Database Management Systems (DBMS)

Lec 25: Transaction Processing, Concurrency Control, and Recovery

Ramesh K. Jallu
IIIT Raichur

Date: 11/06/21

Recap

- Desirable properties
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- States in transaction
- **Schedules**: the order of execution of operations from all the various transactions
  - **Serial schedule**: All serial schedules leave the system in consistence state
  - **Concurrent execution**: Not all leave the system in consistence state
- **Conflict serializability**: Conflict equivalent to a serial schedule

Example

(a)

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Time

Schedule A

(b)

T_1	T_2
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Time

Schedule B

(c)

T_1	T_2
read_item(X);	
$X := X - N$;	
	read_item(X);
	$X := X + M$;
write_item(X);	
read_item(Y);	
	write_item(X);
$Y := Y + N$;	
write_item(Y);	

Time

Schedule C

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Time

Schedule D

Determining a conflict serializability

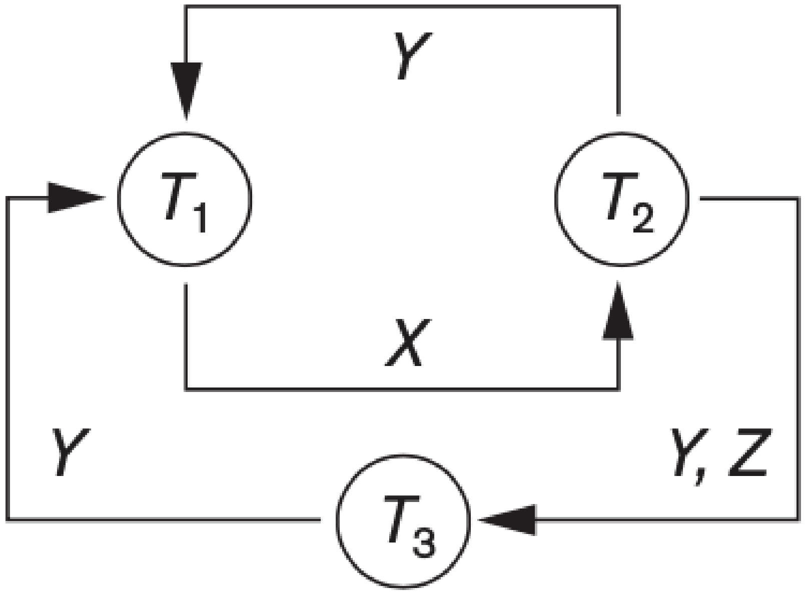
- Let S be a schedule for transactions $T_1, T_2, \ldots, T_n$
- A ***precedence graph*** (or ***serialization graph***) for S is a directed graph $G = (V, E)$, where $V = \{T_1, T_2, \ldots, T_n\}$, and there is an edge between two vertices T_i and T_j if one of the following conditions holds

1. T_i executes ***write(X)*** before T_j executes ***read(X)***
2. T_i executes ***read(X)*** before T_j executes ***write(X)***
3. T_i executes ***write(X)*** before T_j executes ***write(X)***

Example-1

Time

Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

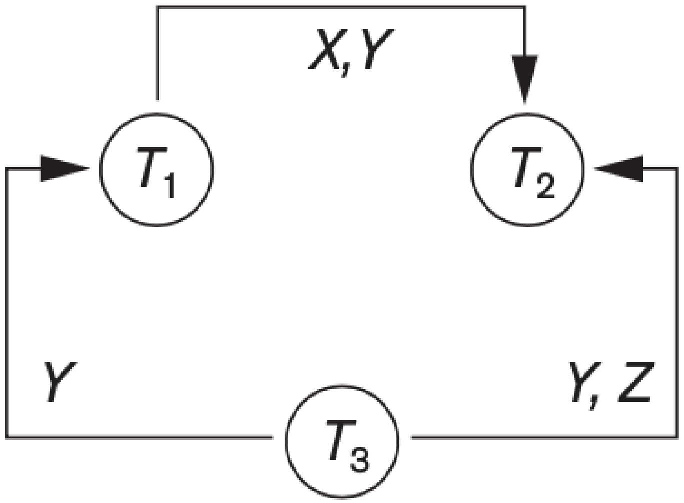


Example-2

Time

Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Y); read_item(Z);
read_item(X); write_item(X);		
		write_item(Y); write_item(Z);
	read_item(Z);	
read_item(Y); write_item(Y);		

	read_item(Y);
	write_item(Y);
	read_item(X);
	write_item(X);	

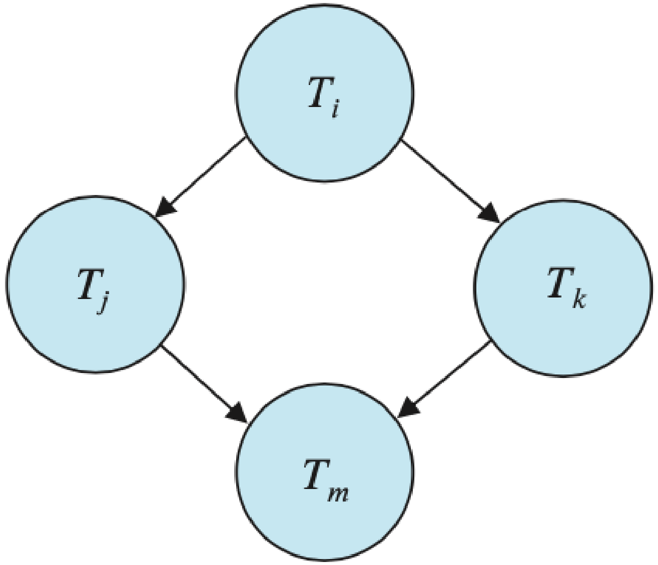


Observations

- If the precedence graph for *S* has a cycle, then schedule *S* is not conflict serializable; otherwise, S is conflict serializable

- If the graph is a DAG, a ***serializable order*** can be found by topological sorting

- In general, several possible linear orders that can be obtained through a topological sort



$<T_i, T_j, T_k, T_m>$
$<T_i, T_k, T_j, T_m>$

Why is concurrency control is needed?

- When several transactions execute concurrently in the database, the consistency of data may no longer be preserved

- Several problems can occur when concurrent transactions execute in an uncontrolled manner

- It is necessary for the system to control the interaction among the concurrent transactions, and this control is achieved through one of a variety of mechanisms called *concurrency control* schemes.

Example

- Consider an airline reservations database in which a record is stored for each airline flight

- Let T_1 *transfers* N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y

- Let T_2 that just *reserves* M seats on the first flight (X) referenced in transaction T_1

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

T_2

read_item(X);
$X := X + M$;
write_item(X);

1. The lost update problem

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

T_2

read_item(X);
$X := X + M$;
write_item(X);

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

2. The temporary update (or dirty read) problem

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

T_2

read_item(X);
$X := X + M$;
write_item(X);

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);	

3. The incorrect summary problem

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

T_2

read_item(X);
$X := X + M$;
write_item(X);

T_1	T_3
	$sum := 0$; read_item(A); $sum := sum + A$; $\vdots$

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$sum := sum + X$;
	read_item(Y);
	$sum := sum + Y$;

read_item(Y);
$Y := Y + N$;
write_item(Y);	

4. Unpredectble read problem

- If transaction *T* reads the same data item twice and the data item is changed by another transaction *T'* between the two reads

- Hence, *T* receives *different values* for its two reads of the same item

- For example, if during an airline reservation transaction, a customer inquires about seat availability on several flights, and upon deciding a particular flight it may end up reading a different value for the item

- The ***phantom read problem*** occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist

Types of failures

1. **A computer failure**: System crash due to h/w or s/w failure

2. **A transaction or system error**: Logical error

3. **Local errors**: Data not found or insufficient account balance

4. **Concurrency control enforcement**: Deadlock or violates serializability

5. **Disk failure**: Disk read/write head crash

6. **Physical problems and catastrophes**: Power or air-conditioning failure, fire, theft, sabotage, etc.

Characterizing Schedules Based on Recoverability

- For some schedules it is easy to recover from transaction and system failures, whereas for other schedules the recovery process can be quite involved

- In some cases, it is even not possible to recover correctly after a failure

- We theoretically characterize the different types of schedules for which ***recovery is possible***, as well as those for which ***recovery is relatively simple***

Characterizing Schedules Based on Recoverability (Contd.)

- Once a transaction is committed, it should *never* be rolled back T; Otherwise, it violates durability property
- The schedules that theoretically meet this criterion are called ***recoverable schedules***
- A schedule where a committed transaction may have to be rolled back during recovery is called ***nonrecoverable***

Recoverable schedules

- A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written some item X that T reads have committed

- A transaction T **reads** from transaction T' in a schedule S if

1. some item X is first written by T' and later read by T,

2. T' should not have been aborted before T reads item X, and

3. there should be no transactions that write X after T' writes it and before T reads it (unless those transactions, if any, have aborted before T reads X)

Thank you!