Database Management Systems

Lecture 3

Transactions. Concurrency Control (II)

Deadlocks - Detection

a. *waits-for graph*

- node / active transaction
- arc from T_i to T_j if T_i is waiting for T_j to release a lock

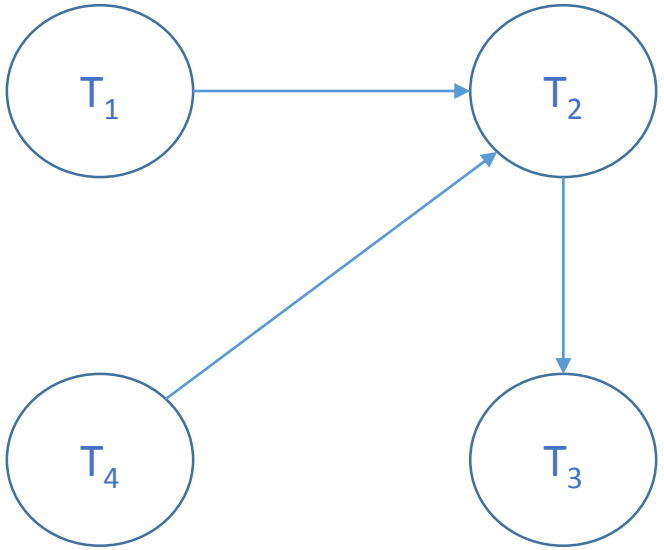
- cycle in the graph => deadlock
- DBMS periodically checks whether there are cycles in the waits-for graph

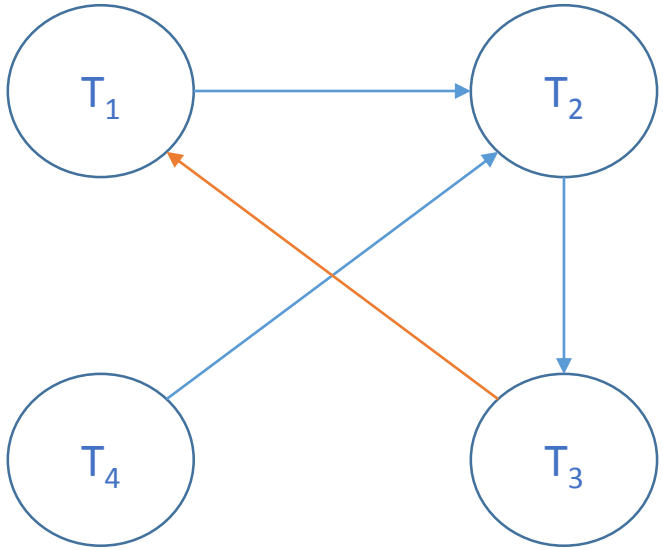
Deadlocks - Detection

a. *waits-for graph*

- example

T_1	T_2	T_3	T_4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	



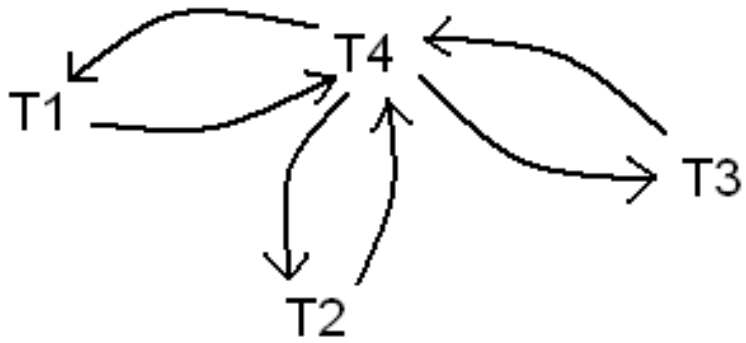


Deadlocks - Detection

a. *waits-for graph*

- example

T_1	T_2	T_3	T_4
S(A)	S(A)	S(A)	
R(A)	R(A)	R(A)	
			S(B)
			R(B)
X(B)	X(B)	X(B)	
			X(A)
…	…	…	…



Deadlocks - Detection

b. *timeout mechanism*

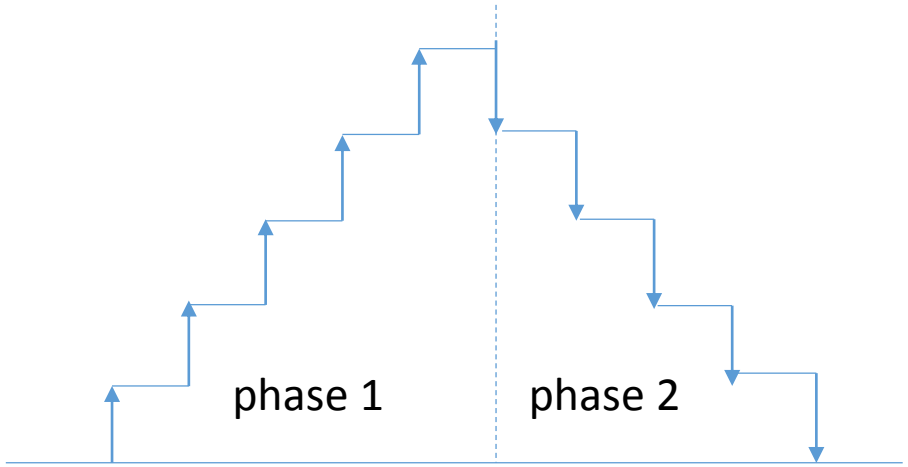
- very simple, practical method of detecting deadlocks used by several DBMSs
- if a transaction T has been waiting too long for a lock on an object, a deadlock is assumed to exist and T is terminated

Deadlocks – Choosing the Deadlock Victim

- possible criteria to consider when choosing the deadlock victim
  - the number of objects modified by the transaction
  - the number of objects that are to be modified by the transaction
  - the number of locks held
- the policy should be “fair”, i.e., if a transaction is repeatedly chosen as a victim, it should be eventually allowed to proceed

Two-Phase Locking (2PL)

- once a transaction releases a lock, it cannot request other locks
- phase 1
  - *growing phase*
  - transaction acquires locks
- phase 2
  - *shrinking phase*
  - transaction releases locks

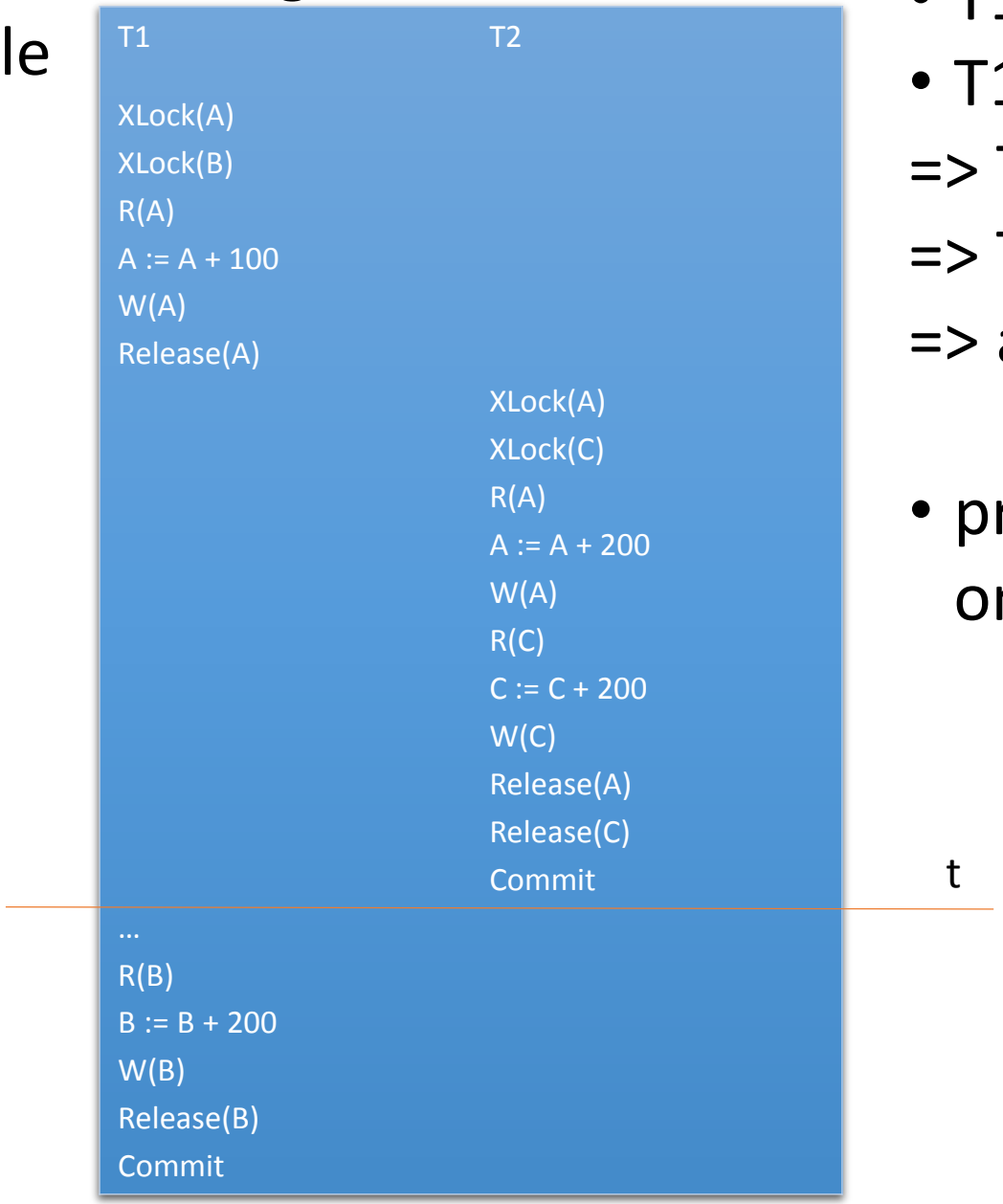


Two-Phase Locking

- *C* – set of transactions
- *Sch(C)* – set of schedules for *C*
- if all transactions in *C* obey 2PL, then any schedule $S \in Sch(C)$ that completes normally is serializable

Two-Phase Locking

- example



- T1, T2 – 2PL
- T1 forced to terminate at time t

=> T1's updates are undone

=> T2's update to A is lost

=> atomicity is compromised

- problem – T1 release its exclusive lock on A prior to completion

Strict Schedules

- transaction T_i wrote object A => transaction T_j can read / write A only after T_i's completion (commit / abort)
- strict schedules => avoiding cascading aborts => recoverable schedules

- Strict 2PL only allows strict schedules

The Phantom Problem

example 1. Researchers[RID, …, ImpactFactor, Age]

- Page1: <R1, 5, 30>, <R2, 5, 20>
- Page2: <R4, 5, 100>
- Page3: <R8, 6, 18>, <R9, 6, 19>
- concurrent transactions T1 and T2
- transaction T1
  - retrieve age of oldest researcher for impact factor values 5 and 6
- transaction T2
  - add new researcher with impact factor 5
  - remove researcher R9
- T1 and T2 obey Strict 2PL

The Phantom Problem

- Page1: <R1, 5, 30>, <R2, 5, 20>
- Page2: <R4, 5, 100>
- Page3: <R8, 6, 18>, <R9, 6, 19>

- outcome of interleaved schedule on the right:
  - IF 5, Max Age 100
  - IF 6, Max Age 18
- outcome of serial schedule (T1T2):
  - IF 5, Max Age 100
  - IF 6, Max Age 19
- outcome of serial schedule (T2T1):
  - IF 5, Max Age 102
  - IF 6, Max Age 18

T1	T2
XLock(Page1)	
XLock(Page2)	
compute max age for IF 5 => 100	
	XLock(Page4)
	XLock(Page3)
	add record <R5, 5, 102> on Page4
	delete researcher R9
	commit – all locks are released
XLock(Page3)	
compute max age for IF = 6 => 18	
…	

The Phantom Problem

=> the interleaved schedule is not serializable, as its outcome is not identical to the outcome of any serial schedule

- however, the schedule is conflict serializable (acyclic precedence graph)

=> in the presence of insert operations, i.e., if new objects can be added to the database, conflict serializability does not guarantee serializability

The Phantom Problem
example 2.

- T1 executes the same query twice
- between the 2 read operations, another transaction T2 inserts a row that meets the condition in T1's query; T2 commits

T1	T2
`SELECT * FROM Students WHERE GPA >= 7`	
	`INSERT INTO Students VALUES (23, 'Ioana', 'Gheorghiu', 10)`
	COMMIT
`SELECT * FROM Students WHERE GPA >= 7`	
…	

Isolation Levels *

- determine the degree to which a transaction is isolated from the changes made by other concurrently running transactions
- greater concurrency -> concurrency anomalies
- 4 isolation levels
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ
  - SERIALIZABLE

* check seminar 3

Isolation Levels

- READ UNCOMMITTED
  - a transaction must acquire an exclusive lock prior to writing an object
  - no locks are requested when reading objects
  - exclusive locks are released at the end of the transaction
  - lowest degree of isolation

Isolation Levels

- READ UNCOMMITTED
  - dirty reads ✓
  - unrepeatable reads ✓
  - phantoms ✓

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	C
R(B)	
W(B)	
A	

T1	T2
R(A)	
	R(A)
	W(A)
	C
R(A)	
W(A)	
C	

T1	T2
R(students with id between 100 and 110)	
	insert student with id 101
	C
R(students with id between 100 and 110)	
C	

Isolation Levels

- READ COMMITTED
  - a transaction must acquire an exclusive lock prior to writing an object
  - a transaction must acquire a shared lock prior to reading an object (i.e., the last transaction that modified the object is finished)
  - exclusive locks are released at the end of the transaction
  - shared locks are immediately released

Isolation Levels

- READ COMMITTED
  - dirty reads ✗
  - unrepeatable reads ✓
  - phantoms ✓

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	C
R(B)	
W(B)	
A	

T1	T2
R(A)	
	R(A)
	W(A)
	C
R(A)	
W(A)	
C	

T1	T2
R(students with id between 100 and 110)	
	insert student with id 101
	C
R(students with id between 100 and 110)	
C	

Isolation Levels

- REPEATABLE READ
  - a transaction must acquire an exclusive lock prior to writing an object
  - a transaction must acquire a shared lock prior to reading an object
  - exclusive locks are released at the end of the transaction
  - shared locks are released at the end of the transaction

Isolation Levels

- REPEATABLE READ
  - dirty reads ✘
  - unrepeatable reads ✘
  - phantoms ✓

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	C
R(B)	
W(B)	
A	

T1	T2
R(A)	
	R(A)
	W(A)
	C
R(A)	
W(A)	
C	

T1	T2
R(students with id between 100 and 110)	
	insert student with id 101
	C
R(students with id between 100 and 110)	
C	

Isolation Levels

- SERIALIZABLE
  - a transaction must acquire locks on objects before reading / writing them
  - a transaction can also acquire locks on sets of objects that must remain unmodified
  - locks are held until the end of the transaction
  - highest degree of isolation

Isolation Levels

- SERIALIZABLE
  - dirty reads ✘
  - unrepeatable reads ✘
  - phantoms ✘

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	C
R(B)	
W(B)	
A	

T1	T2
R(A)	
	R(A)
	W(A)
	C
R(A)	
W(A)	
C	

T1	T2
R(students with id between 100 and 110)	
	insert student with id 101
	C
R(students with id between 100 and 110)	
C	

References

- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [Le99] LEVENE, M., LOIZOU, G., A Guided Tour of Relational Databases and Beyond, Springer, 1999
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book, Prentice Hall Press, 2008
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si10] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, McGraw-Hill, 2010, http://codex.cs.yale.edu/avi/db-book/
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html