Transaction

- Properties and states
- Concurrent execution
- Serializability
- concurrency control
  - Lock-Based protocols
  - 2 phase locking protocol
  - Graph based protocol
  - Time stamp based protocols
  - Deadlock handling.

Transaction :

- set of logical work done on data of database is called **transaction** :
- Work can be - inserting
  - updating
  - deleting

  } data from current db.
- Three steps in DB transaction
  - Read data
  - Write data
  - Commit

Transaction Property : maintains consistency in db before and after transaction.

ACID

- **Atomicity** - either all successful or none
- **Consistency** - Brining the db from one to another consistent state.
- **Isolation** - Ensures that transaction is isolated from other transaction.
- **Durability** - means once a transaction has been committed, it will remain so.

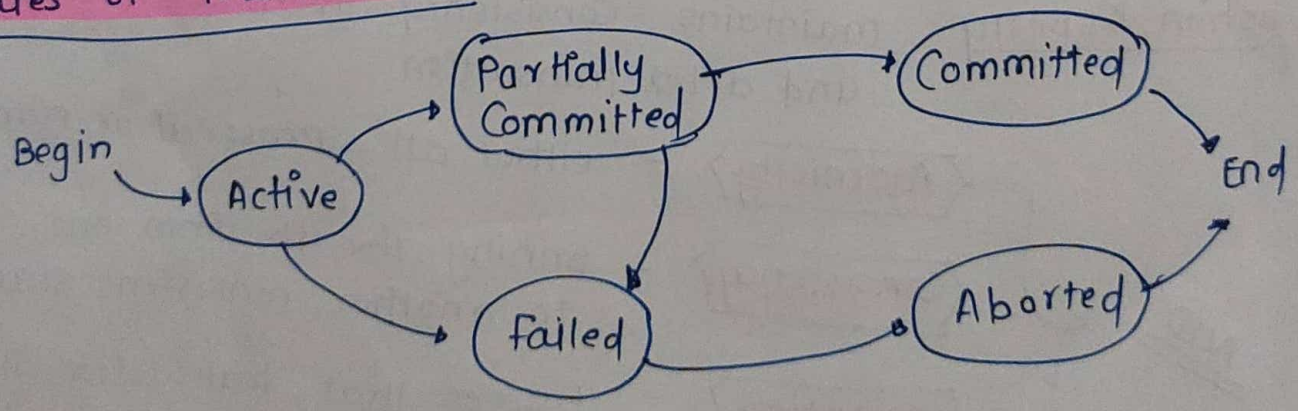
Atomicity :- all operations of transaction take place at once if not, then transaction is aborted.
- transaction can't occur partially.
- Involves
  ① abort : If abort, then all changes made are not visible.
  ② Commit : If commits, then all changes made are visible.

Consistency :- DB should be consistent before & after transaction.
- Execution of transaction will leave db in either its prior stable state or a new stable state.

Isolation : - data used at the time of execution of transaction can't be used by 2nd transaction until first one completed.

Durability : - states, transaction made the permanent changes.
- The changes can't be lost.

States of transactions :



Active State : - First state
- In this state, transaction is being executed

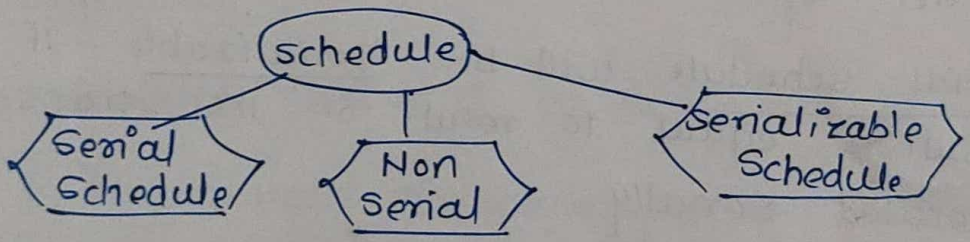
Partially committed : - transaction executes its final operation, but the data is still not saved to db.

Committed : if all operations of transaction are executed successfully.

Failed state : If any of checks made by database recovery system fails, then transaction is in failed state.

Aborted :
- when transaction fails db recovery system will make sure that the db is in previous consistent state.
- If not then it will abort or roll back transaction to bring the db into consistent state.
- After abort :
  i) Restart the transaction.
  ii) kill the transaction.

Schedule :
- Series of operation from one transaction to another is known as Schedule.



(1) Serial schedule :
- Here one transaction is executed completely before starting another transaction starts.
- One schedule is followed by other.

T_1	T_2
Read(x)	
write(x)	
read(y)	
write(y)	
	read(x)
	read(y)
	write(y)

T_1 followed by T_2

(2) Non-serial Schedule :

- If interleaving of operation is allowed, then there will be non-serial schedule.

eg

T_1	T_2
redd(A)	
	read(A)
write(A)	
read(B)	
	write(A)
write(B)	

chapter VI
महत्वपूर्ण

(3) Serializable Schedule :

- Used to find non-serial Schedules that allow transaction to execute concurrently without interfering with one another
- Identifies which schedules are correct when executions of transaction have interleaving their operations.

* Non-serial schedule will be Serializable if its result is equal to result of its transactions executed serially.

Serializability :

- db consistency is preserved by serial execution of set of transaction.
- Schedule is serializable if it is ~~equivalent~~ equivalent to serial schedule.

(1) Conflict serializability :-

Schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into serial schedule.

Conflicting actions

i) → If both action belongs to separate transaction.

ii) → They have same data item.

iii) → Contain at least one write operation/ action

Conflict equivalent :

Two schedules are said to be conflict equivalent iff

1. They contain the same set of transaction
2. If each pair of conflict operations are ordered in same way.

* Eg. Non serial schedule

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

S_1 →

Serial schedule

T_1	T_2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

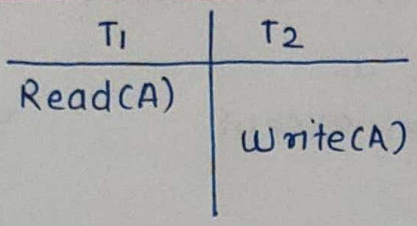
∴ S_1 is conflict equivalent / Serializable

② View serializability :

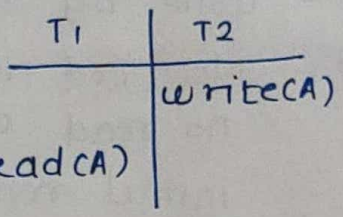
- It is view serializable, if it is view equivalent to serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.

conditions for view equivalent :

1] Initial Read :



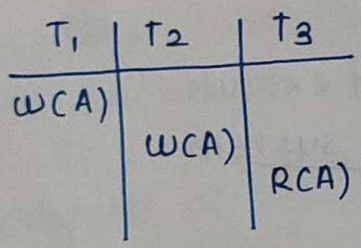
S1



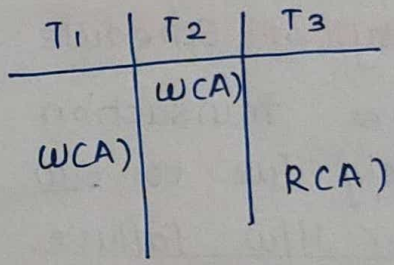
S2

- Initial read of both schedules must be the same.

2] Update Read :



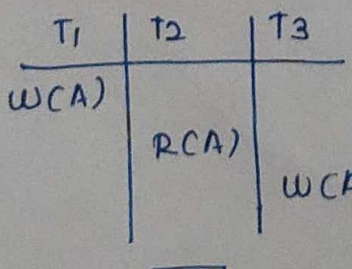
S1



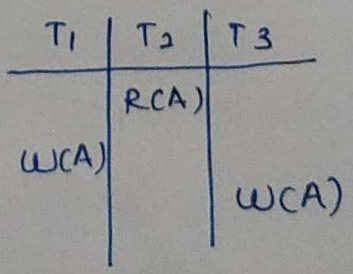
S2

∴ it is not view equivalent because in S1 T3 is reading A updated by T2
and in S2 T3 is reading A updated by T1

3] final write :



S1



S2

∴ Here final write must be done by same transactio

Example

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

S_1

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

S_2

① Initial Read - done by T_1 in both schedules ✓

② Update Read - No need to check as no read operation except initial read ✓

③ Final write - final write done by T_3 in both schedules ✓

∴ as three conditions statisfied S_1 and S_2 are view equivalent

→ Recoverability of Schedule:

- Sometimes transaction may not execute completely due to s/w issue, system crash or H/w failure.
- In that case, failed transaction has to be rollback.
- some other transaction may also have used value produced by failed transaction. So rollback those transactions too.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	Commit
Failed	
Commit	

Here,
- T_1 reads & writes the value of A and that value is read and write by T_2.
- T_2 commits but, T_1 fails lateron.
- due to fallure we have to rollback T1 and T2 (But T_2 can't be rollback as it is committed already)

∴ It is irrecoverable schedule.

Irrecoverable Schedule

→ Schedule will be irrecoverable if T_j reads updated value of T_i. And T_j committed before T_i commit.

Recoverable with cascading rollback:

→ schedule will be recoverable with cascading roll back if T_j reads the updated value of T_i.
Commit of T_j is delayed till commit of T_i.

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

→ Here, we have to rollback T1.
→ T_2 should be rollback becaz T_2 has read value written by T1.
→ Here roll back of T_2 is possible becaz T_2 is not committed before T_1

∴ It is recoverable with cascade rollback.

Cascadeless recoverable schedule

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

Here A is read and write by T_1 and committed.
Similarly after commit of A T_2 reads and write A

∴ Here cascadeless recoverable schedule

* Concurrency :

- multiple transaction at a time
- Adv : - waiting time is less
  - Response time is high
  - Resource utilization is more
  - Efficiency
- disAdv
  - Inconsistent system
  - Difficult to manage

- Problems on concurrent transaction
  1. Dirty Read problem : (W-R problem)
  2. Read - Write conflict : (phantom Read problem)
  3. Write - Write problem : (Blind Write)

* write - Read problem (dirty Read)

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

i.e transaction reads the data written by other transaction before committing.

* Read - Write problem

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

Transaction T2 is writing data which is previously read by T1.

* Write - Write problem (Blind write)

T_1	T_2
W(A)	
	W(B)
W(B)	
comm.	
	W(A)
	Commit

T2 writes data which is already written by T1.
∴ T2 overwrites the data written by T1.

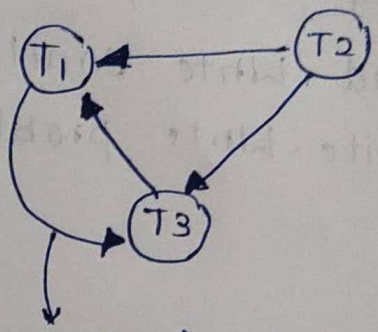
Conflict serializable by precedence graph

- check conflict pairs in other transactions and draw edges from that transaction to current
- If cycle formed then ~~not~~ seriable not possible
  If no cycle —11— Seriable possible.

↳ Now to print order of execution
→ Start with vertex with no incoming edge

EX 1

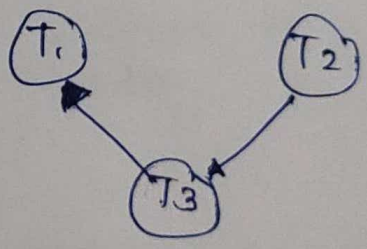
T_1	T_2	T_3
R(x)		
		R(z)
		W(z)
	R(Y)	
R(Y)		
	W(Y)	
		W(x)
	W(z)	
W(x)		



cycle is formed
∴ can't be serializable

EX 2

T_1	T_2	T_3
R(x)		
	R(Y)	
		R(Y)
	W(Y)	
W(x)		
		W(x)
	R(y)	
	W(x)	



Here no cycle
∴ can be serializable.

→ Order
$T_2 - T_3 - T_1$

∴ solution

T_1	T_2	T_3
	R(Y)	
	W(Y)	
	R(Y)	
	W(x)	
		R(Y)
		W(x)
R(x)		
W(x)		