

UNIT - IVTransaction ProcessingTransaction :-

- * Transaction is a collection of operations that form a single logical unit of work.
- * A transaction can access & update various data items. Each transaction must succeed or fail as a complete unit i.e., it can not remain in an intermediate state.
- * e.g., A transaction is a transfer of money between two bank accounts.
This transaction consists of debiting the source account, crediting the target account & recording the updated information.

Transaction Properties :

- * The main properties of a transaction are called as ACID (Atomicity, Consistency, Isolation, Durability) properties.
- * A database must contain these properties in order to ensure accuracy, completeness & data integrity.

① Atomicity :- \rightarrow This property says that at the end of each transaction, either no changes have been done to the database or the database has been changed in a consistent manner.

\rightarrow To make the changes permanent, COMMIT statement is issued and to abort the changes, ROLLBACK statement is issued.

e.g., if we transfer Rs. 500 from a/c A to B, if the transaction is completed, the amount of 500 Rs must be deducted from A's a/c & same must be added to B's a/c. But if the transaction is aborted then neither of the balance will be changed.

② Consistency :- \rightarrow Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

③ Isolation :- \rightarrow Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j , it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i finished.

\rightarrow Thus each system is unaware of the other transactions executing concurrently in the system.

- ②
- ④ Durability :- Once a transaction completes successfully, then the changes it has made to the database are permanent.

Transaction Model

* There are five states through which a transaction passes during its life time.

* These states are :-

- ① Active :- A transaction under execution is said to be in active state.
- ② Partially Committed :- A transaction executing its last statement is said to be in partially committed state.
- ③ Failed :- A transaction is said to be in failed state if its normal execution can't proceed.
- ④ Committed :- A transaction is said to be in committed state if its execution is successfully completed.
- ⑤ Aborted :- A transaction is said to be in aborted state if it is rolled back because of some failure.

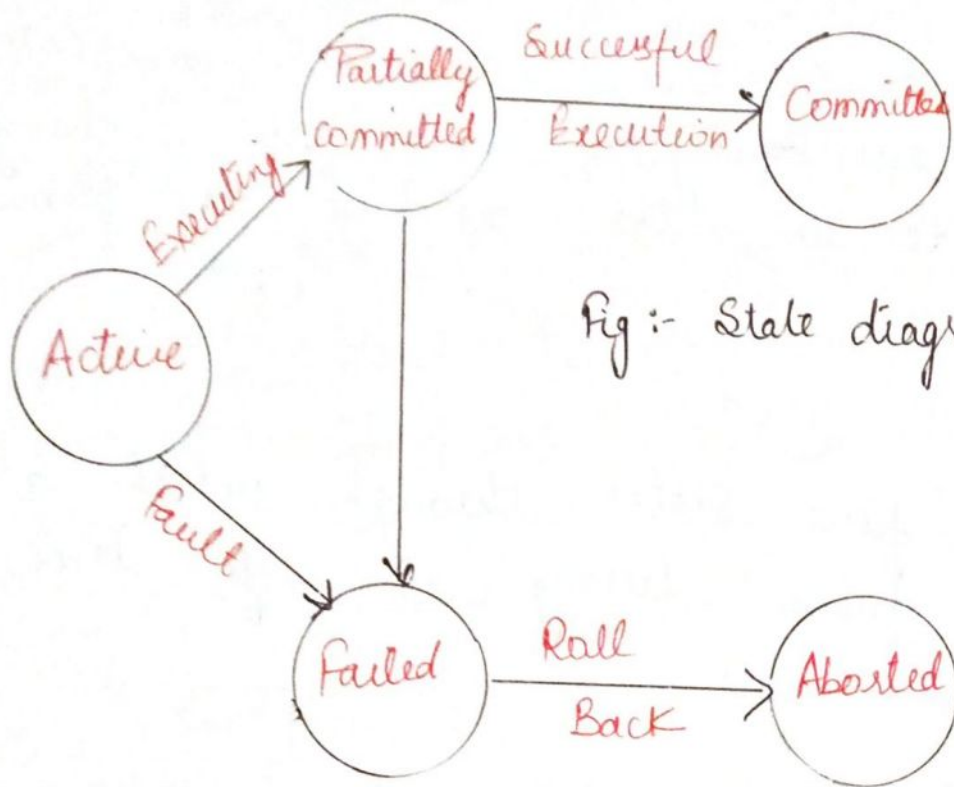


Fig:- State diagram of Transaction

Schedule :- A schedule is an order in which the transactions are executed in the system. There are two types of schedules :-

1. Serial Schedule
2. Concurrent Schedule

① Serial Schedule :-

* A Serial schedule is an order in which the transactions are ~~scheduled~~ executed serially i.e., One transaction at a time. When a transaction is running then, no other transaction can start its execution unless until the already running transaction is completed.

* For 'n' number of transactions, there are $n!$ valid serial schedules. Serial schedules always result in consistent database.

- * → For example, consider account A with balance 2000 and account B with balance 1000.
 - Consider two transactions T_1 & T_2 .
 - T_1 transfers Rs. 100 from account A to B while T_2 transfers Rs. 500 from A to B.
- Then $\langle T_1, T_2 \rangle$ and $\langle T_2, T_1 \rangle$ are two valid schedules as;

Serial Schedule $\langle T_1, T_2 \rangle$:

T_1	T_2
Read (A); $A = A - 100$; Write (A); Read (B); $B = B + 100$; Write (B);	Read (A); $A = A - 500$; Write (A); Read (B); $B = B + 500$; Write (B);

- * Schedule 1 executes T_1 first & then T_2 . after completion of schedule 1, the balance in A & B are 1400 & 1600 respectively. Thus this schedule preserves A+B and hence the database is consistent.

{ Note : Similarly we can schedule another schedule 2, in which T_2 executes first & then T_1 executes. }

Concurrent Schedule :-

- * A concurrent schedule is an order in which several transactions are executed concurrently.
- * Some of concurrent schedules result in inconsistent database while some of them may result in consistent database.
- * Let us see two concurrent schedules, out of which schedule 1 results in consistent database.

Concurrent Schedule 1

Consistent database

T ₁	T ₂
Read(A); A = A - 100; write(A);	Read(A); A = A - 500; write(A);
Read(B); B = B + 100; write(B);	Read(B); B = B + 500; write(B);

A = 2000

A = 1900

A = 1900

A = 1400 *

B = 1000

B = 1100

B = 1100

B = 1600 *

Concurrent Schedule 2

Inconsistent database

T ₁	T ₂
Read(A); A = A - 100;	Read(A); A = A - 500; write(A);
write(A); Read(B); B = B + 100; write(B);	Read(B); B = B + 500; write(B);

A = 2000

A = 1900

A = 2000

A = 1500 *

B = 1000

B = 1100

B = 1600 *

Serializability of Schedules :-

- * As we know that serial execution of the transactions always result in the consistent database whereas concurrent execution may result in the inconsistent database.
- * Thus database must control the concurrent execution of transactions to ensure that the database remain in the consistent state.
- * For this, we need to check for the Serializability of concurrent schedules.
- * A concurrent schedule is serializable or has the serializability property, if its outcome is equal to the outcome of its equivalent serial schedule.
- * Transactions are normally executed concurrently because this is the most efficient way.
- * Serializability is the major correctness criterion for concurrent transaction execution.

There are two types of Serializability:

1. Conflict Serializability
2. View Serializability

① Conflict Serializability :

- ♥ A schedule is said to be conflict serializable if it is conflict-equivalent to serial schedule.
- ♥ for conflict-equivalent schedules, we need to check for the conflicts between two consecutive operations of two different transactions in a schedule.
- ♥ operation upon data can be read or write.
- ♥ There are two possibilities :

① If two consecutive operations are on different data items, then they don't need conflict i.e., their order of execution does not matter and we can swap them without affecting their result.

② → If two consecutive operations are on same data item, then they can conflict i.e., their order of execution matters & we can't swap them.

→ Consider a schedule S in which there are two consecutive operations O_i & O_j of two transactions T_i & T_j and both refers to the same data item A . Then there may be following four cases :

(i) $O_i = \text{read}(A)$ and $O_j = \text{read}(A)$. Then their order doesn't matter because the same value of A is read by T_i & T_j regardless of their order.

(ii) $O_i = \text{read}(A)$ and $O_j = \text{write}(A)$. Then their order matters. If O_i comes before O_j , then O_i doesn't read the value of A written by O_j . If O_j comes before O_i then O_i reads the value of A that is written by O_j .

(iii) $O_i = \text{write}(A)$ and $O_j = \text{read}(A)$. Then their order matters. If O_i comes before O_j then O_j reads the value of A written by O_i . If O_j comes before O_i then O_j doesn't read the value of A that is written by O_i .

(iv) $O_i = \text{write}(A)$ and $O_j = \text{write}(A)$. Then their order matters. If O_i comes after O_j , then value of A written by O_i is stored in the database. If O_j comes after O_i then the value of A written by O_j is stored in database.

Thus we can say that two operations are conflicting, if they are of different transactions, refer to the same data item and at least one of them is write operation.

- * Each such pair of conflicting operations has a conflict type, It is either read-write, or write-read or a write-write conflict.
- * The transaction of the second operation in the pair is said to be in conflict with the transaction of the first operation.
- * The order of non-conflicting operations doesn't matter hence, we can swap them, But the order of conflicting operations matters, hence, we can't swap them.
- * If a schedule can be transformed to any serial schedule without changing orders of conflicting operations but changing orders of non-conflicting operations, then the outcome of both schedules is the same & the schedule is said to be conflict-serializable.

Thus two operations are said to be in ^⑥ conflict if they satisfy all the following three conditions :-

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

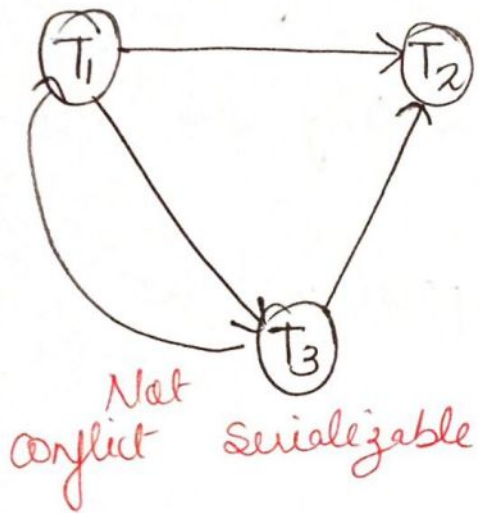
Examples :-

- Q 1 :- $W(A)$ of T_1 & $R(A)$ of T_2 - conflicting
- Q 2 :- $W(A)$ of T_1 & $W(A)$ of T_2 - conflicting
- Q 3 :- $W(A)$ of T_1 & $W(B)$ of T_2 - Non-conflicting
- Q 4 :- $R(A)$ of T_1 & $R(A)$ of T_2 - Non-conflicting
- Q 5 :- $W(A)$ of T_1 & $R(A)$ of T_2 - Non-conflicting

Conflict Equivalent Schedules :

- * Two schedules are said to be conflict equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Examples :-

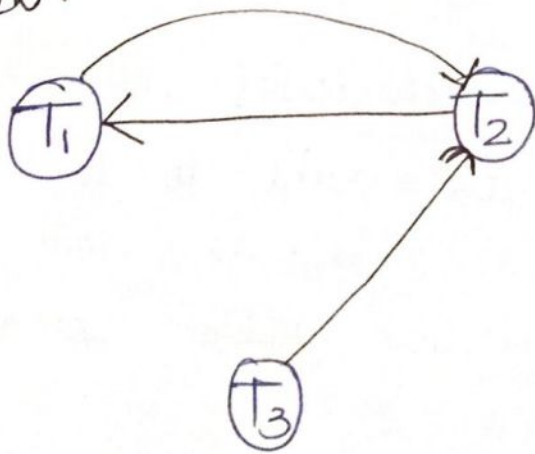


T_1	T_2	T_3
$R(A)$ ①	R	$R(C)$ ②
	$R(B)$ ④	$W(C)$ ③
$R(B)$ ⑤		
	$W(B)$ ⑥	
	$W(C)$ ⑧	$W(A)$ ⑦
$W(A)$ ⑨		

Q 1: check whether the schedule is conflict Serializable or not.

$S: R_1(A), R_2(A), R_1(B), R_2(B), R_3(B), W_1(A),$
 $T_2 \rightarrow T_1 \quad R_2(A) \quad W_2(B)$

Precedence Graph :-

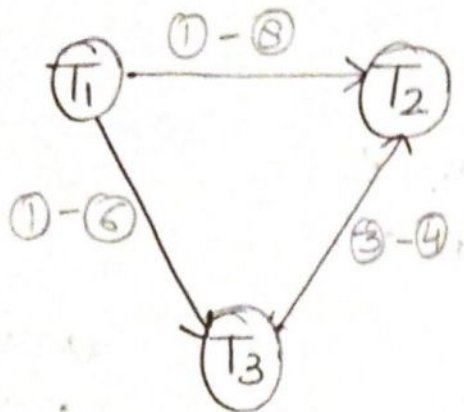


Mole %

as \exists a cycle in the graph. $w_1(A)$
The given schedule is not
conflict serializable.

$$\begin{array}{ccc} T_1 & T_2 & T_3 \\ R_1(A) & & \\ & R_2(A) & \\ R_1(B) & & \\ & R_2(B) & \\ & & R_3(B) \\ W_1(A) & & \\ & W_2(B) & \end{array}$$

Q2.



No cycle - it is
conflict Serializable
in order

① ②
No incoming Edge

②
No outgoing Edge

S			
T ₁	T ₂	T ₃	
① R(X)	② R(Y)	③ R(Y)	
	④ W(Y)		
⑤ W(X)		⑥ W(X)	
	⑦ R(X)		
	⑧ W(X)		

Q3.

①

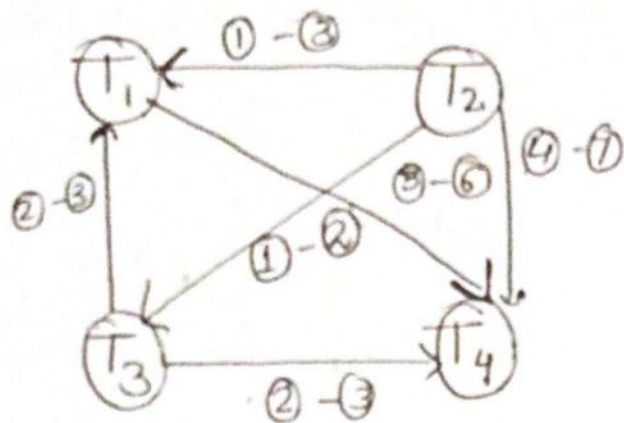
②

③

T ₁	T ₂	T ₃
R(a)	R(b)	R(c)
		W(c)
	W(b)	
W(a)		

* It is conflict serializable
No edge 3! serial
schedules are possible

Q4.

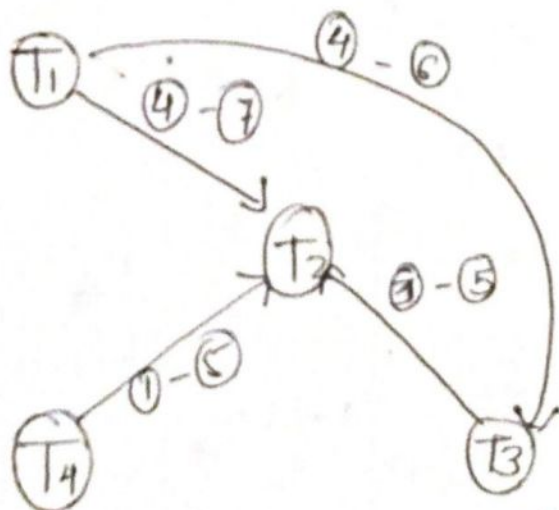


T_1	T_2	T_3	T_4
	① R(X)		
		② W(X)	
	③ W(X)		
	④ W(Y)		
	⑤ R(Z)		
			⑥ R(X)
			⑦ R(Y)

* as there is no cycle in the graph,

therefore the given schedule is conflict serializable

Q5.



T_1	T_2	T_3	T_4
			① R(A)
	② R(A)		
		③ R(A)	
	④ W(CB)		
	⑤ W(A)		
	⑦ W(B)		
		⑥ R(B)	

* No cycle, conflict serializable.

Possible Serial Schedules:-

1. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

2. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

3. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

② 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.

* consider two schedules S_1 & S_2 each consisting of two transactions T_1 & T_2 .

* S_1 & S_2 are called view equivalent if the following three conditions hold :-

Condition # 1 :- For each data item X , if the transaction T_i reads X from the database initially in schedule S_1 , then in S_2 also, T_i must perform the initial read of X from the database.

Rule :- Initial Readers must be same for all data items

Condition # 2 :- If a transaction T_i reads a data item that has been updated by the transaction T_j in schedule S_1 , then in S_2 also, transaction T_i must read the same data item that has been updated by T_j .

Rule :- Write - read sequence must be same.

Condition #3 :- For each data item X , if X has been updated at last by transaction T_i in schedule S_1 , then in schedule S_2 also, X must be updated at last by transaction T_i .

Rule :- Final writers must be same for all the data items.

Testing for View Serializability :-

Method - 1 :- check whether it is conflict serializable.

- * If a given schedule is conflict serializable, then it is definitely view serializable.
- * otherwise it may or may not view serializable. (check for other methods)

Method - 2 :-

check if there exists any blind write operation.

Note :- Writing without reading is called as a blind write.

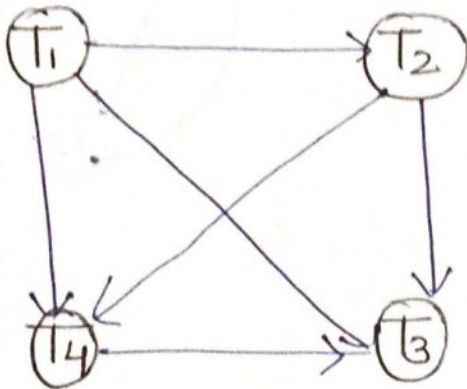
- * No blind write means not a view serializable schedule.

Practice Problems on View Serializability: (9)

Problem # 1 :-

check for conflict serializability

Step-1

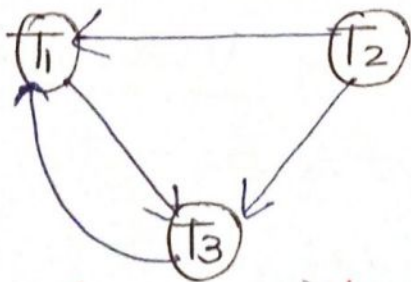


T ₁	T ₂	T ₃	T ₄
R(A)			
	R(A)		
		R(A)	
			R(A)
W(B)			
	W(B)		
		W(B)	
			W(B)

* No cycle, conflict serializable. Thus it must be view serializable.

Problem # 2 :-

Step-1: check for conflict serializability



T ₁	T ₂	T ₃
R(A)		
	R(A)	
		W(A)
W(A)		

* As there exists a cycle, it is not conflict serializable.

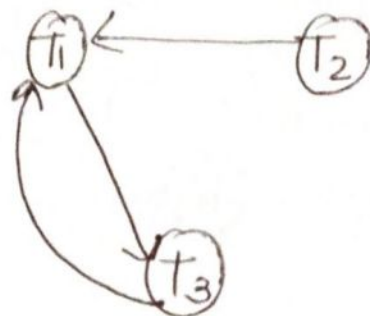
Step-2: check for blind write.

* there exists a blind write in W₃(A)

* Thus it may or may not be view serializable.

Step-3 : Drawing a dependency Graph :

* T_1 firstly reads A and T_3 firstly update A.



Recoverable Schedules :

* If in a schedule,

→ A transaction performs a dirty read operation from an uncommitted transaction

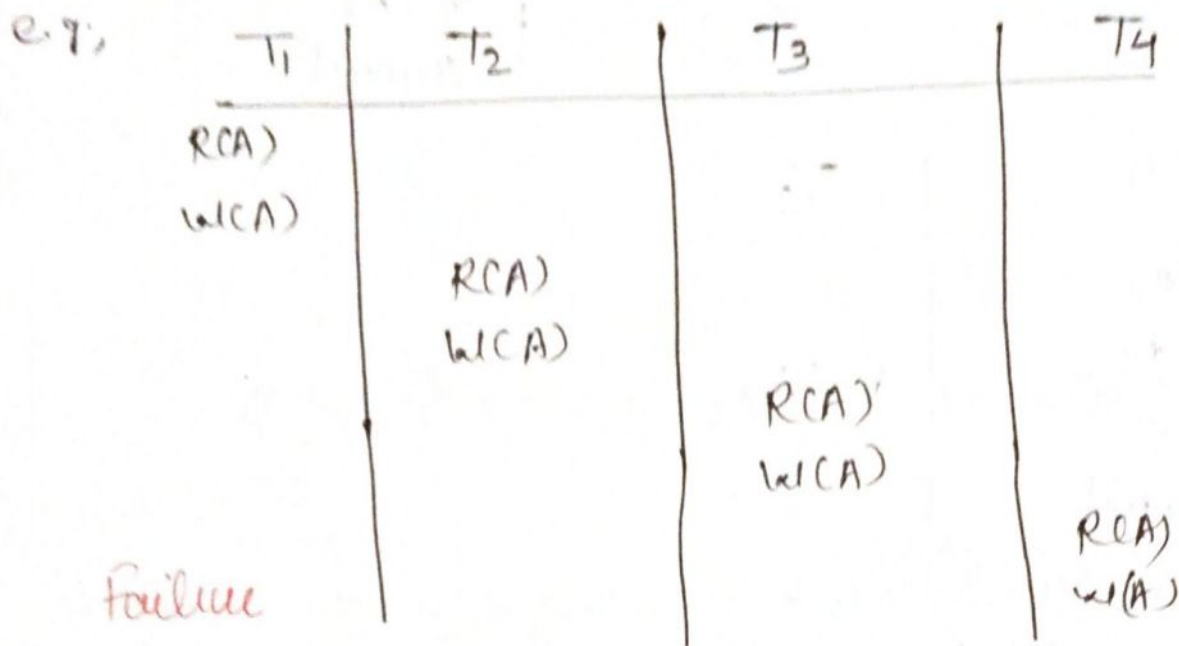
→ And its commit operation is delayed till the uncommitted transaction either commits or roll back then such a schedule is called as a Recoverable Schedule.

Types of Recoverable Schedule :-

- Cascading Schedules
- Cascadeless "
- Strict "

1. Cascading Schedules :-

* If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called cascading schedule.



* Here T₂ depends upon T₁, T₃ depends upon T₂ & so on.

2. Cascadeless Schedules :-

* If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a cascadeless schedule.

e.g. 1)

T ₁	T ₂	T ₃
R(A) W(A) commit	R(A) W(A) commit	R(A) W(A) commit

e.g. 2

T ₁	T ₂
R(A) W(A) commit	W(A) // uncommitted write

* Cascadeless Schedule allows only committed read operations. However it allows uncommitted write operations