More Locking

CS157B

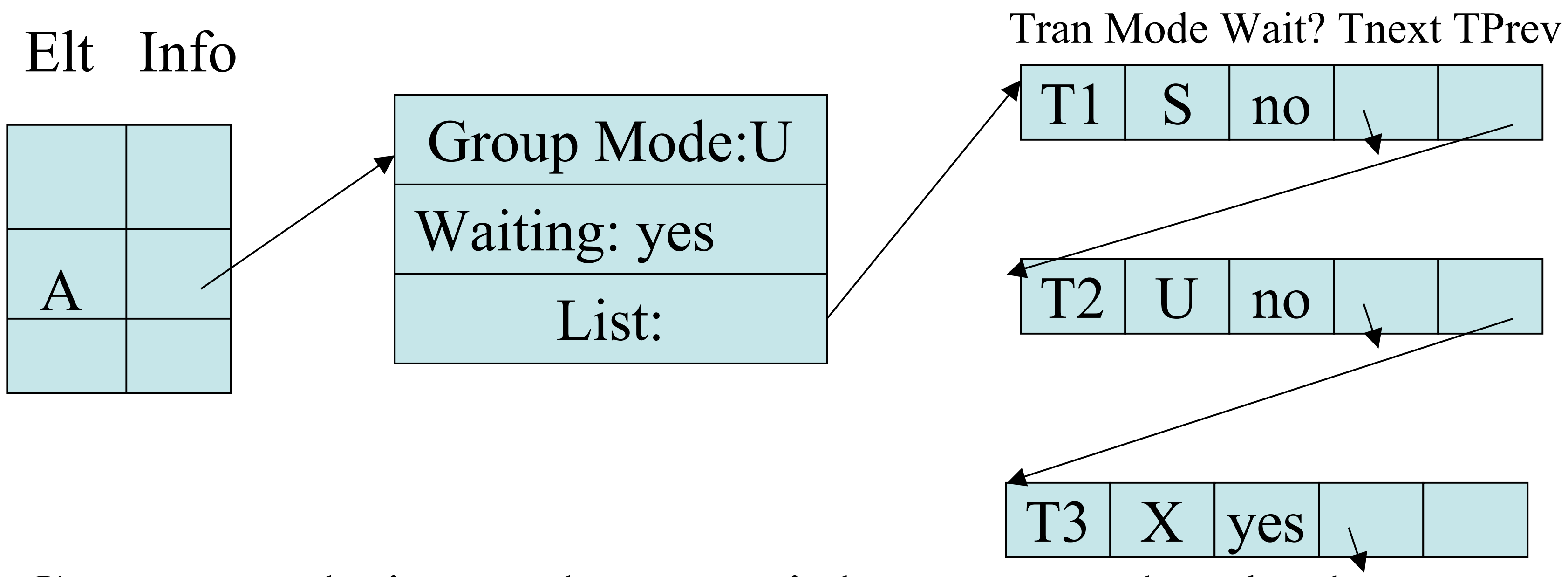
Chris Pollett

May 4, 2005.

Outline

- Handling Lock and Unlock Requests
- Lock Granularity
- The Tree Protocol

A Lock Table Entry



- Group mode is used as a quick test on what locks are currently held. S - indicates only shared. U - some shared one update, X - only one exlusive lock.
- Waiting indicates if some transaction is waiting for a lock on A.

Handling Locking

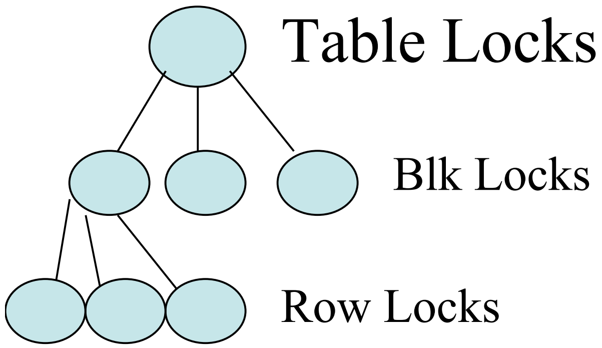
- To lock A:
  - We first check does there exists an entry for A?
    - If no, we create it, we add the transaction to the entry, and we grant the lock.
    - If yes, we check the group mode:
      - if the mode is S or U and the request is a shared lock, we grant it and add the transaction to the list.
      - If the mode is S and the request is U, we grant the lock request, and add the transaction to the list.
      - If the mode is X or U and the request is S, X or U, we deny the request, and add the transaction to the list but with Wait? having value ‘yes’.

Handling Unlocking

- If T unlocks A:
  - T’s entry in the list for A is deleted.
  - If T had a U lock then we change the group mode to S if there are any transaction still in the list.
  - If T had value a U or X and Waiting is ‘yes’ then we can grant to one of the waiting transactions the lock. This grant can be based on one of the following strategies:
    - First come first serve.
    - Priority to shared locks -- First come first serve to shared lock, then updates, then exclusive locks.
    - Priority to upgrading -- if T has U and is waiting for an X-lock, grant T first. Otherwise, follow one of the previous strategies.

Lock Granularity

- We want to be able to have hierarchies of possible locks -- from locks on only one element to a single lock for many elements.
- These hierarchies will typically come in the form of trees.
- For instance, a table lock would be higher in such a tree structure than a block level lock which is turn higher than a row lock.
- As another example, we might want to be able to lock single index entries or whole subtrees of a B-tree.
- An example of the advantage of having big locks is if a transaction is frequently accessing a table it is faster to only do one lock that one for each access.
- If Oracle sees you doing multiple reads on a table it tries to increase the granularity of the shared lock you have.



Warning Locks

- One way to manage such hierachies is to introduce a new kinds of lock called a warning locks, IS and IX in addition to S and X. The I indicates intention to acquire a lock. (The idea is similar to update locks)
- Let's assume we have three levels of locks: table, block, row.
- Let T be the tree of all locks currently held on table S.
  - Start at root of hierarchy. If we are at the element we want request either an S or X lock.
  - If not traverse down tree tell get to the element we want. For each node on this path, we request an IS or IX lock.

Warning Lock Compatibility Matrix

- Here's what the compatibility matrix looks like:

		Lock Requested			
		IS	IX	S	X
Lock held in mode	IS	Y	Y	Y	N
	IX	Y	Y	N	N
	S	Y	N	Y	N
	X	N	N	N	N

Phantoms and Handling Insertions Correctly

- Notice in our locking scheme we can only lock items that already exist.
- Consider the query where we sum the lengths of all movies made by Disney.
- While we are performing this query someone might insert a row with a Disney movie.
- If we repeat our query in the same transaction we might see a different sum. This problem tuple is called a *phantom* tuple.
- The solution to the problem is to require locking the whole table for reads or writes, but this come with a heavy performance hit.

The Tree Protocol

- Standard locking protocols make it very hard to do concurrency when doing locking with B-trees.
- The problem is we would typically need to lock a whole path down the tree in case the tree changes during an update. This prevents other people from looking at the tree.
- Instead, to get concurrency for B-trees, we don't use two phase locking and:
  - a transaction's first lock in the tree may be at any node of the tree (typically lock first node that could be changed by your transaction.)
  - subsequent locks can be obtained iff the transaction has the parent lock.
  - nodes can be unlocked at any time
  - a transaction may not relock a node it has released.