UNIT-4

DATE: __/__/____
PAGE _____

Transaction Processing System and Concurrency Control

Transaction

It is a logical unit of program execution (set of operations) of a database item.

★ Every transaction in case of failure ~~can be~~ is restarted but not resumed.

Begin transaction
{ set of read / write operations + ALU operations
End transaction
Commit

Basic operations of Transaction

1. Read_item (x)
2. Write_item (x)

Eg: A = 1000 B = 500
Transfer 50 Rs from A to B.

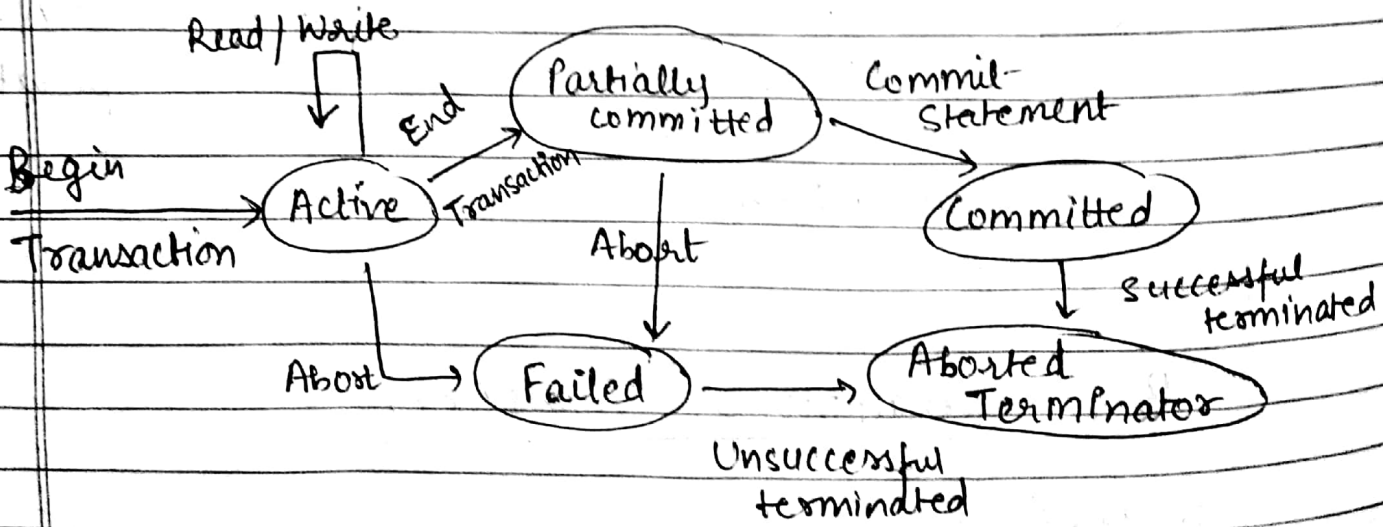
Read_item (A) A → 1000
A = A - 50 A → 950 (ALU)

Write (A)	A → 950
Read_item (B)	B → 500
B = B + 50	B → 550
Write (B)	B → 550
Commit	changes are made in the database

ACID Properties of a Transaction

- Atomicity
- Consistency
- Isolation
- Durability

Transaction States



Schedules (Histories) of Transactions

Suppose S1 (schedule 1) and S2 (schedule 2)

(1) $S1 = r_1(x), w_1(x), r_1(y), w_1(y), r_2(x), w_2(x)$

T_1		T_2	
read (x)	80		
x = x - N	75		
write (x)	75		
read (y)	100		
y = y + N	105		
write (y)	105		
		read (x)	75
		x = x + m	79
		write (x)	79

Time ↓

(T1 followed by T2)

OR

(2) $S2 = r_2(x), w_2(x), r_1(x)\ w_1(x)\ r_1(y)\ w_2(y)$

T_2		T_1	
read (x)	80		
x = x + m	84		
write (x)	84		
		read (x)	84
		x = x - N	79
		write (x)	79
		read (y)	100
		y = y + N	105
		Write (y)	105

Time ↓

(T2 followed by T1)

NOTE: The above two schedules: S1 and S2 are serial schedules.

DATE: __/__/____
PAGE _____

(3) S3: $r_1(x)$ $r_2(x)$ $w_1(x)$ $r_1(y)$ $w_2(x)$ $w_1(y)$

	T_1		T_2	
	read (x)	80		
	$x = x - N$	75		
			read (x)	80
			$x = x + M$	84
Time	write (x)	84		
	read (y)	100		
			write (x)	84
	$y = y + N$	105		
↓	write (y)	105		

(4) S4: $r_1(x)$, $w_1(x)$, $r_2(x)$, $w_2(x)$, $r_1(y)$, $w_1(y)$

	T_1		T_2	
	read (x)	80		
	$x = x - N$	75		
	write (x)	75		
Time			read (x)	75
			$x = x + M$	79
			write (x)	79
	read (y)	100		
↓	$y = y + N$	105		
	write (y)	105		

NOTE: S3 and S4 are non-~~schedule~~ serial schedules. Execution is done in an interleaved way (in case of concurrent execution of transactions)

GOOD WRITE

* when same database elements is accessed by two other / different transactions, then anamolies occur.

Eg: S3 in which x is accessed by T1 and T2.

Q. Why Concurrency control is needed?

There are three types of problems:

① Lost Update Problem

In S3, the final value of x is incorrect because T2 reads the value of x before T1 changes it in the database. Hence, updated value from T1 is lost.

② Dirty Read Problem (Temporary Update)

If in S4, transaction T1 fails and must change the value of x back to its old value; meanwhile T2 has read the temporary incorrect value of x.

S4	T1		T2	
	read(x)	80		
	x = x - N	75		
	write(x)	75		
time			read(x)	75
			x = x + M	79
failure →			write(x)	79
	read(y)			
	y = y + N			
GOOD WRITE	write(y)			

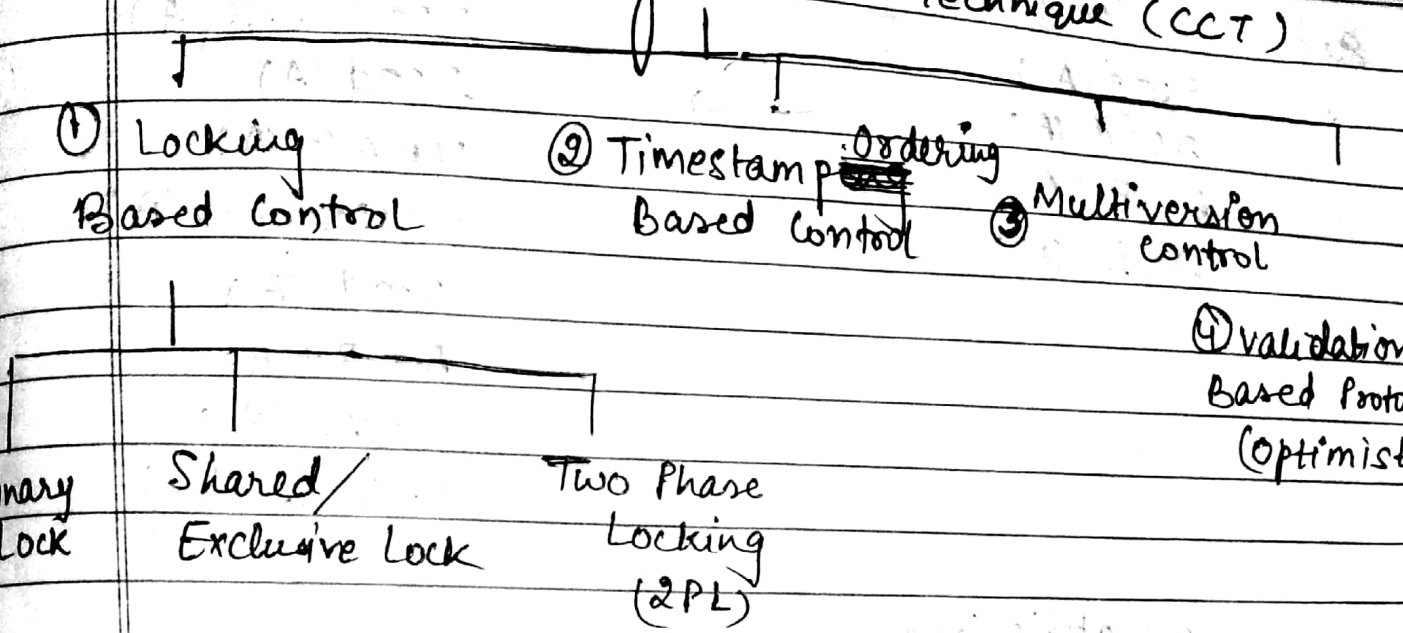
(3) Incorrect Summary Problem.

T_1	T_2
	sum = 0;
	read (A);
	sum = sum + A;
read (x);	
x = x - N;	
write (x);	
	read (x);
	sum = sum + x;
	read (y);
read (y);	
y = y + N;	
write (y);	
	sum = sum + y;

T2 reads the value of x after N has been subtracted from it but it reads the value of y before N has been added to it. Hence, a wrong summary is the result (off by N)

DATE: __/__/____
PAGE _____

Concurrency Control Technique (CCT)



Binary Lock

lock(A) → Here, A is any data item used in a Transaction T

unlock(A) → release A so that the other transactions can take A.

Shared / Exclusive Lock

Shared - read only (also called share-locked)
Exclusive - read and write both (also called exclusive-locked)

3 operations :-

1. read-lock(A) → shared-lock
2. write-lock(A) → exclusive-lock
3. unlock(A) → Both shared lock and exclusive lock

Eg:

T_1

read (A)
read (B)
B = B + A
Write (B)

→

read-lock (A)
read (A)
unlock (A)
write-lock (B)
read (B)
B = B + A
write (B)
unlock (B)

Compatible Graph.

T_1 \ T_2	S	X
S	True	False
X	False	False

Q1 T1 : Account X transfers Rs 50 to account Y

	T_1	T_i (Lock S/X)
200	read (x)	write lock (x)
150	x = x - 50	read (x)
150	write (x)	x = x - 50
100	read (y)	write (x)
150	y = y + 50	unlock (x)
150	write (y)	write-lock (y)
		read (y)
		y = y + 50
		write (y)
		unlock (y)

X = 150
Y = 150
Sum = 300

GOOD WRITE

Serial Order $T_1 \rightarrow T_2$
OR
$T_2 \rightarrow T_1$

DATE: __/__/____
PAGE _____

Q2 T2: To simply add x and y and display their sum in sum variable

	T2		T2'	
150	read (x)		read_lock (x)	
150	read (y)		read (x)	200
300	sum = x + y	⟶	unlock (x)	
300	display (sum)		read_lock (y)	
			read (y)	100
			unlock (y)	
			sum = x + y	300
			display (sum)	300

x = 200
y = 100
sum = 300

* Interleaved transactions next page →

Two Phase Locking

The two phases are :-

1. Growing / Expanding phase -: new locks on the items can be acquired but none can be released
2. Shrinking phase -: the existing locks can be released but no new locks can be acquired

Eg:

read (A)	read_lock (A)
read (B)	read (A)
write (B)	write_lock (B)
read (C)	read (B)
write (C)	write (B)
	write_lock (C)
	read (C)
	write (C)
	unlock (C)
	unlock (A)
	unlock (B)

Q1 and Q2.

DATE: __/__/____
PAGE _____

Interleaved way.

T_1		T_2	
write_lock (x)	200		
read (x)	200		
x = x − 50	150		
write (x)	150		
unlock (x)			
		read_lock (y)	100
		read (y)	100
		read_lock (x)	150
		read (x)	150
		unlock (x)	
		Sum = x + y	250
		display (sum)	250
write_lock (y)	100		
read (y)	100		
y = y + 50	150		
write (y)	150		
unlock (y)			

x = 150
y = 150
sum = 250

Here, incorrect sum is calculated because T_1 unlocked data from x too early, as a result T2 shows an inconsistent state.

DATE: __/__/____
PAGE _____

Rules for time stamp protocols

1) If a transaction Ti issues read operation (a,b,c variables or data items)
   a) If TS(Ti) >= WTS(Q), the read operation is executed and RTS(Q) is set to max (RTS(Q), TS(Ti)).
   b) Otherwise
      if TS(Ti) < WTS(Q), the read operation is rejected and transaction Ti is restarted with a new time stamp value.

2) If a transaction Ti issues write operation
   a) If Ts(Ti) >= WTS(Q), the write operation is executed and WTS(Q) is set to TS(Ti).
   b) If TS(Ti) < RTS(Q), the write operation is rejected and transaction Ti is restarted with new time stamp value.

Example

Step	T1 (200)	T2 (150)	T3 (175)
1	Read(b)		
2		Read(a)	
3			Read(c)
4	Write(b)		
5	Write(a)		
6		Write(c)	
7			Write(a)

a	b	c
RT(a) = 0	RT(b) = 0	RT(c) = 0
WT(a) = 0	WT(b) = 0	WT(c) = 0

RT(b) = 200

RT(a) = 150

RT(c) = 175

WT(b) = 200

WT(a) = 200

WT(c) = 150

TS(T1) < WTS(b) (False)

Step 1 : TS(T1) = 200 >= 0 = (Yes)

RTS(b) = max (0, 200) = 200

Step 2 : TS(T2) = 150 >= 0 (Yes)

Read operation permit

RTS(a) = max (0, 150) = 150

Step 3 : TS(T3) = 175 >= 0 (Yes)

Read operation permit

RTS(c) = 175

Step 4 :

TS(T1) ~~RTS~~ >= WTS(b)

200 >= 0 (True)

Write operation permits

WTS(b) = TS(T1) = 200

Step 5 : TS(T1) >= WTS(a)

200 >= 0 (True)

Write operation permit

WTS(a) = 200

Step 6 : TS(T2) >= WTS(c)

150 >= 0 (True)

write operation permits

WTS(C) = 150

Step 7 : TS(T3) < WTS(a) {175 < 200}

Write operation is rejected

GOOD WRITE

DATE: __/__/____
PAGE _____

Validation or optimistic concurrency Control Techniques.

→ It assumes that conflicts between the transactions are rare.
→ This method does not require locking and hence no deadlock occurs.
→ This method checks for ~~conflicts~~ ~~commit~~ conflicts just before commit.

Basic terminologies used

Read set (T_j) : set of items read by T_j (transaction j)
Write set (T_j) : set of items written by T_j

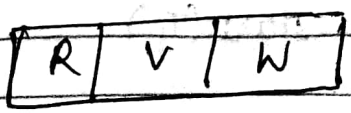
Start (T_j)] 2 local variables of T_j containing
Finij (T_j)] timestamp value that are needed for validation phase

Start(T_j) → when transaction T_j started
Finish(T_j) → when transaction T_j finished its read phase

TS(T_j) : Unique timestamp value assigned to T_j only after write phase if the validation succeeds.

Three phases of validation / optimistic CC Technique

1. Read phase (R)
2. Validation phase (V)
3. Write phase (W)



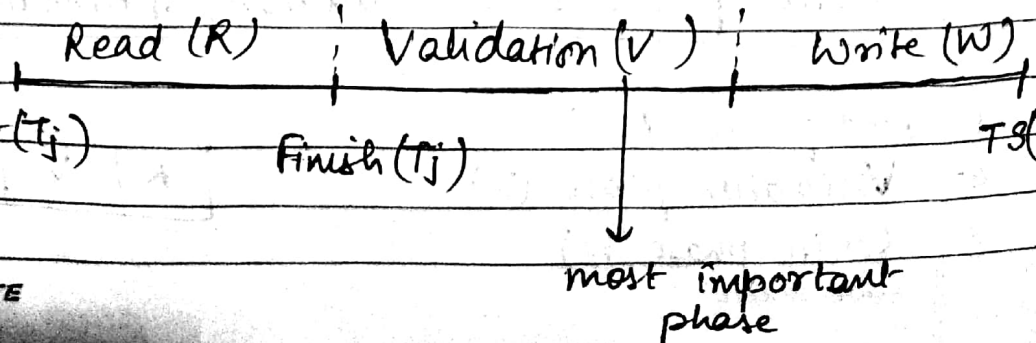
1. Read Phase → In this phase, a transaction reads the data item from the database, performs computation and writes all results into local or private variables i.e writes locally. These variables are not written into the database.

2. Validation Phase (to check consistency) → This phase ensures that for those data items that were read and modified (updated) in the read phase will not cause the database to become inconsistent.

If any inconsistency occurs, then transaction will be rolled back and restarted with new timestamp value.

3. Write phase → If the validation phase succeeds, the transaction updates (writes) are applied permanently to the database. Otherwise, transactions are restarted with new timestamp value.

Structure of transaction T_j



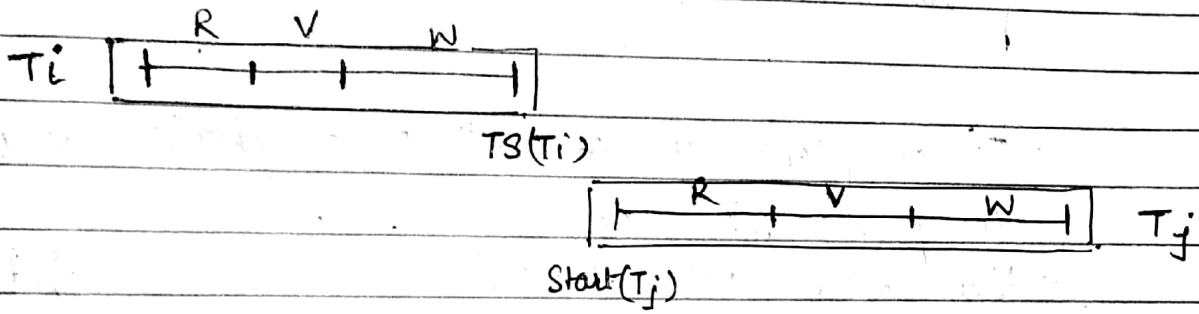
DATE: __/__/____
PAGE _____

⇒ Validation phase can be completed successfully if atleast one of the following 3 conditions is satisfied:

Condition 1:

$$TS(T_i) < Start(T_j)$$

All the older transactions (T_i) must have completed their write phase before T_j started its read phase.



They must execute in serial order ∵ no interference.

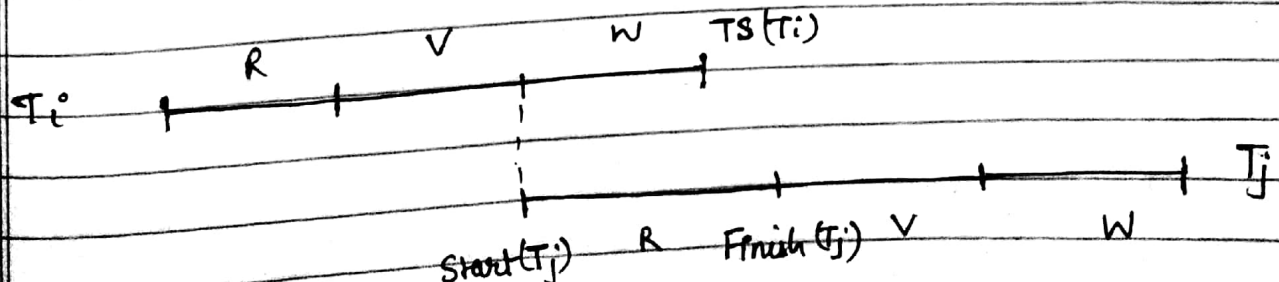
Condition 2:

$$Start(T_j) < TS(T_i) < Finish(T_j)$$

and

$$Write\ set(T_i) \cap Read\ set(T_j) = \phi$$

i.e no common data elements



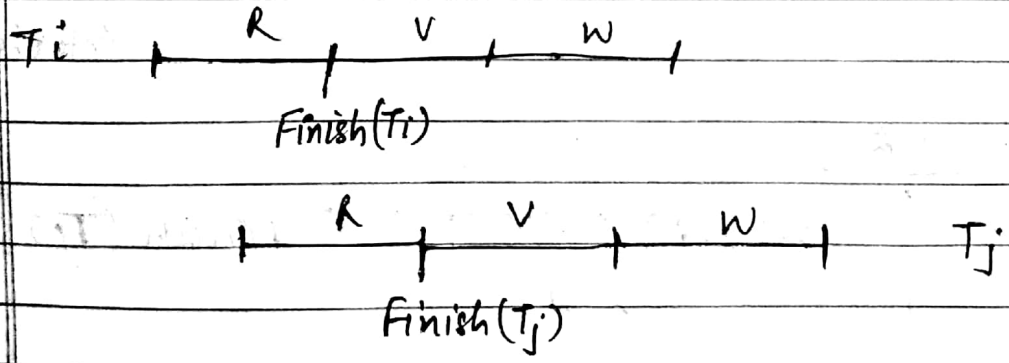
GOOD WRITE

Ti completes its write phase before Tj starts its validation phase and the write set of Ti does not intersect with the readset of Tj.

Condition 3

Finish Ti < Finish (Tj)
and
Write set Ti ∩ (readset (Tj) ∪ writeset (Tj)) = ϕ

→ Ti completes its ~~reading~~ read phase before Tj completes its read phase.
→ The write set of Ti does not intersect with readset or write set of Tj.



DATE: __/__/____
PAGE _____

Backup and Recovery

System Log

Start (T)
Write (T, X, old-value, new-value)
Read (T, X)
Commit (T)
Rollback (T)

Recovery Techniques

1. Deferred Update Recovery. (NO-UNDO / REDO technique)

→ During transaction execution, updates are written to the log file.

→ After transaction reaches the commit-point, the log file is force-written to disk, then updates are recorded in the database.

→ A transaction cannot change the database on disk until it reaches its commit point.

Example : T_1: Ra Rd Wd C
T_2: Rb Wb Rd Wd C

Log File

Start (T_1)
Read (T_1, a)
Read (T_1, d)
Write (T_1, d, old, new)

GOOD WRITE

DATE: __/__/____
PAGE _____

Commit (T1) → Redo
Start (T2)
Read (T2, b)
Write (T2, b, old, new)
System crash.

~~T1 has~~ Since T1 has reached commit,
∴ changes in T1 are written to disk.
However, T2 did not commit, hence its changes were not written to disk. ~~To recover;~~

* UNDO (Backward Recovery) → It is applied to a single operation rather than to whole transaction.
* REDO (Forward Recovery) → operations in a certain transaction must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database.