What Can Go Wrong?

14

- Concurrent process may end up violating Isolation property of transaction if not carefully scheduled
- Transaction may be aborted before committed
  - undo the uncommitted transactions
  - undo transactions that sees the uncommitted change before the crash

Schedule

15

- A schedule S of n transactions T1,T2,…Tn is an ordering of the operations of the transactions subject to the constraint that,
  - for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti.
- Informally, a schedule is a sequence of interleaved actions from all transactions
- Example:

S_a: R1(A),R2(A),W1(A),W2(A), Abort1,Commit2;

T1	T2
Read(A)	
	Read(A)
Write(A)	
	Write(A)
Abort T1	
	Commit T2

Scheduling Transactions

16

- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules:* For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule:* A schedule that is equivalent to some serial execution of the transactions.

Serial Schedule

17

T1	T2
R(A)	
A := A+100	
R(B)	
B := B+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2

S: R1(A),W1(A),R1(B), W1(B), R2(A), W2(A), R2(B), W2(B)

T1 T2

Adapted from M. Balazinska

A Serializable Schedule

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
R(B)	
B := B+100	
	R(B)
	B:= B*2

Notice: this is not a serial schedule, i.e., there is interleaving of operations

Net effect is the same as the serial schedule

S: R1(A),W1(A), R2(A), W2(A), R1(B), W1(B), R2(B), W2(B)

A Non-Serializable Schedule

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2
R(B)	
B := B+100	

What's different?

Ordering of operations is not consistent

S: R1(A),W1(A), R2(A), W2(A), R2(B), W2(B), R1(B), W1(B)

Conflict operations

- Two operations in a schedule are said to be *conflict* if they satisfy all three of the following conditions:
  (1) They belong to different transactions
  (2) They access the same item A;
  (3) At least one of the operations is a write(A)

Example in Sa: R1(A), R2(A), W1(A), W2(A), A1, C2;

- R1(A),W2(A) conflict, so do R2(A),W1(A),
- R1(A), W1(A) do not conflict because they belong to the same transaction,
- R1(A),R2(A) do not conflict because they are both read operations.

Conflicts

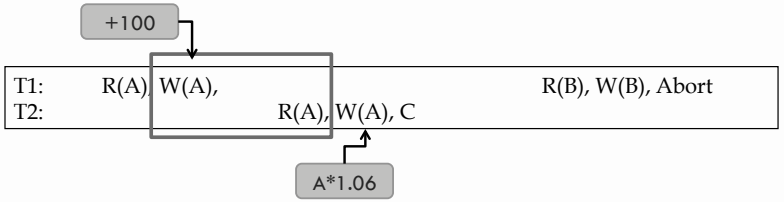
What can go wrong:

- Reading uncommitted data (WR), aka "dirty reads"
- Unrepeatable reads (RW)
- Lost updates (WW)

Reading Uncommitted Data

22

- WR Conflicts, "dirty reads"



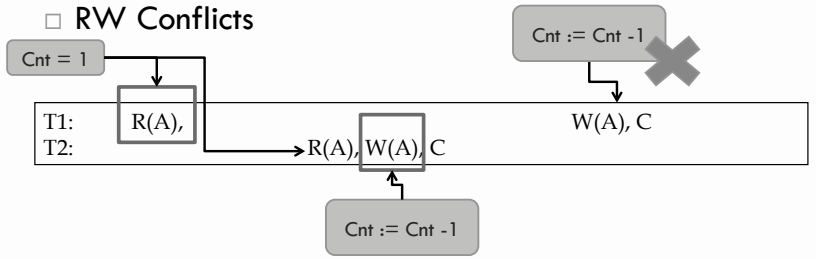
- What's the problem?

T2 has read an updated value A, which is later aborted!

Unrepeatable Reads

23

- RW Conflicts



- What's the problem?

T2 has changed A that has been read by T1, while T1 is still in progress

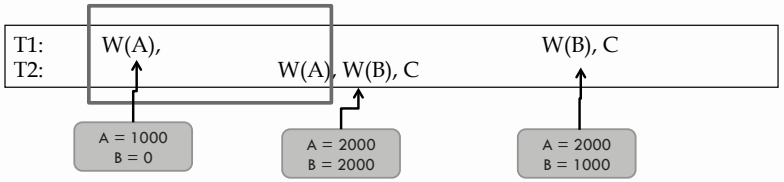
Lost Updates

24

T1: sets salaries to $1000
T2: sets salaries to $2000

Require: A and B salaries to be equal

- Overwriting uncommitted data
- WW Conflicts



- What's the problem?

T2 overwrites T1's update of A, while T1 is still in progress.

Schedules with Aborts

25

Withdraw $100 from A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

Bank reads A, B, and credits 6% interest

What now? (T2 read a value that is no longer valid)

Options

- If T2 did not commit, we abort T1 and *cascade* to T2.
- But T2 committed, so we cannot undo
- This schedule is *unrecoverable*

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

Aborting a Transaction

- If a transaction *Ti* is aborted, all its actions have to be undone. Not only that, if *Tj* reads an object last written by *Ti*, *Tj* must be aborted as well!
- Most systems try to avoid such *cascading aborts*
  - If *Ti* writes an object, *Tj* can read this only after *Ti* commits.

Aborting a Transaction (cont'd)

- In order to *undo* the actions of an aborted transaction, the DBMS maintains a *log* in which every write is recorded.
- This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up.

Recoverable Schedules and Avoid Cascading Aborts

Recoverable

- Aborting T1 requires aborting T2
  - But T2 has already committed!
- A recoverable schedule is one in which this cannot happen.
  - i.e. a Xact commits only after all the Xacts it "depends on" (i.e. it reads from) commit.

	No		Yes	
	T1	T2	T1	T2
	R(A)		R(A)	
	W(A)		W(A)	
		R(A)		R (A)
		W(A)		W (A)
		Commit	Commit	
	Abort			Commit

Avoid cascading abort (ACA)

- Aborting T1 requires aborting T2!
- Aborting a Xact can be done without cascading the abort to other Xacts.
- A Xact only reads data from committed Xacts.
- ACA implies recoverable (but not vice-versa!)

T1	T2	T1	T2
R (A)		R (A)	
W (A)		W (A)	
	R (A)	Commit	
	W (A)		R (A)
Abort			W (A)

Example

W1(X), R2(Y), R1(Y), R2(X), C2, C1

T1, T2: W1(X), R1(Y), R2(Y), R2(X)

- Serializable: Yes, equivalent to T1,T2
- Recoverable: No. Yes, if C1 and C2 are switched
- ACA: No.

  Yes, if T1 commits before T2 reads X.