Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery

DEADLOCK

- 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:

T_1:	write (A)	T_2:	write(A)
	write(B)		write(B)

Conn..

- 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	

Conn..

- **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.
- Approach2
  - Assign a unique timestamp to each transaction.
  - These timestamps only to decide whether a transaction should wait or rollback. schemes:
    - wait-die scheme
    - wound-wait scheme

Conn..

- **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

Conn..

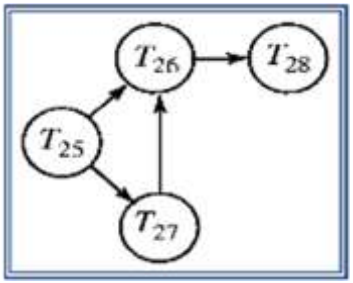
- never wait for older ones; they are rolled back instead.
- – A transaction may die several times before acquiring needed data item
- **Example**.
  - Transaction T_1,T_2,T_3 have time stamps 5,10,15,respectively.
  - if T_1 requests a data item held by T_2,then T_1 will wait.
  - If T_3 request a data item held by T_2,then T_3 will be rolled back.
- .**wound-wait scheme**
  - - Preemptive technique
  - - When transaction T_i requests a data item currently held by T_j,T_i is allowed to wait only if it has a timestamp larger than that of T_j. Otherwise T_j is rolled back
  - – Older transaction wounds (forces rollback) of younger transaction instead of waiting for it.
  - Younger transactions may wait for older ones.

Conn..

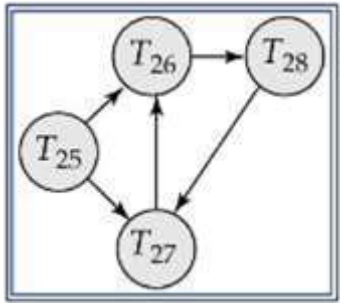
- **Example**
  - Transaction T_1,T_2,T_3 have time stamps 5,10,15,respectively.
  - if T_1 requests a data item held by T_2,then the data item will be preempted from T_2,and T_2 will be rolled back.
  - If T_3 requests a data item held by T_2,then T_3 will wait.
- **Deadlock Detection**
  - Deadlocks 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 $T_i \rightarrow T_j$ is in E, then there is a directed edge from T_i to T_j, implying that T_i is waiting for T_j 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.

Conn..

- **Wait-for graph without a cycle**



Wait-for graph with a cycle



Conn..

- **Recovery from deadlock**
  - The common solution is to roll back one or more transactions to break the deadlock.
  - Three action need to be taken
    - – Selection of victim

Conn..

  - Rollback
  - Starvation

- **Selection of victim**
  - Set of deadlocked transations,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**
  - Ensure that a transaction can be picked as victim only a finite number of times.

Conn..

- **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 T_1 reads record r_{a2} in file $F_{a.}$ Then,T_1 needs to lock the database, area $A_{1,and}$ F_a in IS mode, and finally lock r_{a2} in S mode.

Conn..

- **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 T_2 modifies record r_{a9} in file $F_{a.}$ Then,T_2 needs to lock the database,
      - area $A_{1,and}$ F_a in IX mode, and finally to lock r_{a9} 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.

Conn…

Thank You…….