

Transaction Management

Chapter 16

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 1

Transactions

- Foundation for concurrent execution and recovery in DBMS
- Transaction is an **atomic** unit of work
 - E.g., Debit \$500 from my bank account
- Transaction consists of multiple actions
- For performance, DBMS can **interleave** actions from different transaction
- Must guarantee same result as executing transactions **serially**

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 2

Example 1

Read (A);
 Check (A > \$25);
 Pay (\$25);
 A = A - 25;
 Write (A);

```

    graph TD
      BB1[Bank Balance : $100] --> YouRead[Read Balance: $100]
      BB1 --> OtherRead[Read Balance: $100]
      YouRead --> YouCheck{Sufficient funds?}
      OtherRead --> OtherCheck{Sufficient funds?}
      YouCheck -- Yes --> YouPay[Pay $25]
      OtherCheck -- Yes --> OtherPay[Pay $25]
      YouPay --> YouNew[New balance: $75]
      OtherPay --> OtherNew[New balance: $75]
      YouNew --> BB2[Bank Balance : $75]
      OtherNew --> BB2
  
```

- Interleaving actions of different transactions can cause inconsistency
- DBMS should provide users an illusion of a single-user system
- Could insist on admitting only one transaction at a time
 - Lower utilization: CPU / IO overlap
 - Long running queries starve other queries, reduce overall response time

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 3

Example 2

```

    graph TD
      T1[You] --> R1[Read (A);  
Check (A > $25);  
Pay ($25);  
A = A - 25;  
Write (A);]
      T1 --> R2[Read Balance: $100]
      T1 --> Q1[Sufficient funds?]
      T1 --> P1[Pay $25]
      T1 --> R3[New balance: $75]
      T1 --> B1[Bank Balance: $100]
      T1 --> B2[Bank Balance: $75]
      T2[ ] --> R4[Read (A);]
      T2 --> R5[Check (A > $25);]
      T2 --> R6[A = A - 25;]
      T2 --> R7[Write (A);]
      T2 --> R8[Read Balance: $100]
  
```

- DBMS must also guarantee that changes made by partially completed transactions are not seen by other transactions

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 4

The ACID Properties

- TM** (Trans. Mgmt. logging) --- **Atomicity**: All actions in the Xact happen, or none happen.
- User** --- **Consistency**: Consistent DB + consistent Xact \Rightarrow consistent DB
- CC** (Concurrency Ctrl. locking) --- **Isolation**: Execution of one Xact is isolated from that of other Xacts.
- RM** (Recovery Mgmt. WAL, ...) --- **Durability**: If a Xact commits, its effects persist.

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 5

Schedules

- Transaction seen as a list of actions
 - Read(X), Write(X), commit, abort
- Schedule: An interleaving of the actions from a set of transactions
 - Complete Schedule: Each transaction ends in commit or abort
 - Serial Schedule: No interleaving actions among transactions
- Initial State + Schedule = Final State

	T1	T2
	begin	
	R(A)	
	W(A)	
		begin
		R(B)
		W(B)
	R(C)	
	W(C)	
		commit
	abort	

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 6

Acceptable Schedules

- Serial schedule is "isolated" and "consistent"
- Serializable schedule:**
 - Final state is what **some complete** serial schedule of **committed** transactions would have produced.
 - Can different serial schedules have different final states?
 - Yes, and we assume **all** are OK!
 - Aborted Xacts?
 - Ignore them for a little while (made to 'disappear' using logging)
 - Other external actions (besides R/W to DB)
 - e.g. display a printed value, fire a missile, ...
 - Assume (for this class) these values are written to the DB, and can be undone

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 7

Anomalies due to Interleaving

- Two actions on the same data "object" conflict if at least one is a write()
- Three anomalous situations for transactions T1 and T2
 - Write-read (WR) conflict
 - Read-write (RW) conflict
 - Write-write (WW) conflict

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 8

WR Conflict

- "Dirty read"
- Could lead to a non-serializable execution
- Example:
 - @Start (A,B) = (1000, 100)
 - End (990, 210)
 - T1→T2:
 - (900, 200) → (990, 220)
 - T2→T1:
 - (1100, 110) → (1000, 210)

T1: Transfer \$100 from A to B	T2: Add 10% interest to A & B
begin	
R(A) /A -= 100	begin
W(A)	
	R(A) /A *= 1.1
	W(A)
	R(B) /B *= 1.1
	W(B)
	commit
R(B) /B += 100	
W(B)	
commit	

Database Inconsistent

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 9

RW Conflicts

- “Unrepeatable read”
- T1 reads the value of object X, then T2 updates X (before T1 has committed)

T1	T2
R(X) = 3	
	W(X) = 5
R(X) = 5	
Commit	
	Commit

Why isn't this schedule serializable?

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

10

WW Conflict

- Overwriting uncommitted data
- T2 overwrites what T1 wrote
- Usually occurs in conjunction with other anomalies
 - Unless you have “blind writes”

Students in same group (X and Y) should get the same project grade

T1 (Prof)	T2 (GSI)
W(X) = A	
	W(X) = B
	W(Y) = B
W(Y) = A	
Commit	
	Commit

Why isn't this schedule serializable?

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

11

What about aborts?

- Serializable schedule: Effect equivalent to a serial schedule of *committed* transactions
- As if aborted transactions *never happened*

T1 (Prof)	T2 (GSI)
	W(X) = B
	W(Y) = B
W(X) = A	
W(Y) = A	
Abort	
	Commit

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

12

What about aborts?

- How does one undo the effects of a transaction T1?
 - We'll cover this in logging and recovery
- What if another transaction T2 sees these effects??
 - Must abort and undo T2 too!
 - (Called **cascading abort**)

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

13

What about aborts?

- Can we always undo effects of aborted transactions?

No! Not all schedules are recoverable

Example:

T1 reads, decrements account balance by \$100

T2 adds 5% interest

T1	T2
R(balance) = 1000	
W(balance) = 1000 - 100	
	R(balance) = 900
	W(balance) = 900 * 1.05
	Commit
Abort	

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

14

What about aborts?

- Additional goals:
 - Recoverable schedule** - Transactions commit only after all transactions whose changes they have read commit.
 - Avoid cascading aborts (ACA)** - Transactions read only the changes of committed transactions.

3/29/09

EECS 484: Database Management Systems, Kristen LeFevre

15

Intro to Locking

- What can DBMS do to guarantee an acceptable schedule?
- *Locking* commonly used to control access to shared resources (data objects) among concurrent transactions
- Locking implemented by software module called a *lock manager*
 - Transactions must request lock before performing actions (R or W) on object
 - Release lock when done
 - Following *locking protocol*
 - Lock manager can't always grant lock right away (maintain a *wait queue*)

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 16

Intro to Locking

- Lock modes: Shared (S), Exclusive (X)
- Lock manager stores (XID, RID, mode) triples
- Lock compatibility:

		Existing Lock		
		NL	S	X
Requested Lock	S	Yes	Yes	No
	X	Yes	No	No

T1: S lock on R101?

T3: X lock on R100?

T2: S lock on R100?

T3: X lock on R102?

(Simplified) Lock Table

XID	RID	mode
T3	R101	S
T1	R100	X
T2	R101	S

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 17

Coming Up

- Locking protocols with desirable properties
 - E.g., serializable, recoverable, avoid cascading aborts
- How does the DBMS provide durability?
 - Discussion of recovery manager

3/29/09 EECS 484: Database Management Systems, Kristen LeFevre 18
