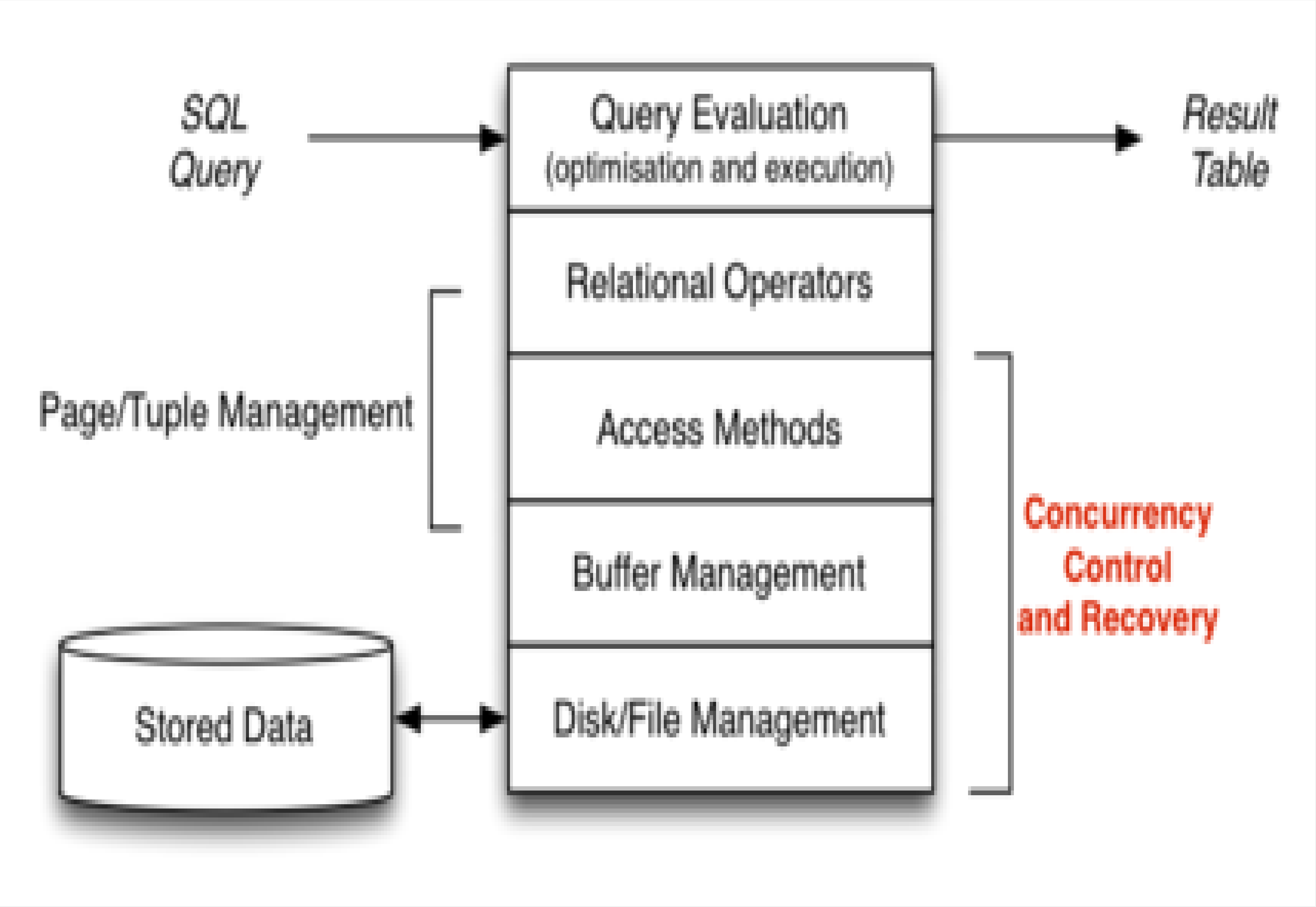
Lecture 42

Concurrency Control – 2PL

DATABASE CONCURRENCY CONTROL

Purpose of Concurrency Control

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.


SQL Query
Query Evaluation
(optimisation and execution)
Result Table
Relational Operators
Page/Tuple Management
Access Methods
Concurrency Control and Recovery
Buffer Management
Stored Data
Disk/File Management

Two-Phase Locking Technique

Locking is an operation which secures

(a) permission to Read

(b) permission to Write a data item for a transaction.

Example: Lock (X). Data item X is locked in behalf of the requesting transaction.

Unlocking is an operation which removes these permissions from the data item.

- Example: Unlock (X): Data item X is made available to all other transactions.

Lock and Unlock are Atomic operations.

Two-Phase Locking Techniques: Essential components

Two locks modes:

- (a) shared (read) (b) exclusive (write).

Shared mode: shared lock (X)

- More than one transaction can apply share lock on X for reading its value but no write lock can be applied on X by any other transaction.

Exclusive mode: Write lock (X)

- Only one write lock on X can exist at any time and no shared lock can be applied by any other transaction on X.

Conflict matrix

	Read	Write
Read	Y	N
Write	N	N

Two-Phase Locking Techniques: Essential components

Lock Manager:

- Managing locks on data items.

Lock table:

- Lock manager uses it to store the identify of transaction locking a data item, the data item, lock mode and pointer to the next data item locked. One simple way to implement a lock table is through linked list.

Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Two-Phase Locking Techniques:

Two Phases:

- (a) Locking (Growing)
- (b) Unlocking (Shrinking).

Locking (Growing) Phase:

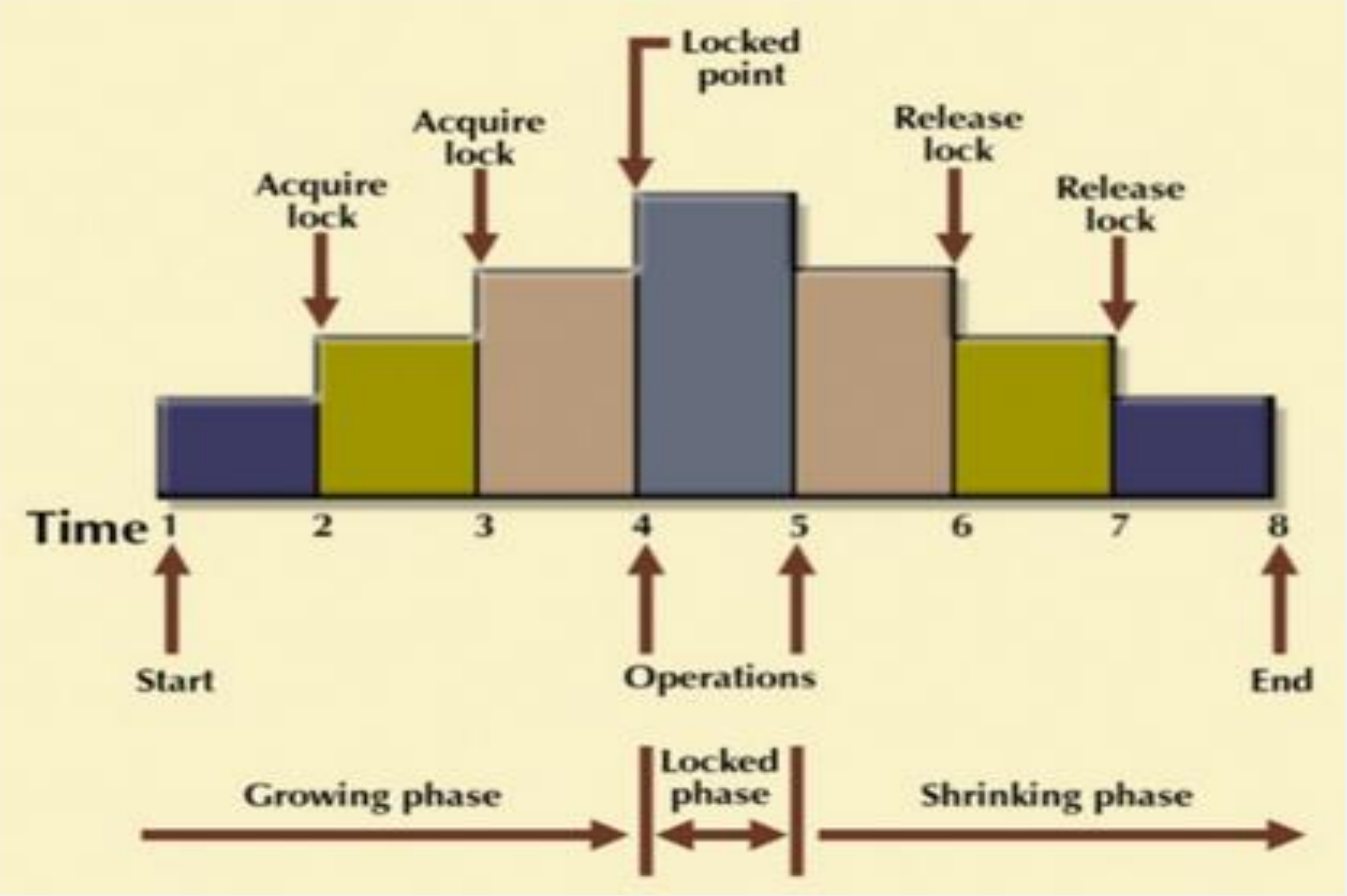
- A transaction applies locks (read or write) on desired data items one at a time.

Unlocking (Shrinking) Phase:

- A transaction unlocks its locked data items one at a time.

Requirement:

- For a transaction these two phases must be mutually exclusively, that is, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin.


Locked point
Acquire lock
Acquire lock
Release lock
Release lock
Time 1
2
3
4
5
6
7
8
Start
Operations
End
Growing phase
Locked phase
Shrinking phase

Two-Phase Locking Techniques: Example

T1	T2	Result
read_lock (Y);	read_lock (X);	Initial values: X=20; Y=30
read_item (Y);	read_item (X);	Result of serial execution
unlock (Y);	unlock (X);	T1 followed by T2
write_lock (X);	Write_lock (Y);	X=50, Y=80.
read_item (X);	read_item (Y);	Result of serial execution
X:=X+Y;	Y:=X+Y;	T2 followed by T1
write_item (X);	write_item (Y);	X=70, Y=50
unlock (X);	unlock (Y);	

Two-Phase Locking Techniques: Example

T1	**T2**	**Result**
read_lock (Y);		X=50; Y=50
read_item (Y);		Nonserializable because it.
unlock (Y);		violated two-phase policy.
	read_lock (X);	
	read_item (X);	
	unlock (X);	
	write_lock (Y);	
	read_item (Y);	
	Y:=X+Y;	
	write_item (Y);	
	unlock (Y);	
write_lock (X);		
read_item (X);		
X:=X+Y;		
write_item (X);		
unlock (X);		

Time

Two-Phase Locking Techniques:

Two-phase policy generates two locking algorithms

- (a) **Basic**
- (b) **Conservative**

Conservative:

- Prevents deadlock by locking all desired data items before transaction begins execution.

Basic:

- Transaction locks data items incrementally. This may cause deadlock which is dealt with.

Strict:

- A more stricter version of Basic algorithm where unlocking is performed after a transaction terminates (commits or aborts and rolled-back). This is the most commonly used two-phase locking algorithm.

2PL

Strict 2PL

Same as 2PL but Hold all exclusive locks until the transaction has already successfully committed or aborted. –It guarantees cascadeless recoverability

Rigorous 2PL

Rigorous two-phase locking is even stricter: here all locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

Strict 2PL	
T1	T2
s-lock(A)	
read(A)	
	s-lock(A)
x-lock(B)	
unlock(A)	
read(B)	
write(B)	
	read(A)
	unlock(A)
commit	
unlock(B)	
	s-lock(B)
	read(B)
	unlock(B)
	commit

Rigorous 2PL	
T1	T2
s-lock(A)	
read(A)	
	s-lock(A)
x-lock(B)	
	read(A)
read(B)	
write(B)	
commit	
unlock(B)	
	s-lock(B)
	read(B)
unlock(A)	
	commit
	unlock(A)
	unlock(B)

THE NORTHCAP UNIVERSITY

Practice Problem

Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

Identify the Type of Schedule?

Practice Problem Solution

Consider the following schedule involving two transactions T_1 and T_2.

T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

This is a strict schedule since T_2 reads and writes A which is written by T_1 only after the commit of T_1.

Practice Problem

Consider the following database schedule with
two transactions, T1 and T2.

S = r2(X); r1(X); r2(Y); w1(X); r1(Y); w2(X); a1; a2;
where ri(Z) denotes a read operation by transaction Ti
on a variable Z, wi(Z) denotes a write operation by Ti on
a variable Z and ai denotes an abort by transaction Ti .Which one
of the following statements about the above schedule is TRUE?

(A) S is non-recoverable
(B) S is recoverable, but has a cascading abort
(C) S does not have a cascading abort
(D) S is strict

Practice Problem Solution

T1	T2
	R(X)
R(X)	
	R(Y)
W(X)	
R(Y)	
	W(X)
a1	
	a2

T1 performs Write operation on X

T2 performs write operation on same variable X

PRACTICE PROBLEM SOLUTION

As we can see in figure,

- T2 overwrites a value that T1 writes
- T1 aborts: its "remembered" values are restored.
- Cascading Abort could have arised if – > Abort of T1 required abort of T2 but as T2 is already aborted , its not a cascade abort. Therefore, **Option C**
- **Option A** – is **not** true because the given schedule is recoverable
- **Option B –** is **not** true as it is recoverable and avoid cascading aborts;
- **Option D** – is **not** true because T2 is also doing abort operation after T1 does, so NOT strict.

Practice Problem

Consider the following schedule involving two transactions T_1 and T_2.

	T_1	T_2
1	LOCK-S(A)	
2		LOCK-S(A)
3	LOCK-X(B)	
4	…….	……
5	UNLOCK(A)	
6		LOCK-X(C)
7	UNLOCK(B)	
8		UNLOCK(A)
9		UNLOCK(C)
10	…….	……

Is this transaction implements 2PL?

Practice Problem Solution

Yes this transaction implements 2PL

This is just a skeleton transaction which shows how unlocking and locking works with 2-PL. Note for:

Transaction T_1:

Growing Phase is from steps 1-3.

Shrinking Phase is from steps 5-7.

Lock Point at 3

Transaction T_2:

Growing Phase is from steps 2-6.

Shrinking Phase is from steps 8-9.

Lock Point at 6

Dealing with Deadlock and Starvation

- **Deadlock**

T'1	**T'2**
read_lock (Y); read_item (Y);	
	read_lock (X); read_item (Y);
write_lock (X); (waits for X)	
	write_lock (Y); (waits for Y)

T1 and T2 did follow two-phase policy but they are deadlock

- Deadlock (T'1 and T'2)

Dealing with Deadlock and Starvation

Deadlock prevention

- A transaction locks all data items it refers to before it begins execution.
- This way of locking prevents deadlock since a transaction never waits for a data item.
- The conservative two-phase locking uses this approach.

Dealing with Deadlock and Starvation

Deadlock detection and resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a **wait-for-graph** for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled-back.

A **wait-for-graph** is created using the lock table. As soon as a transaction is blocked, it is added to the graph. When a chain like: Ti waits for Tj waits for Tk waits for Ti or Tj occurs, then this creates a cycle.

Dealing with Deadlock and Starvation

Deadlock Avoidance

- There are many variations of two-phase locking algorithm.
- Some avoid deadlock by not letting the cycle to complete.
- That is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.
- Wound-Wait and Wait-Die algorithms use timestamps to avoid deadlocks by rolling-back victim.

Dealing with Deadlock and Starvation

Starvation

- Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further.
- In a deadlock resolution it is possible that the same transaction may consistently be selected as victim and rolled-back.
- This limitation is inherent in all priority based scheduling mechanisms.
- In Wound-Wait scheme a younger transaction may always be wounded (aborted) by a long running older transaction which may create starvation.

THANKS!!