Problem with Locks

DEADLOCK and DIRTY READ
Rules of Locks

• Several transactions can hold read locks on same data item
• Only one holds write lock on any one data item
• Transaction which tries for exclusive lock and cannot get it goes to sleep
• When exclusive lock is released, some sleeping transaction wakened
# Deadlock

<table>
<thead>
<tr>
<th>Transaction A</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read X</td>
<td>Read Y</td>
</tr>
<tr>
<td>Read Y</td>
<td>Read X</td>
</tr>
<tr>
<td>Write X</td>
<td>Write Y</td>
</tr>
</tbody>
</table>
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read Y
Read X
Write Y
Deadlock

Transaction A
Read X
Write X
Read Y

Transaction B
Writelock Y
Read Y
Read X
Write Y
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read X
Read Y
Write Y

Writelock X
Writelock Y
Readlock Y
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read Y
Read X
Write Y

Writelock X
Readlock Y
Cannot get lock due to B’s Writelock -- A sleeps
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read Y
Read X
Write Y

Writelock X
Readlock Y

Cannot get lock due to B’s Writelock -- A sleeps
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read Y
Read X
Write Y

- Cannot get lock due to B’s Writelock -- A sleeps
- Cannot get lock due to A’s Writelock -- B sleeps
Deadlock

Transaction A
Read X
Read Y
Write X

Transaction B
Read Y
Read X
Write Y

Readlock Y
Cannot get lock due to B's Writelock -- A sleeps

Writelock X
Cannot get lock due to A's Writelock -- B sleeps

Each sleeps, waiting for the other
Deadlock Prevention Strategy

- Change some rule about locks
- Normally: Can’t get lock ⇒ Sleep on Queue
- New rule: Can’t get lock ⇒ Some transaction may die
- Strategy: find something (X) that always happens when there is going to be a deadlock:
  - If **deadlock** about to happen Then **X** is true
- Turn it around to make the rule:
  - If **X** is true Then **deadlock** is about to happen
  - And someone must die to prevent it.
A 2PL Solution

• New Rule: Get all locks at the beginning of transaction
• If a needed lock is not available
  – Abort transaction
  – wait a little
  – Restart
• Notice change in rule about locks
• Problem: reduces concurrency
• Called Conservative 2PL
• Will see other solutions later
Second Problem: Dirty Read

- Source: DB policy may allow time between last unlock and DB promise to remember all.
- So this may occur
  - $T_1$ writes A
  - $T_1$ last unlock
  - $T_2$ reads A
  - $T_1$ fails
- Value of A is now INVALID
- $T_2$ must fail as well.
Dirty Read

Transaction A
Read X
Write X

Transaction B

Read X
Dirty Read

Transaction A
- Read X
- Write X

Transaction B
- Writelock X
- Read X
Dirty Read

Transaction A
  Read X
  Write X
  Writelock X
  Unlock X

Transaction B
  Read X
Dirty Read

Transaction A
  Read X
  Write X
  Writelock X
  Unlock X

Transaction B
  Readlock X
  Read X
Dirty Read

Transaction A
Read X
Write X
Unlock X

Transaction B
Read X
Unlock X
Writelock X
Readlock X
Dirty Read

Transaction A
Read X
Write X
Unlock X
ABORT and ROLL BACK

Transaction B
Writelock X
Readlock X
Read X
Unlock X
ABORT and ROLL BACK
Dirty Read

Transaction A
Read X
Write X
Unlock X
ABORT and ROLL BACK

Transaction B
Readlock X
Read X
Unlock X

Dirty Read

Writelock X
Dirty Read

Transaction A
- Read X
- Write X
- Unlock X

Transaction B
- Writelock X
- Read X
- Unlock X
- Readlock X

ABORT and ROLL BACK

Cascading ROLL BACK
A 2PL Solution

• New Rule: Keep all locks until commit
• No uncommitted write can be read
• Called Strict 2PL
• Problem: reduces concurrency
• Will see other solutions later