TRANSACTION

- Why we study transaction?
  - According to general computation principle (operating system) we may have partially executed program, as the level of atomicity is instruction i.e. either an instruction is executed completely or not
  - But in DBMS view, user perform a logical work(operation) which is always atomic in nature i.e. either operation is execute or not executed, there is no concept like partial execution. For example, Transaction T_1 which transfer 100 units from account A to B

T_1
Read(A)
A = A-100
Write(A)
Read(B)
B = B+100
Write(B)

  - In this transaction if a failure occurs after Read(B) then the final statue of the system will be inconsistent as 100 units are debited from account A but not credited in account B, this will generate inconsistency.
  - Here for 'consistency' before (A + B) == after (A + B)"

What is Transaction

- To remove this partial execution problem, we increase the level of atomicity and bundle all the instruction of a logical operation into a unit called transaction.
- So formally 'A transaction is a Set of logically related instructions to perform a logical unit of work'.
- As here we are only concerned with DBMS so we well only two basic operation on database
  - **READ (X)** - Accessing the database item x from disk (where database stored data) to memory variable also name as X.
  - **WRITE (X)** - Writing the data item from memory variable X to disk.

Q A transaction can include following basic database access operations: **(NET-JUNE-2011)**

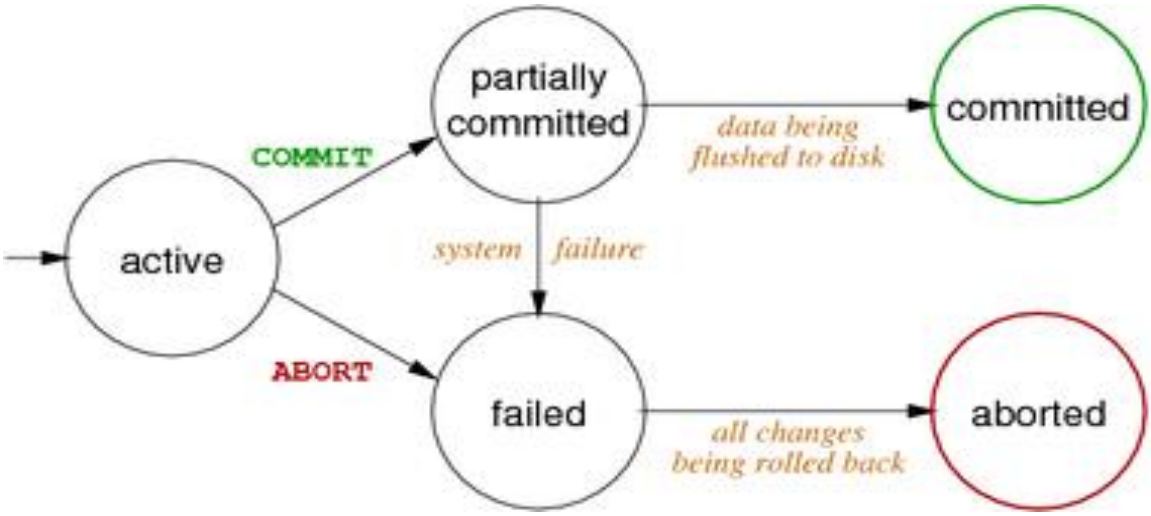
(A) Read_item(X)
(B) Write_item(X)
(C) Both (A) and (B)
(D) None of these

Ans: c

Desirable Properties of Transaction

- Now as the smallest unit which have atomicity in DBMS view is transaction, so if want that our data should be consistent then instead of concentrating on data base, we must concentrate on the transaction for our data to be consistent.
- Transactions should possess several properties, often called the **ACID** properties; to provide integrity and consistency of the data in the database. The following are the ACID properties:
    - **Atomicity -** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all. It is the responsibility of ***recovery control manager / transaction control manager of DBMS*** to ensure atomicity
    - **Consistency** - A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another. The definition of consistency may change from one system to another. The preservation of consistency of database is the responsibility of programmers(users) or the DBMS modules that enforces integrity constraints.
    - **Isolation -** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently. The isolation property of database is the responsibility of ***concurrency control manager of database***.
    - **Durability -** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure. It is the responsibility of ***recovery control manager of DBMS.***

Transaction states



- **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.
- **PARTIALLY COMMITTED** - After the final statement of a transaction has been executed, the state of transaction is partially committed as it is still possible that it may have to be aborted (due to any failure) since the actual output may still be temporarily residing in main memory and not to disk.
- **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back
- **ABORTED** - A transaction is said to be in aborted state when the when the transaction has been rolled back and the database has been restored to its state prior to the start of execution.
- **COMMITTED** - A transaction enters committed state after successful completion of a transaction and final updation in the database

Q if the transaction is in which of the state that we can guarantee that data base is in consistent state

a) aborted **b)** committed **c)** both aborted & committed **d)** none

Q Match:

Column I	Column II
1. Atomicity	A. Recovery Manager
2. Durability	B. Concurrency control manager
3. Isolation	C. Programmer
4. Consistency	

a) 1-a, 2–a, 3-b, 4–c **b)** 1-a, 2-b,3-b,4-c **c)** 1-a,2-a,3-b,4-b **d)** none of these

PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

Concurrent execution is necessary because-

- It leads to good database performance.
- Disk accesses are frequent and relatively slow.
- Overlapping I/O activity with CPU increases throughput and response time.

But interleaving of instructions between transactions may also lead to many problems that can lead to inconsistent database. Sometimes it is possible that even though individual transaction are satisfying the acid properties even though the final statues of the system will be inconsistent.

Lost update problem / Write - Write problem

- If there is any two write operation of different transaction on same data value, and between them there is no read operations, then the second write over writes the first write.

T_1	T_2
Read(A)	
Write(A)	
	Write(A)
	Commit
Commit	

Dirty read problem/ Read -Write problem

- In this problem, the transaction reads a data item updated by another uncommitted transaction, this transaction may in future be aborted or failed.
- The reading transactions end with incorrect results.

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Commit
Abort	

Q The problem that occurs when one transaction updates a database item and then the transaction fails for some reason is _____. **(NET-JUNE-2012)**

(A) Temporary Select Problem
(B) Temporary Modify Problem
(C) Dirty Read Problem
(D) None

Ans: d

Unrepeatable read problem/phantom read problem

- When a transaction tries to read a value of a data item twice, and another transaction updates the data item in between, then the result of the two read operation of the first transaction will differ, this problem is called, Non-repeatable read problem

T_1	T_2
Read(A)	
	Read(A)
	Write(A)
Read(A)	

- Phantom read problem

T_1	T_2
Read(A)	
	Delete(A)
Read(A)	

Q The following schedule is suffering from

T_1	T_2
R(y)	
	R(x) R(y) y = x + y w(y)
R(y)	

a) Lost update problem
b) Unpredictable read problem
c) Both A and B
d) Neither A and B

Q Which of the following scenarios may lead to an irrecoverable error in a database system? **(GATE – 2008) (1 Marks)**
(A) A transaction writes a data item after it is read by an uncommitted transaction
(B) A transaction reads a data item after it is read by an uncommitted transaction
(C) A transaction reads a data item after it is written by a committed transaction
(D) A transaction reads a data item after it is written by an uncommitted transaction
Answer: (D)

Solution is Schedule

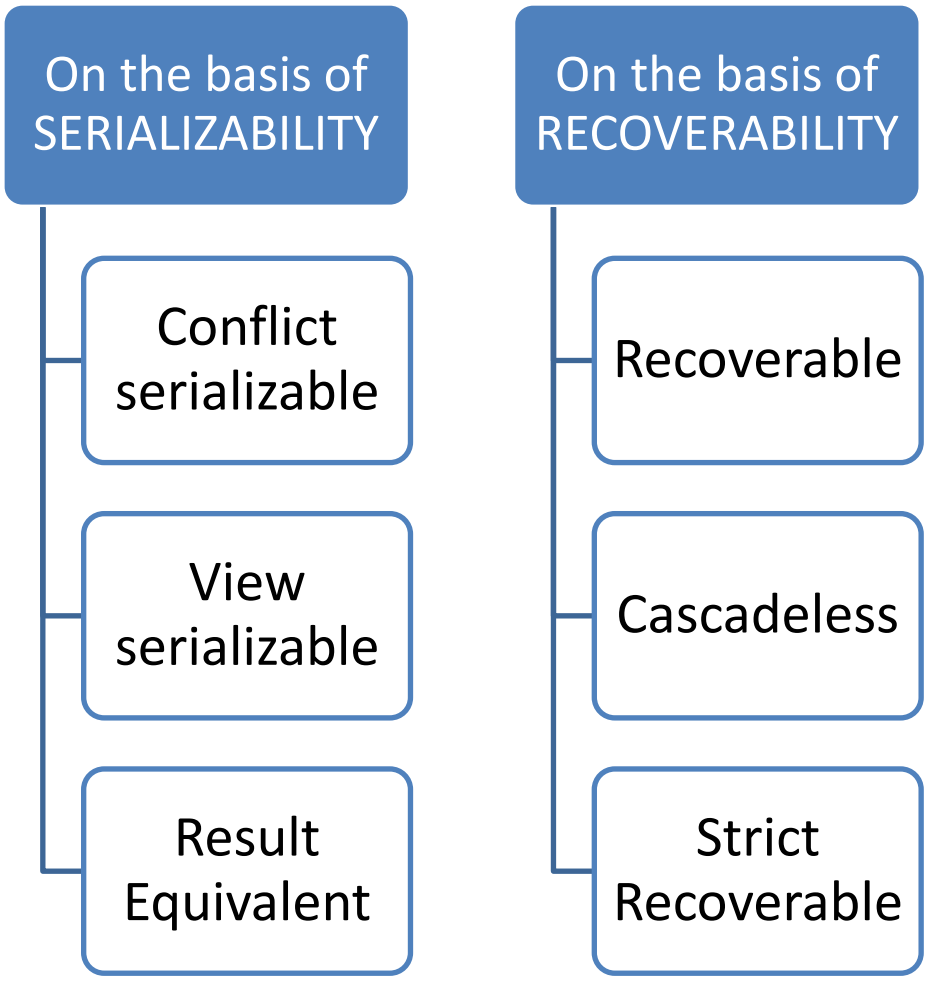
- When two or more transaction executed together or one after another then they can be bundled up into a higher unit of execution called schedule, A **schedule** of *n* transactions *T*1, *T*2, ..., *Tn* is an ordering of the operations of the transactions. Operations from different transactions can be interleaved in the schedule *S*.
- However, schedule for a set of transaction must contain all the instruction of those transaction, and for each transaction *Ti* that participates in the schedule *S*, the operations of *Ti* in *S* must appear in the same order in which they occur in *Ti*.
- Schedule can be of two types-
  - **Serial schedule** - A serial schedule consists of sequence of instruction belonging to different transactions, where instructions belonging to one single transaction appear together. Before complete execution of one transaction another transaction cannot be started.
  - For a set of n transactions, there exist **n!** different valid serial schedules. Every serial schedule lead database into consistent state. Throughput of system is less.

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

  - **Non-serial schedule** - A schedule in which sequence of instructions of a transaction appear in the same order as they appear in individual transaction but the instructions may be interleaved with the instructions of different transactions i.e. concurrent execution of transactions takes place.

T_2	T_3
read(B)	
	read(B)
	write(B)
read(A)	
	read(A)
	write(A)

- So the number of schedules for n different transaction $T_1, T_2, T_3, \text{---------} T_N$ where each transaction conations $n_1, n_2, n_3, \text{----------}, n_4$ respectively will be
- $\{(n_1, n_2, n_3, \text{----------}, n_n)!\} / (n_1!\ n_2!\ n_3!\ \text{----------} n_n!)$
- **Conclusion of schedules**
  - We do not have any method to proof that a schedule is consistent , but from the above discussion we understand that a serial schedule will always be consistent , so if somehow we proof that a non-serial schedule will also have same effects as of a serial schedule that we get a proof that , this particular non-serial schedule will also be consistent “find those schedules that are logically equal to serial schedules”.
  - For a concurrent schedule to result in consistent state, it should be equivalent to a serial schedule. i.e. it must be serializable.



SERIALIZABILITY

Conflicting instructions **-** Let I and J be two consecutive instructions belonging to two different transactions T_i and T_j in a schedule S, the possible I and J instruction can be as-

I= READ(Q), J=READ(Q) ->*Non-conflicting*
I= READ(Q), J=WRITE(Q) ->*Conflicting*
I= WRITE(Q), J=READ(Q) ->*Conflicting*
I= WRITE(Q), J=WRITE(Q) ->*Conflicting*

So, the instructions I and J are said to be conflicting, if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

- **Conflict equivalent** – if one schedule can be converted to another schedule by swapping of non- conflicting instruction then they are called conflict equivalent schedule.

T_1	T_2
R(A)	
A=A-50	
	R(B)
	B=B+50
R(B)	
B=B+50	
	R(A)
	A=A+10

T_1	T_2
	R(B)
	B=B+50
R(A)	
A=A-50	
R(B)	
B=B+50	
	R(A)
	A=A+10

CONFLICT SERIALIZABLE

- The schedules which are conflict equivalent to a serial schedule are called conflict serializable schedule. If a schedule S can be transformed into a schedule S' by a series of swaps of non- conflicting instructions, we say that S and S' are ***conflict equivalent.***
- A schedule S is ***conflict serializable***, if it is conflict equivalent to a serial schedule.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Procedure for determining conflict serializability of a schedule

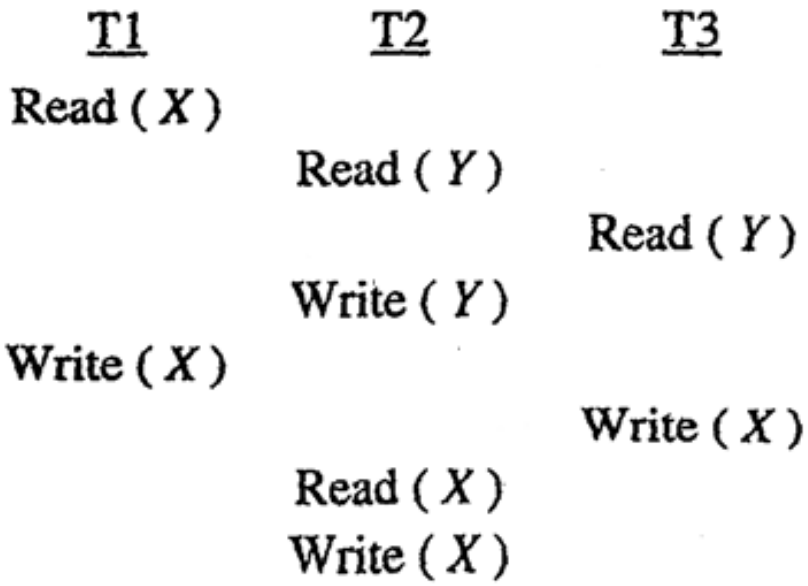
- It can be determined using PRECEDENCE GRAPH method:
- A precedence graph consists of a pair G (V, E)
  - V= set of vertices consisting of all the transactions participating in the schedule.
  - E= set of edges consists of all edges T_i -> T_j, for which one of the following conditions holds:
    - T_i executes write(Q) before T_j executes read(Q)
    - T_i executes read(Q) before T_j executes write(Q)
    - T_i executes write(Q) before T_j executes write(Q)

- If an edge T_i -> T_j exists in the precedence graph, then in any serial schedule S' equivalent to S, Ti must appear before T_j.

- **If the precedence graph for S has no cycle, then schedule S is conflict serializable, else it is not.** This cycle detection can be done by cycle detection algorithms, one of them based on depth first search takes **$O(n^2)$** time.

- The serialializibility order of transactions of equivalent serial schedule can be determined using **topological order** in a precedence graph.

Q Consider the following schedule for transactions T1, T2 and T3: **(GATE – 2010) (2 Marks)**



Which one of the schedules below is the correct serialization of the above?

(A) T1->>T3->>T2 **(B)** T2->>T1->>T3 **(C)** T2->>T3->>T1 **(D)** T3->>T1->>T2

Answer: (A)

Q Consider two transactions T_1 and T_2 which form schedules S_1, S_2, S_3 and S_4 as follows: -

T_1: R_1 [A], W_1 [A], W_1 [B] **T_2**: R_2 [A], R_2 [B], W_2 [B]

S_1: R_1[A], R_2 [A], R_2[B], W_1[A], W_2[B], W_1[B]

S_2: R_1[A], R_2 [A], R_2[B], W_1[A], W_1[B], W_2 [B]

S_3: R_2[A], R_1 [A], R_2[B], W_1[A], W_1[B], W_2[B]

S_4: R_1[A], W_2 [A], R_2[A], W_1[B], R_2[B], W_2[B]

Which of the above schedules is conflicts serializable? **(GATE - 2009) (2 Marks)**

a) Only S_1 **b)** Both S_1 and S_2 **c)** Both S_1 and S_4 **d)** Both S_3 and S_4

Q (GATE - 2009) (2 Marks)

Consider two transactions T_1 and T_2, and four schedules S_1, S_2, S_3, S_4 of T_1 and T_2 as given below:

T_1: R_1 [x]W_1 [x]W_1 [y]

T_2: R_2 [x]R_2 [y]W_2 [y]

S_1: R_1 [x]R_2 [x]R_2 [y]W_1 [x]W_1 [y]W_2 [y]

S_2: R_1 [x]R_2 [x]R_2 [y]W_1 [x]W_2 [y]W_1 [y]

S_3: R_1 [x]W_1 [x]R_2 [x]W_1 [y]R_2 [y]W_2 [y]

S_4: R_2 [x]R_2 [y]R_1 [x]W_1 [x]W_1 [y]W_2 [y]

Which of the above schedules are conflict-serializable?

a) S_1 and S_2 **b)** S_2 and S_3 **c)** S_3 only **d)** S_4 only

Ans: b

Q Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item x, denoted by r(x) and w(x) respectively. Which one of them is conflict serializable. **(GATE - 2014) (2 Marks)**

(a) $r_1(x); r_2(x); w_1(x); r_3(x); w_2(x)$

(b) $r_2(x); r_1(x); w_2(x); r_3(x); w_1(x)$

(c) $r_3(x); r_2(x); r_1(x); w_2(x); w_1(x)$

(d) $r_2(x); w_2(x); r_3(x); r_1(x); w_1(x)$

Ans: d

Q Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below. **(GATE - 2006) (2 Marks)**

T1: r1(X); r1(Z); w1(X); w1(Z)

T2: r2(Y); r2(Z); w2(Z)

T3: r3(Y); r3(X); w3(Y)

S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z);w3(Y); w2(Z); r1(Z); w1(X); w1(Z)

S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z);r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

Which one of the following statements about the schedules is TRUE?

(A) Only S1 is conflict-serializable.

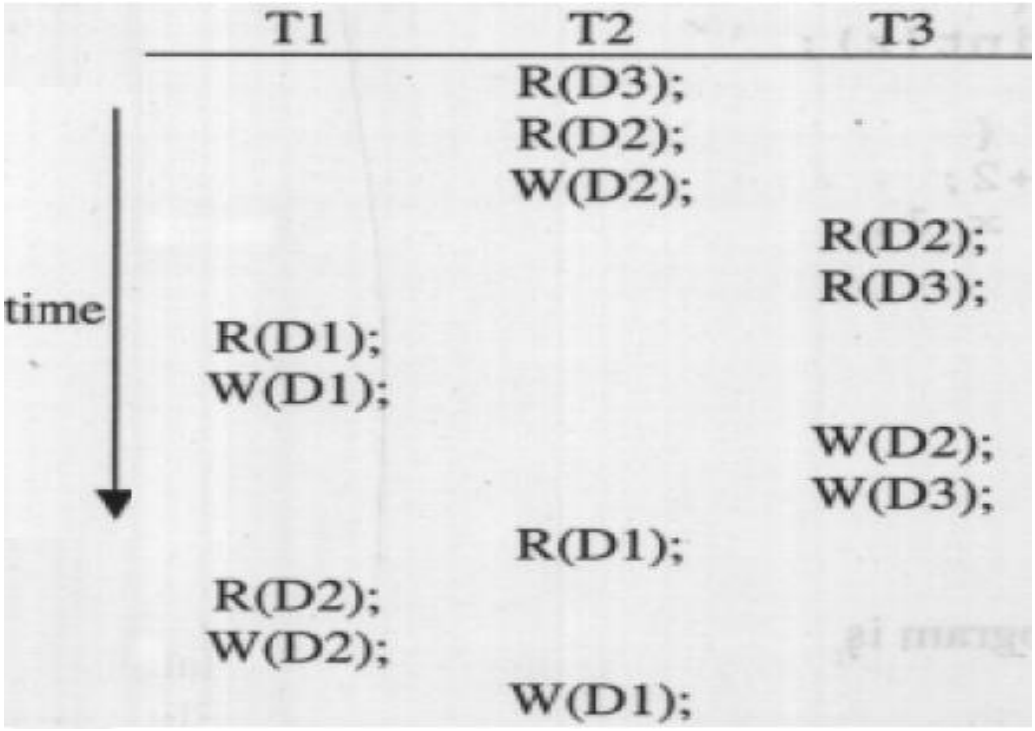
(B) Only S2 is conflict-serializable.

(C) Both S1 and S2 are conflict-serializable.

(D) Neither S1 nor S2 is conflict-serializable.

Answer: (A)

Q Consider three data items D1, D2 and D3 and the following execution schedule of transactions T1, T2 and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively. **(GATE – 2003) (2 Marks)**



Which of the following statements is correct?
(A) The schedule is serializable as T2; T3; T1 **(B)** The schedule is serializable as T2; T1; T3
(C) The schedule is serializable as T3; T2; T1 **(D)** The schedule is not serializable
Answer: (D)

Q Consider following schedules involving two transactions : S 1 : r1(X); r1(Y); r2(X); r2(Y); w2(Y); w1(X) S 2 : r1(X); r2(X); r2(Y); w2(Y); r1(Y); w1(X) Which of the following statement is true ? **(NET-JAN-2017)**
(1) Both S1 and S2 are conflict serializable.
(2) S1 is conflict serializable and S2 is not conflict serializable.
(3) S1 is not conflict serializable and S2 is conflict serializable.
(4) Both S1 and S2 are not conflict serializable.
Ans: c

Q Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item X, denoted by r(X) and w(X) respectively. Which one of them is conflict serializable? **(NET-NOV-2017)**
S1: r1(X); r2(X); w1(X); r3(X); w2(X) **S2**: r2(X); r1(X); w2(X); r3(X); w1(X)
S3: r3(X); r2(X); r1(X); w2(X); w1(X) **S4**: r2(X); w2(X); r3(X); r1(X); w1(X)
(1) S1 **(2)** S2 **(3)** S3 **(4)** S4
Ans: d

Q Consider the following transactions with data items P and Q initialized to zero: **(GATE-2012) (2 Marks)**

```
T1: read (P);
    read (Q);
    if P = 0 then Q: = Q + 1;
    write (Q);

T2: read (Q);
    read (P);
    if Q = 0 then P: = P + 1;
    write (P);
```

Any non-serial interleaving of T1 and T2 for concurrent execution leads to
(A) A serializable schedule
(B) A schedule that is not conflict serializable

(C) A conflict serializable schedule
(D) A schedule for which a precedence graph cannot be drawn
Answer: (B)

Q Consider the following transaction involving two bank accounts x and y. **(GATE - 2015) (1 Marks)**

```
read(x);
x: = x – 50;
write(x);
read(y);
y: = y + 50;
write(y)
```

The constraint that the sum of the accounts x and y should remain constant is that of
(A) Atomicity **(B)** Consistency **(C)** Isolation **(D)** Durability
Answer: (B)

Q Consider the following schedule S of transactions T1, T2, T3, T4 **(GATE - 2014) (2 Marks)**

T1	T2	T3	T4
	Reads(X)		
		Writes(X) Commit	
Writes(X) Commit			
	Writes(Y) Reads(Z) Commit		
			Reads(X) Reads(Y) Commit

Which one of the following statements is CORRECT?
(A) S is conflict-serializable but not recoverable
(B) S is not conflict-serializable but is recoverable
(C) S is both conflict-serializable and recoverable

(D) S is neither conflict-serializable nor is it recoverable
Answer: (C)

Q Consider the following partial Schedule S involving two transactions T1 and T2. Only the read and the write operations have been shown. The read operation on data item P is denoted by read(P) and the write operation on data item P is denoted by write(P).

Time	Transaction-id	
	$T1$	$T2$
1	$read(A)$	
2	$write(A)$	
3		$read(C)$
4		$write(C)$
5		$read(B)$
6		$write(B)$
7		$read(A)$
8		$commit$
9	$read(B)$	

Suppose that the transaction T1 fails immediately after time instance 9. Which one of the following statements is correct? **(GATE-2015) (2 Marks)**
(A) T2 must be aborted and then both T1 and T2 must be re-started to ensure transaction atomicity
(B) Schedule S is non-recoverable and cannot ensure transaction atomicity
(C) Only T2 must be aborted and then re-started to ensure transaction atomicity
(D) Schedule S is recoverable and can ensure atomicity and nothing else needs to be done
Ans: b

Q Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T4);
(write, T4, y, 2, 3);
(start, T1);

(commit, T4);
(write, T1, z, 5, 7);
(checkpoint);
(start, T2);
(write, T2, x, 1, 9);
(commit, T2);
(start, T3);
(write, T3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list **(GATE - 2015) (2 Marks)**

(A) Undo: T3, T1; Redo: T2
(B) Undo: T3, T1; Redo: T2, T4
(C) Undo: none; Redo: T2, T4, T3; T1
(D) Undo: T3, T1, T4; Redo: T2
Answer: (A)

VIEW SERIALIZABLE

- If a schedule is not conflict serializable, still it can be consistent, so let us study a weaker form of serializability called View serializability, and even if a schedule is view serializable still it can be consistent.
- If a schedule is conflict serializable then it will also be view serializable, so we must check view serializability only if a schedule is not conflict serializable.
- If a schedule is not conflict serializable then it must have at least one blind write to be eligible for view serializable. i.e. if a schedule is not conflict serializable and it does not contain any blind write then it can never be view serializable, but if not conflict serializable and have blind write then may or may not be view serializable.
- If a schedule is not conflict serializable and if there exist a blind write. First tabulate all serial schedules possible. Then check one by one whether given schedule is view equivalent to any of the serial schedule. If yes then schedule is view serializable otherwise not.
- Two schedules S and S' are view equivalent, if they satisfy following conditions –
    - For each data item Q, if the transaction T_i reads the initial value of Q in schedule S , then then the transaction T_i must, in schedule S' ,also read the initial value of Q.
    - If a transaction T_i in schedule S reads any data item Q, which is updated by transaction T_j, then a transaction T_i must in schedule S' also read data item Q updated by transaction T_j in schedule S'.
    - For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S, then the same transaction must also perform final write(Q) in schedule S'.
- Complexity wise finding the schedule is view serializable or not is a **NP- complete** problem.

View Serializable

A schedule S is view serializable, if it is view equivalent to a serial schedule.

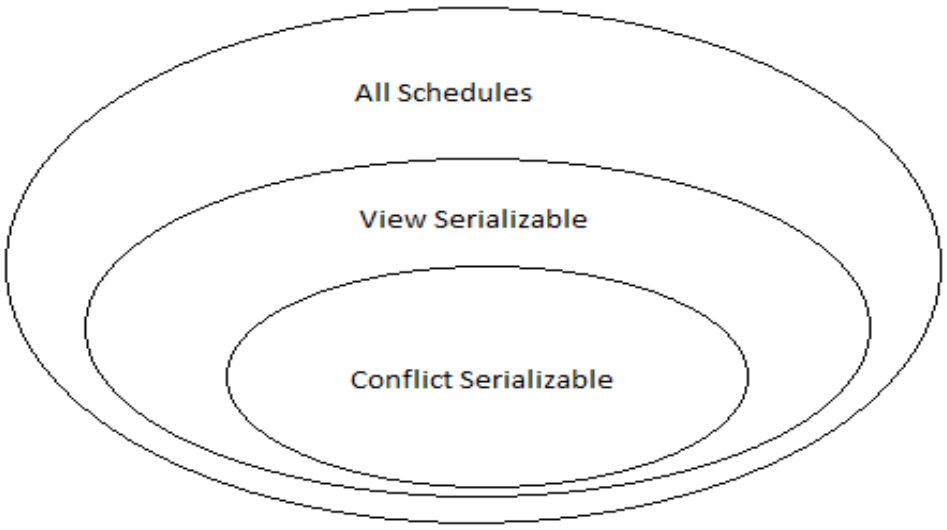
E.g.

Schedule A		
T3	T4	T6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Serial schedule <T3,T4,T6>		
T3	T4	T6
read(Q)		
write(Q)		
	write(Q)	
		write(Q)

In above example schedule A is view equivalent to serial schedule <T3 T4 T6>,
But the schedule A is not conflict equivalent to any serial schedule.
Hence the schedule A is VIEW SERIALIZABLE, but not CONFLICT SERIALIZABLE.

- ***BLIND WRITES***- In the above example, transaction T4 and T6 perform write operation on data item Q without accessing (reading the data item), such updation without knowing/accessing previous value of data item, are called Blind updation or **BLIND WRITE.**
- Every view serializable that is not conflict serializable has a **BLIND WRITE**



Q Which of the following statements (s) is/are TRUE?
S_1: All view serializable schedules are also conflict serializable.
S_2: All conflict serializable schedules are also view serializable.
S_3: If a schedule is not conflict serializable then it is not view serializable
S_4: If a schedule is not view serializable then it is not conflict serializable.
a) S_1 and S_2 only **b)** S_2 and S_3 only **c)** S_2 and S_4 only **d)** S_1 and S_3 only

Q Consider the following schedule 'S' with three transactions.
S: $R_1(B)$; $R_3(C)$; $R_1(A)$; $W_2(A)$; $W_1(A)$, $W_2(B)$; $W_3(A)$; $W_1(B)$; $W_3(B)$, $W_3(C)$
Which of the following is TRUE with respect to the above schedule?
a) It is conflict serializable with sequence [T_1, T_2 T_3]
b) It is conflict serializable with sequence [T_2, T_1 T_3]
c) It is view serializable but not conflict serializable
d) It is neither conflict serializable nor view serializable

Q The following schedule S is having 4 transactions and is executed concurrently. The order of their operations is given below.
S: $R_1(x)$; $R_2(y)$; $W_2(x)$; $W_3(z)$; $R_4(z)$; $R_3(x)$; $W_3(y)$; $W_1(x)$; $W_2(y)$ $W_3(x)$; $R_4(x)$; $W_4(y)$; commit 1; commit 2; commit 3; commit 4;
The schedule is
a) Conflict serializable with sequence [T_1 T_2 T_3 T_4]
b) Conflict serializable with sequence [T_2 T_1 T_3 T_4]
c) View serializable but not conflict serializable
d) Neither conflict serializable nor view serializable

RECOVERABILITY

RECOVERABLE SCHEDULE-

A schedule in which for each pair of transaction T_i and T_j, such that if T_j reads a data item previously written by T_i, then the commit or abort of T_i must appear before T_j. Such a schedule is called Recoverable schedule.

E.g. –

Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1; is ***recoverable***

Sc: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1; ***is not recoverable***

NON- RECOVERABLE SCHEDULE-

A schedule in which for each pair of transaction T_i and T_j, such that if T_j reads a data item previously written by T_i, then the commit or abort operation of T_i appears before T_j. Such a schedule is called Non- Recoverable schedule.

CASCADING ROLLBACK

- It is a phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback.
- Even if the schedule is recoverable the, the commit of transaction may lead lot of transaction to rollback.
- Cascading rollback is undesirable, since it leads to undoing of a significant amount of work.
- Uncommitted reads are not allowed in cascade less schedule.
- ***E.g.***

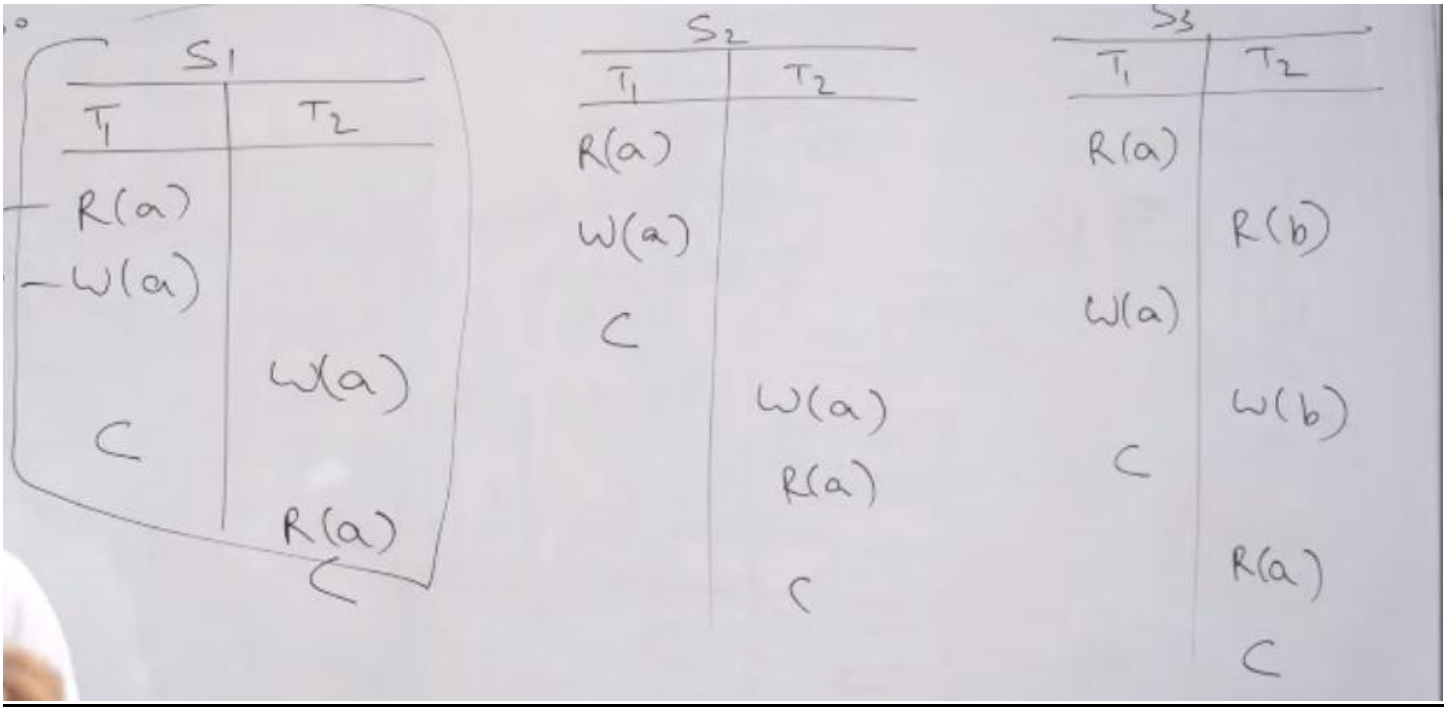
T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

In above schedule T11 reads a data item written by T10 and T12 reads a data item written by T11. So aborting of T10 may lead T11 and so T12 to roll back.

CASCADELESS SCHEDULE-

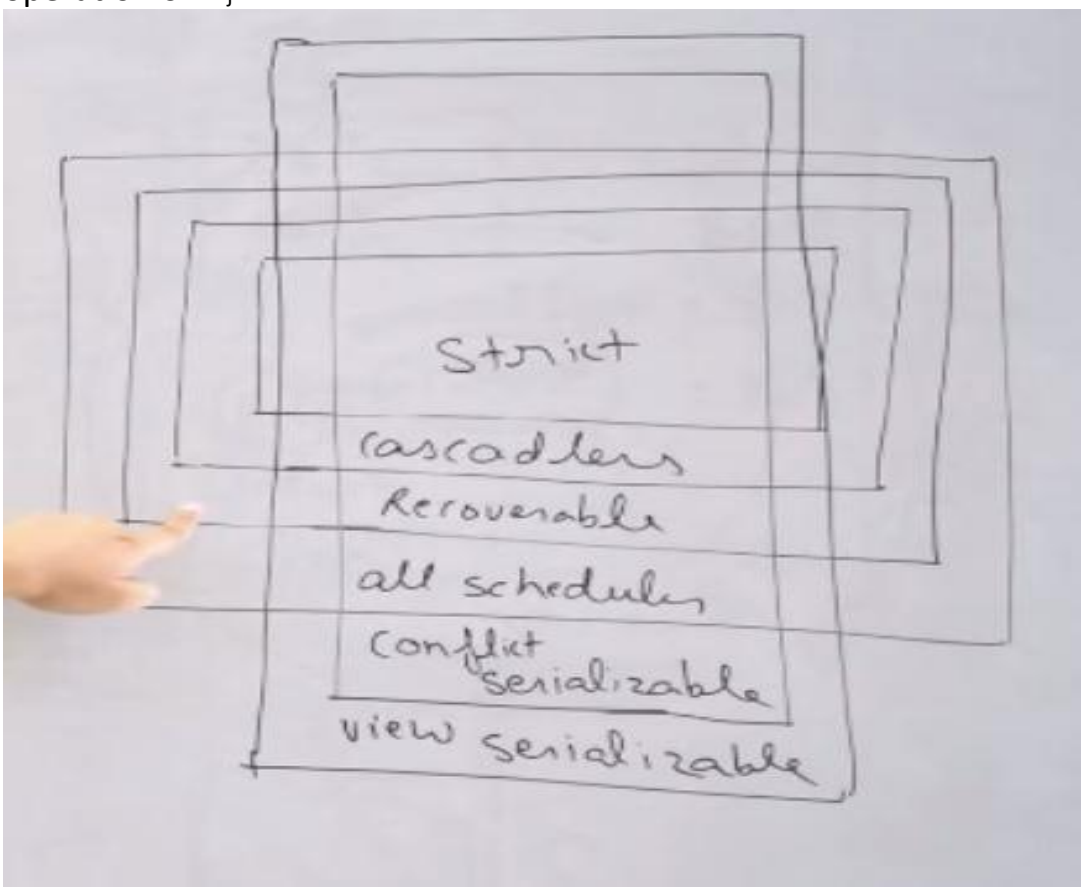
- To avoid cascading rollback, cascade less schedule are used.
- A schedule in which for each pair of transactions T_i and T_j, such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read operation of T_j. Such a schedule is called cascade less schedule.

STRICT



SCHEDULE

A schedule in which for each pair of transactions T_i and T_j, such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read and write operation of T_j.



Q Consider the following schedules:

S1: $R_1(x)\ W_1(x)R_1(y)R_2(x)W_2(x)C_2,\ C_1;$

S2: $R_2(x)\ W_2(x)R_1(y)R_1(x)W_2(x)C_2,\ C_1;$

Which of the following is true?

a) Both S1 and S2 are recoverable **b)** S1 is recoverable but S2 is not

c) S2 is recoverable but S1 is not **d)** Both schedule are not Recoverable

Q Consider the following schedule 'S'.

S: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); c1; w2 (Z); w3 (Y); w2 (Y); c3; c2; The schedule 'S' is

a) Recoverable **b)** Cascade less **c)** recoverable and cascade-less **d)** None

Q Consider the given schedule

R1 (X), R2 (Z), R1 (Z), R3 (X), R3 (Y), W1 (X), Commit1, W3 (Y), Commit3, R2 (Y), W2 (Z), W2 (Y), Commit2. The given schedule is

a) Recoverable only **b)** Cascade less only **c)** recoverable and cascade-less **d)** None

Q Which of the following statement is/are correct

a) Every view serializable schedule is conflict serializable

b) Every strict schedule is conflict serializable

c) Every conflict serializable schedule is cascade less

d) None of the above

Q A Schedule that is not conflict serializable and contains at least one blind write then the schedule is

a) Always view serializable **b)** Always non-serializable

c) May be serializable **d)** None of the above

Q Consider the following schedule

S; $R_2(x),\ w_2(x),\ R_3(y),\ R_1(x),\ R_1(y),\ w_1(x),\ w_3(y),\ R_3(x), R_1(y),\ C_3,\ C_2,\ C_1;$

The above schedule is

a) Recoverable but not cascade less **b)** Recoverable and cascade less but not strict

c) Recoverable, cascade less and also strict

d) Not recoverable

Q Two transactions T1 and T2 are given as:

T1: r1(X)w1(X)r1(Y)w1(Y)
T2: r2(Y)w2(Y)r2(Z)w2(Z)

where $r_i(V)$ denotes a read operation by transaction Ti on a variable V and $w_i(V)$ denotes a write operation by transaction Ti on a variable V. The total number of conflict serializable schedules that can be formed by T1 and T2 is _____**(GATE-2017) (2 Marks)**

Answer: 54

Q Suppose a database schedule S involves transactions T1, T2, ...,Tn. Consider the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule? **(NET-NOV-2017)**

(1) Topological order

(2) Depth - first order

(3) Breadth - first order

(4) Ascending order of transaction indices

Ans: a

System log /transaction log

- To be able to Recover from failures, recovery subsystem of database maintains transaction log to keep track of all operations of a transaction that effects database items, as well as other information that may be needed in recovery operation until the commit/abort point of a transaction.
- Log records can be used to trace out the transaction steps in the event of failure where the transaction need to be rolled back and all the operations of a transaction either need to be redone/ undone.
- A log file is written on the disk to avoid hardware/logical failures.
- A log record for a transaction has following syntax-

  - **[start_transaction, *T*] -** Indicates that transaction *T* has started execution.
  - **[write_item, *T*, *X*, *old_value*, *new_value*]** - Indicates that transaction *T* has changed the value of database item *X* from *old_value* to *new_value.*
  - **[read_item, *T*, *X*]**- Indicates that transaction *T* has read the value of database item *X*.
  - **[commit, *T*]**- Indicates that transaction *T* has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
  - **[Abort, *T*]**. - Indicates that transaction *T* has been aborted.
  - Following operation may be taken at time of failure-
  - **a.)Undo operation-**We can undo the effect of each WRITE operation of a transaction T by tracing backward through the log and resetting all items changed by the WRITE operation of T to their old values.
  - **b.)Redo operation-**We can also redo the effect of the WRITE operations of a transaction T by tracing forward through the log and setting all items changed by a WRITE operation of T to their new values.
  - **Commit Point of a Transaction-** A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log.
    Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
    The transaction then writes an entry [commit,T] into the log. No entry is made in the log after the commit entry, so a transaction cannot be aborted after it is committed.

Q Usage of Pre-emption and Transaction Rollback prevents _____. **(NET-SEP-2013)**

(A) Unauthorized usage of data file
(B) Deadlock situation
(C) Data manipulation
(D) File pre-emption

Ans: b

Q **(NET-DEC-2018)**

Consider the following sequence of two transactions on a bank account (A) with initial balance 20,000 that transfers 5,000 to another account (B) and then apply 10% interest.

(i) T1 start

(ii) T1 A old = 20,000 new 15,000

(iii) T1 B old = 12,000 new = 17,000

(iv) T1 commit

(v) T2 start

(vi) T2 A old = 15,000 new = 16,500

(vii) T2 commit

Suppose the database system crashes just before log record (vii) is written. When the system is restarted, which one statement is true of the recovery process ?

Options :

91394342529. We must redo log record (vi) to set A to 16,500.

91394342530. We must redo log record (vi) to set A to 16,500 and then redo log records (ii) and (iii).

91394342531. We need not redo log records (ii) and (iii) because transaction T1 has committed.

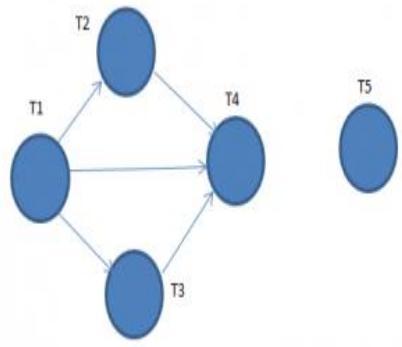
91394342532. We can apply redo and undo operations in arbitrary order because they are idempotent.

Topological order in a precedence graph

- Visit the vertex V in a graph G with in degree zero and delete it from the graph.
- Repeat the step 1 till the graph is empty.
- The order in which the vertex is deleted is the serializability order of the equivalent serial schedule. **The number of conflict equal schedules is equal to no. of topological orders possible in given acyclic precedence graph**

T1	T2	T3	T4	T5
	read(X)			
read(Y) read(Z)				
				read(V) read(W) write(W)
	read(Y) write(Y)			
		write(Z)		
read(U)				
			read(Y) write(X) read(Z) write(Z)	
read(V) write(U)				

Precedence Graph-



`

Equivalent serial schedules

T1->T2->T3->T4->T5
T1->T3->T2->T4->T5
T5->T1->T2->T3->T4
T5->T1->T3->T2->T4
T1->T5->T2->T3->T4
T1->T2->T5->T3->T4
T1->T2->T3->T5->T4
T1->T5->T3->T3->T4
T1->T3->T2->T5->T4

Q (GATE - 2007) (2 Marks)

Consider the following schedules involving two transactions. Which one of the following statements is **TRUE**?

S_1: $r_1(X); r_1(Y); r_2(X); r_2(Y); w_2(Y); w_1(X)$

S_2: $r_1(X); r_2(X); r_2(Y); w_2(Y); r_1(Y); w_1(X)$

a) Both S_1 and S_2 are conflict serializable
b) S_1 is conflict serializable and S_2 is not conflict serializable
c) S_1 is not conflict serializable and S_2 is conflict serializable
d) Both S_1 and S_2 are not conflict serializable.
Ans: C

Q Two transactions T_1 and T_2 are given as
T_1: $r_1(X)w_1(X)r_1(Y)w_1(Y)$
T_2: $r_2(Y)w_2(Y)r_2(Z)w_2(Z)$ where $r_i(V)$ denotes a read operation by transaction T_i on a variable V and $w_i(V)$ denotes a write operation by transaction T_i on a variable V. The total number of conflict serializable schedules that can be formed by T_1 and T_2 is _____ **(GATE-2017) (1 Marks)**
Ans: 54

Q NOT a part of the **ACID** properties of database transactions? **(GATE- 2016) (1 Marks)**
(a) Atomicity
(b) Consistency
(c) Isolation
(d) Deadlock-freedom
Ans: d

Q Consider the following database schedule with two transactions, T_1 and T_2.
$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$
where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a write operation by T_i on a variable Z and a_i denotes an abort by transaction T_i.
Which one of the following statements about the above schedule is **TRUE**? **(GATE- 2016) (1 Marks)**
(a) S is non-recoverable
(b) S is recoverable, but has a cascading abort
(c) S does not have a cascading abort
(d) S is strict
Ans: c

Consider the following transaction involving two bank account x and y.

read (x) ; x : = x – 50; write (x) ; read (y); y : = y + 50 ; write (y)

The constraint that the sum of the accounts x and y should remain constant is that of **(GATE – 2015) (1 Marks)**

(a) Atomicity **(b)** Consistency **(c)** Isolation **(d)** Durability

Ans: b

Q Suppose a database schedule *S* involves transactions $T_1, ..., T_n$. Construct the precedence graph of *S* with vertices representing the transactions and edges representing the conflicts. If *S* is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule? **(GATE- 2016) (1 Marks)**

(a) Topological order **(b)** Depth-first order

(c) Breadth-first order **(d)** Ascending order of transaction indices

Ans: a

Q Consider the following schedules involving two transactions.
S1: r1(X); r1(Y); r2(X); r2(Y); w2(Y); w1(X) S2: r1(X); r2(X); r2(Y); w2(Y); r1(Y); w1(X) Which one of the following statements is correct with respect to above? **(NET-JULY-2018)**

(1) Both S1 and S2 are conflict serializable.

(2) Both S1 and S2 are not conflict serializable.

(3) S1 is conflict serializable and S2 is not conflict serializable.

(4) S1 is not conflict serializable and S2 is conflict serializable

Ans: 3

Q In a certain database there are 2 transactions, one contains 5 instructions and another contains 3 instructions. Then number of concurrent schedules possible is _____.