Concurrency Control
(Ref: Chapters 17 and 18, Silberchatz' Text)

INTRODUCTION

- In a Multiprogramming Environment
  - Multiple transactions (trans) may be executed concurrently
  - Need a concurrency control scheme to prevent transactions from destroying database consistency

INTRODUCTION (2)

- Example: A “lost update” problem caused by uncontrolled concurrent transactions
  - Transaction T_1: Cancels N reservations from a flight (1) whose number of reserved seats is in X (Database); then reserves the same number of seats on another flight (2) whose number of reserved seats is in Y.
  - Transaction T_2: Reserves M seats on flight (1)

INTRODUCTION (3)

- Example (cont.):

SCHEDULES

- A schedule: a chronological order in which transactions are executed
- A schedule for a set of transactions:
  - Must consist of all instructions (operations) of those transactions
  - Must preserve the order of instructions within each transaction
- Example: in Transaction T_1
  - if READ (X) appears before WRITE (Y)

  => Any valid schedule that consists of T_1: READ (X) is performed before WRITE (Y) by T_1.

SCHEDULES (2)

e.g. A=$1000 B=$2000

Trans T_0: Transfer $50 from A to B

T_0: Read (A)
A=A-50
Write (A)
Read (B)
B=B+50
Write (B)

Trans T_1: Transfer 10% of balance from A to B

T_1: Read (A)
Temp:= A*0.1
A:=A-Temp
Write (A)
Read (B)
B=B+ Temp
Write (B)

Consistency Constraint: Total balance in A+B must be unchanged after execution of T_0, T_1.

SCHEDULES (3)

- Questions: what are the possible serial schedules? Are they correct schedules?
- Answer:

SCHEDULES (4)

- Question: show one correct non-serial schedule.
- Answer:

SCHEDULES (5)

- Question: show one incorrect non-serial schedule.
- Answer:

SCHEDULES (6)

- Assumptions:
  - Every transaction is correct if executed on its own
  - Transactions do not depend on one another
  - Every Serial Schedule is correct
- REQUIREMENT: for concurrent execution
  - A schedule, after execution, must leave the database in a consistent state, i.e. it must be equivalent to some serial schedule.
  - i.e. it must be a Serializable Schedule

SCHEDULES (7)

- Definition: A serializable schedule of n concurrent transactions: equivalent to some serial schedule of the same n transactions
- Otherwise => Non-serializable Schedule

SCHEDULES (8)

- From a scheduling point of view, only significant operations of a trans are READ and WRITE
- Let I_i be an operation of Transaction T_i
- Let I_j be an operation of Transaction T_j
- I_i and I_j are <u>conflict</u> if I_i and I_j <u>access the same data item</u> and either I_i or I_j is a <u>WRITE</u> operation (Read vs. Write, Write vs. Write)

- In a schedule S, if I_i and I_j <u>are not conflict</u>, then we can swap the order of I_i and I_j to produce a new schedule S′:

  $S \equiv S'$ (Equivalent)

SCHEDULES (9)

e.g.

S

T0	T1
Read (A)	
	Read (B)
Write (C)	

SWAP

S′

T0	T1
	Read(B)
Read (A)	
Write (C)	

$$S \equiv S'$$

SCHEDULES (10)

e.g.

S_1

T_0	T_1
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

S_2

T_0	T_1
Read (A)	
Write (A)	
	Read (A)
Read (B)	
	Write (A)
Write (B)	
	Read (B)
	Write (B)

S_4

T_0	T_1
Read (A)	
Write (A)	
Read (B)	
	Read (A)
Write (B)	
	Write (A)
	Read (B)
	Write (B)

S_3

T_0	T_1
Read (A)	
Write (A)	
Read (B)	
	Read (A)
	Write (A)
Write (B)	
	Read (B)
	Write (B)

SCHEDULES (11)

T_0	T_1
Read (A)	
Write (A)	
Read (B)	
Write (B)	
	Read (A)
	Write (A)
	Read (B)
	Write (B)

Serial Schedule

$S_1 \equiv S_2 \equiv S_3 \equiv S_4 \equiv S_5$

=> S_1, S_2, S_3, S_4 : **Serializable Schedules**

CONCURRENCY CONTROL TECHNIQUES

- To ensure serializability of transaction schedules
- Three major types:
  - Lock-based techniques
  - Timestamp-based techniques
  - Validation-based techniques

CONCURRENCY CONTROL TECHNIQUES (2)

a) Lock-based Techniques:

- Allow a trans to access a data item only if it is currently holding a lock on that item.
  - Example: 2-phase locking, graph-based locking

- Locks: 2 types
  - Shared Lock: S
    - If trans T holds an S lock on data item Q => T can read Q but cannot write Q
  - Exclusive Lock: X
    - If trans T holds an X lock on data item Q => T can read Q and can write Q

CONCURRENCY CONTROL TECHNIQUES (3)

Lock Compatibility Matrix

Locks requested by T_j

Locks held by T_i

	S	X
S	TRUE	False
X	False	False

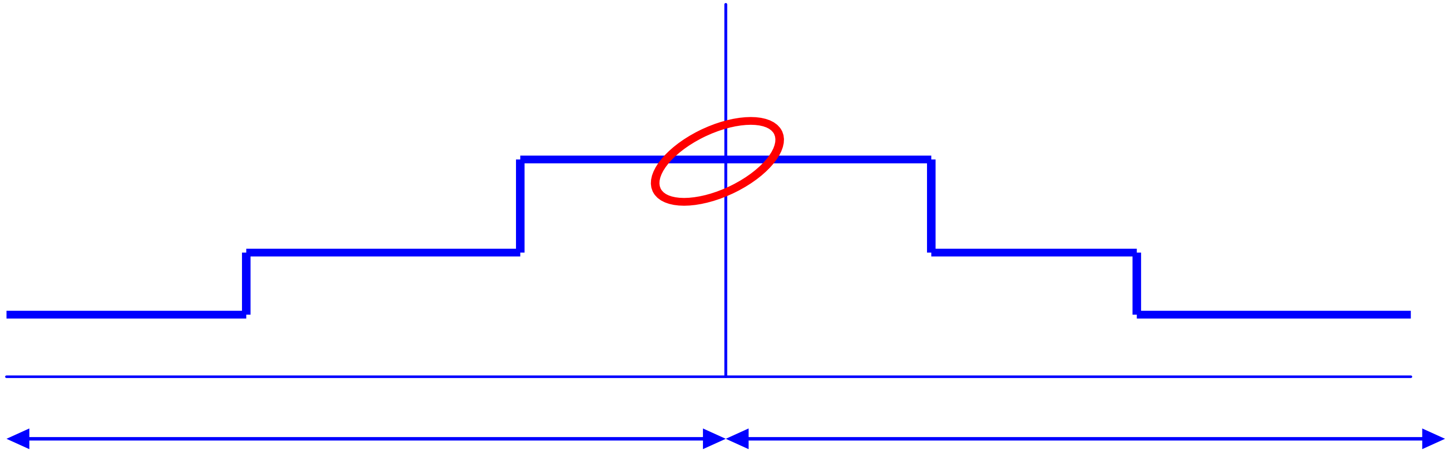
CONCURRENCY CONTROL TECHNIQUES (4)

- A transaction requests an S lock on Q by executing lock_S (Q)
- A transaction requests an X lock on Q by executing lock_X (Q)
- A transaction unlocks a data item Q by executing unlock (Q)
- To access a data item Q, trans T_i must:
  - request a lock on Q
    - if Q is currently locked by another trans in an incompatible mode => T_i must wait.

CONCURRENCY CONTROL TECHNIQUES (5)

- Two-phase locking Technique:
  - Ensures serializability
  - Each trans issues lock and unlock requests in 2 phases
    - Growing phase: A trans may obtain locks but may not release any lock
    - Shrinking phase: A trans may release locks but may not obtain any new locks

CONCURRENCY CONTROL TECHNIQUES (6)



Growing phase **Shrinking phase**

This technique: does not ensure freedom from deadlock

CONCURRENCY CONTROL TECHNIQUES (7)

- Example:

CONCURRENCY CONTROL TECHNIQUES (8)

b) Timestamp-based techniques:

- Determine the serializability order by selecting an ordering among transactions using timestamps in advance.
- Example:
  - Timestamp-ordering scheme
  - Thomas' write rule

CONCURRENCY CONTROL TECHNIQUES (9)

- Timestamp:
  - Associate a unique timestamp TS with each trans T_i $TS(T_i)$ when T_i enters the system
  - If trans T_j enters the system after T_i, then:

$$TS(T_j) > TS(T_i)$$

  - Associate with each data item X two timestamp values:
    - W_TS (X): Write Timestamp of X: largest timestamp among all timestamps of trans that have successfully executed Write(X).
    - R_TS (X): similar for Read Timestamp of X

CONCURRENCY CONTROL TECHNIQUES (10)

- Timestamp-Ordering Technique:
  - Orders trans based on their timestamps
  - A serializable schedule: equivalent to a serial schedule that corresponds to the order of trans timestamps
  - Ensures that any conflicting READ and WRITE operations are executed in the timestamp order as follows:

CONCURRENCY CONTROL TECHNIQUES (11)

- When T_i issues READ (X), check:
  - If TS (T_i) <W_TS (X):
  - If TS (T_i) >= W_TS (X):

CONCURRENCY CONTROL TECHNIQUES (12)

- When Ti issues WRITE (X), check:
  - If TS (T_i) <R_TS (X):
  - If TS (Ti) < W_TS (X):
  - Otherwise:
- Note: in the Timestamp-ordering scheme:
  - There is no deadlock
  - Cascading rollback is possible -> Why?

CONCURRENCY CONTROL TECHNIQUES (13)

c) Validation-Based Techniques (Optimistic):

- No conflict checking is done during trans execution
- Each trans goes through 2 or 3 phases:
  - 1-READ PHASE:
- At the end of trans execution:
  - 2-VALIDATION PHASE:
  - 3-WRITE PHASE:
- Phase 1+2: for what trans?
- Phases 1+2+3: for what trans?

END OF TOPIC
“Concurrency Control”