Database Concurrency Control

Robert M. Keller
Harvey Mudd College
April 2004
Modeling with Sequence Diagrams

- Sequence diagrams are from UML
- Each line in a diagram corresponds to some object, in our case:
  - Database items (relations, tuples, ...)
  - Transaction
- We use message direction to show what is being read vs. written, **not** the originator of the message. (All messages originate from transactions.)
Example 1

$T_1$ writes $A$, (then) $T_2$ reads $A$
Example 2

$T_2$ reads $A$, $T_1$ writes $A$
Assumption

🔹 When a transaction writes, the value it writes is an unknown (but fixed) \textit{function} of the values it has read (and only those).

🔹 In Example 1, $T_1$ writes $f()$ to $A$, then $T_2$ reads $f()$ as the value of $A$.

🔹 In Example 2, $T_2$ reads $A_0$ (the initial value) as the value of $A$, then $T_1$ writes $f()$ to $A$. 
Example 3

Final value of $A$ is $g(f())$
Example 4

Final value of A is _______
Possible Phenomena

- Lost Update
- Inconsistent Retrieval
Lost Update Example
(compare Example 3)

Final value of A is _______
Inconsistent Retrieval Example

Final value of C is ________
Correctness

- A set of transactions is assumed to operate \textbf{correctly} if the net effect is the same as \textbf{some serial} execution.
Interleavings

- An interleaving consists of the steps of all transactions in some order.
- All steps of each transaction must be included.
- Within each transaction, the steps are in the same order as in the serial case.
Serializable Interleaving

- An interleaving is serializable if the net effect is the same as some serialization of the transactions.
Is this interleaving serializable?

Equivalent to
How about this one?

Equivalent to
And This?

Equivalent to
Conflicts

◆ Two operations of different transactions conflict if either:
  ▶ One reads from a data item to which the other reads (R-W conflict)
  ▶ One writes to a data item to which the other writes (W-W conflict)
◆ If two overlapping transactions have conflicting operations, then there is an order-dependence among the operations.
◆ We want to preserve such an order in any serialized interleaving.
Conflict Example

Must preserve the order $T_2 T_1$
Conflict Analysis

- When two operations in different transactions conflict, the order of those two operations must be preserved in any equivalent interleaving.

- We may have “conflicting” requirements of this form.
Visualizing Conflicts

- Scan the read/write sequences for each data item.
- If a write operation to the item is present with other reads or writes in different transactions, then there is a conflict between those transactions.
Conflict Orders

Requires $T_2 T_1$. 

WW conflict
Conflicting Conflict Orders

Requires $T_1 T_2$; we can’t have this and $T_2 T_1$. 
Non-Conflict Orders

Requires nothing.

No RR conflict
Conflict Graph for an Interleaving

- The conflict requirements can be expressed as a graph:
  - Transactions are nodes
  - There is an arrow if the transactions have conflicting operations, indicating the necessary serial order for equivalence with the interleaving.

\[ T_1 \rightarrow T_2 \]
Conflict Graph Example
Conflict Graph Example
Conflict Graph Example

T₁
f

T₂
h

g
Result 1

- If there is a serializable interleaving, then the conflict graph is acyclic.
- Proof:
  - Assume that there is a serializable interleaving $J$.
  - Let $K$ be a *serial* interleaving equivalent to $J$.
  - If operations in two different transactions conflict in $K$, then the implied arrows in the conflict graph must all be the same direction, since *one transaction occurs before the other* in $K$.
  - Therefore, if there is an arrow from $T_i$ to $T_j$ in the conflict graph, it must be the case that $T_i$ *precedes* $T_j$ in $K$.
  - But $K$ is a *linear* order, so there can be no cycle.
Example of Result 1: Interleaving J (non-serial):

\[ A B C \]
\[ A_0 B_0 C_0 \]
Example of Result 1:
Interleaving K (equiv. serial):

Acylic graph
Result 2

- If the conflict graph is acyclic, then the interleaving is serializable.

- Proof:
Example of Result 2:
Summary of Results 1&2

An interleaving is serializable iff the conflict graph is acyclic.
Serializability Requirement

- The execution of a set of transactions must be serializable.
Alternatives for Ensuring Serializability

- Allow only one transaction at a time.
- Some more liberal policy that guarantees serializability.
- Analyze transactions statically; only allow compatible transactions to run concurrently.
- Construct conflict graph dynamically; abort (rollback) a transaction if it can create a cycle.
- Use locking.
- Use time stamps.
Locking

✓ Simple model:
  ▶ Only one kind of lock
  ▶ Lock data item before any read or write
  ▶ Unlock data item after reads and writes

✓ By itself, doesn’t guarantee serializability:
  ▶ Example?

✓ Can produce deadlocks if no further protocol imposed.
Locking Implies Reading or Writing

T1
LOCK A
UNLOCK A

T2
LOCK A
UNLOCK A
LOCK B
UNLOCK B

LOCK B
UNLOCK B

A

T1

B

T2
2-Phase Locking Policy (2PL)

Every transaction is divided into two phases that occur in sequence:

**Phase I:** All requesting of locks (no releasing)

**Phase II:** All releasing of locks (no further requesting)

[Similar-sounding, but distinct idea: 2-Phase Commit used in distributed databases.]
2PL Theorem

- Under 2-phase locking, every interleaving is serializable.
- Proof:
  - Define the lock point of a transaction within an interleaving to be the point at which it acquires the last of the locks it requests.
  - Order the transactions by lock point, earliest to latest.
  - **Claim**: The serial interleaving in which transactions are done in lock point order is equivalent to the original interleaving.
Proof of Claim

◆ Claim: The serial interleaving in which transactions are done in lock point order is equivalent to the original interleaving.

◆ Proof: If $T_i$ is before $T_j$ in the lock point ordering, then $T_j$ cannot read or write an item A until after $T_i$ has released the lock on A.

◆ Therefore, in the conflict graph there can be no arrow from $T_j$ to $T_i$.

◆ Hence the conflict graph is a partial order consistent with the linear lock point order. In particular, the conflict graph is acyclic.
Non-2PL Example

T1
LOCK A
UNLOCK A

T2
LOCK A
UNLOCK A
LOCK B
UNLOCK B

conflict graph
(non-serializable)

T1 -- B -- T2
2PL Example
(lock points underlined)

**T1**
- Lock A
- Lock B
- **Lock C**
- Unlock A, B, C

**T2**
- Lock A (waiting)
- Unlock A, B, C
- Lock C (waiting)
- (C acquired)

**T3**
- Lock B (waiting)
- Unlock B, C
- Lock C
- (B acquired)
- Unlock A, C
2PL Example
(lock points underlined)

T1
Lock A
Lock B

Lock C
Unlock A,B,C

T2
Lock A (waiting)

T3
Lock B (waiting)

Lock C (waiting)

Unlock B, C

(B acquired)

Unlock A,C

(C acquired)

conflict graph

consistent with lockpoint linear order

43
Same Transactions, Different Lock Point Order

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lock A (waiting)</td>
<td>Lock A</td>
<td>Lock B</td>
</tr>
<tr>
<td></td>
<td>(A acquired)</td>
<td>Lock C</td>
<td>(waiting)</td>
</tr>
<tr>
<td></td>
<td>Unlock A, C</td>
<td>(waiting)</td>
<td>(C acquired)</td>
</tr>
<tr>
<td></td>
<td>Unlock B, C</td>
<td></td>
<td>Unlock B, C</td>
</tr>
</tbody>
</table>

conflict graph
Deadlock Danger

- 2PL itself does not avoid deadlock
  - Example?
- Need to impose additional mechanism to prevent deadlock
Serialization with Read- and Write-Locks

- So far, we have assumed all locks are the same (= write-locks).
- This is overly-restrictive: inhibits concurrent reads.
- When both types of locks are used, the cycle criterion for serializability can be applied.
- The conflict graph construction is modified, however.
- 2PL still works!
Modified Conflict Graph Construction (for read/write locks)

- If Ti read- or write-locks A, and Tj is the next transaction in the interleaving to write-lock A, then there is an arc from Ti to Tj.

- If Ti write-locks A, and Tk read-locks A after Ti unlocks A, but before any other transaction write-locks A, then there is an arc from Ti to Tk.
**Example: Is this serializable?**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>T2</td>
<td>T3</td>
<td>T4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wlock A</td>
<td>Rlock B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wlock A</td>
<td>Unlock A</td>
<td>Unlock B</td>
<td></td>
</tr>
<tr>
<td>Unlock A</td>
<td>Unlock B</td>
<td>Unlock B</td>
<td>Unlock B</td>
<td></td>
</tr>
<tr>
<td>Unlock A</td>
<td>Unlock A</td>
<td>Unlock B</td>
<td>Unlock B</td>
<td></td>
</tr>
<tr>
<td>Unlock B</td>
<td>Rlock B</td>
<td>Unlock A</td>
<td>Unlock A</td>
<td></td>
</tr>
<tr>
<td>Unlock B</td>
<td>Unlock B</td>
<td>Unlock B</td>
<td>Unlock B</td>
<td></td>
</tr>
</tbody>
</table>