Transaction Management

- Instructor

  Dr. Sanam Shahla Rizvi

  PhD in Information and Communication from Ajou University, Korea

Transaction

- Transaction = one execution of a user program.
  - Example: transfer money from account A to account B.
  - A Sequence of Read & Write Operations

- Step of transaction processing:
  - Begin the transaction
  - Execute several data manipulations and queries
  - If no errors occur then **commit** the transaction and end it
  - If errors occur then **abort** and rollback the transaction and end it

ACID Properties

- The DBMS's abstract view of a transaction: a sequence of read and write actions.
- Atomicity: Either all of the actions related to transaction are completed or none of them is carried out.
  - Transfer 1000PKR from account A to account B: R(A), A=A-1000, W(A), R(B), B=B+1000, W(B) => all actions or none (retry).
  - The system ensures this property.
  - How can a transaction be incomplete? Three reasons: aborted by DBMS, system crash, or error in user program.
  - How to ensure atomicity during a crash? Maintain a log of write actions in partial transactions. Read the log and undo these write actions.

ACID Properties (2)

- Consistency: Falls under the area of concurrency control. Consistent state of database must be maintained.

- Isolation: Each transaction should see the consistent database at all the times.

  No other transaction can read or modify data that is being modified by another transaction.

  Two types of problems arise if this property is not maintained:

  1. Lost Updates.
  2. Cascading Aborts.

Time	T1	T2
Time 1	Read x	
Time 2	X=x*2	Read x
Time 3	Write x	x=x+20
Time 4		Write x

(a) Lost Updates

Time	T1	T2
Time 1	(…)	(…)
Time 2	(…)	(…)
Time 3	ABORT	ABORT

(b) Cascading Abort

ACID Properties (3)

- Durability: Once transaction commits its results are permanent and cannot be erased from the database whether system crashes or aborts of other transactions.

    - The system (crash recovery) ensures durability property and atomicity property.

    - DBMS maintains a log of write actions in partial transactions. If system crashes before the changes are made to disk, read the log to remember and restore changes when the system restarts.

Concurrent Execution of Transactions

- Why do concurrent executions of transactions?
  - Better performance.
  - Disk I/O is slow. While waiting for disk I/O on one transaction (T1), switch to another transaction (T2) to keep the CPU busy.
- System throughput: the average number of transactions completed in a given time (per second).
- Response Time: difference between transaction completion time and submission time.
  - Concurrent execution helps response time of small transaction (T2).

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

Schedules

- A transaction is seen by DBMS as a list of read and write actions on DB objects (tuples or tables).
  - Denote $R_T(O)$, $W_T(O)$ as read and write actions of transaction T on object O.
- A transaction also need to specify a final action:
  - commit action means the transaction completes successfully.
  - abort action means to terminate and undo all actions
- A schedule is an execution sequence of actions (read, write, commit, abort) from a set of transactions, over time.
  - A schedule can interleave actions from different transactions.
  - A serial schedule has no interleaving actions from different transactions.

Examples of Schedules

Serial Schedule

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

Schedule with Interleaving Execution

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

Serializable Schedule

- A serializable schedule is a schedule that produces identical result as some serial schedule.
  - A serial schedule has no interleaving actions from multiple transactions.
- We have a serializable schedule of T1 & T2.
  - Assume that T2:W(A) does not influence T1:W(B).
  - It produces the same result as executing T1 then T2 (denote T1;T2).

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

Anomalies in Interleaved Execution

- Situations when non-serializable schedule produces inconsistent results.
- It happens in three possible situations in interleaved execution.
  - Write-Read (WR) conflict
  - Read-Write (RW) conflict
  - Write-Write (WW) conflict

WR Conflict (Dirty Read)

- Situation: T2 reads an object that has been modified by T1, but T1 has not committed.
- A Transaction must leave DB in a consistent state after it completes!

Inconsistent State

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

RW Conflicts (Unrepeatable Read)

- Situation: T2 modifies A that has been read by T1, while T1 is still in progress.
  - When T1 tries to read A again, it will get a different result, although it has not modified A in the meantime.
- A is the number of available copies of a book = 1. T1 wants to buy one copy. T2 wants to buy one copy. T1 gets an error.
  - The result is different from any serial schedule

T1	T2
R(A=1)	
Check if (A>0)	
	R(A=1)
	Check if (A>0)
	W(A=0)
W(A) Error!	
	Commit
Commit	

WW Conflict (Overwriting Uncommitted Data)

- Situation: T2 overwrites the value of an object A, which has already been modified by T1, while T1 is still in progress.
- This is called lost update (T2 overwrites T1's A value, so T1's value of A is lost.)

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

PRECEDENCE GRAPH (DEPENDENCY GRAPH)

- Precedence Graph : In order to know that a particular transaction schedule can be serialized, we can draw a precedence graph.

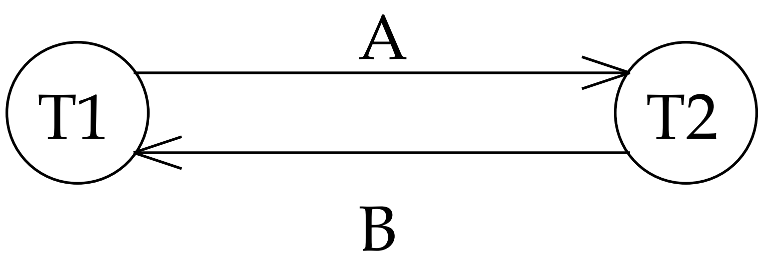
  This is a graph of nodes and vertices, where the nodes are the transaction names and the vertices are attribute collisions.

- The schedule is said to be serialized if and only if there are no cycles in the resulting diagram.

EXAMPLE

- Schedule is not serializable:

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



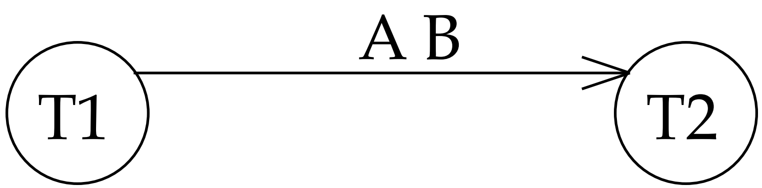
Precendence graph

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

Example

- Schedule is serializable:

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

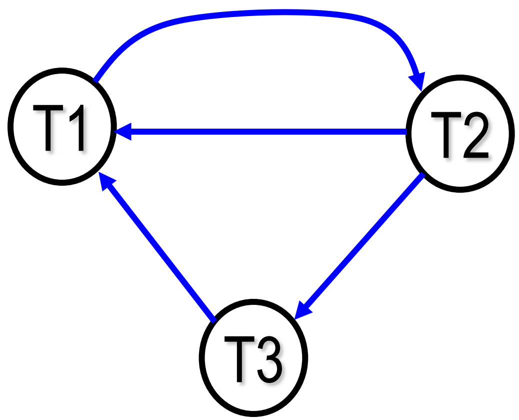


Precedence graph

- No Cycle Here!

Example on Serializability

T1	T2	T3
	read(Z)	
	read(Y)	
	write(Y)	
		read(Y)
		read(Z)
read(X)		
write(X)		
		write(Y)
		write(Z)
	read(X)	
read(Y)		
write(Y)		
	write(X)	



Schedule is NOT serializable

Schedules Involving Aborted Transactions

- Unrecoverable Schedule
  - When a transaction aborts whose values are read, modify and committed by other transaction.
- Recoverable Schedule
  - All transactions commit at same time.
- Strict Schedule
  - Transactions can neither read nor write an item X until the last transaction that wrote X has committed or aborted.
- Avoid Cascading Aborts
  - Read only the changes from already committed transactions.

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