***Lost Update Problem***

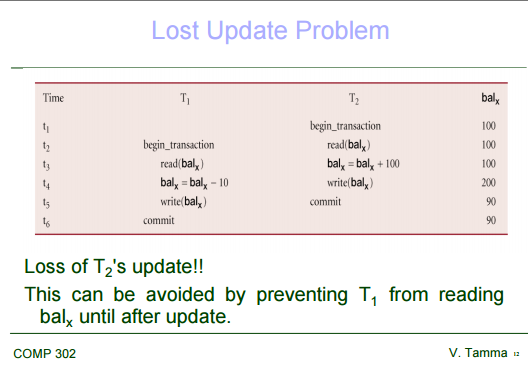
Successfully completed update is overridden by another user.

Example:

• T1 withdraws £10 from an account with bal x, initially £100.

• T2 deposits £100 into same account

. • Serially, final balance would be £190.



---

The "lost update" problem relates to concurrent reads and updates to data, in a system where readers do not block writers. It is not necessary for the transactions to be exactly simultaneous.

1. Session #1 reads Account A, gets 100.
2. Session #2 reads Account A, gets 100.
3. Session #2 updates Account A to 150 (+50) and commits.
4. Session #1 updates Account A to 120 (+20) and commits.

In this scenario, because Session #1 does not know that another session has already modified the account, the update by Session #2 is overwritten ("lost").

There are several ways to solve this, e.g. version numbers or before-and-after compares.

-------------

Need for Concurrency Control

***NEED FOR CONCURRENCY CONTROL***

**Concurrency Control**

Process of managing simultaneous operations on the database without having them interfere with one another.

• Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.

• Although two transactions may be correct in themselves, interleaving of operations may produce

an incorrect result.

**Need**

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

1) **The Lost Update Problem**

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

Successfully completed update is overridden by another user.

Example:

• T1 withdraws £10 from an account with balx, initially £100.

• T2 deposits £100 into same account.

• Serially, final balance would be £190.

*Loss of T2's update!!*

This can be avoided by preventing T1 from reading balx until after update.

**The Temporary Update (or Dirty Read) Problem**

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

Occurs when one transaction can see intermediate results of another transaction before it has committed.

Example:

• T4 updates balx to £200 but it aborts, so balx should be back at original value of £100.

• T3 has read new value of balx (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.

Problem avoided by preventing T3 from reading balx until after T4 commits or aborts.

**The Incorrect Summary Problem**

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

Occurs when transaction reads several values but second transaction updates some of them during execution of first.

Example:

• T6 is totaling balances of account x (£100), account y (£50), and account z (£25).

• Meantime, T5 has transferred £10 from balx to balz, so T6 now has wrong result (£10 too high).

 Problem avoided by preventing T6 from reading balx and balz until after T5 completed updates.