UNIT - IV

NITISH SHARMA

1

TRANSACTION

A transaction is a collection of operations that forms a single logical unit of work.

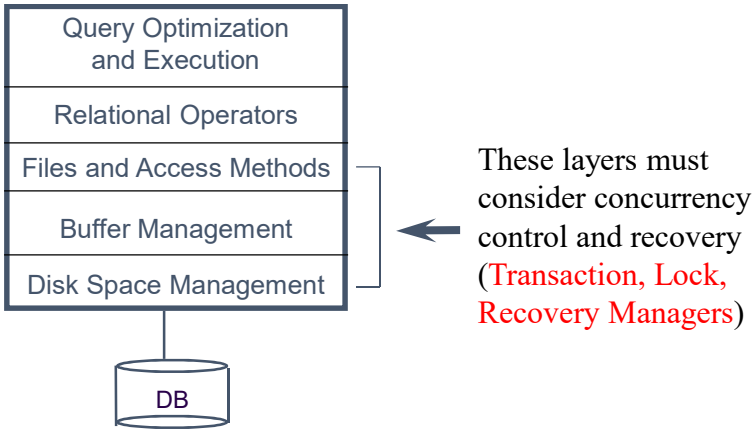
2

Transaction
Any action that reads from and/or writes to a database may consist of:

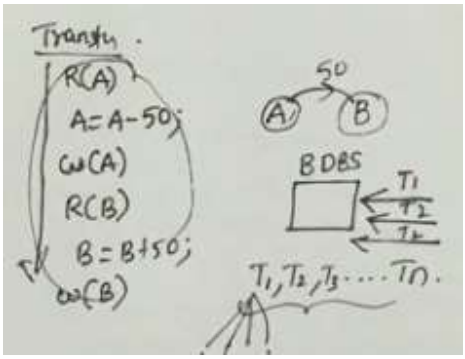
- Simple SELECT statement to generate a list of table contents.
- A series of related UPDATE statements to change the values of attributes in various tables.
- A series of INSERT statements to add rows to one or more tables.
- A combination of SELECT, UPDATE, and INSERT statements.

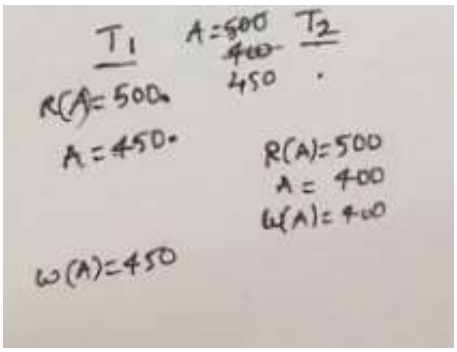
3

Structure of a DBMS



4





5

ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

6

Atomicity (All or none) -------- Transaction manager

- Transactions are atomic – they don't have parts (conceptually).
- can't be executed partially.

Consistency (Correctness) ------- User/Application program

- Transactions take the database from one consistent state into another
- In the middle of a transaction the database might not be consistent

7

Isolation ---------- Concurrency control manager

- The effects of a transaction are not visible to other transactions until it has completed.
- Each transaction must be executed without knowing what is happening with other transactions.

Durability ---------- Recovery manager

- Once a transaction has completed, its changes are made permanent
- Even if the system crashes, the effects of a transaction must remain in place

8

Example of transaction

- Transfer £50 from account A to account B

```
Read(A)
A = A - 50
Write(A)
Read(B)
B = B+50
Write(B)
```

transaction

Atomicity - shouldn't take money from A without giving it to B

Consistency - money isn't lost or gained

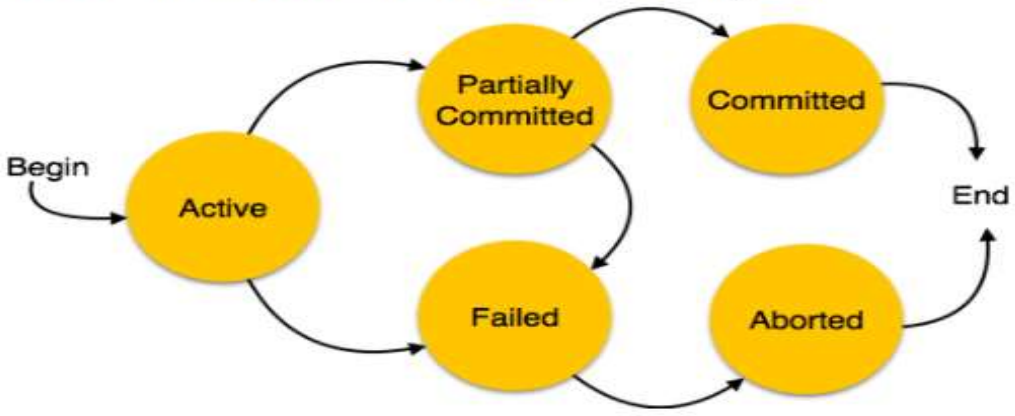
Isolation - other queries shouldn't see A or B change until completion

Durability - the money does not go back to A

9

States of Transactions

A transaction in a database can be in one of the following states —



10

Active state – Initial state

Any transaction will be in this state during its execution. Once the transaction starts executing from the first instruction begin transaction, the transaction will be considered in active state. During this state it performs operations READ and WRITE on some data items.

From active state, a transaction can go into one of two states, a partially committed state or a failed state.

Partially committed state – After the execution of final statement

This is the state of a transaction that successfully executing its last instruction. That means, if an active transaction reaches and executes the COMMIT statement, then the transaction is said to be in partially committed state.

From partially committed state, a transaction can go into one of two states, a committed state or a failed state

11

Committed – After successful completion of transaction

At partially committed state the database recovery system will perform certain actions to ensure that a failure at this stage should not cause lose of any updates made by the executing transaction. If the current transaction passed this check, then the transaction reaches committed state.

From committed state, a transaction can go into terminated state.

Failed – If any failure occurs

While a transaction is in the active state or in the partially committed state, the issues like transaction failure, user aborting the transaction, concurrency control issues, or any other failure, would happen. If any of these issues are raised, then the execution of the transaction can no longer proceed. At this stage a transaction will go into a failed state.

From failed state, a transaction can go into only aborted state.

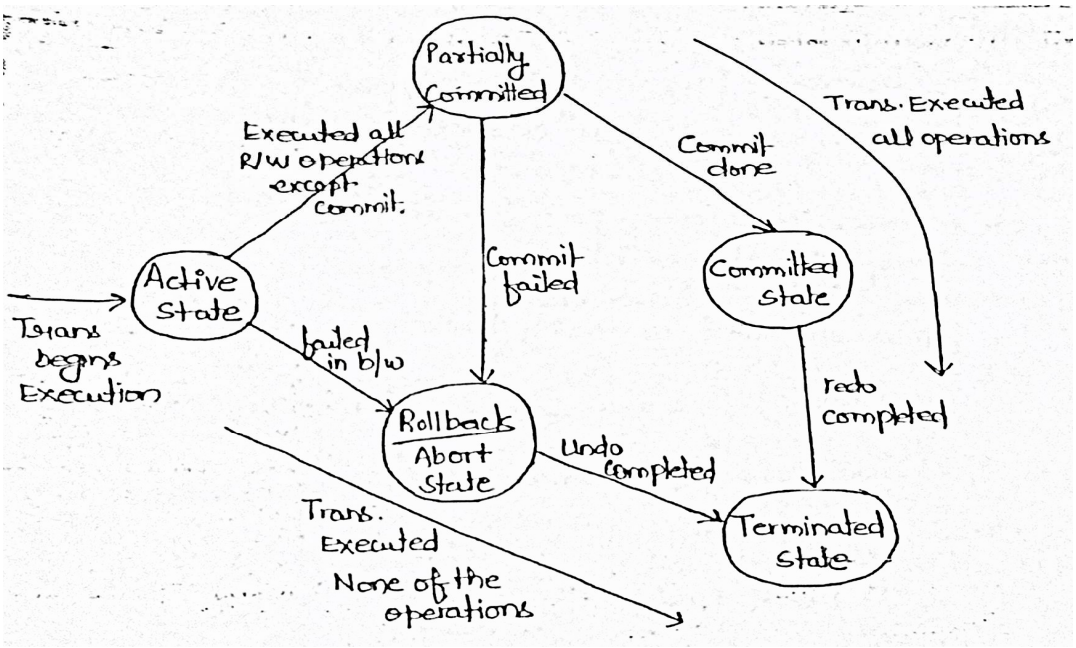
12

Aborted – After rolled back to the old consistent state.

After the failed state, all the changes made by the transaction has to be rolled back and the database has to be restored to its state prior to the start of the transaction. If these actions are completed by the DBMS then the transaction considered to be in aborted state.

From aborted state, a transaction can go into terminated state.

13



14

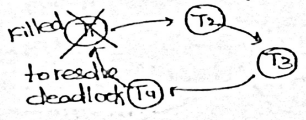
Possibility of Trans Failure :-

1) Power Failure

2) S/w Crash
   a) OS Restarted
   b) DBMS s/w Restarted

3) OS/DBMS concurrency control components may kill the transaction.

[Trans. are killed in case of deadlock.



4) H/w Crash
   [Disk Crash]

15

(*) Concurrency Control: The DB should remain consistent in case of concurrent trans.

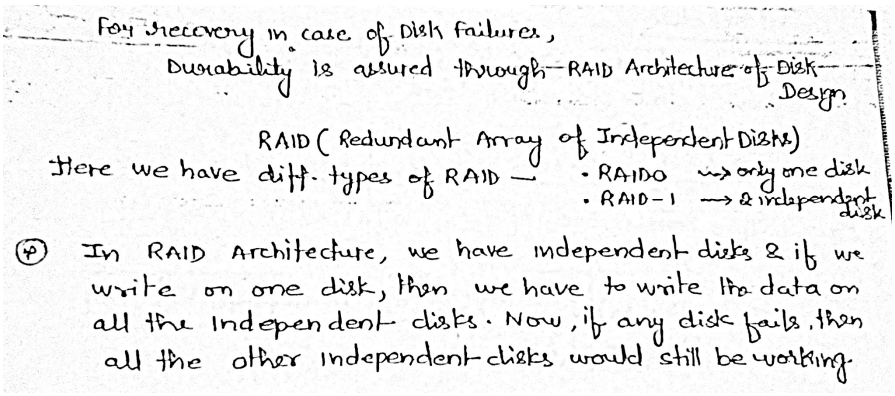
To maintain consistency, the system must be able to recover from any failure.

(*) Power failure
S/w crash
OS/DBMS Concurrency control kills transaction

↕ Can be recovered from log file stored in the disk.

But if disk crashes (where all log files are stored), then it is the duty of Recovery Mgmt. to maintain data (This comes under Durability).

16



17

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

•**Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

•**Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first.

When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other.

18

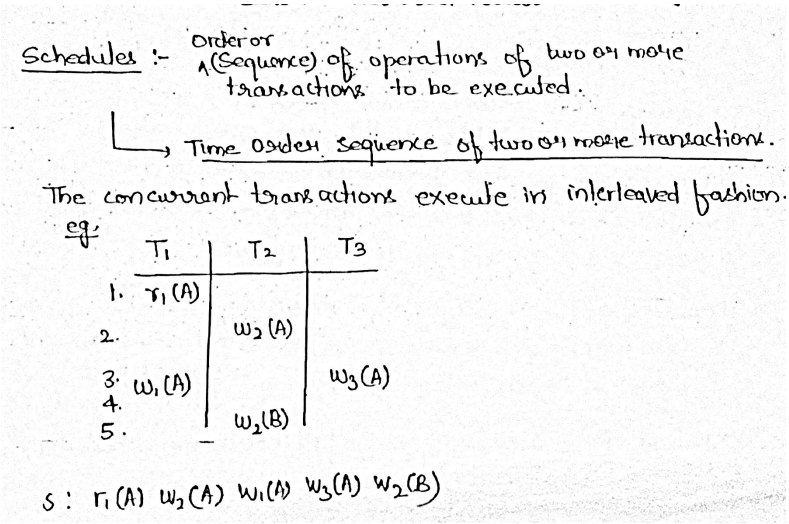
In a multi-transaction environment, serial schedules are considered as a benchmark.

The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion.

This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the **same data**, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

19



20

Serial Schedule

After termination of current trans. then only allowed to begin other trans.

eg T_1 ⇒ Transfer 500 from A to B $[R_1(A)\ W_1(A)\ R_1(B)\ W_1(B)]$

T_2 ⇒ disp total balance of A $[R_2(A)\ R_2(B)\ display(A+B)]$

21

T_1	T_2
2000: $R_1(A)$	
1500: $W_1(A)$	
3000: $R_1(B)$	
3500: $W_1(B)$	
	$R_2(A)$: 1500
	$R_2(B)$: 3500
	disp(A+B): 5000 (correct)

$T_1 \rightarrow T_2$ serial schedule

Advantage

Result of every serializable schedule is correct output (i.e. no inconsistency possible if transactions executing serially) because at a time only one transaction is accessing all the resources.

Disadvantage - Less degree of concurrency
Less throughput (No. of transactions executed per unit time)

To avoid these disadv., we go for concurrent schedules.

22

Concurrent Schedules –

Trans allowed to execute in non serial order

[Simultaneous / concurrent / Interleaved order]

(*) Every serial schedule is also concurrent schedule.

Drawback: Concurrent Execution may cause inconsistency.

T_1	T_2
2000: $R_1(A)$	
1500: $W_1(A)$	
	$R_2(A)$: 1500
	$R_2(B)$: 3000
	Disp(A+B): 4500 Incorrect
3000 $R_1(B)$	
3500 $W_1(B)$	

23

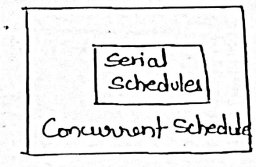
Serializable Schedule

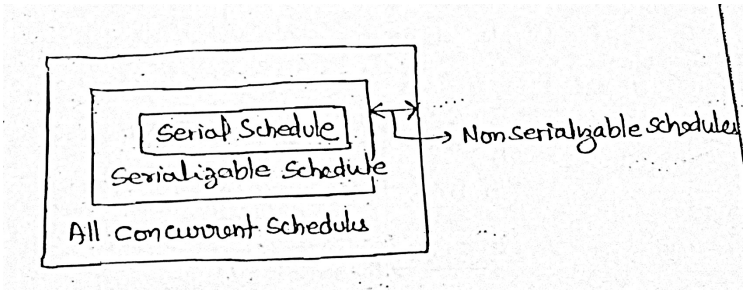
Schedule for which concurrent execution of 2 or more transactions is equal to result of any serial schedule are called searializable schedules.

Isolation ≡ Serializability

All serial schedules are serializable

concurrent schedules may / may not be serializable.





24

Q. $T_1, T_2, T_3 \ldots T_n$ are n Trans. # of possible serial schedules:

$n!$

Q. T_1: $R_1(A)$ $W_1(A)$ $R_1(B)$ $W_1(B)$

T_2: $R_2(A)$ $R_2(B)$

# of concurrent schedules b/w T_1 & T_2 : ${}^6C_4 * {}^2C_2 = 15$

13 Non Serial + 2 Serial

Q. T_1 & T_2 trans. with n & m operations respectively

# of Concurrent Schedules $= {}^{n+m}C_n = \frac{(n+m)!}{n!\,m!}$

Q. T_1, T_2, T_3 trans with n, m, p operations respectively

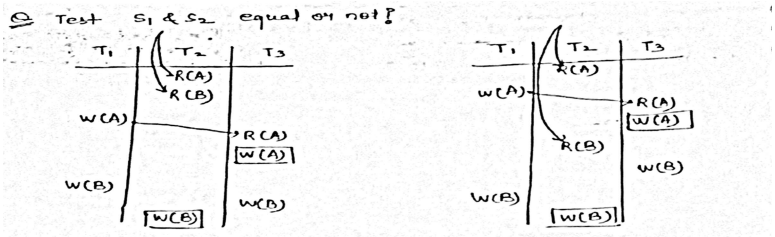
# of concurrent schedule $= {}^{n+m+p}C_n \cdot {}^{m+p}C_n$

$= \frac{(n+m+p)!}{n! \cdot m! \cdot p!}$

Includes Serial Schedules also.

25

Q. Test S_1 & S_2 equal or not?



Initial Read :-

S_1: R(A), R(B)] from initial DB

S_2: R(A), R(B)] from initial DB

Updated Read :-

S_1: T_1 W(A) → T_3 R(A)

S_2: T_1 W(A) → T_3 R(A)

Final Write :-

S_1: T_2 W(B), T_3 W(A)

S_2: T_2 W(B), T_3 W(A)

$S_1 = S_2$

or S_1 & S_2 are view equal-schedules.

26

Problems because of Concurrent Execution

W R Problem
R W Problem
W W Problem

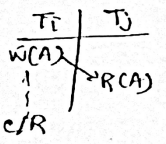
} Result of schedule can be incorrect becoz simultaneous Read-Write/ Write-Read/ Write-Write operations over same data from different transaction.

WR Problem

a) Non-serializable schedule

(and)

b) simultaneous WR exists

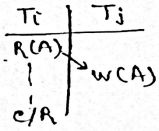


RW Problem

a) Non serializable schedule

(and)

b) simultaneous RW exists

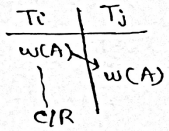


WW Problem

a) Non serializable schedule

(and)

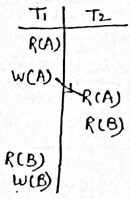
b) simultaneous WW exists



27

Q. Test whether the given schedule contains what suffers from which problem ?

i)



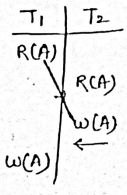
I) Not Serializable

II) Simultaneous WR Problem exist

⇓

WR Problem

ii)



I) Not Serializable

II) RW conflict

Not W-W problem as T_2 can commit here.

⇓

(RW Problem)

28

Conflict Equivalence

Two schedules would be conflicting if they have the following properties −

- Both belong to separate transactions.
- Both accesses the same data item.
- At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

29

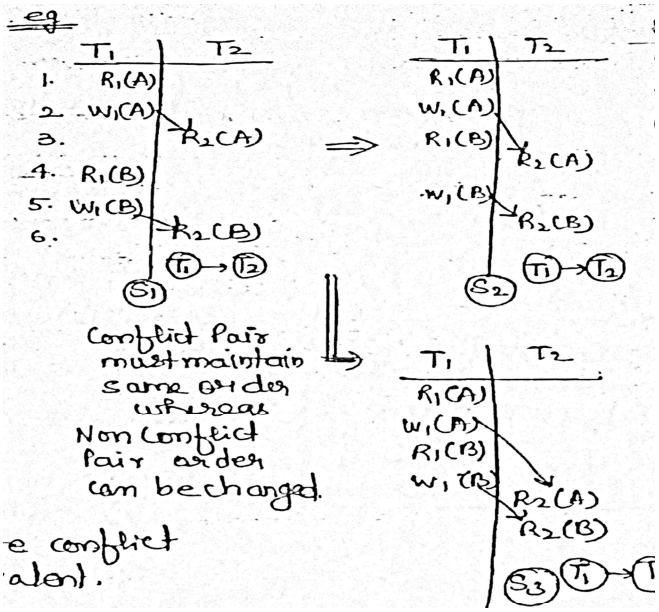
Conflict Equal Schedules

S_1 & S_2 schedules are conflict equal iff S_2 schedule derived by exchanging of consecutive non conflict pairs of schedule (S_1).

$[S_1]$ ⟷ (by exchanging Consecutive Non Conflict Pairs) $[S_2]$

30

eg



31

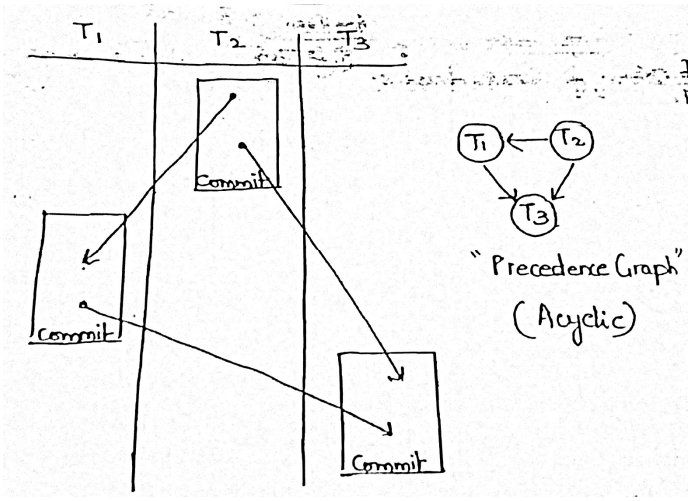
Conflict Serializable Schedule :-

Schedule (S) is conflict serializable schedule iff there exists some serial schedule (S') that must be conflict equal to schedule (S).

[Acyclic Precedence Graph] ← S → Conflict Equal → S' (serial schedule) [Acyclic Precedence Graph]

then conflict serializable schedule.

32



33

IRRECOVERABLE SCHEDULE:- Transaction Ti Reads 'A' which is modified by some other Trans Tj and Ti commit before commit / rollback of Tj.

Tj	Ti
W(A)	
|	R(A)
|	commit
C/R	

RECOVERABLE SCHEDULE:- schedule (S) is recoverable iff

a) No Uncommitted Reads in schedule (S)

(OR)

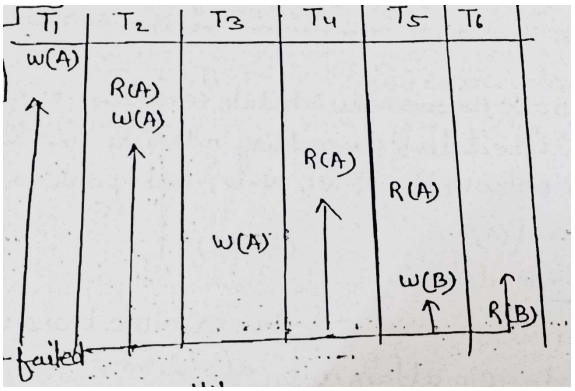
b) If Ti reads A which is modified by Tj then commit of Ti must delay until commit or rollback of Tj.

Tj	Ti
W(A)	
|	R(A): Dirty Read
C/R	(wait)
	Commit

34

Cascading Rollback Problem

Becoz of failure some trans forced to Rollback some set of other transactions.



35

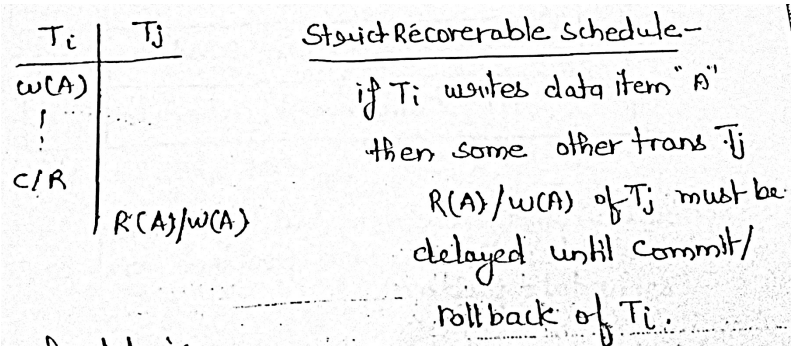
Lost Update Problem

This problem occurs if two or more trans writing same data simultaneously.

Ti	Tj
W(A)	
|	W(A)
|	
C/R	

If Ti writes (A)
Then Tj also writes A
before Ti C/R. Then
Lost Update Problem can occur.

36



37

Concurrency Control

Lock based protocol:

This protocol requires that all the data items must be accessed in a mutually exclusive manner, i.e. when one transaction is executing then no other transaction should interrupt the same object.

To implement lock based protocol we need 2 types of locks.

a. **Shared Lock:** When we take this lock we can just read the item but cannot write.

b. **Exclusive Lock:** In this type of lock we can write as well as read the data item.

Below table will give clear idea about what we can do and cannot while having shared or exclusive lock.

	Shared	Exclusive
Shared	Yes	No
Exclusive	No	No

38

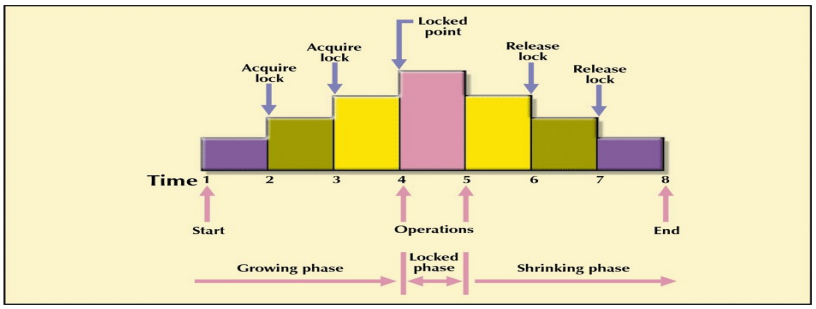
What is Two-Phase Locking (2PL)?

- Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.
- It ensures conflict serializable schedules.
- If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.

This protocol can be divided into two phases,

1. In Growing Phase, a transaction obtains locks, but may not release any lock.

2. In Shrinking Phase, a transaction may release locks, but may not obtain any lock.



39

Following are the types of two – phase locking protocol:

1. Strict Two – Phase Locking Protocol
2. Rigorous Two – Phase Locking Protocol
3. Conservative Two – Phase Locking Protocol

1. Strict Two-Phase Locking Protocol

- Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
- This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.
- It is not deadlock free.
- It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.
- Most of the database systems implement rigorous two – phase locking protocol.

2. Rigorous Two-Phase Locking

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

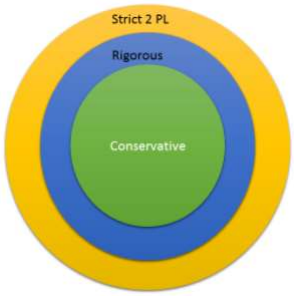
3. Conservative Two-Phase Locking Protocol

- Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.

40

Example 1: >>>

T1
Lock-s(A)
Read(A)
Lock-x(B)
Read(B)
Unlock(A)
Write(B)
Unlock(B)



Now we will see what is the above schedule following the properties discussed above

2 PL: There is growing and shrinking phase.

Strict 2 PL: There is Lock-x(B) and it is unlocked before commit so no strict 2 PL.

Rigorous: If it is not strict 2 PL then it can't be Rigorous.

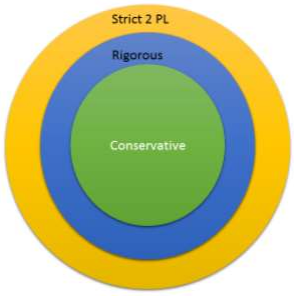
Conservative: If it is not strict 2 PL then it can't be conservative.

41

Example 2:

>>>

T1
Lock-s(A)
Read(A)
Lock-x(B)
Unlock(A)
Read(B)
Write(B)
commit
Unlock(B)



2 PL: There is growing and shrinking phase so it is 2 PL.

Strict 2 PL: Exclusive locks are unlocked after commit. So yes it is.

Rigorous: We have taken Lock-s(A) and we have unlocked it before commit. So no rigorous.

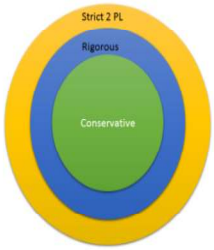
Conservative: We have not taken all the locks at first then start the transaction so no conservative.

42

Example 3:

>>>

T1
Lock-s(A)
Read(A)
Lock-x(B)
Read(B)
Write(B)
commit
Unlock(B)
Unlock(A)



2 PL: There is growing and shrinking phase so it is 2 PL.

Strict 2 PL: Exclusive locks are unlocked after commit. So yes it is.

Rigorous: We have unlocked all the locks after commit so it is rigorous.

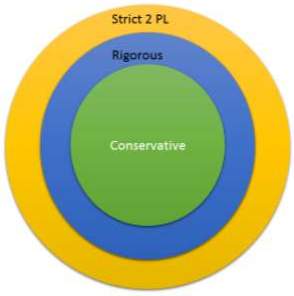
Conservative: We have not taken all the locks at first then start the transaction so no conservative.

43

Example 4:

>>>

T1
Lock-s(A)
Lock-x(B)
Read(B)
Write(B)
Read(A)
commit
Unlock(A)
Unlock(B)



2 PL: There is growing and shrinking phase so it is 2 PL.

Strict 2 PL: Exclusive locks are unlocked after commit. So yes it is.

Rigorous: We have unlocked all the locks after commit so it is rigorous.

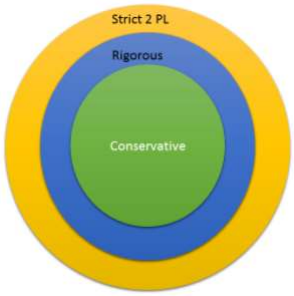
Conservative: We have taken all the locks at first then start the transaction so yes it is conservative.

44

Example 5:

>>>>

T1
Lock-s(A)
Read(A)
Unlock(A)
Lock-x(B)
Read(B)
Write(B)
Unlock(B)
Unlock(A)
Commit



2 PL: There is no growing and shrinking phase so it is not 2 PL.

Strict 2 PL: Because it is not 2 P L so not either of it.

Rigorous: Because it is not 2 P L so not either of it.

Conservative: Because it is not 2 P L so not either of it.

45

What is deadlock?

- A deadlock is a condition that occurs when two or more different database tasks are waiting for each other and none of the task is willing to give up the resources that other task needs.
- It is an unwanted situation that may result when two or more transactions are each waiting for locks held by the other to be released.
- In deadlock situation, no task ever gets finished and is in waiting state forever.
- Deadlocks are not good for the system.

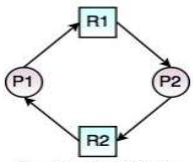


Fig. Deadlock State

DeadLock Problem Example

T1	T2
Lock-X(A)	
	Lock-x(B)
Lock-x(B)	
	Lock-x(A)

So T1 is waiting for B and T2 is waiting for A and both A and B are held by T1 and T2 respectively.

46

Time Stamp Protocol

It says a very basic rule that if a transaction Tj that enters after Ti then TimeStamp(Ti) < TimeStamp(Tj) which means that producing schedule must be equivalent to a serial schedule Ti->Tj.

In Time Stamp Protocol ensures that any conflicting read and write operations are executed in time stamp order if not such an operation is rejected and transaction will be rolled back.

The rolled back transaction will be restarted with a new Time Stamp.

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

Here you could see that conflict is occurring between T2->T1 and it is given that Time Stamp (T1) < Time Stamp (T2) which means it the generated conflict must be resolved in T1->T2. But which is not possible so we rollback transaction T1.

47

Thomos Write Rule

We allow write-write conflict by ignoring.

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

Note The conflict occurred says T1->T2 and it is given that Time Stamp (T2) < Time Stamp (T1) which means it the conflict can't be resolved but

Thomos write rule says that we can ignore the write done by T1 as it has been overwritten by T2 later.

48

Example 1

T1	T2
R(B)	
	R(B)
	B=B-50
	W(B)
R(A)	
	R(A)
Display(A+B)	A=A+50
	W(A)
	Display(A+B)

The above two arrows are showing conflict of type Read-Write.

Time Stamp (T1) < Time Stamp (T2), this is already given and conflicts are saying T1 -> T2. So we are good.

Example 2

T1	T2	T3	T4
W(X)			
	W(X)		
		W(X)	
	R(X)		
			W(X)

It is given that Time Stamp (T2) < Time Stamp (T3) so according to Time Stamp T2 -> T3 should be there.

But the conflict says it should be T3 -> T2

And is also not allowed in Thomos write Rule as well.

49

Advantages

1. Serializability
2. Ensures freedom from dead lock

Disadvantage

Starvation may occur due to continuously getting aborted and restarting the transaction.

50