p> <p>Returning to our example, with transactions T_{22}, T_{23}, and T_{24}, if T_{22} requests a data item held by T_{23}, then the data item will be preempted from T_{23}, and T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, then T_{24} will wait.</p>			1	2

c)	<p>Data Replication</p> <p>Data replication is the process where in a relation (a table) or portion of a relation (a fragment of a table) is duplicated and those duplicated copies are stored in multiple sites (servers) to increase the availability of data.</p> <p>Advantages:</p> <ol style="list-style-type: none"> 1. Increased reliability and availability 2. Queries requesting replicated copies of data are always faster (especially read queries) 3. Less communication overhead <p>Disadvantages:</p> <ol style="list-style-type: none"> 1. More storage space is needed when compared to a centralized system 2. Update operation is costly 3. Maintaining data integrity is complex <p>Data Fragmentation</p> <p>Data can be stored in different computers by fragmenting the whole <u>database</u> into several pieces called fragments. Each piece is stored at a different site.</p> <p>Fragments are logical data units stored at various sites in a distributed <u>database</u> system.</p> <p>Advantages of fragmentation</p> <p>Usage: It seems appropriate to work with subsets of relation as the unit of distribution.</p> <p>Efficiency: Data is stored close to where it is most frequently used.</p> <p>Parallelism: This should increase the degree of concurrency, or parallelism, in the system, thereby allowing transactions that can do so safely to execute in parallel.</p> <p>Security: Data not required by local applications is not stored, and consequently not available to unauthorized users.</p> <p>Disadvantages</p> <p>Performance: The performance of global application that requires data from several fragments located at different sites may be slower.</p> <p>Integrity: Integrity control may be more difficult if data and functional dependencies are fragmented and located at different sites.</p>	1
d)	<p>The lock and unlock instructions for Two-Phase locking are:</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: left;"> <p>T_{31}:</p> <p>lock-S(A)</p> <p>read(A)</p> <p>lock-X(B)</p> <p>read(B)</p> <p>if $A = 0$</p> <p>then $B := B + 1$</p> <p>write(B)</p> <p>unlock(A)</p> <p>unlock(B)</p> </div> <div style="text-align: left;"> <p>T_{32}:</p> <p>lock-S(B)</p> <p>read(B)</p> <p>lock-X(A)</p> <p>read(A)</p> <p>if $B = 0$</p> <p>then $A := A + 1$</p> <p>write(A)</p> <p>unlock(B)</p> <p>unlock(A)</p> </div> </div>	2

e)	<p>Two Phase Locking protocol is conflict serializable. So this is a modified version of the basic 2PL protocol, So serializability should be guaranteed.. and we can get a serializable scheduling by ordering based on Lock points(same as in basic 2PL)</p> <p>Now in Step 1, exclusive locks are acquired to O_1, O_2, O_3, \dots in increasing order of addresses.. Since it is mentioned as exclusive lock, only one transaction can lock the object.</p> <p>Due to acquiring of locks based on ordering of addresses.. and locks aren't released until the transaction completes its operation.. we can prevent the circular wait condition, and hence making it deadlock free. So, it guarantees serializability and deadlock freedom.</p>	2
Section-B		Total Marks : 3*5 = 15
2	Attempt ANY ONE part from the following	
a)	<p>For decomposition D1:</p> <p>$R1(PQST), R2(PTX), R3(QY), R4(YZW)$</p> <p>$R1 \cap R2 = (PT)^+ = PTYXZW$, it is a super key, so we can merge R1 and R2.</p> <p>combined table T1 is PQSTX</p> <p>similarly,</p> <p>$R3 \cap R4 = (Y)^+ = YZW$, it is a super key, so we can merge R3 and R4.</p> <p>So, another combined table T2 is QYZW.</p> <p>Now, Q is common in both T1 and T2.</p> <p>$T1 \cap T2 = Q^+ = QYZW$, it is a super key, so we can merge T1 and T2.</p> <p>after combining, we get original table PQSTXYZW,</p> <p>Hence D1 is lossless join decomposition.....</p> <p>For decomposition D2:</p> <p>$R1(PQS), R2(TX), R3(QY), R4(YZW)$</p> <p>since R2 has no common attributes as the primary key, so R2 cannot be merge with any other table,</p> <p>Hence D2 is lossy decomposition.....</p>	<p>2.5</p> <p>5</p>

b)

Ans- 2(b)

Given $R(A, B, C, D, E)$ $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$ $R_1(A, B, C)$ & $R_2(C, D, E)$ Let F_1 & F_2 are FD's for relations R_1 & R_2 respectively.

So, for Dependency preservation

$$(F_1 \cup F_2)^+ = F^+$$

Find FD's for $R_1(A, B, C)$ i.e. F_1

$$A^+ = \{A, B, C, D\} \Rightarrow A \rightarrow BC$$

$$B^+ = \{B, C, D, A\} \Rightarrow B \rightarrow AC$$

$$C^+ = \{C, D, A, B\} \Rightarrow C \rightarrow AB$$

$$AB^+ = \{A, B, C, D\} \Rightarrow AB \rightarrow C$$

$$AC^+ = \{A, B, C, D\} \Rightarrow AC \rightarrow B$$

$$BC^+ = \{A, B, C, D\} \Rightarrow BC \rightarrow A$$

$$\text{So } F_1 = \{A \rightarrow BC, B \rightarrow AC, C \rightarrow AB, AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$$

FD's $\{A \rightarrow B, B \rightarrow C\}$ of F are already part of F_1 .

$$\text{So we can write } F_1 = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, C \rightarrow A, C \rightarrow B, AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$$

Find FD's for $R_2(C, D, E)$ i.e. F_2

$$C^+ = \{C, D, A, B\} \Rightarrow C \rightarrow D, \text{ this is already part of } F \text{ (ignore)}$$

$$D^+ = \{D, A, B, C\} \Rightarrow D \rightarrow C$$

$$E^+ = \{E\} \text{ trivial so ignore}$$

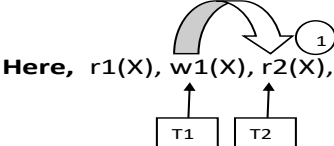
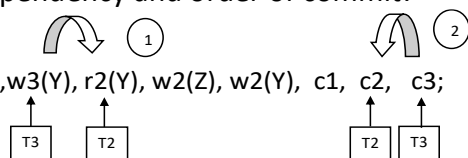
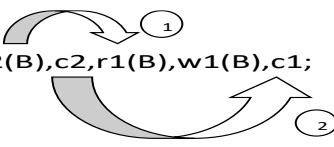
$$CD^+ = \{C, D, A, B\} \Rightarrow CD \rightarrow CD, \text{ trivial so ignore.}$$

$$CE^+ = \{C, D, A, B, E\} \Rightarrow CE \rightarrow D$$

$$DE^+ = \{D, A, B, C, E\} \Rightarrow DE \rightarrow C$$

$$\text{So } F_2 = \{C \rightarrow D, D \rightarrow C, CE \rightarrow D, DE \rightarrow C\}$$

Now take $F_1 \cup F_2$ which includes all FD's of F . hence
 Dependency Preserving. $\because D \rightarrow C, C \rightarrow A \Rightarrow D \rightarrow A$

3	Attempt ANY ONE part from the following	
	<p>As we already know that:</p> <p>a)</p> <ul style="list-style-type: none"> • $ri(X)$ = the transaction T_i reads the DB element X • $w_i(X)$ = the transaction T_i writes the DB element X <p>A schedule is recoverable if there is no dirty read at all. But if dirty read exists and commits operations appear in order of dirty read, then schedules are recoverable. Otherwise, schedules are irrecoverable.</p> <p>Considering S1, for a schedule to be T1 must commit before commit of T_2. Plotting of dependency and order of commit:</p> <div style="text-align: center;">  <p>Here, $r_1(X), w_1(X), r_2(X), r_1(Y), r_2(Y), w_2(X), w_1(Y), c_1, c_2$;</p> </div> <ul style="list-style-type: none"> • At this point 1: dirty read is performed by T_2. Now dependency is created, T_2 is dependent on T_1. Now we will check whether the order of commit is in order of dependency. • At point 2: Yes, the order of commit is the same as the order of dependency. Now the issue related to dirty read is resolved as the commit operation of T_1 appears before the commit operations of T_2; hence this schedule becomes recoverable. <p>Considering S2 schedule now, Plotting of dependency and order of commit:</p> <div style="text-align: center;">  <p>Here, $r_1(X), r_2(X), r_1(Z), r_3(X), r_3(Y), w_1(X), w_3(Y), r_2(Y), w_2(Z), w_2(Y), c_1, c_2, c_3$;</p> </div> <p>At point 1: Here, Dirty read exists, i.e., T_2 is dependent on T_3. This schedule will be recoverable only if the commit order is also same as dependency.</p> <p>At point 2: But commit operation of T_3 appears after commit operation of T_2. Therefore, the order of commit is different from the order of dependency. Hence schedule is not recoverable.</p>	<p>2.5</p> <p>5</p>
	<p>b)</p> <p>A schedule is cascadeless if there is no dirty read or dependent transaction reads the committed value only.</p> <p>Consider S1: $r_1(A), w_2(A), r_1(B), c_1, w_3(B), r_3(B), w_3(A), c_3, r_2(C), c_2$; as there is no dirty read hence schedule is cascadeless.</p> <div style="text-align: center;">  <p>Consider S2: $r_1(A), w_2(B), c_2, r_1(B), w_1(B), c_1$;</p> </div> <ul style="list-style-type: none"> • At point 1, T_1 is reading the value of data item B, which is written by T_2, i.e., T_1 is dependent on T_2. Then we need to analyze whether this read is a dirty read or not? • At point 2, it is clearly visible that T_2 commits immediately after $w_2(B)$, and after that, T_1 reads the committed value of T_2; hence this read is not dirty read. Therefore, the schedule is cascadeless. 	<p>2.5</p> <p>5</p>
4	Attempt ANY ONE part from the following	

a)	<p>Lock Conversions</p> <p>Two-phase locking with lock conversions:</p> <ul style="list-style-type: none">First Phase:<ul style="list-style-type: none">can acquire a lock-S on itemcan acquire a lock-X on itemcan convert a lock-S to a lock-X (upgrade)Second Phase:<ul style="list-style-type: none">can release a lock-Scan release a lock-Xcan convert a lock-X to a lock-S (downgrade) <p>This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.</p> <table><tr><th>T_8</th><th>T_9</th></tr><tr><td>lock-S(a_1)</td><td>lock-S(a_1)</td></tr><tr><td>lock-S(a_2)</td><td>lock-S(a_2)</td></tr><tr><td>lock-S(a_3)</td><td></td></tr><tr><td>lock-S(a_4)</td><td></td></tr><tr><td></td><td>unlock(a_1)</td></tr><tr><td></td><td>unlock(a_2)</td></tr><tr><td>lock-S(a_n)</td><td></td></tr><tr><td>upgrade(a_1)</td><td></td></tr></table> <p>Incomplete schedule with a lock conversion.</p>	T_8	T_9	lock-S(a_1)	lock-S(a_1)	lock-S(a_2)	lock-S(a_2)	lock-S(a_3)		lock-S(a_4)			unlock(a_1)		unlock(a_2)	lock-S(a_n)		upgrade(a_1)		5
T_8	T_9																			
lock-S(a_1)	lock-S(a_1)																			
lock-S(a_2)	lock-S(a_2)																			
lock-S(a_3)																				
lock-S(a_4)																				
	unlock(a_1)																			
	unlock(a_2)																			
lock-S(a_n)																				
upgrade(a_1)																				
b)	<p>Multiversion schemes keep old versions of data item to increase concurrency.</p> <ul style="list-style-type: none">Multiversion Timestamp OrderingMultiversion Two-Phase Locking <ul style="list-style-type: none">✓ Each successful write results in the creation of a new version of the data item written.✓ Use timestamps to label versions.✓ When a read(Q) operation is issued, select an appropriate version of Q based on the timestamp of the transaction, and return the value of the selected version.✓ Reads never have to wait as an appropriate version is returned immediately. <p>Multiversion Timestamp Ordering</p> <ul style="list-style-type: none">✓ Each data item Q has a sequence of versions $\langle Q_1, Q_2, \dots, Q_m \rangle$. Each version Q_k contains three data fields:<ul style="list-style-type: none">• Content -- the value of version Q_k.• W-timestamp(Q_k) -- timestamp of the transaction that created (wrote) version Q_k• R-timestamp(Q_k) -- largest timestamp of a transaction that successfully read version Q_k <ul style="list-style-type: none">✓ When a transaction T_i creates a new version Q_k of Q, Q_k's W-timestamp and R-timestamp are initialized to $TS(T_i)$.✓ R-timestamp of Q_k is updated whenever a transaction T_j reads Q_k, and $TS(T_j) > R\text{-timestamp}(Q_k)$. <ul style="list-style-type: none">✓ Suppose that transaction T_i issues a read(Q) or write(Q) operation. Let Q_k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to $TS(T_i)$.<ol style="list-style-type: none">If transaction T_i issues a read(Q), then the value returned is the content of version Q_k.If transaction T_i issues a write(Q)<ol style="list-style-type: none">if $TS(T_i) < R\text{-timestamp}(Q_k)$, then transaction T_i is rolled back.if $TS(T_i) = W\text{-timestamp}(Q_k)$, the contents of Q_k are overwrittenelse a new version of Q is created.	5																		
Section-C																				
Total Marks : 5*10 = 50																				
5	Attempt ANY ONE part from the following																			

a)

Ans 5-a(i)

given $R(W, X, Y, Z)$

$F = \{WY \rightarrow XZ, X \rightarrow Y\}$

Find candidate key

W is essential Attribute. find W^+

$W^+ = \{W\}$ so its not super key. but will be part of CK.

Now $(W, X, Y, Z)^+ = \{W, X, Y, Z\} \rightarrow \textcircled{1}$

as $WY \rightarrow XZ$

$\Rightarrow (WY)^+ = \{W, X, Y, Z\} \Rightarrow (WY)$ is Candidate Key

Now using $X \rightarrow Y$ on $\textcircled{1}$

$(WXZ)^+ = \{W, X, Y, Z\}$ Now checking subsets for SK.

$WX^+ = \{X, Y, W, Z\} = R$ hen (WX) is also CK.

$WZ^+ = \{W, Z\} \neq R$

hence Prime Attributes are $\{W, X, Y\}$

✓ Non Prime Attributes are $\{Z\}$

Check for BCNF

$WY \rightarrow XZ$, WY is superkey so BCNF condⁿ met.

$X \rightarrow Y$, X is not SK. so BCNF condⁿ not met.

Check for 3NF

$X \rightarrow Y$, Y is prime attribute, 3NF condⁿ met

$WY \rightarrow XZ$, WY is super key, 3NF condⁿ met.

Hence given Relation is in 3NF.

Ans 5 a (ii)

given $R(A, B, C, D, E)$

$$F = \{A \rightarrow CD, C \rightarrow B, B \rightarrow AE\}$$

As there is no Essential Attribute

$$\text{So } (A, B, C, D, E)^+ = \{A, B, C, D, E\} \quad \text{--- (1)}$$

$$\text{as } A \rightarrow CD$$

$$\Rightarrow (ABE)^+ = \{A, B, C, D, E\}$$

$$B \rightarrow AE \Rightarrow (B)^+ = \{A, B, C, D, E\} \Rightarrow \underline{B \text{ is CK.}}$$

Now ~~starting~~ starting with $B \rightarrow AE$ on (1)

$$\Rightarrow (BCD)^+ = \{A, B, C, D, E\}$$

$$\text{as } C \rightarrow B \Rightarrow (CD)^+ = \{A, B, C, D, E\} \text{ so } \underline{CD \text{ is SK.}}$$

Now checking for subsets

$$C^+ = \{C, B, A, E, D\} \Rightarrow \underline{C \text{ is CK.}}$$

$$D^+ = \{D\}$$

$$\text{Using same } A^+ = \{A, C, D, B, E\} \Rightarrow \underline{A \text{ is also CK.}}$$

there are Three CK (A), (B), (C).

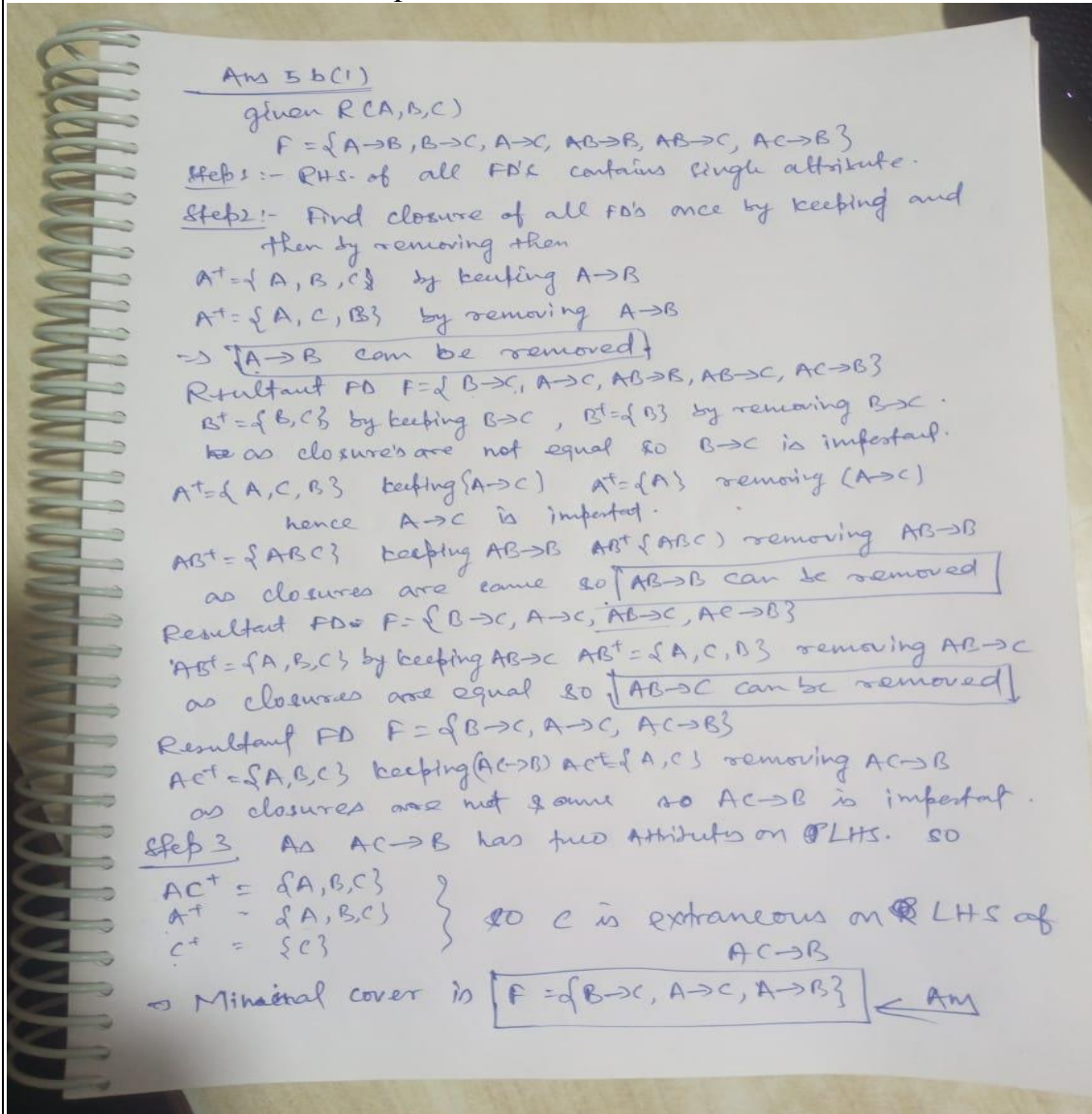
For 3NF Decomposition

As L.H.S. of all FD's in F are candidate keys so relation is already in BCNF.

So, Not required to decompose.

b) i) We can define a set of functional dependencies FD to be minimal (canonical cover) if it satisfies the following conditions:

1. Every dependency in FD has a single attribute for its right-hand side.
2. We cannot remove any dependency from FD and still have a set of dependencies equivalent to FD.
3. On the left-hand side (LHS) of any functional dependency (FDs remaining after condition two is met), we should not have any extraneous attribute (applicable when LHS has more than one attribute). An attribute of a functional dependency is extraneous if we can remove it without changing the closure of the set of functional dependencies.



5

ii) If we decompose a Relation $r(R)$ into $r_1(R_1)$ and $r_2(R_2)$ such that $R_1 \cup R_2 = R$ (attribute preservation condition), then it is said to be lossless if it satisfies $r_1 \bowtie r_2 = r$ with no new tuples added and no tuples eliminated.

r(R):			r ₁ (R ₁):		r ₂ (R ₂):		r ₁ (R ₁) ⋈ r ₂ (R ₂)		
A	B	C	A	B	A	C	A	B	C
a1	b1	c1	a1	b1	a1	c1	a1	b1	c1
a2	b2	c1	a2	b2	a2	c1	a1	b1	c2
a1	b1	c2	a3	b2	a1	c2	a1	b1	c3
a3	b2	c3			a3	c3	a2	b2	c1
a1	b1	c3			a1	c3	a2	b2	c4
a2	b2	c4			a2	c4	a3	b2	c3

we can see that $R_1 \cup R_2 = R$ and $r_1 \bowtie r_2 = r$. It is a lossless join decomposition.

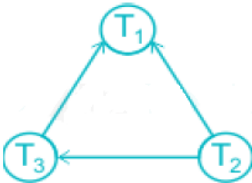
10

Schedule S1:

a)

T1	T2	T3
r(X)		
		r(Y)
		r(X)
	r(Y)	
	r(Z)	
		w(Y)
	w(Z)	
r(Z)		
w(X)		
w(Z)		

Precedence graph of S1:

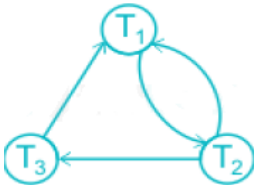


Since there is no loop in S1, it is is conflict serializable. Serial Order: T2 → T1 → T2.

Schedule S2:

T1	T2	T3
r(X)		
		r(Y)
	r(Y)	
		r(X)
r(Z)		
	r(Z)	
		w(Y)
w(X)		
	w(Z)	
w(Z)		

Precedence Graph for S2:

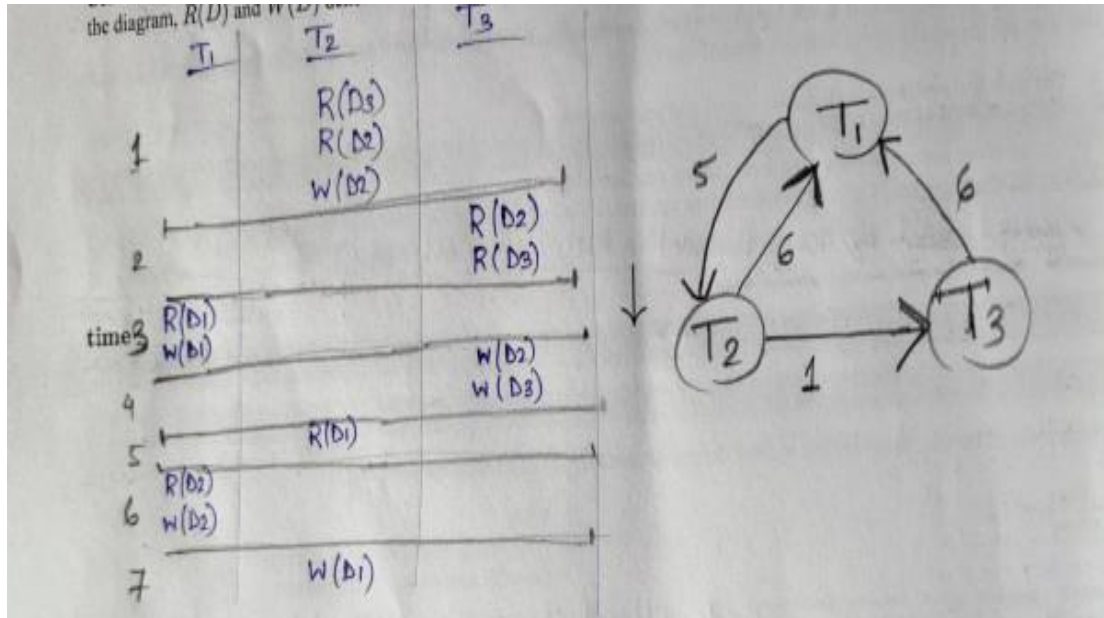


So there is loop in S2 so S2 is not conflict serializable.
Schedule S1 is equivalent to only this serial schedule T2 : T3 : T1.

5

10

b)



There is a cycle in precedence graph so schedule is **not conflict serializable**.

Check View Serializability:

Checking View Serializability by contradiction..

1. Initial Read

T_2 read D_2 value from initial database and T_1 modify D_2 so T_2 should execute before T_1 .

i.e., $T_2 \rightarrow T_1$

2. Final write.

final write of D_1 in given schedule done by T_2 and T_1 modify D_1 i.e. $W(D_1)$.. that means T_2 should execute after T_1 . i.e., $T_1 \rightarrow T_2$

So, schedule not even View Serializable.

5

10

- a) A **log** is kept on stable storage. The log is a sequence of **log records**, and maintains a record of update activities on the database.
- Different types of log records are as follows –
- $\langle T_i, X_i, V_1, V_2 \rangle$ – update log record, where T_i =transaction, X_i =data, V_1 =old data, V_2 =new value.
 - $\langle T_i, \text{start} \rangle$ – Transaction T_i starts execution.
 - $\langle T_i, \text{commit} \rangle$ – Transaction T_i is committed.
 - $\langle T_i, \text{abort} \rangle$ – Transaction T_i is aborted

The log records can be written as follows –

Create a log for the given transaction T1 and T2.		
T1	T2	Log
Read A	Read A	$\langle T_1, \text{start} \rangle$
$A = A - 2000$	$A = A + 5000$	$\langle T_1, A, 5000, 3000 \rangle$
Write A	Write A	$\langle T_1, B, 8000, 10000 \rangle$
Read B	Read B	$\langle T_1, \text{commit} \rangle$
$B = B + 2000$	$B = B + 7000$	$\langle T_2, \text{start} \rangle$
Write B	Write B	$\langle T_2, A, 3000, 8000 \rangle$
		$\langle T_2, B, 10000, 17000 \rangle$
		$\langle T_2, \text{commit} \rangle$

Log based recovery techniques

Log based recovery uses one of the techniques –

Deferred database modification

It modifies the database after completion of transaction. The database modification is deferred or delayed until the last operation of the transaction is executed. Update log records maintain the new value of the data item.

Recover system uses one operation which is as follows –

- Redo(T_i) – All data items updated by the transaction T_i are set to a new value.

Immediate database modification

It modifies the database after a write operation, database modification is immediately done when a transaction performs an update/ write operation. Update log records maintain both old and new values of data items.

The recovery system uses two operations, which are as follows –

- Undo(T_i) – All data items updated by the transaction T_i , are set to old value.
- Redo(T_i) – All data items updated by the transaction T_i are set to a new value.

Part B:

By looking on logs and options in this question, it seems to follow immediate database modification. It means till crash of system, few of log records has written in database. But system does not know after crashing that how many log records already written in database.

So after restart system, undo all uncommitted transactions like T2 (reverse log record 6 to set B back to 10000) and then redo all committed transactions like T1 (log records 2 and 3). Thus, it needs to undone active transactions (T2) and redo committed transactions (T1).

Process Steps:

1. Go to log record (vii) where system crashed and reads the logs backwards.
2. If find some committed transaction then puts to Redo List; If find some uncommitted transaction then executes its logs in reverse like T2 writes B to 10000.
3. Redo T1's log records like write B to 10000 and M to 2000 without caring already existing values of A & B in database.

b)

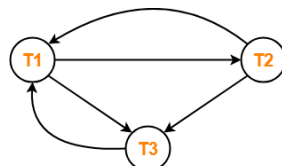
For simplicity and better understanding, we can represent the given schedule pictorially as-

T1	T2	T3
R (A)		
	W (A)	
W (A)		R (A)
		W (A)

We know, if a schedule is conflict serializable, then it is surely view serializable.

So, let us check whether the given schedule is conflict serializable or not.

Draw the precedence graph-



Clearly, there exists a cycle in the precedence graph. Therefore, the given schedule S is not conflict serializable.

Now, Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.

For view serializable the given schedule must be view equivalent to a serial schedule.

Consider this serial schedule.

T1	T2	T3
R(A)		
W(A)		
	W(A)	
		R(A)
		W(A)

The above concurrent schedule **follows all three conditions** for view equivalent with serial schedule i.e.

The schedules S and S are said to be **view equivalent** if three conditions are met:

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S' , also read the initial value of Q.
2. For each data item Q, if transaction T_i executes $read(Q)$ in schedule S, and if that value was produced by a $write(Q)$ operation executed by transaction T_j , then the $read(Q)$ operation of transaction T_i must, in schedule S' , also read the value of Q that was produced by the same $write(Q)$ operation of transaction T_j .
3. For each data item Q, the transaction (if any) that performs the final $write(Q)$ operation in schedule S must perform the final $write(Q)$ operation in schedule S' .

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule.

Blind writes: Observe that, in schedule 8, transactions T_4 and T_6 perform $write(Q)$ operations without having performed a $read(Q)$ operation. Writes of this sort are called blind writes. **Blind writes appear in any view-serializable schedule that is not conflict serializable.**

T ₃	T ₄	T ₆
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

Schedule 8

9 Attempt ANY ONE part from the following

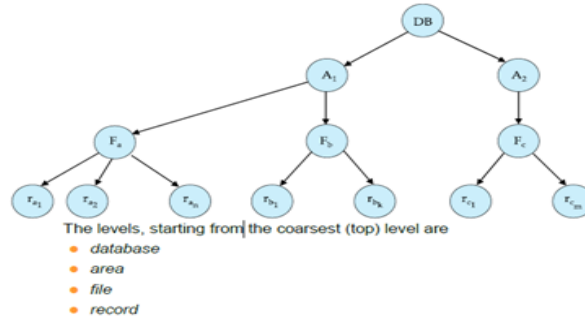
- a) **Multiple granularity concurrency control scheme:**
Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones that can be represented graphically as a tree. When a transaction locks a node in the tree *explicitly*, it *implicitly* locks all the node's descendents in the same mode.

Granularity of locking (level in tree where locking is done):

Fine granularity (lower in tree): high concurrency, high locking overhead

Coarse granularity (higher in tree): low locking overhead, low concurrency

Example of Granularity Hierarchy



Intention Lock Modes

In addition to S and X lock modes, there are three additional lock modes with multiple granularity:

intention-shared (IS): Indicates explicit locking at a lower level of the tree but only with shared locks.

intention-exclusive (IX): Indicates explicit locking at a lower level with exclusive or shared locks

shared and intention-exclusive (SIX): The subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.

Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

Compatibility Matrix with Intention Lock Modes

■ The compatibility matrix for all lock modes is:

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

Multiple Granularity Locking Scheme

- Transaction T_i can lock a node Q , using the following rules: The lock compatibility matrix must be observed.
- The root of the tree must be locked first, and may be locked in any mode.
- A node Q can be locked by T_i in S or IS mode only if the parent of Q is currently locked by T_i in either IX or IS mode.
- A node Q can be locked by T_i in X, SIX, or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.
- T_i can lock a node only if it has not previously unlocked any node (that is, T_i is two-phase).
- T_i can unlock a node Q only if none of the children of Q are currently locked by T_i .

The Two-Phase Locking Protocol

b) This is a protocol which ensures conflict-serializable schedules.

- Phase 1: **Growing Phase** transaction may obtain locks transaction may not release locks
- Phase 2: **Shrinking Phase** transaction may release locks transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e. the point where a transaction acquired its final lock).

There can be conflict serializable schedules that cannot be obtained if two-phase locking is used.

- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
- Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable

T_5	T_6	T_7
lock-X(A) read(A) lock-S(B) read(B) write(A) unlock(A)	lock-X(A) read(A) write(A) unlock(A)	lock-S(A) read(A)

Figure Partial schedule under two-phase locking.

6

Strict two-phase locking protocol.

- **Cascading rollbacks** can be avoided by a modification of two-phase locking called the strict two-phase locking protocol.
- This protocol requires not only that locking be two phase, but also that all **exclusive-mode** locks taken by a transaction be held until that **transaction commits**.
- This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

10