Lec 11 Normalisation

↳ NF. → DB optimisation.

* Functional dependency (FD) → usually.
  - relationship b/w the primary key attribute of relation to that of other attribute of relation

determinant → dependent

A → B.

A se B determine ho raha

A → C

Table

A	B	C	D
1	2	1	5

i.e agar A ki value 1 th
1 the corresponding 2 aa skta
h by A → B

e.g. emp

emp id | name | dept.

FD:

emp id → emp name

emp id → dept.

* Types of FD

→ Trivial FD

A → B , B is subset of A

e.g. {emp id, name} → emp id

e.g. A → A, A ⊆ A

B → B, B ⊆ B.

→ Non Trival FD.

{Emp ID, name} → {Emp address}

B ⊆ A ✗ with subset.

⇒ NTFD.

A → B, B ⊄ A, A ∩ B = NULL

* Rules of FD (armstrong's axioms)

→ Reflexive.

* X = {a, b, c, d, e}

subset Y = {a, b, c}

⇒ Y is subset X.

⇒ X → Y

→ Augmentation.

X → Y (Given) R(X, Y, Z)

Now, then

XZ → YZ.

→ valid

→ Transitivity.

A → B , B → C

⇒ A → C

→ Armstrong inference rules.

* Derived rules.

• Union: if X → Y & X → Z

then X → YZ

X → XX

⇒ XX → XY

XZ → Z

⇒ XX → YZ

X → XY & XY → YZ

⇒ X → YZ

• Decomposition

if Y → YZ, then X → Y & X → Z

• Pseudotransitivity

if X → Y & WY → Z then

XW → Z

Q R(A B C D E)

Given FD : $A \to B$, $A \to C$, $CD \to E$, $B \to D$, $E \to A$.

Can $CD \to AC$?

$CD \to E$ & $E \to A$ $\Rightarrow$ $CD \to A$ (transitivity)

$C \subseteq CD \Rightarrow CD \to C$ ~~(thr, vat)~~ (reflexive)

So, $CD \to A$ & $CD \to C$ (UNION)

$\Rightarrow CD \to AC$

* Why normalisation.

To avoid redundancy in DB not to store redundant data. as insertion, deletion and updation anomalies arises due to redundant data.

→ Anomalies

- Insertion

id	name	age	Branch_code	Branch name	Branch HoD
1	A	18	1	CS	X
2	B	19	1	CS	X
3	C	18	2	ECE	~~X~~ X
4	D	20	3	ME	Z

If a new student comes but not yet selected branch

5	E	19	let @ NULL	let NULL	let NULL

Phir baad wale null ko update karo ye h. anomalie

ye h ki student ki existence or branch ki existence bh independent h

Now, if university added IT branch bh jb at the moment bh koi student hi nhi h

- Deletion

agar id=4 ko delete krde bs wo pass out hogya bh ye deletion ke karan ME branch hi hatt gyi, as suppose uss Branch mae kewal wahi tha.

Deletion of data results in unintended loss of some other important data.

- updation

left HoD of CS change to Q. bh wallab ab har student ke detail ko update krna padega.

id	name	age	BR. code

BR code	BR. name	HoD.

* In normalisation.

Table ⟶ decomposes into multiple tables.
until SRP is achieved.
↓
Sugle responsibility principle.

* Types of normal forms

→ 1NF

- every relation cell must have atomic value.
- Relation must not have multivalued attribute.

e.g.

emp

id	name	Phone
1	A	88
2	B	12
2	B	99

← Convert 1NF

emp

id	name	Phone
1	A	88
2	B	12, 99

→ 2NF

- Relation must be in 1NF
- There should not be any partial dependency.
  - All non prime attributes must be fully dependent on PK.
  - Non Prime attribute cannot depend on part of PK

R (A B C D)

{AB} → PK : A aur B mila ke PK h

A, B → prime attribute

C, D → Non prime

=> FD : => B → C → Partial dependency.

Ye hona chahiye

AB → C

AB → D

but B → C mai B alone C ko determine kr rha h ✗

$\{A, B\} \rightarrow PK$.

A	B
NULL	1
2	NULL
NULL	NULL
3	4

→ A, B CK saath null nhi hoga as PK ← $\{A, B\}$

R (A B C D)
B → C
ABD
ABC

Suppose B = NULL

NOW $B \rightarrow C$

$NULL \rightarrow C$ ✗ not holds.

2NF Conversion.

R_1 (A B D) R_2 (B C)

PK PK. $B \rightarrow C$

$AB \rightarrow D$

C.K

Student ID	Proj ID	Student name	Project name
S89	P09	Jacob	Geo
S76	P07	AVO	IOT

PK: {Student ID, proj ID}

F.D: Given
Student ID → student name
project ID → project name.

Non prime is being determined by part of PK.

2NF solution.

Student | Student ID | Proj ID | Student name

Project | project ID | projectname

3NF

- Relation must be in 2NF
- No transitivity dependency exists.
  - Non prime attribute should not find a non prime attribute.

R (A B C)

PK {A}

1NF ✓

2NF ✓ B & C fully dependent on PK.

F.D:- $A \rightarrow B$, $B \rightarrow C$ } Transitivity dependency

A	B	C
a	1	x
b	1	x
c	1	x
d	2	y
e	2	y
f	3	z
g	3	z

$A \rightarrow$ Prime

$B \rightarrow C$ (C → Prime)

Non Prime

R (A B C)
A → B
B → C

3NF form

R_1 (A B) R_2 (B C)

A	B
a	1
b	1
c	1
d	2
e	2
f	3
g	3

B	C
1	x
2	y
3	z

for R_1
1NF ✓ atomic value ✓
2NF ✓ full dependency as A → B in R_1 table & B → C in R_2 table ✓ full dependency for this table ✓
3NF ✓ R_1 A → B prime se non prime derive ho rha h ✓ & R_2 B → C ✓ No duplicacy.

BCNF (Boyce-codd normal form)

- Relation must be 3NF
- FD: A → B, A must be super key & CK.
  - we must not derive prime attribute from any prime or non prime attribute.
  - ~~NP → P~~
  - ~~P → P~~

e.g

std-ID	subject	professor
101	Java	PJ
101	CPP	PC
102	Java	PJ2
103	C#	PC#
104	Java	PJ

- one student can enroll in multiple subjects.
- for each subject, a professor is assigned to student.
- multiple prof. can teach single subject.
- one prof. can teach only one subject.

* Fourth Normal form (4NF)

Relation is 4NF if it is in
- BCNF form
- has no multi valued dependency.

i.e for a dependency A, B if for a single value of A, multiple values of B exists, then the relation will be multi valued dependency.

e.g
Vehicle → 3 Phone no.
→ 3 manufid

Ven.	M, E1
Ven	M, E2
"	M, E3
"	M2 E1
"	M2 E3

Split into

Ven	M1
Ven	M2
Ven	M3

Ven	E1
Ven	E2
Ven	E3

* Fifth Normal form (5NF) or project join Normal form

If it is in
- 4NF & not contains any join dependency & joining should be lossless.

- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- Lossless

* Indexes

CREATE [UNIQUE] INDEX indexname ON
tablename (columnname1, ..., columnnamen);

* Lossy Decomposition

table → R(A B C) ← attributes

A	B	C
1	2	1
2	2	2
3	3	2

→ R_1(A B)

A	B
1	2
2	2
3	3

→ R_2(B C)

B	C
2	1
2	2
3	2

R_1 aur R_2 ke bich ek common attribute hona chahiye for decomposition

Ab agar baad me R_1 aur R_2 ka join karna ho to ek common attribute ke base pe karenge.

find the value of C in the table of A = 1.

=> Join i.e Natural join

Natural join = cross product of tables + C.

Select R_2.C from R_2 Natural join R_1 where R_1.A = 1;

A	B	C
1	2	1
1	2	2
2	2	1
2	2	2
3	3	2

After join R' (Original table which but tuples both are)

A jab (1) h
tb C ki value
1 or 2 ho sakti which is wrong.

remove duplication of B

i.e Lossy decomposition

PK: {Std-ID, Subject} as dono koga tb hi professor kaun kya padhaega

FD:

{Std-ID, Subject} → Professor.

Professor → subject (as koi ek prof. ko koi ek subject hi padhana hoga)

Prime

non prime

~~Y~~ ~~Y~~ NP → Prime

So tables not in BCNF

=> BCNF form.

Student

Sid	Pid
101	1
101	2
1	1
1	1

Here P → NP

PK

Prime Prof

Pid	Prof.	Sub
1	P1	Java
2	P2	CPP.
1		

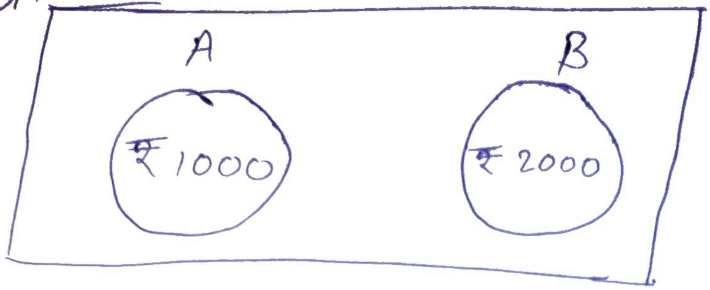
Here P → NP

Here P → NP => BCNF ✓

* Advantages of Normalisation
- Helps to minimise data redundancy.
- Greater overall DB Organisation.
- Data consistency is maintained in DB.

L-12 Acid properties && Transactions

*** Transaction**



Task l → ₹50 from Bank A/C A to A/C B

T_i :
read (A)
A = A − 50;
write (A);) 990 karna
read (B)
B = B + 50
write (B)

6 steps

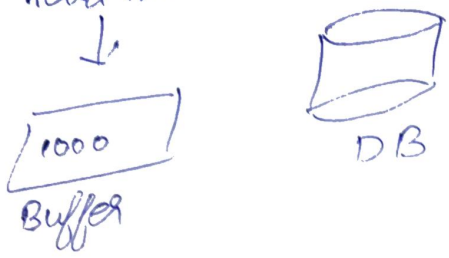
atomic — They are considered to be a single task.

- a unit of work done against the DB in logical sequence.
- It is a logical unit of work that contains one & more SQL statements. The result of all these statements in a transaction either gets completed successfully (all the changes made to the database are permanent) or if at any point any failure happens it gets rollbacked (all the changes being done are undone).

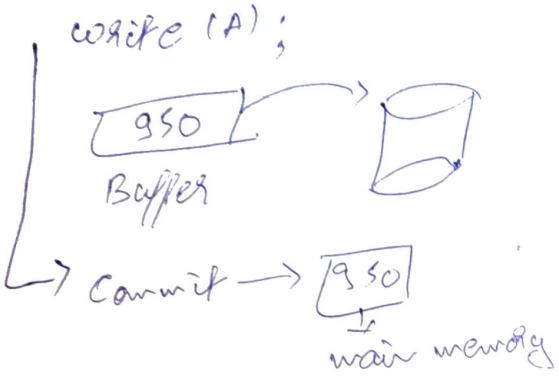
Acid properties

- to ensure integrity of data, we req. that DB system maintain following properties of transaction.

~~Atomicity~~



Ti : read (A)
A := A - 50
write (A);
read (B);
B := B + 50;
write (B);



Real DB
commit karne ke Baad.
950 → DB
DB me data ^

* Atomicity

Either all operations of transactions are reflected properly in DB, & none are.

account A	account B
Read 1000	Read 2000
950 = 1000 - 50	
write (A)	
950	
	Read 2000
	2000 + 50
	write (2050)
	2050

transaction ⟶ successful
⟶ fail ⟶ old state recovery.

DB → maintain ⟶ old state
↓
intermediate state
↓ success ✗
Rollback old state.

* Consistency.
- Integrity constraint must be maintained before and after transaction.
- DB must be consistent after transaction happens.

ye agar DB total ₹3000 h th operation ke baad bhi ₹3000 hi hona chahiye.

* Isolation
Banking system.

T_1, T_2, T_3 - - -

agar hm GPay se payment kare or netbanking se bhi transaction kare th coo dono isolated honge.

T_1 (G pay) t=0 read (A) → 1000
A: A−50 = 950
write (950)

t=10 T_2 (Netbanking) CK saath read krliya
read (A) → 1000
1000 − 50 = 950
B → 50 + 50

th aise B ko double so chla gya so kp possible nhi h.

Jb tk T_1 success/failure nhi hogi .:. T_2 ko start bhi nhi hoga th aise isolation aajaega.

- Even though multiple transactions may execute concurrently, the system guarantees that for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in system.

- Multiple transactions can happen on the system in isolation, without interfering each other.

* Durability

- After transaction completes successfully, the changes it has made to the database persist even of there are system failures.

T_1 —> success

↓

DB updates permanent (persist)

↓

even if there is a system failure after T_1 completes trans.

→ recovery - management component.

read(A)

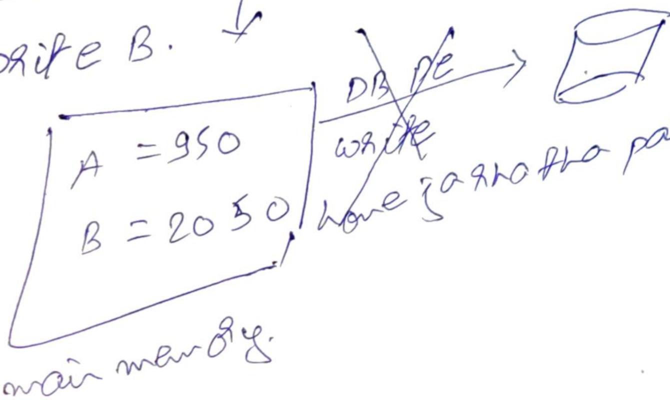
A: A-50 → log(50)

write(A) → log(write A)

Read(B)

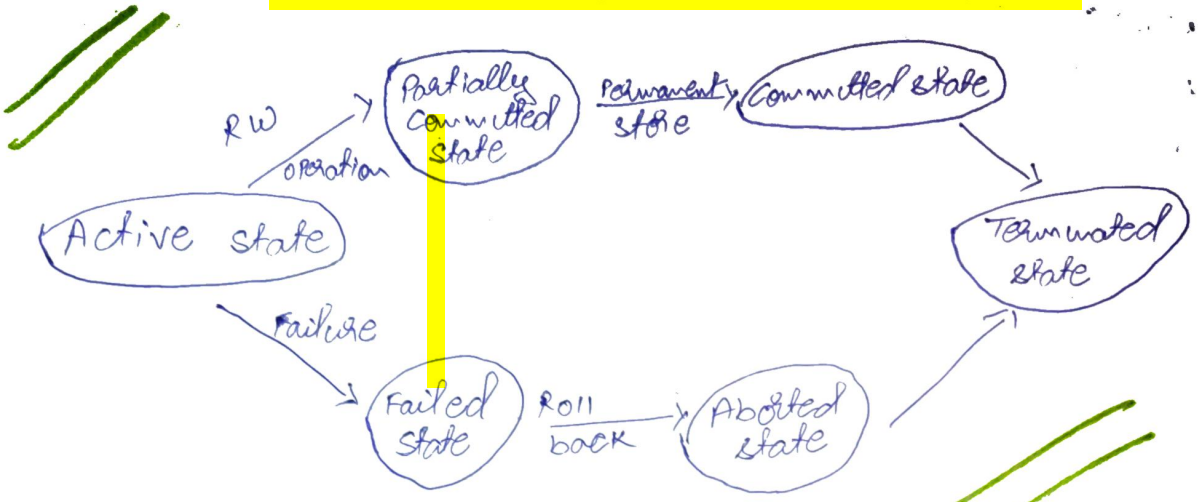
B: = B+50

write B.



hone ja raha tha par issi time failure aa gyo.

* Transaction states.



1. Active state

The very first state of life cycle of transaction, all the R & W operations are being performed. If they execute without any error the T comes to partially committed state. Although if any error occurs then it leads to a failure.

2. Partially committed state.

After transaction is executed the changes are saved in the buffer in the main memory. If the changes made are permanent on the DB the state will transfer to committed state & if there is any failure, the T will go to failed state.

3. Committed state.

When updates are made permanent on DB. Then the T is said to be in committed state. Roll back can't be done from the committed states. New consistent state is achieved at this stage.

4. Failed state

when T is being executed & some failure occurs due to this it is impossible to continue the execution of T.

5. Aborted state

When T reaches the failed state, all the changes made in the buffer are reserved. After that the T roll back completly. T reaches abort state after roll back. DB's state prior to the T is achieved.

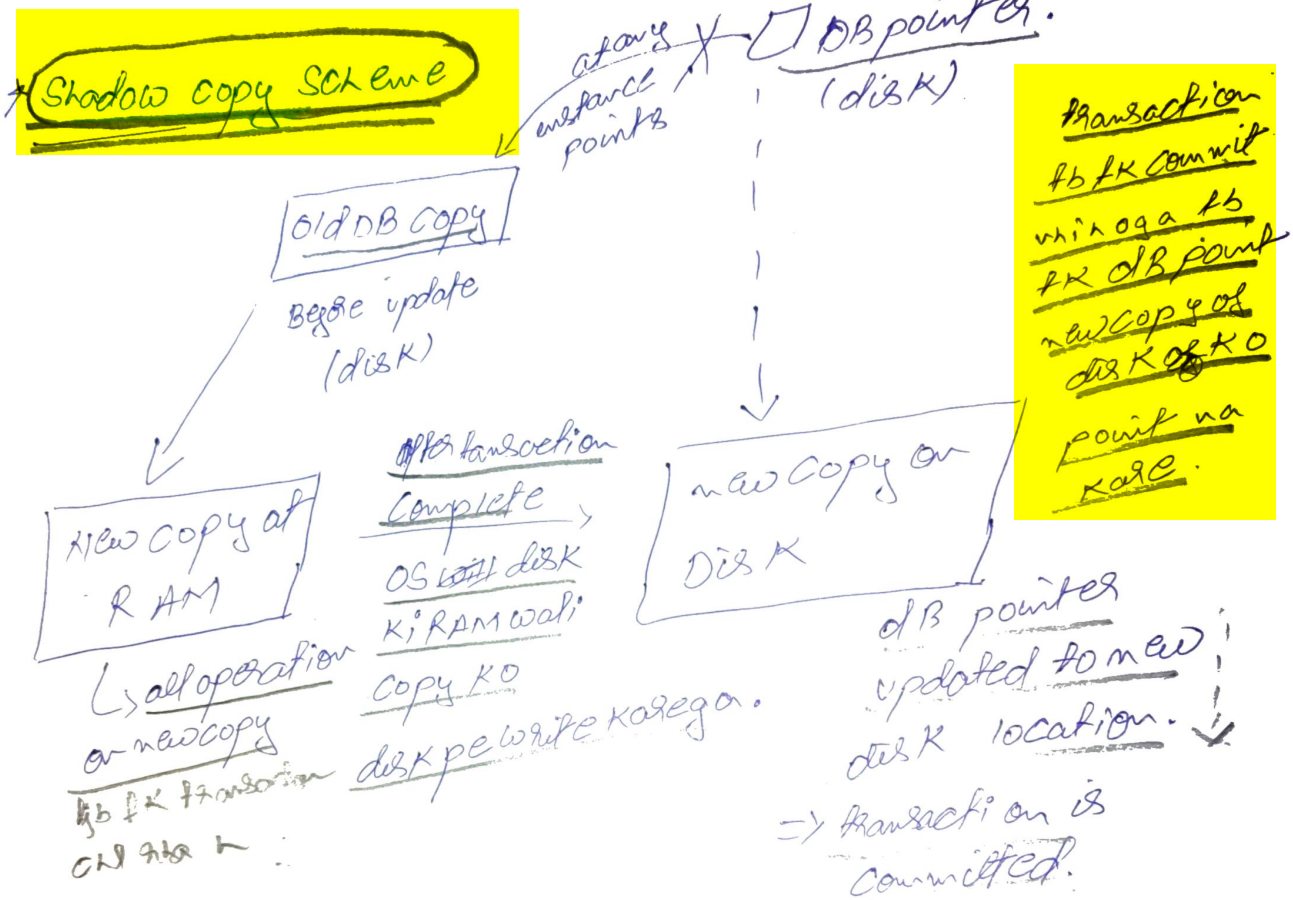
6. Terminated state

A transaction is said to have terminated if has either committed or aborted.

L-13 How to implement atomicity & Durability?

- Recovery Mechanism component of DBMS supports atomicity and durability.

Shadow copy scheme

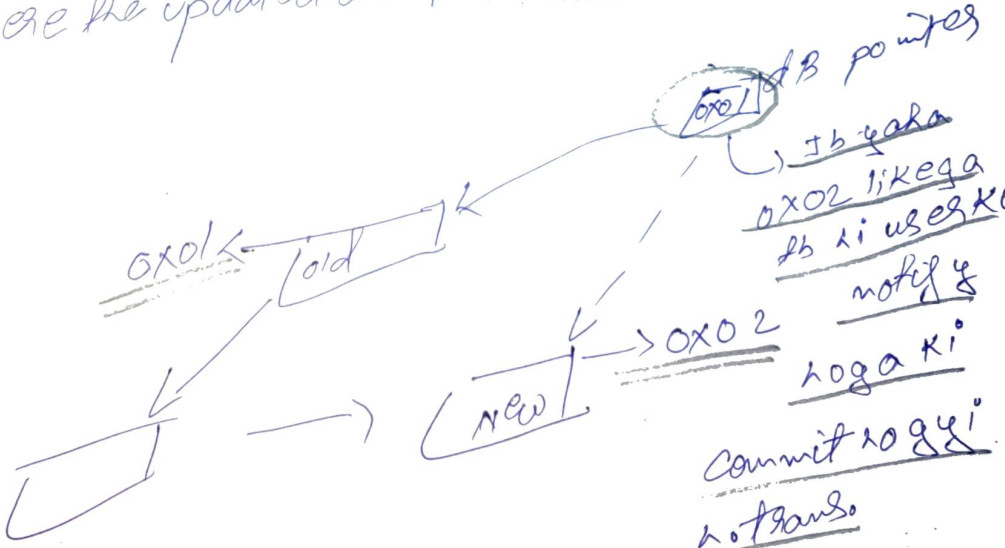


- Based on making copies of DB (aka shadow copies)
- Assumption only one TR ansaction (T) is active at a time.
- A pointer called DB pointer is maintained on the disk; which at any instant points to current copy of DB.

- All further updates are done on new DB copy leaving the original copy (Shadow copy) untouched.

- If at any point the T has to be aborted the system deletes the new copy and old copy is not affected.

- If T success, it is committed as
  - OS will make sure all the pages of the new copy of DB written on the disk.
  - DB system updates the db pointer to point to the new copy of DB.
    - New copy is now the current of DB.
    - The old copy is deleted.
  - The T is said to have been committed at the point where the updated db-pointer is written to disk.

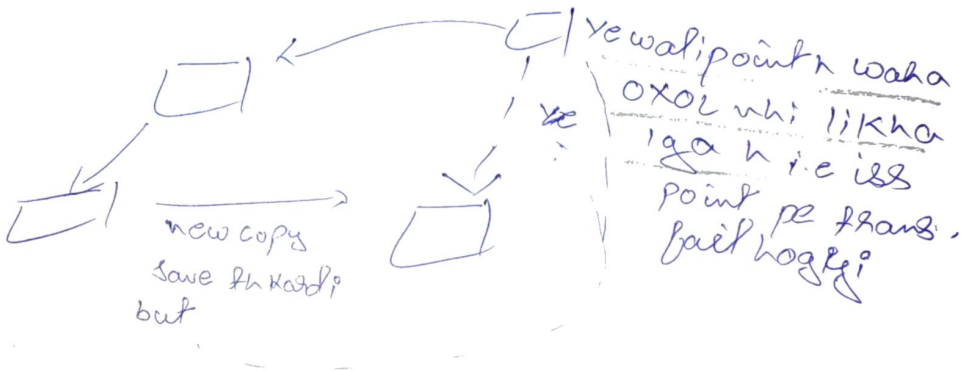


- Atomicity

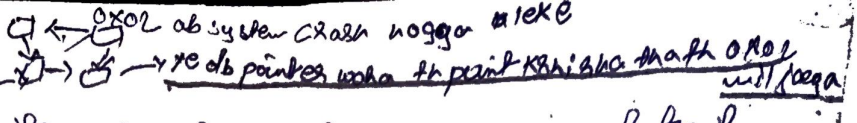
- If T fails at any time before db pointer is updated, the old content of DB are not affected.
- T abort can be done by just deleting new copy of DB.
- Hence, either all updates are reflected or none.

- Durability:

- Suppose, system fails are at any time before the updated dbpointer is written to disk.



- Then the system restarts, it will read db pointer & will thus, see the orginal content of DB and none of the effects of T will be visible.
- Trans. is assumed to be successful only when db pointer is updated.



Now

Suppose if system fails after db pointer has been updated. Before that all the pages of new copy were written to disk. Hence when system restarts it will read new DB copy.

trans ko commit bol diya, bolne ke baad system restart hua th durability maintain krne ke liye ki db mae changes hue h persist kare.

ab yaha persist ho rhaa.

- agar 0x01 ⟶ 0x02 iske bich mae fail hogya th

The implementation is dependent on write to the db pointer being atomic.

Luckily, disk system provide atomic updates to entire block or at least a disk sector. ↳ update hoga ya nhi hoga

So, we make sure db pointer lies entirely in a single sector. by storing db pointer at the beginning of a block.

- Inefficient, as entire DB is copied for every Trans.

Method

* Log-based Recovery systems.

→ guarantee atomicity.

- The log is a sequence of records. Log of each trans is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed of the data base, then it will be recorded in the log.

A	B	Logs (stable storage)
To : read (A)		< To start > → variable
A: A − 50.		< To, A, 950 > → new value
write (A)		< To, B, 2050 >
B: B + 50		< To commit >
write (B).		

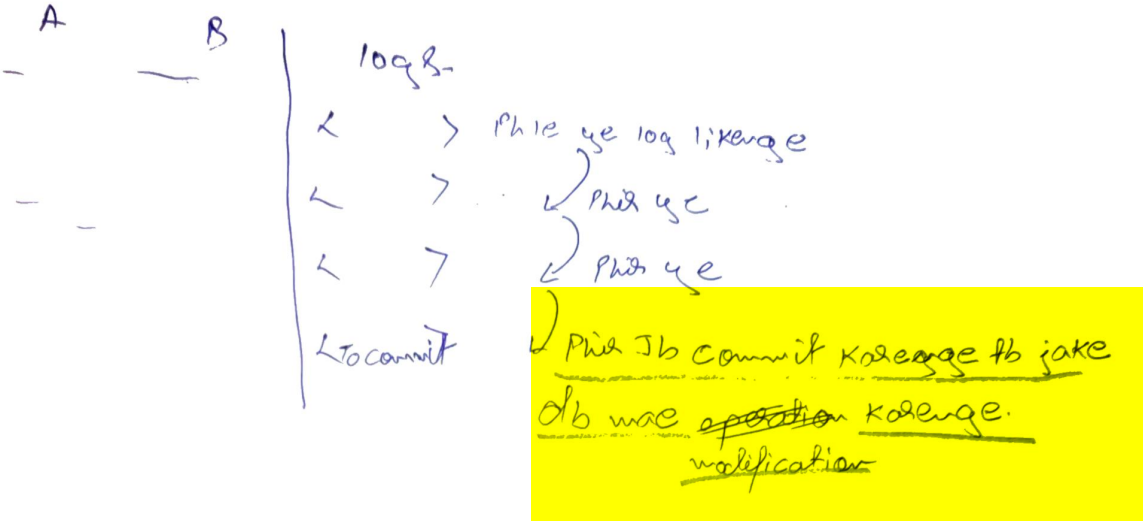
- but the process of storing the logs should be done before the actual trans is applied to the db.

logs phle likenge phir baad me operation likenge.

// Stable storage: is a classification of computer data storage technology that guarantees atomicity for any given write operation and allows software to be written that is robust against some hardware and power failures.

Deferred DB modification.

- Ensuring atomicity by recording all DB modifications in the log but deferring the execution of all the write operations until the final action of the T has been executed.



- Log-info is used to execute deferred writes when T is completed.
- If system crashed before the Trans completes, & if T is aborted, the info in the logs are ignored.
- If Trans. completes, the records associated to it in the log file are used in executing the deferred writes.
- If failure occur while this updating is taking place, we perform redo → wapas se log ko read karenge or same opn. fir se karenge

Immediate DB modification

- DB modi to be output to the DB while the T is still in active state.
- DB modi written by active T are called uncommitted modifications.

Operation ko log karenge pr phir just baad hi usse write kar denge.

A | B

<T0 start> → old values
<T0, A, 1000, 950> → new values
<T0, B, 2000, 2050>
<T0 commit>

- For durability

In the event crash or T failure system uses old value field of the log records to restore modified values.

<T0 start>
<T0, A, 1000, 950>

1000
950

< >

trans fail/abort for atomicity, rollback pe phir 950 ki old value log se wapas change i.e 1000

1000

- update takes place only after log records in a stable storage

- Failure handling.

  System failure before Trans completes or if Trans aborted, then old value field is used to undo the Trans.

  If Trans completes and system crashes then new value field is used to redo Trans having commit logs in the logs.

  < >
  < >
  < >
  <T0 commit>

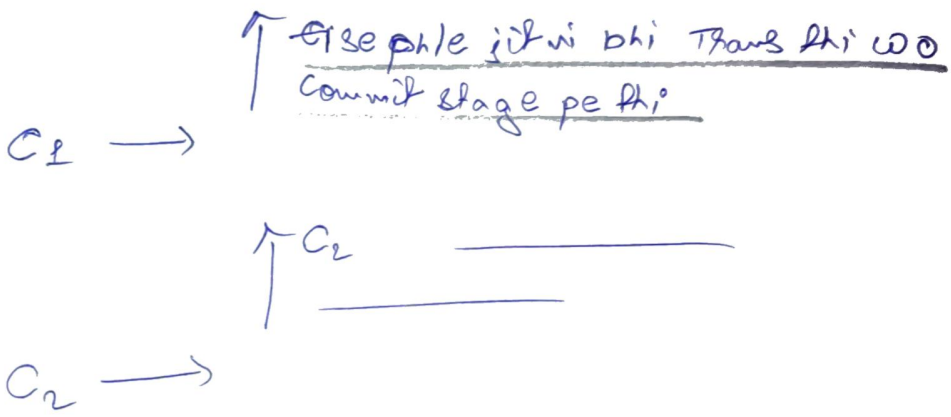
log ke andar likh chuke h Trans complete hogyi h. DB me update kar rhe the tbhi db fail hogya th DB ke andar actual commit nhi hopaya h abhi. th durability ke liye

Fir jb bhi system wapas se restart se hoke recover hoga wo aake dhkega ki yeh <T0 commit> hui thi ki nhi, yaha th likha h hogyi thi th.. Trans ko redo karenge

// check point.

If there are 1000s of logs then stable storage will keep getting full i.e not possible in real time as cost etc increases

So use check point log based recovery method.



Agar trans fail hoti h to wapas check point pe recover ho jate h