CSCI3170 Introduction to Database Systems

Tutorial 9
Concurrency Control and Recovery

Outline

- **Concurrency Control**
  - Transaction and Schedule
  - Conflict Serializability and Precedence Graph
  - View Serializability
  - Shared Lock and Exclusive Lock
- **Recovery**
  - Log-Based Recovery
  - Immediate Database Modification
  - Deferred Database Modification

Serializability and Strict Two-Phase Locking Protocol

CONCURRENCY CONTROL

Transaction

- A sequence of read/write operations
- Examples

T_1: read(a) a = a + 100 write(a)	T_2: a = 100 write(a) read(a)
$T_1 = r_1[a]\ w_1[a]$	$T_2 = w_2[a]\ r_2[a]$

Schedule

- A sequence of read/write operations from a set of transactions
- Examples

  Suppose there are two transactions, T_1 and T_2

  $T_1 = r_1[a]\ w_1[a]$ $\qquad$ $T_2 = w_2[a]\ r_2[a]$

T1:	$r_1[a]$		$w_1[a]$	
T2:		$w_2[a]$		$r_2[a]$

T1:	$r_1[a]$	$w_1[a]$		
T2:			$w_2[a]$	$r_2[a]$

Possible schedules

Serial Schedule

- The operations belonging to one single transaction appear together
- Examples

T1:	$r_1[a]$	$w_1[a]$		
T2:			$w_2[a]$	$r_2[a]$

Denoted by T_1T_2

✓

T1:	$r_1[a]$		$w_1[a]$	
T2:		$w_2[a]$		$r_2[a]$

✗

Serializable schedule

- The effect is equivalent to some serial schedules
- Examples

T1:	$r_1[a]$		$w_1[a]$	
T2:		$w_2[b]$		$r_2[a]$

Equivalent to T_1T_2

T1:	$r_1[a]$	$w_1[a]$		
T2:			$w_2[b]$	$r_2[a]$

Conflict Operations

- Two operations are **conflict** if
  - They are operations from different transactions on the same data object
  - At least one of them is a **Write** operation

T1: $r_1[a]$	T1: $w_1[a]$	T1: $w_1[a]$
T2: $w_2[a]$	T2: $r_2[a]$	T2: $w_2[a]$

Conflict Equivalent

- Two schedules S_1 and S_2 are **conflict equivalent** if
  - S_1 and S_2 involve the same operations of the same set of transactions
  - **Every pair** of conflicting operations is ordered in the same way in S_1 and S_2

S_1

T1:	$w_1[a]$		$r_1[b]$	
T2:		$r_2[b]$		$w_2[a]$

S_2

T1:	$w_1[a]$	$r_1[b]$		
T2:			$r_2[b]$	$w_2[a]$

Conflict equivalent?

Operations:
$w_1[a]$, $r_1[b]$, $r_2[b]$, $w_2[a]$

Operations on a:
$w_1[a]w_2[a]$ **(the same order)**

Operations on b:
$r_2[a]r_1[a]$ and $r_1[a]r_2[a]$
(not conflict operations)

Conflict Serializability

- S_1 is **conflict serializable** if it is conflict equivalent to a serial schedule S_2

S_1

T1:	$w_1[a]$		$r_1[b]$	
T2:		$r_2[b]$		$w_2[a]$

Conflict equivalent
(From previous slide)

S_2

T1:	$w_1[a]$	$r_1[b]$		
T2:			$r_2[b]$	$w_2[a]$

S_2 is a serial schedule
(Can be denoted by T_1T_2)

Is S_1 Conflict serializable? ✓

Precedence Graph

- Test for conflict serializability
- A directed graph **G**=(**V**,**E**), where
  - **V** includes all transactions involved in the schedule
  - **E** consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:
    - T_i executes **write(X)** before T_j executes **read(X)**
    - T_i executes **read(X)** before T_j executes **write(X)**
    - T_i executes **write(X)** before T_j executes **write(X)**

Conflict pairs

R - W

W - R

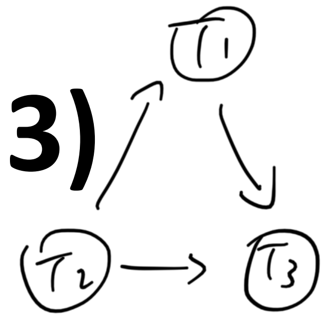
W - W

Consider the conflict operations

Precedence Graph

- A schedule **S_1** is **conflict serializable** iff **$G(S_1)$** is **acyclic** (i.e. no cycle)
- If the precedence graph is acyclic, the serialization order (execution order) can be obtained through **topological sorting**

Precedence Graph (3)



- Example

T1:	~~$w_1[a]$~~			~~$w_1[b]$~~		
T2:		~~$r_2[b]$~~			~~$w_2[c]$~~	
T3:			~~$w_3[a]$~~			$r_3[c]$

Operations on a:
$w_1[a]\ w_3[a]$

Operations on b:
$r_2[b]\ w_1[b]$

Operations on c:
$w_2[c]\ r_3[c]$

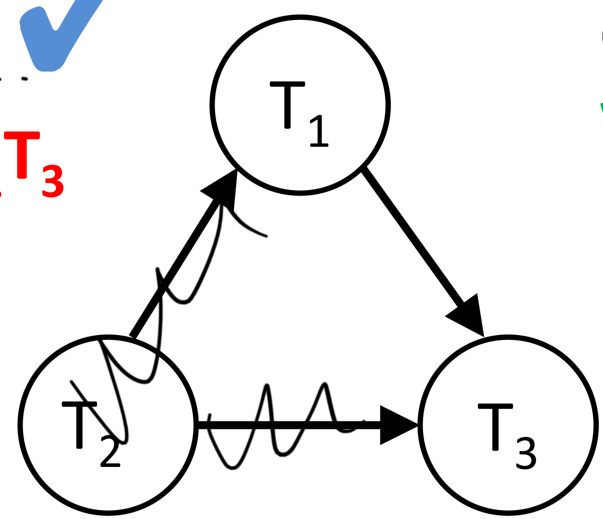
Conflict serializable? ✓

Yes, it is acyclic.

equivalent to $T_2T_1T_3$

$T_2 \to T_1 \to T_3$

(check no edge coming)



View Equivalent

- Two schedules S_1 and S_2 are **view equivalent**
  - If T_i reads the initial value of a data item in S_1, T_i also reads the initial value of the item in S_2.
  - If T_i reads an item produced by T_j in S_1, T_i also reads the item produced by T_j in S_2.
  - If T_i writes the final value of a data item in S_1, T_i also writes the final value of the item in S_2.

T1:		$w_1[a]$	
T2:	$r_2[b]$		$r_2[a]$

T1:	$w_1[a]$		
T2:		$r_2[a]$	$r_2[b]$

In both schedules,

- T2 reads b from initial value
- T2 reads a from T1
- The final write of a is T1

View Serializability

- S_1 is **view serializable** if it is view equivalent to a serial schedule

T_1	T_1
$r(a)$	
	$r(b)$
	$r(a)$

T1:		$w_1[a]$	
T2:	$r_2[b]$		$r_2[a]$

T2 reads b from initial value
T2 reads a from T1
The final write of a is T1

view serializable? ✓

→ equivalent to T_1T_2

T1:		$w_1[a]$		$r_1[b]$
T2:	$w_2[b]$		$r_2[a]$	

T1 reads b from T2
T2 reads a from T1
The final write of a is T1
The final write of b is T2

view serializable? ✗

Example (Final Exam 09 Fall Q7)

$w_3[a]\ r_2[a]\ w_1[a]\ w_2[a]$

T_1	T_2	T_3
w(a)		
		w(a)
	r(a)	
	w(a)	

T_2 read a from T_3

(a) Please state all the ***read-from*** relations.

T_2 reads x from T_3

(b) Please state which transaction executes the ***final write*** operation on *a*.

T_2 issues the final write on x

(c) Is the history **view serializable**? Why?

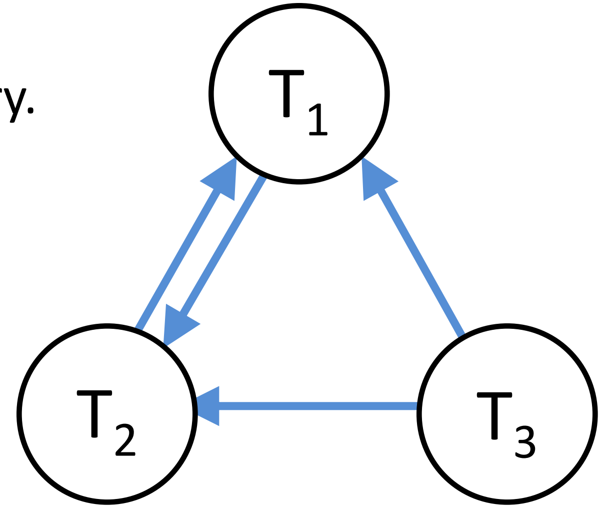
Please state the **serialization order** if it is view serializable.

Yes. The history has the same read from relation and same final write operation as T1 T3 T2

Example (Final Exam 09 Fall Q7)

$w_3[a]\ r_2[a]\ w_1[a]\ w_2[a]$

(d) Draw the serialization graph of the above history.



(e) Is the history conflict serializable? Why?
Please state the serialization order if it is conflict serializable.

No, because there is a cycle in the serialization graph.

Strict Two-Phase Locking Protocol

- If a transaction wants to **read/modify** an object, it first requests a **shared/exclusive** lock on the object.
- All locks held by a transaction are released when the transaction is completed.
- **Pro:** Allow only conflict serializable schedules.
- **Con:** Deadlock may occur.

Shared Lock and Exclusive Lock

- Shared Lock - **lock-S**
  - Used when the transaction only reads the data object
  - **Allow read** (for other transactions)
  - **Not allow write** (for other transactions)
- Exclusive Lock - **lock-X**
  - Used when the transaction has a write operation on the data object
  - **Not allow read** (for other transactions)
  - **Not allow write** (for other transactions)

T1: $r_1[a]$ $r_1[b]$ $w_1[b]$

T1: lock-S(a) $r_1[a]$ lock-X(b) $r_1[b]$ $w_1[b]$ unlock(a) unlock(b)

Deadlock

T1	T2
lock-S(a)	
	lock-S(b)
	$r_2[b]$
$r_1[a]$	
lock-X(b)	
	lock-X(a)
	$r_2[a]$
	$w_2[a]$
$r_1[b]$	
$w_1[b]$	
unlock(a)	
unlock(b)	
	unlock(b)
	unlock(a)

T_1 waits for the lock of b (i.e. T_1 waits for T_2 to release the lock of b)

T_2 waits for the lock of a (i.e. T_2 waits for T_1 to release the lock of a)

Log-Based Recovery for Immediate and Deferred Database Modifications

RECOVERY

Log-Based Recovery

- Recording database modifications in the **log**
- Log record
  - <**transaction_name**, **start**>
  - <**transaction_name**, **item_name**, **new_value**>
  - <**transaction_name**, **commit**>

T0:
Read(A,a1)
a1 := a1 – 50
Write(A,a1)
Read(B,b1)
b1 := b1 + 50
Write(B,b1)

Log:
<T0,starts>
<T0,A,950>
<T0,B,2050>
<T0,commits>

Guarantees

Values have to be **updated** if the transaction is **committed**

Values have to be **restored** if the transaction is **uncommitted**

Immediate Database Modification

- Allow database modifications to be output to the database while the transaction is active

Log record	Database process
<T0,starts>	A = 300, B = 1000
<T0,A,950>	**A = 950**, B = 1000
<T0,B,2050>	A = 950, **B = 2050**
<T0,commits>	A = 950, B = 2050

It is possible that the updated values have not been flushed to the hard disk yet.

Recovery

Redo the transaction if the transaction is **committed**

Undo the transaction if the transaction is **uncommitted**

Deferred Database Modification

- Recording all database modifications in the log, but deferring the execution of all write operations until the transaction commits

Log record	Database process
<T0,starts>	A = 300, B = 1000
<T0,A,950>	A = 300, B = 1000
<T0,B,2050>	A = 300, B = 1000
<T0,commits>	**A = 950, B = 2050**

It is possible that the updated values have not been flushed to the hard disk yet.

Recovery

Redo the transaction if the transaction is **committed**