Concurrency Control :

- procedure required for controlling concurrent execution of the operations that take place on database

Concurrent execution:

- multiple user can access and use the same database at one time, which is known as concurrent execution.

→ Goal is to develop concurrency control protocol to ensure serialibity.
→ To develop a schedule which is serializable.

Protocols

① Lock-Based protocol :

- Lock mechanism is used to control concurrent access of data item.

Two modes in which data items can be locked.

Exclusive (X) mode

→ Transaction can perform read and write operations.
→ lock-x(Q)
→ Any other transaction can't obtained either exclusive/shared lock.

shared (S) mode

→ can perform read operation
→ lock-s(Q)
↳ transaction.
→ we can apply same lock on same data item at same time in any other transaction

* Compatibility

	S	X
S	T	F
X	F	F

T1 :
```
lock-S(A)
read(A)
unlock(A)
lock-S(B)
read(B)
unlock(B)
display(A+B)
```
} Not sufficient to gurantee Serializability

⇓

T1
```
lock-S(A)
lock-S(B)
Read(A)
read(B)
display(A+B)
unlock(A)
unlock(B)
```
} Serializability achieved

2) Two-phase locking (2PL)

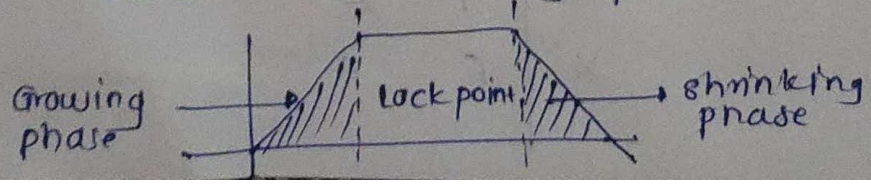
- Protocol that ensures conflict serializable schedules.
- Requires both lock and unlock being done in two phases.

Phases

① Growing phase
- New lock on items can be acquired
- But no lock can be released.

② Shrinking phase
- Release existing locks
- But no new lock can be acquire.

③ Lock point :- point at which growing phase acquires its final lock and then shrinking phase can be started



Variations of 2PL locking protocol :

① Conservative (static) 2PL :
- Acquire all locks before it starts
- Release all locks after commit
- It is deadlock free
- Avoids cascading rollback.

② strict 2PL :
- Transaction holds all exclusive locks till commit / abort
- avoids cascading rollback.
- deadlock may occur.

③ Rigorous 2PL :
- In regrous 2PL both shared & exclusive locks till commit
- avoids cascading rollback
- deadlock may occur.

Implementation of Locking :
- Separate process is runned by lock manager
- send Lock / Unlock request
- Lock manager will Grant / reject request (Transaction will wait for answer)
- Lock table will be maintained by Lock manager.

3) Graph Based protocol :
- Alternative for 2phase locking protocol.
- Additional info about transaction i.e "how each transaction will access data item" will be required.
- There are various method to get that additional information.

- one method can be having prior knowledge about the order in which database items will be accessed.

- i.e partial ordering
  if data items $D = \{d_1, d_2 \dots d_n\}$
  and if $d_i \rightarrow d_j$
  then d_i must access first then access d_j

- Set $D \Rightarrow$ directed acyclic graph called as database graph

- It is free from deadlock
- No roll back required
- ensures conflict serializability

4) Time-stamp based protocol:

- It is tag that can be attach to any data item or transaction, which denotes specific time when transaction or data itemes are activated
  - unique timestamp
  - Newer transactions timestamp strictly < older —11—
  - Timestamp order = execution order.

methods → system clock
→ logical counter

→ W-TS(Q) = write timestamp of transaction
→ R-TS(Q) = Read —11—

Time-stamp ordering protocol:

- Ensures that conflicting Read & write operations are executed in time-stamp order.
- Free from deadlock
i) Ti issues Read(Q):

Cases :

(i) If $TS(T_i) < W(Q) - TS$, T_i is an older transaction than last transaction that wrote the value of Q.

→ Request failed.

(ii) If $TS(T_i) \geq W(Q) - TS$

→ Here Request granted
→ T_i is allowed to read updated value of Q.

ii) T_i issues write(Q) :

Cases

[i] If $TS(T_i) \geq W(Q) - TS$ and $TS(T_i) \geq R(Q) - TS$

→ then T_i is allowed to modify/write value of Q.
→ and $TS(T_i)$ will become current value of W(Q)

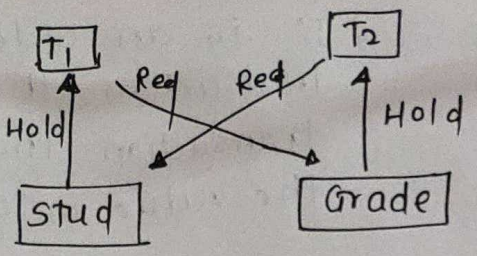
[ii] If $TS(T_i) < R(Q) - TS$, Younger transaction is already using value of Q

→ updation not allowed

[iii] if $R(Q) - TS \leq TS(T_i) \leq W(Q) - TS$, Younger transaction has updated Q.

Deadlock in transaction :

- conditions where two or more transactions are waiting indefinitely for one another to give up the locks
- None task ever gets finished and is in waiting state forever



deadlock

- T_1 is waiting for T_2 to release lock & vice versa for T_2
- All activity comes to halt until deadlock detected & resolved.

Deadlock Avoidance :

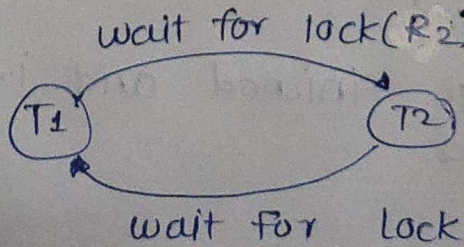
- when DB stuck in deadlock state, then it is better to avoid the database rather than aborting / Restarting
- for smaller databases = wait for graph method is used

  larger databases = deadlock prevention method is suitable.

Deadlock Detection :

- If transaction waits indefinitely to obtain a lock, then DB should check transaction is in deadlock state or not?

- **wait for graph method :**
  - Graph is created, based on transaction & their lock.
  - If resultant graph has cycle then deadlock is present.
  - It is maintained by system for transactions waiting for data.
  - Regularly check for cycles.



Deadlock prevention :

- used for large databases.
- DBMS analyses transactions and never executes those transactions that are creating deadlocks.

Two methods

(1) Wait - Die scheme

- If transaction request for resource which is already held with a conflicting lock by another transaction then DBMS simply check for timestamps of both transactions

i) if $TS(T_i) < TS(T_j)$

then Older transaction waits until resource becomes available

ii) if $TS(T_i) \geq TS(T_j)$

T_i aborted and rolled back

(2) Wound - wait scheme

- If older one request resources held by younger one, the older forces younger to abort.
- If older one holds resources requested by younger one, then younger one must wait until it is released.