Course Name:
Database Management Systems

Lecture 22
Topics to be covered

- Serializability
  - *Introduction*
  - *Conflict Serializability*
  - *View Serializability*
  - *Testing for Serializability*
  - *Applications*
  - *Scope of Research*

Introduction

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. **conflict serializability**
  2. **view serializability**
- *Simplified view of transactions*
  - We ignore operations other than **read** and **write** instructions
  - We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
  - Our simplified schedules consist of only **read** and **write** instructions.

Conflicting Instructions

- Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both l_i and l_j, and at least one of these instructions wrote Q.
    1. l_i = **read**(Q), l_j = **read**(Q). l_i and l_j don't conflict.
    2. l_i = **read**(Q), l_j = **write**(Q). They conflict.
    3. l_i = **write**(Q), l_j = **read**(Q). They conflict
    4. l_i = **write**(Q), l_j = **write**(Q). They conflict
- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them.
    - If l_i and l_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

Conflict Serializability

- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are **conflict equivalent**.
- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions.
  - Therefore Schedule 3 is conflict serializable.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 3

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule 6

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule $< T_3, T_4 >$, or the serial schedule $< T_4, T_3 >$.

View Serializability

- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are **view equivalent** if the following three conditions are met, for each data item *Q*,
  1. If in schedule S, transaction T_i reads the initial value of *Q*, then in schedule *S'* also transaction T_i must read the initial value of *Q*.
  2. If in schedule S transaction T_i executes **read**(*Q*), and that value was produced by transaction T_j (if any), then in schedule *S'* also transaction T_i must read the value of *Q* that was produced by the same **write**(Q) operation of transaction T_j .
  3. The transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must also perform the final **write**(*Q*) operation in schedule *S'*.

As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

View Serializability (Cont.)

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- What serial schedule is above equivalent to?
- Every view serializable schedule that is not conflict serializable has **blind writes.**

Other Notions of Serializability

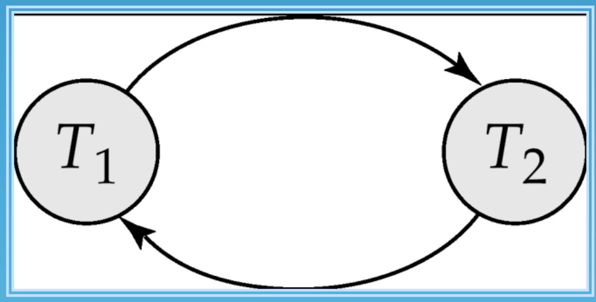
- The schedule below produces same outcome as the serial schedule < T_1, T_5 >, yet is not conflict equivalent or view equivalent to it.

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

- Determining such equivalence requires analysis of operations other than read and write.

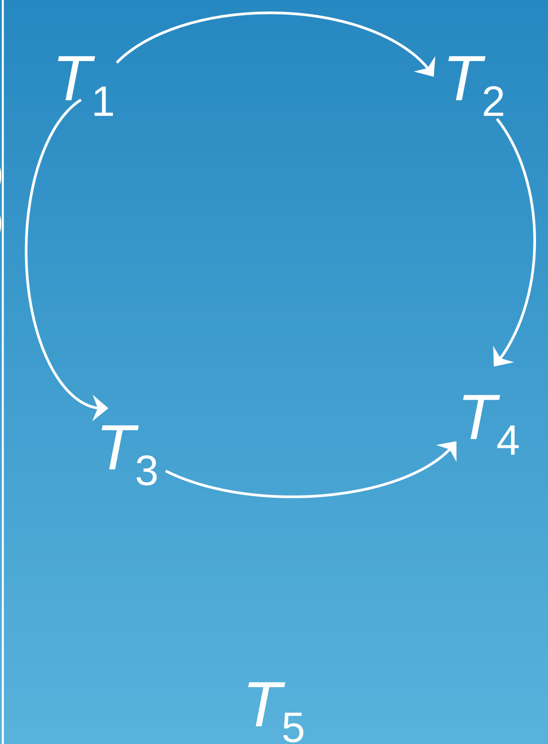
Testing for Serializability

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.
- **Example 1**



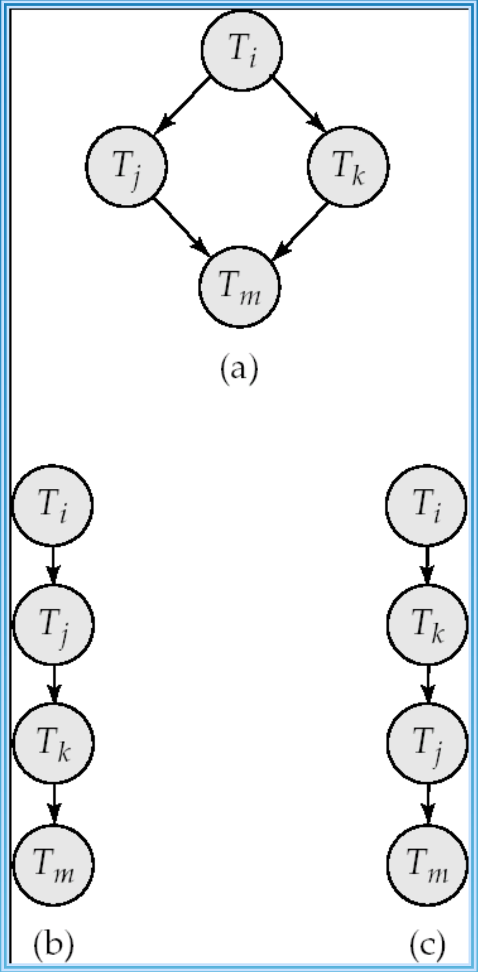
Example Schedule (Schedule A) + Precedence Graph

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Test for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.
  - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a *topological sorting* of the graph.
  - This is a linear order consistent with the partial order of the graph.
  - For example, a serializability order for Schedule A would be $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
    - Are there others?



Test for View Serializability

- The precedence graph test for conflict serializability cannot be used directly to test for view serializability.
  - Extension to test for view serializability has cost exponential in the size of the precedence graph.
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems.
  - Thus existence of an efficient algorithm is *extremely* unlikely.
- However practical algorithms that just check some **sufficient conditions** for view serializability can still be used.

Recoverable Schedules

Need to address the effect of transaction failures on concurrently running transactions.

- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i , then the commit operation of T_i appears before the commit operation of T_j.
- The following schedule (Schedule 11) is not recoverable if T_9 commits immediately after the read

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

Cascading Rollbacks

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

  If T_{10} fails, T_{11} and T_{12} must also be rolled back.

- Can lead to the undoing of a significant amount of work

Cascadeless Schedules

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless

Applications

- Serializability is the major correctness criterion for concurrent transactions' executions. It is considered the highest level of isolation between transactions, and plays an essential role in concurrency control. As such it is supported in all general purpose database systems. Strict and two phase (SS2PL) is a popular serializability mechanism utilized in most of the database systems (in various variants) since their early days.

Scope of research

- Intrusion Detection and Containment in Database Systems
- Crime File management
- Mobile database research
- Computer Integrated Manufacturing
- Spatial databases