Doubt Clearing Session

Complete Course on DBMS for GATE

Sanchit Jain • Lesson 8 • Mar 25, 2022

Educator highlights

- Works at **Knowledge Gate**
- Studied at **Delhi Technological University**
- Have qualified gate Have experience of more than 8 years Have a YouTube channel with 5 lakh follower
- Lives in **Ghaziabad, Uttar Pradesh, India**
- Unacademy Educator since **19th July, 2019**
- **844,388 live minutes** taught in last 30 days
- Knows **Punjabi, Hinglish, Hindi and English**

Sanchit Jain

Follow

Legend in GATE & ESE

I am passionate for teaching computer science, having experience of more than 10 years. I teach all computer science subjects for GATE.

56M Watch mins | **2M** Watch mins (last 30 days) | **27K** Followers | **2K** Dedications

Months Wise Calendar for GATE-2023 (Unacademy Plus) (Sanchit Jain)

Month	Morning Class (6:30am – 8:00am)	Evening Class (7:00pm – 8:30pm)
Feb-2022	CN(16-Feb to 26-March) (54 Hours)	DM (2-Feb to 2-Mar) (50 Hours)
Mar-2022	**TOC (6-Apr to 26-May) (58 Hours)**	**DBMS (16-Mar to 2-May) (50 Hours)**
Apr-2022	Compiler (27-May to 15-June) (34 Hours)	DE (2-May to 1-June) (40 Hours)
May-2022	DS & Programming) (16-June to 05-July) (56 Hours)	COA (23-June to 20-July) (38 Hours)
Jun-2022	Algo (7-July to 5-Aug) (38 Hours)	OS (21-July to 24-Aug) (58 Hours)
Jul-2022		
Aug-2022	DM (11-Aug to 20-Sept) (50 Hours)	CN (25-Aug to 5-Oct) (54 Hours)
Sep-2022	DBMS (22-Sept to 29-Oct) (50 Hours)	TOC (6-Oct to 20-Oct) (58 Hours)
Oct-2022	DE (3-Nov to 23-Nov) (40 Hours)	Compiler (24-Oct to 20 Nov) (34 Hours)
Nov-2022	COA (24-Nov to 13-Dec) (38 Hours)	**DS & C** (22-Nov to 20-Dec) **(56 Hours)**
Dec-2022	OS (15-Dec to 31-Jan) (58 Hours)	Algo (21-Dec to 3-Feb) (38 Hours)
Jan-2023		
Feb-2023		

TOC Schedule For GATE-2023

Morning Batch					
Date	Day	Subject-1	Topic	Time	Duration
06-04-2022	Wednesday	TOC	Basics of TOC Part-1	6:30 am to 8:00 am	1.5
07-04-2022	Thursday	TOC	Basics of TOC Part-2	6:30 am to 8:00 am	1.5
08-04-2022	Friday	TOC	Deterministic Finite Automata Part-1	6:30 am to 8:00 am	1.5
11-04-2022	Monday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
12-04-2022	Tuesday	TOC	Deterministic Finite Automata Part-2	6:30 am to 8:00 am	1.5
13-04-2022	Wednesday	TOC	Deterministic Finite Automata Part-3	6:30 am to 8:00 am	1.5
14-04-2022	Thursday	TOC	Deterministic Finite Automata Part-4	6:30 am to 8:00 am	1.5
15-04-2022	Friday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
18-04-2022	Monday	TOC	Deterministic Finite Automata Part-5	6:30 am to 8:00 am	1.5
19-04-2022	Tuesday	TOC	Deterministic Finite Automata Part-6	6:30 am to 8:00 am	1.5
20-04-2022	Wednesday	TOC	Non-Deterministic Finite Automata Part-1	6:30 am to 8:00 am	1.5
21-04-2022	Thursday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
22-04-2022	Friday	TOC	Non-Deterministic Finite Automata Part-2	6:30 am to 8:00 am	1.5
25-04-2022	Monday	TOC	Non-Deterministic Finite Automata Part-3	6:30 am to 8:00 am	1.5
26-04-2022	Tuesday	TOC	Non-Deterministic Finite Automata Part-4	6:30 am to 8:00 am	1.5
27-04-2022	Wednesday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
28-04-2022	Thursday	TOC	Identification of Regular Language	6:30 am to 8:00 am	1.5
29-04-2022	Friday	TOC	Regular Expression Part-1	6:30 am to 8:00 am	1.5
02-05-2022	Monday	TOC	Regular Expression Part-2	6:30 am to 8:00 am	1.5
03-05-2022	Tuesday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
04-05-2022	Wednesday	TOC	Regular Expression Part-3	6:30 am to 8:00 am	1.5
05-05-2022	Thursday	TOC	Regular Expression Part-4	6:30 am to 8:00 am	1.5
06-05-2022	Friday	TOC	Regular Grammar	6:30 am to 8:00 am	1.5
09-05-2022	Monday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
10-05-2022	Tuesday	TOC	Moore and Mealy Machine	6:30 am to 8:00 am	1.5
11-05-2022	Wednesday	TOC	Puch Down Automata Part-1	6:30 am to 8:00 am	1.5
12-05-2022	Thursday	TOC	Puch Down Automata Part-2	6:30 am to 8:00 am	1.5
13-05-2022	Friday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
16-05-2022	Monday	TOC	Context Free Grammar Part-1	6:30 am to 8:00 am	1.5
17-05-2022	Tuesday	TOC	Context Free Grammar Part-2	6:30 am to 8:00 am	1.5
18-05-2022	Wednesday	TOC	Turing Machine Part-1	6:30 am to 8:00 am	1.5
19-05-2022	Thursday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
20-05-2022	Friday	TOC	Turing Machine Part-2	6:30 am to 8:00 am	1.5
23-05-2022	Monday	TOC	Turing Machine Part-3	6:30 am to 8:00 am	1.5
24-05-2022	Tuesday	TOC	Turing Machine Part-4	6:30 am to 8:00 am	1.5
25-05-2022	Wednesday	TOC	Doubt Clearing Session	6:30 am to 8:00 am	1.5
26-05-2022	Thursday	TOC	Complexity Theory	6:30 am to 8:00 am	1.5

Evening Batch

Date	Day	Subject-2	Topic	Time	Duration
16-03-2022	Wednesday	DBMS	Basics and Introduction	7:00 pm to 8:30 pm	1.5
17-03-2022	Thursday	DBMS	Transaction & Concurrency Control Part-1	7:00 pm to 8:30 pm	1.5
18-03-2022	Friday	DBMS	Transaction & Concurrency Control Part-2	7:00 pm to 8:30 pm	1.5
21-03-2022	Monday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
22-03-2022	Tuesday	DBMS	Transaction & Concurrency Control Part-3	7:00 pm to 8:30 pm	1.5
23-03-2022	Wednesday	DBMS	Transaction & Concurrency Control Part-4	7:00 pm to 8:30 pm	1.5
24-03-2022	Thursday	DBMS	Transaction & Concurrency Control Part-5	7:00 pm to 8:30 pm	1.5
25-03-2022	Friday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
28-03-2022	Monday	DBMS	E R Diagram Part - 1	7:00 pm to 8:30 pm	1.5
29-03-2022	Tuesday	DBMS	E R Diagram Part - 2	7:00 pm to 8:30 pm	1.5
30-03-2022	Wednesday	DBMS	E R Diagram Part - 3	7:00 pm to 8:30 pm	1.5
31-03-2022	Thursday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
01-04-2022	Friday	DBMS	Functional Dependency Part -1	7:00 pm to 8:30 pm	1.5
04-04-2022	Monday	DBMS	Functional Dependency Part -2	7:00 pm to 8:30 pm	1.5
05-04-2022	Tuesday	DBMS	Functional Dependency Part -3	7:00 pm to 8:30 pm	1.5
06-04-2022	Wednesday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
07-04-2022	Thursday	DBMS	Normalization Part -1	7:00 pm to 8:30 pm	1.5
08-04-2022	Friday	DBMS	Normalization Part -2	7:00 pm to 8:30 pm	1.5
11-04-2022	Monday	DBMS	Normalization Part -3	7:00 pm to 8:30 pm	1.5
12-04-2022	Tuesday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
13-04-2022	Wednesday	DBMS	Indexing and Physical Structure Part-1	7:00 pm to 8:30 pm	1.5
14-04-2022	Thursday	DBMS	Indexing and Physical Structure Part-2	7:00 pm to 8:30 pm	1.5
15-04-2022	Friday	DBMS	Indexing and Physical Structure Part-3	7:00 pm to 8:30 pm	1.5
18-04-2022	Monday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
19-04-2022	Tuesday	DBMS	Relational Algebra Part - 1	7:00 pm to 8:30 pm	1.5
20-04-2022	Wednesday	DBMS	Relational Algebra Part - 2	7:00 pm to 8:30 pm	1.5
21-04-2022	Thursday	DBMS	Relational Algebra Part - 3	7:00 pm to 8:30 pm	1.5
22-04-2022	Friday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
25-04-2022	Monday	DBMS	SQL Part -1	7:00 pm to 8:30 pm	1.5
26-04-2022	Tuesday	DBMS	SQL Part -2	7:00 pm to 8:30 pm	1.5
27-04-2022	Wednesday	DBMS	SQL Part -3	7:00 pm to 8:30 pm	1.5
28-04-2022	Thursday	DBMS	Doubt Clearing Session	7:00 pm to 8:30 pm	1.5
29-04-2022	Friday	DBMS	Relational Calculus Part-1	7:00 pm to 8:30 pm	1.5
02-05-2022	Monday	DBMS	Relational Calculus Part-2	7:00 pm to 8:30 pm	1.5

DBMS Schedule For GATE-2023

GATE & ESE | Plus | All courses | All languages | CS & IT

HINDI CS & IT

Complete Course on DBMS for GATE

Lesson 5 • Today, 7:00 PM

Sanchit Jain

HINDI CS & IT

Droppers Course on Operating System

Lesson 25 • Mar 23, 2022 1:30 PM

Sanchit Jain

HINDI CS & IT

Complete Course on Computer Networks for GATE

Lesson 29 • Today, 7:30 AM

Sanchit Jain

HINDI CS & IT

Orientation Course for GATE 2023 & 2024

Ended on Feb 11, 2022 • 1 lessons

Sanchit Jain

HINDI CS & IT

Foundation Course on Discrete Mathematics for GATE

Ended on Mar 8, 2022 • 25 lessons

Sanchit Jain

HINGLISH CS & IT

Foundation Course on Compiler for GATE 2022

Ended on Feb 3, 2022 • 17 lessons

Sanchit Jain

HINGLISH CS & IT

Foundation Course on Theory of Computation for GATE 2022

Ended on Feb 3, 2022 • 21 lessons

Sanchit Jain

HINGLISH CS & IT

Complete Course on Operating System for GATE

Ended on Jan 11, 2022 • 28 lessons

Sanchit Jain

HINGLISH ELECTRICAL ENGINEERING

Complete Course on Computer Architecture for GATE

Ended on Dec 13, 2021 • 19 lessons

Sanchit Jain

HINGLISH CS & IT

Complete Course on Computer Networks for GATE

Ended on Jan 7, 2022 • 27 lessons

Sanchit Jain

HINDI CS & IT

Complete Course on Digital Electronics for GATE

Ended on Nov 23, 2021 • 18 lessons

Sanchit Jain

HINDI CS & IT

Complete Course on Algorithms for GATE

Ended on Nov 9, 2021 • 19 lessons

Sanchit Jain

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)"

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

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

- 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)

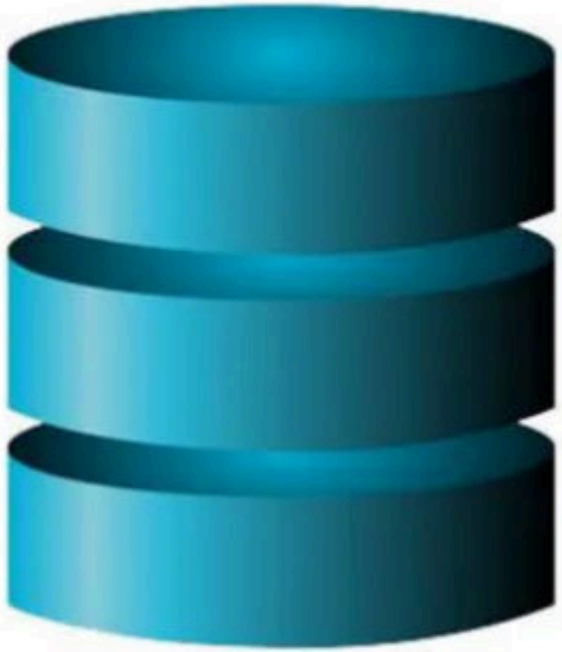
(C) Both (A) and (B)

(D) None of these

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.

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



- 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:

A = Atomicity

C = Consistency

I = Isolation

D = Durability

- **<u>Atomicity</u> -** 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.

- **<u>Isolation</u> -** 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.***

Q NOT a part of the **ACID** properties of database transactions? **(GATE- 2016) (1 Marks)**

(a) Atomicity

(b) Consistency

(c) Isolation

(d) Deadlock-freedom

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

Q All Oracle transactions obey the basic properties of a database transaction. What is the name of the following property?
'All tasks of a transaction are performed or none of them are. There are no partial transactions.' **[Asked in Wipro NLTH 2021]**

a) Atomicity

b) Durability

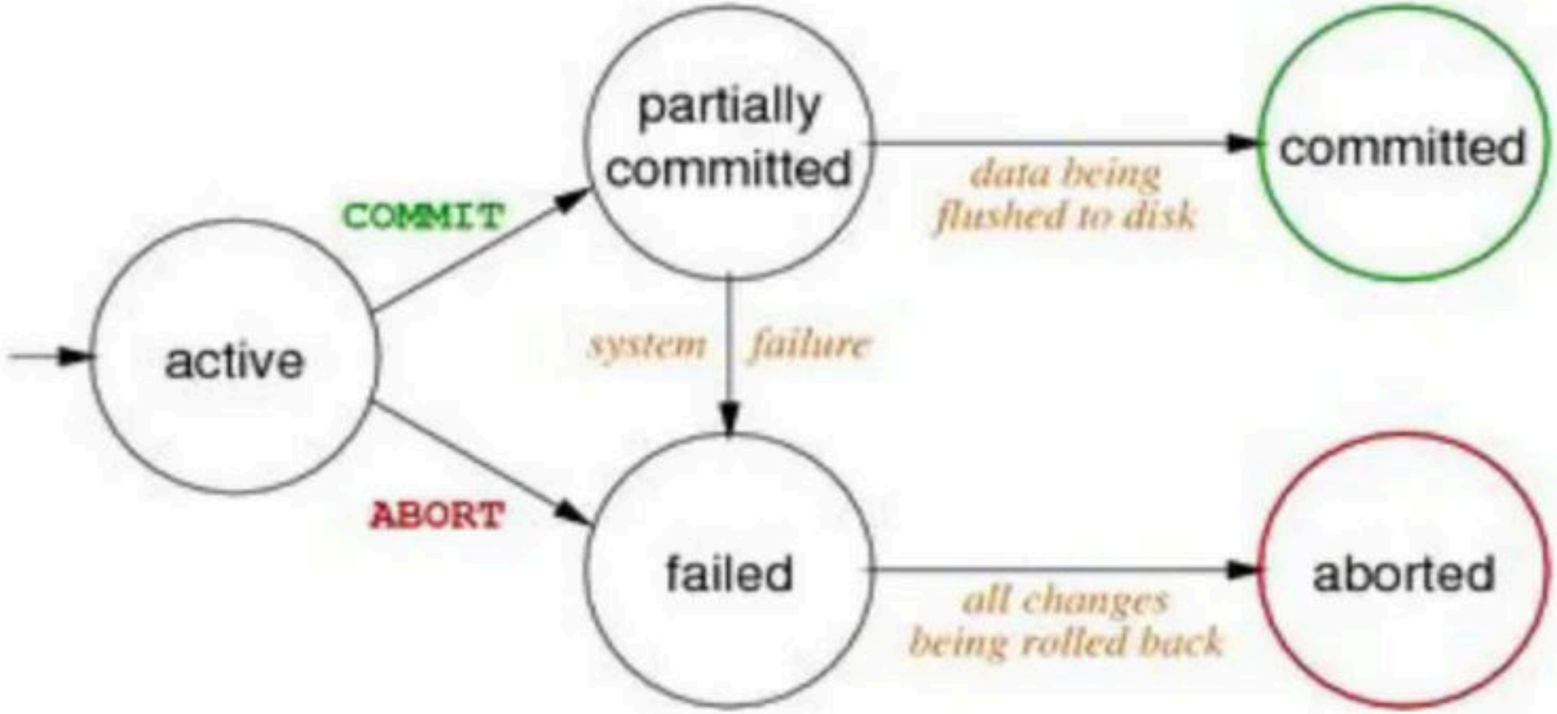
c) Consistency

d) Isolation

Break

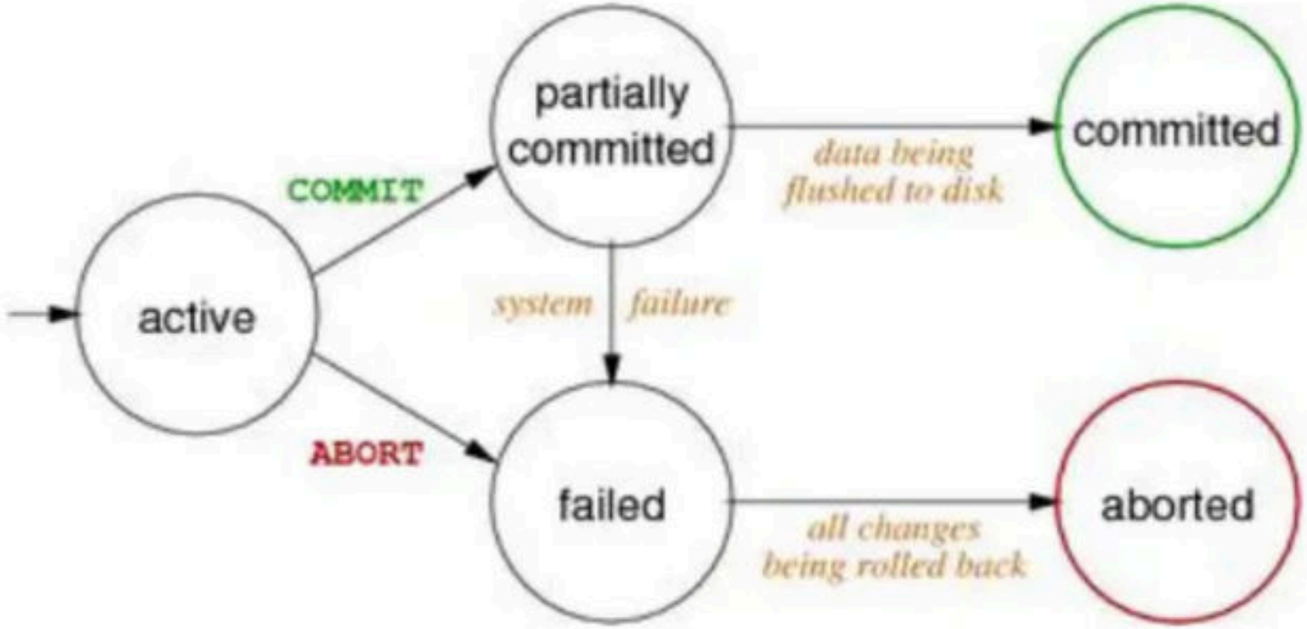
Transaction states

- **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.



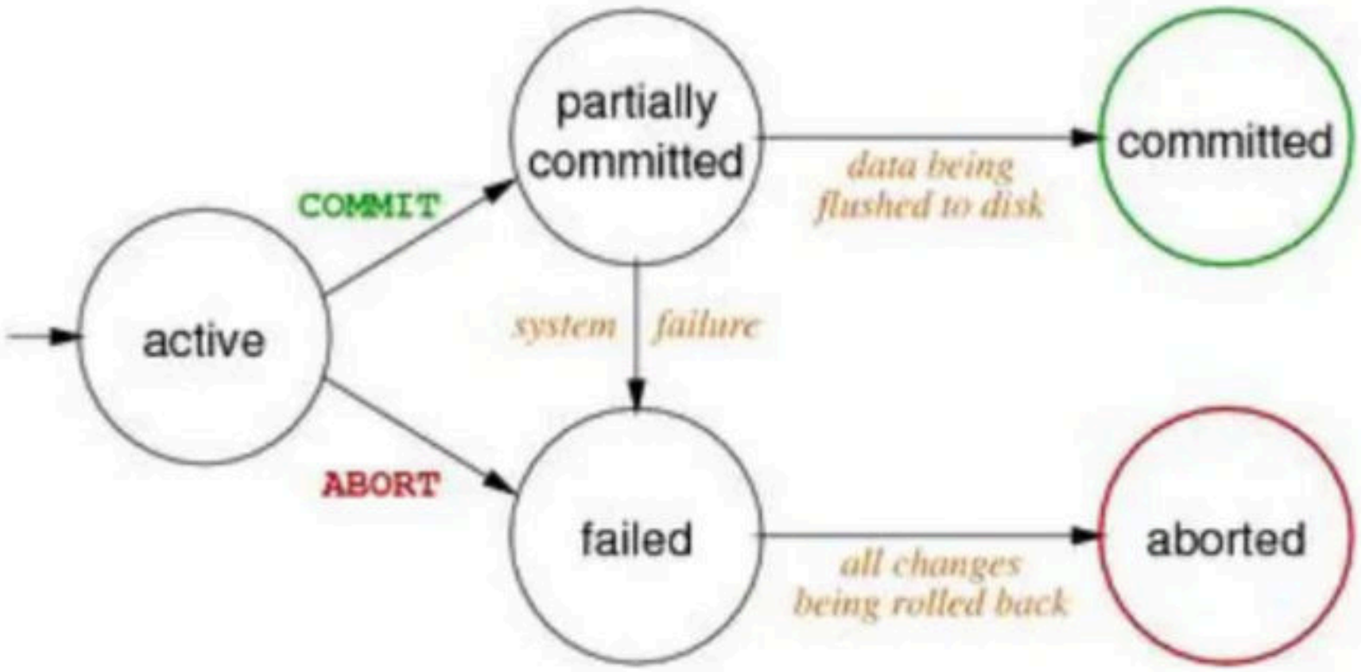
Transaction states

- **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.



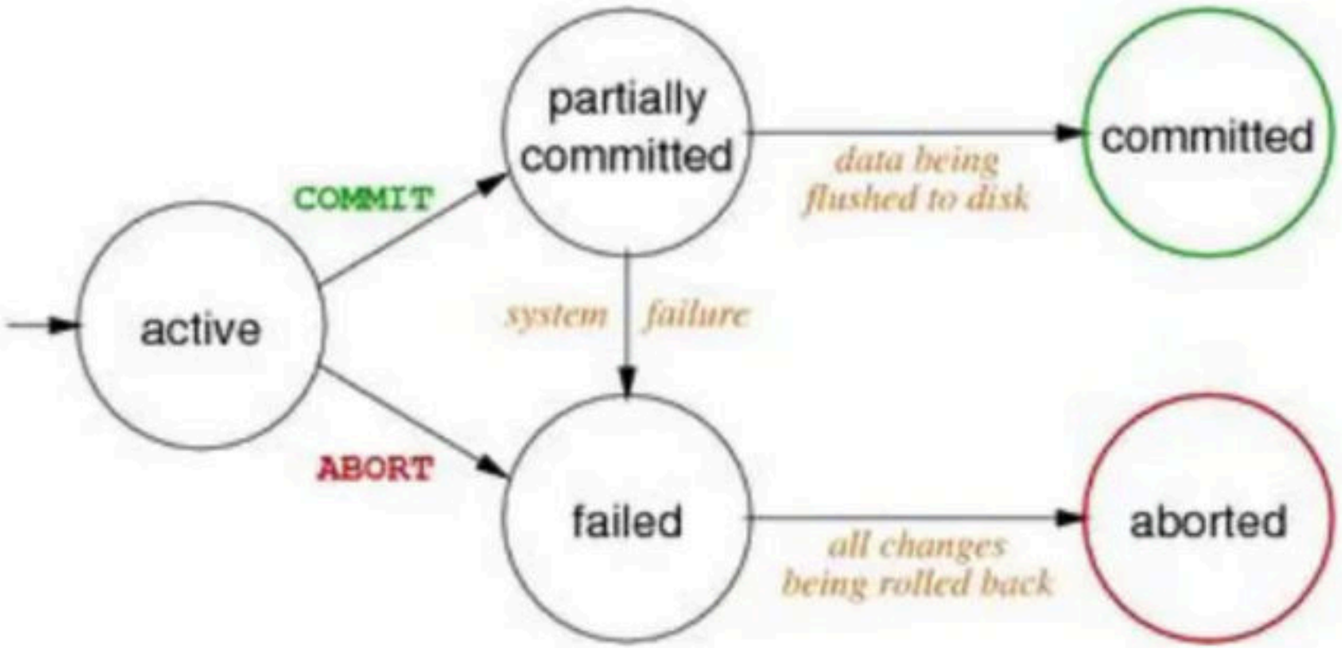
Transaction states

- **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back



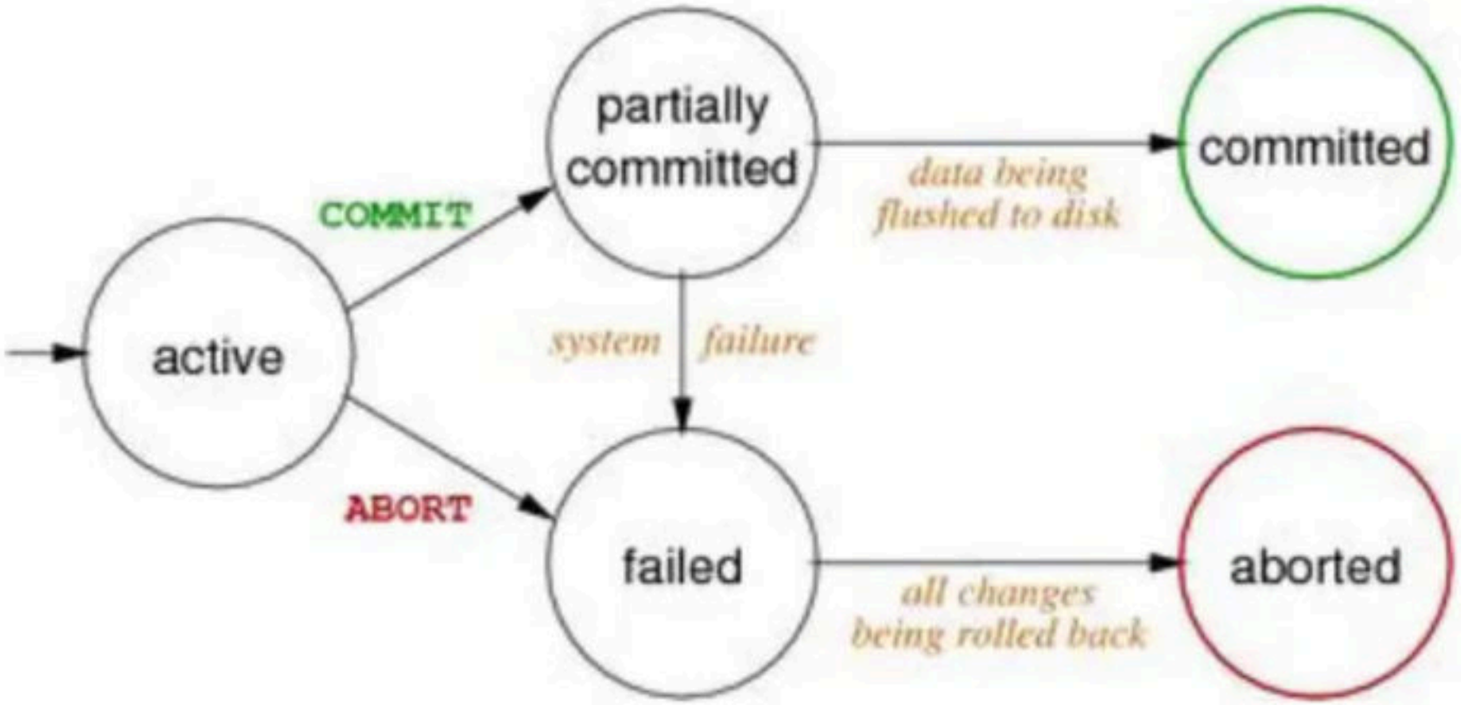
Transaction states

- **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.



Transaction states

- **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

Break

Why we need concurrent execution

- Concurrent execution is necessary because-
  - It leads to good database performance , less weighting time.
  - Overlapping I/O activity with CPU increases throughput and response time.

PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

- 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

Unrepeatable 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) Unrepeatable 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

Break

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 $T_1, T_2, ..., T_n$ 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 T_i that participates in the schedule S, the operations of T_i in S must appear in the same order in which they occur in T_i.

- **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.

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

- 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.

- **<u>Non-serial schedule</u>** - 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)

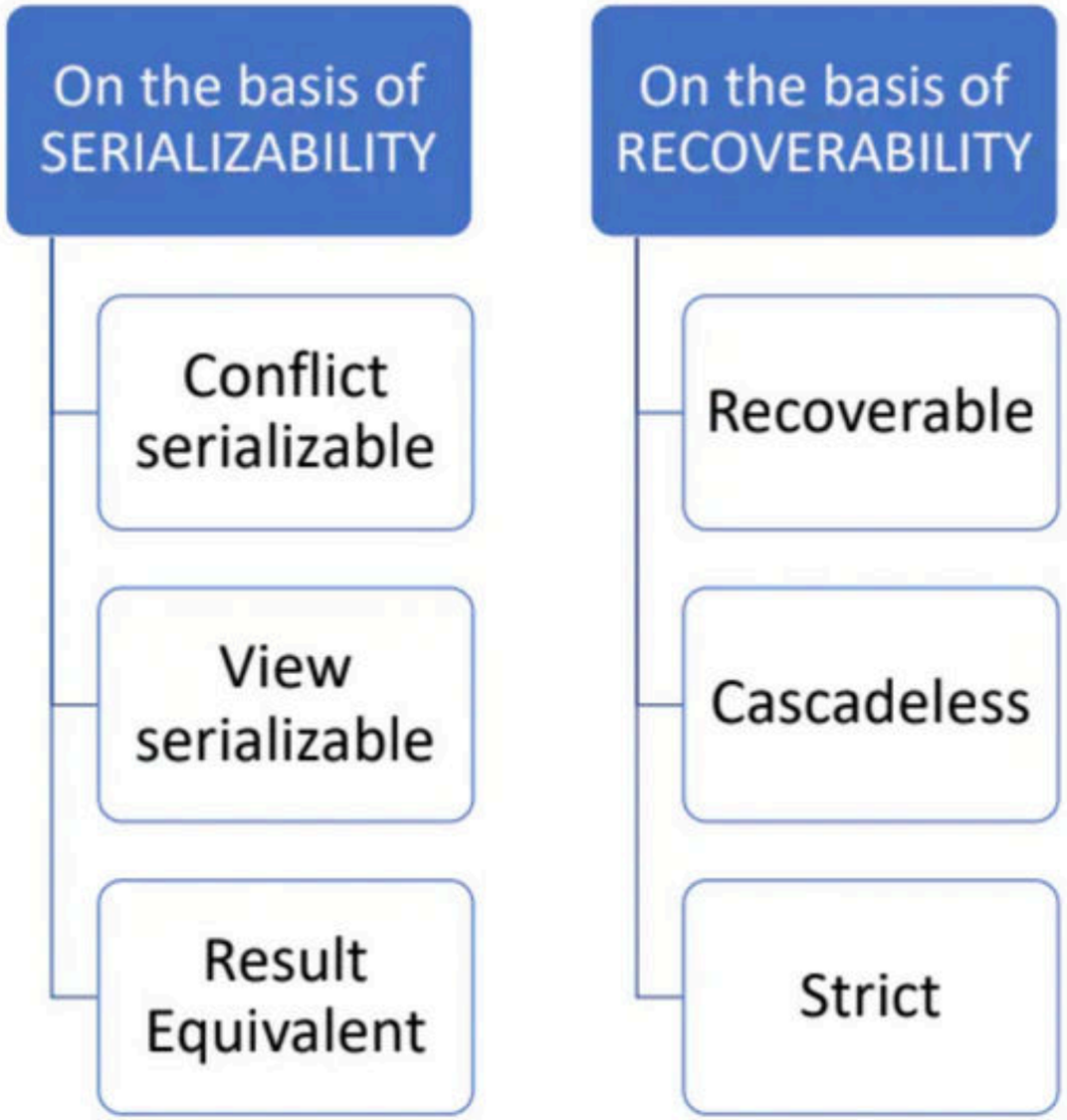
- So the number of schedules for n different transaction T_1, T_2, T_3,---------T_N where each transaction conations n_1, n_2, n_3, ----------,n_4 respectively will be.

- $\{(n_1 + n_2 + n_3 + \text{----------} + n_n)!\} / (n_1!\ n_2!\ n_3!\ \text{----------} n_n!)\}$

- $\{(n_1 + n_2 + n_3 + \text{----------} + n_n)!\} / (n_1!\ n_2!\ n_3!\ \text{----------} n_n!)\} - n!$

- **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.


On the basis of SERIALIZABILITY
Conflict serializable
View serializable
Result Equivalent
On the basis of RECOVERABILITY
Recoverable
Cascadeless
Strict

T_1	T_2
R(A)	
	R(B)

T_1	T_2
	R(B)
R(A)	

T_1	T_2
R(A)	
	W(A)

T_1	T_2
	W(A)
R(A)	

T_1	T_2
R(A)	
	W(B)

T_1	T_2
	W(B)
R(A)	

T_1	T_2
	R(A)
W(A)	

T_1	T_2
W(A)	
	R(A)

T_1	T_2
R(A)	
	R(A)

T_1	T_2
	R(A)
R(A)	

T_1	T_2
	W(A)
W(A)	

T_1	T_2
W(A)	
	W(A)

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

Q Consider a schedule of transactions T_1 and T_2:
Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?**(Gate-2020) (2 Marks)**

S	
T_1	T_2
RA	
	RB
	WB
RC	
	RD
WD	
	WC
WB	
Commit	
	Commit

a)

S	
T_1	T_2
	RB
	WB
	RD
RA	
RC	
WD	
WB	
	WC
Commit	
	Commit

b)

S	
T_1	T_2
RA	
RC	
WD	
WB	
	RB
	WB
	RD
	WC
Commit	
	Commit

c)

S	
T_1	T_2
RA	
RC	
WD	
	RB
	WB
	RD
WB	
	WC
Commit	
	Commit

d)

S	
T_1	T_2
	RB
	WB
	RD
	WC
RA	
RC	
WD	
WB	
Commit	
	Commit

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 $T_i \rightarrow T_j$, 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 $T_i \rightarrow T_j$ 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.

Break

Q Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i. Consider the following two schedules.

- $S_1: r_1(x) r_1(y) r_2(x) r_2(y) w_2(y) w_1(x)$
- $S_2: r_1(x) r_2(x) r_2(y) w_2(y) r_1(y) w_1(x)$

Which one of the following options is correct? **(GATE - 2021) (2 Marks)**

a) S_1 is conflict serializable, and S_2 is not conflict serializable

b) S_1 is not conflict serializable, and S_2 is conflict serializable

c) Both S_1 and S_2 are conflict serializable

d) Neither S_1 nor S_2 is conflict serializable

Q Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i. Consider the following two schedules.

- $S_1: r_1(x) r_1(y) r_2(x) r_2(y) w_2(y) w_1(x)$
- $S_2: r_1(x) r_2(x) r_2(y) w_2(y) r_1(y) w_1(x)$

Which one of the following options is correct? **(GATE - 2021) (2 Marks)**

a) S_1 is conflict serializable, and S_2 is not conflict serializable

b) S_1 is not conflict serializable, and S_2 is conflict serializable

c) Both S_1 and S_2 are conflict serializable

d) Neither S_1 nor S_2 is conflict serializable

S_1	
T_1	T_2
R(X)	
R(Y)	
	R(X)
	R(Y)
	W(Y)
W(X)	

S_2	
T_1	T_2
R(X)	
	R(X)
	R(Y)
	W(Y)
R(Y)	
W(X)	

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-2018)** **(GATE - 2014) (2 Marks)**

S_1: $r_1(X); r_2(X); w_1(X); r_3(X); w_2(X)$

S_2: $r_2(X); r_1(X); w_2(X); r_3(X); w_1(X)$

S_3: $r_3(X); r_2(X); r_1(X); w_2(X); w_1(X)$

S_4: $r_2(X); w_2(X); r_3(X); r_1(X); w_1(X)$

(a) S_1 **(b)** S_2 **(c)** S_3 **(d)** S_4

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-2018)** **(GATE - 2014) (2 Marks)**

S_1: $r_1(X); r_2(X); w_1(X); r_3(X); w_2(X)$

	S_1	
T_1	T_2	T_3
R(X)		
	R(X)	
W(X)		
		R(X)
	W(X)	

S_2: $r_2(X); r_1(X); w_2(X); r_3(X); w_1(X)$

	S_2	
T_1	T_2	T_3
	R(X)	
R(X)		
	W(X)	
		R(X)
W(X)		

S_3: $r_3(X); r_2(X); r_1(X); w_2(X); w_1(X)$

	S_3	
T_1	T_2	T_3
		R(X)
	R(X)	
R(X)		
	W(X)	
W(X)		

S_4: $r_2(X); w_2(X); r_3(X); r_1(X); w_1(X)$

	S_4	
T_1	T_2	T_3
	R(X)	
	W(X)	
		R(X)
R(X)		
W(X)		

(a) S_1 **(b)** S_2 **(c)** S_3 **(d)** S_4

Q Consider following schedules involving two transactions :

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

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

Which of the following statement is true ? **(NET-JAN-2017)** **(GATE - 2007) (2 Marks)**

(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.

Q Consider following schedules involving two transactions :

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

T_1	T_2
r(x)	
r(y)	
	r(x)
	r(y)
	w(y)
w(x)	

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

T_1	T_2
r(x)	
	r(x)
	r(y)
	w(y)
r(y)	
w(x)	

Which of the following statement is true ? **(NET-JAN-2017)** **(GATE - 2007) (2 Marks)**

(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.

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

T_1: read (P);
read (Q);
if P = 0 then Q: = Q + 1;
write (Q);

T_2: read (Q);
read (P);
if Q = 0 then P: = P + 1;
write (P);

Any non-serial interleaving of T_1 and T_2 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

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

T_1: read (P);
read (Q);
if P = 0 then Q: = Q + 1;
write (Q);

T_2: read (Q);
read (P);
if Q = 0 then P: = P + 1;
write (P);

T_1
r(P)
r(Q)
w(Q)

T_2
r(Q)
r(P)
w(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

Q Consider the following schedule for transactions T_1, T_2 and T_3: **(GATE – 2010) (2 Marks)**
Which one of the schedules below is the correct serialization of the above?

T1	T2	T3
Read (X)		
	Read (Y)	
		Read (Y)
	Write (Y)	
Write (X)		
		Write (X)
	Read (X)	
	Write (X)	

(A) T_1->>T_3->>T_2

(B) T_2->>T_1->>T_3

(C) T_2->>T_3->>T_1

(D) T_3->>T_1->>T_2

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

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], R_2 [A], R_2[B], W_1[B], R_2[B], W_2[B]

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

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_1	
T_1	T_2
R(A)	
	R(A)
	R(B)
W(A)	
	W(B)
W(B)	

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

S_2	
T_1	T_2
R(A)	
	R(A)
	R(B)
W(A)	
W(B)	
	W(B)

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

S_3	
T_1	T_2
	R(A)
R(A)	
	R(B)
W(A)	
W(B)	
	W(B)

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

S_4	
T_1	T_2
R(A)	
	R(A)
	R(B)
W(B)	
	R(B)
	W(B)

Q Consider the transactions T_1, T_2, and T_3 and the schedules S_1 and S_2 given below. **(GATE - 2006) (2 Marks)**

T_1: $r_1(X)$; $r_1(Z)$; $w_1(X)$; $w_1(Z)$

T_2: $r_2(Y)$; $r_2(Z)$; $w_2(Z)$

T_3: $r_3(Y)$; $r_3(X)$; $w_3(Y)$

S_1: $r_1(X)$; $r_3(Y)$; $r_3(X)$; $r_2(Y)$; $r_2(Z)$;$w_3(Y)$; $w_2(Z)$; $r_1(Z)$; $w_1(X)$; $w_1(Z)$

S_2: $r_1(X)$; $r_3(Y)$; $r_2(Y)$; $r_3(X)$; $r_1(Z)$;$r_2(Z)$; $w_3(Y)$; $w_1(X)$; $w_2(Z)$; $w_1(Z)$

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

(A) Only S_1 is conflict-serializable.

(B) Only S_2 is conflict-serializable.

(C) Both S_1 and S_2 are conflict-serializable.

(D) Neither S_1 nor S_2 is conflict-serializable.

Q Consider the transactions T_1, T_2, and T_3 and the schedules S_1 and S_2 given below. **(GATE - 2006) (2 Marks)**

T_1: $r_1(X)$; $r_1(Z)$; $w_1(X)$; $w_1(Z)$

T_2: $r_2(Y)$; $r_2(Z)$; $w_2(Z)$

T_3: $r_3(Y)$; $r_3(X)$; $w_3(Y)$

S_1: $r_1(X)$; $r_3(Y)$; $r_3(X)$; $r_2(Y)$; $r_2(Z)$;$w_3(Y)$; $w_2(Z)$; $r_1(Z)$; $w_1(X)$; $w_1(Z)$

S_2: $r_1(X)$; $r_3(Y)$; $r_2(Y)$; $r_3(X)$; $r_1(Z)$;$r_2(Z)$; $w_3(Y)$; $w_1(X)$; $w_2(Z)$; $w_1(Z)$

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

(A) Only S_1 is conflict-serializable.

(B) Only S_2 is conflict-serializable.

(C) Both S_1 and S_2 are conflict-serializable.

(D) Neither S_1 nor S_2 is conflict-serializable.

	S_1	
T_1	T_2	T_3
R(X)		
		R(Y)
		R(X)
	R(Y)	
	R(Z)	
		W(Y)
	W(Z)	
R(Z)		
W(X)		
W(Z)		

	S_2	
T_1	T_2	T_3
R(X)		
		R(Y)
	R(Y)	
		R(X)
R(Z)		
	R(Z)	
		W(Y)
W(X)		
	W(Z)	
W(Z)		

Q Consider three data items D_1, D_2 and D_3 and the following execution schedule of transactions T_1, T_2 and T_3. 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 T_2; T_3; T_1

(B) The schedule is serializable as T_2; T_1; T_3

(C) The schedule is serializable as T_3; T_2; T_1

(D) The schedule is not serializable

	T1	T2	T3
		R(D3);	
		R(D2);	
		W(D2);	
			R(D2);
			R(D3);
time ↓	R(D1);		
	W(D1);		
			W(D2);
			W(D3);
		R(D1);	
	R(D2);		
	W(D2);		
		W(D1);	

Q Suppose a database schedule S involves transactions T_1, T_2, ...,T_n. 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) (GATE- 2016) (1 Marks)

(a) Topological order

(b) Depth - first order

(c) Breadth - first order

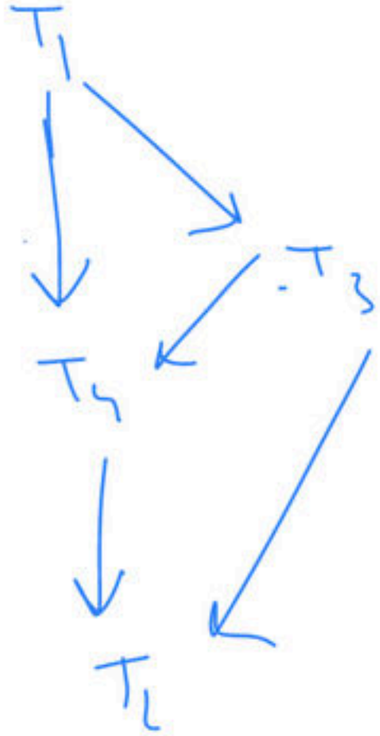
(d) Ascending order of transaction indices

Let $R_i(z)$ and $W_i(z)$ denote read and write operations on a data element z by a transaction T_i, respectively. Consider the schedule S with four transactions.

(GATE- 2022) (2 Marks)

$$S: \cancel{R_4(x)}\cancel{R_2(x)}\cancel{R_3(x)}\cancel{R_1(y)}W_1(y)W_2(x)W_3(y)R_4(y)$$

$T_i \longrightarrow T_j$



Which one of the following serial schedules is conflict equivalent to S?

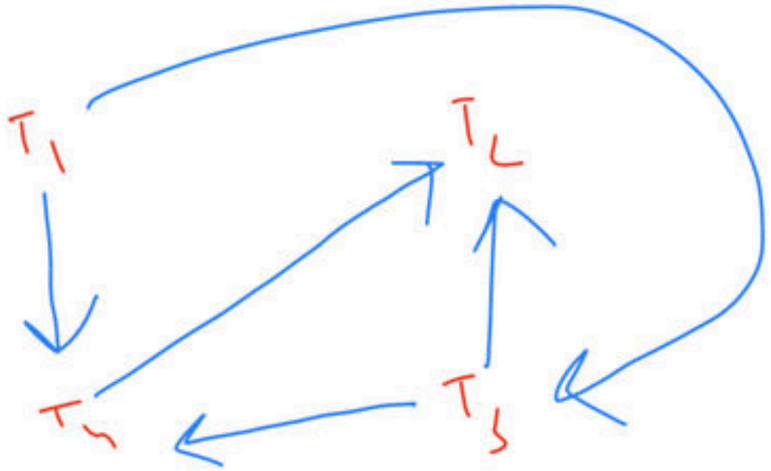
5 → A. $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

11 → B. $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

19 → C. $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

6 → D. $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$

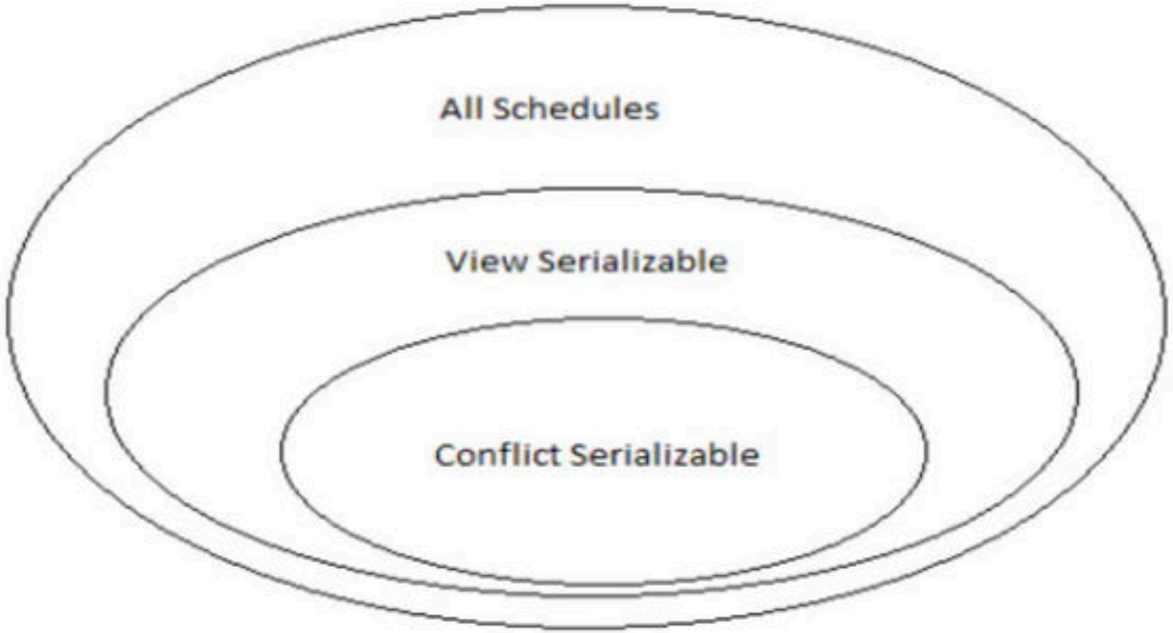
T_1	T_2	T_3	T_4
			$R(x)$
	$R(x)$		
		$R(x)$	
$R(y)$			
$W(y)$			
	$W(x)$		
		$W(y)$	
			$R(y)$



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.

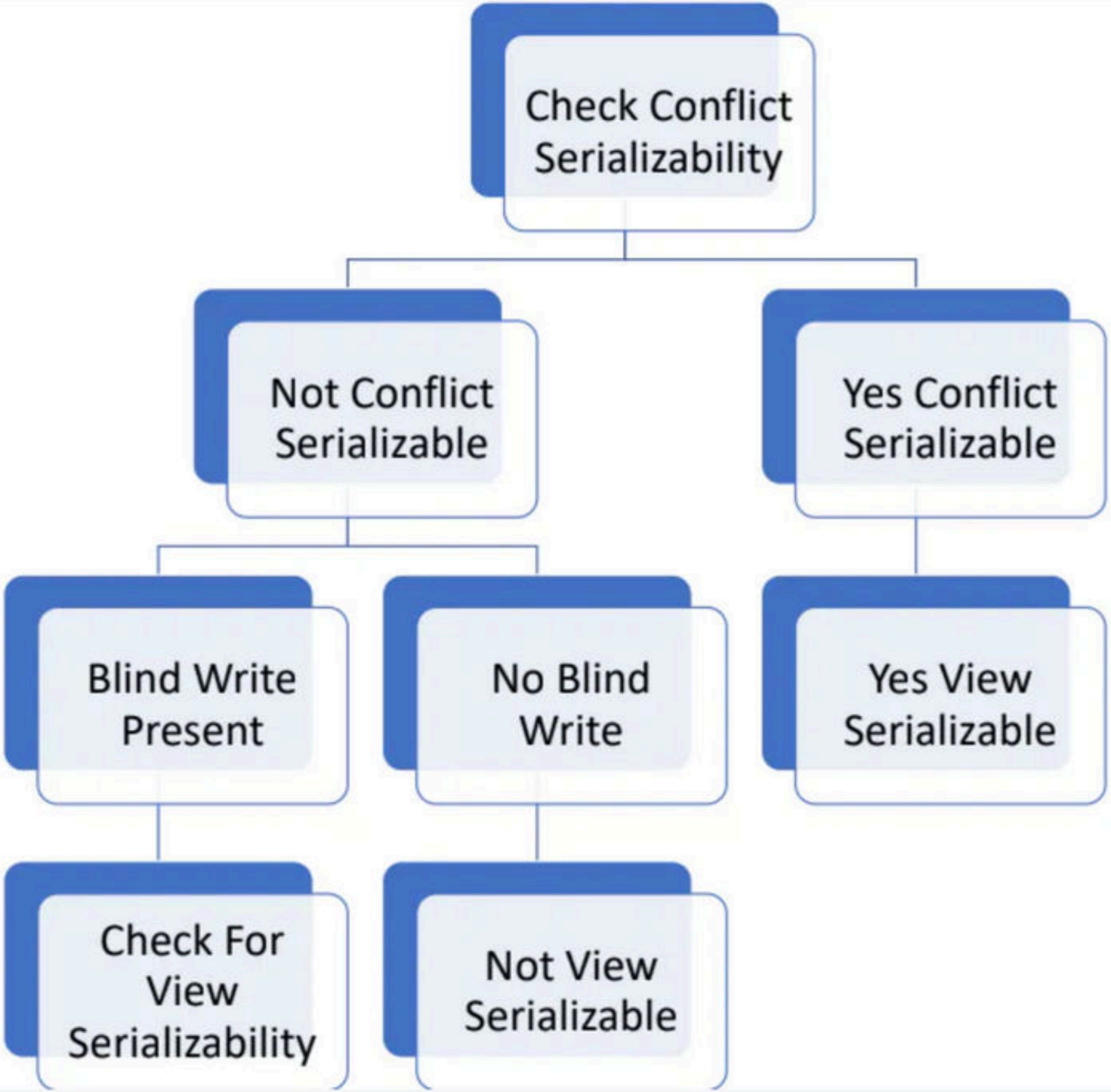
- If a schedule is conflict serializable then it will also be view serializable, so we must check view serializability only if a schedule is not conflict serializable.



- 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.


Check Conflict Serializability
Not Conflict Serializable
Yes Conflict Serializable
Blind Write Present
No Blind Write
Yes View Serializable
Check For View Serializability
Not View Serializable

- 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'.

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)

- 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.

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

View Serializable

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

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

- ***BLIND WRITES***- In the above example, transaction T_4 and T_6 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.**

Break

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 View serializable

d) None of the above

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 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

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

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

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

S			
T_1	T_2	T_3	T_4
R(X)			
	R(Y)		
	W(X)		
		W(Z)	
			R(Z)
		R(X)	
		W(Y)	
W(X)			
	W(Y)		
		W(X)	
			R(X)
			W(Y)
C			
	C		
		C	
			C

Break

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.

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
	C
C	

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.

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
C	
	C

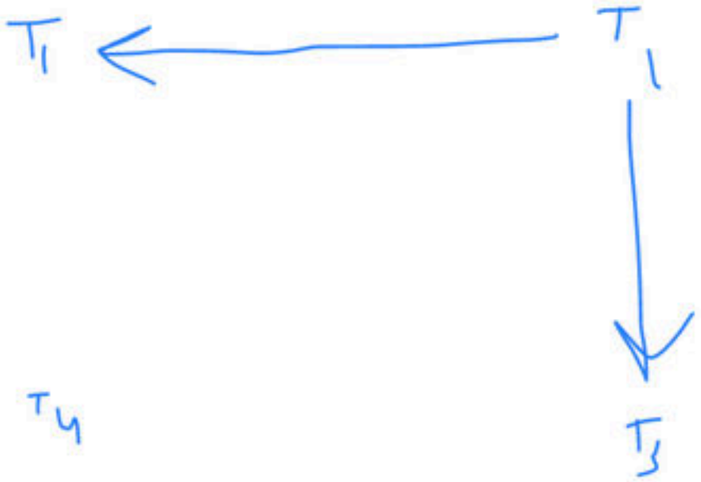
Q Consider the following schedule S of transactions T_1, T_2, T_3, T_4 **(GATE - 2014) (2 Marks)**
Which one of the following statements is CORRECT?

(A) S is conflict-serializable but not recoverable — 18

(B) S is not conflict-serializable but is recoverable — 15

(C) S is both conflict-serializable and recoverable — 61

(D) S is neither conflict-serializable nor is it recoverable — 4



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

Let S be the following schedule of operations of three transactions T_1, T_2 and T_3 in a relational database system:

(GATE-2021)

$$R_2(Y), R_1(X), R_2(Z), R_1(Y)W_1(X), R_2(Z), W_2(Y), R_3(X), W_3(Z)$$

T_1	T_2	T_3
	~~R(Y)~~	
~~R(X)~~		
		~~R(Z)~~
~~R(Y)~~		
~~W(X)~~		
	~~R(Z)~~	
	~~W(Y)~~	
		R(X)
		W(Z)

Consider the statements P and Q below:

- P: S is conflict-serializable.
- Q: If T_3 commits before T_1 finishes, then S is recoverable.

Which one of the following choices is correct?

A. ~~Both P and Q are true~~ 20
B. P is true and Q is false — 7
C. ~~P is false and Q is true~~ — 5
D. ~~Both P and Q are false~~ — 7

-66

$T_1 \rightarrow T_2 \rightarrow T_3$

$T_1 \rightarrow T_2$, $T_1 \rightarrow T_3$, $T_2 \rightarrow T_3$

Break

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.

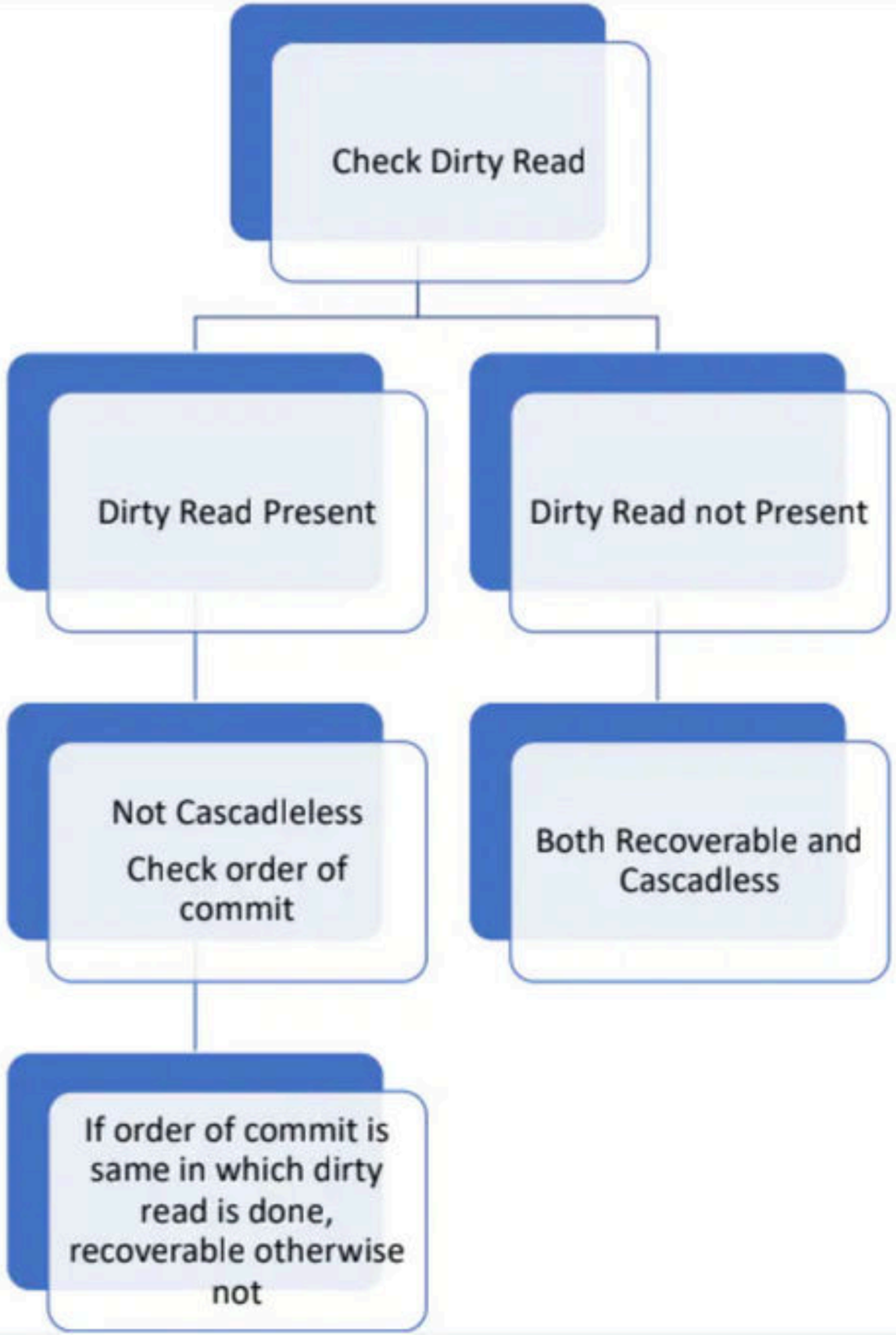
S		
T_1	T_2	T_3
R(X)		
W(X)		
	R(X)	
	W(X)	
		R(X)
C		
	C	
		C

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.

S		
T_1	T_2	T_3
R(X)		
W(X)		
C		
	R(X)	
	W(X)	
	C	
		R(X)
		C


Check Dirty Read
Dirty Read Present
Dirty Read not Present
Not Cascadleless
Check order of commit
Both Recoverable and Cascadless
If order of commit is same in which dirty read is done, recoverable otherwise not

Q Consider the following partial Schedule S involving two transactions T_1 and T_2. 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).

Suppose that the transaction T_1 fails immediately after time instance 9. Which one of the following statements is correct? **(GATE-2015) (2 Marks)**

(A) T_2 must be aborted and then both T_1 and T_2 must be re-started to ensure transaction atomicity — 8

(B) Schedule S is non-recoverable and cannot ensure transaction atomicity — 61

(C) Only T_2 must be aborted and then re-started to ensure transaction atomicity — 14

(D) Schedule S is recoverable and can ensure atomicity and nothing else needs to be done — 17

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)*	

Break

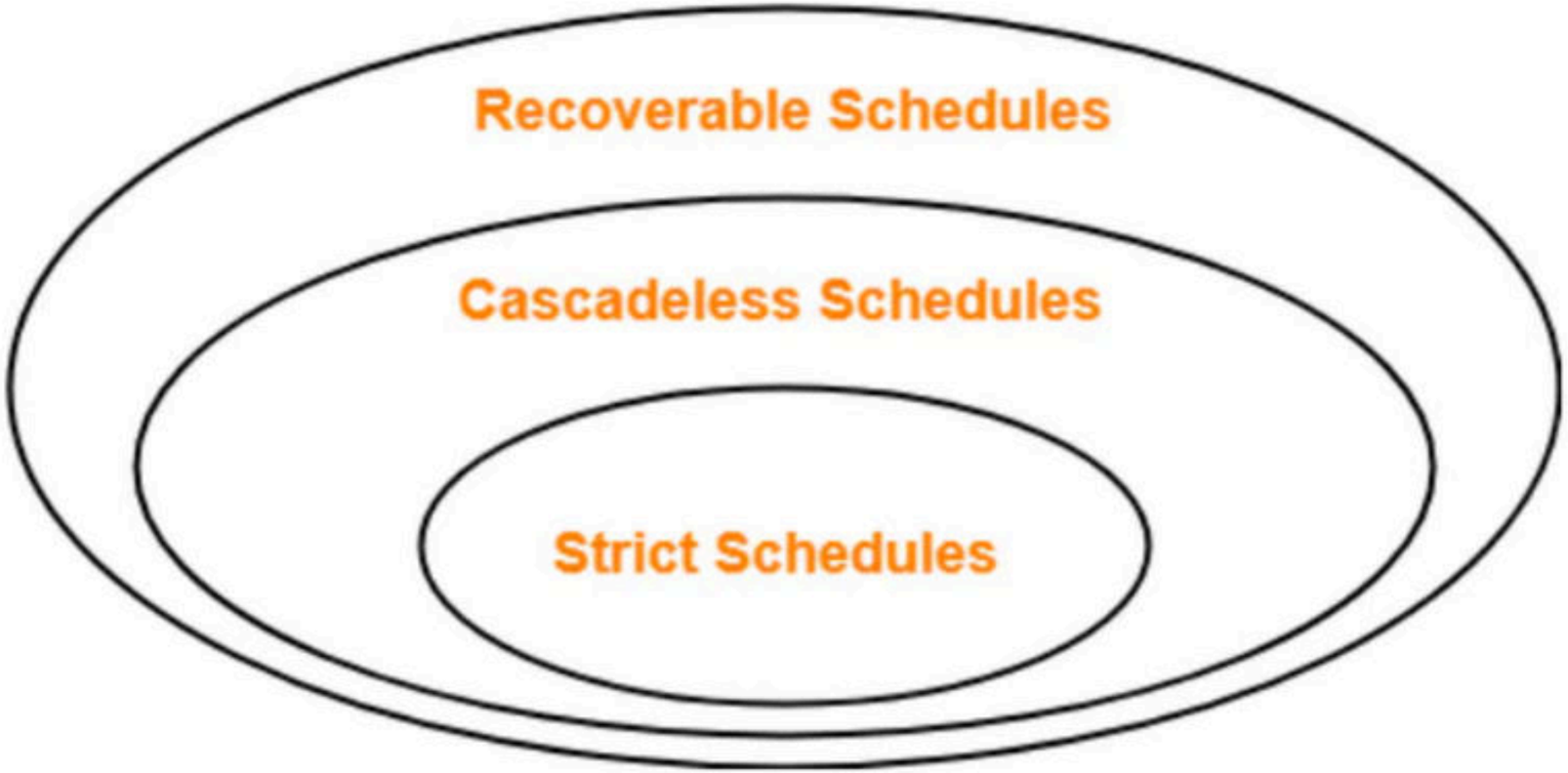
Strict Schedule

- 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.

S_1	
T_1	T_2
R(a)	
W(a)	
	W(a)
C	
	R(a)
	C

S_2	
T_1	T_2
R(a)	
W(a)	
C	
	W(a)
	R(a)
	C

S_3	
T_1	T_2
R(a)	
	R(b)
W(a)	
	W(b)
C	
	R(a)
	C


Recoverable Schedules
Cascadeless Schedules
Strict Schedules

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

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

S	
T_1	T_2
	R(x)
R(X)	
	R(Y)
W(X)	
R(Y)	
	W(X)
a_1	
	a_2

Q Consider the following schedules:

S_1: $R_1(x)$ $W_1(x)$ $R_1(y)$ $R_2(x)$ $W_2(x)$ C_2, C_1;

S_2: $R_2(x)$ $W_2(x)$ $R_1(y)$ $R_1(x)$ $W_2(x)$ C_2, C_1;

Which of the following is true?

a) Both S_1 and S_2 are recoverable

b) S_1 is recoverable but S_2 is not

c) S_2 is recoverable but S_1 is not

d) Both schedule are not Recoverable

Q Consider the following schedules:

S_1: $R_1(x)\ W_1(x)\ R_1(y)\ R_2(x)\ W_2(x)\ C_2,\ C_1;$

S_1	
T_1	T_2
R(x)	
W(x)	
R(y)	
	R(x)
	W(x)
	C
C	

S_2: $R_2(x)\ W_2(x)\ R_1(y)\ R_1(x)\ W_2(x)\ C_2,\ C_1;$

S_2	
T_1	T_2
	R(x)
	W(x)
R(y)	
R(x)	
	W(x)
	C
C	

Which of the following is true?

a) Both S_1 and S_2 are recoverable

b) S_1 is recoverable but S_2 is not

c) S_2 is recoverable but S_1 is not

d) Both schedule are not Recoverable

Q Consider the following schedule 'S'.

S: $r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); c_1; w_2(Z); w_3(Y); w_2(Y); c_3; c_2;$

The schedule 'S' is

a) Recoverable

b) Cascade less

c) recoverable and cascade-less

d) None

Q Consider the following schedule 'S'.

S: $r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); c_1; w_2(Z); w_3(Y); w_2(Y); c_3; c_2;$

The schedule 'S' is

a) Recoverable

b) Cascade less

c) recoverable and cascade-less

d) None

S		
T_1	T_2	T_3
R(X)		
	R(Z)	
		R(X)
R(Z)		
	R(Y)	
		R(Y)
W(X)		
C		
	W(Z)	
		W(Y)
	W(Y)	
		C
	C	

Q Consider the given schedule

$R_1(X)$, $R_2(Z)$, $R_1(Z)$, $R_3(X)$, $R_3(Y)$, $W_1(X)$, $Commit_1$, $W_3(Y)$, $Commit_3$, $R_2(Y)$, $W_2(Z)$, $W_2(Y)$, $Commit_2$. The given schedule is

a) Recoverable only

b) Cascade less only

c) recoverable and cascade-less

d) None

Q Consider the given schedule

$R_1(X)$, $R_2(Z)$, $R_1(Z)$, $R_3(X)$, $R_3(Y)$, $W_1(X)$, $Commit_1$, $W_3(Y)$, $Commit_3$, $R_2(Y)$, $W_2(Z)$, $W_2(Y)$, $Commit_2$. The given schedule is

a) Recoverable only

b) Cascade less only

c) recoverable and cascade-less

d) None

S		
T_1	T_2	T_3
R(X)		
	R(Z)	
R(Z)		
		R(X)
		R(Y)
W(X)		
C		
		W(Y)
		C
	R(Y)	
	W(Z)	
	W(Y)	
	C	

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 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

S		
T_1	T_2	T_3
	R(X)	
	W(X)	
		R(Y)
R(X)		
R(Y)		
W(X)		
		W(Y)
		R(X)
R(Y)		
		C
	C	
C		

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

Break

Q Two concurrent executing transactions T1 and T2 are allowed to update same stock item say 'A' in an uncontrolled manner. In such scenario. Following problems may occur:

(a) Dirty read problem
(b) Lost update problem
(c) Transaction failure
(d) Inconsistent database state

Which of the following option is correct if database system has no concurrency module and allows concurrent execution of above two transactions? **[Asked in L&T Infotech (LTI) 2021]**

A) (a), (b) and (c) only
B) (c) and (d) only
C) (a) and (b) only
D) (a), (b) and (d) only

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

(start, T_4);
(write, T_4, y, 2, 3);
(start, T_1);
(commit, T_4);
(write, T_1, z, 5, 7);
(checkpoint);
(start, T_2);
(write, T_2, x, 1, 9);
(commit, T_2);
(start, T_3);
(write, T_3, 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: T_3, T_1; Redo: T_2
(B) Undo: T_3, T_1; Redo: T_2, T_4
(C) Undo: none; Redo: T_2, T_4, T_3; T_1
(D) Undo: T_3, T_1, T_4; Redo: T_2

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

(start, T_4);
(write, T_4, y, 2, 3);
(start, T_1);
(commit, T_4);
(write, T_1, z, 5, 7);
(checkpoint);
(start, T_2);
(write, T_2, x, 1, 9);
(commit, T_2);
(start, T_3);
(write, T_3, z, 7, 2);

S			
T_1	T_2	T_3	T_4
			START
			(write, y, 2, 3)
START			
			COMMIT
(write, z, 5, 7)			
(checkpoint)	(checkpoint)	(checkpoint)	(checkpoint)
	START		
	(write, x, 1, 9)		
	COMMIT		
		START	
		(write, 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: T_3, T_1; Redo: T_2
(B) Undo: T_3, T_1; Redo: T_2, T_4
(C) Undo: none; Redo: T_2, T_4, T_3, T_1
(D) Undo: T_3, T_1, T_4; Redo: T_2

Suppose a database system crashes again while recovering from a previous crash. Assume checkpointing is not done by the database either during the transactions or during recovery. **(Gate-2021) (1 Marks)**

Which of the following statements is/are correct?

A. The same undo and redo list will be used while recovering again
B. The system cannot recover any further
C. All the transactions that are already undone and redone will not be recovered again
D. The database will become inconsistent

CONCURRENCY CONTROL

- Now we understood that if there is a schedule how to check whether it will work correctly or not i.e. weather it will maintain the consistency of the data base or not. (conflict serializability, view serializability, recoverability and cascade less)

- Now we will understand those protocol which guarantee how to design those schedules which ensure conflict serializability or other properties.

- There are different approach or idea to ensure conflict serializability which is the most important property.

- So first we must understand what is the possibility of conflict between two instruction and if somehow, we manage than the generated schedule will always be conflict serializable

- If we remember, two instructions are conflicting if and only if three things happen simultaneously

1. Belong to different transactions
2. Must operate on same data value
3. At least one of them should be a write instruction

- if we think sufficiently, there will be no way to change any of these three conditions. But actual problem is not that two instructions are trying to access same data base but, they are trying to do that at same time.

- So point if we somehow by any technique manage that two transaction do not access same data at same time then ensuring conflict serializability will be easy

- Now question is how to approach conflict serializability, there are Three popular approaches to go forwards.

- **Time stamping based method: -** where before entering the system, a specific order is decided among the transaction, so in case of a clash we can decide which one to allow and which to stop.

- **Lock based method:** - where we ask a transaction to first lock a data item before using it. So that no different transaction can use a data at the same time, removing any possibility of conflict.

  - 2 phase locking
    1. Basic 2pl
    2. Conservative 2pl
    3. Rigorous 2pl
    4. Strict 2pl

  - Graph based protocol

- **Validation based protocol –** Majority of transactions are read only transactions, the rate of conflicts among the transaction may be low, thus many of transaction, if executed without the supervision of a concurrency control scheme, would nerveless leave the system in a consistent state.

Goals of a Protocol: - We desire the following properties from schedule generating protocols

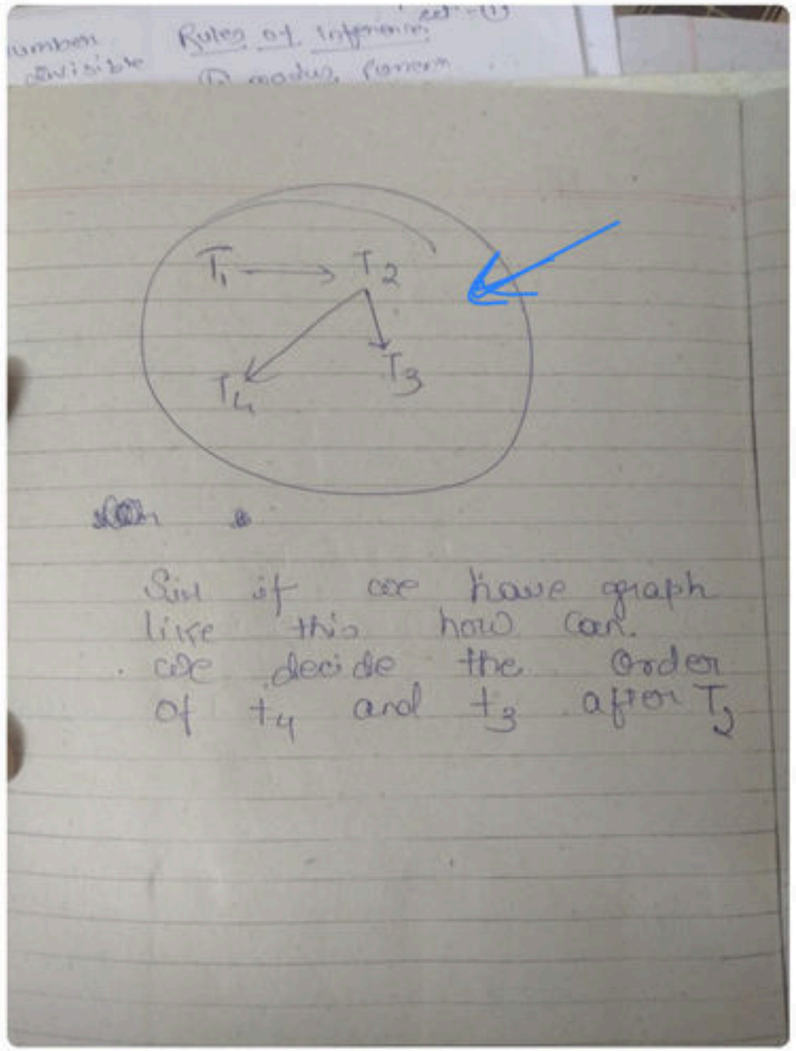
- Concurrency should be as high as possible, as this is our ultimate goal because of which we are making all the effort.
- The time taken by a transaction should also be less.
- Desirable Properties satisfied by the protocol
- Easy to understand and implement

Break

unacademy

▲ 2 • Asked by Aditya

Please help me with this doubt



$T_1 \to T_2 \to T_3 \to T_1$

$T_1 \to T_1 \to T_4 \to T_3$

TIME STAMP ORDERING PROTOCOL

- Basic idea of time stamping is to decide the order between the transaction before they enter in the system using a stamp (time stamp), in case of any conflict during the execution order can be decided using the time stamp.

- Let’s understand how this protocol works, here we have two idea of timestamping, one for the transaction, and other for the data item.

- Time stamp for transaction,
  - With each transaction t_i, in the system, we associate a unique fixed timestamp, denoted by $TS(t_i)$.
  - This timestamp is assigned by database system to a transaction at time transaction enters into the system.
  - If a transaction has been assigned a timestamp $TS(t_i)$ and a new transaction t_j , enters into the system with a timestamp $TS(t_j)$, then always $TS(t_i) < TS(t_j)$.

- Two things are to be noted

  1. First time stamp of a transaction remain fixed throughout the execution

  2. Second it is unique means no two transaction can have the same timestamp.

- The reason why we called time stamp not stamp, because for stamping we use the value of the system clock as stamp, advantage is,
  - Tt will always be unique as time never repeats
  - There is no requirement of refreshing and starting with fresh value.

- The time stamp of the transaction also determines the serializability order.

- Thus if $TS(t_i) < TS(t_j)$, then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction t_i appears before transaction t_j.

- Time stamp with data item, in order to assure such scheme, the protocol maintains for each data item Q two timestamp values:

  1. **W-timestamp(Q)** is the largest time-stamp of any transaction that executed write(Q) successfully.

  2. **R-timestamp(Q)** is the largest time-stamp of any transaction that executed read(Q) successfully.

- These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

- Suppose a transaction Ti request a ***read(Q)***

1. If **TS(Ti) < W-timestamp(Q),** then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.

2. If **TS(Ti)≥ W-timestamp(Q),** then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T_i).

- Suppose that transaction Ti issues ***write(Q).***
  1. If **TS(Ti) < R-timestamp(Q),** then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and Ti is rolled back.

  2. If **TS(Ti) < W-timestamp(Q),** then Ti is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and Ti is rolled back.

  3. If **TS(Ti) ≥ R-timestamp(Q),** then the write operation is executed, and W-timestamp(Q) is set to max(W-timestamp(Q), TS(Ti)).

  4. If **TS(Ti) ≥ W-timestamp(Q),** then the write operation is executed, and W-timestamp(Q) is set to max(W-timestamp(Q), TS(Ti)).

- If a transaction T_i is rolled back by the concurrency control scheme as a result of either a read or write operation, the system assigns it's a new timestamp and restarts it.

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering					
Thomas Write Rule					
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

Properties

- Time stamp ordering protocol ensures conflict serializability. Because conflicting operations are processed in timestamp order, since all the arcs in the precedence graph are of the form thus, there will be no cycles in the precedence graph.
- As we know that view is liberal form conflict so view serializability also holds good
- As there is a possibility of dirty read, and no restriction on when to commit, so can be irrecoverable and may suffer from cascading rollback.
- At the time of request, here either we allow or we reject, so there is no idea of deadlock.
- If a schedule is not conflict serializable then it is not allowed by time stamp ordering scheme.
- But it is not necessary that all conflict serializable schedule generated by time stamping.

Conclusion

- It may cause starvation to occur, as if a sequence of conflicting transactions causes repeated restarting of the long transaction, and then there is a possibility of starvation of long transaction.

- It is relatively slow as before executing every instruction we have to check conditions before. Time stamping protocol ensure that the schedule designed through this protocol will always be conflict serializable.

- This protocol can also be used for determining the serializability order (order in which transaction must execute) among the transaction in advance.

- Further modifications are possible if we want to ensure recoverability and cascade lessness, using different approaches
  - By performing all writes together at the end of the transaction, i.e. while writers are in progress, no transaction is permitted to access any of data items that have been written.
  - By using a limited form of locking, where by read of uncommitted items are postponed until the transaction that uploaded the item commit.
  - Recoverability alone can be ensured by tracking uncommitted writes and allowing a transaction t_i to commit only after the commit of any transaction that wrote a value that t_i read.

Break

THOMAS WRITE RULE

- Thomas write is an improvement in time stamping protocol, which makes some modification and may generate those protocols that are even view serializable, because it allows greater potential concurrency.

- It is a Modified version of the timestamp-ordering protocol in which Blind write operations may be ignored under certain circumstances.

- The protocol rules for read operations remain unchanged. while for write operation, there is slightly change in Thomas write rule than timestamp ordering protocol.

When T_i attempts to write data item Q,

- **if $TS(T_i) < W\text{-}timestamp(Q)$,** then T_i is attempting to write an obsolete value of {Q}. Rather than rolling back T_i as the timestamp ordering protocol would have done, this {write} operation can be ignored.

- This modification is valid as the any transaction with **TS(T_i) < W-timestamp(Q),** the value written by this transaction will never be read by any other transaction performing Read(Q) ignoring such obsolete write operation is considerable.

- Thomas' Write Rule allows greater potential concurrency. Allows some view-serializable schedules that are not conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule					
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

Q In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let $TS(T_1)$ and $TS(T_2)$ be the timestamps of transactions T_1 and T_2 respectively. Besides, T_1 holds a lock on the resource R and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp.

```
if TS(T2)<TS(T1) then
    T1 is killed
else T2 waits.
```

Assume any transaction that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks? **(GATE-2017) (2 Marks)**

(a) The database system is both deadlock-free and starvation-free

(b) The database system is deadlock-free, but not starvation-free

(c) The database system is starvation-free, but not deadlock-free

(d) The database system is neither deadlock-free nor starvation-free

Q Which of the following time stamp ordering protocol(s) allow(s) the following schedules?

S: $\mathbf{W_1(A);\ W_2(A);\ W_3(A);\ R_2(A)\ ;\ R_4(A);}$

Time stamps : $T_1 : 5,\ T_2 : 10,\ T_3 : 15;\ T_4 : 20$

a) Thomas write rule

b) Basic time stamp

Break

Lock Based Protocols

- To ensure isolation is to require that data items be accessed in a mutually exclusive manner i.e. while one transaction is accessing a data item, no other transaction can modify that data item. Locking is the most fundamental approach to ensure this.

- Lock based protocols ensure this requirement. Idea is first obtain a lock on the desired data item then if lock is granted then perform the operation and then unlock it.

- In general, we support two modes of lock because, to provide better concurrency.

- ***<u>Shared mode</u>***
  - If transaction T_i has obtained a shared-mode lock (denoted by S) on any data item Q, then T_i can read, but cannot write Q, any other transaction can also acquire a shared mode lock on the same data item(this is the reason we called this shared mode).

- **<u>Exclusive mode</u>**
  - If transaction T_i has obtained an exclusive-mode lock (denoted by X) on any data item Q, then T_i can both read and write Q, any other transaction cannot acquire either a shared or exclusive mode lock on the same data item. (this is the reason we called this exclusive mode)

Lock –Compatibility Matrix

- Conclusion shared is compatible only with shared while exclusive is not compatible either with shared or exclusive.

- To access a data item, transaction T_i must first lock that item, if the data item is already locked by another transaction in an incompatible mode, or some other transaction is already waiting in non-compatible mode, then concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. The lock is then granted.

Current State of lock of data items

Requested Lock	Exclusive	Shared	Unlocked
Exclusive	N	N	Y
Shared	N	Y	Y
Unlock	Y	Y	-

- Lock based protocol *do not ensure serializability* as granting and releasing of lock do not follow any order and any transaction any time may go for lock and unlock. Here in the example below we can see, that even this transaction in using locking but neither it is conflict serializable nor independent from deadlock.

T_1	T_2
LOCK-X(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK-S(B)
	READ(B)
	UNLOCK(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(A)

- If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states, as there exists a possibility of dirty read. On the other hand, if we do not unlock a data item before requesting a lock on another data item, concurrency will be poor.

- We shall require that each transaction in the system follow a set of rules, called a **locking protocol**, indicating when a transaction may lock and unlock each of the data items for e.g. 2pl or graph based locking.

- Locking protocols restrict the number of possible schedules.

Break

Two phase locking protocol(2PL)

- The protocol ensures that each transaction issue lock and unlock requests in two phases, note that each transaction will be 2 phased not schedule.

- **Growing phase**- A transaction may obtain locks, but not release any locks.

- **Shrinking phase**- A transaction may release locks, but may not obtain any new locks.

- Initially a transaction is in growing phase and acquires lock as needed and in between can perform operation reach to lock point and once a transaction releases a lock, it can issue no more lock requests i.e. it enters the shrinking phase.

T_1	T_2
LOCK-X(A)	
READ(A)	
WRITE(A)	
	LOCK-S(B)
	READ(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(B)
UNLOCK(A)	
UNLOCK(B)	
	UNLOCK(A)

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

Properties

- 2PL ensures conflict serializability, and the ordering of transaction over lock points is itself a serializability order of a schedule in 2PL.
- If a schedule is allowed in 2PL protocol then definitely it is always conflict serializable. But it is not necessary that if a schedule is conflict serializable then it will be generated by 2pl. Equivalent serial schedule is based on the order of lock points.
- View serializability is also guaranteed.
- Does not ensure freedom from deadlock
- May cause non-recoverability.
- Cascading rollback may occur.

Q Which of the following concurrency protocol ensures both conflict serializability and freedom from deadlock? **(NET-JUNE-2015) (NET-JUNE-2014) (GATE-2010) (2 Marks)**

(1) 2 - phase Locking

(2) Time stamp - ordering

(a) Both (1) and (2)

(b) (1) only

(c) (2) only

(d) Neither (1) nor (2)

Q Which of the following is correct? **(NET-DEC-2014)**

I. Two phase locking is an optimistic protocol.

II. Two phase locking is pessimistic protocol

III. Time stamping is an optimistic protocol.

IV. Time stamping is pessimistic protocol.

(A) I and III **(B)** II and IV **(C)** I and IV **(D)** II and III

Q Which of the following statements is wrong? **(NET-JUNE-2007)**

(A) 2-phase Locking Protocols suffer from deadlocks

(B) Time-Stamp Protocols suffer from more aborts

(C) Time-Stamp Protocols suffer from cascading roll back where as 2-Phase locking Protocol do not

(D) None of these

Q Two phase protocol in a database management system is: **(NET-DEC-2006)**

(A) a concurrency mechanism that is not deadlock free

(B) a recovery protocol used for restoring a database after a crash

(C) Any update to the system log done in 2-phases

(D) not effective in Database

Break

Conservative 2PL

- The idea is there is no growing phase transaction start directly from lock point, i.e. transaction must first acquire all the required locks then only it can start execution. If all the locks are not available then transaction must release the acquired locks and must wait.

  - Shrinking phase will work as usual, and transaction can unlock any data item anytime.

  - we must have a knowledge in future to understand what is data required so that we can use it

Q In conservative two phase locking protocol, a transaction

a) Should release exclusive locks only after the commit operation

b) Should release all the locks only at beginning of the transaction

c) should acquire all the locks at beginning of the transaction

d) Should acquire all the exclusive locks at beginning transaction

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

RIGOROUS 2PL

- Requires that all locks be held until the transaction commits.
- This protocol requires that locking be two phase and also all the locks taken be held by transaction until that transaction commit.
- Hence there is no shrinking phase in the system.

Q In a Rigorous 2 phase protocol

a) All shared locks held by the transaction are related after the transaction is committed

b) All exclusive locks held by the transaction are released after the transaction is committed

c) All locks held by the transaction are released after the transaction is committed

d) All locks held by the transaction are released before the transaction is committed

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL					
Strict 2PL					

STRICT 2PL

- that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.

- This protocol requires that locking be two phase and also that exclusive –mode locks taken by transaction be held until that transaction commits.

- So it is simplified form of rigorous 2pl

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL	YES	YES	YES	YES	NO
Strict 2PL					

Q Consider the following two statements about database transaction schedules:

I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.

II. Timestamp-ordering concurrency control protocol with Thomas Write Rule can generate view serializable schedules that are not conflict serializable.

Which of the above statements is/are TRUE? **(GATE-2019) (1 Marks)**

(a) Both I and II

(b) Neither I nor II

(c) II only

(d) I only

Q Consider the following two-phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects$\{O_1,\ldots,O_k\}$. This is done in the following manner:

Step1. T acquires exclusive locks to $O_1,\ldots,O_k$ in increasing order of their addresses.

Step2. The required operations are performed.

Step3. All locks are released.

This protocol will **(GATE- 2016) (1 Marks)**

(a) guarantee serializability and deadlock-freedom

(b) guarantee neither serializability nor deadlock-freedom

(c) guarantee serializability but not deadlock-freedom

(d) guarantee deadlock-freedom but not serializability

Q Which of the following statement is/are correct

a) Every conflict serializable schedule allowed under 2PL protocol is allowed by basic time stamping protocol.

b) Every schedule allowed under basic time stamping protocol is allowed by Thomas-write rule

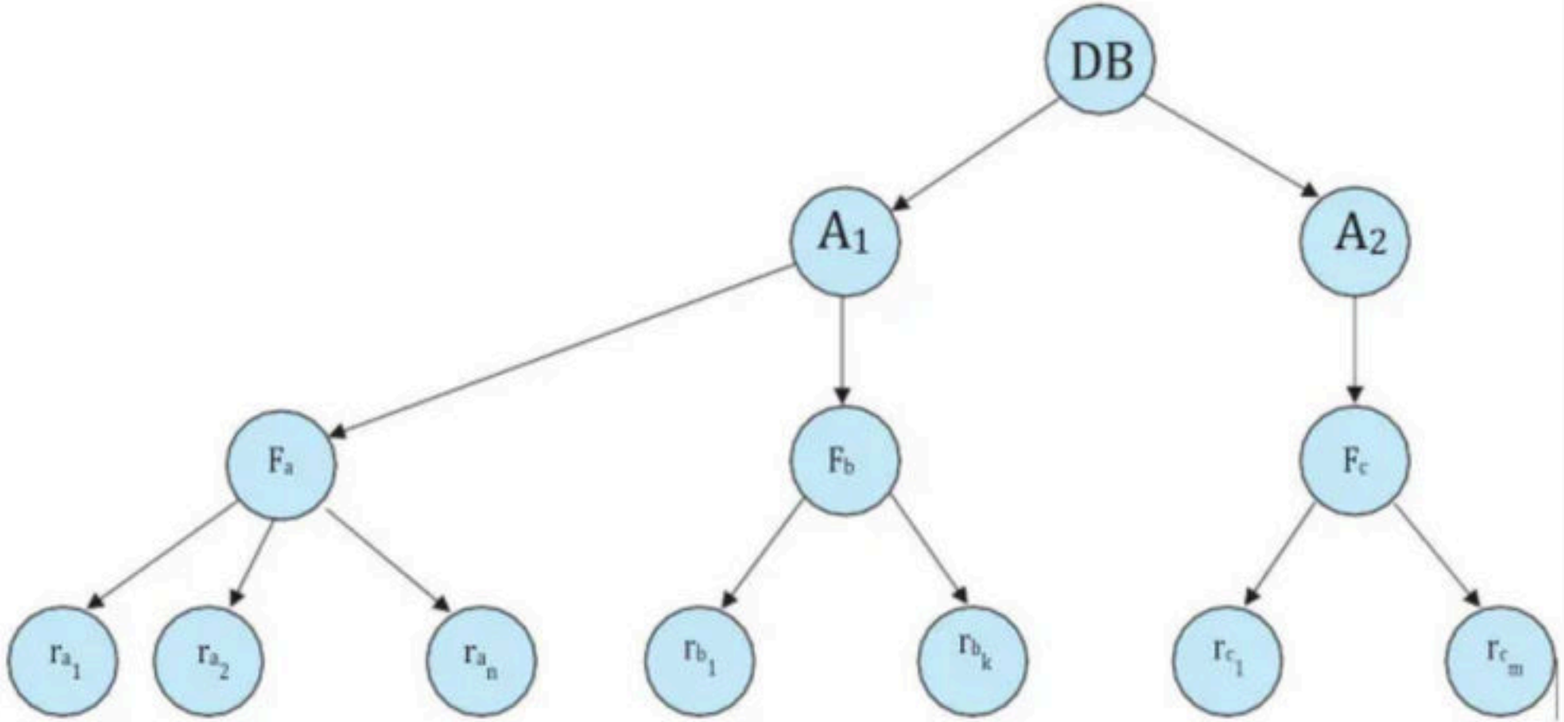
c) Every schedule allowed under Thomas-write rule is allowed by basic time stamping protocol

d) none

Break

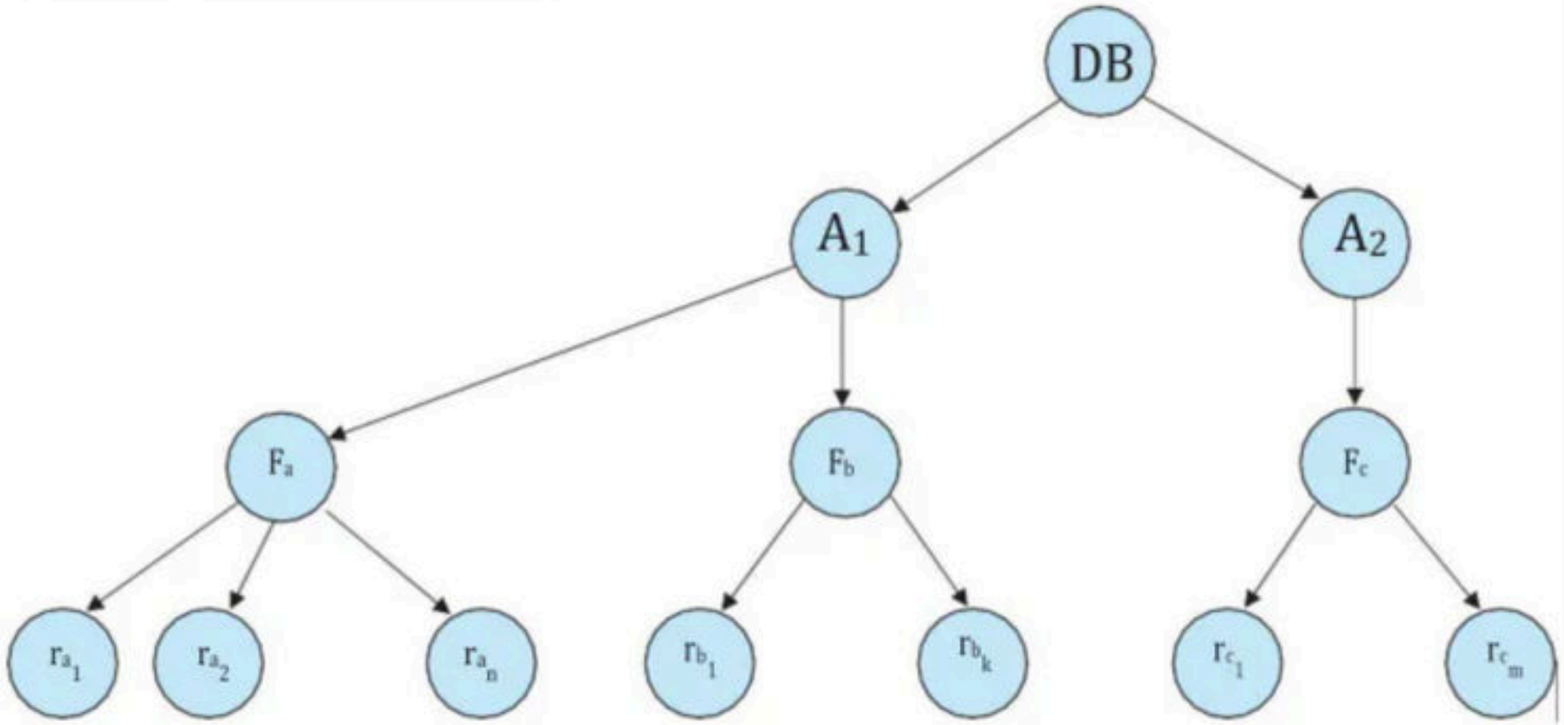
Multiple Granularity

	IS	IX	S	X
IS				
IX				
S				
X				



Multiple Granularity

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



Break

Validation-Based Protocols

- In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low.
- Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state.
- A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead.
- A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for *monitoring* the system.

- The **validation protocol** requires that each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order:

- **Read phase**. During this phase, the system executes transaction T_i. It reads the values of the various data items and stores them in variables local to T_i. It performs all write operations on temporary local variables, without updates of the actual database.

- **Validation phase**. The validation test (described below) is applied to trans- action T_i. This determines whether T_i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction.

- **Write phase**. If the validation test succeeds for transaction T_i, the temporary local variables that hold the results of any write operations performed by T_i are copied to the database. Read-only transactions omit this phase.