Transaction and Concurrency

Transaction :-

A Transaction is a collection of operation that forms a single logical unit of work

Ex

```
R(A)
A = A-50
W(A)
R(B)
B = B+50
W(B)
```

} Example

* NOTE Two main issues to deal with Transaction

1. Failure of various kind
2. Concurrent execution of multiple transaction (inconsistency) problem.

Ex

initial A = 100

T_1	T_2
R(A)	
A = A-50	
	R(A)
	A = A-25
W(A)	
	W(A)
R(B)	
B = B+50	
W(B)	

→ Now in order to manage all this we need transaction management System.

ACID property

1) Atomicity : All (or) None

* Execute all the steps/operations of one transaction or in case of power failure or any interrupt just rollback-whatever is done
* Transaction manager software is responsible

2) Consistency :- Correctness

* Execute of a transaction in isolation preserve the consistency of a database
* Application programer is responsible

3) Isolation :-

→ Each Transaction T_x must be executed without knowing what is happening with other Transactions.

→ It is going to control the concurrent execution of multiple Transactions

→ Concurrency control manager is responsible

Durability :-

After Transaction completes successfully the change it has made to database persists.

→ Even there is a system failure, all updates done by Transaction must be permanent

→ Recovery Manager is responsible.

Transaction States

Active → Partially Committed → Committed (permanent)

Active → Failed → Aborted (rollback)

Concurrent Execution :-

Multiple Transactions are allowed to run.

Advantages

Increases Disk Utilization.

Reduce Average Response Time

problems

1) Lost Update Problem (W-W conflict)

initial A = 100

T_1	T_2
R(A) 100	
A = A-50 50	
W(A) = 50	R(A) 100
R(B)	A = A+50 150
B = B+50	W(A) = 150
W(B)	

Lost

NOTE

When you have 2 write on the same data item without any read operation in the middle of the data item, then it is going to lead us to W-W conflict.

2) Dirty Read problem (W-R conflict)

Initial A = 100.

T_1	T_2
R(A) 100	
A = A-50	
W(A)	
Failed due to Issue → roll back	R(A) 50
	A = A+10 = 60
	W(A) = 60
R(B)	
B = B+50	
W(B)	
commit	

here T_1 is not commited and T_2 is read the value this is not correct

→ Just wait for the transaction to complete, then only you are supposed to read it.

→ Reading an uncommitted data is called "Dirty Read"

② Non-Repeatable Read (R-W Conflict).

A = 500.

T_1	T_2
R(A) ↳ 500	
	R(A) A = A - 100 W(A) → 400
400 ↑ R(A)	

First time in T_1 Reads 500 as value after performing the value becomes 400. of same Transaction.

③ Phantom Tuple (Phantom Phenomena). (Extra Tuple)

T_1

Eno	Ename	sal
1	A	500
3	C	400

select * from Emp.
Sal ≥ 300

Eno	Ename	Sal
1	A	500
3	C	400
4	D	350

T_2

Insert into Emp values (4, D, 350)

here Tuple is add in T_2 but changes to had made in T_1 Transaction.

Schedule

A schedule represents the order in which the operations of Transactions are executed.

$T_1 = a_1 a_2$

$T_2 = b_1 b_2$

S_1		S_2		S_3		S_4		S_5		S_6	
T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
a_1			b_1	a_1		a_1	b_1		b_1		b_1
a_2			b_2		b_1		b_2	a_1		a_1	
	b_1	a_1		a_2		a_2			b_2	a_2	
	b_2	a_2			b_2			a_2			b_2

* Within a transaction you are supposed to follow the order
* Two schedule among them will always be right S_1 and S_2
* S_1 and S_2 are Serial Schedules. This only problem with serial schedules is performance is bad

Schedule

T_1 : n operations

T_2 : m operations

Number of schedule $= \frac{(n+m)!}{n!\,m!}$

If $n=2$, $m=2$ then

no. of schedule $= \frac{(2+2)!}{2!\,2!} = \frac{4!}{2!2!} = \frac{4 \times 3 \times 2 \times 1}{2 \times 1 \times 2 \times 1} = 6$

Total 6 the Schedule is possible but on 6 we have 2 Serial Schedules other are Non-Serial Schedules.

Total no. of Serial Schedule possible = n!
where n is the no. of Transactions.

→ Non-Serial Schedules = Total schedules − Total Serial Schedules

→ Non-Serial schedules are also called as Concurrent Schedules.

Ex $T_1 = 2, T_2 = 3, T_3 = 4$ find the no. of Schedule

$$\frac{(2+3+4)!}{2!\,3!\,4!} = \frac{9!}{2!\,3!\,4!} = \frac{9\times 8\times 7\times 6\times 5\times 4!}{2!\times 3!\times 4!}$$

$$= 36\times 7\times 5$$

$$= 1260 \text{ Total Schedule}$$

no. of Serial Schedule = n! = 3!

= (6) ✓

Complete Schedule

A Schedule is said to be complete if the last operation of each Transaction is either Abort or Commit

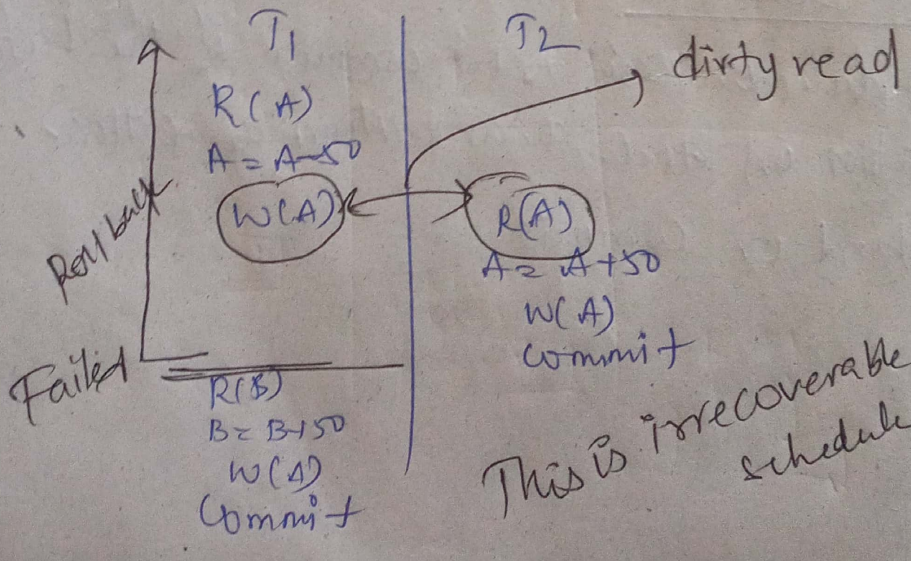
S1

T_1	T_2
R(A	
	R(A)
A = A-50	
W(A)	
Commit	
	A = A+50
	W(A)
	abort

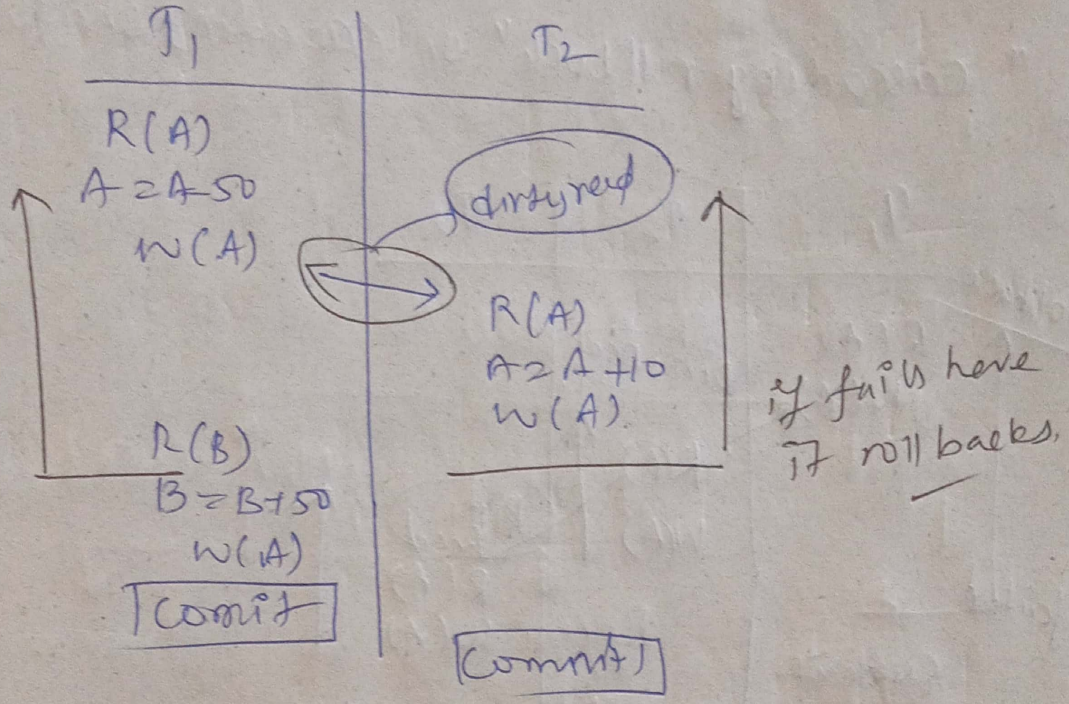
Commit :- Whatever changes are done till now they have to be permanent and we are not supposed to change them.

Recoverable Schedule

If a transaction T_j Read the data item that was previously written by transaction T_i then the commit operation of Transaction T_i must appear before the commit operation of T_j otherwise irrecoverable



When you are performing the "Dirty Read", just wait for the other one to commit the only you commit.

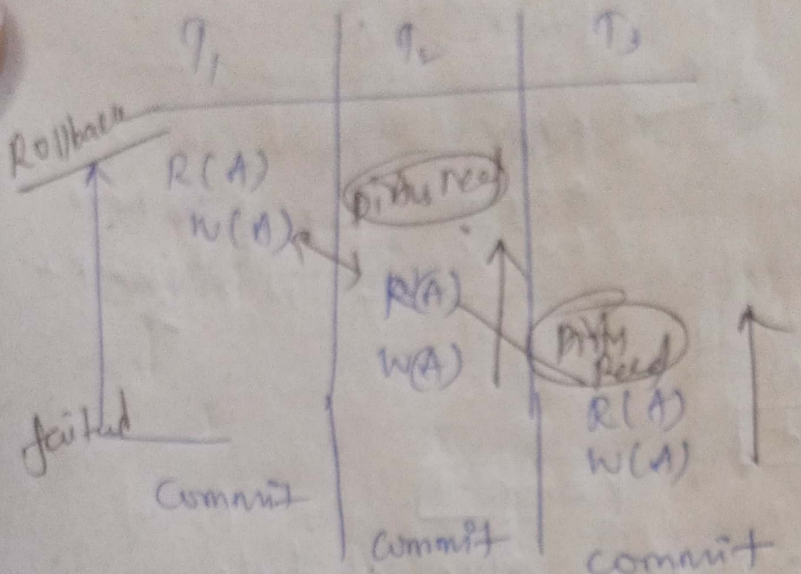


This is recoverable schedule

If other one rollback then you to have a chance to rollback and that is called as cascading rollback. If you have that chance then that is called Recoverable Schedule.

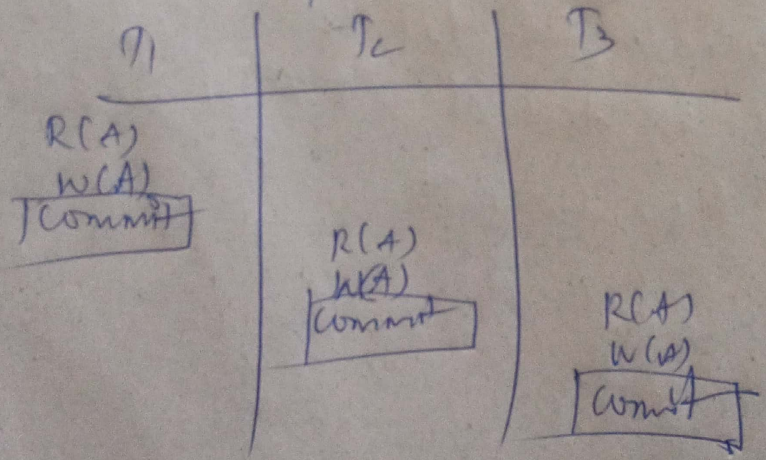
Cascading Schedule

If one transaction failure causes multiple transaction to roll back, it is called "cascading roll back" or cascading schedule



Cascadeless Schedule

A cascadeless schedule is one, where each pair of transactions T_i and T_j such that T_j reads data item is written by T_i. Then the commit operation of T_i should appear before the read operation of T_j



Note

If a Transaction is cascadeless then it is already recoverable but ~~ee~~ every recoverable schedule is not cascadeless.

Strict schedule

A schedule is strict if a value written by a transaction cannot be read (or) over written by other transactions until the transaction either commit (or) abort

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

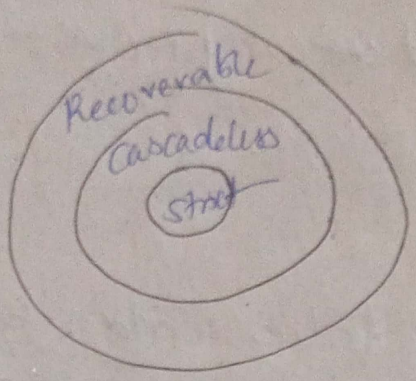
you can perform all operations after commiting T_1 Schedule

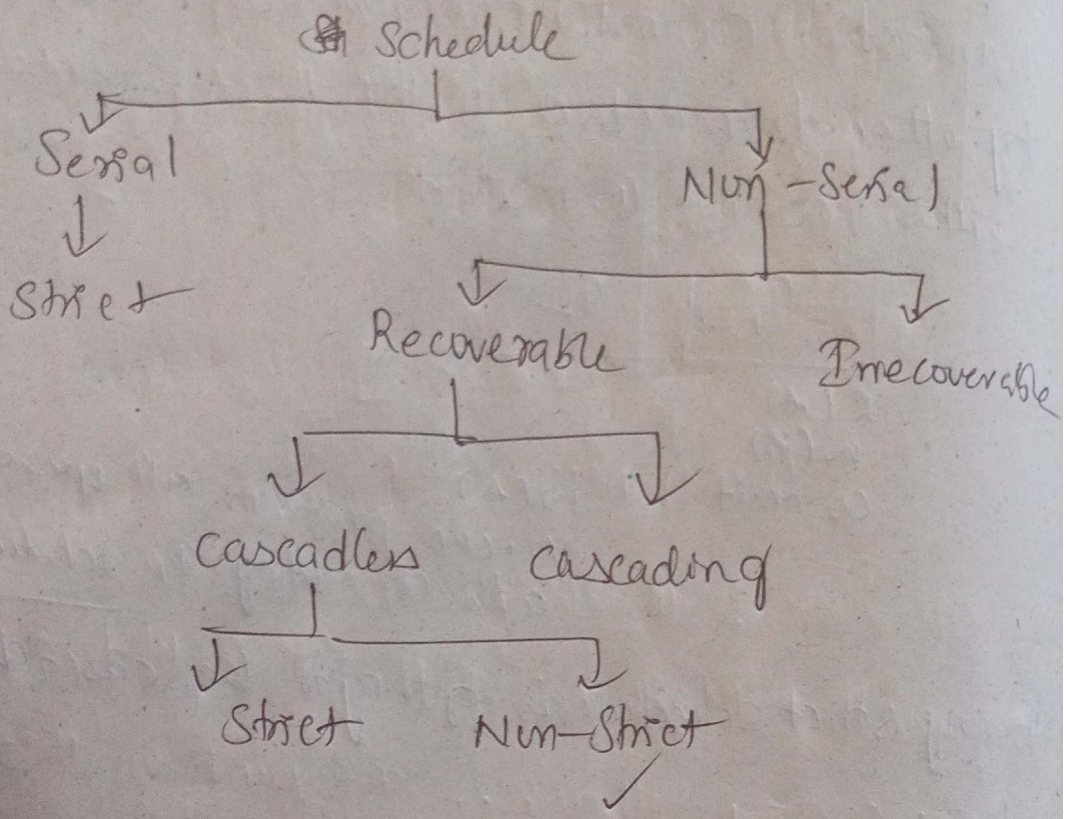
Every strict schedule is going to be cascadeless.

Summary

Recoverable		cascadeless		Strict	
T_1	T_2	T_1	T_2	T_1	T_2
commit		commit		commit	
	commit		R		R
			commit		W
					commit

Summary


Recoverable
Cascadeless
Strict


Schedule
Serial
Non-Serial
Strict
Recoverable
Irrecoverable
Cascadeless
Cascading
Strict
Non-Strict

Serializability

A Transaction ~~the~~ Schedule is serializable if its outcome is equal to the outcome of its transaction executed serially.

→ That means sequentially without overlapping in time.

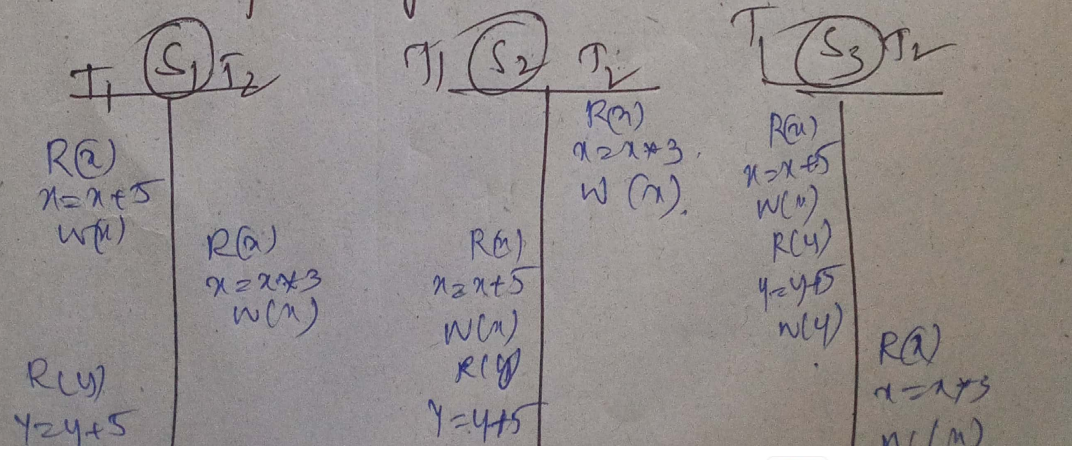
Types of Serializability

1) Result Equivalent
2) Conflict serializability
3) View Serializability

1) Result Equivalent Schedule

Let the two schedule S_1 and S_2 are said to be equivalent schedules if they produce the same final database state

Ex is the following schedule are result equivalent for the initial values of x and y if (x, y) is (2, 5) respectively.



after calculation.

$S_1 = (\overset{x}{21}, \overset{y}{10})$

$S_2 = (4, 10)$

$S_3 = (21, 10)$

* S_1 S_2 are result equivalent schedules
* S_2 is not result equivalent to S_3 & S_1
* Therefore according to the result S_1 is result equivalent to serializable Schedule and it is safe to execute.

* Conflict Serializability *

Conflict equivalent ✓