TRANSACTION MANAGEMENT

TRANSACTION CONCEPT

- **Goals:** Understand the basic properties of a transaction and learn the concepts underlying transaction processing as well as the concurrent executions of transactions.

- A transaction is a unit of a program execution that accesses and possibly modies various data objects (tuples, relations).

PROPERTIES OF TRANSACTION:

- DBMS has to maintain the following properties of transactions:

- **Atomicity:** A transaction is an atomic unit of processing, and it either has to be performed in its entirety or not at all.
- **Consistency:** A successful execution of a transaction must take a consistent database state to a (new) consistent database state. (; integrity constraints).

Continue…

- **Isolation:** A transaction must not make its modications visible to other transactions until it is committed, i.e., each transaction is unaware of other transactions executing concurrently in the system. (; concurrency control)

- **Durability:** Once a transaction has committed its changes, these changes must never get lost due to subsequent (system) failures. (; recovery).

Continue…

- Model used for representing database modications of a transaction:

- **read(A,x):** assign value of database object A to variable x;
- **write(x,A):** write value of variable x to database object A.

Continue…

- Example of a Transaction T

read(A,x)
x := x - 200
write(x,A)
read(B,y)
y := y + 100
write(y,B)

Transaction Schedule reflect chronological order of operations

Continue…

- **Goal:** Synchronization" of transactions; allowing concurrency (instead of insisting on a strict serial transaction execution, i.e., process complete T1, then T2, then T3 etc.) ; increase the throughput of the system, ; minimize response time for each transaction.

Continue…

- **Lost Update**

Time	Transaction T1	Transaction T2
1	read(A,x)	
2	x:=x+200	
3		read(A,y)
4		y:=y+100
5	write(x,A)	
6		write(y,A)
7		commit
8	commit	

- The update performed by T1 gets lost; possible solution: T1
- locks/unlocks database object A
- =) T2 cannot read A while A is modied by T1

SERIALIZABILITY :

- DBMS must control concurrent execution of transactions to ensure read consistency, i.e., to avoid dirty reads etc.
- A (possibly concurrent) schedule S is serializable if it is equivalent to a serial schedule S', i.e., S has the same result database state as S'.

- How to ensure serializability of concurrent transactions?

Continue…

- Conflicts between operations of two transactions:

Ti	Tj
read(A,x)	
	read(A,y)

(order does not matter)

Ti	Tj
read(A,x)	
	write(y,A)

(order matters)

Ti	Tj
write(x,A)	
	read(A,y)

(order matters)

Ti	Tj
write(x,A)	
	write(y,A)

(order matters)

- A schedule S is serializable with regard to the above conicts i
- S can be transformed into a serial schedule S' by a series of swaps
- of non-conicting operations.

Continue…

- Checks for serializability are based on precedence graph that describes dependencies among concurrent transactions; if the graph has no cycle, then the transactions are serializable.

- they can be executed concurrently without aecting each others transaction result.

Continue…

CHARACTERIZING SCHEDULES BASED ON SERIALIZABILITY

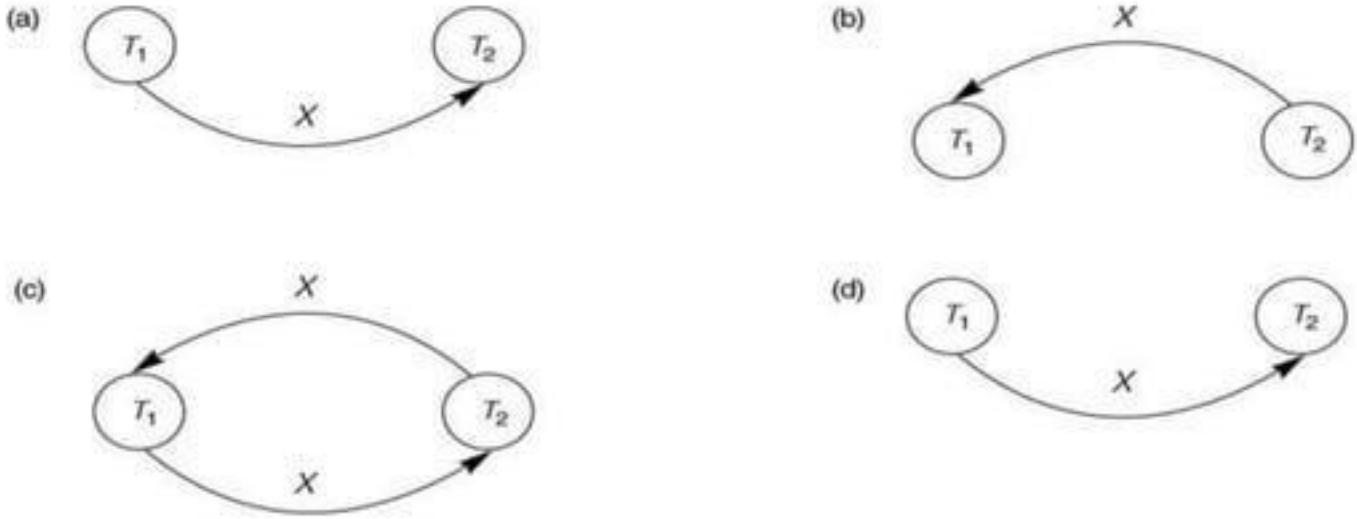
Testing for conflict serializability: Algorithm

- Looks at only read_Item (X) and write_Item (X) operations.
- Constructs a precedence graph (serialization graph) - a graph with directed edges.
- An edge is created from Ti to Tj if one of the operations in Ti appears before a conflicting operation in Tj.
- The schedule is serializable if and only if the precedence graph has no cycles.

Continue…

CONSTRUCTING THE PRECEDENCE GRAPHS:

- **FIGURE 17.7 Constructing the precedence graphs for schedules A and D from Figure 17.5 to test for conflict serializability.**
  - (a) Precedence graph for serial schedule A.
  - (b) Precedence graph for serial schedule B.
  - (c) Precedence graph for schedule C (not serializable).
  - (d) Precedence graph for schedule D (serializable, equivalent to schedule A).



CONCURRENCY CONTROL:

LOCK-BASED PROTOCOLS

- One way to ensure serializability is to require that accesses to data objects must be done in a mutually exclusive manner.
- Allow transaction to access data object only if it is currently holding a lock on that object.
- Serializability can be guaranteed using locks in a certain fashion.

=> Tests for serializability are redundant !

- Types of locks that can be used in a transaction T:

- **slock(X):** shared-lock (read-lock); no other transaction than T can write data object X, but they can read X.

- **xlock(X):** exclusive-lock; T can read/write data object X; no other transaction can read/write X, and

- **unlock(X):** unlock data object X.

Continue…

CONTINUE…

- Lock-Compatibility Matrix:

requested lock	existing lock	
	slock	xlock
slock	OK	No
xlock	No	No

- E.g., xlock(A) has to wait until all slock(A) have been released.

Continue…

- Using locks in a transaction (lock requirements, LR):

- before each read(X) there is either a xlock(X) or a slock(X) and no unlock(X) in between
- before each write(X) there is a xlock(X) and no unlock(X) in between
- a slock(X) can be tightened using a xlock(X)
- after a xlock(X) or a slock(X) sometime an unlock(X) must occur.
- But: "Simply setting locks/unlocks is not sucient"

- replace each **read(X) → slock(X); read(X) unlock(X),**
  and write(X) → xlock(X);write(X);unlock(X).

- **Two-Phase Locking Protocol (TPLP):**
- A transaction T satises the TPLP iff
- after the rst unlock(X) no locks xlock(X) or slock(X) occur
- That is, rst T obtains locks, but may not release any lock (growing phase)
  and then T may release locks, but may not obtain new locks (shrinking phase)

- **Strict Two-Phase Locking Protocol:**
- All unlocks at the end of the transaction T =) no dirty reads are possible, i.e., no other transaction can write the (modied) data objects in case of a rollback of T.

SUMMARY:

- Transaction Concept
- Properties of transaction
- Serializability of transaction
- Testing of serializability
- Two-Phase Commit protocol

THANK YOU