SE/CS 351

ACID isolation continued:

- the common scheduler
- update locks, lock-upgrading
- deadlocks

schedulers with read locks

- disadvantage of simple scheduler:
- write-disjoint transactions may have to wait for each other, although this is not necessary.
- Schedulers used in practice often have several types of locks, including a non-exclusive read lock.
  - several transactions can have read-locks on the same object.
- Such schedulers require more complex case distinctions for e.g. upgrading locks from read-locks to write-locks.

the common scheduler

- Akin to schedulers used in practice.
- uses several kinds of locks:
  - read locks, also known as shared locks (S locks)
    - Several transactions can have a read lock on the same object:
    - an object with read locks can only be read, not written.
  - write locks, also known as exclusive locks (X locks)
    - If an object has a write lock on x, no other lock can be set on x.
    - The owner can read and write the object.
    - A lone read lock on x can be *upgraded* to a write lock by owner.
  - update locks (a.k.a. upgrade locks),
    - Help in acquiring write locks in certain conditions
- Advantage: Objects that are only read can be accessed concurrently.

scheduling with shared locks

- s: $r_1[x]$, $r_2[x]$, c_2,$r_1[y]$, $r_3[z]$, $w_1[y]$, c_1,$w_3[x]$, c_3
- TA1: $r_1[x]$, $r_1[y]$, $w_1[y]$, c_1
- TA2: $r_2[x]$, c_2
- TA3: $r_3[z]$, $w_3[x]$ ____________ , c_3,

transaction TA2 not waiting any more.

transaction TA3 is waiting, cannot acquire write lock

problem with read and write locks:

- As long as several transactions have a read lock on the same object, a write lock cannot be acquired. Writing transactions have to wait.
- Without further precautions, a writing transaction might never be able to acquire the write lock, because new transactions continuously start to read: the writing transaction would be in a *live-lock:*
  - s: $r_1[x], r_2[x], r_3[x], c_2, r_4[x], r_5[x], c_3, c_4, r_6[x], \ldots\ldots$
  - TA1: $r_1[x]$, $w_1[x]$ ____________________________?

solution: update locks

- The first writing transaction gets a third type of lock, an **update lock** a.k.a. upgrade lock (U lock):
  - No new reader can access this object, before the write was de-queued and executed.
  - Once all readers have finished, the writing transaction gets the exclusive lock: this process is a *lock upgrade*
  - s: $r_1[x]$, $r_2[x]$, $r_3[x]$, c_2, c_3, $w_1[x]$, c_1,$r_4[x]$,.....
  - TA1: $r_1[x]$, $w_1[x]$ ____________,c_1
  - TA4: $r_4[x]$ ____________, ...

transaction TA1 waiting, but acquires update lock.

transaction TA4 is waiting, cannot acquire read lock

update locks continued

- Only one transaction can acquire an update lock
- Later transactions that try to do so will be blocked
- Often expressed in a matrix:
  - the columns denote the locks owned by other transactions that are already present.
  - the rows represent the lock that a transaction wants to acquire.

lock present (columns); lock requested (rows)

	S	U	X
S	y	n	n
U	y	n	n
X	n	n	n

y: lock granted and transaction not blocked

n: lock not granted and transaction blocked

Deadlocks

- example deadlocks
- deadlock prevention
- deadlock detection
- queue graph
- wait-for graph
- resource hierarchies

Deadlocks while attempting lock upgrade

- Transactions mostly read x before they write x.
- This results in a lock upgrade: the transaction first has a read lock on x, then upgrades this to a write lock on x.
- This process can result in a *deadlock*:
    - s: $r_1[x]$, $r_2[x]$, ... ?
    - TA1: $r_1[x]$, $w_1[x]$ ______________?
    - TA2: $r_2[x]$, $w_2[x]$ ________?
- This is only one example of a deadlock.
- Alternative: Early declaration of intention to upgrade:
- SQL: “SELECT ... FOR UPDATE”
- should acquire higher lock (but is up to the scheduler).

Early declaration of intention to upgrade

- The aforementioned *deadlock*:
  - s: $r_1[x]$, $r_2[x]$, ... ?
  - TA1: $r_1[x]$, $w_1[x]$ _____________?
  - TA2: $r_2[x]$, $w_2[x]$ _______?
- Alternative: Use "SELECT ... FOR UPDATE"
- This is a read operation that expresses intent to write.
- We indicate this operation with capital R[] in the schedule (this is an extension of the basic transaction model):
- Implementation is left to the particular database.
- One natural possible semantics in our scenario:
- R[] requests an update lock and an exclusive lock.

Early declaration of intention to upgrade

- The aforementioned *deadlock*:
  - s: $r_1[x]$, $r_2[x]$, ... ?
  - TA1: $r_1[x]$, $w_1[x]$ _______________?
  - TA2: $r_2[x]$, $w_2[x]$ ________?
- Alternative using R ("SELECT ... FOR UPDATE")
  - s: $R_1[x]$, $w_1[x]$, c_1 $R_2[x]$, $w_2[x]$, c_2
  - TA1: $R_1[x]$, $w_1[x]$, c_1
  - TA2: $R_2[x]$ ______________, $w_2[x]$, c_2

transaction TA2 is waiting, because TA1 has the update lock on x

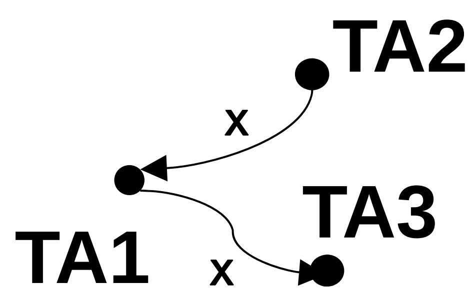
- the simple scheduler: all reads are "FOR UPDATE"

Finding deadlocks: Queue graph (QG)

- is a directed graph, edge-labelled graph
  the nodes are transactions.
- If a transaction TAn is entering a queue on object x, and will get the lock on x eventually from TAm, then an edge is drawn from TAm to TAn, with edge label x.
- TAm is either the transaction holding the lock on x, or the predecessor of TAn in the waiting queue for x.
- s: $r_2[x]$, $r_2[y]$, $w_2[x]$, c_2, $R_1[x]$, $w_1[x]$, c_1, $r_3[x]$, $w_3[z]$, c_3
- TA1: $R_1[x]$ ________________, $w_1[x]$, c_1
- TA2: $r_2[x]$, $r_2[y]$, $w_2[x]$, c_2
- TA3: $r_3[x]$ ________________________, $w_3[z]$, c_3

t

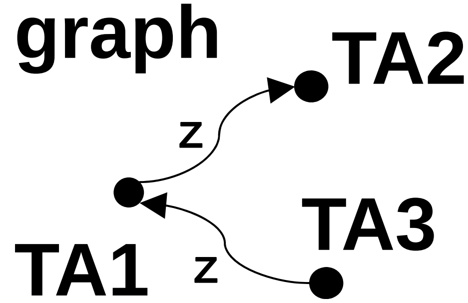
queue graph at time t



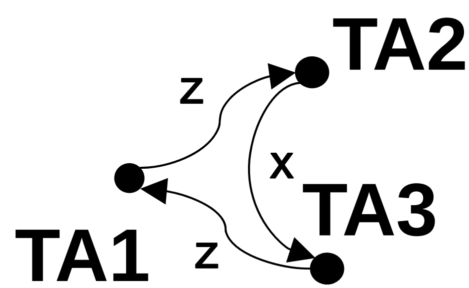
Finding deadlocks

- Cycles in the QG are deadlocks:
- The cycle will not be resolved by any user command because no transaction in the cycle can commit.
- Continuous deadlock-detection: whenever a transaction enters a waiting queue, a check for cycles is performed.
- Periodic deadlock-detection: From time to time, a check for cycles is performed.
- We assume continuous deadlock detection.

queue-for graph



Deadlock:



Deadlock prevention by application programmer

- A strategy to reduce deadlocks:
- If possible, access different data items always in the same order.
- Example: purchases in a shop always access the central balance account b and the central tax account t.
- All transactions access first the tax and then the balance.

s: $w_1[t]$, $w_1[b]$, c_1, $w_2[t]$

TA1: $w_1[t]$, $w_1[b]$, c_1

TA2: $w_2[t]$ ______________

TA2 and TA1 are supposed to be issued by different programs

prevents this deadlock:

s: $w_1[t]$, $w_2[b]$,............?

TA1: $w_1[t]$, $w_1[b]$ ___?

TA2: $w_2[b]$, $w_2[t]$ ___?