CSE 411 / L-14 / 16.10.2024 /

Concurrency Control

* Data lock mode:

    (i) Exclusive Lock (X) ⇒ Read + Write

    (ii) Shared Lock (S) ⇒ Read only

* Lock compatibility matrix:

    - applicable only between two conflict transaction.

	S	X
S	Allowed	Not Allowed
X	Not Allowed	Not Allowed

* Account ⇒ 1001

T_1 : Withdraw money from 1001

T_2 : check balance of 1001

⇒ Possible schedule:

T_1	T	
LOCK-S(1001)		→ Granted
READ(1001)		
value = value - 1000		
	LOCK-S(1001)	→ Granted
	READ(1001)	
LOCK-X(1001)		→ wait until, T_2
WRITE(1001)		

⇒ more possible schedule:

T_1	T_2
✓ S	
	S ✓
× S→X ↑	

T_1	T_2
✓ X	
	S ×

T_1	T_2
✓ S	
✓ X	

same transaction,
-no need to check compatability.

Example: Slide ⇒ 3-6

T_1	T_2	T_3	T_4	
LOCK-S(P)				→ GRANT (P, T_1)
READ (P)				
	LOCK-X(Q)			→ GRANT (Q, T_2)
	WRITE(Q)			
		LOCK-S (P)		→ GRANT (P, T_3)
		READ (P)		
			LOCK-X (P)	→ WAIT (P, T_4) ↪ until, T_1 & T_3
			WRITE (P)	

⊛ Lock - Based Protocols

⊛ Unlock as ~~a~~ soon as possible, so that other transaction can run.

T_2	T_3
LOCK-S(A)	
READ (A)	
UNLOCK-S(A)	
	Lock-X (A)
	WRITE (A)
	UNLOCK-X(A)
LOCK-S(B)	
READ (B)	
UNLOCK-S(B)	
Display (A+B)	

Growing Phase for T_2 ↑

Lock point

Shrinking Phase ↓

Growing Phase - T_3

Shrinking phase

this will show wrong output.

⇒ unlock asap. is not a good solution.

⊛ Locking Protocol:

(i) Growing Phase ⇒ until getting/accuring all locks
- unlock is not allowed.

(ii) Shrinking Phase ⇒ No possibility of getting new lock.

Growing Phase

T_2		
LOCK-S(A)		→ Granted
	LOCK-X (A)	→ wait until T_2
LOCK-S (B)		→ Granted
		re-initiate
UNLOCK-S(A)		
UNLOCK-S (B)		
	LOCK-X (A)	→ Granted.

Shrinking Phase

* But there is a deadlock in this protocol.
  - Like: ~~casecading~~ cascading rollback.

⇒ Solution - Strict two phase locking

⇒ UNLOCK-X will be applicable only after commit.

* Two Phase Protocols

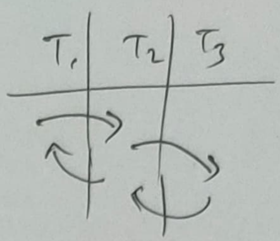
LOCK-S (P)
READ (P)
LOCK-X (Q)
WRITE (Q)
LOCK-S (R)
READ (R)
Display (P+R)
UNLOCK-S(P)
UNLOCK-S(R)
UNLOCK-X(Q) → not allowed in strict two phase protocol
Commit.
UNLOCK-X(Q)

not allowed in Rigorous two phase protocol.

all locks will be held until commit.

Replace for strict two phase protocol.

Cascading Roll back:



- Data flowed from

$T_2 \rightarrow T_2 \rightarrow T_3$

⇒ Somehow, T_2 is invalid then?

T_2 & T_3 also needs to be roll back, as they used the data of T_2.

L-15/21.10.2024/

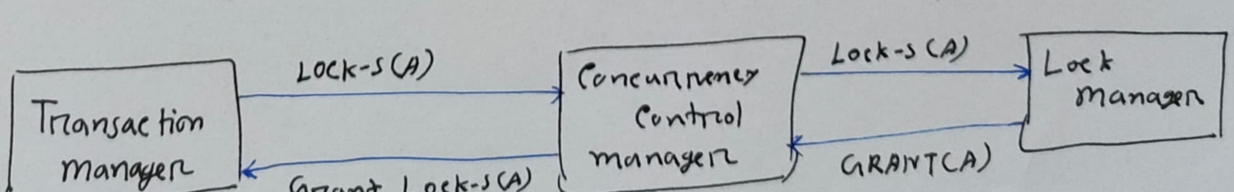
Database Transaction:

- Transaction manager : Concurren schedule generation
- Concurrency Control manger: concurren schedule execution.
- Lock manager: use lock-based protocol.
- Recovery manager: Handling failure and recovery.

Lock Conversion

- upgraded lock must be downgraded first, then the lock can be release.

Automatic Acquisition of Lock:



* Two algorithm for READ & WRITE are given in $\boxed{\text{slide - 15,16}}$

* Implementation of Locking:

  - transaction send lock and unlock requests and waits until its request is answered.

  - lock manager maintain a data-structure called a "lock table" to records granted locks and pending requests.

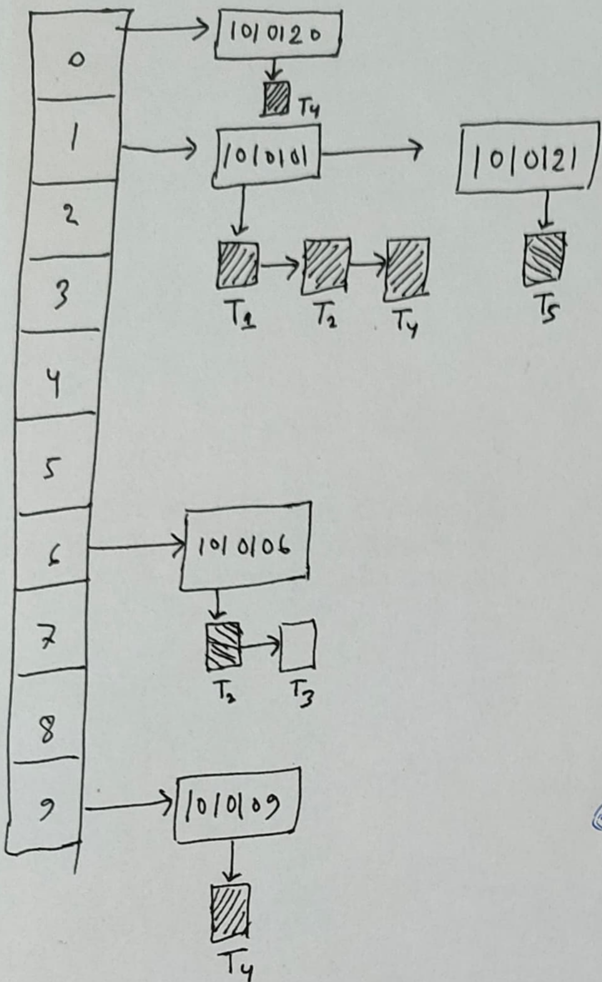
  - lock table implemented as an in-memory hash table indexed on the name of the data item being locked.

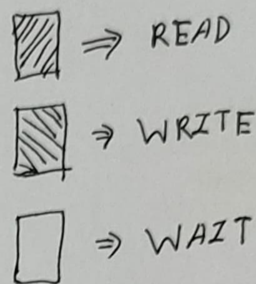
  - may keep a list of locks held by each transaction, to implement this efficiently.

* Example:

Account Number	Transaction:
1010101	T_1 : READ (1010101)
1010106	T_2 : READ (1010101), WRITE (1010106)
1010109	T_3 : READ (1010106)
1010120	T_4 : READ (1010101), READ (1010109), READ (1010120)
1010121	T_5 : WRITE (1010121)

h(f) = account number mod 10

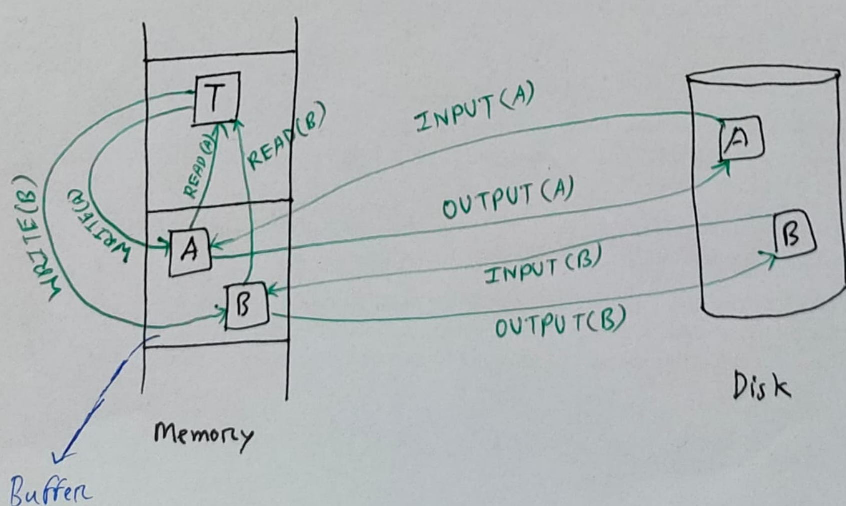




* Starvation deadlock can only be occure for LOCK-X.

L-16/23.10.2024

Database Recovery System



⇒ All ~~Re~~ READ, WRITE will occure on Buffer.

⇒ Input, Output will occure on disk.

⊛ T_1

READ(A) → input from disk

A = A - 500

WRITE (A) → output to disk

← Failure

READ (B)

B = B + 500

WRITE (B)

← Failure

COMMIT

← Failure

⊛ Different recovery algorithm for different point of failure.

⊛ Failure Classification:

⇒ Transaction failure:

- logical errors - internal error condition
- system errors - deadlock

⇒ System crash : hardware / software failure

- Fail-stop assumption : non-volatile storage are assumed to not be ~~cor~~ corrupted by system crash.

⇒ Disk failure:

- head crash
- destroys all or part of disk storage.

Recovery Algorithm:

⇒ Recovery algorithm have two parts:

- action taken during normal transaction process.
  - ⇒ stored logs for each write instructions before executing WRITE.
- action taken after a failure to recover the database content.

Storage structure:

- volatile storage : does not survive system crashes
- non-volatile storage : survive system crashes
  - but there is a chance of loosing data.
- stable storage:
  - mythical form of storage that survives all failure
  - maintain multiple copies on distinct nonvolatile storage.

Data access :

- Physical blocks : residing on the disk.
- Buffer blocks : residing temporarily in main memory.

Block movement between disk and main memory through the two operations :

- input(A)
- output(A)

⇒ transfering data item between buffer and private work area done by:

- read (x)
- write (x)

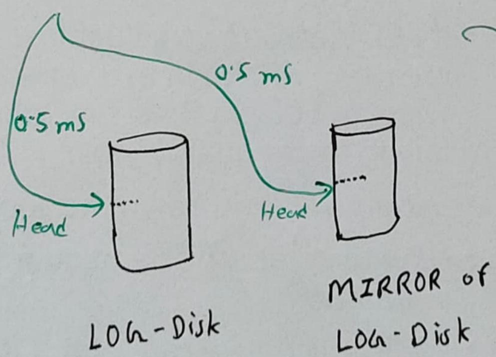
stored local copies

Diagram - important
Slide - 28

⊛ Recovery & Atomicity:

⇒ Log-file stored on a stable storage, with fast writing capabilities. (RAID-1)

WRITE LOG (c):



Total time = 0.5 ms
⇒ writing in parallel.

⊛ LOG- Records includes:

- transaction start : ⟨T_i, START⟩
- Writing/modifieing instruction : ⟨T_i, X, V_1, V_2⟩
  - before executing WRITE instruction.

object/Id
New Value
Previous Value

- transaction end : <T$_i$, COMMIT>

* For the given transaction T_1 :

<u>Log-Records</u>

<T_1 , START>

<T_1 , A, 2000, 1500>

<T_1 , B, 3000, 3500>

<T_1 , COMMIT>

* Answere of the slide Question - $\boxed{\text{Slide-31}}$

⇒

T	Log-Records
	<T, START>
READ (C)	
value = value * 0.20	<T, C, 1500, 1200>
C = C - value	<T, A, 500, 650>
WRITE (C)	<T, B, 1000, 1150>
	<T, COMMIT> → record stored before executing the last instruction; don't care about COMMIT.
READ (A)	
A = A + (value/2)	
WRITE (A)	
READ (B)	
B = B + (value/2)	
WRITE (B)	
COMMIT	

* Two approaches using logs & database modification:

  - Immediate database modification:

    - updates to buffer / disk performed before ~~commits~~ each modification instruction.

  - ~~Deff~~ Deferred database modification:

    - updates to buffer/disk performed only at the time of transaction commit.

* A transaction is said to have commited when its commit log record is output to stable storage.

$\boxed{\text{Example - Slide - 35}}$