5. Transaction Management

Page No.:
Date: / /

Transaction concept

i) Transaction is a set of logically related operations
ii) A transaction is a unit of program execution that accesses & possibly updated various data items
iii) A transaction is delimited by statements of the form begin transaction & end transaction, the transaction consists of all operations executed between begin transaction & end transaction
iv) E.g transaction to transfer $50 from account A to account B

read (A)
A : A - 50
write (A)
read (B)
B = B + 50
write (B)

ACID properties in DBMS

A = Atomicity → The entire transaction takes place at once or doesn't happen at all.
C = consistency → The database must be consistent before and after the transaction.
I = Isolation → Multiple Transactions occur independently without interference.
D = Durability → The changes of successfully transaction occur even if the system failure occurs.

① Atomicity

i) Either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e transactions do not occur partially. Each

transaction is considered as one unit & either runs to completion or is not executed at all. It involves following two operations.

① Abort :- If transaction aborts, changes made to db are not visible.

② commit :- If transaction commits changes made are visible. Atomicity is also known as the "All or nothing rule."

e.g →

- consider two transaction T_1 & T_2, Transfer of ₹00 from account X to account Y.

T_1	T_2
Before	
X : 500	Y : 200
Read (x)	Read (Y)
X : X - 100	Y = Y + 100
write (x)	write (Y)
X : 400	Y = 300

- If the transaction fails after completion of T_1 but before completion of T_2 then amount has been deducted in entirety in order to ensure correctness of DB state.

2) Consistency

i) This means that integrity constraints must be maintained so that DB is consistent before & after the transaction.

ii) Referring to the above example, the total amount before & after the transaction must be maintained.

Total before T occurs = 500 + 200 = 700

Total after T occurs = 400 + 300 = 700

Therefore database is consistent. Inconsistency occurs in case T_1 completes but T_2 fails As a result T is incomplete

Page No.:
Date: 1 /

3) Isolation

i) This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of DB state.

ii) Transaction occur independently without interference

iii) changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that trans. is written to memory or has been committed.

iv) This property ensures that execution of trans. concurrently will result in a state i.e equivalent to state achieved these were serially in some order, Let X = 500, Y = 500

e.g → T_1 & T_2

T_1	T_2
Read(X)	Read(X)
X := X * 100	Read(Y)
write(X)	Z := X + Y
Read(Y)	write(Z)
Y = Y - 50	
write	

- suppose T_1 has been executed till Read(Y) & then T_2 start, as a result, interleaving of operations takes place due to which T_2 reads correct value of X but incorrect value of Y & sum computed by

$T_2 = X + Y = 50,000 + 500 = 50,500$

is thus not consistent with the sum at end of trans.

$T_1 = (X + Y) = 50,000 + 500 = 50,450$

so result is inconsistency due to loss of 50 units.

hence, transaction must take

4) Durability

i) This property ensures that once the transaction has completed execution, the updates & modifications to the DB are stored in & written to disk & they persist even if a system failure occures

ii) These updates now become permanent & are stored in non-volatile memory.

So the ACID properties in totality provide a mechanism to ensure correctness & consistency of DB in a way such that each trans. is a group of operations that acts a single unit, produces consistent results, acts in DB isolation from other operations & updates that it makes are durably stored.

Transaction state

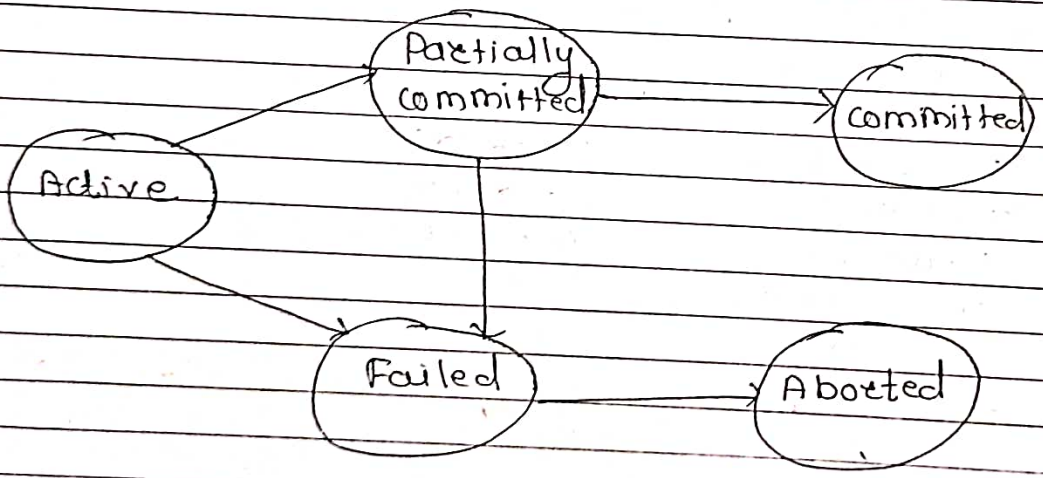


fig → state diagram of a transaction

1) Active,

i) It is a initial state

ii) The transaction stays in this state while it is executing.

iii) If transaction is in execution then it is said to be in active state.

iv) It doesn't matter which step is in execution until

D D M M Y Y Y Y

unless the transaction is executing, it remains in active state.

Failed Partially committed

i) In above fig. transaction goes into partially committed state from the active state when there are read & write operations present in transaction

ii) A transaction contains no. of read & write operⁿ once the whole trans. is successfully executed, the transaction goes into partially committed state where we have all the read & write operations performed on the main memory instead of the actual DB.

iii) This state is used because transaction can fail during execution so if we are making changes in the actual DB instead of local memory, DB may be left in an inconsistent state in case of any failure.

iv) This state helps us to rollback the changes made to the DB in case of failure during execution.

3) Committed

i) If transaction completes the execution successfully then all the changes made in the local memory during partially committed state are permanently stored in DB

ii) In above dia. transaction goes from partially committed state when everything is successful.

4) Aborted state

i) If transaction fails during execution then the transaction goes into a failed state, the changes made into the local memory are rolled back to the previous consistent state & transaction goes into aborted state from the failure state.

Implementation of atomicity & Durability

i) The recovery management component of DB system implements the support of atomicity & durability

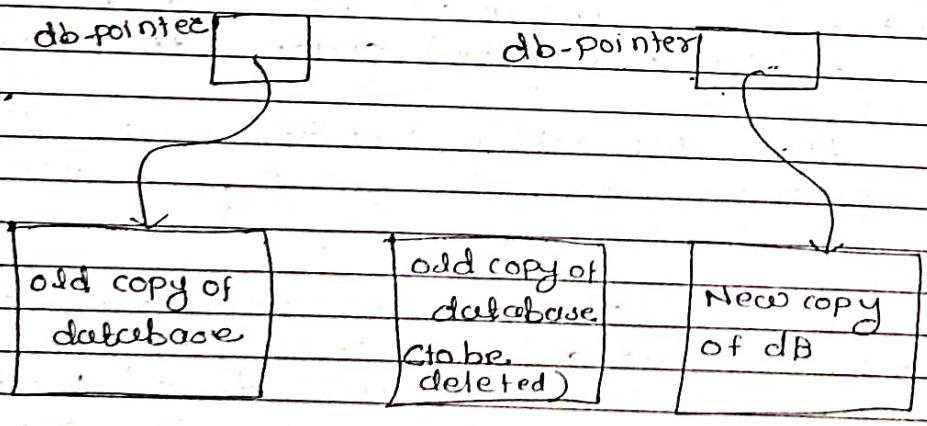
Shadow Database scheme

i) Assume that only one transaction is active at a time.

ii) A pointer called db-pointer always points to the current consistent copy of the db.

iii) All updates are made on a shadow copy of the DB & db-pointer is made to point to the updated shadow copy only after the transaction reaches partial commit & all updated pages have been flushed to disk

iv) In case transaction fails, old consistent copy pointed to by db-pointer can be used, & the shadow copy can be deleted



a) Before update

b) After update

D D M M Y Y Y Y

Transaction Isolation.

i) The transaction isolation is primarily used as means to provide accurate & reliable access to data within concurrent transactions.

ii) e.g → Two different transactions might be accessing the same data simultaneously. Therefore, if the change made on the data by one transaction is not passed to other transaction, it can affect DB operations.

There are 4 different types of transaction isolation level.

1) Serializable → Implements read & writes locks until the transaction is finished also implements range locks.

2) Repeatable reads
Implement read & write locks until the transaction is completed, doesn't manage range lock.

3) Read committed
Implements write lock until the transaction completed but releases read locks when SELECT operation is performed.

4) Read uncommitted
one transaction can see the uncommitted changes made by other transaction.

D D M M Y Y Y Y

Serializability

Types of schedules in DBMS

i) Schedule is a process of lining the transaction & executing them one by one.

ii) Schedule is a sequences of function instructions that specify the chronological order in which instructions of concurrent transactions are executed.

iii) A schedule for a set of transaction must consist of all instructions of those transactions.

iv) It must preserve the order in which the instructions appear in each individual transaction.

v) A transaction that successfully completes it's execution will have a commit instructions as the last statement

Schedule (1)

- Let T_1 transfer $50 from A to B & T_2 transfer 10% of the balance from A to B
- A serial schedule in which T_1 is followed by T_2 :

T_1	T_2
read(A)	
A := A - 50	
write(A)	
read(B)	
B := B + 50	
write(B)	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
	read(B)
	B := B + temp
	write(B)

Page No.:
Date: / /

Schedule ②

- A serial schedule where T_2 is followed by T_1

T_1	T_2
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
	read(B)
	B := B + temp
	write(B)
read(A)	
A := A - 50	
write(A)	
read(B)	
B := B + 50	
write(B)	

Schedule ③ (Non Serial)

Let T_1 & T_2 be the transactions defined previously. The following schedule is not a serial schedule but it is equivalent to schedule 1.

T_1	T_2
read(A)	
A := A - 50	
write(A)	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
read(B)	
B := B + 50	
write(B)	
	read(B)
	B := B + temp
	write(B)

In schedule 1, 2, & 3 the A+B preserved

Schedule (4)

The following concurrent schedule does not preserve the value of (A+B)

T_1	T_2
read(A)	
A = A - 50	
	read (A)
	temp = A * 0.1
	A = A - temp
	write (A)
	read (B)
write (A)	
read (B)	
B = B + 50	
write (B)	
	B = B + temp
	write (B)

Serial schedule:

i) schedules in which the transactions are executed i.e a serial schedule is one in which no transaction starts until a running transactions has ended are called serial schedules. i.e In serial schedule a transaction is executed completely before starting the execution of another transaction.

ii) A transaction does not start execution until the currently running transaction finished execution.

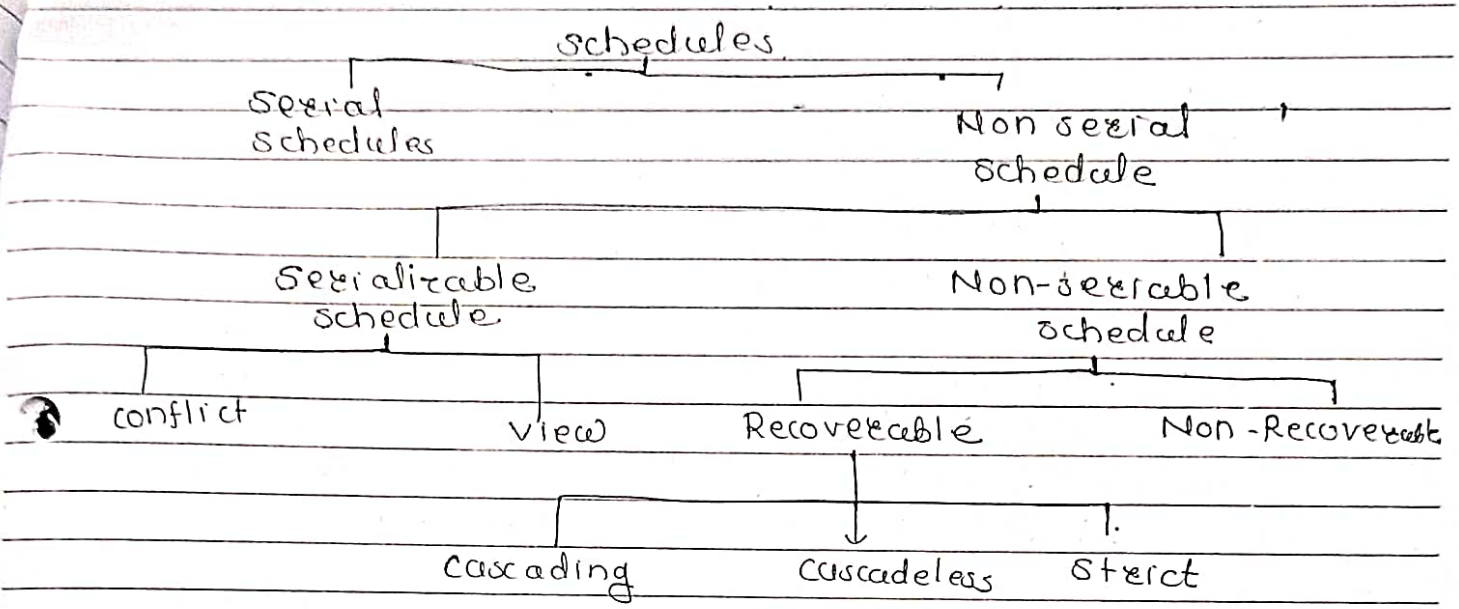
Non serial schedule

i) This is a type of scheduling where the operations of multiple transactions are interleaved.

ii) The transactions are executed in a non serial manner, keeping the end result is correct & same as serial schedule.

i) In serial schedule where one trans. proceed without waiting for previous the non serial

ii) In non serial schedule where one transaction proceeds without waiting for the previous trans. to complete.



Serializability

i) serial schedule does not need the serializability because it follows transaction only when the previous transaction is complete.

ii) The non serial schedule is said to be in serializable schedule only when it is equivalent to the serial schedules, for an no. of transactions.

There are two types
1) conflict
2) view

1) conflict serializability

i) If a schedule s can be transformed into a schedule s' by a series of swaps of non-conflicting instructions, we say that s & s' are conflict equivalent.

ii) we say that schedule s is conflict serializable if it is conflict equivalent to a serial schedule

iii) schedule ① can be transformed into schedule ② a serial schedule where T2 follows T1 by series of swaps of non conflicting instructions. therefore schedule ① is conflict serializable

e.g →

T1	T2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

schedule 1

T1
read(A)
write(A)
Read(B)
write(B)
read(A)
write(A)
read(B)
write(B)

schedule 2

iv) E.g of a schedule that is not conflict serializable

T3	T4
read(Q)	
	write(Q)
write(Q)	

v) we are enable to swap instruction in above schedule to obtain either the serial schedule $\langle T_3, T_4 \rangle$ or serial schedule $\langle T_4, T_3 \rangle$

Page No.:
Date: / /

view serializability

i) Let S & S' be two schedules with the same set of transactions.

ii) S & S' are view equivalent if the following 3 conditions are met.

a) For each data item Q if transaction T_1 reads the initial value of Q in schedule S, then transaction T_1 must in schedule S' also read the initial of value of Q.

b) For each data item Q if trans. T_1 executes read(Q) in schedule S, & that value was produced by trans. T_2, then tran. T_1 must in schedule S' also read the value of Q that was produced by transaction T_2.

c) For each data item Q, the trans. that performs the final write(Q) operation in schedule S must perform the final write(Q) oper in schedule S'

As can be seen, view equivalence is also based purely on reads & writes alone.

iii) A schedule S is view serializable it is view equivalent to a serial schedule.

iv) Every conflict serializable schedule is also view serializable.

v) Below is a schedule which is view serializable. but not conflict serializable.

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

vi) Every view serializable schedule i.e not conflict serializable has blind writes

vii) The schedule below produces same outcome as th serial schedule $\langle T_1, T_5 \rangle$ yet is not conflict equivalent or view.

T_1	T_5
read(A)	
A = A-50	
write(A)	
	read(B)
	B = B-10
	write(B)
read(B)	
B = B+50	
write(B)	
	read(A)
	A = A+10
	write(A)

Determining such equivalence requires analysis

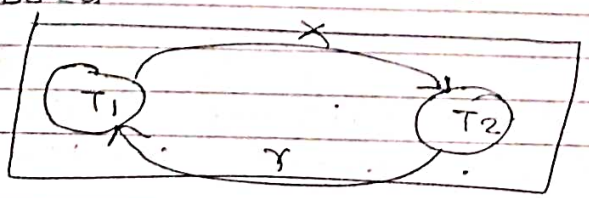
Testing for serializability. Precedence graph

i) consider some schedule of a set of trans. $T_1, T_2 \cdots T_n$

Precedence graph

i) Precedence graph is a direct graph where the vertices are transaction (names)
ii) we draw an arc from T_i to T_j if two transaction conflict & T_i accessed the data item on which the conflict arise earlier
iii) we may label the arc by the item that was accessed

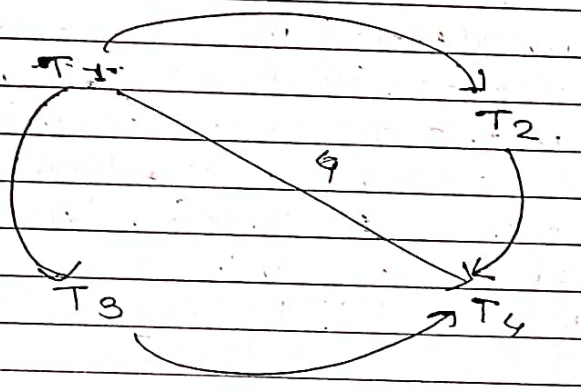
e.g



Page No.:
Date: / /

Example :-

T_1	T_2	T_3	T_4	T_5
	read(A)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Test for conflict serializability

i) A schedule is conflict serializable if & only if it's precedence graph is a acyclic

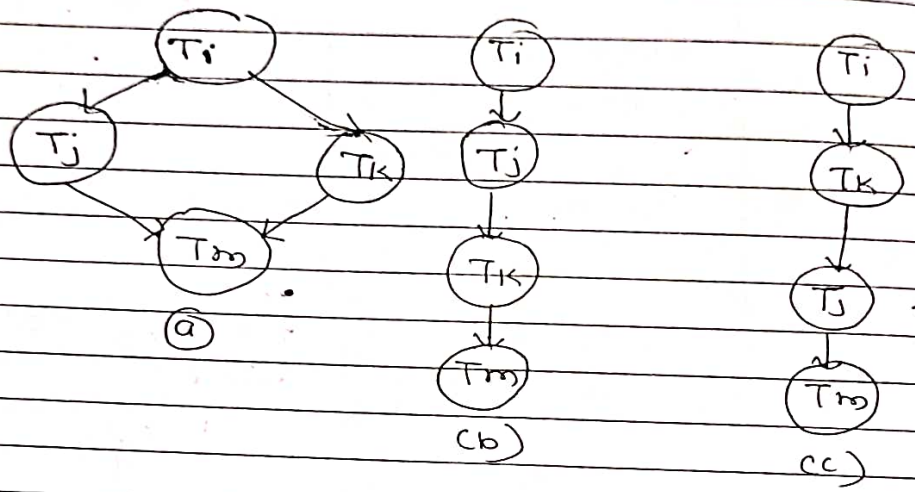
ii) cycle detection algo. exist which take order n^2 time, where n is the no. of vertices in the graph

iii) If precedence graph is acyclic, the serializability order can be obtained by topological sorting of graph

iv) This is a linear order consistent with the partial order of the graph.

e.g → serializability order for schedule would be

$T_5 \rightarrow T_1 \rightarrow T_8 \rightarrow T_2 \rightarrow T_4$



- A schedule is conflict serializable if & only if its precedence graph is acyclic.
- The precedence graph test for conflict serializability cannot be used directly to test for view —11— .
- Extension to test for view serializability has cost exponential in the size of precedence graph.
- The problem of checking if schedule is view serializable falls in the class of NP complete problems.
- So it check some sufficient conditions for view serializability can still be used.

Recoverable schedules

i) If a transaction T_i reads a data item previously written by trans. T_i, then the commit operation of T_i appears before the commit operation of T_j

ii) The following schedule is not recoverable if T_9 commits immediatly after the read.

T8	Tg
read(A)	
write(A)	
	read(A)
read(B)	

iii) If T8 should abort, Tg would have read an inconsistent DB state, hence db must ensure that schedules are recoverable.

1) Cascading Rollbacks

i) A single transaction failure leads to a series of transaction rollbacks.

ii) Consider the following schedule where none of the transactions has yet committed

T_1	T_2	T_3
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

iii) If T_1 fails, T_2, T_3 must also rolled back. It can lead to the undoing of a significant amount of work.

2) cascadeless schedules

i) In cascadeless schedules, cascading rollbacks cannot occur, for each pair of transactions T_i & T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j

ii) Every cacadeless schedule is also recoverable.

iii) It is desirable to restrict the schedules to those that are cascadeless

Page No.:
Date: / /

3) Strict schedule

i) If schedule contains no read or write before commit then it is known as strict schedule.

ii) In this schedule no read write or write, write conflict arises before commit hence its strict schedule.

T1	T2	T3
Read(x)		
	Read(x)	
Read(z)		
		Read(x)
		Read(y)
write(x)		
commit		
		write(y)
		commit
	Read(y)	
	write(z)	
	write(y)	
	commit	

Page No.:
Date: / /

Lock based protocols

i) Lock is a mechanism to control concurrent access to a data item.

ii) Data items can be locked in two modes
   a) Exclusive
   b) Shared.

a) Exclusive (X)
- If transaction Ti obtained an exclusive-mode lock on item Q, then Ti can both read & write Q
- It is denoted by X

b) shared
- If transaction Ti is obtained a shared mode lock on item Q then Ti can read but cannot write Q
- It is denoted by S.

- Lock requests are made to the concurrency control manager by the programmer. Transaction can proceed only after request is granted.

* Lock compatibility matrix

	S	X
S	True	false
X	false	false

i) A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions.
ii) Any no. of transactions can hold shared locks on an item.
iii) But if any transaction holds an exclusive on the item not other transaction may hold any lock on the item.
iv) If a lock cannot be granted, the requesting transaction is made to wait till all incompatible

locks held by other transactions have been released. The lock is then locked granted.

v) example :

```
T2 : lock S(A);
     read(A);
     unlock(A);
     lock S(B);
     read(B);
     unlock(B);
     display(A+B)
```

vi) locking as above is not sufficient to guarantee serializability, if A & B get updated in between the read of A & B, the displayed sum would be wrong.

vii) A locking protocol is set of rules followed by all transactions while requesting & releasing locks. Locking protocols restrict the set of possible schedules.

Pitfalls of lock based protocols:

i) consider the partial schedule.

T_3	T_4
lock X(B)	
read(B)	
B := B - 50	
write(B)	
	lock S(A)
	read(A)
	lock S(B)
lock X(A)	

ii) Neither T_3 nor T_4 can make progress executing lock S(B) causes T_4 to wait for T_3 to release it's lock on B while executing lock X(A) causes T_3 to wait for T_4 to release it's lock on A

such situation is called as deadlock.

iii) To handle a deadlock one of T_3 or T_4 must be rolled back & it's locked released.

iv) The potential for deadlock exists in most locking protocols, deadlocks are necessary evil.

v) Starvation is also possible if concurrency control manager is badly designed.

e.g → A trans. may be waiting for an X-lock on an item, while a sequence of other trans. request & are granted an s lock on same item
- The same trans. is repeatedly rolled back due to deadlocks.

vi) concurrency control manager can be designed to prevent starvation.

Two Phase locking

i) This is a protocol which ensures conflict serializable schedules.

ii) It is a concurrency control method which divides the execution phase of trans. into 2 parts.

iii) If read & write operations introduce the 1st unlock operation in the transaction then it is said to be two-phase locking protocol.

It can be divided into two phases
1) Growing phase
2) Shrinking phase

Growing phase
- A transaction obtains locks but may not release any lock.

Shrinking phase
- Transaction may release locks, but may not obtain any lock.

- Two phase locking does not ensure freedom from deadlocks.
- cascading roll back is possible under two phase locking to avoid this, follow modified protocol called strict two 2PC

Types of 2 phase locking protocol

i) strick two phase locking protocol.
ii) Rigorous two —"—
iii) conservative —"—

i) Strict two phase locking protocol

i) It avoids cascade rollback.
ii) This protocol not only requires two phase locking but also all exclusive locks should be held until transaction commits or abort.
iii) It is not deadlock free.
iv) It ensures that if data is being modified by one transaction, then other transaction cannot read it until 1st transaction commits.
v) most DB system implement rigorous two phase locking protocol.

2) Rigorous [illegible] locking

i) It avoids cascading rollbacks.
ii) It requires that all the share & exclusive locks to be held until the transaction commits

DATE / /

Implementation of locking

i) A lock manager can be implemented as a separate process to which transactions send lock & unlock requests.

ii) The lock manager replies to a lock request by sending a lock grant msg.

iii) The requesting transaction waits until it's request is answered.

iv) It maintains data structure called a lock table to record granted locks & pending requests.

v) The locks table is usually implemented as an in memory hash table indexed on the name of data item being locked.

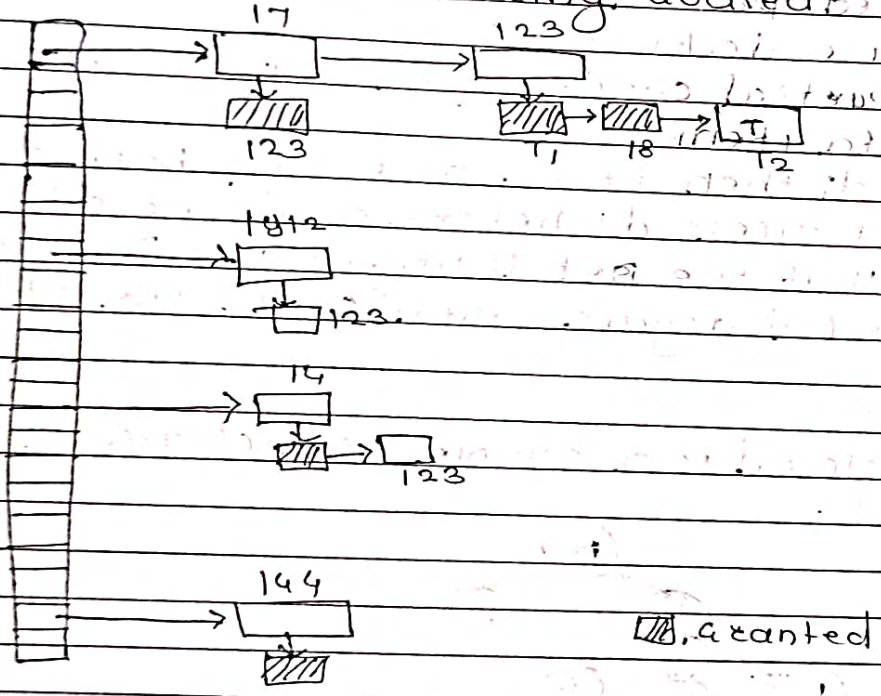


fig. lock table.

- Dashed rectangles indicates granted locks white one indicates waiting requests.
- Lock table also records the type of lock granted or requested.

- New requests result in the request being deleted & later request both are checked to see if they can now be granted.
- If the Unlock requests result in the request being deleted & later response requests are checked to see if they can now be granted.
- If transactions aborts all waiting or granted requests of the transaction are deleted.

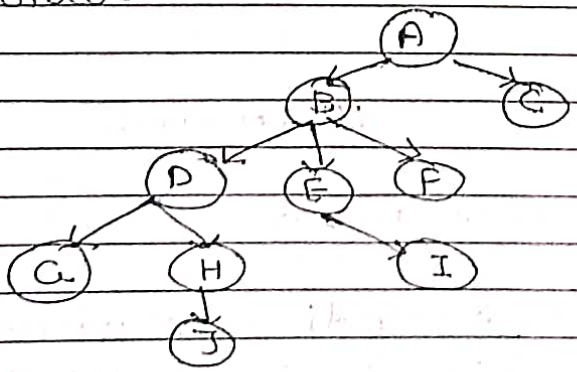
vi) lock manager keep a list of locks held by each transaction to implement this efficiently

* Graph Based Protocol

i) Graph based protocols are an alternative to two phase locking.

ii) Impose partial ordering on set $D = (d_1, d_2 \dots d_n)$ of all data items.
- if $d_i \rightarrow d_j$ then trans. accessing both d_i & d_j must access d_i before accessing d_j.
- Implies that the set D may now be viewed as a directed acyclic graph called the DB graph.

iii) The protocol is a simple kind of graph protocol.



a) only exclusive locks are allowed.
b) The first lock by T_i may be on any data item. subsequently a data Q can be locked by T_i only if the parent of Q is currently locked by T_i
c) Data items may be unlocked at any time
d) A data item that has been locked & unlocked by T_i cannot subsequently be relocked by T_i

iv) The tree protocol ensures conflict serializability as well as freedom from deadlock.
v) Unlocking may occur earlier in the tree locking protocol than in two phase locking protocol.
- waiting time is short & concurrency increased
- protocol is deadlock free, no rollbacks required

vi) Drawbacks

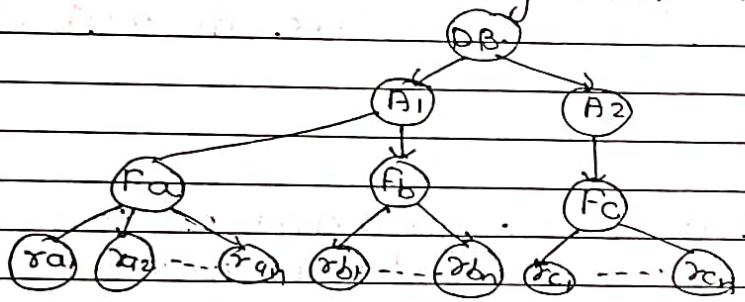
i) Protocol does not guarantee recoverability or cascade freedom. Need to introduce commit dependencies to ensure recoverability
ii) Trans. may have to lock data items that they don't access.

DATE / /

Multiple Granularity

- i) Allow data items to be of various sizes & define a hierarchy of data granularities, where the small granularities are nested within large ones
- ii) It can be represented graphically as tree:
- iii) when transaction locks a node in the tree explicitly it implicitly locks all the nodes descendents in the same mode.
- iv) Granularity of locking
  - a) fine granularity → lower in tree, high concurrency high locking overhead:
  - b) coarse granularity → Higher in tree, low locking overhead, low concurrency.

Example : Granularity Hierarchy



Deadlock Handling

- Consider the following two transactions

T1 : write (X)
write (Y)

T2 : write (Y)
write (X)

- schedule with deadlocks

T1	T2
lock X on X	
write(X)	
	lock X on Y
	write(X)
	wait for lock X on X
wait for lock X on Y	

Page No.:
Date : /

i) system is deadlocked if there is set of transaction in the set is waiting for another transaction in the set

ii) Deadlock prevention protocols ensure that the system will never enter into deadlock state

some prevention strategies:

a) - Require that each transaction locks all it's data items before it begins execution

b)