

CS194-3/CS16x Introduction to Systems

Lecture 8

Database concurrency control, Serializability, conflict serializability, 2PL and strict 2PL

September 24, 2007

Prof. Anthony D. Joseph

<http://www.cs.berkeley.edu/~adj/cs16x>

Review: ACID Summary

- **A**tomicity: All actions in the Xact happen, or none happen
- **C**onsistency: If each Xact is consistent, and the DB starts consistent, it ends up consistent
- **I**solation: Execution of one Xact is isolated from that of other Xacts
- **D**urability: If a Xact commits, its effects persist

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.2

Goals for Today

- **Concurrency Control & Recovery**
 - Very valuable properties of DBMSs (makes them useful)
 - Based on concept of transactions with ACID properties
 - » How to implement these properties?
- **Today: focus on Isolation property**
 - Serial schedules safe but slow
 - Try to find schedules *equivalent* to serial ...

There are three side effects of acid.
Enhanced long term memory, decreased short
term memory, and I forget the third.
- Timothy Leary

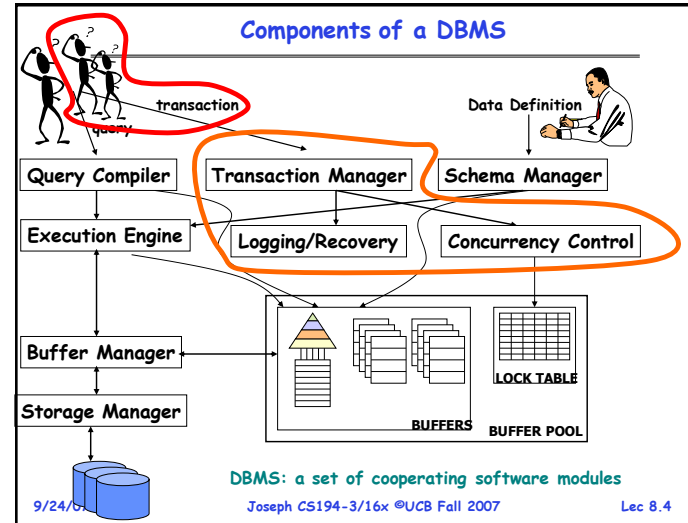
Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Slides courtesy of Kubiawicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.3

Components of a DBMS



DBMS: a set of cooperating software modules

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.4

Statement of Problem

- Concurrent execution of independent transactions
 - Utilization/throughput ("hide" waiting for I/Os.)
 - Response time and Fairness

- Example:

T1:	T2:
t0: tmp1 := read(X)	tmp2 := read(X)
t1: tmp1 := tmp1 - 20	tmp2 := tmp2 + 10
t3: write tmp1 into X	write tmp2 into X
t4: write tmp1 into X	
t5: write tmp1 into X	

- Arbitrary interleaving can lead to
 - Temporary inconsistency (ok, unavoidable)
 - "Permanent" inconsistency (bad!)
- Need formal correctness criteria

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.5

Definitions

- A program may carry out many operations on the data retrieved from the database
- However, the DBMS is only concerned about what data is read/written from/to the database
- database - a fixed set of named data objects (*A, B, C, ...*)
- transaction - a sequence of read and write operations (*read(A), write(B), ...*)
 - DBMS's abstract view of a user program
- Our correctness criteria - the ACID properties

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.6

A

Atomicity of Transactions

- Two possible outcomes of executing a transaction:
 - Xact might *commit* after completing all its actions
 - or it could *abort* (or be aborted by the DBMS) after executing some actions
- DBMS guarantees that Xacts are atomic.
 - From user's point of view: Xact always either executes all its actions, or executes no actions at all
- We'll talk about mechanisms for ensuring atomicity in the next lecture (logging and shadow pages)

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.7

C

Transaction Consistency

- "Consistency" - data in DBMS is accurate in modeling real world and follows integrity constraints
- User must ensure transaction consistent by itself
 - I.e., if DBMS consistent before Xact, it will be after also
- Key point:



9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.8

C

Transaction Consistency (cont.)

- Recall: Integrity constraints
 - Must be true for DB to be considered consistent
 - Examples:
 - FOREIGN KEY R.sid REFERENCES S
 - ACCT-BAL >= 0
- System checks ICs and if they fail, the transaction rolls back (i.e., is aborted).
 - Beyond this, DBMS does not understand the semantics of the data.
 - e.g., it does not understand how interest on a bank account is computed

9/24/07

Joseph CS194-3/16x @UCB Fall 2007

Lec 8.9

I

Isolation of Transactions

- Users submit transactions, and
- Each transaction executes **as if** it was running **by itself**
 - Concurrency is achieved by DBMS, which interleaves actions (reads/writes of DB objects) of various transactions
- Many techniques have been developed. Fall into two basic categories:
 - Pessimistic - don't let problems arise in the first place
 - Optimistic - assume conflicts are rare, deal with them *after* they happen

9/24/07

Joseph CS194-3/16x @UCB Fall 2007

Lec 8.10

I

Example

- Consider two transactions (*Xacts*):

T1:	BEGIN	A=A+100,	B=B-100	END
T2:	BEGIN	A=1.06*A,	B=1.06*B	END

- 1st xact transfers \$100 from B's account to A's
- 2nd credits both accounts with 6% interest
- Assume at first A and B each have \$1000. What are the **legal outcomes** of running T1 and T2???
- \$2000 * 1.06 = \$2120
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together.
But, the net effect must be equivalent to these two transactions running serially in some order

9/24/07

Joseph CS194-3/16x @UCB Fall 2007

Lec 8.11

I

Example (Contd.)

- Legal outcomes: A=1166, B=954 or A=1160, B=960
- Consider a possible interleaved **schedule**:

T1:	A=A+100,	B=B-100
T2:	A=1.06*A,	B=1.06*B

- This is OK (same as T1;T2). But what about:

T1:	A=A+100,	B=B-100
T2:	A=1.06*A,	B=1.06*B

- Result: A=1166, B=960; A+B = 2126, bank loses \$6
- The DBMS's view of the second schedule:

T1:	R(A), W(A),	R(B), W(B)
T2:	R(A), W(A), R(B), W(B)	

9/24/07

Joseph CS194-3/16x @UCB Fall 2007

Lec 8.12

I

Formal Properties of Schedules

- **Serial schedule:** Schedule that does not interleave the actions of different transactions.
- **Equivalent schedules:** For any database state, the effect 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.)

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.13

Administrivia

- Class communication
 - Setting up newsgroup
 - Will be accessible via authnews.berkeley.edu
 - Also, google groups (read-only) is good for searching
 - Use google groups to search cs162 newsgroup
- First design document due 9/26
 - Has to be in by 11:59pm
 - Contact Kai if you have questions or need help
- Schedule design reviews with Kai

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.14

I

Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, "dirty reads"):

T1:	R(A), W(A),	R(B), W(B), Abort
T2:	R(A), W(A), C	

- Unrepeatable Reads (RW Conflicts):

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

- Overwriting Uncommitted Data (WW Conflicts):

T1:	W(A),	W(B), C
T2:	W(A), W(B), C	

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.15

I

Lock-Based Concurrency Control

- Here's a simple way to allow concurrency but avoid the anomalies just described...

- **Strict Two-phase Locking (Strict 2PL) Protocol:**

- Each Xact must obtain an **S (shared)** lock on object before reading, and an **X (exclusive)** lock on object before writing
- System can obtain these locks *automatically*
- Lock rules:
 - » If an Xact holds an X lock on an object, no other Xact can acquire a lock (S or X) on that object
 - » If an Xact holds an S lock, no other Xact can get an X lock on that object
- Two phases: acquiring locks, and releasing them
 - » No lock is ever acquired after one has been released
 - » All locks held by a transaction are released when the xact completes

- Strict 2PL allows only serializable schedules

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.16

Aborting a Transaction (i.e., Rollback)

- If an xact T_i aborted, all actions must be undone
- Also, if T_j reads object last written by T_i , T_j must be aborted!
 - Most systems avoid such *cascading aborts* by releasing locks only at EOT (i.e., strict locking).
 - If T_i writes an object, T_j can read this only after T_i finishes
- To *undo* actions of an aborted transaction, DBMS maintains *log* which records every write
- Recovery protocol
 - Log also used to recover from system crashes:
All active Xacts at time of crash are aborted when system comes back up - more on logging in next lecture

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.17

Summary

- **Concurrency control** and **recovery** are among the most important functions provided by a DBMS
- Concurrency control is automatic
 - System automatically inserts lock/unlock requests and schedules actions of different Xacts
 - Property ensured: resulting execution is equivalent to executing the Xacts one after the other in some order
- Recovery protocol used to:
 1. undo the actions of aborted transactions, and
 2. restore the system to a consistent state after a crash

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.18

Conflicting Operations

- Need a tool to decide if 2 schedules are equivalent
- Use notion of "conflicting" operations
- Definition: Two operations **conflict** if:
 - They are by different transactions,
 - they are on the same object,
 - and at least one of them is a write

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.19

Conflict Serializable Schedules

- Definition: Two schedules are **conflict equivalent** iff:
 - They involve the same actions of the same transactions, and
 - every pair of conflicting actions is ordered the same way
- Definition: Schedule S is **conflict serializable** if:
 - S is conflict equivalent to some serial schedule
- Note, some "serializable" schedules are NOT conflict serializable
 - A price we pay to achieve efficient enforcement

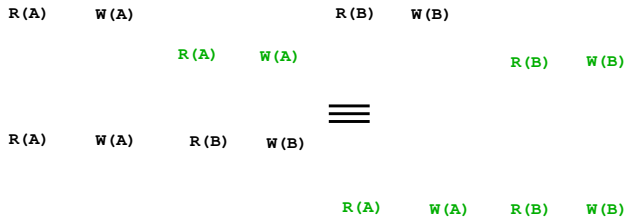
9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.20

Conflict Serializability - Intuition

- A schedule S is conflict serializable if:
 - You are able to transform S into a serial schedule by swapping consecutive non-conflicting operations of different transactions.
- Example:**



9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.21

Conflict Serializability (Continued)

- Here's another example:



- Serializable or not????

NOT!

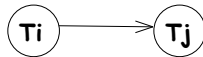
9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.22

Dependency Graph

- Dependency graph:**
 - One node per Xact
 - Edge from T_i to T_j if:
 - An operation O_i of T_i conflicts with an operation O_j of T_j and
 - O_i appears earlier in the schedule than O_j .
- Theorem:** Schedule is conflict serializable if and only if its dependency graph is acyclic.



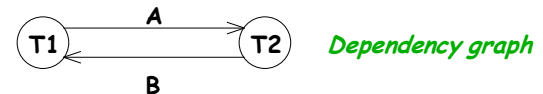
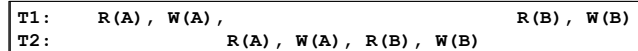
9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.23

Example

- A schedule that is not conflict serializable:



- The cycle in the graph reveals the problem. The output of $T1$ depends on $T2$, and vice-versa

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.24

An Aside: View Serializability

- Alternative (weaker) notion of serializability.
- Schedules S1 and S2 are **view equivalent** if:
 1. If T_i reads initial value of A in S1, then T_i also reads initial value of A in S2
 2. If T_i reads value of A written by T_j in S1, then T_i also reads value of A written by T_j in S2
 3. If T_i writes final value of A in S1, then T_i also writes final value of A in S2
- Basically, allows all conflict serializable schedules + "blind writes"

T1: R(A)	W(A)		
T2: W(A)			
T3: W(A)			

view

T1: R(A), W(A)			
T2: W(A)			
T3: W(A)			

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.25

Notes on Serializability Definitions

- View Serializability allows (slightly) more schedules than Conflict Serializability does
 - Problem is that it is difficult to enforce efficiently
- Neither definition allows all schedules that you would consider "serializable"
 - This is because they don't understand the meanings of the operations or the data
- In practice, Conflict Serializability is what gets used, because it can be enforced efficiently
 - To allow more concurrency, some special cases do get handled separately, such as for travel reservations, etc.

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.26

BREAK

Two-Phase Locking (2PL)

Lock
Compatibility
Matrix

	S	X
S	✓	-
X	-	-

Rules:

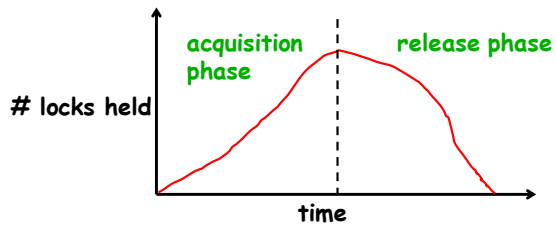
- Xact must obtain a S (*shared*) lock before reading, and an X (*exclusive*) lock before writing
- Xact cannot get new locks after releasing any locks

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.28

Two-Phase Locking (2PL), cont.



2PL guarantees conflict serializability



But, does not prevent Cascading Aborts



9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

L.29

Strict 2PL

- **Problem: Cascading Aborts**
- **Example: rollback of T1 requires rollback of T2!**

T1:	R(A), W(A),	R(B), W(B), Abort
T2:	R(A), W(A)	

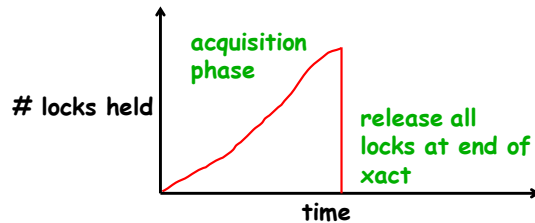
- **Strict Two-phase Locking (Strict 2PL) protocol:**
Same as 2PL, except:
Locks released only when transaction completes
i.e., either:
 - (a) transaction has committed (commit record on disk),
 - or
 - (b) transaction has aborted and rollback is complete

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.30

Strict 2PL (continued)



- Next ...
- A few examples

9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.31

Non-2PL, A= 1000, B=2000, Output =?

Lock_X(A)	
Read(A)	Lock_S(A)
A := A-50	
Write(A)	
Unlock(A)	
	Read(A)
	Unlock(A)
	Lock_S(B)
Lock_X(B)	
	Read(B)
	Unlock(B)
	PRINT(A+B)
Read(B)	
B := B +50	
Write(B)	
Unlock(B)	

9/24/07

2PL, A= 1000, B=2000, Output =?

Lock_X (A)	
Read (A)	Lock_S (A)
A: = A-50	
Write (A)	
Lock_X (B)	
Unlock (A)	
	Read (A)
	Lock_S (B)
Read (B)	
B := B +50	
Write (B)	
Unlock (B)	Unlock (A)
	Read (B)
	Unlock (B)
	PRINT (A+B)

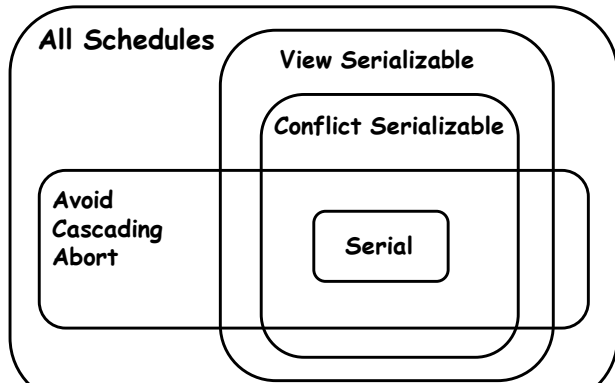
9/24/07

Strict 2PL, A= 1000, B=2000, Output =?

Lock_X (A)	
Read (A)	Lock_S (A)
A: = A-50	
Write (A)	
Lock_X (B)	
Read (B)	
B := B +50	
Write (B)	
Unlock (A)	
Unlock (B)	
	Read (A)
	Lock_S (B)
	Read (B)
	PRINT (A+B)
	Unlock (A)
	Unlock (B)

9/24/07

Venn Diagram for Schedules



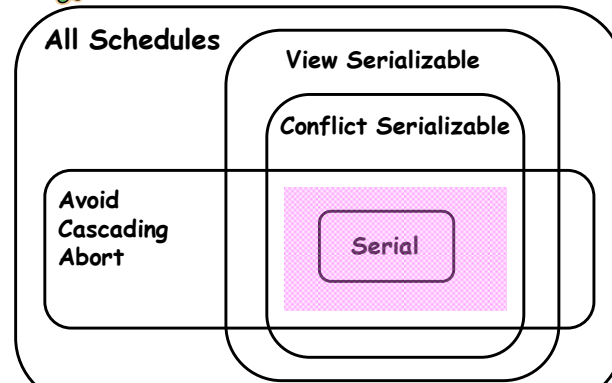
9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.35



Which schedules does Strict 2PL allow?



9/24/07

Joseph CS194-3/16x ©UCB Fall 2007

Lec 8.36

Summary

- **Correctness criterion for isolation is "serializability"**
 - In practice, we use "conflict serializability," which is somewhat more restrictive but easy to enforce
- **Two Phase Locking and Strict 2PL: Locks implement the notions of conflict directly.**
 - The lock manager keeps track of the locks issued
 - Deadlocks may arise; can either be prevented or detected (more in the next lecture)