Lec 73) <u>Introduction to Transaction Concurrency</u>

Transaction: It is a set of operations used to perform a logical unit of work

→ A transaction generally represents change in database.

Read, Write are the two types of transactions

Lec 74: ~~Intro~~ ACID properties of a Transaction

ACID properties
↓
Atomicity Consistency Isolation Durability

Atomicity - Either all or none

If all the operations in the transaction do not get executed before commit, the transaction must be rolled back. (i.e. start again)

Consistency :- Before the transaction has started and after the transaction ends, the sum of total money (or value) must be same.

Eg:- A = 2000 } 5000
B = 3000 }

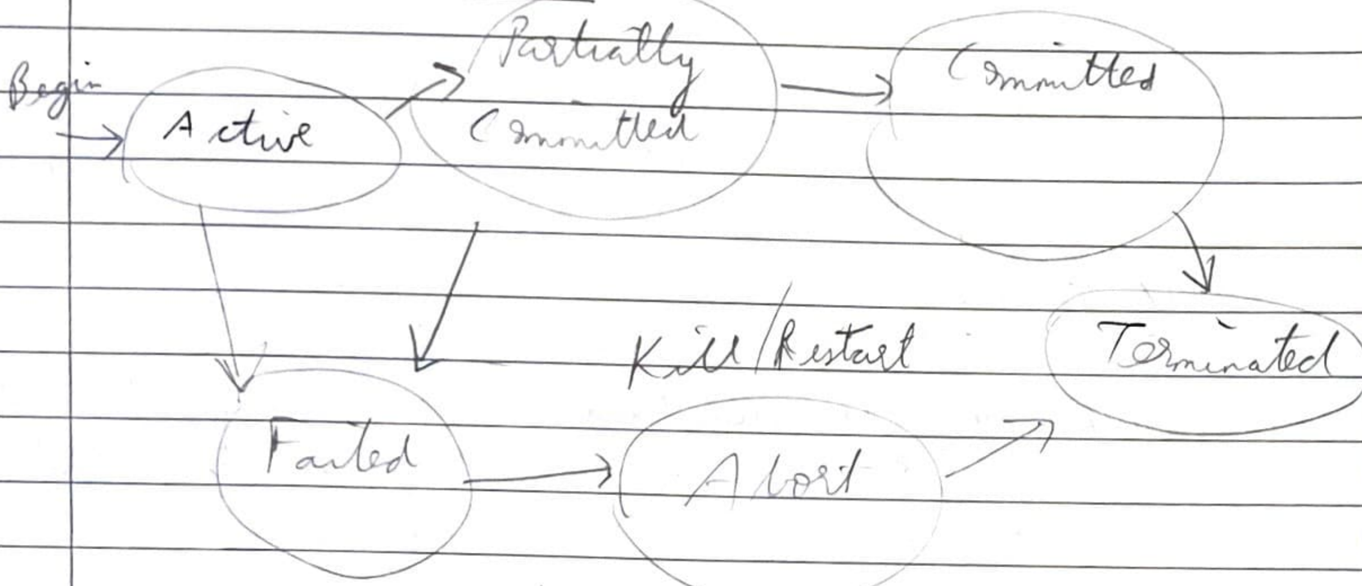
T

R(A) 2000 A = A - 1000 W(A) - 1000 R(B) 3000 B = B + 1000 W(B) - 4000	Commit After committing, A = 1000 } 5000 B = 4000 }

Isolation :- If there are many transactions running can I convert parallel schedule to a serial schedule?

Durability :- Whatever changes done should be permanent.

Lec 75: Transaction states



* When the transaction is partially committed, all changes are saved in the RAM

* When the transaction is fully committed, all changes are saved in the hard disc.

* Termination means that all the resources are delocated.

Lec 76: What is Schedule?

Schedule: It is chronological execution sequence of multiple transactions

Schedule

serial Parallel

* If there are 3 transactions T_1, T_2, T_3, if T_1 starts, T_2 starts if T_1 ends and so on, in serial schedule.

* In parallel schedule, ~~if~~ any transaction can start at any time.

Disadvantage of serial ~~Trans~~ schedule

* Decrease in throughput (No. of transactions per unit time)

Lec-77:- All Concurrency Problems

Types of problems in concurrency

1) Dirty Read
2) Incorrect summary
3) Lost update
4) Unrepeated read
5) Phantom read

1) Dirty Read or Uncommited Read or Read-after-write

* If transaction T2 reads the resources before T1 commits, dirty read problem occurs

2) Incorrect summary

* If T1 is ~~to~~ doing read-write operation and T2 is finding the aggregate (say sum) of the resources, the ~~error~~ error occurs.

3) * Lost update

When T1 and T2 are writing to a memory, if any one transaction writes the update is lost when the next transaction writes the memory.

4) Unrepeatable read

When one transaction writes the memory the other transaction reads different values at different times.

5) Phantom read

When a transaction deletes the value in the memory and still if ~~the~~ another transaction reads ~~the~~ that ~~value~~ memory then it is phantom read ~~e~~ error.

Lec-78: Write-Read conflict or dirty-read problem

	T_1	T_2
70 ←	R(A)	
20 ←	A = A - 50	
20 ←	W(A)	
		R(A) → 20
		A = A * 2 → 40
		W(A) → 40
	R(B)	
	W(B)	

Time

* If T_1 fails, then there will be a rollback
* Due to this the value of A will be set to 70 again.
* But, the T_2 has read the value of A as 20. There arises a conflict here. This is called dirty read conflict.

Lec 79: Read-Write Conflict or Unrepeatable Read Problem

A=2	T_1	T_2
	2 R(A)	
		R(A) 2
Inconsistency		W(A) → A=A-2 → 0
↓		Commit
R-W conflict	0 R(A)	
	W(A)	
	Commit	

Lec 80: Irrecoverable VS Recoverable schedules in transactions

A=10	T_1	T_2
10	R(A)	
5	A=A-5	
5	W(A)	
		R(A) 5
Rollback		A=A-2 3
		W(A) 3
		Commit
	R(B)	
Fail	~~Commit~~	

Since T_1 has to rollback the value of A, changes to the value of A will be lost. So, this schedule is called an irrecoverable schedule.

Lec 81 :- Cascading VS Cascadeless schedule with example

T_1	T_2	T_3	T_4
R(A)			
W(A)			
	R(A)		
		R(A)	
			R(A)
T_1 fail			

In this schedule if T_1 fails and rolls back, all the other transactions T_2, T_3, T_4 will also rollback. This type of schedule is called cascading schedule.

Disadvantage :- CPU performance will degrade as CPU utilisation will be high and unnecessary.

Cascadeless schedule :- Is a schedule where one transaction cannot read a memory until another transaction commits the memory.

Write - Write Problem

* Cascadeless schedule does not have a restriction on one transaction writing after another transaction writing.

* This arises to write - write problem.

Lec 82 :- Introduction to Serialisability

Serialisable schedule is a set of transactions where the transactions execute one after ~~the~~ another

Eg :- 1) S

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

$T_1 \to T_2$

2) S

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

$T_1 \leftarrow T_2$

~~Eg~~ Eg 2 :- Examples of parallel schedule

S_1

(1)

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

If we want to convert this to a serial ~~schedule~~ schedule, we have 2 possibilities

$T_1 \to T_2$ OR $T_2 \to T_1$

(2)

T_1	T_2	T_3
	R(A)	
		R(A)
		W(A)
	W(A)	
R(B)		
W(B)		
	W(B)	

Possibilities :-

$T_1 \to T_2 \to T_3$

$T_1 \to T_3 \to T_2$

$T_2 \to T_3 \to T_1$

$T_2 \to T_1 \to T_3$

$T_3 \to T_1 \to T_2$

$T_3 \to T_2 \to T_1$

There are $3! = 6$ possibilities of equivalent serial schedule

The state whether there exists a serial schedule which is equivalent to the parallel schedule is called serialisability

Serialisability

View | Conflict

Lec 83: Conflict Equivalent Schedules with Examples

R(A) R(A) } Non conflict pairs

R(A) W(A)
W(A) R(A)
W(A) W(A) } conflict pairs

R(B) R(A)
W(B) R(A)
R(B) W(A)
W(A) W(B) } Non-conflict pairs

* To serialise schedules find out non-conflict pairs which are adjacent and swap the positions

Eg: S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	

S''

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)

S'

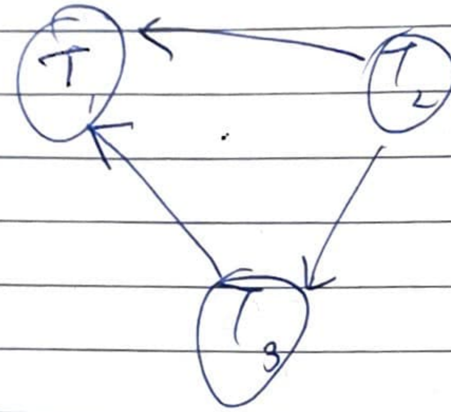
T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

$S \equiv S'$

Lec 84) : Conflict Serialisability – Precedence Graph

- Check conflict pairs in other transactions and draw edges.

T_1	T_2	T_3
R(X)		
		R(Y)
		R(X)
	R(Y)	
	R(Z)	
		W(Y)
	W(Z)	
R(Z)		
W(X)		
W(Z)		



i) R(X) → No conflict pairs

ii) R(Y) → No conflict pairs

iii) R(X) in T_3 → W(X) in T_1, so edge from T_3 to T_1

iv) R(Y) in T_2 → W(Y) in T_3, so edge from T_2 to T_3

v) R(Z) in T_2 → W(Z) in T_1, so edge from T_2 to T_1

vi) $W(Y)$ in T_3 - No conflict

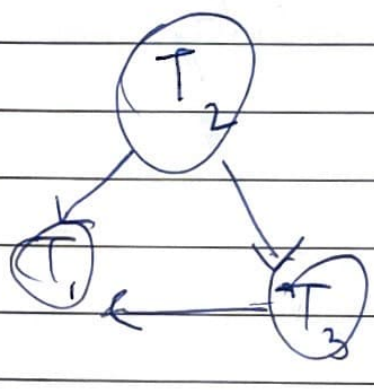
vii) $W(Z)$ in T_2 - $R(Z)$ in T_1

viii) Next, we have to check whether there is a loop / cycle.

In this graph, there is no loop. If there is no loop / cycle, The schedule is conflict-serializable, hence the schedule is consistent.

To find the serial equivalent use vertex deletion method to find the topological order of the transactions in the precedence graph.

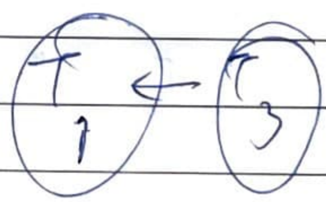
Eg:-



Indegree :- T_1 — 2
T_2 — 0
T_3 — 1

$T_2 \rightarrow T_3$

Remove T_2



Remove T_3

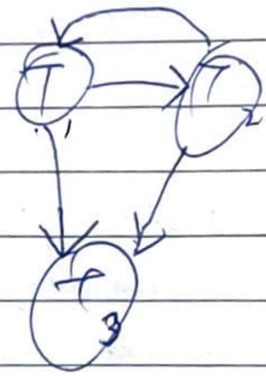
∴ The serial equivalent schedule is $T_2 \rightarrow T_3 \rightarrow T_1$

Lec-85: View Serialisability

S

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

Check whether schedule is conflict - serialisable or not.



This is a non-conflict serialisable

If there is a loop in the precedence graph, & to check serialisability, we check view serialisability

~~View~~

$$S \equiv S'$$

S

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

S'

T_1	T_2	T_3
R(A)		
W(A)		
	W(A)	
		W(A)

This may not be conflict-serialisable, but these 2 schedules are view-serialisable.

Lec 86: Shared / Exclusive Locking Protocol with example

→ Shared lock (S) => if transaction has locked data item in shared mode then it is allowed to read only.

→ Exclusive lock (X) => if transaction has locked data item in exclusive mode then it is allowed to read and write both.

* Problems in S/X locking

1) May not be sufficient to produce only serialisable schedule

2) May not be free from irrecoverability

3) May not be free from deadlock

4) May not be free from starvation

Request →

		S	X
grant ↓	S	✓	×
	X	×	×

1) Since we are using locks to remove ~~conflicts~~ conflicts, the order of operations may not be altered.

2)

PTO

2)

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	Commit

If T_1 fails and rolls back, T_2 won't rollback. So, this is not a recoverable schedule

3)

	T_1	T_2	
G	X(A)		
		X(B)	G
W	X(B)		
		X(A)	W

This is a deadlock situation as T_1 is waiting ~~T_2~~ for T_2 to release the lock on B ~~A~~ and T_2 is waiting for T_1 to release the lock on ~~T_2~~ A.

4)

T_1	T_2	T_3	T_4
	S(A)		
X(A)			
		S(A)	
	U(A)		
			S(A)
		U(A)	
			U(A)

Until T_4 releases the shared lock, T_1 has to keep waiting.

Lec-88. 2 Phase Locking (2PL) Protocol in Transaction

Concurrency Control

→ Growing Phase: Locks are acquired and no locks are released.

→ Shrinking Phase: Locks are released and no locks are acquired.

S/X

	T1	T2
	Lock S(A)	
		Lock S(A)
Lock point	Lock X(B)	
	Unlock (A)	
		Lock X(D)
	Unlock (B)	
		→ Lock point
		Unlock (A)
		Unlock (B)

T1	
X(A)	Growing phase
S(B)	
R(A)	
W(A)	
R(B)	
S(A)	
R(C)	
S(D)	
R(D)	
U(A)	→ Shrinking phase

	T1	T2
Growing phase	X(A)	
	R(A)	
	W(A)	
		S(A)
		R(A)
	S(B)	
Shrinking phase	R(B)	
	U(A)	

T1	T2
U(B)	

Through 2PL, Serialisability is achieved.

If one transaction asks for X lock when another transaction already has S lock, then that lock will be blocked.

Lec-89: Drawbacks in 2 PL Protocol with examples

Advantages: Always ensures serialisability

Disadvantages: 1) ~~May~~ not free from irrecoverability

2) Not free from deadlocks

3) Not free from starvation

4) Not free from cascading rollback

1)

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
⋮	⋮
	commit
fail	

If T_2 commits before T_1 and T_1 rolls back, then T_2 cannot rollback. This makes the schedule irrecoverable.

2) Due to the above problem, cascading rollback is not possible.

_/_/_

Lec 90: Strict 2PL, Rigorous 2PL, and conservative 2PL

Schedule

Strict 2PL: It should satisfy the basic ~~2~~ 2PL and all exclusive locks should hold until commit/abort.

Rigorous 2PL: It should satisfy the basic 2PL and all shared, exclusive locks should hold until commit/abort.

Strict 2PL are always strict recoverable and cascadeless.

~~There~~ In These 2 locks, there is no solution for ~~death~~ deadlock and starvation

Conservative 2PL gives all the resources to only one transaction.