DEADLOCK

In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example :

Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

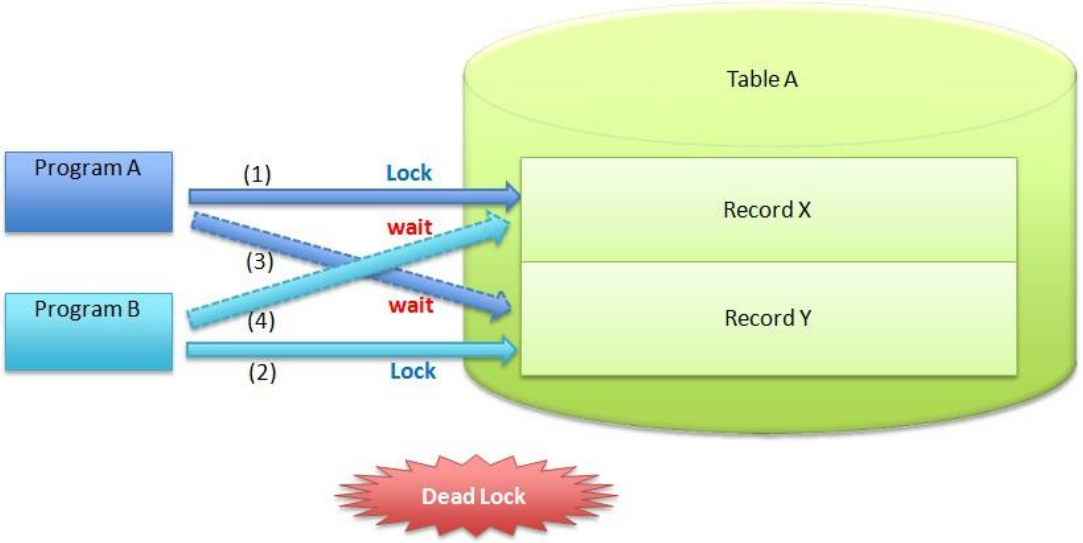
Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up lock, and similarly, transaction T2 will wait for transaction T1 to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set. Consider the following two transactions:

T1:	T2
write (A)	write(A)
write (B)	write(B)

Schedule with deadlock

T_1	T_2
lock-X on A write (A)	
	lock-X on B write (B) wait for **lock-X** on A
wait for **lock-X** on B	



Deadlock Handling

Deadlock prevention protocol Ensure that the system will never enter into a deadlock state.

Some prevention strategies :

Approach1

– Require that each transaction locks all its data items before it begins execution either all are locked in one step or none are locked.

Disadvantages

- Hard to predict, before transaction begins, what data item need to be locked.
- Data item utilization may be very low.

Approach 2 – Assign a unique timestamp to each transaction. – These timestamps only to decide whether a transaction should wait or rollback.

Deadlock prevention Schemes:

- wait-die scheme
- wound-wait scheme

(i) wait-die scheme

- Non preemptive technique
- When transaction Ti request a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp smaller than that of Tj. otherwise ,Ti rolled back(dies)
- Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead.
- A transaction may die several times before acquiring needed data item

Example.

Transaction T1,T2,T3 have time stamps 5,10,15,respectively.

- if T 1 requests a data item held by T2,then T1 will wait.
- If T3 request a data item held by T2,then T3 will be rolled back.

(ii) Wound-wait scheme

- Preemptive technique
- When transaction Ti requests a data item currently held by Tj,Ti is allowed to wait only if it has a timestamp larger than that of Tj. Otherwise Tj is rolled back
- Older transaction wounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones.

Example

Transaction T1,T2,T3 have time stamps 5,10,15,respectively.

- if T1 requests a data item held by T2,then the data item will be preempted from T2,and T2 will be rolled back.
- If T3 requests a data item held by T2,then T3 will wait.

DeadLock Detection

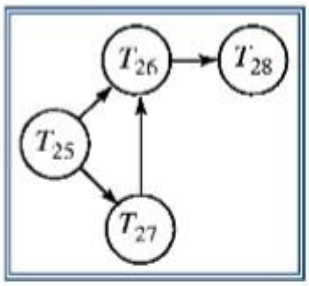
Deadlocks can be described as a wait-for graph, which consists of a pair G = (V,E)

– V is a set of vertices

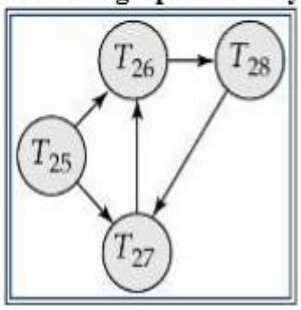
– E is a set of edges

- If Ti ->Tj is in E, then there is a directed edge from Ti to Tj, implying that Ti is waiting for Tj to release a data item.
- The system is in a deadlock state if and only if the wait-for graph has a cycle. Must invoke a deadlock-detection algorithm periodically to look for cycles.

Wait-for graph without a cycle



Wait-for graph with a cycle



Recovery from deadlock

• The common solution is to roll back one or more transactions to break the deadlock.

• Two action need to be taken

– Selection of victim

– Rollback

Selection of victim

• Set of deadlocked transactions, must determine which transaction to roll back to break the deadlock.

• Consider the factor minimum cost

Rollback

• once we decided that a particular transaction must be rolled back, must determine how far this transaction should be rolled back

• Total rollback

• Partial rollback

Starvation :

Starvation or Livelock is the situation when a transaction has to wait for a indefinite period of time to acquire a lock.

Reasons of Starvation –

- If waiting scheme for locked items is unfair. (priority queue)
- Victim selection. (same transaction is selected as a victim repeatedly)
- Resource leak.
- Via denial-of-service attack.

Starvation can be best explained with the help of an example

- Suppose there are 3 transactions namely T1, T2, and T3 in a database that are trying to acquire a lock on data item ' I '.
- Now, suppose the scheduler grants the lock to T1(maybe due to some priority), and the other two transactions are waiting for the lock.
- As soon as the execution of T1 is over, another transaction T4 also come over and request unlock on data item I. Now, this time the scheduler grants lock to T4, and T2, T3 has to wait again.
- In this way if new transactions keep on requesting the lock, T2 and T3 may have to wait for an indefinite period of time, which leads to **Starvation**.

What are the solutions to starvation –

(i) Increasing Priority –

Starvation occurs when a transaction has to wait for an indefinite time, In this situation, we can increase the priority of that particular transaction/s. But the drawback with this solution is that it may happen that the other transaction may have to wait longer until the highest priority transaction comes and proceeds.

(ii) Modification in Victim Selection algorithm –

If a transaction has been a victim of repeated selections, then the algorithm can be modified by lowering its priority over other transactions.

(iii) First Come First Serve approach –

A fair scheduling approach i.e FCFS can be adopted, In which the transaction can acquire a lock on an item in the order, in which the requested the lock.

(iv) Wait die and wound wait scheme –

These are the schemes that use the timestamp ordering mechanism of transaction.

Intent locking

- Intent locks are put on all the ancestors of a node before that node is locked explicitly.
- If a node is locked in an intention mode, explicit locking is being done at a lower level of the tree.

Types of Intent Locking

- Intent shared lock(IS)
- Intent exclusive lock(IX)
- Shared lock (S)
- Shared Intent exclusive lock (SIX)
- Exclusive lock (X)

Intent shared lock(IS)

- If a node is locked in indent shared mode, explicit locking is being done at a lower level of the tree, but with only shared-mode lock
- Suppose the transaction T1 reads record ra2 in file Fa. Then,T1 needs to lock the database, area A1,and Fa in IS mode, and finally lock ra2 in S mode.

Intent exclusive lock(IX)

- If a node is locked in intent locking is being done at a lower level of the tree, but with exclusive mode or shared-mode locks.
- Suppose the transaction T2 modifies record ra9 in file Fa. Then,T2 needs to lock the database, area A1,and Fa in IX mode, and finally to lock ra9 in X mode.

Shared Intent exclusive lock (SIX)

If the node is locked in Shared Intent exclusive mode, the subtree rooted by that node is locked explicitly in shared mode, and that explicit locking is being done at lower level with exclusive mode

Shared lock (S)

-T can tolerate concurrent readers but not concurrent updaters in R.

Exclusive lock (X)

-T cannot tolerate any concurrent access to R at all. Lock compatibility