UNIT III

TRANSACTIONS

Transaction Concepts – ACID Properties – Schedule and immediate update – Shadow paging – ARIES Algorithm– Serializability – Transaction support in SQL – Need for Concurrency – Concurrency control –Two Phase Locking- Timestamp – Multiversion –Multiple Granularity locking – Deadlock Handling – Recovery Concepts – Recovery based on deferred.

Transaction Concepts

Definition: A transaction can be defined as a group of tasks that form a single logical unit.

Part I: Introduction to Transactions

Transaction Concepts

Definition: A transaction can be defined as a group of tasks that form a single logical unit.

For example - Suppose we want to withdraw 100 from an account then we will follow following operations:

1) Check account balance
 If sufficient balance is present request for withdrawal.

2) Get the money

3) Calculate Balance = Balance -100

4) Update account with new balance.

The above mentioned four steps denote one transaction.

In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

Possible Question

1. Write a short note on - Transaction Concept.

ACID Properties

This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.

ACID Properties

1) Atomicity:

This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.

• No transaction in the database is left half completed.

• Database should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.

• For example In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

2) Consistency:

• The database must remain in consistent state after performing any transaction.

• For example: In ATM withdrawal operation, the balance must be updated appropriatelyafter performing transaction. Thus the database can be in consistent state.

3) Isolation:

In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.

• No transaction will affect the existence of any other transaction.

• For example: If a bank manager is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

4) Durability:

• The database should be strong enough to handle any system failure.

• If there is any set of insert /update, then it should be able to handle and commit to the database.

• If there is any failure, the database should be able to recover it to the consistent state.

• For example: In ATM withdrawal example, if the system failure happens after Customer getting the money then the system should be strong enough to update Database with his new balance, after system recovers. For that purpose the system has to keep the log of each transaction and its failure. So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to Database.

Possible Questions

***1.** Explain with an example the properties that must be satisfied by transaction.*

***2.** Explain the ACID properties of transaction*

***3.** Discuss in detail about the ACID properties of transaction.*

***4.** Discuss the properties of a transaction that ensure integrity of data in the database system.*

Transaction States

This is the first state of transaction. For example: insertion, deletion or updation of record is done here. But data is not saved to database.

Transaction States

Each transaction has following five states:

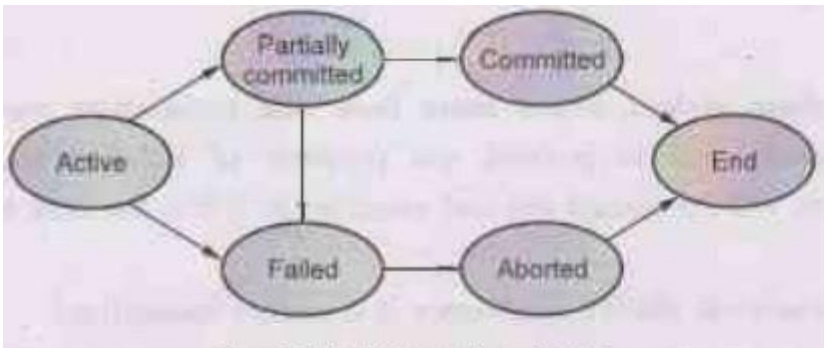


Fig. 3.3.1 : Transaction states

1) Active: This is the first state of transaction. For example: insertion, deletion or updation of record is done here. But data is not saved to database.

2) Partially Committed: When a transaction executes its final operation, it is said to be in a partially committed state.

3) Failed: A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

4) Aborted: If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction.

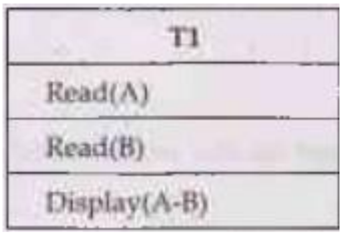
5) Committed: If a transaction executes all its operations successfully, it is said to be committed. This is the last step of a transaction, if it executes without fail.

Example 3.3.1 *Define a transaction. Then discuss the following with relevant examples:*

1. A read only transaction 2. A read write transaction 3. An aborted transaction

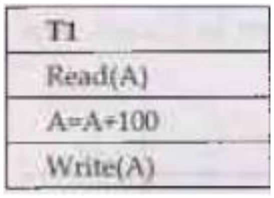
Solution:

(1) Read only transaction



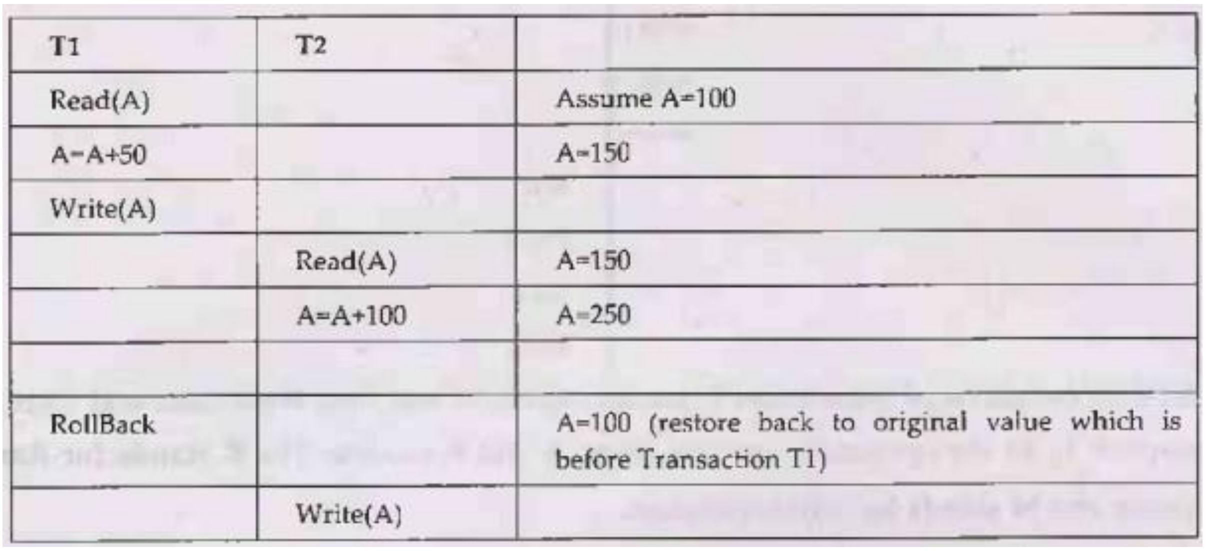
T1
Read(A)
Read(B)
Display(A-B)

(2) A read write transaction



T1
Read(A)
A=A+100
Write(A)

(3)



T1	T2	
Read(A)		Assume A=100
A=A+50		A=150
Write(A)		
	Read(A)	A=150
	A=A+100	A=250
RollBack		A=100 (restore back to original value which is before Transaction T1)
	Write(A)	

Possible Questions

***1.** During execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which transaction may pass. Explain why each state transaction may occur?*

***2.** With a neat sketch explain the states of transaction.*

***3.** Brief the states of a transaction with a neat diagram.*

Schedules

Schedule is an order of multiple transactions executing in concurrent environment.

Schedules

Schedule is an order of multiple transactions executing in concurrent environment. Following figure represents the types of schedules.

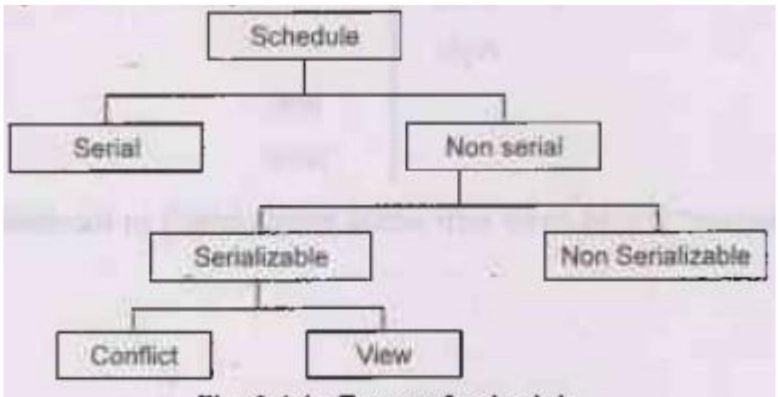
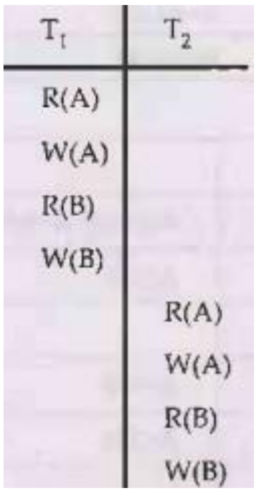


Fig. 3.4.1 : Types of schedule

Serial schedule: The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions T_1 and T_2

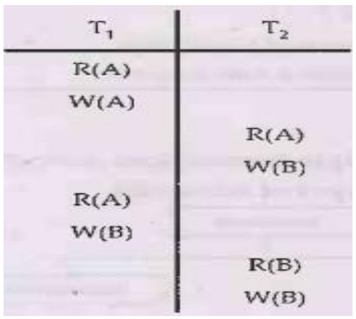


T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T_1 on data items A and then B executes and then intransaction T_2 all the operations on data items A and B execute. The R stands for Read operation and W stands for write operation.

Non serial schedule: The schedule in which operations present within the transaction are intermixed. This may lead to conflicts in the result or inconsistency in the resultant data. For example-

Consider following two transactions,



T_1	T_2
R(A)	
W(A)	
	R(A)
	W(B)
R(A)	
W(B)	
	R(B)
	W(B)

The above transaction is said to be non serial which result in inconsistency or conflicts in the data.

Serializability

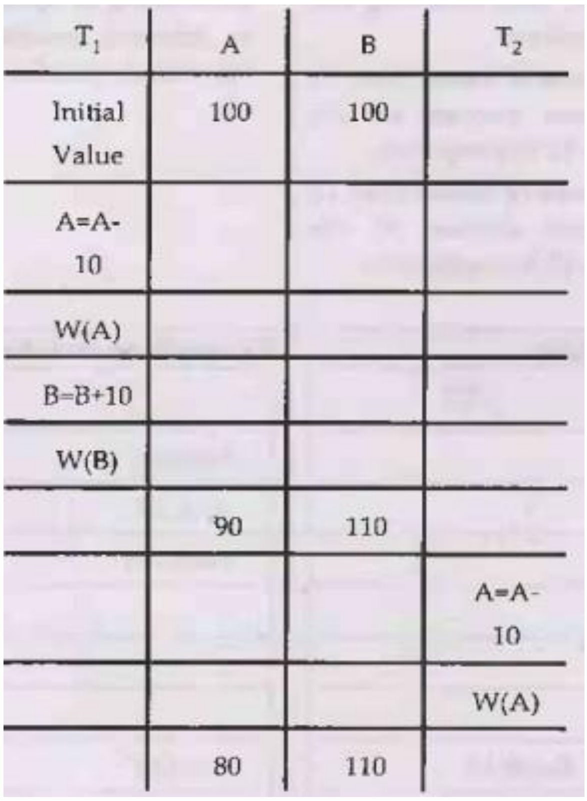
When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transaction).

Serializability

• When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transaction).

• Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.

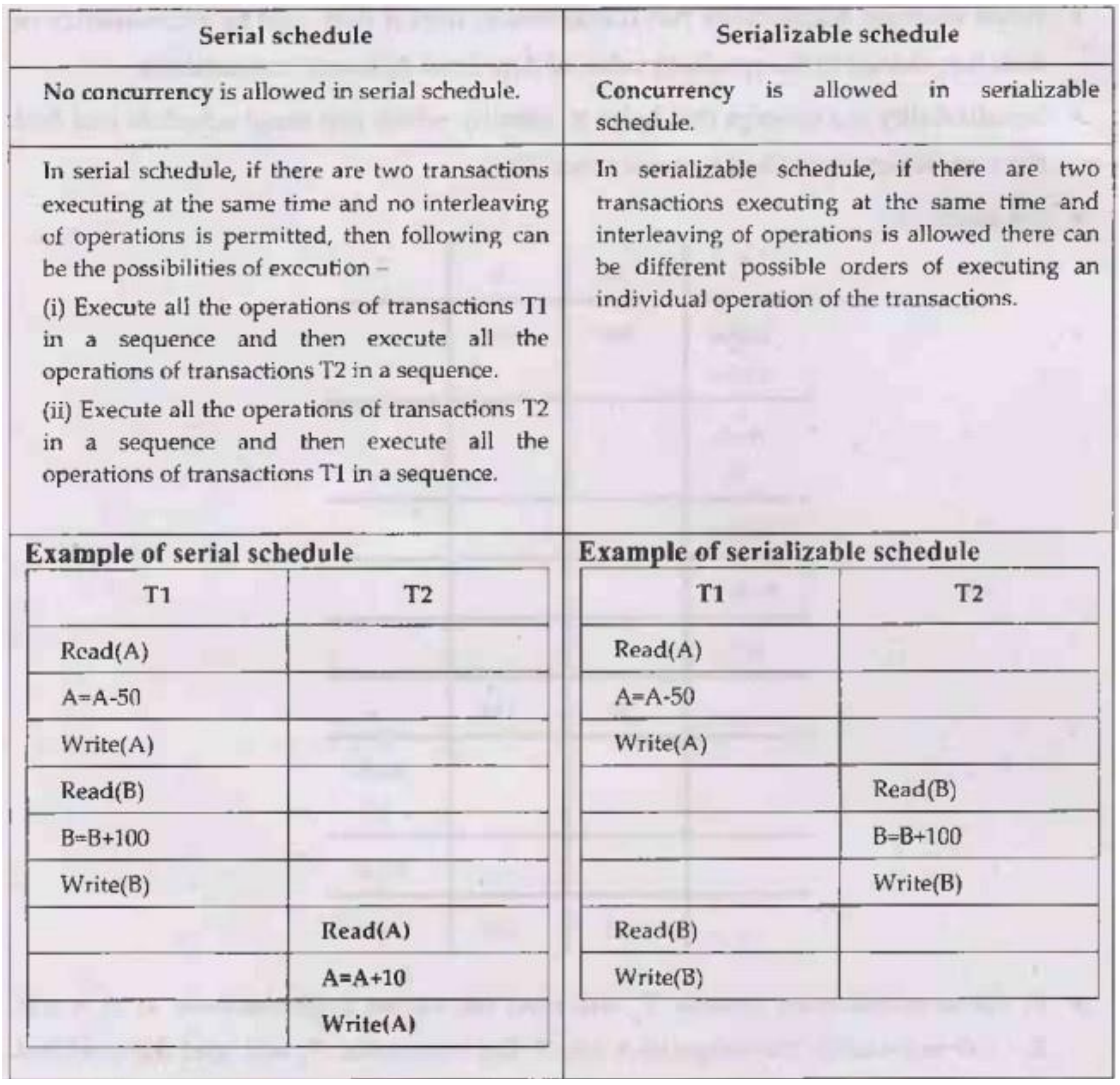
• For example:



T_1	A	B	T_2
Initial Value	100	100	
A=A-10			
W(A)			
B=B+10			
W(B)			
	90	110	
			A=A-10
			W(A)
	80	110	

• In above transaction initially T_1 will read the values from database as A= 100, B= 100 and modify the values of A and B, transaction T_2 will read the modified value i.e. 90 and will modify it to 80 and perform write operation. Thus at the end of transaction T_1 value of A will be 90 but at end of transaction T_2 value of A will be 80. Thus conflicts or inconsistency occurs here. This sequence can be converted to a sequence which may give us consistent result. This process is called serializability.

Difference between Serial schedule and Serializable schedule



Serial schedule	Serializable schedule
No concurrency is allowed in serial schedule.	Concurrency is allowed in serializable schedule.
In serial schedule, if there are two transactions executing at the same time and no interleaving of operations is permitted, then following can be the possibilities of execution – (i) Execute all the operations of transactions T1 in a sequence and then execute all the operations of transactions T2 in a sequence. (ii) Execute all the operations of transactions T2 in a sequence and then execute all the operations of transactions T1 in a sequence.	In serializable schedule, if there are two transactions executing at the same time and interleaving of operations is allowed there can be different possible orders of executing an individual operation of the transactions.

Example of serial schedule

T1	T2
Read(A)	
A=A-50	
Write(A)	
Read(B)	
B=B+100	
Write(B)	
	Read(A)
	A=A+10
	Write(A)

Example of serializable schedule

T1	T2
Read(A)	
A=A-50	
Write(A)	
	Read(B)
	B=B+100
	Write(B)
Read(B)	
Write(B)	

• There are two types of serializabilities: conflict serializability and view serializability

Conflict Serializability

• **Definition:** Suppose T_1 and T_2 are two transactions and I_1 and I_2 are the instructions in T_1 and T_2 respectively. Then these two transactions are said to be conflict Serializable, if both the instruction access the data item d, and at least one of the instruction is write operation.

• **What is conflict?:** In the definition three conditions are specified for a conflict in conflict serializability -

1) There should be different transactions

2) The operations must be performed on same data items

3) One of the operation must be the Write(W) operation

• We can test a given schedule for conflict serializability by constructing a precedence graph for the schedule, and by searching for absence of cycles in the graph.

• Predence graph is a directed graph, consisting of G=(V,E) where V is set of vertices and E is set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges Ti→Tj for which one of three conditions holds :

1. Ti executes write(Q) before Tj executes read(Q).

2. Ti executes read(Q) before Tj executes write(Q).

3. Ti executes write(Q) before Tj executes write(Q).

• A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

Testing for serializability

Following method is used for testing the serializability: To test the conflictserializability we can draw a graph G = (V,E) where V = vertices which represent the number of transactions. E = edges for conflicting pairs.

Step 1: Create a node for each transaction.

Step 2: Find the conflicting pairs(RW, WR, WW) on the same variable(or data item) by different transactions.

Step 3: Draw edge for the given schedule. Consider following cases

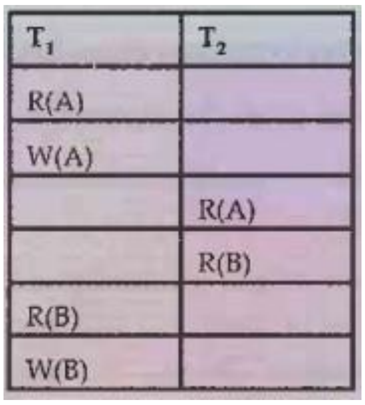
1. Ti executes write(Q) before Tj executes read(Q), then draw edge from T_i to T_j.

2. Ti executes read(Q) before Tj executes write(Q), then draw edge from T_i to T_j

3. Ti executes write(Q) before Tj executes write(Q),, then draw edge from T_i to T_j

Step 4: Now, if precedence graph is cyclic then it is a non conflict serializable schedule and if the precedence graph is acyclic then it is conflict serializable schedule.

Example 3.5.1 *Consider the following two transactions and schedule (time goes from top to bottom). Is this schedule conflict-serializable? Explain why or why not.*



T_1	T_2
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	

Solution :

Step 1: To check whether the schedule is conflict serializable or not we will check from top to bottom. Thus we will start reading from top to bottom as

T_1: R(A) -> T_1:W(A) ->T_2:R(A) -> T_2:R(B) ->T_1:R(B)->T_1:W(B)

Step 2: We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them-

i) Both the operations belong to different transactions.

ii) Both the operations are on same data item.

iii) At least one of the two operations is a write operation

From above given example in the top to bottom scanning we find the conflict as T_1:W(A)->T_2:R(A).

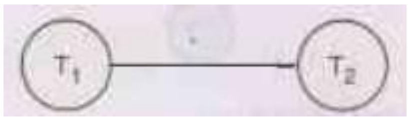
i) Here note that there are two different transactions T_1 and T_2,

ii) Both work on same data item i.e. A and

iii) One of the operation is write operation.

Step 3: We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions, there will be two nodes namely T_1 and T_2

Step 4: Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1:W(A) to T_2:R(A). Hence edge must be from T_1 to T_2.



Step 5: Repeat the step 4 while reading from top to bottom. Finally the precedence graph will be as follows

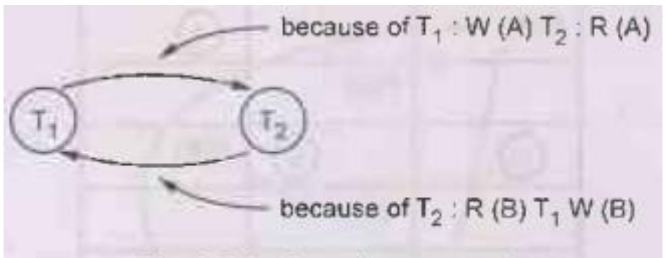
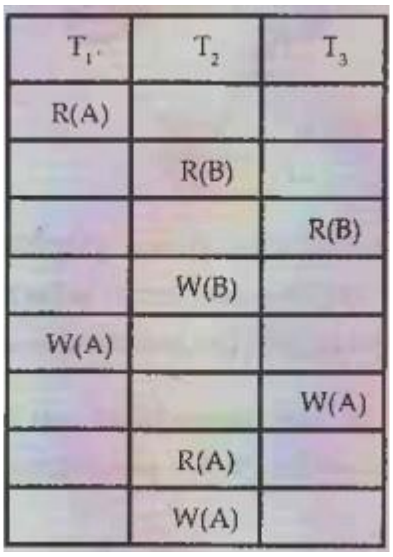

because of T1 : W (A) T2 : R (A)
T1
T2
because of T2 : R (B) T1 W (B)

Fig. 3.5.1 : Precedence graph

Step 6: Check if any cycle exists in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the is cycle found then schedule is not conflict serializable. In the step 5 we get a graph with cycle, that means given schedule is not conflict serializable.

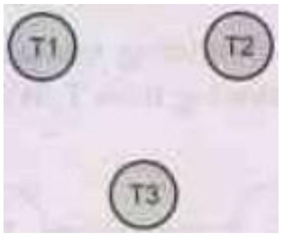
Example 3.5.2 *Check whether following schedule is conflict serializable or not. If it is not conflict serializable then find the serializability order.*



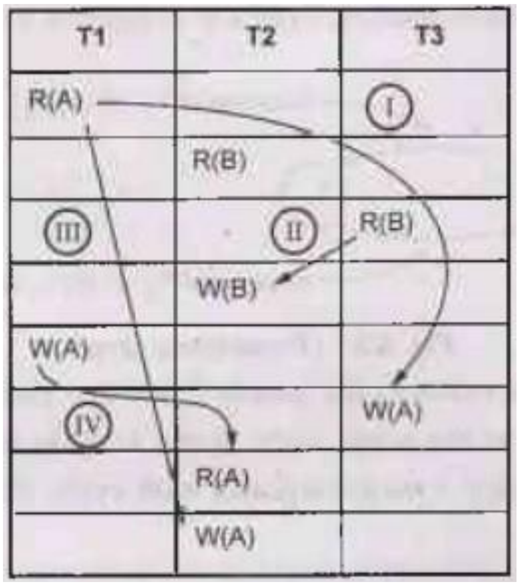
T_1	T_2	T_3
R(A)		
	R(B)	
		R(B)
	W(B)	
W(A)		
		W(A)
	R(A)	
	W(A)	

Solution:

Step 1: We will read from top to bottom, and build a precedence graph for conflicting entries. We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are three transactions, there will be two nodes namely T1 T2, and T3

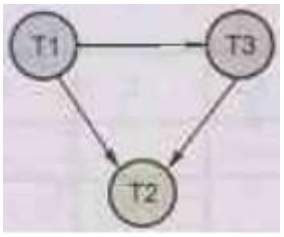


Step 2: The conflicts are found as follows –



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

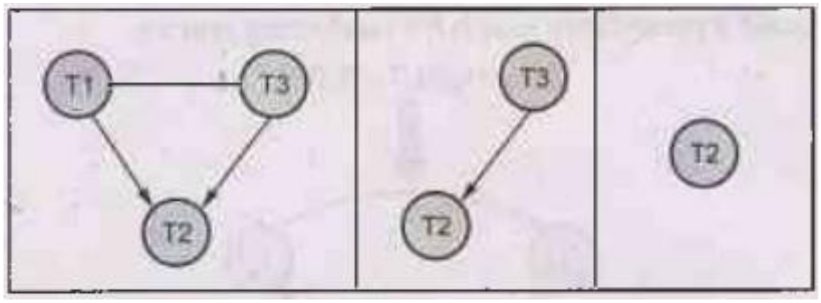
Step 3: The precedence graph will be as follows –



Step 4: As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For thatpurpose we will follow these steps to find the serializable order.

Step 5: A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

Step 6: Find the vertex which has no incoming edge which is T1. If we delete T1 node then T3 is a node that has no incoming edge. If we delete T3, then T2 is a node that has no incoming edge.

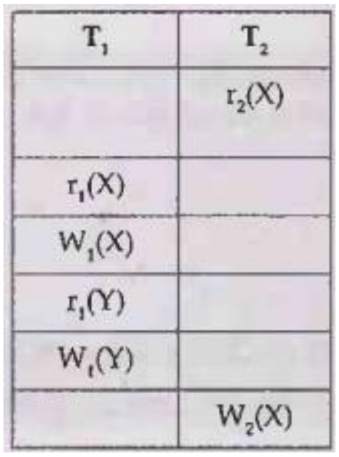


Thus the nodes can be deleted in a order T1, T3 and T2. Hence the order will be T1-T3-T2

Example 3.5.3 *Check whether the below schedule is conflict serializable or not.* {B2,r2(X),b1,r1(X),W1(X),r1(Y),W1(Y),W2(X),e1,C1,e2,C2}

Solution: b2 and b1 represents begin transaction 2 and begin transaction 1. Similarly, el ande2 represents end transaction 1 and end transaction 2.

We will rewrite the schedule as follows-



T_1	T_2
	$r_2(X)$
$r_1(X)$	
$W_1(X)$	
$r_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

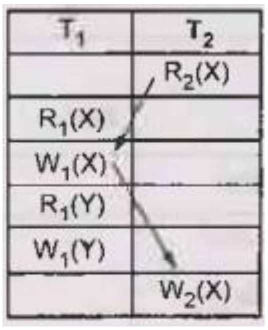
Step 1: We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the operations belong to different transactions.

ii) Both the operations are on same data item.

iii) At least one of the two operations is a write operation.

The conflicting entries are as follows –



T_1	T_2
	$R_2(X)$
$R_1(X)$	
$W_1(X)$	
$R_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

Step 2: Now we build a precedence graph for conflicting entries.

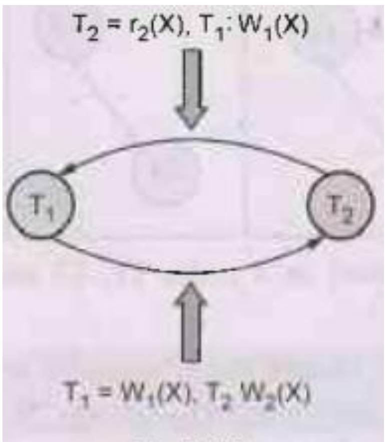


Fig. 3.5.2

As there are two transactions only two nodes are present in the graph.

Step 3: We get a graph with cycle, that means given schedule is not conflict serializable.

Example 3.5.4 *Consider the three transactions T1, T2, and T3 and schedules S1 and S2 given below. Determine whether each schedule is serializable or not? If a schedule isserializable write down the equivalent serial schedule(S).*

T1: R1(x) R1(z);W1(x);

T2: R2(x);R2(y);W2(z);W2(y)

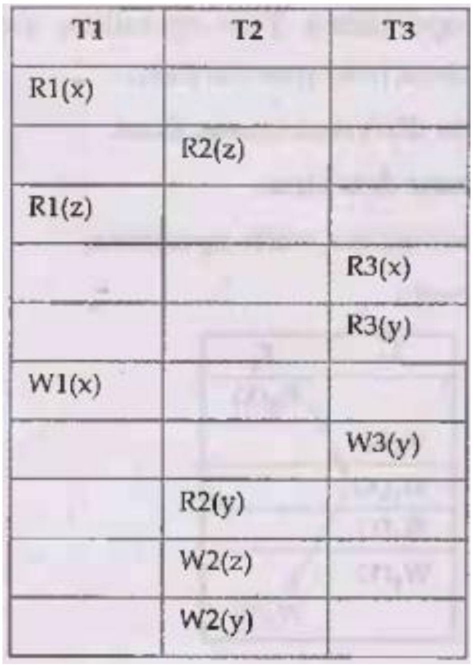
T3:R3(x);R3(y);W3(y);

S1: R1(x);R2(z);R1(z); R3(x);R3(y);W1(x);W3(y);R2(y); W2(z);W2(y);

S2: R1(x);R2(z);R3(x);R1(z);R2(y);R3(y);W1(x);W2(z);W3(y);W2(y);

Solution:

Step 1: We will represent the schedule S1 as follows



T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

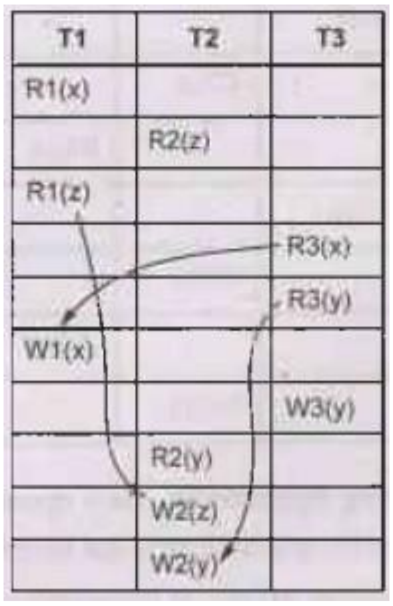
Step (a): We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the operations belong to different transactions.

ii) Both the operations are on same data item.

iii) At least one of the two operations is a write operation

The conflicting entries are as follows -



T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (b): Now we will draw precedence graph as follows-

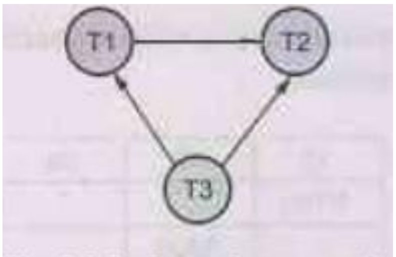


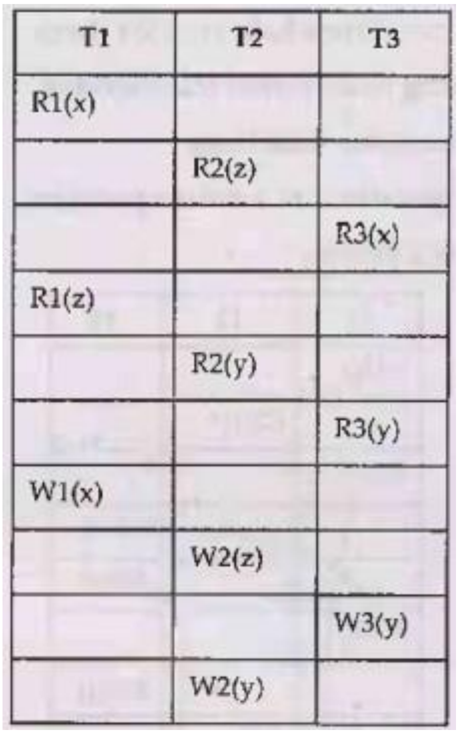
Fig. 3.5.3 : Precedence graph

As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c): A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

Step (d): Find the vertex which has no incoming edge which is T3. If we delete T3, then T1 is the edge that has no incoming edge. Finally find the vertex having no outgoing edge which is T2. Hence the order will be T3-T1-T2.

Step 2: We will represent the schedule S2 as follows -



T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

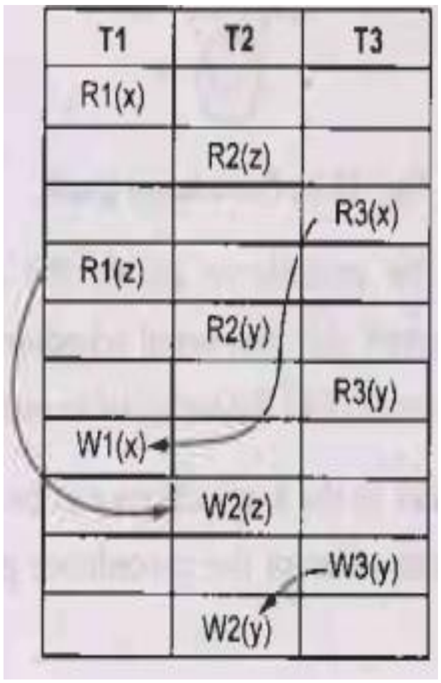
Step (a): We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the operations belong to different transactions.

ii) Both the operations are on same data item.

iii) At least one of the two operations is a write operation

The conflicting entries are as follows -



T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

Step (b): Now we will draw precedence graph as follows-

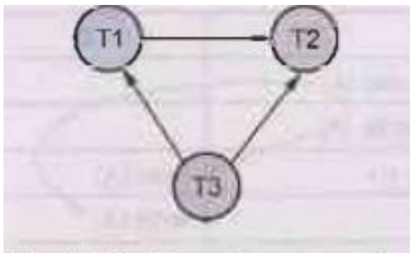


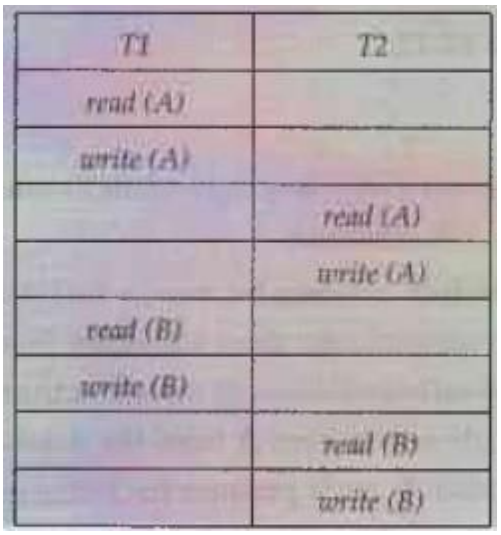
Fig. 3.5.4 : Precedence graph

As there is no cycle in the precedence graph, the given sequence is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c): A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called topological sorting.

Step (d): Find the vertex which has no incoming edge which is T3. Finally find the vertex having no outgoing edge which is T2. So in between them is T1. Hence the order will be T3-T1-T2

Example 3.5.5 *Explain the concept of conflict serializability. Decide whether following schedule is conflict serializable or not. Justify your answer.*

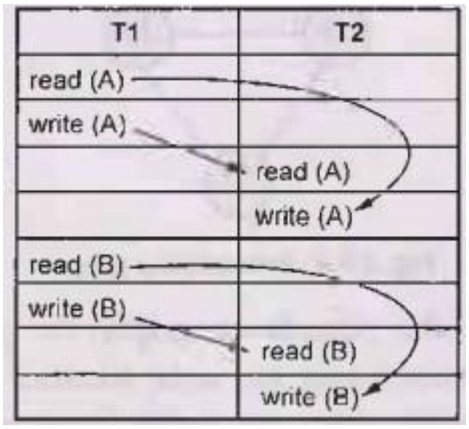


T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Solution:

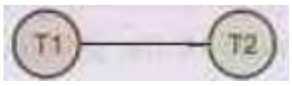
Step 1: We will read from top to bottom, and build a precedence graph for conflicting entries.

The conflicting entries are as follows-



T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Step 2: Now we will build precedence graph as follows



Step 3: There is no cycle in the precedence graph. That means this schedule is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow the following steps to find the serializable order.

1) Find the vertex which has no incoming edge which is T1.

2) Then find the vertex having no outgoing edge which is T2. In between them there is no other transaction.

3) Hence the order will be T1-T2.

View Serializability

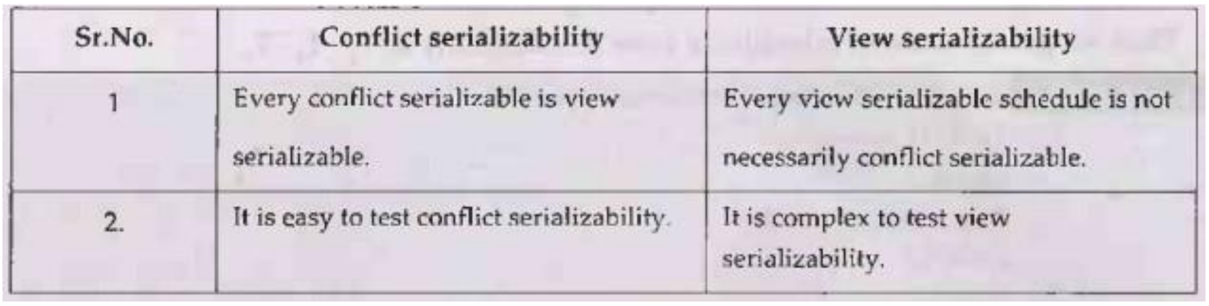
• If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

• **View Equivalent Schedule:** Consider two schedules S_1 and S_2 consisting of transactions T_1 and T_2 respectively, then schedules S_1 and S_2 are said to be view equivalent schedule if it satisfies following three conditions:

• If transaction T_1 reads a data item A from the database initially in schedule S_1, then in schedule S_2 also, T_1 must perform the initial read of the data item X from the database. This is same for all the data items. In other words –the initial reads must be same for all data items.

• If data item A has been updated at last by transaction T_1 in schedule S_1, then in schedule S_2 also, the data item A must be updated at last by transaction T_1.

• If transaction T_1 reads a data item that has been updated by the transaction T_1 in schedule S_1 then in schedule S_2 also, transaction T_1 must read the same data item that has been updatedby transaction T_1. In other words the Write-Read sequence must be same.



Sr.No.	Conflict serializability	View serializability
1	Every conflict serializable is view serializable.	Every view serializable schedule is not necessarily conflict serializable.
2.	It is easy to test conflict serializability.	It is complex to test view serializability.

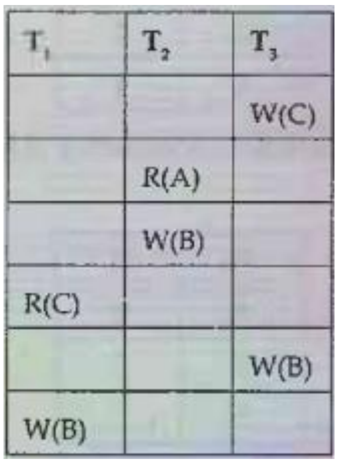
Steps to check whether the given schedule is view serializable or not

Step 1: If the schedule is conflict serializable then it is surely view serializable because conflict serializability is a restricted form of view serializability.

Step 2: If it is not conflict serializable schedule then check whether there exist any blind write operation. The blind write operation is a write operation without reading a value. If there does not exist any blind write then that means the given schedule is not view serializable. In other words if a blind write exists then that means schedule may or may not beview conflict.

Step 3: Find the view equivalence schedule

Example 3.5.6 *Consider the following schedules for checking if these are view serializable or not.*



T_1	T_2	T_3
		W(C)
	R(A)	
	W(B)	
R(C)		
		W(B)
W(B)		

Solution:

i) The initial read operation is performed by T_2 on data item A or by T_1 on data item C. Hence we will begin with T_2 or T_1. We will choose T_2 at the beginning.

ii) The final write is performed by T_1 on the same data item B. Hence T_1 will be at the last position.

iii) The data item C is written by T_3 and then it is read by T_1. Hence T_3 should before T_1.

Thus we get the order of schedule of view serializability as T_2 - T_1 – T_3

Example 3.5.7 Consider following two transactions:

T_1: read(A)read(B)

if A=0 then B:=B+1;

write(B)

T_2: read(B);

read(A);

if B=0 then A:=A+1;

write(A)

Let consistency requirement be A=0 V B=0 with A=B=0 the initial values.

1) Show that every serial execution involving these two transactions preserves the consistency of the Database?

2) Show a concurrent execution of T_1 and T_2 that produces a non serializable

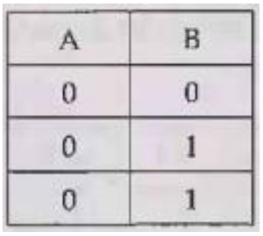
schedule?

3) Is there a concurrent execution of T_1 and T_2 that produces a serializable

schedule?

Solution: 1) There are two possible executions: T_1 ->T_2 or T_2->T_1

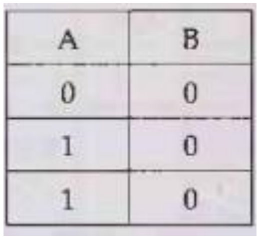
Consider case T_1->T_2 then



A	B
0	0
0	1
0	1

AvB =A OR B=FVT-T. This means consistency is met.

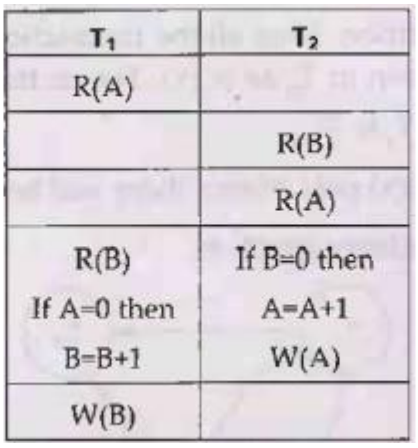
Consider case T_2->T_1 then



A	B
0	0
1	0
1	0

AvB = A OR B = FvT=T. This means consistency is met.

2) The concurrent execution means interleaving of transactions T_1 and T_2. It can be



T_1	T_2
R(A)	
	R(B)
	R(A)
R(B) If A=0 then B=B+1	If B=0 then A=A+1 W(A)
W(B)	

This is a non-serializable schedule.

(3) There is no concurrent execution resulting in a serializable schedule.

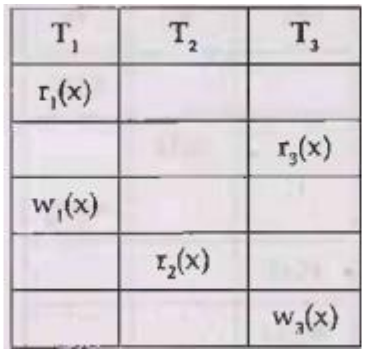
Example 3.5.8 *Test serializability of the following schedule:*

i) r1(x);r,(x);w,(x);r2(x);w (x) ii) 1,(x);12(x);W;(x);1,(x);w,(x)

Solution:

i) r_1(x);r_3(x);w_1(x);r_2(x);w_3(x)

The r_1 represents the read operation of transaction T_1, w_3 represents the write operation on transaction T_3 and so on. Hence from given sequence the schedule for three transactions can be represented as follows:



T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

Step 1: We will use the precedence graph method to check the serializability. As there are three transactions, three nodes are created for each transaction.

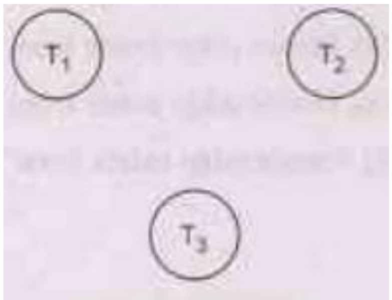


Fig. 3.5.5 : Nodes

Step 2: We will read from top to bottom. Initially we read r,(x) and keep on moving bottomin search of write operation. Here all the transactions work on same data item i.e. x. Now we get a write operation in T_1 as $w_3(x)$. Hence the dependency is from T_1 to T_3. Therefore we draw edge from T_1 to T_3.

Similarly, for $r_3(x)$ we get $w_1(x)$ pair. Hence there will be edge from T_3 to T_1. Continuing in this fashion we get the precedence graph as

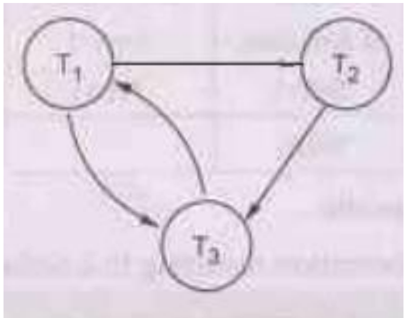
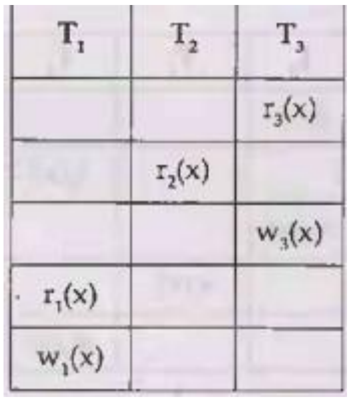


Fig. 3.5.6 : Precedence Graph

Step 3: As cycle exists in the above precedence graph, we conclude that it is not serializable.

ii) $r_3(x);r_2(x);w_3(x);r_1(x);w_1(x)$

From the given sequence the schedule can be represented as follows:



T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		

Step 1: Read the schedule from top to bottom for pair of operations. For $r_3(x)$ we get $w_1(x)$ pair. Hence edge exists from T, to T, in precedence graph.

There is a pair from $r_2(x)$: $w_3(x)$. Hence edge exists from T_2 to T_3.

There is a pair from $r_2(x)$: $w_1(x)$. Hence edge exists from T_2 to T_1.

There is a pair from $w_2(x)$: $r_1(x)$. Hence edge exists from T_3 to T_1.

Step 2: The precedence graph will then be as follows-

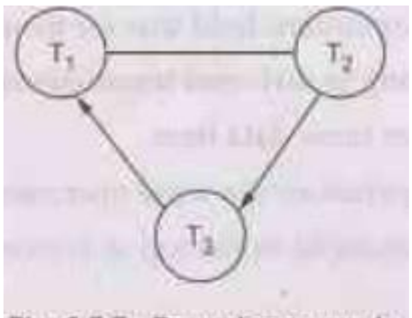
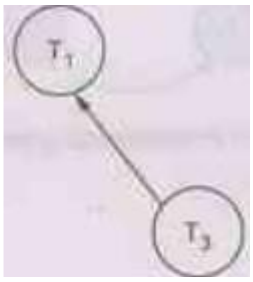


Fig. 3.5.7 : Precedence graph

Step 3: As there is no cycle in the above graph, the given schedule is serializable.

Step 4: The searializability order for consistent schedule will be obtained by applying topological sorting on above drawn precedence graph. This can be achieved as follows,

Sub-Step 1: Find the node having no incoming edge. We obtain T_2 is such a node. Hence T_2 is at the beginning of the serializability sequence. Now delete T_2. The Graph will be



Sub-Step 2: Repeat sub-Step 1, We obtain T_3 and T_1 nodes as a sequence.

Thus we obtain the sequence of transactions as T_2, T_3 and T_1. Hence the serializability order is

$r_2(x);r_3(x);w_3(x):r_1(x);w_1(x)$

Example 3.5.9 *Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.*

S1: T1: R(X), T2: R(X), T1: W(Y), T2: W(Y)TI: R(Y), T2: R(Y)

*S2: T3: W(X), TI: R(X), T1: W(Y), T2: R(Z),*T2: W(Z) T3: R(Z)

For each of the schedules, answer the following questions:

i) What is the precedence graph for the schedule?

ii) Is the schedule conflict-serializable ? If so, what are all the conflict equivalent serial schedules?

iii) *Is the schedule view-serializable? If so, what are all the view equivalent serial schedules ?*

Solution: i) We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them -

- Both the operations belong to different transactions.
- Both the operations are on same data item.
- At least one of the two operations is a write operation

For S1: From above given example in the top to bottom scanning we find the conflict as

- **T1: W(Y), T2: W(Y) and**
- **T2: W(Y), T1: R(Y)**

Hence we will build the precedence graph. Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1:W(Y) to T_2:W(Y). Hence edge must be from T_1 to T_2. Similarly for second conflict, there will be the edge from T_2 to T_1

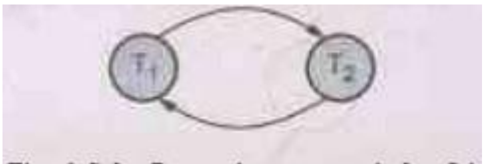


Fig. 3.5.8 : Precedence graph for S1

For S2: The conflicts are

- **T3: W(X), T1: R(X)**
- **T2: W(Z) T3: R(Z)**

Hence the precedence graph is as follows –

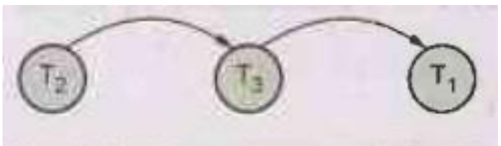


Fig. 3.5.9 : Precedence graph for S2

i)

• S1 is not conflict-serializable since the dependency graph has a cycle.

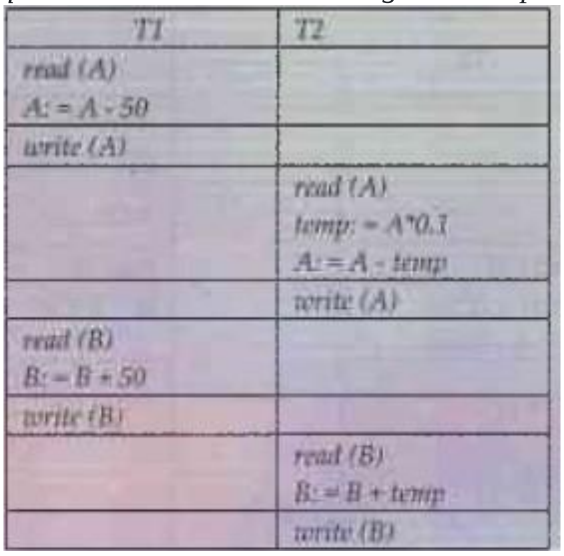
• S2 is conflict-serializable as the dependency graph is acylic. The order T2-T3-T1 is the only equivalent serial order

ii)

• S1 is not view serializable.

• S2 is trivially view-serializable as it is conflict serializable. The only serial order allowed is T2-T3-T1.

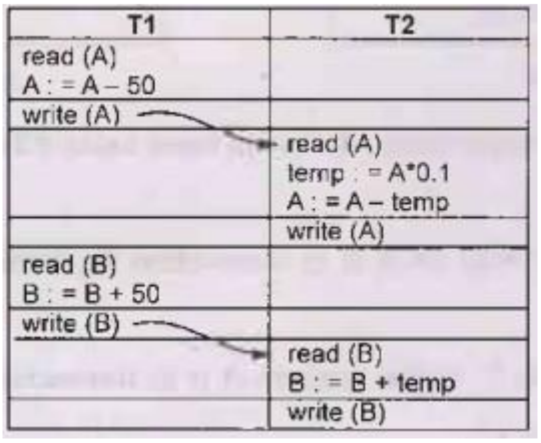
Example 3.5.10 *Check whether following schedule is view serializable or not. Justify your answer. (Note: T1 and T2 are transactions). Also explain the concept of view equivalent schedules and conflict equivalent schedule considering the example schedule given below :*



T1	T2
read (A) A: = A - 50	
write (A)	
	read (A) temp: = A*0.1 A: = A - temp
	write (A)
read (B) B: = B + 50	
write (B)	
	read (B) B: = B + temp
	write (B)

Solution:

Step 1: We will first find if the given schedule is conflict serializable or not. For that purpose, we will find the conflicting operations. These are as shown below –



T1	T2
read (A) A : = A – 50	
write (A)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
read (B) B : = B + 50	
write (B)	
	read (B) B : = B + temp
	write (B)

The precedence graph is as follows -

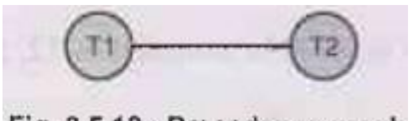


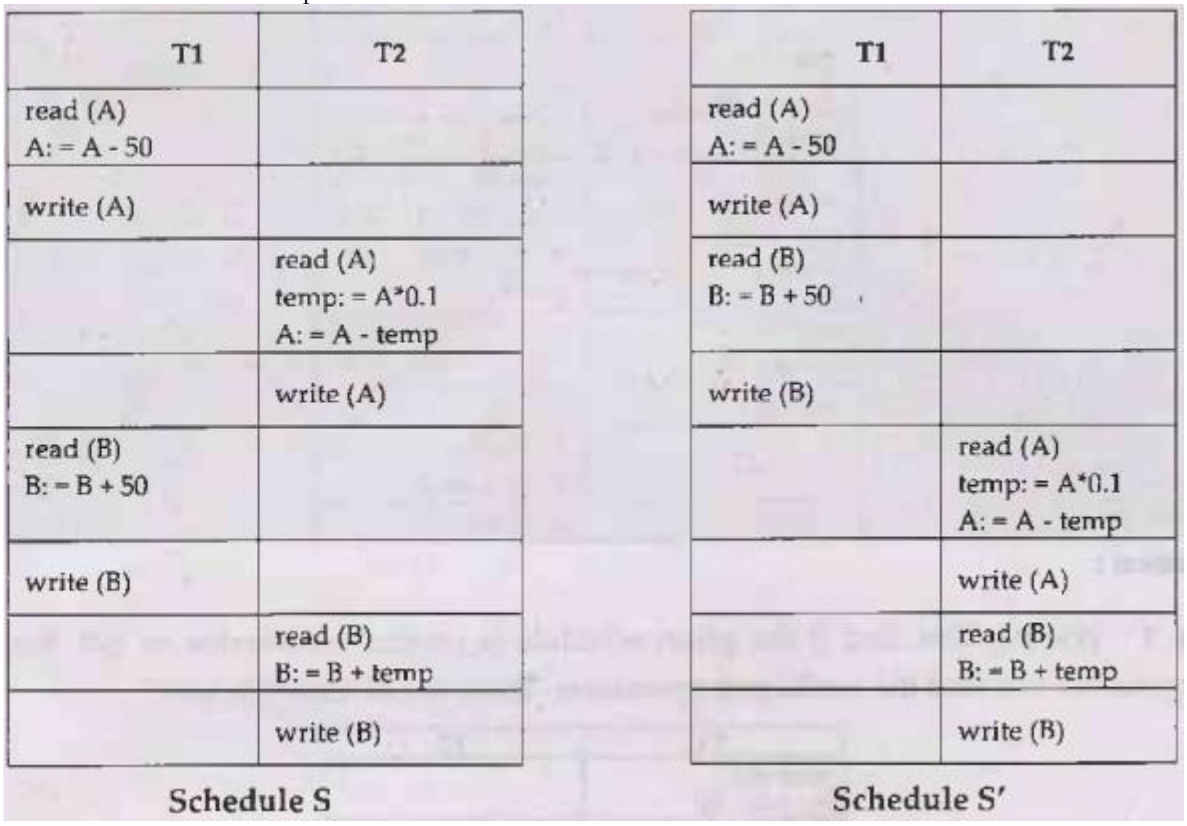
Fig. 3.5.10 : Precedence graph

As there exists no cycle, the schedule is conflict serializable. The possible serializability order can be T1 - T2

Now we check it for view serializability. As we get the serializability order as T1 - T2, we will find the view equivalence with the given schedule as serializable schedule.

Let S be the given schedule as given in the problem statement. Let the serializable schedule isS'={T1,T2}.

These two schedules are represented as follows:



T1	T2
read (A) A: = A - 50	
write (A)	
	read (A) temp: = A*0.1 A: = A - temp
	write (A)
read (B) B: = B + 50	
write (B)	
	read (B) B: = B + temp
	write (B)

Schedule S

T1	T2
read (A) A: = A - 50	
write (A)	
read (B) B: = B + 50	
write (B)	
	read (A) temp: = A*0.1 A: = A - temp
	write (A)
	read (B) B: = B + temp
	write (B)

Schedule S′

Now we will check the equivalence between them using following conditions -

(1) Initial Read

In schedule S initial read on A is in transaction T1. Similarly initial read on B is in transaction T1.

Similarly in schedule S', initial read on A is in transaction T1. Similarly initial read on B is in transaction T1.

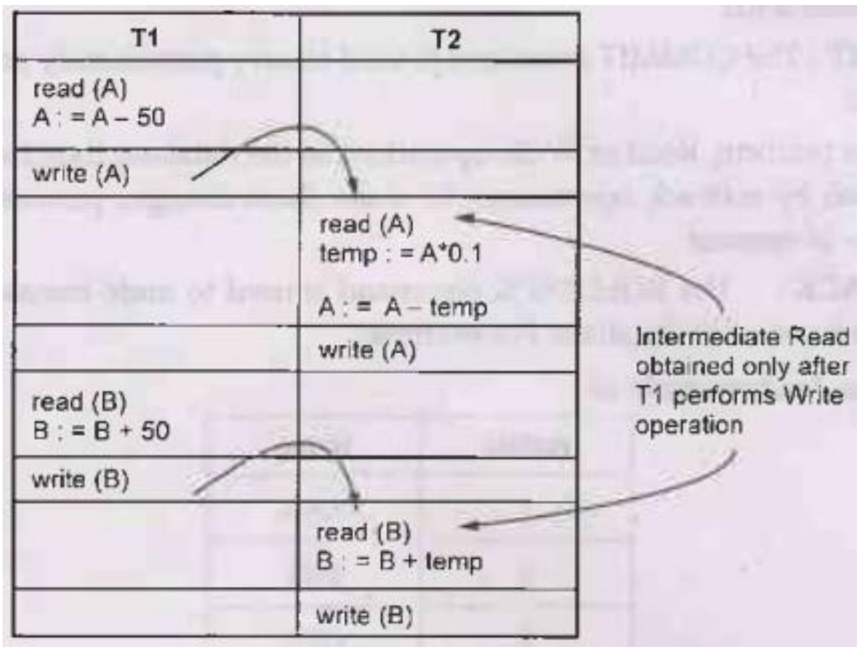
(2) Final Write

In schedule S final write on A is in transaction T2. Similarly final write on B is in transaction T2.

In schedule S' final write on A is in transaction T2. Similarly final write on B is in transaction T2

(3) Intermediate Read

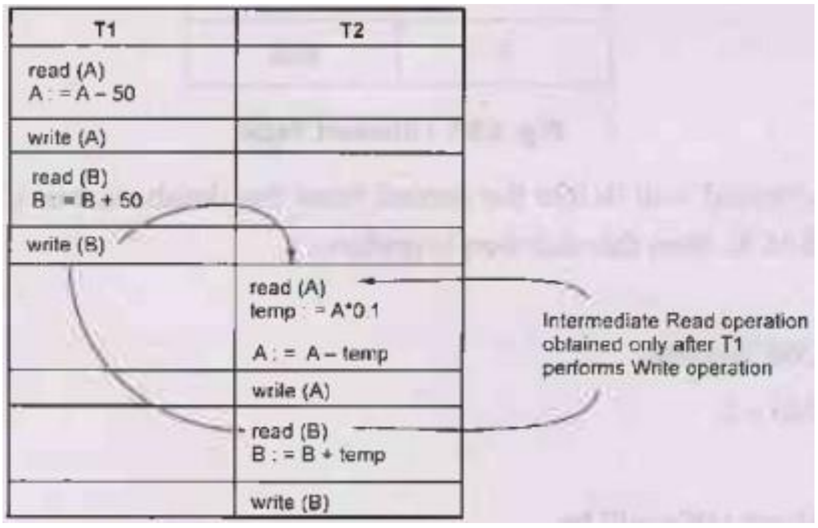
Consider schedule S for finding intermediate read operation.



T1	T2
read (A) A : = A – 50	
write (A)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
read (B) B : = B + 50	
write (B)	
	read (B) B : = B + temp
	write (B)

Intermediate Read obtained only after T1 performs Write operation

Similarly consider schedule S` for finding intermediate read operation



T1	T2
read (A) A : = A – 50	
write (A)	
read (B) B : = B + 50	
write (B)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
	read (B) B : = B + temp
	write (B)

Intermediate Read operation obtained only after T1 performs Write operation

In both the schedules S and S', the intermediate read operation is performed by T2 only after T1 performs write operation.

Thus all the above three conditions get satisfied. Hence given schedule is view serializable.

Possible Questions

1. *Explain Conflict serializability and view serializability*

2. *Discuss in detail about the testing of serializability.*

Transaction Support in SQL

The COMMIT, ROLLBACK, and SAVEPOINT are collectively considered as Transaction Commands

Transaction Support in SQL

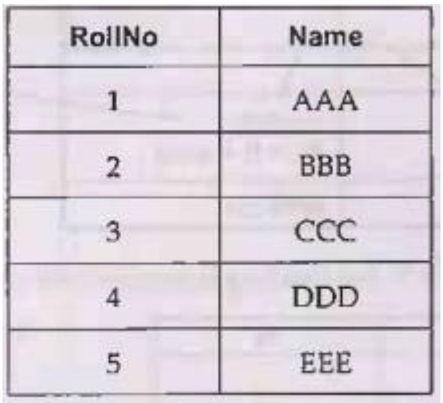
The COMMIT, ROLLBACK, and SAVEPOINT are collectively considered as Transaction Commands

(1) COMMIT: The COMMIT command is used to save permanently any transaction to database.

When we perform, Read or Write operations to the database then those changes can be undone by rollback operations. To make these changes permanent, we should make use of commit

(2) ROLLBACK: The ROLLBACK command is used to undo transactions that have not already saved to database. For example

Consider the database table as



RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

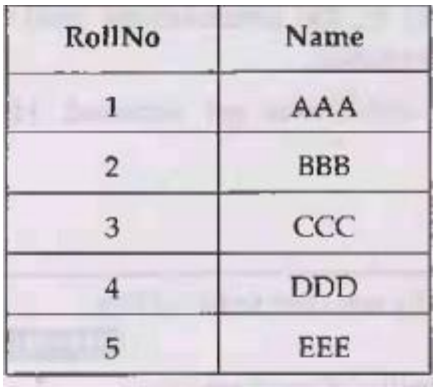
Fig. 3.6.1 : Student Table

Following command will delete the record from the database, but if we immediately performs ROLLBACK, then this deletion is undone.

For instance -

```
DELETE FROM Student
WHERE RollNo = 2;
ROLLBACK;
```

Then the resultant table will be



RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

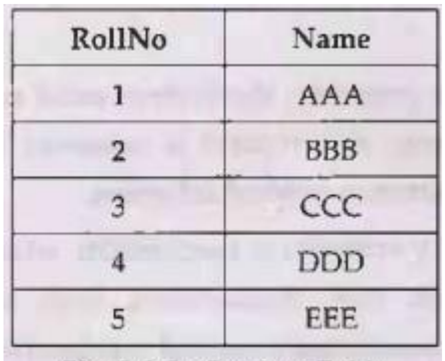
(3) SAVEPOINT: A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction. The SAVEPOINT can be created as

SAVEPOINT savepoint_name;

Then we can ROLLBACK to SAVEPOIT as

ROLLBACK TO savepoint_name;

For example - Consider Student table as follows –



RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 3.6.2 : Student Table

Consider Following commands

SQL> SAVEPOINT S1 SQL>DELETE FROM Student

Where RollNo=2;

SQL> SAVEPOINT S2

SQL>DELETE FROM Student

Where RollNo=3;

SQL> SAVEPOINT S3

SQL>DELETE FROM Student

Where RollNo=4

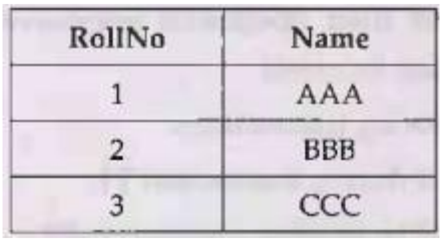
SQL> SAVEPOINT S4

SQL>DELETE FROM Student

Where RollNo=5

SQL> ROLLBACK TO S3;

Then the resultant table will be



RollNo	Name
1	AAA
2	BBB
3	CCC

Thus the effect of deleting the record having Roll No 2, and Roll No3 is undone.

Concurrency Control

One of the fundamental properties of a transaction is isolation.When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.

Part II Concurrency Control

Concurrency Control

• One of the fundamental properties of a transaction is isolation.

• When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.

• A database can have multiple transactions running at the same time. This is called concurrency.

• To preserve the isolation property, the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called concurrency control schemes.

• **Definition of concurrency control:** A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called concurrency control mechanism.

• The concurrency control can be achieved with the help of various protocols such as - lock based protocol, Deadlock handling, Multiple Granularity, Timestamp based protocol, and validation based protocols.

Possible Question

1. What is concurrency control? How it is implemented in DBMS? Briefly elaborate diagrams and examples.

Need for Concurrency

Concurrent execution of transactions over shared database creates several data integrity and consistency problems

Need for Concurrency

Following are the purposes of concurrency control -

• To ensure isolation

• To resolve read-write or write-write conflicts

• To preserve consistency of database

• Concurrent execution of transactions over shared database creates several data integrity and consistency problems - these are

(1) Lost update problem: This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

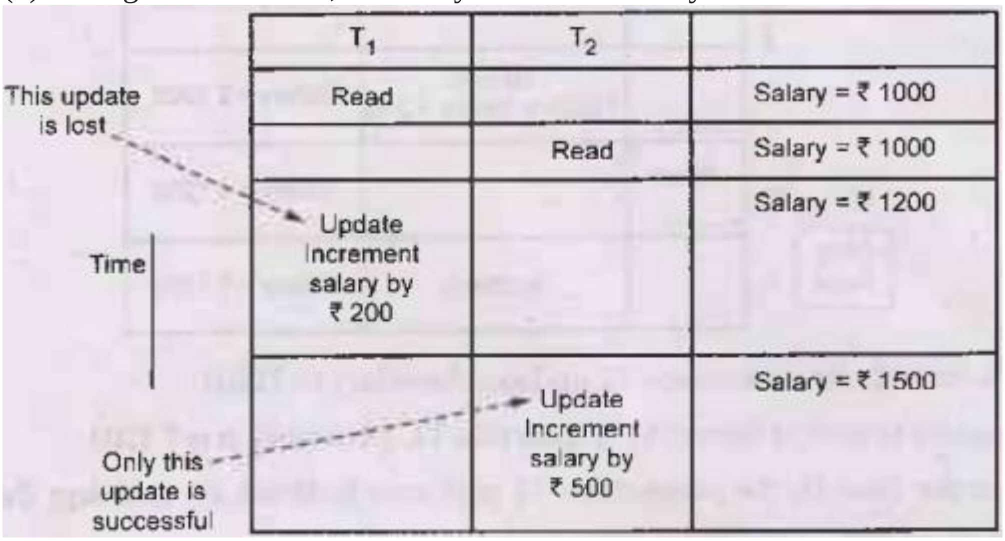
For example - Consider following transactions

(1) Salary of Employee is read during transaction T1.

(2) Salary of Employee is read by another transaction T2.

(3) During transaction T1, the salary is incremented by 200

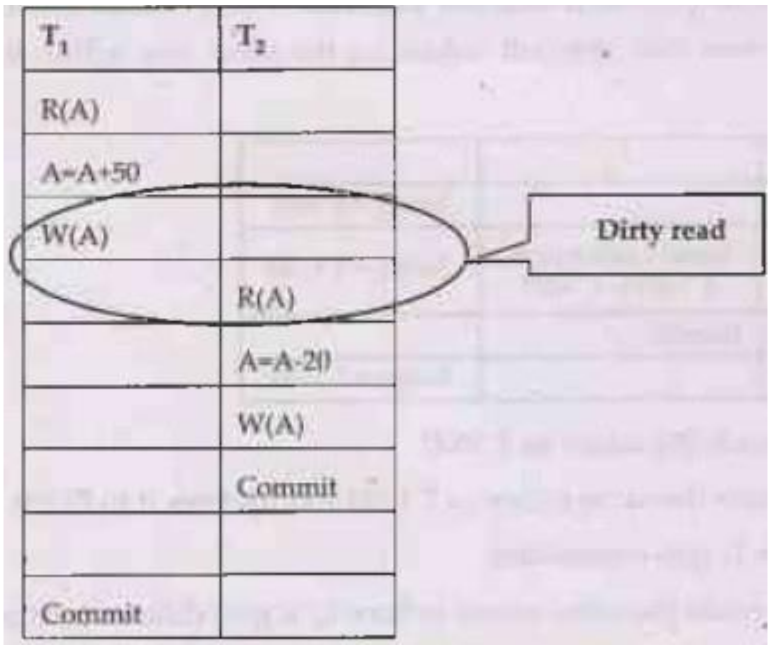
(4) During transaction T2, the salary is incremented by 500


This update is lost
Time
Only this update is successful

T_1	T_2	
Read		Salary = ₹ 1000
	Read	Salary = ₹ 1000
Update Increment salary by ₹ 200		Salary = ₹ 1200
	Update Increment salary by ₹ 500	Salary = ₹ 1500

The result of the above sequence is that the update made by transaction T1 is completely lost. Therefor this problem is called as lost update problem.

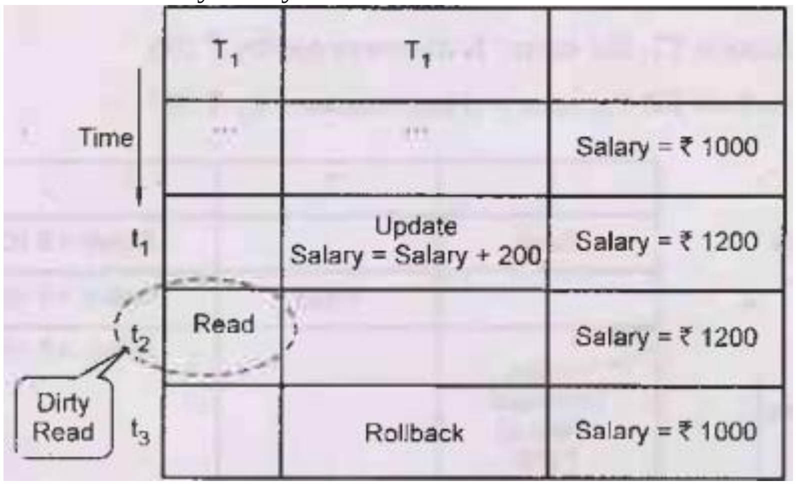
(2) Dirty read or Uncommited read problem: The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction



T_1	T_2
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A-20
	W(A)
	Commit
Commit	

For example - Consider following transactions -

Assume initially salary is = 1000



Time	T_1	T_1	
	...	...	Salary = ₹ 1000
t_1		Update Salary = Salary + 200	Salary = ₹ 1200
t_2	Read		Salary = ₹ 1200
t_3		Rollback	Salary = ₹ 1000

(1) At the time t1, the transaction T2 updates the salary to 1200

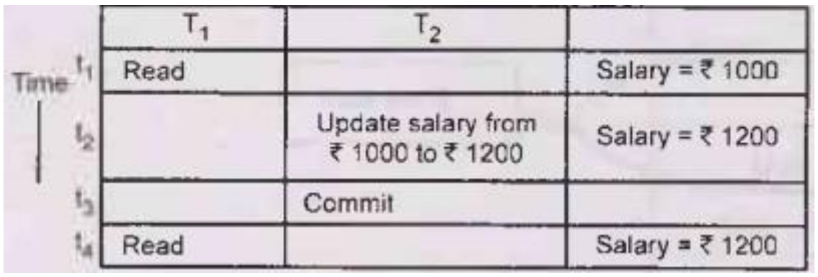
(2) This salary is read at time t2 by transaction T1. Obviously it is 1200

(3) But at the time t3, the transaction T2 performs Rollback by undoing the changes made by T1 and T2 at time t1 and t2.

(4) Thus the salary again becomes = 1000. This situation leads to Dirty Read or Uncommited Read because here the read made at time t2(immediately after roid update of another transaction) becomes a dirty read.

(3) Non-repeatable read problem

This problem is also known as inconsistent analysis problem. This problem occurs when a particular transaction sees two different values for the same row within its lifetime. For example-



Time		T_1	T_2	
	t_1	Read		Salary = ₹ 1000
	t_2		Update salary from ₹ 1000 to ₹ 1200	Salary = ₹ 1200
	t_3		Commit	
	t_4	Read		Salary = ₹ 1200

(1) At time t_1, the transaction T_1 reads the salary as 1000

(2) At time t_2 the transaction T_2 reads the same salary as 1000 and updates it to 1200

(3) Then at time t_3, the transaction T_2 gets committed.

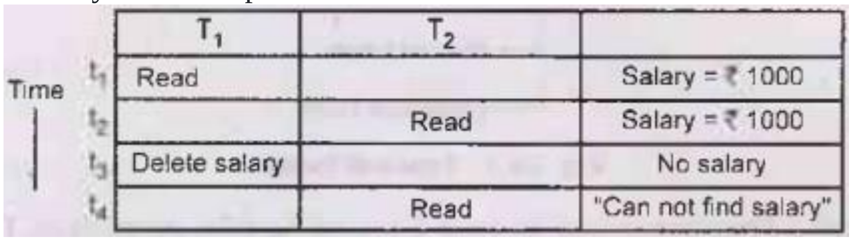
(4) Now when the transaction T_1 reads the same salary at time t_4, it gets different value than what it had read at time t_1. Now, transaction T_1, cannot repeat its reading operation. Thus inconsistent values are obtained.

Hence the name of this problem is non-repeatable read or inconsistent analysis problem.

(4) Phantom read problem

The phantom read problem is a special case of non repeatable read problem.

This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently. For example -



Time		T_1	T_2	
	t_1	Read		Salary = ₹ 1000
	t_2		Read	Salary = ₹ 1000
	t_3	Delete salary		No salary
	t_4		Read	"Can not find salary"

(1) At time t_1, the transaction T_1 reads the value of salary as 1000

(2) At time t_2, the transaction T_2 reads the value of the same salary as 1000

(3) At time t_3, the transaction T_1 deletes the variable salary.

(4) Now at time t_4, when T_2 again reads the salary it gets error. Now transaction T_2 can not identify the reason why it is not getting the salary value which is read just few time back.

This problem occurs due to changes in the database and is called phantom read problem.

Possible Questions

1. *Discuss the violations caused by each of the following: dirty read, non repeatable read and phantoms with suitable example.*

2. *What is concurrency control? How it is implemented in DBMS ? Briefly elaborate diagrams and examples.*

Locking Protocols

One of the method to ensure the isolation property in transactions is to require that data itemsbe accessed in a mutually exclusive manner.

Locking Protocols

Why Do We Need Locks?

• One of the method to ensure the isolation property in transactions is to require that data items be accessed in a mutually exclusive manner. That means, while one transaction is accessing a data item, no other transaction can modify that data item.

• The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

• Thus the lock on the operation is required to ensure the isolation of transaction.

Simple Lock Based Protocol

• Concept of Protocol: The lock based protocol is a mechanism in which there is exclusive use of locks on the data item for current transaction.

• **Types of Locks:** There are two types of locks used -

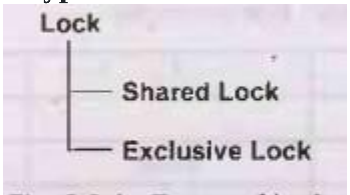


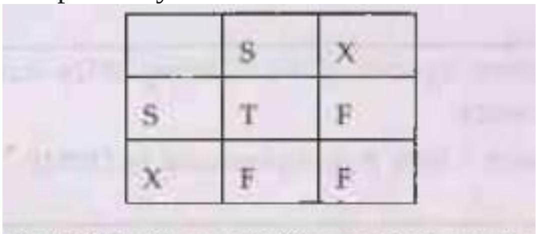
Fig. 3.9.1 : Types of locks

i) Shared Lock: The shared lock is used for reading data items only. It is denoted by Lock-S. This is also called as read lock.

ii) Exclusive Lock: The exclusive lock is used for both read and write operations. It is denoted as Lock-X. This is also called as write lock.

• The compatibility matrix is used while working on set of locks. The concurrency control manager checks the compatibility matrix before granting the lock. If the two modes of transactions are compatible to each other then only the lock will be granted.

• In a set of locks may consists of shared or exclusive locks. Following matrix represents the compatibility between modes of locks.



	S	X
S	T	F
X	F	F

Fig. 3.9.2 : Compatibility matrix for locks

Here T stands for True and F stands for False. If the control manager get the compatibility mode as True then it grant the lock otherwise the lock will be denied.

• **For example:** If the transaction T_1 is holding a shared lock in data item A, then the no control manager can grant the shared lock to transaction T_2 as compatibility is True.

But it cannot grant the exclusive lock as the compatibility is false. In simple words if transaction T_1 is reading a data item A then same data item A can be read by another transaction T_2 but cannot be written by another transaction.

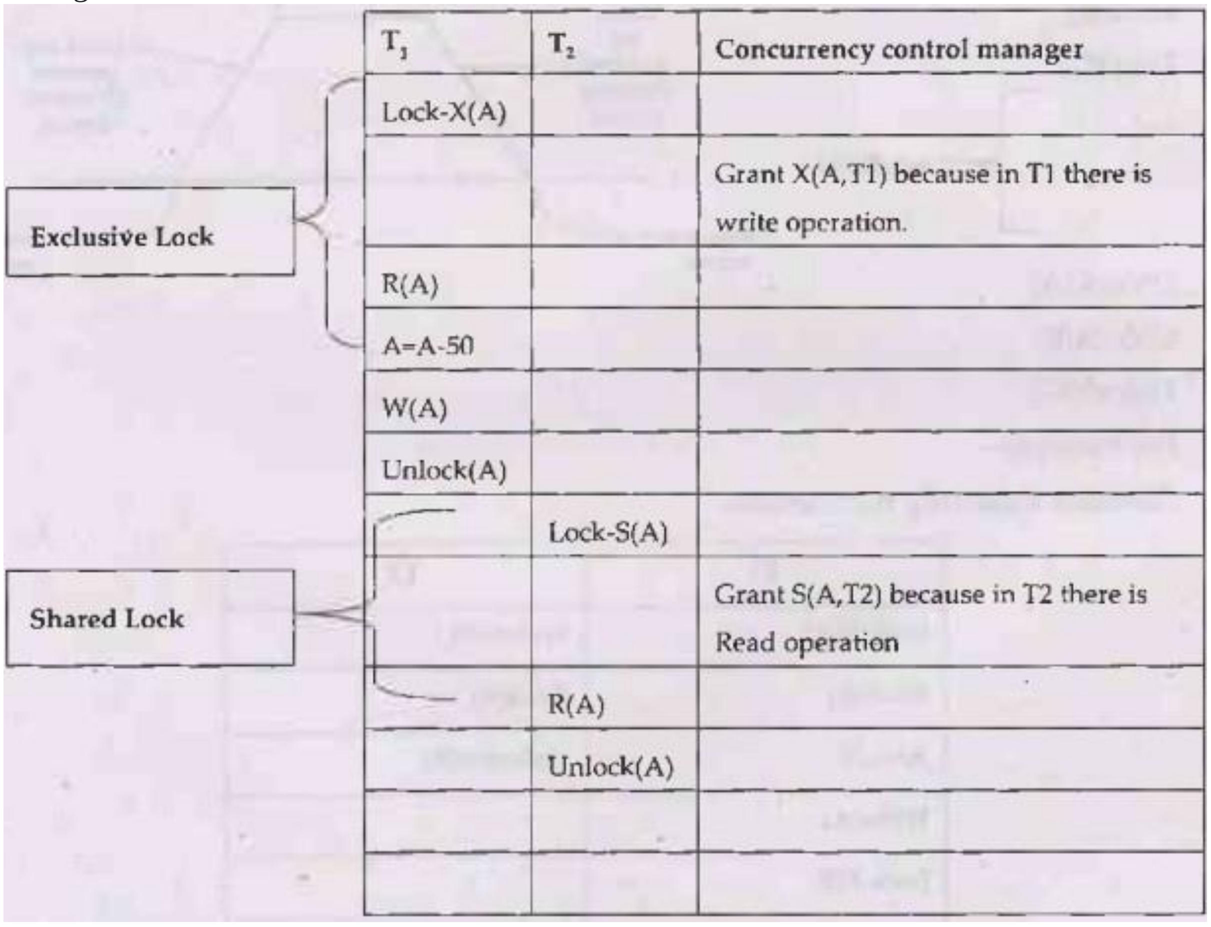
• Similarly if an exclusive lock (i.e. lock for read and write operations) is hold on the data item in some transaction then no other transaction can acquire Share or exclusive lock as the compatibility function denotes F. That means of some transaction is writing a data item A then another transaction can not read or write that data item A.

Hence the rule of thumb is

i) Any number of transactions can hold shared lock on an item.

ii) But exclusive lock can be hold by only one transaction.

• **Example of a schedule denoting shared and exclusive locks:** Consider following schedule in which initially A=100. We deduct 50 from A in T, transaction and Read the data item A in transaction T2. The scenario can be represented with the help of locks andconcurrency control manager as follows:


Exclusive Lock
Shared Lock

T_1	T_2	Concurrency control manager
Lock-X(A)		
		Grant X(A,T1) because in T1 there is write operation.
R(A)		
A=A-50		
W(A)		
Unlock(A)		
	Lock-S(A)	
		Grant S(A,T2) because in T2 there is Read operation
	R(A)	
	Unlock(A)	

Possible Questions

1.*State and explain the lock based concurrency control with suitable example.*
2.*What is Concurrency control? How is implemented in DBMS? Illustrate with suitable example.*

Two Phase Locking

The two phase locking is a protocol in which there are two phases: i) Growing phase (Locking phase),ii) Shrinking phase (Unlocking phase)

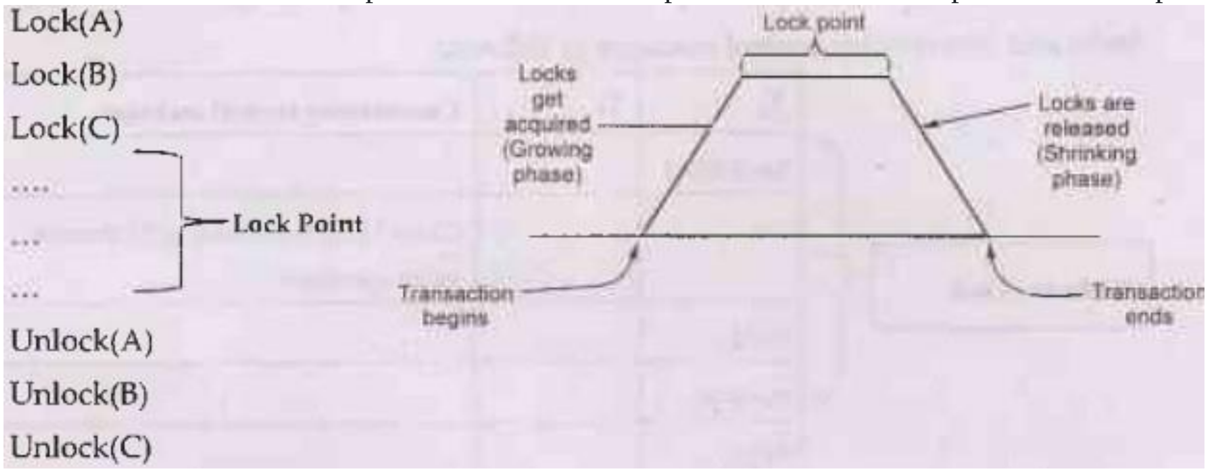
Two Phase Locking

• The two phase locking is a protocol in which there are two phases:

i) Growing phase (Locking phase): It is a phase in which the transaction may obtain locks but does not release any lock.

ii) Shrinking phase (Unlocking phase): It is a phase in which the transaction may release the locks but does not obtain any new lock.

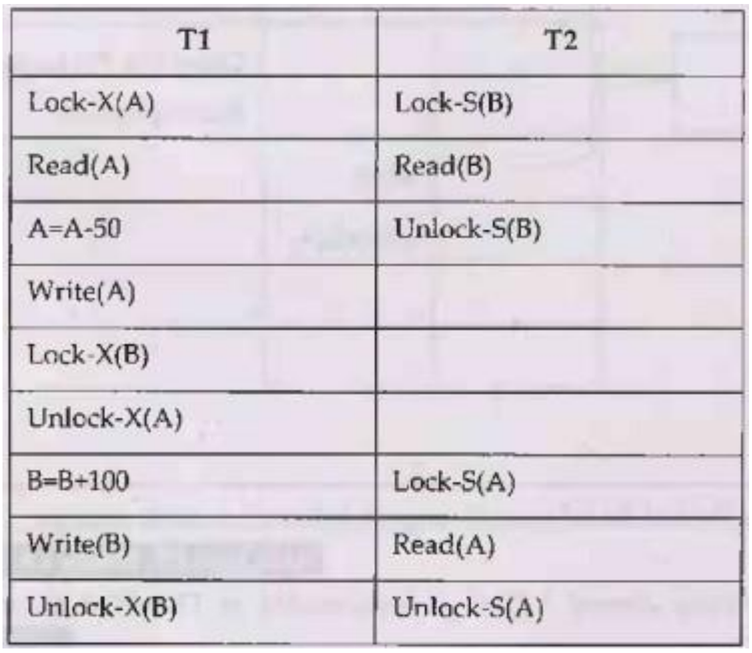
• **Lock Point:** The last lock position or first unlock position is called lock point. For example



Lock(A)
Lock(B)
Lock(C)
....
...
...
Unlock(A)
Unlock(B)
Unlock(C)

Lock Point
Lock point
Locks get acquired (Growing phase)
Locks are released (Shrinking phase)
Transaction begins
Transaction ends

Consider following transactions



T1	T2
Lock-X(A)	Lock-S(B)
Read(A)	Read(B)
A=A-50	Unlock-S(B)
Write(A)	
Lock-X(B)	
Unlock-X(A)	
B=B+100	Lock-S(A)
Write(B)	Read(A)
Unlock-X(B)	Unlock-S(A)

The important rule for being a two phase locking is - All Lock operations precede all the unlock operations.

In above transactions T_1 is in two phase locking mode but transaction T_2 is not in two phase locking. Because in T_2, the Shared lock is acquired by data item B, then data item B is read and then the lock is released. Again the lock is acquired by data item A, then the data item A is read and the lock is then reloased. Thus we get lock-unlock-lock-unlock sequence. Clearly this is not possible in two phase locking.

Example 3.10.1 *Prove that two phase locking guarantees serializability.*

Solution:

• Serializability is mainly an issue of handling write operation. Because any inconsistency may only be created by write operation.

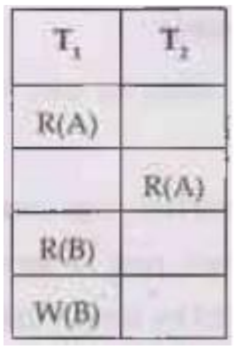
• Multiple reads on a database item can happen parallely.

• 2-Phase locking protocol restricts this unwanted read/write by applying exclusive lock.

• Moreover, when there is an exclusive lock on an item it will only be released in shrinking phase. Due to this restriction there is no chance of getting any inconsistent state.

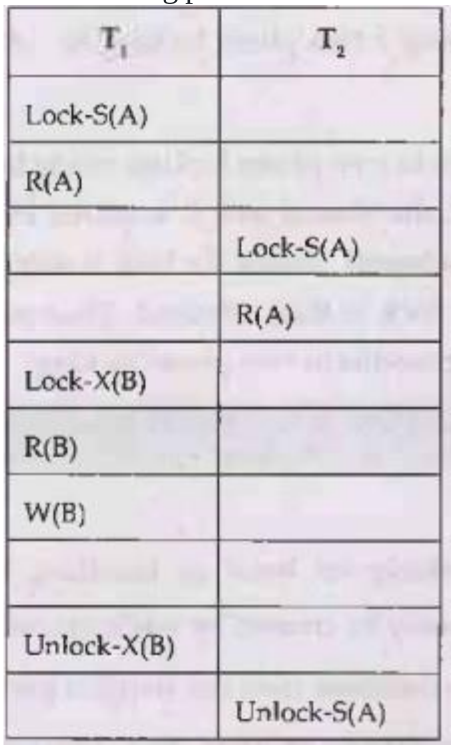
The serializability using two phase locking can be understood with the help of following example

Consider two transactions



T_1	T_2
R(A)	
	R(A)
R(B)	
W(B)	

Step 1: Now we will apply two phase locking. That means we will apply locks in growing and shrinking phase



T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-X(B)	
R(B)	
W(B)	
Unlock-X(B)	
	Unlock-S(A)

Note that above schedule is serializable as it prevents interference between two transactions. The serializability order can be obtained based on the lock point. The lock point is either last lock operation position or first unlock position in the transaction.

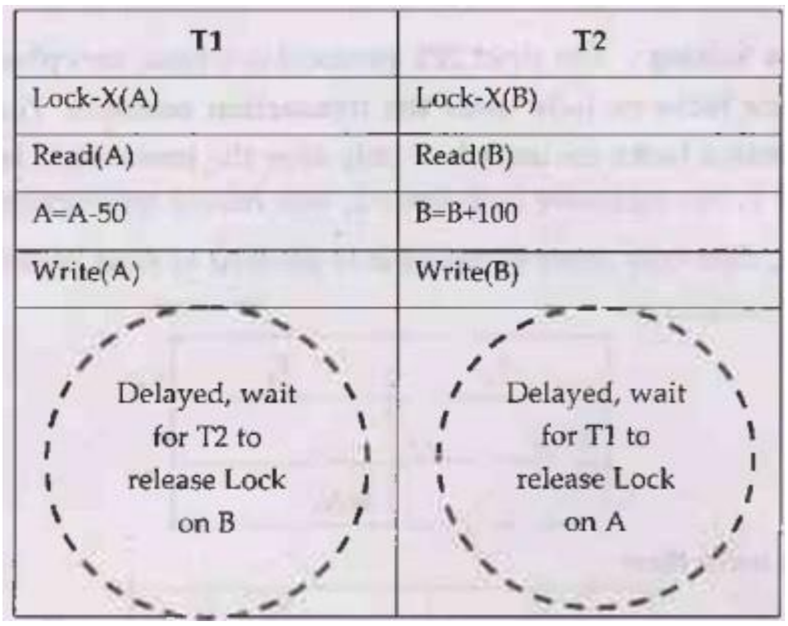
The last lock position is in T_1, then it is in T_2. Hence the serializability will be T_1->T_2 based on lock points. Hence the sequence can be **R1(A);R2(A);R1(B);W1(B)**

Limitations of Two Phase Locking Protocol

The two phase locking protocol leads to two problems - deadlock and cascading roll back.

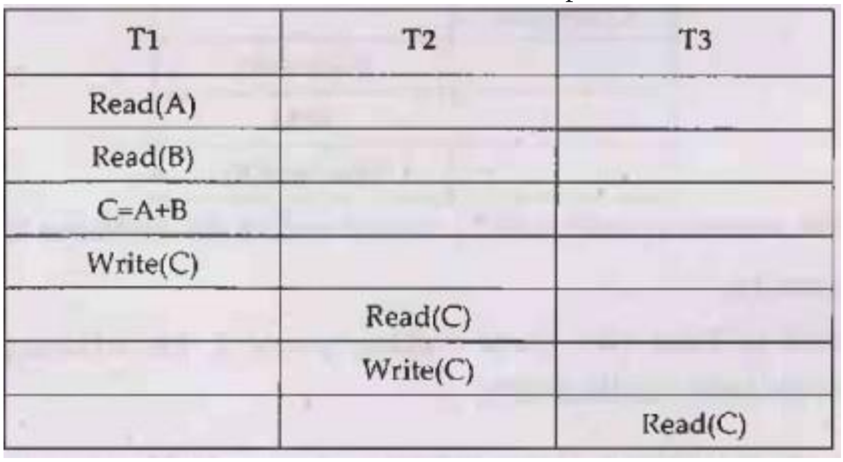
(1) Deadlock: The deadlock problem can not be solved by two phase locking. Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

For example



T1	T2
Lock-X(A)	Lock-X(B)
Read(A)	Read(B)
A=A-50	B=B+100
Write(A)	Write(B)
Delayed, wait for T2 to release Lock on B	Delayed, wait for T1 to release Lock on A

(2) Cascading Rollback: Cascading rollback is a situation in which transaction failure leads to a series of transaction rollback. For example -



T1	T2	T3
Read(A)		
Read(B)		
C=A+B		
Write(C)		
	Read(C)	
	Write(C)	
		Read(C)

When T_1 writes value of C then only T_2 can read it. And when T_2 writes the value of C then only transaction T_3 can read it. But if the transaction T_1 gets failed then automatically transactions T_2 and T_3 gets failed.

The simple two phase locking does not solve the cascading rollback problem. To solve the problem of cascading Rollback two types of two phase locking mechanisms can be used.

Types of Two Phase Locking

(1) Strict two phase locking: The strict 2PL protocol is a basic two phase protocol but all the exclusive mode locks be held until the transaction commits. That means in other words all the exclusive locks are unlocked only after the transaction is committed. That also means that if T1 has exclusive lock, then T, will release the exclusive lock only after commit operation, then only other transaction is allowed to read or write. For example Consider two transactions

T_1	T_2
W(A)	
	R(A)

If we apply the locks then

T_1	T_2
Lock-X(A)	
W(A)	
Commit	
Unlock(A)	
	Lock-S(A)
	R(A)
	Unlock-S(A)

Thus only after commit operation in T1, we can unlock the exclusive lock. This ensures the strict serializability.

Thus compared to basic two phase locking protocol, the advantage of strict 2PL protocol is it ensures strict serializability.

(2) Rigorous two phase locking: This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits. The transactions can be seriealized in the order in which they commit.

example - Consider transactions

T_1
R(A)
R(B)
W(B)

If we apply the locks then

T_1
Lock-S(A)
R(A)
Lock-X(B)
R(B)
W(B)
Commit
Unlock(A)
Unlock(B)

Thus the above transaction uses rigorous two phase locking mechanism

Example 3.10.2 *Consider the following two transactions:*

T1:read(A)

Read(B);

If A=0 then B=B+1;

Write(B)

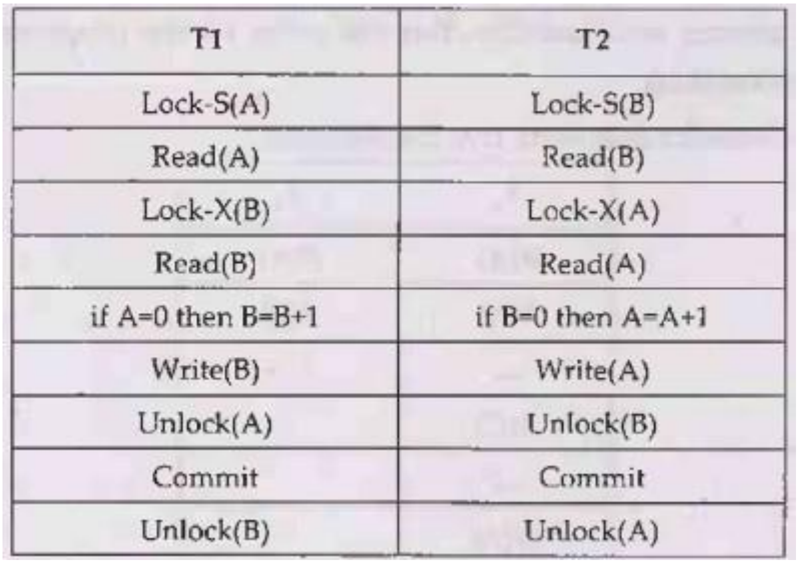
T2:read(B); read(A)

If B=0 then A=A+1

Write(A)

Add lock and unlock instructions to transactions T1 and T2, so that they observe two phase locking protocol. Can the execution of these transactions result in deadlock?

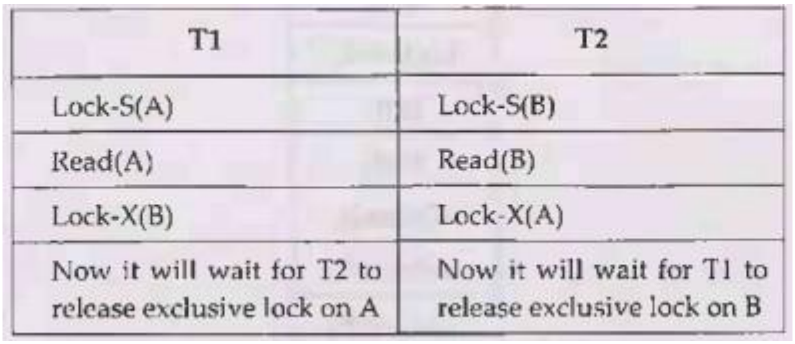
Solution:



T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Read(B)	Read(A)
if A=0 then B=B+1	if B=0 then A=A+1
Write(B)	Write(A)
Unlock(A)	Unlock(B)
Commit	Commit
Unlock(B)	Unlock(A)

This is lock-unlock instruction sequence help to satisfy the requirements for strict two phase locking for the given transactions.

The execution of these transactions result in deadlock. Consider following partial executionscenario which leads to deadlock.



T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Now it will wait for T2 to release exclusive lock on A	Now it will wait for T1 to release exclusive lock on B

Lock Conversion

Lock conversion is a mechanism in two phase locking mechanism - which allows conversion of shared lock to exclusive lock or exclusive lock to shared lock.

Method of Conversion :

First Phase:

- can acquire a lock-S on item
- can acquire a lock-X on item

• can convert a lock-S to a lock-X (upgrade)

Second Phase:

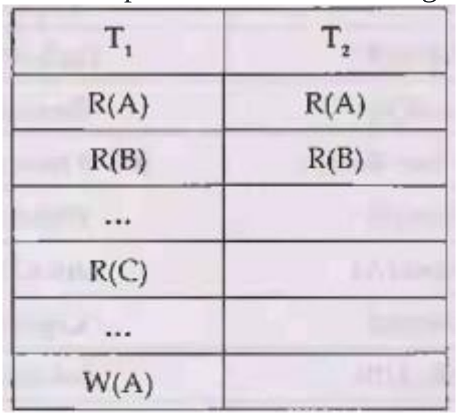
• can release a lock-S

• can release a lock-X

• can convert a lock-X to a lock-S (downgrade)

This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

For example - Consider following two transactions -

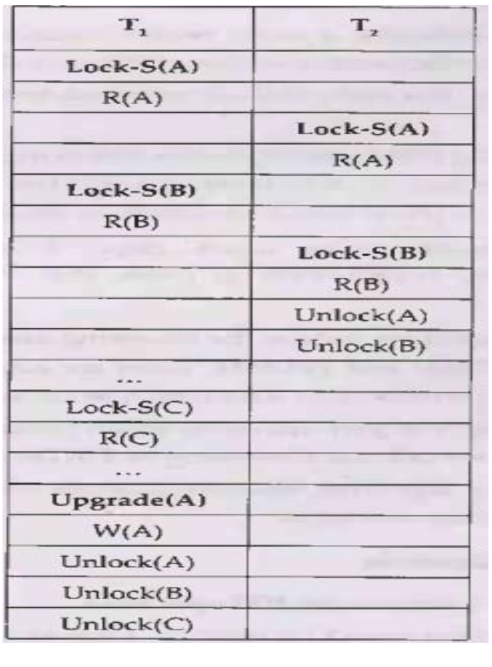


T_1	T_2
R(A)	R(A)
R(B)	R(B)
...	
R(C)	
...	
W(A)	

Here if we start applying locks, then we must apply the exclusive lock on data item A, because we have to read as well as write on data item A. Another transaction T_2 does not get shared lock on A until transaction T_1 performs write operation on A. Since transactionT_1 needs exclusive lock only at the end when it performs write operation on A, it is better ifT_1 could initially lock A in shared mode and then later change it to exclusive mode lock when it performs write operation. In such situation, the lock conversion mechanism becomesuseful.

When we convert the shared mode lock to exclusive mode then it is called upgrading and when we convert exclusive mode lock to shared mode then it is called downgrading.

Also note that upgrading takes place only in growing phase and downgrading takes place only in shrinking phase. Thus we can refine above transactions using lock conversion mechanism as follows -



T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-S(B)	
R(B)	
	Lock-S(B)
	R(B)
	Unlock(A)
	Unlock(B)
...	
Lock-S(C)	
R(C)	
...	
Upgrade(A)	
W(A)	
Unlock(A)	
Unlock(B)	
Unlock(C)	

Possible Questions

1. *What is concurrency control? Explain two phase locking protocol with an example.*

2. *Illustrate two phase locking protocol with an example.*

3. *Discuss elaborately the two phase locking protocol that ensures serializability.*

Timestamp Based Protocol

The time stamp ordering protocol is a scheme in which the order of transaction' is decided in advance based on their timestamps. Thus the schedules are serialized according to their timestamps.

Timestamp Based Protocol

• The time stamp ordering protocol is a scheme in which the order of transaction' is decided in advance based on their timestamps. Thus the schedules are serialized according to their timestamps.

• The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

• A larger timestamp indicates a more recent transaction or it is also called as younger transaction while lesser timestamp indicates older transaction.

• Assume a collection of data items that are accessed, with read and write operations, by transactions.

• For each data item X the DBMS maintains the following values:

• **RTS(X):** The Timestamp on which object X was last read (by some transaction T1, i.e., RTS(X):=TS(T;)) [Note that: RTS stands for Read Time Stamp]

• **WTS(X):** The Timestamp on which object X was last written (by some transaction T,, i.e., WTS(X):=TS(T)) [Note that: WTS stands for Write Time Stamp]

• For the following algorithms we use the following assumptions: - A data item X in the database has a RTS(X) and WTS(X). These are actually the timestamps of read and write operations performed on data item X at latest time.

• A transaction T attempts to perform some action (read or write) on data item X on some timestamp and we call that timestamp as TS(T).

• By timestamp ordering algorithm we need to decide whether transaction T has to be aborted or T can continue execution.

Basic Timestamp Ordering Algorithm

Case 1 (Read): Transaction T issues a read(X) operation

i) If $TS(T) < WTS(X)$, then read(X) is rejected. T has to abort and be rejected.

ii) If $WTS(X) \leq TS(T)$, then execute read(X) of T and update RTS(X).

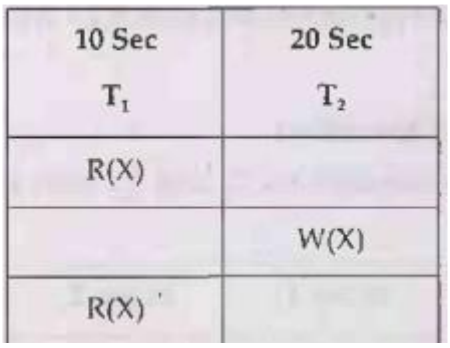
Case 2 (Write): Transaction T issues a write(X) operation

i) If $TS(T) < RTS(X)$ or if $TS(T) < WTS(X)$, then write is rejected

ii) If $RTS(X) \leq TS(T)$ or $WTS(X) \leq TS(T)$, then execute write(X) of T and update WTS(X).

Example for Case 1 (Read operation)

(i) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.



10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
R(X)	

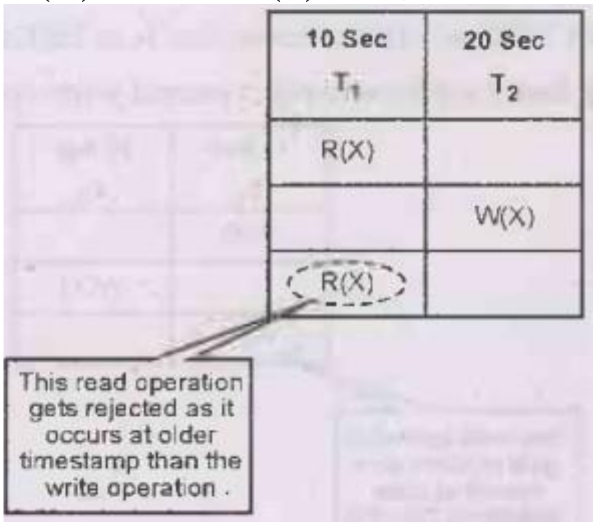
RTS(X) and WTS(X) is initially = 0

Then RTS(X)=10, when transaction T_1 executes

After that WTS(X) =20 when transaction T_2 executes

Now if Read operation R(X) occurs on transaction T_1 at $TS(T_1)$ = 10 then

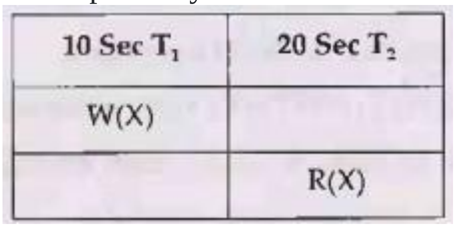
$TS(T_1)$ i.e. 10 <WTS(X) i.e. 20, hence we have to reject second read operation on T_1 i.e.



10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
R(X)	

This read operation gets rejected as it occurs at older timestamp than the write operation .

(ii) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.



10 Sec T_1	20 Sec T_2
W(X)	
	R(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T_1 executes.

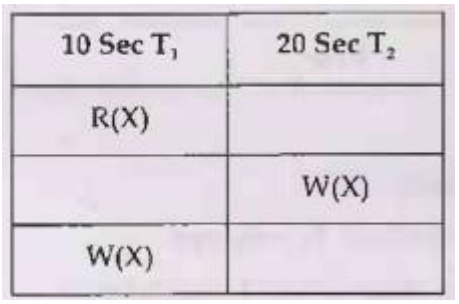
Now if Read operation R(X) occurs on transaction T_2 at $TS(T_2)$ = 20 then

$TS(T_2)$ i.e. 20 >WTS(X) which is 10, hence we accept read operation on T_2.

The transactionT_2 will perform read operation and now RTS will be updated as RTS(X)=20

Example for Case 2 (Write Operation)

(i) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec and 20 sec respectively.



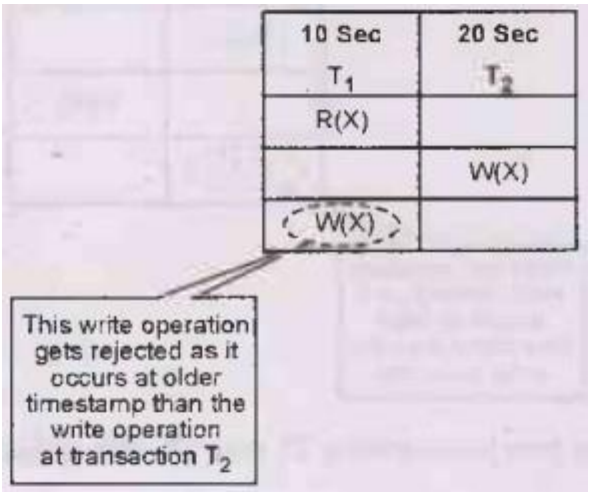
10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
W(X)	

RTS(X) and WTS(X) is initially = 0

Then RTS(X)=10, when transaction T_1 executes

After that WTS(X) =20 when transaction T_2 executes

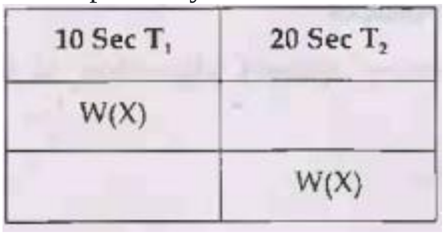
Now if Write operation W(X) occurs on transaction T_1 at $TS(T_1)$ = 10 then $TS(T_1)$ i.e. 10 <WTS(X), hence we have to reject second write operation on T_1 i.e.



10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
W(X)	

This write operation gets rejected as it occurs at older timestamp than the write operation at transaction T_2

(ii) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec and 20 sec respectively.



10 Sec T_1	20 Sec T_2
W(X)	
	W(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T_1 executes.

Now if write operation W(X) occurs on transaction T_2 at $TS(T_2)$ = 20 then

$TS(T_2)$ i.e. 20 >WTS(X) which is 10, hence we accept write operation on T_2. The transaction T_2 will perform write operation and now WTS will be updated as

WTS(X) = 20

Advantages and disadvantages of time stamp ordering

Advantages

(1)Schedules are serializable
(2)No waiting for transaction and hence there is no deadlock situation.

Disadvantages

(1) Schedules are not recoverable once transactions occur.

(2) Same transaction may be continuously aborted or restarted.

Multi-version Concurrency Control

The DBMS maintains multiple physical versions of single logical object in the database. When a transaction writes to an object, the database creates a new version of that object.

Multi-version Concurrency Control

• The DBMS maintains multiple physical versions of single logical object in the database.

• When a transaction writes to an object, the database creates a new version of that object.

• When a transaction reads an object, it reads the newest version that exists when the transaction started.

• In this technique, the writers don't block readers or readers don't block writer.

• This scheme makes use of:

i) Locking protocol and

ii) Timestamp protocol.

• The multiversion is now used in almost all database management system as a modern technique of concurrency control.

Validation and Snapshot Isolation

The optimistic concurrency control algorithm is basically a validation based protocol.

Validation and Snapshot Isolation

Validation Based Protocol

• The optimistic concurrency control algorithm is basically a validation based protocol.

• It works in three phases -

• **Read phase:**

• In this phase, transaction T is read.

• The values of various data items are read and stored in temporary variables.

• All the operations are then performed on temporary variables without updating the actual database.

• **Validation phase :**

• In this phase, the temporary variable value is validated against the actual data in the database and it checked whether the transaction T follows serializability or not.

• **Write phase**

• If the transaction T is validated then only the temporary results are written to database, otherwise the system rolls back.

• Each phase has following different timestamps

• **Start (T_i):**

• It contains the timestamp when T_i starts the execution.

• **Validation (T_i):**

• It contains the timestamp when transaction T_i finishes the read phase and starts its validation phase.

• **Finish (T_i):**

• It contains the timestamp when transaction T, finishes its write phase.

• With the help of timestamp in validation phase, this protocol determines if the transaction will commit or rollback. Hence

TS (T_i) = validation (T_i).

• The serializability is determined at the validation process, it can't be determined in advance

• While executing the transactions, this protocol gives greater degree of concurrency when there are less number of conflicts. That is because the serializability order is not pre-decided (validated and then executed) and relatively less transaction will have to be rolled back.

Snapshot Isolation

• The snapshot isolation is a multi-version concurrency control technique.

• In snapshot isolation, we can imagine that each transaction is given its own version, or snapshot, of the database when it begins. It reads data from this private version and is thus isolated from the updates made by other transactions.

• If the transaction updates the database, that update appears only in its own version, not in the actual database itself.

• Information about these updates is saved so that the updates can be applied to the "real" database if the transaction commits.

• When a transaction T enters the partially committed state, it then proceeds to the committed state only if no other concurrent transaction has modified data that T intends to update. Transactions that, as a result, cannot commit abort instead.

• Snapshot isolation ensures that attempts to read data never need to wait.

Multiple Granularity Locking

In database management system there are data at various levels depending upon the data sizes.

Multiple Granularity Locking

• In database management system there are data at various levels depending upon the data sizes.

• For example: A database contains a table and a table contains rows and each row contains some data value. Thus different levels of data can be database, tables, the records.

This can be represented in the form of tree. For example.

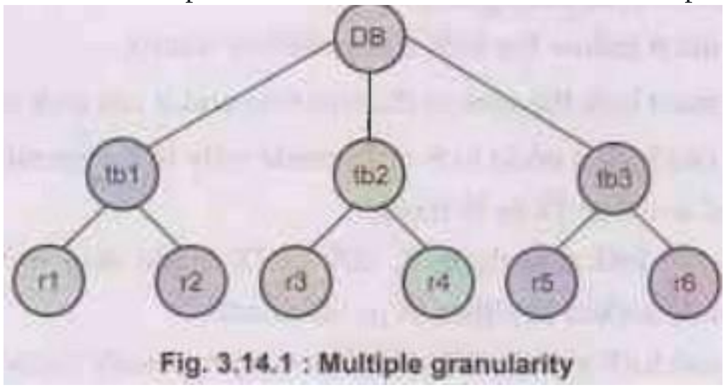


Fig. 3.14.1 : Multiple granularity

• Each node can be locked individually. When a transaction locks a node, in either shared or exclusive mode, the transaction also implicitly locks all the descendants of that node in the same lock mode.

• For example - if transaction T1 gets an explicit lock on table tbl in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that table. It does not need to lock the individual records r1 and r2.

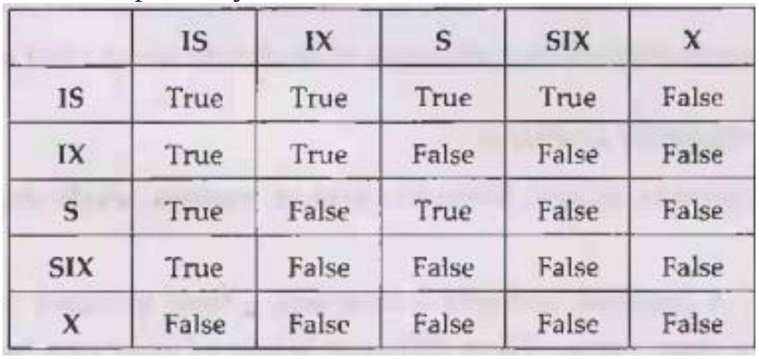
• A new mode of lock is introduced along with exclusive and shared locks. This is called intention mode lock. Thus In addition to S and X lock modes, there are three wo additional lock modes with multiple granularity:

• Intention-Shared (IS): Explicit locking at a lower level of the tree but only with shared locks.

• Intention-Exclusive (IX): Explicit locking at a lower level with exclusive or shared locks.

• Shared and Intention-Exclusive (SIX): The sub-tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.

• The compatibility matrix for these lock modes are described below:



	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

• With help of intention lock modes on the transactions the multiple-granularity locking protocol ensures serializability.

• The protocol follows following rules -

1) Transaction T_i must follow the lock-compatibility matrix.

2) Transaction T_i must lock the root of the tree first and it can lock it in any mode.

3) Transaction T_i can lock a node in S or IS mode only if T_i currently has the parent of the node locked in either IX or IS mode.

4) Transaction T_i can lock a node in X, SIX, or IX mode only if T_i currently has the parent of the node locked in either IX or SIX mode.

5) Transaction T_i, can lock a node only if T_i, has not previously unlocked any node. That means T_i, is two phase.c

6) Transaction T_i, can unlock a node only if T_i, currently has none of the children of the node locked.

Advantages of multiple granularity protocol

1) It enhances concurrency.

2) It ensures serializability.

3) It keeps track of what to lock and how to lock.

4) This type of locking can be hierarchically represented.

5) It makes it easy to decide which data item to be locked and which data item need to be unlocked.

Isolation levels

The consistency of the database is maintained with the help of isolation property(one of the property from ACID properties) of transaction.

Isolation Levels

• The consistency of the database is maintained with the help of isolation property(one of the property from ACID properties) of transaction.

• The transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system at particular instance.

As we know, concurrent execution of transaction over shared database creates various problems. Following are three commonly occurring anomalies

(1) Dirty read: This occurs when we read the uncommitted data. We should not read the uncommitted data because it causes many errors in the database.

For example - Let there are two transactions T1 and T2.

Step 1: Before T1 and T2 perform any operation Salary = 1000 Rs.

Step 2: T2 Writes the salary as Salary =1200 Rs.

Step 3: T1 Reads immediately the Salary as 1200 Rs.

Step 4: T2 rolls back and now Salary =1000 and commits.

Step 5: Now if T1 reads the Salary then it will get the value as 1000

That means the read operation at step 3 is a dirty read operation because T1 read the data before T2 commits the transaction

(2) Non repeatable read: This problem occurs when a particular transaction sees two different values for the same row within its lifetime.

For example - Let, there are two transactions T1 and T2

Step 1: At time t1, the transaction T1 reads the Salary = 1000 Rs.

Step 2: At time t2 the transaction T2 reads the same salary = 1000 Rs and updates it to 1200 Rs.

Step 3: Then at time t3, the transaction T2 gets committed.

Step 4: Now when the transaction T1 reads the same salary at time t4, it gets different value(as Rs.1200) than what it had read(Rs.1000) at time t1. Now, transaction T1 cannot repeat its reading operation.

Thus inconsistent values are obtained.

(3) Phantom read: It is a special case of non repeatable read. This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently.

For example - Let, there are two transactions T1 and T2

Step 1: At time t1, the transaction T1 reads the value of Salary = 1000 Rs.

Step 2: At time t2, the transaction T2 reads the value of the same salary as 1000Rs.

Step 3: At time t3, the transaction T1 deletes the variable salary.

Step 4: Now at time t4, when T2 again reads the salary it gets error. Now transaction T2 cannot identify the reason why it is not getting the salary value which is read just few time back.

Hence isolation levels are defined so that any two transactions can execute concurrently and integrity of data can be maintained.

- There are four levels of transaction isolation defined by SQL -
  - **Serializable:**
    - This is the Highest isolation level.
    - Serializable execution is defined to be an execution of operations in whichconcurrently executing transactions appears to be serially executing.
  - **Repeatable Read:**
    - This is the most restrictive isolation level.
    - The transaction holds read locks on all rows it references..
    - It holds write locks on all rows it inserts, updates, or deletes.
    - Since other transaction cannot read, update or delete these rows, it avoids non repeatable read.

• **Read Committed:**

• This isolation level allows only committed data to be read.

• Thus it does not allows dirty read (i.e. one transaction reading of data immediately after written by another transaction).

• The transaction hold a read or write lock on the current row, and thus prevent other rows from reading, updating or deleting it.

• **Read Uncommitted:**

• It is lowest isolation level.

• In this level, one transaction may read not yet committed changes made by other transaction.

• This level allows dirty reads.

In this level transactions are not isolated from each other.

To summerize, the above isolation levels -

• SERIALIZABLE - dirty reads, non-repeatable reads and phantoms not allowed; all schedules must be serializable

• REPEATABLE READ - dirty reads, non-repeatable reads not allowed, but phantoms allowed

• READ COMMITTED - dirty reads not allowed, but non-repeatable reads and phantoms allowed

• READ UNCOMMITTED - dirty reads, non-repeatable reads and phantoms allowed

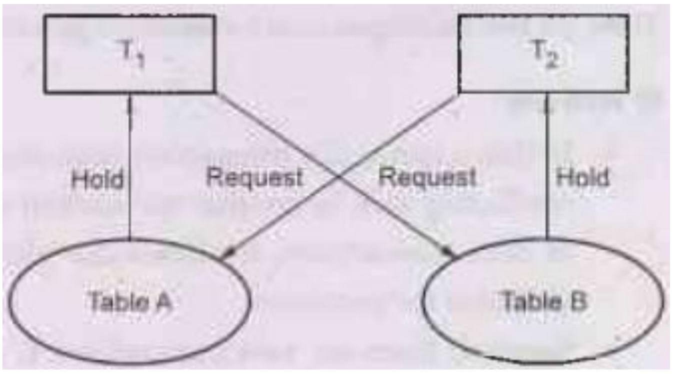
Deadlock Handling

Deadlock is a specific concurrency problem in which two transactions depend on each other for something.

Deadlock Handling

Deadlock is a specific concurrency problem in which two transactions depend on each other for something.

For example- Consider that transaction T1 holds a lock on some rows of table A and needs to update some rows in the B table. Simultaneously, transaction T2 holds locks on some rows in the B table and needs to update the rows in the A table held by Transaction T1.



Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. This situation is called deadlock in DBMS.

Definition: Deadlock can be formally defined as - " A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. "

There are four conditions for a deadlock to occur

A deadlock may occur if all the following conditions holds true.

1. Mutual exclusion condition: There must be at least one resource that cannot be used by more than one process at a time.

2. Hold and wait condition: A process that is holding a resource can request for vd's additional resources that are being held by other processes in the system.

3. No preemption condition: A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.

4. Circular wait condition: A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process In the ... so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

Deadlock can be handled using two techniques -

1. Deadlock Prevention
2. Deadlock Detection and deadlock recovery

1. Deadlock prevention :

For large database, deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occur. The DBMS analyzes the operations whether they can create deadlock situation or not, If they do, that transaction is never allowed to be executed.

There are two techniques used for deadlock prevention -

(i) Wait-Die:

• In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

• Suppose there are two transactions T; and T, and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

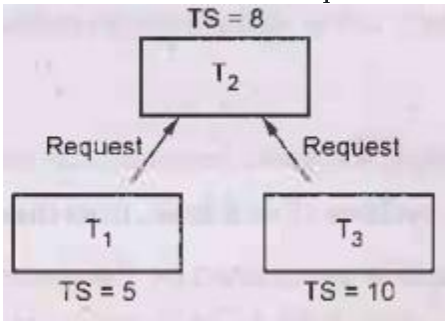
• Check if TS(T) < TS(T) - If T, is the older transaction and T, has held some resource, then T, is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for estb n resource until it is available.

• Check if TS(T;) < TS(T;) - If T; is older transaction and has held some resource and if T, is waiting for it, then T, is killed and restarted later with the random delay but with the same timestamp.

Timestamp is a way of assigning priorities to each transaction when it starts. If timestamp is lower then that transaction has higher priority. That means oldest transaction has highest priority.

For example-

Let T1 is a transaction which requests the data item acquired by Transaction T2. Similarly T3 is a transaction which requests the data item acquired by transaction T2.



Here TS(T1) i.e. Time stamp of T1 is less than TS(T3). In other words T1 is older than T3. Hence T1 is made to wait while T3 is rolledback.

(ii) Wound - wait:

• In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to not kill the transaction and release the resource. After some delay, the younger bevomer transaction is restarted but with the same timestamp.

• If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

Suppose T1 needs a resource held by T2 and T3 also needs the resource held by T2, with TS(T1)=5, TS(T2)=8 and TS(T3)=10, then T1 being older waits and T3 being younger dies. After the some delay, the younger transaction is restarted but with the same timestamp.

This ultimately prevents a deadlock to occur.

To summarize

	Wait-Die	Wound-wait
Older transaction needs a data item held by younger transaction	older transaction waits	younger transaction dies.
Younger transaction needs a data item held by older transaction	Younger transaction dies	Younger transaction dies.

2. Deadlock detection:

• In deadlock detection mechanism, an algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred or not. If deadlock is occurrence is detected, then the system must try to recover from it.

• Deadlock detection is done using wait for graph method.

Wait for graph

• In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.

• The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

• This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges.

• The set of vertices consists of all the transactions in the system.

• When transaction T_i requests a data item currently being held by transaction T_i, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when transactionT_j is no longer holding a data item needed by transaction T_i.

• For example - Consider following transactions, We will draw a wait for graph for this scenario and check for deadlock.

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

We will use three rules for designing the wait-for graph -

Rule 1: If T1 has Read operation and then T2 has Write operation then draw an edge T1->T2.

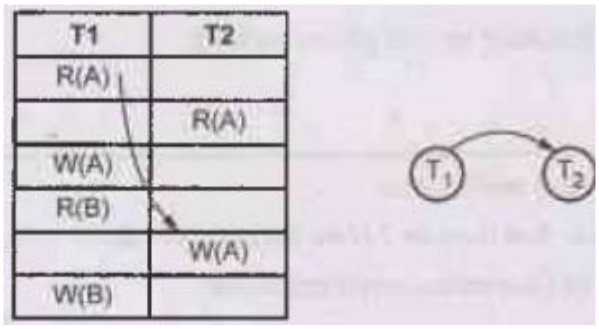
Rule 2: If T1 has Write operation and then T2 has Read operation then draw an edge T1->T2

Rule 3: If T1 has Write operation and then T2 has Write operation then draw an edge T1->T2

Let us draw wait-for graph

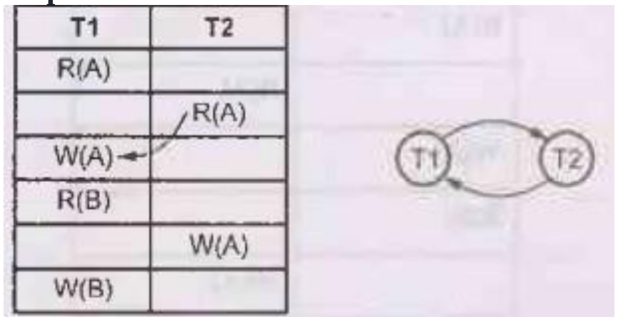
Step 1: Draw vertices for all the transactions

Step 2: We find the Read-Write pair from two different transactions reading from top to bottom. If such as pair is found then we will add the edges between corresponding directions. For instance –



T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

Step 3:

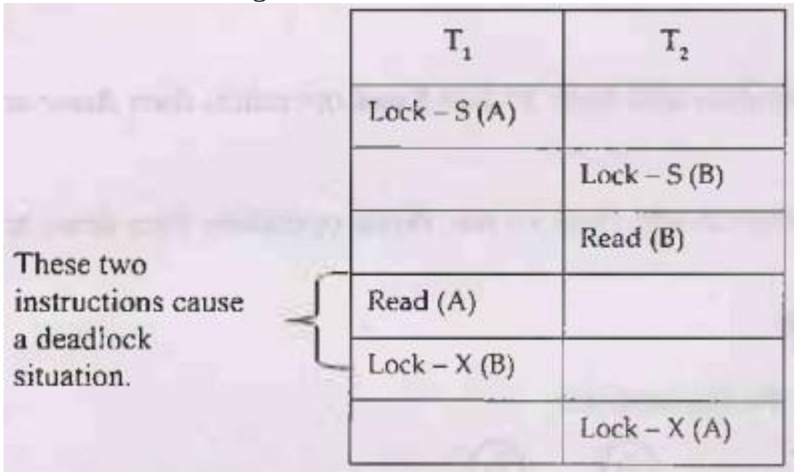


T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

As cycle is detected in the wait-for graph there is no need to further process. The deadlock is present in this transaction scenario.

Example 3.16.1 *Give an example of a scenario where two phase locking leads to deadlock.* **AU: May-09, Marks 4**

Solution: Following scenario of execution of transactions can result in deadlock.



T_1	T_2
Lock – S (A)	
	Lock – S (B)
	Read (B)
Read (A)	
Lock – X (B)	
	Lock – X (A)

These two instructions cause a deadlock situation.

In above scenario, transaction T_1, makes an exclusive lock on data item B and then transaction T_2 makes an exclusive lock on data item A. Here unless and until T_1, does not give up the lock (i.e. unlock) on B; T_2 cannot read/write it. Similarly unless and until T_2 does not give up the lock on A; T_1, cannot read or write on A.

This is a purely deadlock situation in two phase locking.

Possible Questions

1. *Outline deadlock handling mechanisms.*

2. What is deadlock? How does it occur? How transactions can be written to

(i) Avoid deadlock. (ii) Guarantee correct execution.

Illustrate with suitable example.

3. *Write short note on- Deadlock*

4. *Narrate the actions that are considered for deadlock detection and the recovery from deadlock.*

Recovery Concepts

i) This error is caused due to internal conditions such as bad input, data not found, overflow ofresource limit and so on. ii) Due to logical error the transaction can not be continued.

Part III: Recovery

Recovery Concepts

• An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure.

• The recovery scheme must also provide high availability; that means, it must minimize the time for which the database is not usable after a failure.

Failure Classification

Various types of failures are -

1. Transaction Failure: Following are two types of errors due to which the transaction gets failed.

• **Logical Error:**

i) This error is caused due to internal conditions such as bad input, data not found,overflow of resource limit and so on.

ii) Due to logical error the transaction can not be continued.

• **System Error:**

i) When the system enters in an undesired state and then the transaction can not be continued then this type of error is called as system error.

2) System Crash: The situation in which there is a hardware malfunction, or a bug in the database software or the operating system, and because of which there is a loss of the content of volatile storage, and finally the transaction processing come to a halt is called system crash.

3) Disk Failure: A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. The backup of data is maintained on the secondary disks or DVD to recover from such failure.

Storage

A DBMS stores the data on external storage because the amount of data is very huge and must persist across program executions.

The storage structure is a memory structure in the system. It has following categories -

1) Volatile :

• Volatile memory is a primary memory in the system and is placed along with the CPU.

• These memories can store only small amount of data, but they are very fast. Forexample - main memory, cache memory.

• A volatile storage cannot survive system crashes.

• That means data in these memories will be lost on failure.

2) Non Volatile :

Non volatile memory is a secondary memory and is huge in size. For example: Hard disk, Flash memory, magnetic tapes.

These memories are designed to withstand system crashes.

3) Stable :

• Information residing in stable storage is never lost.

• To implement stable storage, we replicate the information in several nonvolatile storage media (usually disk) with independent failure modes.

Stable Storage Implementation

• Stable storage is a kind of storage on which the information residing on it is never lost.

• Although stable storage is theoretically impossible to obtain it can be to approximately built by applying a technique in which data loss is almost impossible.

• That means the information is replicated in several nonvolatile storage media with independent failure modes.

• Updates must be done with care to ensure that a failure during an update to stable storage does not cause a loss of information.

Recovery with Concurrent Transactions

There are four ways for recovery with concurrent transactions:

1) Interaction with concurrency control: In this scheme recovery depends upon the concurrency control scheme which is used for the transaction. If a transaction gets failed, then rollback and undo all the updates performed by transaction.

2) Transaction Rollback: In this scheme, if the transaction gets failed, then the failed ear transaction can be rolled back with the help of log. The system scans the log backward, and with the help of log entries the system can restore the data items.

3) Checkpoints: The checkpoints are used to reduce the number of log records.

4) Restart recovery: When the system recovers from the crash it constructs two lists: Undo-list and Redo-list. The undo list consists of transactions to be undone. The redo list consists of transactions to be redone. These two lists are handled as follows

Step 1: Initially both the lists are empty.

Step 2: The system scans the log entries from backwards. It scans each entry until it finds first checkpoint record.

Shadow Copy Technique

• In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database.

• All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched.

• If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.

• The current copy of the database is identified by a pointer, called db-pointer, which is stored on disk.

• If the transaction partially commits, it is committed as follows:

  • First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk.

  • After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database.

  • The new copy then becomes the current copy of the database.

  • The old copy of the database is then deleted.

• The transaction is said to have been committed at the point where the updated db-pointer is written to disk.

• The disk system guarantees that it will update db-pointer atomically, as long as we make sure that db-pointer lies entirely in a single sector.

Where is shadow copy technique used?

• Shadow copy schemes are commonly used by text editors.

• Shadow copying can be used for small databases.

Recovery Techniques

To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.

Recovery Techniques

To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.

There are two approaches are used for recovery -

1) Log based Recovery
2) Shadow paging

Recovery based on Deferred Immediate

Before discussing the recovery algorithms (deferred and immediate update), let us see the concept of Log and REDO and UNDO operations.

Recovery based on Deferred and Immediate Update

Before discussing the recovery algorithms (deferred and immediate update), let us see the concept of Log and REDO and UNDO operations.

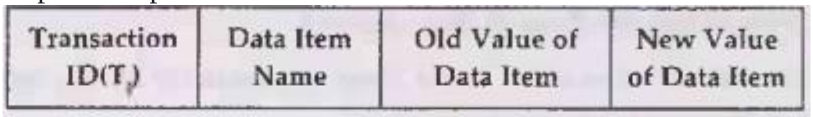
Concept of Log

• Log is the most commonly used structure for recording the modifications that is to be made in the actual database. Hence during the recovery procedure a log file is 50 maintained.

• A log record maintains four types of operations. Depending upon the type of operations there are four types of log records -

1. <start> Log record: It is represented as <T_i, Start>
2. <Update> Log record
3. <Commit> Log record: It is represented as <T_i, Commit>
4. <Abort> Log record: It is represented as <T_i, Abort>

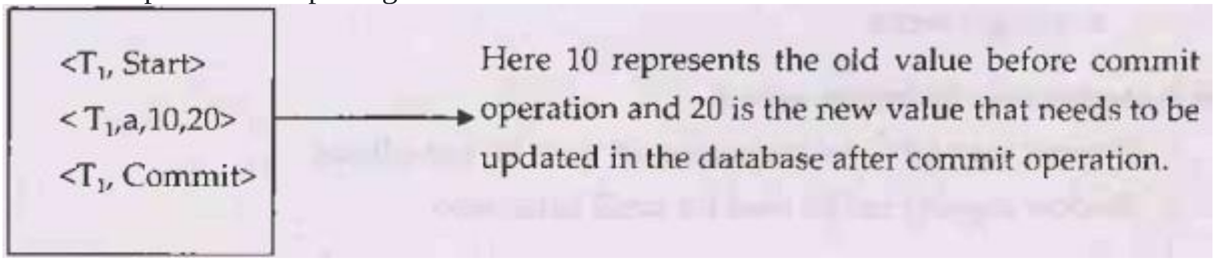
• The log contains various fields as shown in following Fig. 3.19.1. This structure is for <update> operation



Transaction ID(T_i)	Data Item Name	Old Value of Data Item	New Value of Data Item

Fig. 3.19.1

• For example: The sample log file is



<T_1, Start>
<T_1,a,10,20>
<T_1, Commit>

Here 10 represents the old value before commit operation and 20 is the new value that needs to be updated in the database after commit operation.

Here 10 represents the old value before commit operation and 20 is the new value that needs to be updated in the database after commit operation.

• The log must be maintained on the stable storage and the entries in the log file aremaintained before actually updating the physical database.

• There are two approaches used for log based recovery technique - Deferred Database Modification and Immediate Database Modification.

REDO and UNDO Operation

During transaction execution, the updates are recorded only in the log and in the cache buffers. After the transaction reaches its commit point and the log is force written to disk and the updates are recorded in the database.

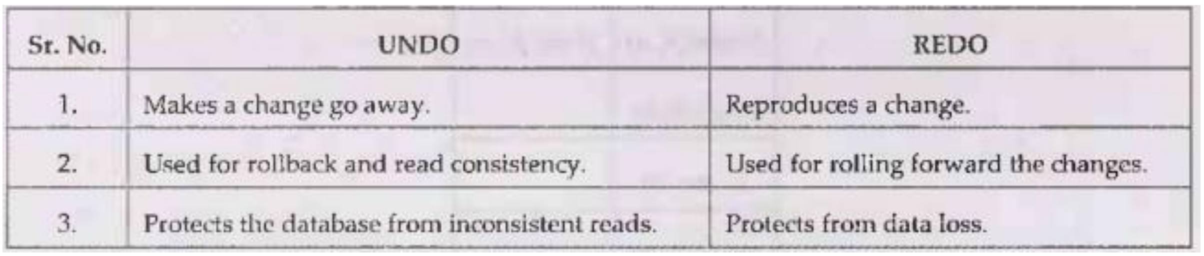
In order to maintain the atomicity of transaction, the operations can be redone or undone.

UNDO: This is an operation in which we restore all the old values (BFIM - BeFore Modification Image) onto the disk. This is called roll-back operation.

REDO: This is an operation in which all the modified values(AFIM - After Modification Image) are restored onto the disk. This is called roll-forward operation.

These operations are recorded in the log as they happen.

Difference between UNDO and REDO



Sr. No.	UNDO	REDO
1.	Makes a change go away.	Reproduces a change.
2.	Used for rollback and read consistency.	Used for rolling forward the changes.
3.	Protects the database from inconsistent reads.	Protects from data loss.

Write Ahead Logging Rule

• Before a block of data in main memory is output to the database, all log records pertaining to data in that block must have been output to stable storage. This rule is called the Write-Ahead Logging (WAL)

• This rule is necessary because - In the event of a crash or ROLLBACK, the original content contained in the rollback journal is played back into the database file to revert the database file to its original state.

Deferred Database Modification

• In this technique, the database is not updated immediately.

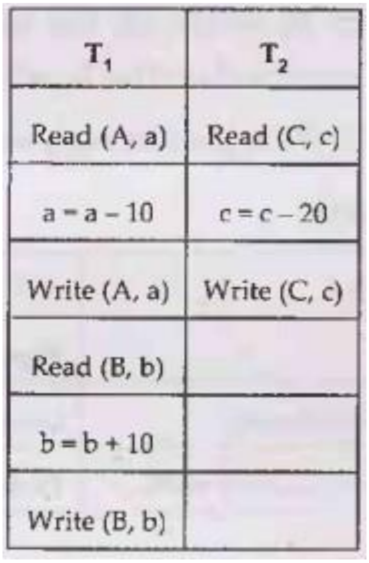
• Only log file is updated on each transaction.

• When the transaction reaches to its commit point, then only the database is physically updated from the log file.

• In this technique, if a transaction fails before reaching to its commit point, it will not have changed database anyway. Hence there is no need for the UNDO operation. The REDO operation is required to record the operations from log file to physical database. Hence deferred database modification technique is also called as NO UNDO/REDO algorithm.

• **For example:**

Consider two transactions T_1 and T_2 as follows:



T_1	T_2
Read (A, a)	Read (C, c)
a = a – 10	c = c – 20
Write (A, a)	Write (C, c)
Read (B, b)	
b = b + 10	
Write (B, b)	

If T_1 and T_2 are executed serially with initial values of A = 100, B = 200 and C = 300, then the state of log and database if crash occurs

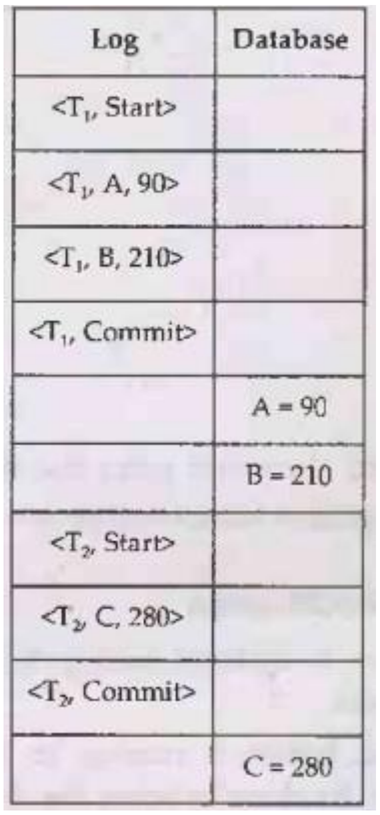
a) Just after write (B, b)

b) Just after write (C, c)

c) Just after <T_2, commit>

The result of above 3 scenarios is as follows:

Initially the log and database will be



Log	Database
$<T_1$, Start>	
$<T_1$, A, 90>	
$<T_1$, B, 210>	
$<T_1$, Commit>	
	A = 90
	B = 210
$<T_2$, Start>	
$<T_2$, C, 280>	
$<T_2$, Commit>	
	C = 280

a) Just after write (B, b)

Just after write operation, no commit record appears in log. Hence no write operation is performed on database. So database retains only old values. Hence A = 100 and B = 200 respectively.

Thus the system comes back to original position and no redo operation take place. The incomplete transaction of T_1 can be deleted from log.

b) Just after write (C, c)

The state of log records is as follows

Note that crash occurs before T_2 commits. At this point T_1 is completed successfully, so new values of A and B are written from log to database. But as T_2 is not committed, there is no redo (T_2) and the incomplete transaction T_2 can be deleted from log.

The redo (T_1) is done as < T_1, commit> gets executed. Therefore A = 90, B = 210 and C=300 are the values for database.

c) Just after < T2, commit>

The log records are as follows:

<T_1, Start>

<T_1, A, 90>

<T_1, B, 210>

<T_1, Commit>

<T_2, Start>

<T2, 6, 280>

<T_2, Commit>

← Crash occurs here

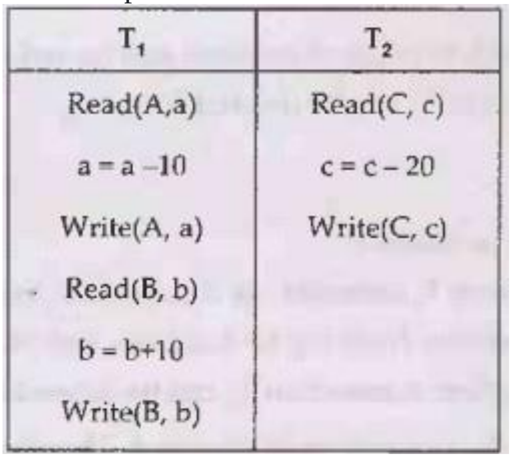
Clearly both T_1 and T_2 reached at commit point and then crash occurs. So both redo (T_1) and redo (T_2) are done and updated values will be A = 90, B = 210, C = 280.

Immediate Database Modification

In this technique, the database is updated during the execution of transaction even before it reaches to its commit point.

If the transaction gets failed before it reaches to its commit point, then the a ROLLBACK Operation needs to be done to bring the database to its earlier consistent state. That means the effect of operations need to be undone on the database. For that purpose both Redo and Undo operations are both required during the recovery. This technique is known as UNDO/REDO technique.

For example: Consider two transaction T_1 and T_2 as follows:



T_1	T_2
Read(A,a)	Read(C, c)
a = a −10	c = c − 20
Write(A, a)	Write(C, c)
Read(B, b)	
b = b+10	
Write(B, b)	

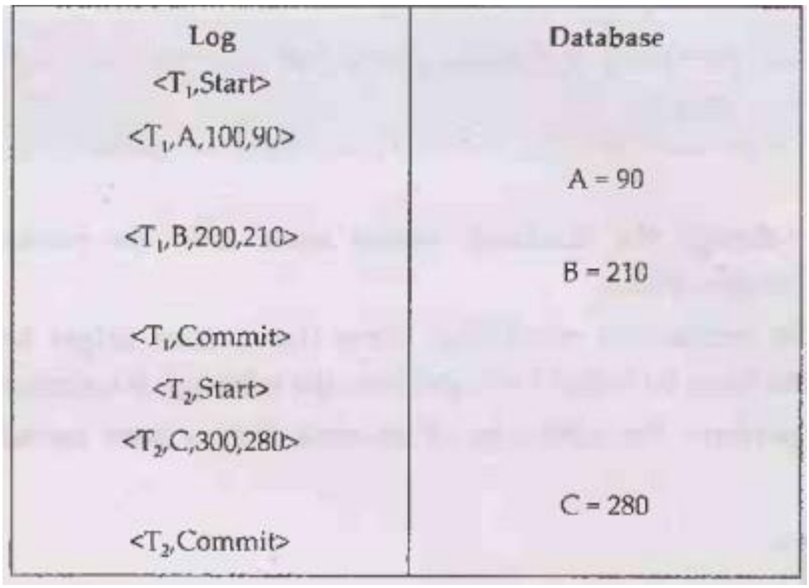
Here T_1 and T_2 are executed serially. Initially A = 100, B = 200 and C = 300

If the crash occurs after

i) Just after Write(B, b) ii) Just after Write(C, c) iii) Just after <T,,Commit>

Then using the immediate Database modification approach the result of above three scenarios can be elaborated as follows:

The contents of log and database is as follows:



Log	Database
$<T_1,Start>$	
$<T_1,A,100,90>$	
	A = 90
$<T_1,B,200,210>$	
	B = 210
$<T_1,Commit>$	
$<T_2,Start>$	
$<T_2,C,300,280>$	
	C = 280
$<T_2,Commit>$	

The recovery scheme uses two recovery techniques -

i) UNDO (T_i): The transaction T, needs to be undone if the log contains <T_i,Start> but does not contain <T_i,Commit>. In this phase, it restores the values of all data items updated by T, to the old values.

ii) REDO (T_i): The transaction T_i needs to be redone if the log contains both <T_i,Start> and <T_i,Commit>. In this phase, the data item values are set to the new values as per thetransaction. After a failure has occurred log record is consulted to determine which transaction need to be redone.

a) Just after Write (B, b): When system comes back from this crash, it sees that there is <T_1, Start> but no <T_1, Commit>. Hence T_1 must be undone. That means old values of A and B are restored. Thus old values of A and B are taken from log and both the transaction T_1 and T_2 are re-executed.

b) Just after Write (C, c): Here both the redo and undo operations will occur.

c) Undo: When system comes back from this crash, it sees that there is <T_2, Start> but no <T_2, Commit>. Hence T_2 must be undone. That means old values of C is restored.

Thus old value of C is taken from log and the transaction T_2 is re-executed.

c) Redo: The transaction T, must be done as log contains both the <T_1,Start> and <T_1,Commit>

So A = 90, B = 210 and C = 300

d) Just after <T_2, Commit>: When the system comes back from this crash, it sees that there are two transaction T_1 and T_2 with both start and commit points. That means T_1 and T_2 need to be redone. So A = 90, B = 210 and C = 280

Example 3.19.1 *Suppose there is a database system that never fails. Is a recovery manager require for this system? Why?*

Solution:

1) Yes. Even-though the database system never fails, the recovery manager is required for this system.

2) During the transaction processing some transactions might be aborted. Such transactions must be rolled back and then the schedule is continued further.

3) Thus to perform the rollbacks of aborted transactions recovery manager is required.

Possible Question

1. Explain deferred and immediate modification versions of the log based recovery scheme.

Shadow Paging

Shadow paging is a recovery scheme in which database is considered to be made up of number of fixed size disk pages.

Shadow Paging

• Shadow paging is a recovery scheme in which database is considered to be made up of number of fixed size disk pages.

• A directory or a page table is constructed with n number of pages where each ith page points to the ith database page on the disk.

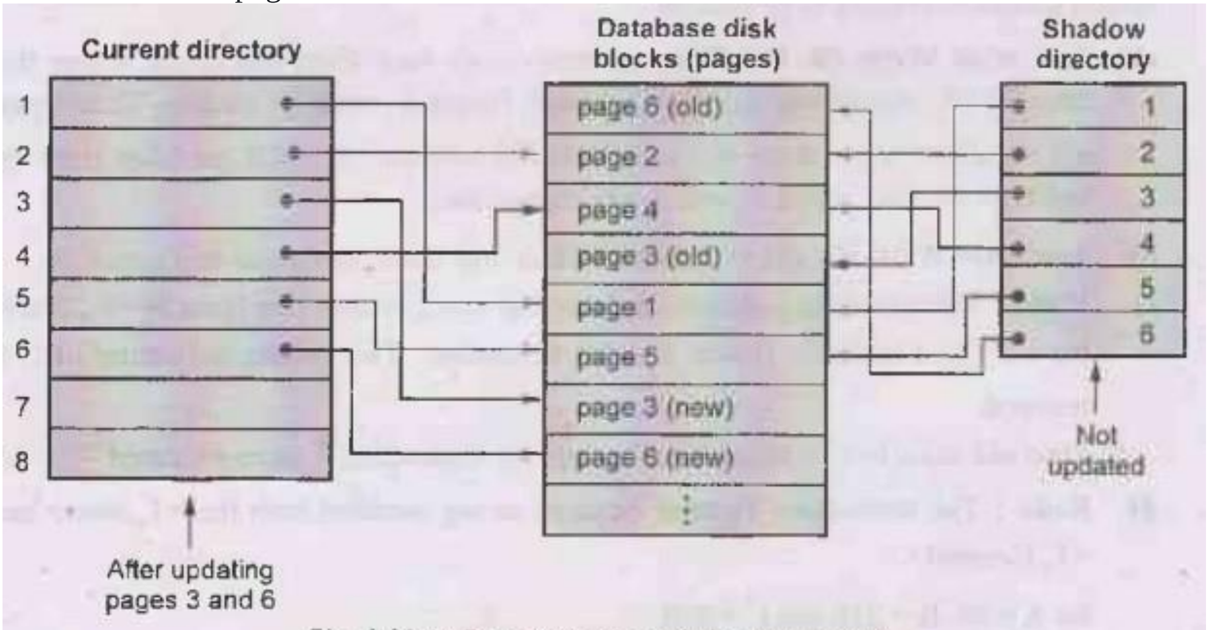


Fig. 3.20.1 : Demonstration of shadow paging

• The directory can be kept in the main memory.

• When a transaction begins executing, the current directory-whose entries point to the most recent or current database pages on disk-is copied into a another directory called shadow directory.

• The shadow directory is then saved on disk while the current directory is used by the transaction.

• During the execution of transaction, the shadow directory is never modified.

• When a write operation is to be performed then the new copy of modified database page is created but the old copy of database page is never overwritten. This newly created database page is written somewhere else.

• The current directory will point to newly modified web page and the shadow page directory will point to the old web page entries of database disk.

• When the failure occurs then the modified database pages and current directory is discarded.

• The state of database before the failure occurs is now available through the shadow directory and this state can be recovered using shadow directory pages.

• This technique does not require any UNDO/REDO operation.

ARIES Algorithm

ARIES is a recovery algorithm. It stands for Algorithm for Recovery and Isolation Exploiting Semantics.

ARIES Algorithm

• ARIES is a recovery algorithm.

• It stands for Algorithm for Recovery and Isolation Exploiting Semantics.

• It is based on Write Ahead Log (WAL) protocol.

• When database crashes during some transaction processing, we have the logs that got saved to the disk.

• ARIES has 3 phases that occur in the following order -

1) Analysis:

• Scan the log from start to reconstruct the transaction and dirty page table. Dirtypages contain data that has been changed but not yet written to disk.

• The active transactions which were present at the time of crash are identified.

• During analysis phase the log is scanned forward from the checkpoint record to construct snapshot of what system looks like at the time of crash.

2) Redo :

• This phase is started only after completion of analysis phase.

• The log is read forward and each update is redone.

3) Undo :

• This phase is started after redo phase.

• The log is scanned backward and updates to corresponding active transactions are undone.

Advantages:

1) It is simple and flexible.

2) It supports concurrency control protocol.
3) Independent recovery of every page.

Two marks Questions with Answers

A transaction can be defined as a group of tasks that form a single logical unit.

Two Marks Questions with Answers

Q.1 What is a transaction?

Ans: A transaction can be defined as a group of tasks that form a single logical unit.

Q.2 What does time to commit mean?

Ans:

• The COMMIT command is used to save permanently any transaction to database.

• When we perform, Read or Write operations to the database then those changes can be undone by rollback operations. To make these changes permanent, we should make use of commit

Q.3 What are the various properties of transaction that the database system maintains to ensure integrity of data.

OR

Q.4 What are ACID properties?

Ans: In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database. These are

(1) Atomicity (2) Consistency (3) Isolation (4) Durability

Q.5 Give the meaning of the expression ACID transaction.

Ans: The expression ACID transaction represents the transaction that follows the ACID Properties.

Q.6 State the atomicity property of a transaction.

Ans: This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.

No transaction in the database is left half completed.

Q.7 What is meant by concurrency control ?

Ans:

A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called concurrency control mechanism.

Q.8 State the need for concurrency control. AU: OR

Q.9 Why is it necessary to have control of concurrent execution of transactions? How is it made possible?

Ans: Following are the purposes of concurrency control-

• To ensure isolation

• To resolve read-write or write-write conflicts

• To preserve consistency of database

Q.10 List commonly used concurrency control techniques.

Ans: The commonly used concurrency control techniques are -

i) Lock

ii) Timestamp

iii) Snapshot Isolation

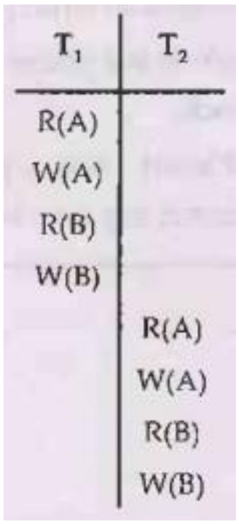
Q.11 What is meant by serializability? How it is tested?

Ans.: Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.

It is tested using precedence graph technique.

Q.12 What is serializable schedule?

Ans.: The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions T_1 and T_2



T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T_1 on data items A and then B executes and then intransaction T_2 all the operations on data items A and B execute. The R stands for Read operation and W stands for write operation.

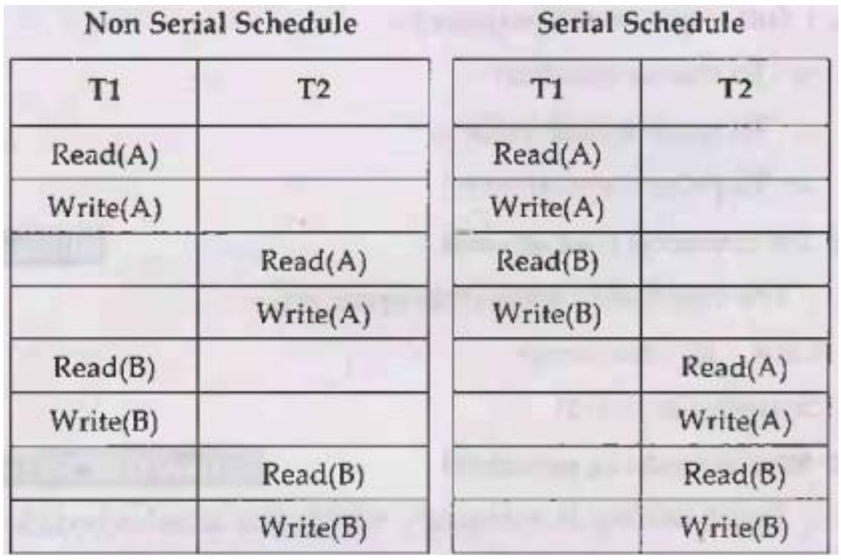
Q.13 When are two schedules conflict equivalent?

Ans.: Two schedules are conflict equivalent if :

• They contain the same set of the transaction.

• every pair of conflicting actions is ordered the same way.

For example -



Non Serial Schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S_2 is a serial schedule because, in this, all operations of T_1 are performed before starting any operation of T_2. Schedule S_1 can be transformed into a serial schedule by swapping non-conflicting operations of S_1.

Hence both of the above the schedules are conflict equivalent.

Q.14 Define two phase locking.

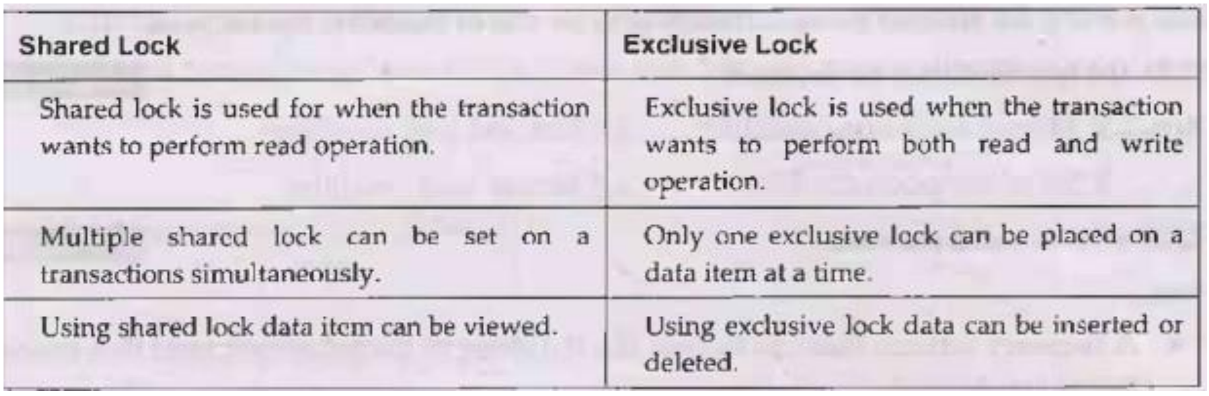
Ans.: The two phase locking is a protocol in which there are two phases:

i) Growing Phase (Locking Phase): It is a phase in which the transaction may obtain locks but does not release any lock.

ii) Shrinking Phase (Unlocking Phase): It is a phase in which the transaction may release the locks but does not obtain any new lock.

Q.15 What is the difference between shared lock and exclusive lock?

Ans.:



Shared Lock	Exclusive Lock
Shared lock is used for when the transaction wants to perform read operation.	Exclusive lock is used when the transaction wants to perform both read and write operation.
Multiple shared lock can be set on a transactions simultaneously.	Only one exclusive lock can be placed on a data item at a time.
Using shared lock data item can be viewed.	Using exclusive lock data can be inserted or deleted.

Q.16 What type of lock is needed for insert and delete operations.

Ans.: The exclusive lock is needed to insert and delete operations.

Q.17 What benefit does strict two-phase locking provide? What disadvantages result?

Benefits:

1. This ensure that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits and preventing other transaction from reading that data.

2. This protocol solves dirty read problem.

Disadvantage:

1. Concurrency is reduced.

Q.18 What is rigorous two phase locking protocol ?

Ans.: This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits.

Q.19 Differentiate strict two phase locking and rigourous two phase locking protocol.

• In Strict two phase locking protocol all the exclusive mode locks be held until the transaction commits.

• The rigourous two phase locking protocol is stricter than strict two phase locking protocol. Here all locks are to be held until the transaction commits

Q.20 Define deadlock.

Ans.: Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

Q.21 List four conditions for deadlock.

1. Mutual exclusion condition
2. Hold and wait condition
3. No preemption condition
4. Circular wait condition

Q.22 Why is recovery needed?

• A recovery scheme that can restore the database to the consistent state that existed before the failure.

• Due to recovery mechanism, there is high availability of database to its users.

Q.23 What are states of transaction?

Ans.: Various states of transaction are - (1) Active, (2) Partially Committed (3) Failed (4) Aborted (5) Committed.

Q.24 What is meant by log based recovery?

Ans.: Log is a most commonly used data structure for recording the modifications that can be made to actual database.

Log based recovery is a technique in which a log of each transaction is maintained in some stable storage so that if failure occurs then it can be recovered from there.

Q.25 List the responsibilities of a DBMS has whenever a transaction is submitted to the system for execution.

Ans.: The system is responsible for making sure that - (1) Either all the operations in the transaction are completed successfully and effect is recorded permanently in the database. (2) The transaction, has no effect whatsoever on the database or on the database or on any other transaction.

Q.26 Brief any two violations that may occur if a transaction executes a lower isolation level than serializable.

(1) For non-repeatable Read the phantom read is allowed.
(2) For read committed non-repeatable reads and phantom reads are allowed.