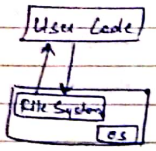
DBMS

Data Base Management System

DBMS - Software System that provides you a quick way to access & modify the data

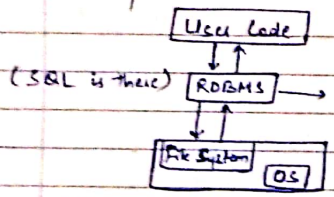
3 Types of Evolution of DBMS :-

1) File Based - Bohot Phle use krte the isse



In this Programmer used to write seperate codes for accessing File System / DB

2) Relational DBMS - Acts as an Interface b/w user & DB. It is a DBMS where data is stored in the form of Tables & there are multiple Table with Schema.



Here No need to write code again & again RDBMS will manage all these things

→ Oracle, MySQL, SQL Server

This method is used a lot & everywhere But it has some problems that they Require a proper structure to store the data & we need that much Space & there is Scalability issues there, as we need to answer & update in the same DB for the Query so if we need to copy that DB than the write operation becomes very costly

3) No SQL (Non Relational DB) - These DB which do not have proper structure in them & they could be stored like Key - value Pair or like Objects or like Documents but they are not Replacements for SQL ones because

* It has been seen that there are consistency issues in them (Consistency in Transaction) (Data that user Read should be latest one) But these donot Guarantee that. They are Horizontally Scalable.

eg- Mongo DB, Dynamo DB (By Amazon), Cassandra etc

→ Entity Relationship Model (ER Model)

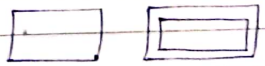
It is a way to design the Database. It is divided into 3 Parts :-

1) Entity Set : Student, Teacher, Course
2) Relationship Set : Teaches, Gives, Joins
3) Attributes : Upar ke dono chizo ki description like Name, Roll no. etc in case of Student or Joining date / Duration etc in Teacher & so on

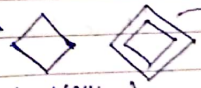
* Entity is a noun ; Relationship is Verb (action)

Representations

- Entity Set →

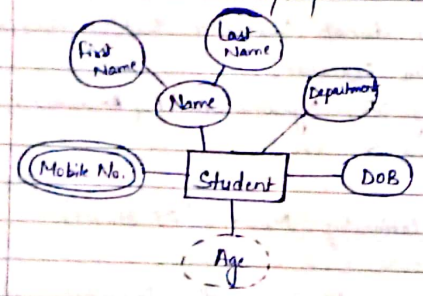


- Relationship Set →



- Attributes → (oval / Ellipse)
  - Composite (Ek hi attribute me multiple chize)
    - Name (First name, last name)
    - Address (S.No, City, Society, State)
  - Multivalued (Ek hi ke multiple bhi ho skte)
    - Mobile No.

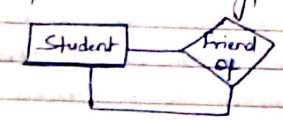
→ Derived (Mtlb Inhe alag se dastne ki zarurat Nhi Balki Kisi aur Attribute ki value given the to Derive the skte ho)
  - Age
  - Duration (Begin & End date)
- Key → which defines all the entities uniquely like Primary key



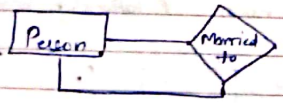
(Department is Single Valued as vsh mobile No. ki Tarah multiple Nhi ho skte Ek Entity ke)

→ Degree of Relationship sets
3 Types

1) Unary - Ek hi Type ki Entity pe kaam ho

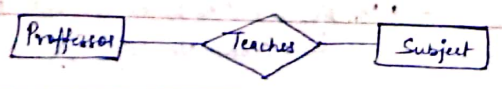


Student Friend of Student

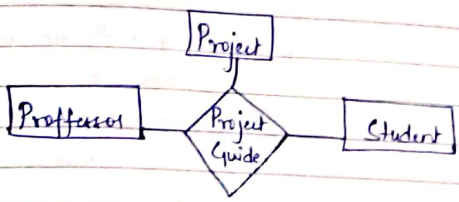


Person married to a Person

2) Binary (Most common)

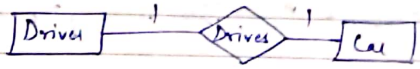


3) n-ary (Not very common) (More than 2)

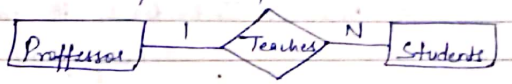


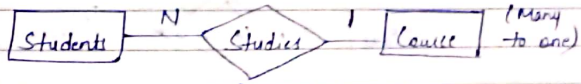
→ Cardinality (Ek Entity Kitni Baar use ho Rhi)
3 Types

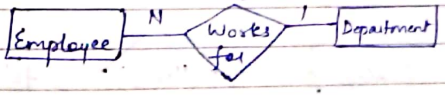
1) One to One (done one time use hua)



2) One to Many (Many to one bhi isime ata)



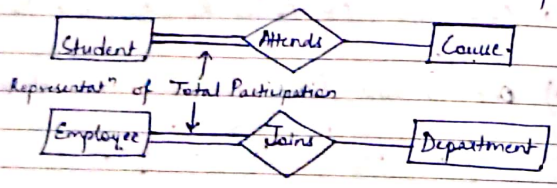




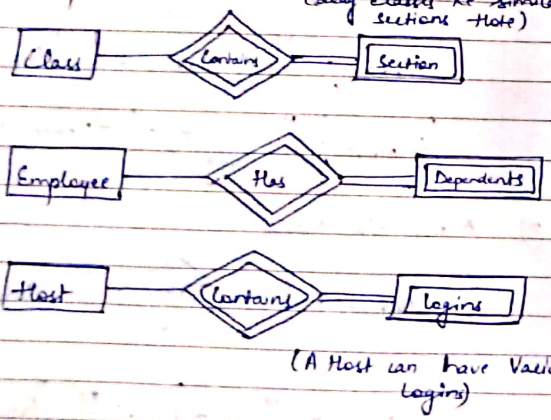
3) Many to Many

→ Participation & Weak Entity Set:

1) Total Participation → Every entity of one side participates in the Relationship
eg:, These things which are compulsory to do like Every student must be involved in some course or Every employee must Join a department



2) Weak Entity Sets → Donot have their own key Always Total Participation & to become a key weak Entity set has to take help of other Entity present. Individually it could not become a key.



→ Keys - An Attribute or set of Attributes that uniquely Identifies set of Entities
Candidate Key ; Primary Key ; Super Key ; Alternate key

Enroll No. (Primary)	Name	Adhar No.	Add^{r}	PAN No.	Ph. No

Now, here in this database (Enroll No., Adhar No. & PAN No.) are
- Candidate Keys (one or more) as they are the minimal set of attribute that derives all other attribute (Candidate Key must be minimal, Kmse km attribute ko to Identify)
- Primary Key (only one) Koi Ek Choose ho Jaati Hai From the options of Candidate Keys & it is most convinient & Baaki Tables me Bhi use ho Rha hota Hai
- Alternate Key - Candidate ki Bchi Kuchi Options After Choosing primary.

- Super Key - (Greater than or equal to Candidate Keys (2-3 Attributes ko milake Bhi)) Any set of attributes that can uniquely identify the Table contents.
If no. is minimal like 1 than Candidate key & from them we shortlist primary key

* In worst case, Hum saare attributes ko ikathe krdete Hai as vo to hai Row ke liye unique hi honge. These will Represent all the keys in that case.

CREATE TABLE Order (
:
Cust_ID REFERENCES Customes (Cust_ID)
ON DELETE CASCADE
); for foreign Key (For Referential Integrity)

* Foreign Key same table ke primary key ko bhi point Kr skta Hai. Refrential Integrity means agar ek Jagah se entry delete Kro to Baaki Sab Jagah pr Bhi pta lg Jaaye ki abh iska Kaam Nhi Koi
∴ CASCADE line is solution for that, more solutions could be Restrict/NULL.

→ Database Normalization
Motive is to decrease Data Redundancy & Increase Data Integrity

- Redundancy is thät same data is occuring at multiple places specially string ones
- Integrity means less erroneous data i.e Kam se Kam error vala
(Related data is Together)
We divide data into multiple Tables so the Redundancy is minimised.

* Objectives of Good DB Design
a) No Updation, Insertion & deletion anomalies
b) Easily Extendible
c) Good Performance for all Query sets
d) More Informative

(This is Bad DB as there is Redundancy)

→ Anomalies

1) Updation anomaly

Student_ID	Student_Name	Subject_ID	Subject_name
1001	ABC	CS101	Database
1002	XYZ	CS101	Database
1001	ABC	CS102	OS
103	BCD	CS102	OS
5th → (ii) 104	PQR	?	?

if we update database to DBMS than due to Redundancy Khi Khi the Jayega Khi Khi Nhi Hoga, Crash ki vjah se ya Bad Query ki vjah se

2) Insertion anomaly → as upar 5th Entry insert Kri toh as Bad DB to he skta Hai Bas Kuchh entries hi the add krne ko & iski Vjah se possibly Kei Restrict" the ki adhi adhuri entries enter Nhi kr skte toh Problem Hogi

3) Deletion Anomaly - As Bad DB toh lete suppose thumne 1st & 2nd Entries delete Krdi (---) (is line se) toh Database subject ka toh sara data hi Ud Jayega i.e subject hi Nhi Rhega.

∴ to Reduce Redundancy we Break data into multiple Tables But Bohot Jyada Tables Bhi Nhi Honi Chahiye Ki Performance issues aa Jaye so there are Rules for normalization

→ Functional Dependency

Enroll No.	Name	Add^{y}	Ph. No.

Enroll No → Name
" → Add^{y}
" → Ph. No.

Student_ID	Subject_ID	Marks	Grade

Subj-ID, Student_ID → Marks, Marks, Grade

A → B ⇒ B is Functionally dependent on A

A	B
x	1
y	1
z	2

(A → B ✓, B → A X) Here A is unique values & could be used to uniquely Identify all the B Values

Enroll_No.	Name	Add^{y}
101	Raj	ABC
102	Ravi	ABC
103	Raj	XYZ

Enroll No. → Name, Add^{y} ✓
Name → Add^{y} X
Add^{y} → Name X

* Why do we study Functional dependencies

Stu_ID	Name	Dept_ID	Dept_Name	Dept.
101	ABC	10	CS	
102	BCD	11	ECE	
103	ABC	10	CS	
104	XYZ	11	ECE	
105	CDE	10	CS	

Dept_ID → Dept_name
(So due to this we could divide it into multiple Tables)

Dept_ID	Dept_Name	Dept
10	CS	
11	ECE	

Divided & Seperated the data into other Table.

Functional Dependency

Trivial	Non-Trivial
$AB \to A$	$A \to B$
$A \to A$	$AB \to C$
$ABC \to AC$	$BC \to DEA$

→ **Database Normalizat^n**

Data Ko much more efficient way me store Krne Ka Tarika so that har ek chiz Efficient way me store ho & Operations bhi like it is said to divide data into multiple Tables. (1NF)

1) **First Normal Form** → If Every attribute contains only single value (Atomic)

Cust_ID	Name	Mob No.
101	ABC	886..., 9960...
102	BCD	8527...

Not in 1NF

a) → ↓ Solut^n

"	"	Mob. No. 1	Mob. No. 2
		886...	9960...
		—	99...

(Solut^n but not always possible as if you are Reqd. to fill ~~the~~ the Skills that Kitne Attribute Bnaloge)

b) → Sol^n

Cust_ID	Name	Mob No.
101	ABC	Mob1
102	BCD	99...
⋮	⋮	⋮
101	ABC	Mob 2

Table A

(This is better Solution But Here Cust_ID would not Remain unique)

(Not in 3NF)

Table A me Ek Attribute aur Jodlo

ID	Customer_ID	Mob. No.

This is not only in First Normal Form

2) **Second Normal Form (2NF)**

Must be in 1NF, & No Partial Dependency

↓

No Non-Prime Attribute Should depend upon partial Candidate Key

User_ID	Course_ID	Course_Fee
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000

(Not in 2NF)

(User_ID, Course_ID) → Total Prime Key

But as course_ID → Course_Fee

∴ Partial Candidate key par ek Non Prime Key ~~de~~ Depend Kr Rhi Hai toh Not in 2NF

Solut^n is

User_ID	Course_ID	Course_ID	Course_Fee

~~If a Table is in~~

3) **Third Normal Form (3NF)**

Must be in 2NF & Non-Prime → Non Prime

(a) (b) Not Allowed

↓

No Non Prime should depend on Non Prime

Stud_No.	Stud_Name	Stud_State	Stud_Country
101	Ram	Haryana	India
102	Ramesh	Punjab	"
103	Suresh	"	"

Stud No. → {Every Attribute}

(Stud_State → Stud_Country) Rule No (b) Violated

* If a table is in 2NF & one non Prime is defining other Non Prime than we take these non prime & make a New Table But Jonsi Chiz define Kr Rhi Hoti. Hai use them original Table pe Bhi RKhte Hai

T_1 : Stud_No, Stud_Name, Stud_State

T_2 : Stud_State, Stud_Country

Now these 2 Tables are in 3NF

eg → | Exam Name | Exam Year | Topper Name | Topper DOB |

(Exam Name, Exam Year) → Topper name, Topper DOB

Topper DOB → Topper DOB

Now To fix it to be in 3NF we put First 3 Attr. in one Table & last 2 Attr in Other Table to divide

Definition-2 ⇒ (i) In 2NF

(ii) Non Prime attr. not Transitively dependent on prime attributes

4) BCNF → More strict than previous. 3

2NF ⇒ P → NP ✗ 3NF ⇒ NP → NP ✗

BCNF ⇒ P/NP ↪ P ✗

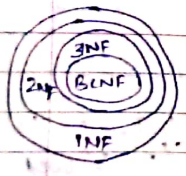
(1, 2 & 3 Normal Form me to ho hi But yeh upar vali cond^n Bhi Satisfy Hoti)

Only Superkeys on the left Hand Side (X → Y) (X → Superkey)

eg- | Stud_ID | Subject | Prof_ID | (Stud_ID, Subject) → Prof_ID

Prof_ID → Subject

This is not BCNF → Partial Cand. Key → NP



Stud_ID	Subject	Prof_ID
1001	DBMS	103
1001	OS	110
1002	DBMS	111
1003	DBMS	103

Redundancy

1001	103	103	DBMS
1001	110	110	OS
1002	111	111	DBMS
(1001)	(104)	(104)	DBMS

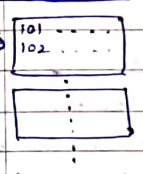
Redundancy is Gone

Sometimes it Creates problems as above. Dependency is gone but some Tables are not correct according to DB so some decompositions might not be acceptable in BCNF. me Lekin paana kuch Tarah se kafi mushkil Hai

→ Indexing in Databases

1) Clustered Indexing - It is to put Indexing on Hard disk Block for data of Database & could be on Basis of any attribute but there is one Rand on primary key.

eg- Disk Blocks



Order

Order_ID	Order_date	Cost	Cust_ID
101	15-06-18	2000	102
102	15-06-19	5000	103

Clustered Indexing
Prime Index → Primary Key

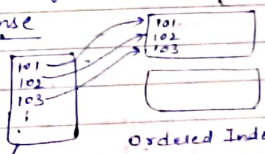
(Now if we don't do Indexing than it would act like a heap ki Bas aye Jis Rha Hai But Fir performance issue Hoga as Searching Time in HDD is more) Clustered Index problem is that it is on Basis of one key only for more comes under case of non clustered

2 Types → Sparse & dense

Search Key	Pointer

Stores Reference to disk Block containing given Key

(i) Dense



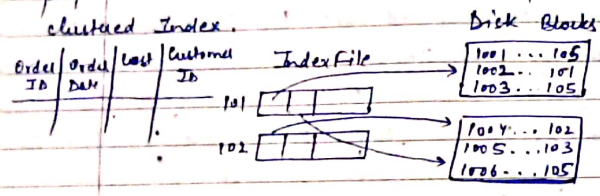
Ordered Index file
Hashing file origin

(Dense doesn't mean ki All the Rows must be present but all unique ones also index file me Hai entry ek Bar)

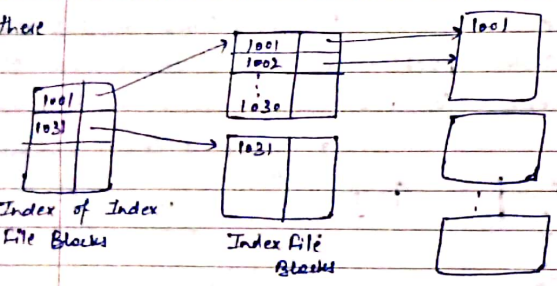
Ek Ek Index Bohot hi densely Present Hoga Hai ek Entry ka → Hai key ke liye Ek Index toh Faster But Updat^n & Delet^n ke Time Bohot dhyaan RKhna Pdta Hai to maintain

(ii) Sparse → as Indexing is sequential toh ek ek Block (disk) ke Base Address ki Keys ko store krleta hai as Baaki sab derive ho jaayega
(Slow but Updat^n & delet^n very convinient).

2) Non Clustered Indexing (only dense) → Clustered Index in DB are by default made on primary keys, also we should not make many secondary indexes as then it would create a problem of performance & overhead/ updation so only that entry which is used very oftenly should have secondary Indexes. If we want to index the DB according to another attribute to Increase performance than we need secondary address of non clustered Index.



3) Multilevel Indexing → Main purpose is to Reduce disk Block access time. But Eventually comes a case where size incr a lot there we need B & B+ Trees. Due to multilevel less access to Blocks are there

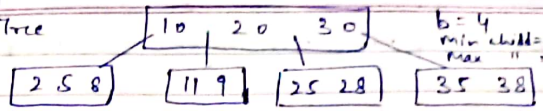


& B+ Trees

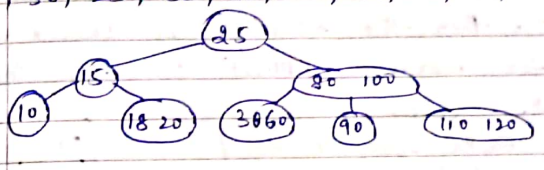
4) B- Trees$_n$ → n- way search trees
↳ If Branching Factor is b, then every node has $[b/2]$ to b children
↳ In B+ Trees leaf node contains all the data present in Internal nodes so that Sequential Access to data is Possible.

These are self Balancing Trees so Height don't exceed ($\log n$). Also
No. of Keys in node = no. of Children - 1

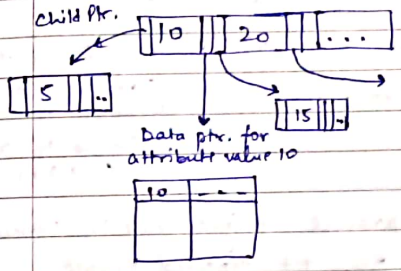
eg- B - Tree



Structure ek dum BST Jaisa Root se chhota left me Bda Right me. (Also yeh Restructure krleta hai & iski Height upwards Bdti hai). Insert^n ↑ses the size & length whereas vice versa for delet^n
eg- 10, 30, 25, 80, 90, 100, 15, 18, 20, 60, 110, 120

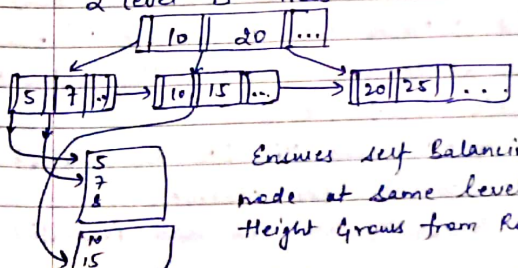


B- Tree Node Structure



* B+ Trees → Internal nodes have pointers to index Blocks only. Leaf nodes store pointers to actual data Blocks. Only the last ptr in the leaf nodes point to next Index Block (leaf node me Sequential access & mostly companies isse use krti hai)

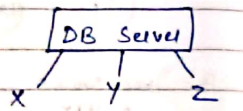
eg- 2 level B+ Trees



Ensures self Balancing & all leaf node at same level & here Height Grows from Root Node.

Advantage → All the nodes has only one Type of Pointer unlike B-Trees & has sequential access.
Disadv. → Key value pair might be more than 1 so duplicate key value pair might occur in parent & children

→ Transaction & Concurrency Control (ACID Properties)
Transact^n is a set of ~~process~~ instructions that should happen all together at same time or nothing should happen. Concurrency is just like in OS preemptive nature like if some process working on I/O then the other process should use the CPU for maximum utilizat^n same for server Request. Transfer
eg- 100 Rs from X to Y
Before X = 500 ; Y = 200
Read(X) → X = X-100 → Write(X)
Read(Y) → Y = Y+100 → " (Y) → if in b/w this Transact^n one more Request comes 200 Rs to X
After X = ~~6~~ 400 ; Y = 300
X → 700

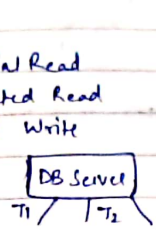


∴ Interleave process might lead to concurrency so might not get expected Result

* ACID Properties
i) Atomicity → Either whole thing executes or Nothing executes
ii) Consistency → means ki in total amount should Remain ~~same~~ constant before & after Transact^n
eg- 500, 200 = 400, 300 (Sum)
iii) Isolated → Means esa lgna chahiye ki pura process akele me chal Rha hai without anyone's Interference
iv) Durability → means ki DB me latest updated values Rhni chahiye.
eg- Bank Transact^{n}s me yeh sab Properties

→ Serializability → It is used to validate interleaved schedule that whether it is valid or not

(i) View Serializability → T1 ke Read T2 ke ya ulta Effect same hi ana chahiye. i.e different Permutat^{n}s of T (Transact^n) ke liye same effect ana chahiye
eg-
For every Item: Initial Read, Updated Read, Final Write



	Transfer 10 Rs X → Y (T1)	Transfer 10 Rs Y → Z (T2)
Schedule	Read(X)	
	Write(X)	
		Read(Y)
		Write(Y)
	Read(Y)	
	Write(Y)	
		Read(Z)
		Write(Z)

Such- T1 T2
Read(X)
Write(X)
Read(Y)
Write(Y)
Read(Y)
⋮
Write(Z)

In View Serializability, the data should be such as agar koi si Bhi Permutat^n me agar process ho such as T1 T2 or ulta or if 6 Transact^n toh 6! ways toh dekhne se esa hi lgna chahiye ki processing same manner me ho Rhi hai. But To check it is very expensive operat^n very time consuming

(ii) Conflict Serializability → In this we only talk about ~~conflict serializability~~ Conflicting process for serializability Conflicting Operat^{n}s → 2 operat^{n}s from 2 different Transact^{n}s but on same data items & one of them must be "write"

Given Schedule

T1	T2
Read A	
Write A	
	Read A
	Write A
Read B	
Write B	
	Read B
	Write B

Serial Schedule (T1 T2)

T1	T2
Read A	
Write A	
Read B	
Write B	
	Read A
	Write A
	Read B
	Write B

Ab upar Niche Krke Bhi different data items so conflict Serializability is fine

Given Schedule

T_1	T_2
Read A	
	Read A
	Write A
Write A	
Read B	
Write B	
	Read B
	Write B

This is neither Conflict Serializable with $T_1 T_2$ nor $T_2 T_1$

* Conflict Serializability could be checked using Precedence Graph

This concept is to check that DB is consistent even after interleaving

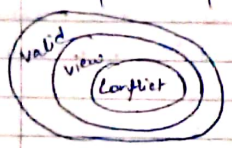
* If we draw Graph & there is a cycle b/w them then it is not at all conflict Serializable at ek kheya phle Sare T_1 pe kro dusra kheya Phle T_2 (case)

This is more strict than view serializable

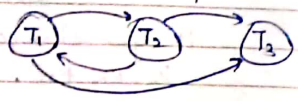
T_1	T_2	T_3
Read A		
	Write A	
Write A		
		Write A

View serializable but not Conflict

But not costly operation



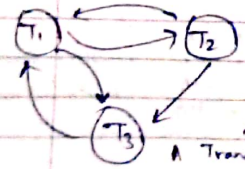
Agar Sbse upper vala satisfy toh Baaki vale Toh thaga hi



Precedence Graph thum kewal Conflicting process ka Bnate hai & cycle detect krte hai

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

$R_1(A)$ & $W_2(A)$ $[T_1 \to T_2]$
$R_1(A)$ & $W_3(A)$ $[T_1 \to T_3]$
$W_2(A)$ & $R_3(A)$ $(T_2 \to T_3)$
$W_2(A)$ & $W_1(A)$ $(T_2 \to T_1)$
$W_2(A)$ & $W_3(A)$ $(T_2 \to T_3)$
$R_3(A)$ & $W_1(A)$ $(T_3 \to T_1)$
$W_3(A)$ & $W_1(A)$ $(T_1 \to T_3)$



As cycles detected not conflict Serializable

A Transaction is Serializable if its outcome is equal to outcome of its Transaction executed serially i.e sequential without overlapping in time

(I) * Recoverable, Cascadeless & Strict

As we know in DB, Transaction may not execute completely either due to Hardware Failure, System Crash, Software issue in that case we have to Roll Back failed Transaction, But if some other Transaction also have used same values so we have to Roll Back those Transactions as well. So we need Recoverability.

eg- A=100

T_1	T_2
Read(A)	
A=150 → Write(A)	
	Dirty Read → A Transaction doing dirty Read must Commit after the Transaction From which it is Reading
	Read(A) → A=200
	Write(A)
Read(B)	
	Commit ↑
Write(B)	
Rollback	

(If don't commit then dirty Reads cause Recoverability Issues)

∴ For Recoverable Schedule transactions must be committed in order.

Dirty Read → when a Transaction Reads a data from uncommitted write in another Transaction

(II) Cascadeless Schedule → Dirty Reads are not allowed i.e reading data written by an uncommitted Transaction not allowed.

T_1	T_2	T_3
Read A		
Write A		
	Read A	
	Write A	
		Read A
		Write A
Read B		
Write B		
	Rollback	

⇒

T_1	T_2	T_3
Read A		
Write A		
Read B		
Write B		
commit		
	Read A	
	Write A	
	commit	
		Read A
		Write A
		commit

(III) Strict → This takes into consideration the Blind writes that is the writes without Reading so if Rollback happens than value would not be updated properly. Therefore, Neither Dirty Reads nor lost update problem allowed. Rule is that Blind writes are only allowed after Transaction is committed ∴ Read, write from uncommitted Transaction not allowed

→ 2 Phase locking Protocol → These are the protocol that has the objective that we want to ensure concurrency, interleaved Transact^n must happen & to ensure that this is conflict Equivalent to a Serial Schedule.

* Lock → it is to lock something in DB & if something is locked then no other Transact^n can Take place. lock is same as the one in OS

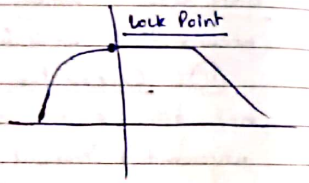
2 Types → Shared → as we need to apply concurrency so Reading ke time pe share krr sab log Hence Shared lock

→ Exclusive → Whereas during writing or modify we need exclusive so that no interferance is there.

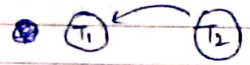
S – Shared
X – Exclusive

eg.

T_1	T_2
lock-S(A)	
Read(A)	
	lock S(A)
	Read(A)
lock-S(B)	
Read(B)	
lock-X(A)	
↓ wait for T_2 to Release Shared lock on A	
	unlock-S(A)
write(A)	
unlock-S(B)	
unlock-X(A)	



Lock point is when all the acquiring of the locks have been done & now Locking Nhi Hogi only Releasing

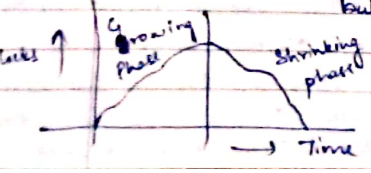


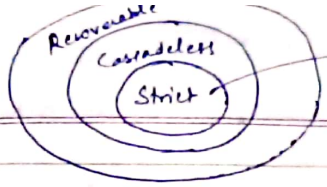
conflict serializable
i. View "

2 Phases in Every Transact^n

→ Growing phase → New locks on data items may be acquired but none can be Released

→ Shrinking phase → Existing locks may be Released but no new locks can be acquired





→ much Strict condⁿ than Rest of 2

* Problems with 2PL (2 Phase Locking)

(1) Deadlock ⇒ conservative 2PL

T_1	T_2
lock X(A)	
	lock X(B)
lock X(B)	
↓ wait for T_2	
	lock X(A)
	↓ wait for T_1

Now this wait would be ∞ hence case of a deadlock

(11) Recoverability ⇒ Strict 2PL, Rigorous 2PL

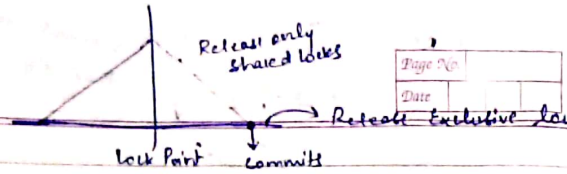
T_1	T_2
lock X(A)	
Read(A)	
write(A)	
unlock X(A)	
	Dirty Read
	lock-S(A)
	Read(A)
	unlock-S(A)
	commit
Roll Back	

* Types of 2PL

1) Conservative 2PL – Handles deadlock → Remove Hold & wait (Aquire all lock First), this protocol Requires Transact^n to lock all the items it access before the Transact^n Begin execu^n. If any of the predeclared items needed can't be locked, the Transact^n does not lock any of the items, instead, it waits untill all the items are available for locking.

NOTE → deadlock is not that Big issue as it happens Rarely, also this doesn't take care of Recoverability which is more important.

2) Strict 2PL → ~~Rastrox~~ Releases all Exclusive locks only after Transact^n commits. Ensures that Schedule is Recoverable, Cascadeless



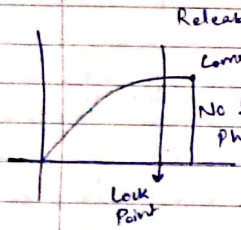
3) Rigorous 2PL → Releases all Exclusive & Shared locks after Transact^n Commits.

Conservative | Strict | Rigorous

Conservative	Strict / Rigorous
i) Deadlock Free	Deadlock Possible
ii) Recoverability & Cascadelessness not Guaranteed	Recoverability & Cascadelessness Guaranteed

Rigorous v/s Strict

- Strict & allow more concurrency
- Simple to Implement & most used (Imp as most used Protocol)
- Rigorous also suitable in distributed Environments



Easily implemented as pichhle dono me Same Lock points Dhundo & then unpe kam Kro

Rigorous → also called Strong Strict

→ Time Stamp Based Protocol → Is protocol ka bhi same objective thi. Timestamp is a unique identifier created by DBMS. Main idea is to order the Transact^n's Based on these Timestamps. This makes it Serializable & equivalent to Serial Schedule.

TS(Ti) → Time stamp of Transact^n Ti

RTS(x) → Timestamp of latest Transact^n that performed Read on object (x) ∴ max Timestamp

WTS(x) → Timestamp of the latest Transact^n that Performed write on object (x) ∴ max Timestamp

eg- T1(10) | T2(20) | T3(30) — T1 T2 T3

eg-

TS(T1) = 10
TS(T2) = 20
RTS(A) = 10
WTS(A) = 10
RTS(A) = 20
WTS(A) = 20
RTS(B) = 20

T1 (10)	T2 (20)
Read(A)	
Write(A)	→ Dirty Read
	Read A
	Write A
	Read B
Write B	
Roll Back	

T1 T2

Yha Problem Aayi so now System will Roll Back & instruct Krega Ki let them come in a new Time stamp which is conflict Free

Now, there is no possibility of deadlock But Recoverability could be an issue as there occurs Dirty Read (as in above example)

* Simple Rules

I) Rule for Read(x) in Ti

if TS(Ti) < WTS(x) then abort Ti

else

Allow Read(x)

RTS(x) = max(RTS(x), TS(Ti)) → current one

II) Rule for write(x) in Ti

if TS(Ti) < RTS(x) OR TS(Ti) < WTS(x)

then abort

else

Allow Write(x)

WTS(x) = TS(Ti)

NOTE→ Extra work / Book Keeping could be done in this Protocol to ensure Recoverability & Cascadelessness by applying Restrict^{n}s to all dirty Read operat^{n}s. This work is done by Scheduler whose job is to Schedule Transact^n in an interleaved manner & then protocol checks for these Rules.