ICS 421 Spring 2010

Transactions & Concurrency Control (i)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa

ACID Properties

4 important properties of transactions

- **Atomicity**: all or nothing
  - Users regard execution of a transaction as atomic
  - No worries about incomplete transactions
- **Consistency**: a transaction must leave the database in a good state
  - Semantics of consistency is application dependent
  - The user assumes responsibility
- **Isolation**: a transaction is isolated from the effects of other concurrent transaction
- **Durability**: Effects of completed transactions persists even if system crashes before all changes are written out to disk

Scheduling Transactions

- *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.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

Example: Transactions & Schedules

T1: BEGIN
A=A+100
B=B-100
END

Transfer $100 from B's a/c to A's a/c.

T2: BEGIN
A=1.06*A
B=1.06*B
END

Credit interest to both a/c

T1	T2
R(A)	
A=A+100	
W(A)	
R(B)	
B=B-100	
W(B)	
	R(A)
	A=1.06*A
	W(A)
	R(B)
	B=1.06*B
	W(B)

T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=1.06*A
	W(A)
R(B)	
B=B-100	
W(B)	
	R(B)
	B=1.06*B
	W(B)

Conflict Serializability

- Two operations in a schedule **conflict** if
  - They belong to different transactions **AND**
  - They access the same item X **AND**
  - At least one of them is a write
- Two schedules are **conflict equivalent** if the order of any two conflicting operations is the same in both schedules.
- A schedule is **conflict serializable** if it is conflict equivalent to some serial schedule

Example: Conflicts & Schedules

T1	T2
R(A)	
A=A+100	
W(A)	
R(B)	
B=B-100	
W(B)	
	R(A)
	A=1.06*A
	W(A)
	R(B)
	B=1.06*B
	W(B)

T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=1.06*A
	W(A)
	R(B)
	B=1.06*B
	W(B)
R(B)	
B=B-100	
W(B)	

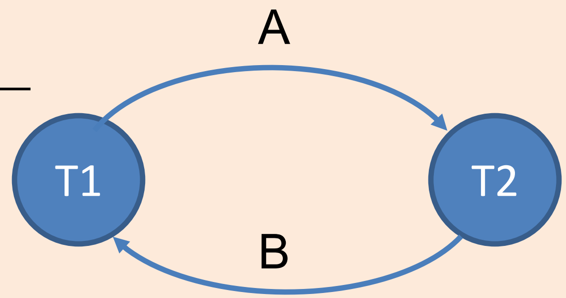
T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=1.06*A
	W(A)
R(B)	
B=B-100	
W(B)	
	R(B)
	B=1.06*B
	W(B)

Testing for Conflict Serializability

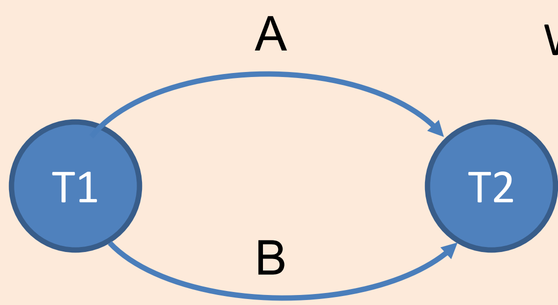
- Construct a **dependency** or **serialization Graph**
  - One node per transaction
  - For each object X
    - If Ti:W(X) followed by Tj:R(X) or Tj:W(X), then add edge (Ti, Tj)
    - If Ti:R(X) followed by Tj:W(X), then add edge (Ti,Tj)
- **<u>Theorem</u>**: Schedule is conflict serializable if and only if its dependency graph is acyclic

Example: Dependency Graphs

T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=1.06*A
	W(A)
	R(B)
	B=1.06*B
	W(B)
R(B)	
B=B-100	
W(B)	



T1	T2
R(A)	
A=A+100	
W(A)	
	R(A)
	A=1.06*A
	W(A)
R(B)	
B=B-100	
W(B)	
	R(B)
	B=1.06*B
	W(B)



Strict Two-Phase Locking

- *Strict Two-phase Locking (Strict 2PL) Protocol*:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - All locks held by a transaction are released when the transaction completes
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic

Example (Strict 2PL)

- Consider the dirty read schedule

T1	T2
A=A+100	
	A=1.06*A
	Commit
B=B-100	
Abort	

A = 20

A = 120

Dirty read on A!

A = 127.2

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

With Strict 2PL, T2 can only access A when T1 aborts

Two-Phase Locking (2PL)

- Two-Phase Locking Protocol
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction can not request additional locks once it releases any locks.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

Example (Non-Strict 2PL)

- Consider the dirty read schedule

T1	T2
A=A+100	
	A=1.06*A
	Commit
B=B-100	
Abort	

A = 20

A = 120

Dirty read on A!

A = 127.2

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

With non-strict 2PL, T2 can still read uncommitted data if T1 aborts!

Lock Management

- Lock and unlock requests are handled by the lock manager
- Lock table entry:
  - Number of transactions currently holding a lock
  - Type of lock held (shared or exclusive)
  - Pointer to queue of lock requests
- Locking and unlocking have to be atomic operations
- Lock upgrade: transaction that holds a shared lock can be upgraded to hold an exclusive lock

Deadlocks

- Cycle of transactions waiting for locks to be released
  - Create a waits-for graph:
    - Nodes are transactions
    - There is an edge from Ti to Tj if Ti is waiting for Tj to release a lock
  - Periodically check for cycles in the waits-for graph
- DBMS has to either prevent or resolve deadlocks
- Common approach:
  - Detect via timeout
  - Resolve by aborting transactions

T1	T2
Req X(A)	**Req X(B)**
Gets X(A)	**Gets X(B)**
…	**….**
Req X(B)	
	Req X(A)