CSE 344: Section 9 Transactions

Aug 10th, 2018

Administrivia

- **HW #8 (due Aug 15th):** Use Java + SQL Server to make application for booking flights
  - final homework of the quarter (yay!)
  - most time-consuming homework, so start early

- **Quiz #7 (due Aug 13th):** transactions & scheduling

Definition of SCHEDULE

The sequence of the read/write operations of several transactions as they are executed in the Database

Serial Schedule

- Transactions execute fully.
- One at a time.
- No interleaving.
- Different orders of execution may produce different final values

Serial

t1	t2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Serializable Schedules

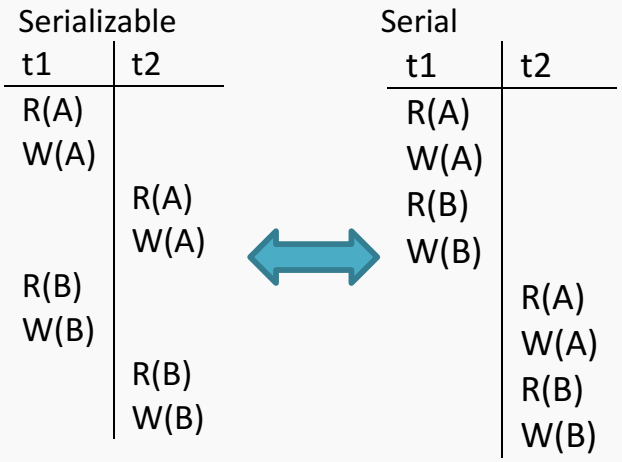
- Interleaved.
- Equivalent to SOME serial schedule.
- Equivalence does NOT mean "ending up with the same values as".
- Equivalence cannot depend on initial values of database items.
- Cannot depend on values written DB doesn’t know logic of transaction.
- Depends only on order of operations.

Serializable schedule

t1	t2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

Serializable Schedules

- Serial Schedule
  - r1(A),w1(A),r1(B),w1(B),r2(A),w2(A),r2(B),w2(B)
- Serializable Schedule
  - r1(A),w1(A),r2(A),w2(A),r1(B),w1(B),r2(B),w2(B)



Conflicting Operations

- Used to define how schedules are equivalent

- 2 OPERATIONS CONFLICT if
  - belong to different transactions
  - access same data item
  - at least one is a write

Conflicts: (i.e., swapping will change program behavior)

Two actions by same transaction T_i:	$r_i(X); w_i(Y)$
Two writes by T_i, T_j to same element	$w_i(X); w_j(X)$
Read/write by T_i, T_j to same element	$w_i(X); r_j(X)$
	$r_i(X); w_j(X)$

- IMPORTANT: they do NOT have to ACTUALLY come into CONFLICT!
  - A better name would be 'Potentially Conflicting Operations'

CONFLICT EQUIVALENCE

- 2 Schedules are Conflict Equivalent
  If the order of any 2 conflicting operations is the same in both schedules.

- SERIALIZABLE SCHEDULE is CONFLICT EQUIVALENT to some serial schedule

CONFLICT EQUIVALENCE

S1: $R_1(A)$, $W_1(A)$, $R_2(A)$, $W_2(A)$, $R_1(B)$, $W_1(B)$, $R_2(B)$, $W_2(B)$

If O_i and O_j are two operations in a transaction and $O_i < O_j$ (O_i is executed before O_j), same order will follow in schedule as well. Using this property, we can get two transactions of schedule S1 as:

T1: $R_1(A)$, $W_1(A)$, $R_1(B)$, $W_1(B)$
T2: $R_2(A)$, $W_2(A)$, $R_2(B)$, $W_2(B)$

Possible Serial Schedules are: T1->T2 or T2->T1

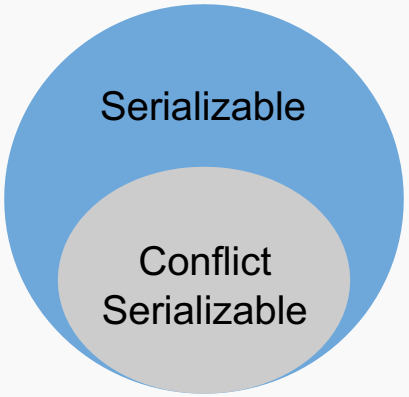
-> **Swapping non-conflicting operations** $R_2(A)$ and $R_1(B)$ in S1, the schedule becomes,

S11: $R_1(A)$, $W_1(A)$, $R_1(B)$, **$W_2(A)$,** $R_2(A)$, **$W_1(B)$,** $R_2(B)$, $W_2(B)$

Serializability

Conflict serializable is stricter than serializable

I.e. Any schedule that is conflict serializable must be serializable.



Serializability

Conflict serializable is stricter than serializable

I.e. Any schedule that is conflict serializable must be serializable.

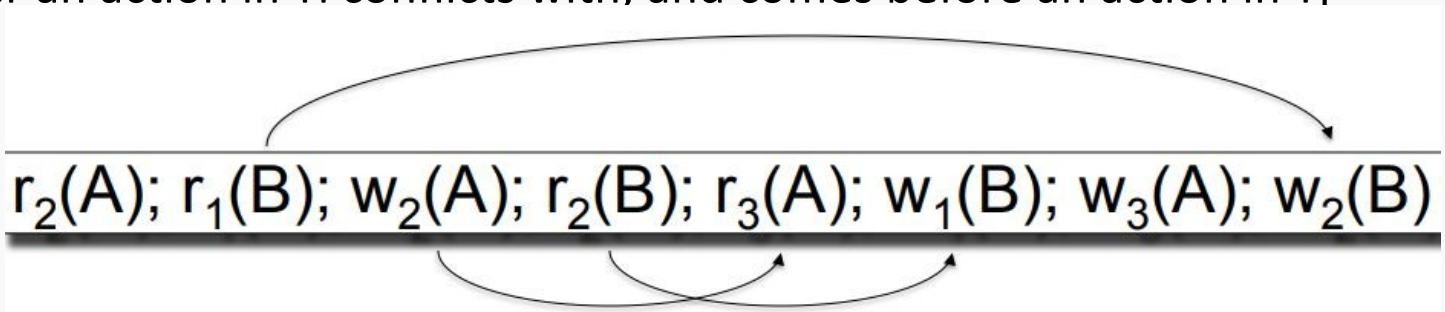
Not all serializable schedules are conflict serializable:

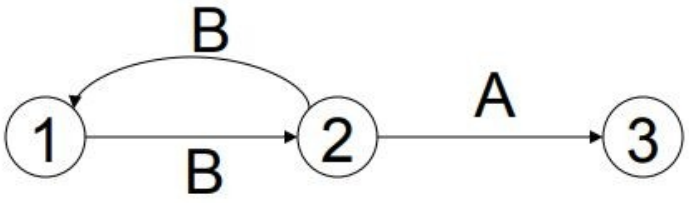
t1	**t2**
W(A, 0)	
	W(A, 0)
R(A)	
	R(B)

Serializability

Checking for conflict serializability -> precedence graph and cycle checking

Precedence graph: • A node for each transaction Ti , • An edge from Ti to Tj whenever an action in Ti conflicts with, and comes before an action in Tj





Serializability

S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

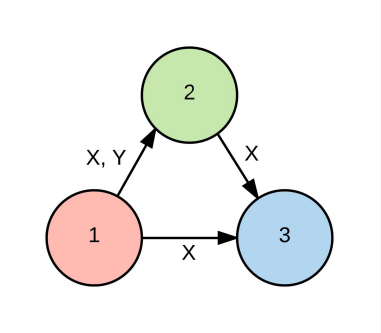
Are these serializable?
Conflict serializable?

S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Serializability

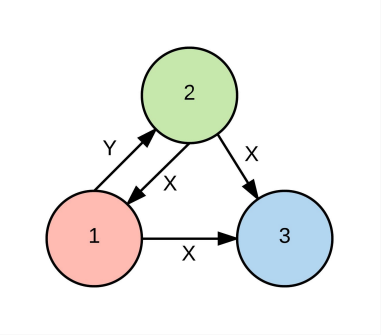
S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

Conflict Serializable



S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Serializable (but not conflict serializable)



2PL v.s. Strict 2PL

2PL:

- In every transaction, all lock requests must precede all unlock requests
- Ensure Conflict Serializability
- Might not be able to recover (Dirty Read: Read on some write that gets rolled back)

Strict 2PL:

- Every lock each transaction is held until commit or abort
- Ensure Conflict Serializability
- Recoverable as each transaction does not affect others until commit/abort

2PL v.s. Strict 2PL

A New Problem: Non-recoverable Schedule

T1	T2
$L_1(A)$; $L_1(B)$; READ(A)	
A :=A+100	
WRITE(A); $U_1(A)$	
	$L_2(A)$; READ(A)
	A := A*2
	WRITE(A);
	$L_2(B)$; BLOCKED…
READ(B)	
B :=B+100	
WRITE(B); $U_1(B)$;	
	…GRANTED; READ(B)
	B := B*2
	WRITE(B); $U_2(A)$; $U_2(B)$;
	Commit
Rollback	

Isolation Level: Read Uncommitted

Write Locks? Strict 2PL

Read Locks? No (Immediate Read)

Problem: Dirty-Read

Reading uncommitted data that can be rolled back

Isolation Level: Read Uncommitted

Example Transaction:

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

T2 is reading value of A updated by T1's write on A, but T1 has not committed yet.

The value of A read by T2 might not even be in the result.

Then T2's action can be influenced by such uncommitted data.

Isolation Level: Read Committed

Write Locks? Strict 2PL

Read Locks? Obtain before read, release after (No more dirty read)

If transaction wants to read, it needs to wait until the lock on the value is released (when the other transaction commits or aborts)

Problem: Unrepeatable Read

> The values of 2 reads on the same tuple can be different in the same transaction

Isolation Level: Read Committed

Example Transaction:

$$S = \begin{bmatrix} T1 & T2 \\ R(A) & \\ & R(A) \\ & W(A) \\ & Com. \\ R(A) & \\ W(A) & \\ Com. & \end{bmatrix}$$

T1's first R(A) and T1's second R(A) might have different results.

Updated by T2's W(A).

Isolation Level: Repeatable Read

Write Locks? Strict 2PL

Read Locks? Strict 2PL (No more unrepeatable read)

Same as Serializable if no insert or delete

Problem: Phantom Read

In the same transaction, some tuples appear sometimes and disappear other times

Isolation Level: Repeatable Read

Suppose there are two blue products, A1, A2:

Phantom Problem

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('A3','blue')
SELECT * FROM Product WHERE color='blue'	

Isolation Level: Serializable

Not the same thing as Serializable schedule!!!

Write Locks: Strict 2PL

Read Locks: Strict 2PL

Predicate Lock/Table Lock (No Phantom)

Difference between Repeatable Read and Serializable is that serializable schedule blocks inserts & deletes from another transaction

Isolation Level: Serializable

Predicate Lock Example:

In Transaction T, we have a statement:

SELECT * FROM People WHERE age > 18;

In this case, the transaction will grab a predicate lock that prevent inserting and deleting tuples that can affect the predicate/statement.

In this case, the lock prevents inserting and deleting tuples with age > 18.

Isolation Level: Summary

Isolation Level	Read Locks	Write Locks	Dirty Reads	Nonrepeatable Reads	Phantom Inserts
Read Uncommitted	None	Strict 2PL	Allowed	Allowed	Allowed
Read Committed	Temporary lock	Strict 2PL	Not Allowed	Allowed	Allowed
Repeatable Read	Strict 2PL	Strict 2PL	Not Allowed	Not Allowed	Allowed
Serializable	Strict 2PL	Strict 2PL + Insert Lock	Not Allowed	Not Allowed	Not Allowed