E R Diagram - Part I

Complete Course on DBMS for GATE

Sanchit Jain • Lesson 9 • Mar 28, 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 12, 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

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

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.

$T_1 \rightarrow$ 24/10/2001

$T_2 \rightarrow 1$ 1/11/2003

$T_3 \rightarrow 2$

$T.S(T_i) < T.S(T_j)$

- Time stamp for transaction,

1/1/12:35

  - 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 $TS(t_i) < TS(t_j)$.

T.S (T_i)

T = R := A

F = R ≠ A

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

    2. **<u>R-timestamp(Q)</u>** 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.

WTS(Q) = ~~NIL~~ 3:05

(Q)

RTS(Q) = NIL = 3:15

- 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).

T.S (T_i) = 5	T.S (T_x) = 10
// R(Q)	
	W(Q)
R(Q)	

T.S (T_i) = 10	T.S (T_x) = 5
	W(Q)
R(Q)	

T.S (T_i) = 10	T.S (T_x) = 10
W(Q)	~~W(Q)~~
R(Q)	

- 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					

$TS(T_1)=1$	$TS(T_2)=2$
R(Q)	
W(Q)	
	R(Q)

C

T = UP → 63

F = 60 → 37

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,

5 10

- **if TS(T_i) < W-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.

$TS(T_i)=5$	$WTS(Q)=10$
$W(Q)=10$	$W(Q)=101$

Should

Thus

$TS(T_i)=5$	$TS(Q)=10$
~~$W(Q)$~~	$W(Q)=101$

Rule

- 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.

5 10

if $TS(T_2)<TS(T_1)$ then
T_1 is killed
else T_2 waits.

$T_1 \leftarrow X \leftarrow T_2$

FCFS

T_2 T_3

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 — 31

(b) The database system is deadlock-free, but not starvation-free — 45

(c) The database system is starvation-free, but not deadlock-free — 11

(d) The database system is neither deadlock-free nor starvation-free — 11

T_6 T_5 T_4 T_3 $\leftarrow T_2$ T_1

T_3

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

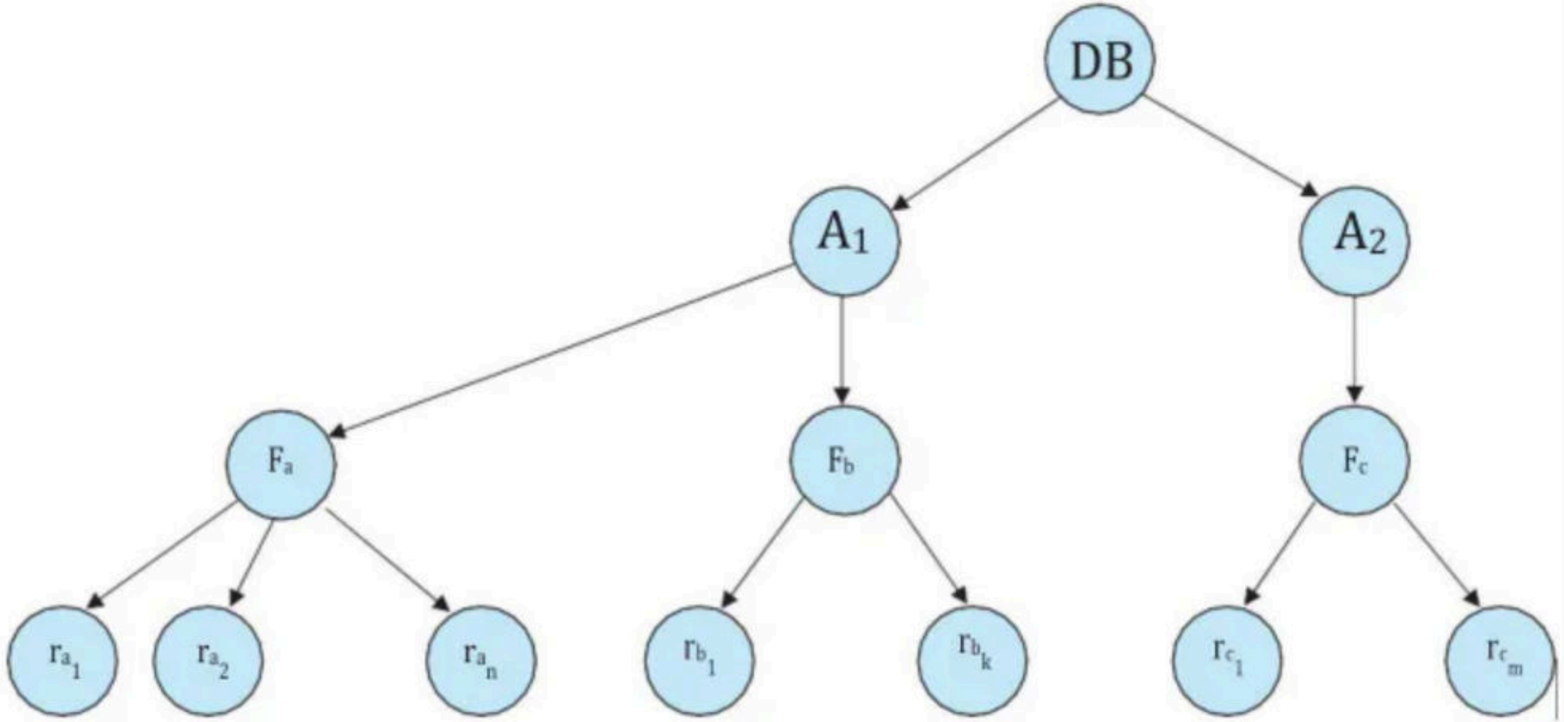
c) Every schedule allowed under Thomas-write rule is allowed by basic time stamping protocol

d) none

Break

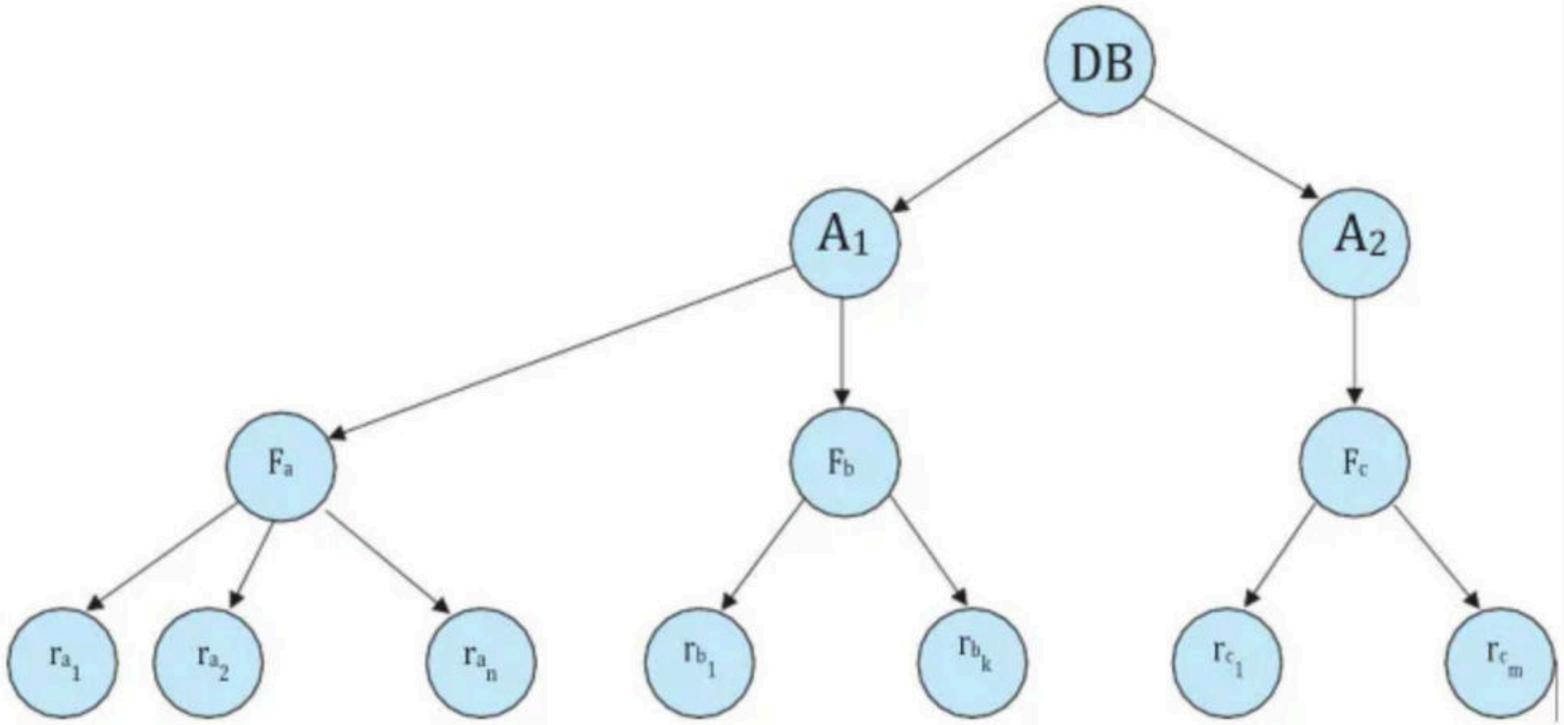
Multiple Granularity

	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.