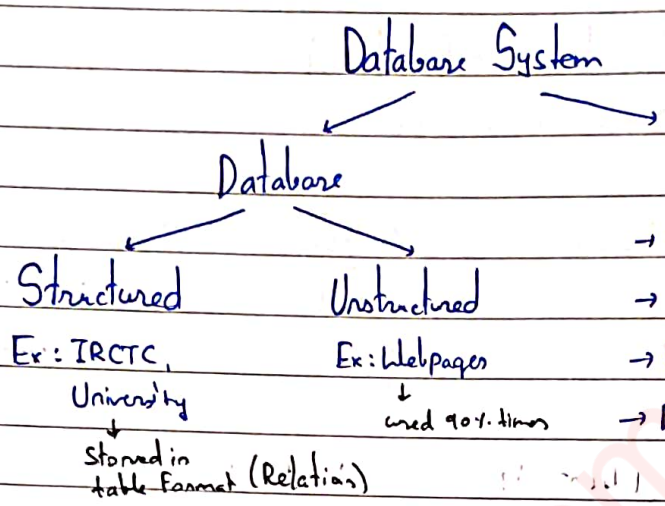
DBMS

☆ Introduction :-



⇒ Database :

→ A database is a collection of related data which represents some aspect of the real world.
→ A database system is designed to be built and populated with data for a certain task.

⇒ Database Management System (DBMS):

→ It is a software for storing and retrieving user's data while considering appropriate security measures.
→ The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data.
→ In large systems, a DBMS helps users and other third-party software to store and retrieve data.

⇒ File System vs DBMS / Advantages of DBMS :

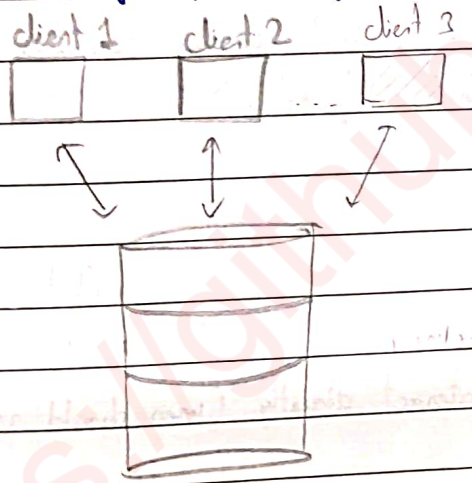
DBMSs were developed to handle the following difficulties of typical file-processing systems supported by conventional operating systems.

1) not used for smaller data, as it is expensive
2) not used for single user

1) Data redundancy and inconsistency (repetition/copies of same data)
2) Difficulty in accessing data (If we want to search a particular file, whole file system will be downloaded)
3) Data isolation (if we make changes in the file then the same file, is also changed of other user)
4) Integrity problem (Ex: Id should be unique)
5) Atomicity of updates (Either run completely or restore to original state)
6) Concurrent access by multiple users (x handle multiple users)
7) Security Problems (Role based usuage not possible) → Ex: College DBMS → student, Instructor

⇒ 2 tier Architecture: (Client-Server Architecture)

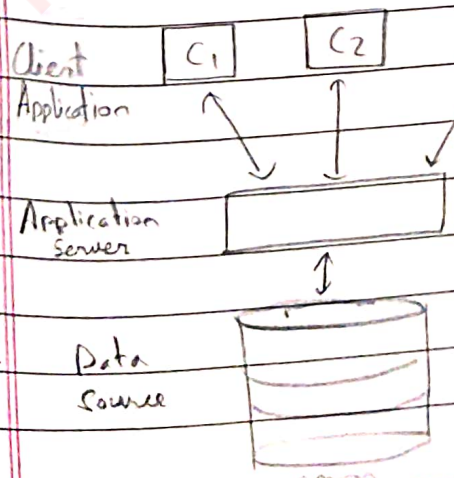
2 tier → 2 layer (client layer and Database Server)



Advantages:
1) easy to maintain (as limited clients)
2) cheap

Disadvantages:
1) Scalability, Security (as client is directly accessing server)
(database → solve query, fetch data)

⇒ 3 tier Architecture:



Advantages:
1) scalability (database → fetch data
Application layer → solve query)
2) security (as client is not directly fetching the data)

Disadvantages
1) difficult to maintain (complex)
2) expensive

★ Schema :-

Schema is logical representation (structure) of a database.

In RDBMS, how do we represent data logically? using tables / relation

In ER model, we represent data logically using entities & relationships.

Ex: Student

Rollno	name	address

Course

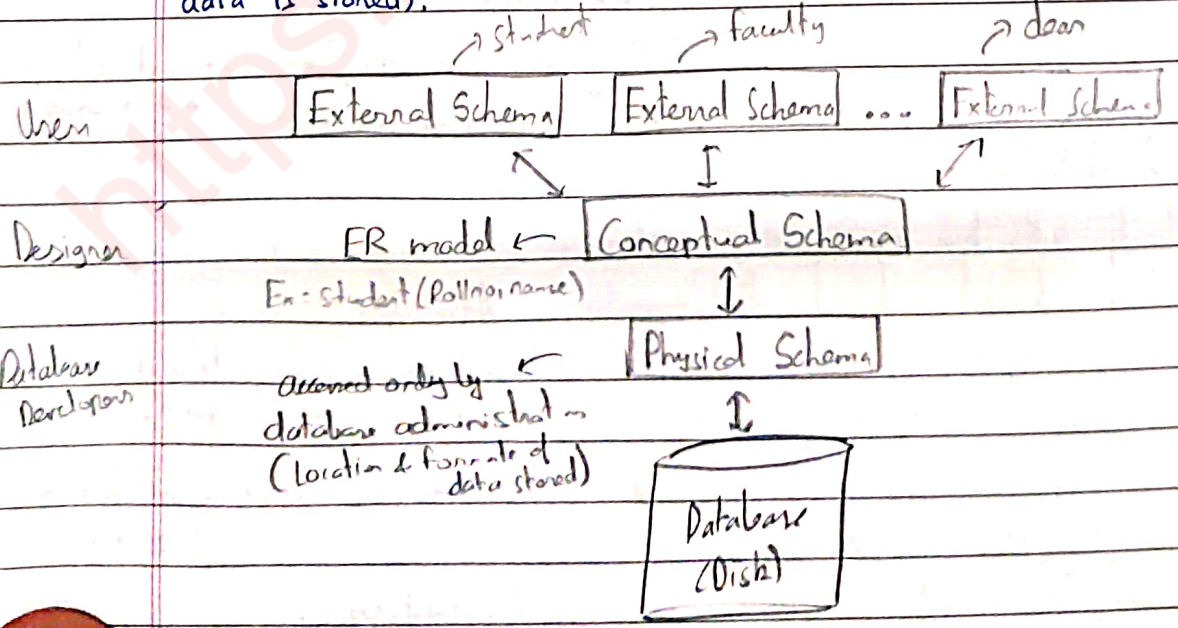
CourseId	name	duration

→ But actually schema is implemented using SQL. (DDL → commands)
→ But schema doesn't show everything that schema contains. Ex: constraints
→ Schema can be a table or a collection of tables.

⇒ 3-Schema Architecture:

(Data abstraction / hiding)

→ The main motive behind constructing this architecture is Data Independency.
i.e. user and data should not interact directly (user should not know where & how data is stored).



→ For users data is appeared in the form of a table but in actually harddisk it is stored in files.

Data independence is achieved using both 1) Logical Data Independence
2) Physical Data Independence

1) Logical Data Independence

→ Changes in conceptual schema does not affect the external schema (view level).

Ex: Conceptual Schema → Student (name, age)

User 1 makes changes in student → student (name, age, roll no.)
User 2 will not see the changes → student (name, age)

2) Physical Data Independence

→ Changes in physical schema does not affect the conceptual schema.

Ex: change in 1) storage structure 2) data structure change 3) index

★ Keys :-

→ Key is just an attribute in a relation which can uniquely identify any two tuples.

Ex: Student

Roll no.	name	age
1	Aryan	18
2	Dhruv	19
3	Aryan	18

Student table can have many keys like:
1) Adhaar card
2) Roll no.
3) Licence no.
4) Voter id
5) Phone no.
6) email - ID

Here set of all these keys are called candidate keys.
Only 1 of them is Primary Key rest others are Alternative Key

⇒ Primary Key:

Candidate Keys (Unique)

↙ ↘

Primary Key (cannot be NULL) | Alternate Keys (can be NULL)

In previous example of student Relation, Roll no. is the primary key and rest others like Adhaard card, phone no, licence no are alternating keys.

⇒ Foreign Key:

It is an attribute or set of attributes that references to primary key of same or another relation.

It maintains referential integrity.
↓
same value in DB

Ex: Student

PK

Rollno.	name	address
1	A	Delhi
2	B	Mumbai
3	A	Pune
⋮	⋮	⋮

Referenced / Base table

Course → FK

Course Id	Course name	Roll no.
C1	DBMS	1
C2	Networks	2
⋮	⋮	⋮

Referencing table

→ A table can have only 1 primary key but it can have more than 1 foreign key.

Referential Integrity: $(dom(FK) \subseteq dom(PK))$

1) Referenced Table

i) Insert → no violation

ii) delete → problem will be there if it has some record in the referencing table also
i.e. Referential Integrity breaks

Sol's : a) on delete cascade (removing from base table, also removes from referencing table)
*b) on delete set null (removing from base table, sets null in referencing table)
c) on delete no action (just remove from base table)

* → FK can be primary key in that table

iii) Update → same as delete operation

2) Referencing Table

i) Insert → may cause problem if there is no record of it in the base table ($dom(FK) \subseteq dom^{(PK)}$)

ii) delete → no violation

iii) Update → same as Insert

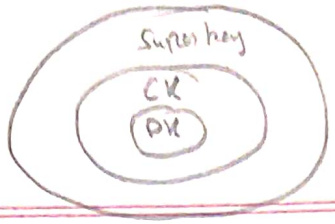
Q Let $R_1(a,b,c)$ and $R_2(x,y,z)$ be two relations in which 'a' is a foreign key in R_1 that refers to primary key of R_2. Consider four options: a) Insert into R1 b) Delete from R1
c) Insert into R2 d) Delete from R2

What is correct regarding referential integrity?

(A) options a) and b) cause violation
(B) options b) and c) cause violation
(C) options c) and d) cause violation
(D) options d) and a) cause violation

⇒ Super Key:

A Super key is a combination of all possible attributes which can uniquely identify two tuples in a table.



Ex: Student

CK	Roll no.	name	age
	.	.	.
	.	.	.
	.	.	.

{{Roll no}, {Roll no, name}, {Roll no, age}, {Roll no, name, age}}

↓ minimal super key

↓ Super keys

Super set of any candidate key is Super key.

Q $R(A_1, A_2, \ldots, A_n)$ then how many super keys are possible. If

a) A_1 is a candidate key

b) A_1, A_2 are both candidate keys

c) A_1, A_2 → composite candidate keys & A_3, A_4 → composite candidate key

a) A_1 must be there $\therefore Ans = 2^{n-1}$ ✓

b) A_1 must be there OR A_2 must be there or A_1 and A_2 both must be there

$\therefore Ans = 2^{n-1} + 2^{n-1} - 2^{n-2}$ (jisme A_1, A_2 both aaye)

A_1 repeat A_2

(A_1, A_2) (A_2, A_1)

$A_1 A_2 A_3$ $A_2 A_3 A_4$

⋮

c) $A_1 A_2 \rightarrow CK_1$ $A_3 A_4 \rightarrow CK_2$

2^{n-2} 2^{n-2}

$\therefore Ans = 2^{n-2} + 2^{n-2} - 2^{n-4}$

★ E-R Model (Entity-Relationship Model):-

Entity set → Student (name, roll No)

Entity → Aryan 21UCS037

⇒ Introduction:

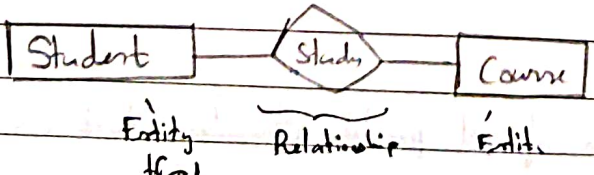
Entity: Any object which has physical existence.

Entity type (Schema)

↓

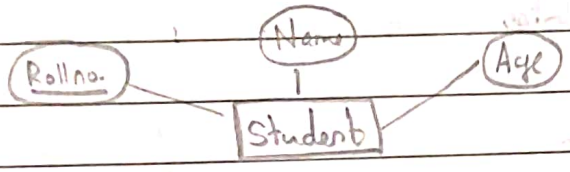
Student (Roll no, age, address)

attributes



⇒ ER diagram:

→ ER diagram is a conceptual model that gives the graphical representation of the logical structure of the database.
→ It shows all the constraints and relationships that exist among the different components.
→ An ER diagram is mainly composed of the following three components - Entity sets, Attributes and Relationship Set.



⇒ Entity Set:

An entity set is a set of the same type of entities.

1) Strong Entity Set
→ A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
→ In other words, a primary key exists for a strong entity set.
→ Primary key of a strong entity set is represented by underlining it.

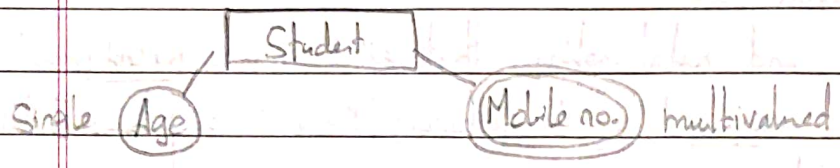
2) Weak Entity Set
→ A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.
→ In other words, a primary key does not exist for a weak entity set.
→ However, it contains a partial key called a discriminator.
→ Discriminator can identify a group of entities from the entity set.
→ Discriminator is represented by underlining with a dashed line.

⇒ Participation Constraint: Participation constraint is applied to the entity participating in the relationship set.

1) Total Participation: Each entity in the entity set must participate in the relationship.
Shown by double line.
Ex: If each student must enroll in a course, the participation of student will be total.

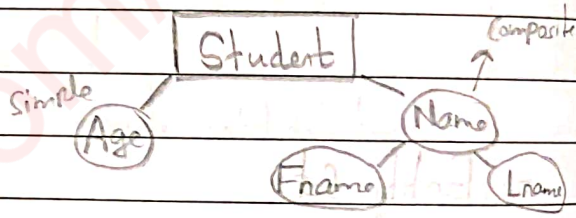
⇒ Types of Attributes:

1) Single v/s Multivalued attributes



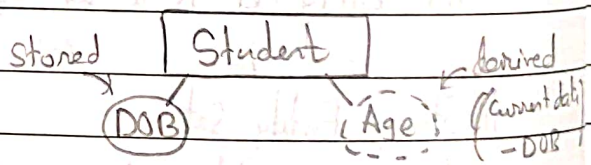
2) Simple v/s Composite attributes

Simple → cannot be divided further
Composite → composed of other simple attributes



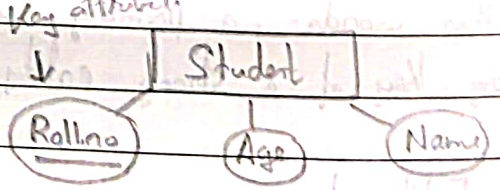
3) Stored v/s Derived attributes

Stored → value fixed/stored
derived → valued derived from other attribute



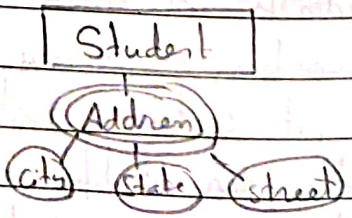
4) Key v/s non-key attributes

Key → uniquely identify each tuple



5) Complex attributes

Complex - composite + multivalued



⇒ Types of Relationships / Degree of Relationship (Cardinality):

2) Partial Participation: The entity in the entity set may or may not participate in the relationship.
Ex: If some courses are not enrolled by any of the student, the participation of course will be partial.

1) One-to-one relationship

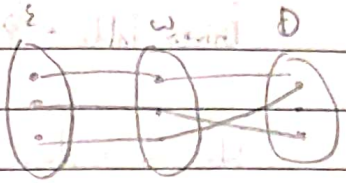


Table Representation:

EMPLOYEE

Eid	Fname	Age
E_1	A	20
E_2	B	25
E_3	C	26
E_4	A	28

Works (FK: Eid → EMPLOYEE, FK: Did → Department)

Eid	Did	...
E_1	D_1	
E_2	D_3	
E_3	D_2	

Department

Did	Dname	Loc
D_1	IT	Banglo
D_2	Prod.	Delhi
D_3	HR	Mumbai
D_4	--	

Number of tables = 3

PK? either Eid / Did

Can we reduce no. of tables? yes, merge employee and works table.

Eid	Ename	Age	Did
E_1	A	20	D_1
E_2	B	25	D_3
E_3	C	26	D_2
E_4	A	28	—

∴ Number of tables = 2

2) One-to-many relationship

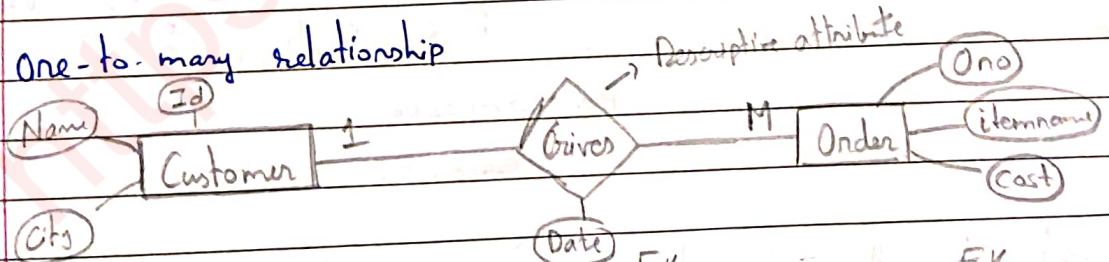


Table Representation:

Customer

Id	Name	city
C_1	A	...
C_2	B	...
C_3	C	...

Gives (FK: Id → Customer, FK: Ono → Order)

Id	Ono.	Date	...
C_1	O_1	--	
C_1	O_2	--	
C_2	O_3	--	
C_3	O_4	--	

Order

Ono.	item_name	cost
O_1	Bucket	300
O_2	Shoes	1500
O_3	Shirt	1200
O_4	Jeans	2000

Gives Table – PK (Ono.)

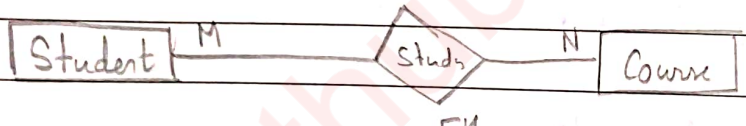
We can reduce the no. of tables by merging 'Gives' and 'Order' tables

Custom

Id	Ono.	item.name	Cost
C_1	O_1		
C_2	O_2		
C_3	O_3		

3) many-to-one → same as one-to-many

4) many-to many relationship



A student studies multiple courses

A course is studied by

Table Representation: Student → Study ← FK → Course

Rollno.	name	Age
1	A	18
2	B	17
3	A	16

Rollno	Cid
1	C_1
1	C_2
2	C_2
2	C_4

Cid	name	credit
C_1	DSA	4
C_2	COA	3
C_3	DSY	3

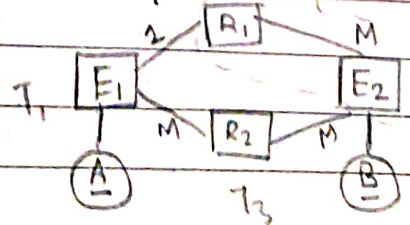
PK = composite

↓

Rollno + Cid

We cannot reduce the number of tables as we cannot merge any tables.

Q What is the minimum no. of tables required to represent the below ER model into Relational model? A) 2 B) 3 C) 4 D) 5



✱ Relational Model :-

⇒ Introduction:

Relational Model represents how data is stored in Relational Databases.
A relational database stores data in the form of relations (tables)

1) Attribute : Attributes are the properties that define a relation.

2) Relation Schema: A relation schema represents name of the relation with its attributes.
Eg: Student (Rollno, Name, Address, Phone, Age) is relation schema for Student.
If a schema has more than 1 relation, it is called Relational Schema.

3) Tuple : Each row in the relation is known as tuple. Ex: 1 Arya --- 9879-- 18

4) Relation Instance: The set of tuples of a relation at a particular instance of time is called as relation instance. It can change whenever there is insertion deletion or updation in the database.

5) Column : Column represents the set of values for a particular attribute.

6) Degree: The number of attributes in the relation is known as degree of the relation.

7) Cardinality : The number of tuples in a relation is known as Cardinality.

⇒ Constraints:

Relational constraints are the restrictions imposed on the database contents and operations.
They ensure the correctness of data in the database.

1) Domain Constraint: Domain constraint defines the domain or set of values for an attribute. It specifies that the value taken by the attribute must be the atomic value from its domain.

2) Tuple Uniqueness Constraint: Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

3) Key Constraint: All the values of primary key must be unique.

4) Entity Integrity Constraint: Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.

5) Referential Integrity Constraint: It specifies that all the values taken by the foreign key must either be available in the relation of primary key or be null.

★ Normalization :-

→ It is a technique to reduce or remove Redundancy (duplicacy) from a table.
→ Two types of redundancies possible : 1) Row level 2) Column level

1) Raw level :

Sid	Sname	Age
1	Anyan	18
2	Jenil	24
1	Anyan	18
3	Neel	19

We can remove this redundancy by just making Sid a primary key.

2) Column level :

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Aryan	C_1	DBMS	F_1	John	50000
2	Jenil	C_2	OTA	F_2	Bob	40000
3	Neel	C_1	DBMS	F_1	John	50000
4	Shree	C_1	DBMS	F_1	John	50000
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Here we can get 3 types of problems : 1) Insertion Anamoly 2) Deletion Anamoly 3) Updation Anamoly

1) Insertion Anamoly:

Let's say we want to add a new course or a new faculty in above table..
We cannot insert because for that we also need a new Sid i.e. new student.

2) Deletion Anamoly:

Let's say we want to delete student 'Jenil' but it will also delete the course & also its faculty.

3) Updation Anamoly:

Let's say faculty 'John' teaches OTA to 'Aryan' but it will also then teach 'OTA' to 'Neel' and 'Shree'.

So, to solve above kinds of anamolies we decompose the tables like

Sid	Sname

Cid	Cname

Fid	Fname	Salary

⇒ Normal Forms:

1) First Normal Form (1NF)

Table should not contain any multivalued attribute.

Student

Roll no.	Name	Course
1	Sai	C/C++
2	Harsh	Java
3	Omkar	C/DBMS

not in 1NF

⇒

Roll no.	Name	Course
1	Sai	C
1	Sai	C++
2	Harsh	Java
3	Omkar	C
3	Omkar	DBMS

✓ in 1 NF

Primary key = Roll no. + Course (composite)

⇒

Roll no.	Name	Course 1	Course 2
1	Sai	C	C++
2	Harsh	Java	NULL
3	Omkar	C	DBMS

✓ in 1NF

← but there may be many courses a person is enrolled only in 1 course then many values will becomes NULL.

decomposition ⇒

Roll no.	Name
1	Sai
2	Harsh
3	Omkar

FK

Roll no.	Course
1	C
1	C++
2	Java
3	C
3	DBMS

✓ in 1NF

⇒ Functional Dependency:

$X \rightarrow Y$ ⇒ X determines Y / Y is determined by X

X: determinant attribute, Y: dependent attribute

	Case 1	Case 2	Case 3	Case 4
Ex:	Sid → Sname	Sid → Sname	Sid → Sname	Sid → Sname
	1 Arya	1 Arya	1 Arya	1 Arya
	1 Arya	2 Arya	2 Jeril	1 Jeril
	✓	✓	✓	✗
	(Same data)	(different data)	(different data)	

Function Dependency are of two types : 1) Trivial Functional Dependency.
2) Non-Trivial Functional Dependency.

1) Trivial Functional Dependency (always true)

$X \rightarrow Y \Rightarrow Y$ must be a subset of X i.e. $X \cap Y \neq \phi$

Ex: i) Sid → Sid ii) Sid Sname → Sid

2) Non-trivial Functional Dependency (check cases)

$X \rightarrow Y \Rightarrow Y$ is not a subset of X i.e. $X \cap Y = \phi$

Ex: i) Sid → Sname

Properties of Functional Dependency:

1) Reflexivity : If Y is subset of X then $X \rightarrow Y$ (Trivial)

2) Augmentation : If $X \rightarrow Y$, then $XZ \rightarrow YZ$

3) Transitive : If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

4) Union : If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

5) Decomposition: If $X \to YZ$ then $X \to Y$ and $X \to Z$

But ~~$XY \to Z$~~ then $X \to Z, Y \to Z$ ✗ ✓

6) Pseudo transitivity: If $X \to Y$ and $WY \to Z$ then $WX \to Z$

7) Composition: If $X \to Y$ and $Z \to W$ then $XZ \to YW$

⇒ Closure Method:

→ Closure Method is used to find all the candidate keys in a relation.

Ex: $R(A,B,C,D)$ FD: $\{A \to B, B \to C, C \to D\}$

(trivial)

$(A)^+ = \{A,B,C,D\}$ ✓ candidate key — $CK = \{A\}$

$(B)^+ = \{B,C,D\}$ ✗ candidate key — Prime attributes = $\{A\}$

$(C)^+ = \{C,D\}$ ✗ candidate key — Non-Prime attributes = $\{B,C,D\}$

$(D)^+ = \{D\}$ ✗

$(AB)^+ = \{A,B,C,D\}$ ✗ because A is already a CK and using it with B makes a S

Ex: $R(A,B,C,D)$ FD: $\{A \to B, B \to C, C \to D, D \to A\}$

$(A)^+ = \{A,B,C,D\}$ ✓ ∴ $CK = \{A,B,C,D\}$

$(B)^+ = \{B,C,D,A\}$ ✓

$(C)^+ = \{C,D,A,B\}$ ✓ Prime Attribute = $\{A,B,C,D\}$

$(D)^+ = \{D,C,B,A\}$ ✓ Non-Prime Attribute = ϕ

Ex: $R(A,B,C,D,E)$ FD: $\{A \to B, BC \to D, E \to C, D \to A\}$

Another Method: Right hand side of all FD's = $\{B,D,C,A\}$

$(E)^+ = \{E,C\}$ ✗ candidate key

So we must include E with other attributes to find CK

Now check that attribute which determines E x

$(AE)^+ = \{A, E, B, C, D\}$ ✓

Now check that attribute which determines A. : $(DE)^+ = \{D, E, A, B, C\}$ ✓

Now check that attributes which determines D : $(BE)^+ = \{B, E, C, D, A\}$ ✓

$(CE)^+ = \{C, E\}$ x

$\therefore$ CK = {AE, DE, BE}

$\therefore$ Prime attributes = {A, D, B, E}

$\therefore$ Non-Prime attributes = {C}

⇒ Full-Functional Dependency v/s Partial-Functional Dependency

If $X \to Y$ & $X - \{A\} \to Y$ ⇒ Partial Dependency
(A: any attribute)

If $X \to Y$ & $X - \{A\} \not\to Y$ ⇒ Full / Total Dependency

Ex: If $AB \to C$ & $A \to C$ ⇒ Partial Dependency ⇒ Condition: LHS should be a proper subset of CK and RHS should be a non-prime attribute

If $AB \to C$ & $A \not\to C$ and $B \not\to C$ ⇒ total Dependency

Proper subset → non-trivial

2) Second Normal Form (2NF): (LHS should not be a proper subset of candidate / super key & RHS should not be non-prime attribute)

FD honi chahiye

→ Table / Relation must be in 1NF

→ All the non-prime attributes should be fully functional dependent on candidate key.

Ex: Customer :

CustomerID	Store ID	Location
1	1	Delhi
1	3	Mumbai
2	1	Delhi
2	2	Banglore
4	3	Mumbai

Candidate Key: Customer Id + Store Id

Prime attributes: {Customer Id, store Id}

Non-Prime attributes: {Location}

Let A → customer Id, B → store Id, (+ the

AB → C ✓

A → C x B → C ✓ ∴ Partially dependent

∴ Not in 2NF

∴ So we decompose :

Customer Id	Store Id
1	1
1	3
2	1
3	2
4	3

✓ 2NF

Store Id	Location
1	Delhi
2	Banglore
3	Mumbai

✓ 2NF

Q $R(A,B,C,D,E,F)$ FD: $\{C \to F, E \to A, EC \to D, A \to B\}$ Is R in 2NF if not decom

RHS of all FDs = $\{F, A, D, B\}$

$(C)^+ = \{C, F\}$ × candidate key

$(E)^+ = \{E, A, B\}$ × candidate key

$(EC)^+ = \{E, C, D, A, B, F\}$ ✓ candidate key

Now check attributes which determines EC ×

Now check attributes which determines C ×

Now check attributes which determines E ×

∴ CK = $\{EC\}$

∴ Prime attributes = $\{C, E\}$

∴ Non-Prime attributes = $\{A, B, D, F\}$

Part of CK is determining Non-Prime, part of CK

FDs : $C \to F$ ~~X~~ → partial dependency

$E \to A$ ~~X~~ → partial dependency

$EC \to D$ ~~X~~ → partial dependency (→ full dependency)

$A \to B$

∴ not in 2NF

used

So, we decompose $R_1(\underline{C}, F)$

$R_2(\underline{E}, A, B)$

$R_3(\underline{E, C}, D)$

~~$R_4(E, C)$~~

3) Third Normal Form (3NF): (LHS must be a CK/SK or RHS is a prime attribute)

→ table or relation must be in 2NF.
→ There should be no transitive dependency in the relation

Ex:

Roll no.	State	City
1	Punjab	Mohali
2	Haryana	Ambala
3	Punjab	Mohali
4	Haryana	Ambala
5	Bihar	Patna

FD: {Roll no → state ✓, state → city ×} why!
due to transitive dependency we will get Roll no. → city which is true

Candidate keys: {Roll no.}
Prime attributes = {Roll no.}
Non-Prime attributes = {state, city}

∴ it is not in 3NF also (it is not in 2NF)
So, we decompose
↳ R1 (state, city)
R2 (Roll no, state)
R3 (Roll no)

Q R(A,B,C,D) FD: {AB→C, C→D}

RHS of all FDs = {C,D}
(A)+ = {A} × candidate key
(B)T = {B} ×
(AB)T = {AB,C,D} ✓

∴ CK = {AB}
∴ Prime attributes = {A,B}
∴ Non-Prime attributes = {C,D}

AB → C ✓
C → D × why! AB → D ✓
but it should not have transitive dependency
∴ not in 3NF

So, we decompose → R1 (A,B) R1 (C,D)
R2 (R2 (A,B,C)

Q R(A,B,C,D) FD: {AB→CD, D→A}

RHS of all FQ = {C,D,A}
(B)+ = {B} × candidate key
(AB)T = {A,B,C,D} ✓
(BD)T = {B,D,A,C} ✓

CK = {AB, BD}
Prime = {A,B,D}
Non-Prime = {C}

AB → CD ✓
D → A ✓
∴ in 3NF

Q. R(A,B,C,D,E,F,G,H,I,J) F.D.: AB→C, A→DE, B→F, F→GH, D→IJ

RHS of all the FDs: {C,D,E,F,G,H,I,J}

$(A)^+$ = {A,D,E,I,J} × candidate key

$(B)^+$ = {B,F,G,H} ×

$(AB)^+$ = {A,B,C,D,E,F,G,H,I,J} ✓

∴ CK = {AB}

∴ Prime attributes = {A,B}, Non-Prime attributes = {C,D,E,F,G,H,I,J}

CK → AB → C ✓ ∴ not in 3NF

CK → A → DE ×

CK → B → F ×

F → GH ×

D → IJ ×

X

2 NF

so, we decompose → R_1 (A,D,E,I,J)

- → R_{11} (A,D,E)
- → R_{12} (D,I,J)

→ R_2 (B,F,G,H)

- → R_{21} (B,F)
- → R_{22} (F,G,H)

→ R_3 (A,B,C)

4) Boyce Codd Normal Form (BCNF): (LHS of each FD should be a CK or SK)

→ table or relation must ~~have dependency on primary / candidate key~~ Super be in 3NF.

Ex: Student

Rollno.	Name	Voter_id	Age
1	Ravi	K0123	20
2	Varun	M034	21
3	Ravi	K768	23
4	Rahul	D286	21

CK : {Rollno, Voter_id}

FD : Roll no → name ✓

Rollno → voterid ✓

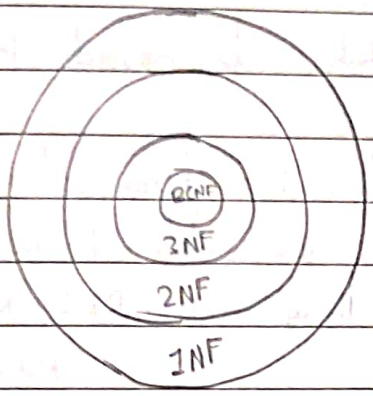
Voter id → age ✓

Voterid → Rollno ✓

∴ in BCNF

→ 3NF, 2NF, 1NF always ensures 'Dependency Preserving decomposition' but not BCNF.

→ All the normal forms ensures lossless decomposition.



⇒ Decomposition of a Relation:

The process of breaking up or dividing a single relation into two or more sub relations is called the decomposition of a relation.

Properties of Decomposition:

1) Lossless Decomposition: It ensures

→ No information is lost from the original relation during decomposition and when the sub relations are joined back, the same relation is obtained that was decomposed.

2) Dependency Preservation: It ensures

→ None of the functional dependencies the hold on the original relation are lost.
→ The sub relations still hold or satisfy the functional dependencies of the original relation.

Types of Decomposition:

1) Lossless Join Decomposition:

→ Consider there is a relation R which is decomposed into sub relations R1, R2, ..., Rn.
→ This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
→ For lossless join decomposition, we always have – $R1 \bowtie R2 \bowtie R3 \ldots \bowtie Rn = R$
(⋈ : Natural Join)

2) Lossy Join Decomposition

→ Consider there is a relation R which is decomposed into sub relations R1, R2, ..., Rn.
→ This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
→ For lossy join decomposition, we always have – $R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n \supset R$
(⋈ : Natural Join)

Note: 3 NF always ensures 'Dependency Preserving decomposition' but not in BCNF. Both 3NF & BCNF ensures lossless decomposition.

Ex: R(A, B, C, D) FD: {AB → CD, D → A}

RHS of FDs: {A, C, D}
$(B)^+ = \{B\}$ x CK

$(AB)^+ = \{A, B, C, D\}$ ✓ CK
$(DB)^+ = \{D, B, A, C\}$ ✓ CK

∴ CK = {AB, BD}
PA = {A, B, D}
NPA = {C}

$AB \rightarrow CD$ ✓ 3NF ✓ BCNF

$D \rightarrow A$ ✓ 3NF × BCNF

∴ R is in 3NF but not in BCNF

∴ we decompose R

$R(A,B,C,D)$

$R_1(D,A)$ | $R_2(B,C)$ → $R_2(D,B,C)$ (to ensure lossless decomposition we must include a common CK)

$D \rightarrow A$

$(D)^+ = (D,A)$

$D \rightarrow CK$

$(B)^+ = \{B\}$ × $(BC)^+ = \{B,C\}$

$(C)^+ = \{C\}$ × $(BD)^+ = \{B,D,A,C\}$ ✓

$(D)^+ = \{D\}$ ×

∴ $BD \rightarrow C$

∴ after decomposition FD $AB \rightarrow CD$ is lost

∴ BCNF may not ensure 'Dependency Preserving decomposition'.

Ex:

R

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

$R_1(AB)$, $R_2(B,C)$

R_1

A	B
1	2
2	2
3	3

R_2

B	C
2	1
2	2
3	2

R' $R'(ABC)$ ⇒ Natural Join

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

Compare

∴ Lossy Decomposition

But we always want lossless decomposition.

→ These extra tuples are called Spurious Tuples

So, to achieve lossless decomposition:

Common Attribute must be a CK or SK of either R_1 or R_2 or both R_1 & R_2.

∴ $R(A,B,C) \rightarrow R_1(A,B)$, $R_2(A,C)$ ⇒ Lossless Decomposition

⇒ Minimal Cover (Irreducible Set of FDs):

Q For the following functional dependencies, find the correct Minimal Cover.
$\{A \to B, C \to B, D \to ABC, AC \to D\}$

a) $A \to B, C \to B, D \to A, AC \to D$

b) $A \to B, C \to B, D \to C, AC \to D$

c) $A \to BC, D \to CA, AC \to D$

d) ✓ $A \to B, C \to B, D \to AC, AC \to D$

Step-1: RHS should be single

$A \to B$ ✓

$C \to B$ ✓

$D \to ABC \Rightarrow D \to A, D \to B, D \to C$

$AC \to D$ ✓

Step-2: Remove redundant functional dependency

$A \to B$ remove

$(A)^+ = \{A\}$ ∴ $A \to B$ ✓ (not remove)

$C \to B$ remove

$(C)^+ = \{C\}$ ∴ $C \to B$ ✓ (not remove)

$D \to A$ remove

$(D)^+ = \{D, B, C\}$ ∴ $D \to A$ ✓ (not remove)

$D \to B$ remove

$(D)^+ = \{D, A, C, B\}$ ∴ $D \to B$ ✗ (remove)

$D \to C$ remove

$(D)^+ = \{D, A, B\}$ ∴ $D \to C$ ✓ (not remove)

$AC \to D$ remove

$(AC)^+ = \{A, C, B\}$ ∴ $AC \to D$ ✓ (not remove)

Step-3: LHS should be single

$AC \to D$ Can we write $C \to D$ iff $(C)^+ = \{C, A\}$ ∴ ✗

But $(C)^+ = C$

Can we write $A \to D$ iff $(A)^+ = \{A, C\}$ But $(A)^+ = AB$ ∴ ✗

Q R(ABCDEF) FD: {AB→C, C→DE, E→F, F→A}, check the highest normal form.

Candidate keys: RHS of all FDs = {C, D, E, F, A}

$(B)^+ = \{B\}$ × CK

$(AB)^+ = \{A, B, C, D, E, F\}$ ✓

$(FB)^+ = \{F, B, A, C, D, E\}$ ✓

$(EB)^+ = \{E, B, F, A, C, D\}$ ✓

$(CB)^+ = \{C, B, D, E, F, A\}$ ✓

∴ CK = {AB, FB, EB, CB}

Prime attributes = {A, B, C, E, F}

Non-Prime attributes = {D}

1) BCNF

AB→C, C→DE, E→F, F→A

✓ × × ×

∴ not in BCNF

2) 3NF

AB→C, C→DE, E→F, F→A

✓ × ✓ ✓

∴ not in 3NF

∴ R is only in 1 NF

3) 2NF

AB→C, C→DE, E→F, F→A

✓ × ✓ ✓

∴ not in 2 NF

Condition for PD: LHS must be an proper subset of any CK and RHS must be a non-PA

Also decompose it.

Q R(ABCDEF) FD: {AB→C, C→D, C→E, E→F, F→A} check all the NFs & if not decompose it.

CK = {AB, FB, EB, CB} Prime Attributes = {A, B, C, E, F} Non-Prime Attributes = {D}

1) 2 NF

$AB \xrightarrow{F} C,\ C \xrightarrow{T} D,\ C \xrightarrow{T} E,\ E \xrightarrow{F} F,\ F \xrightarrow{T} A\}$ (F)

✓ ✗ ✓ ✓ ✓

∴ R is not in 2 NF

∴ we decompose

R(ABCDEF)

$R_1(C,D)$

$C \to D$

$(C)^+ = \{C,D\}$ ✓ CK

CK = {C}

PA = {C}

NPA = {D}

$C \xrightarrow{F} D$ ✓ 2NF

∴ R_1 is in 2NF

$C \to D$ ✓ 3NF

∴ R_1 is in 3NF

$C \to D$ ✓ BCNF

∴ R_1 is in BCNF

$R_2(A,B,E,F)$] to prove lossless decomposition

$R_2(A,B,C,E,F)$

$AB \to C$ $(B)^+ = \{B\}$ ✗ CK

$C \to E$ $(AB)^+ = \{A,B,C,E,F\}$ ✓

$E \to F$ $(FB)^+ = \{$ $\}$ ✓

$F \to A$ $(EB)^+ = \{$ 3 ✓

$(CB)^+ = \{$ 3 ✓

CK = {AB, FB, EB, CB}

PA = {A, B, C, E, F}

$AB \to C$ ✓ $C \to E$ ✓ $E \to F$ ✓ $F \to A$ ✓

∴ R_2 is in 2NF

$AB \to C$ ✓ $C \to E$ ✓ $E \to F$ ✓ $F \to A$ ✓

∴ R_2 is in 3NF

$AB \to C$ ✓ $C \to E$ ✗ $E \to F$ ✗ $F \to A$ ✗

∴ R_2 is not in BCNF

∴ we decompose

$R_2(A,B,C,E,F)$

$R_{21}(A,B,D)$ → $R_{21}(B,C,E,F)$

$C \to E$

$(B)^+ = B$ ✗ $E \to F$

$(C)^+ = C,E,F$

$(BC)^+ = \{B,C,E,F\}$ CK

$R_{22}(C,E)$ $C \to E$ $C^+ = C,E$ ✓ CK ✓ BCNF

$R_{23}(E,F)$ $E \to F$ $E^+ = E,F$ ✓ CK ✓ BCNF

$R_{24}(F,A)$ $F \to A$ $F^+ = F,A$ ✓ CK ✓ BCNF

Q Which of the following schemas are in 3NF but not in BCNF.

Schema 1: Registration (rollno., courses) FD: {roll no → courses} ✓3NF ✓BCNF

Schema 2: Registration (rollno, Course id, email) FD: {rollno course_id → email ✓3NF ✗BCNF, email → rollno}

Schema 3: Registration (rollno, Courseid, marks, grade) FD: {rollno courseid → marks grade, marks → grade ✗3NF ✗BCNF}

Schema 4: Registration (rollno, course id, Credit) FD: {rollno. Courseid → Credit, Courseid → Credit} ✗3NF ✓BCNF

Q Let R(A,B,C,D) with functional dependencies: {A→B, B→C, C→D, D→B}. R is decomposed into $R_1(A,B)$, $R_2(B,C)$, $R_3(B,D)$. Determine whether the decomposition was 'Dependency Preserving' or not.

$FD_{R_1} \cup FD_{R_2} \cup FD_{R_3} = FD_R$

$R_1(A,B)$	$R_2(B,C)$	$R_3(B,D)$
A→B ✓	B→C ✓	B→D ✓
B→A ✗	C→B ✓	D→B ✓
$(A)^+ = \{A,B,C,D\}$	$(B)^+ = \{B,C,D\}$	$(B)^+ = \{B,C,D\}$
$(B)^+ = \{B,C,D\}$	$(C)^+ = \{C,D,B\}$	$(D)^+ = \{D,B,C\}$

$R_1 \cup R_2 \cup R_3$	R
A→B ✓	A→B ✓
B→C ✓	B→C ✓
C→B ✓	C→D ✓
B→D	D→B ✓
D→B ✓	

Preserved

⇓

~~$(A)^+ = \{A,B,C\}$~~

$(C)^+ = \{C,B,D\} \Rightarrow C \to D$

~~$(B)^+ = \{B,D\} \Rightarrow \perp$~~

∴ Decomposition → Dependency Preserving

Q Let R(A,B,C,D) with functional dependencies {AB → CD, D → A}. R is decomposed into R1(A,D) and R2(B,C,D). Determine whether the decomposition is 'Dependency Preserving' or not.

$R_1(A,D)$

$A \to D$ ×

$D \to A$ ✓

$(A)^+ = \{A\}$

$(D)^+ = \{D, A\}$

$R_2(B,C,D)$

$B \to CD$ ×

$C \to BD$ ×

$D \to BC$ ×

$CD \to B$ ×

$BD \to C$ ✓

$BC \to D$ ×

$(B)^+ = \{B\}$

$(C)^+ = \{C\}$

$(D)^+ = \{D, A\}$

$(CD)^+ = \{C, D, A\}$

$(BD)^+ = \{B, D, A, C\}$

$(BC)^+ = \{B, C\}$

$R_1 \cup R_2$

$D \to A$

$BD \to C$

R

$AB \to CD$ ×

$D \to A$ ✓

∴ Decomposition → × Dependency Preserving

$(BD)^+ = \{B, D, C, A\}$

$(AB)^+ = \{A, B\}$

★ Relational Algebra :-

Relational Algebra is procedural query language / formal query language which takes a relation as an input and generates a relation as an output.

Operators

Basic Operator	Derived Operator
Projection (π)	Join ($\bowtie$)
Selection (σ)	Intersection ($\cap$) : $(X \cap Y = X - (X - Y))$
Cross Product ($\times$)	Division ($/, \div$)
Union ($\cup$)	
Rename (ρ)	
Set Difference ($-$)	

1) Projection (π) (to get distinct records of a attribute / column)

Student

Rollno.	Name	Age
1	A	20
2	B	21
3	A	19

Query: Retrieve rollno from Student = $\pi_{Rollno}(Student)$

Rollno.
1
2
3

Projection → distinct *

Ex: $\pi_{Name}(Student)$ ⇒

Name
A
B

$\pi_{Name, Age}(Student)$ ⇒

Name	Age
A	20
B	21
A	19

2) Select (σ) (to get a tuple / record / row)

Same example

Query: Retrieve the name of the student whose rollno = '2'

⇒ $\pi_{Name}(\sigma_{Rollno=2}(Student))$

3) Gross / Cartesian Product

at least 1 col must be same — doesn't require common attribute

R_1			R_2		
A	B	C	C	D	E
1	2	3	3	4	5
2	1	4	2	1	2

Degree = 3, Cardinality = 2

Degree = 3, Cardinality = 2

$R_1 \times R_2$

A	B	C	C	D	E
1	2	3	3	4	5
1	2	3	2	1	2
2	1	4	3	4	5
2	1	4	2	1	2

Degree = 3 + 3 = 6

Cardinality = 2 × 2 = 4

4) Rename (ρ)

$\sigma_{Dno=4}(Employee) \xrightarrow{Rename} Dno4EMP \Rightarrow \rho_{Dno4EMP}(\sigma_{Dno=4}(EMPLOYEE))$

name $\xrightarrow{Rename}$ N
salary $\xrightarrow{Rename}$ S
$\Rightarrow \rho_{(N,S)}(\Pi_{name,salary}(EMPLOYEE))$

5) Union Operation (U)

order of attributes in both the table is important.

Rules: $(R_1 \cup R_2)$

1) Degree of R_1 = Degree of R_2
2) Domain of any attributes in both relation which are same that only can be unioned. same domain so, order is necessary.

Student

Rollno.	Name
1	A
2	B
3	C

Employee

Empno.	Name
7	E
1	A

Student ∪ Employee

Rollno.	Name
1	A
2	B
3	C
7	E

Cardinality of Union ≤ Sum of Cardinalities of relations

Query: Retrieve the name of either students, employees or both

⇒ $\Pi_{Name}(Student) \cup \Pi_{Name}(Employee)$

⇓

Name
A
B
C
E

6) Set Difference (−)

Rules of Union must be followed.

→ Some Example

Query: Retrieve name of students who are not employees

⇒ $\Pi_{Name}(Student) - \Pi_{Name}(Employee)$ ⇒

Name
B
C

7) Division Operation (derived from all the above operators)

Enrolled		Course
Sid	Cid	Cid
S_1	C_1	C_1
S_2	C_1	C_2
S_1	C_2	
S_3	C_2	

Query: Retrieve Sid of students who are enrolled in every/all course.

⇒ Enrolled (Sid, Cid) / Π_{Cid}(Cid) ⇒ It results Sid for that there should be a tuple <Sid, Cid> for every Cid of Course ⇒ S_1

Enrolled(Sid, Cid) ÷ Course(Cid) =

$$\Pi_{Sid}(Enrolled) - [\Pi_{Sid}\{\Pi_{Sid}(Enrolled) \times \Pi_{Cid}(Course)\} - \Pi_{Sid,Cid}(Enrolled)]$$

S_1, S_2, S_3 → S_2, S_3 → S_1

S_1	C_1			S_1	C_1		
S_1	C_2			S_2	C_1		
S_2	C_1	−		S_1	C_2		
S_2	C_2			S_2	C_2		
S_3	C_1						
S_3	C_2						

★ Tuple Calculus:

→ Tuple Relational Calculus is a non-procedural query language unlike relational algebra.

→ In tuple calculus, a query is expressed as $\{t \mid P(t)\}$

t → resulting tuples

$P(t)$ → known as Predicate and these are the conditions that are used to fetch t. Thus, it generates set of all tuples t, such that predicate $P(t)$ is true for t

→ $P(t)$ may have various conditions logically combined with OR (V), AND (∧) and NOT (¬) etc.

→ It also uses quantifiers: $\exists t \in r(P(t))$ → "there exists" a tuple t in relation r such that predicate $P(t)$ is true

$\forall t \in r(P(t))$ → $P(t)$ is true "for all" tuples in relation r

Q Supplier (Sid, Sname, Address)
Parts (Pid, Pname, Color)
Catalog (Sid, Pid)

1) Write a query in SQL to display Sname of suppliers? {t.Sname | Supplier(t)}

2) Write a query in SQL to display Pname of parts whose colon is Red?
{t.Pname | Parts(t) AND Colon='Red'}

3) Write a query in SQL to display Sid of suppliers who supplied some parts?
{t.Sid | Catalog(t)}

4) Write a query in SQL to display Sname of suppliers who supplied some parts?
{t.Sname | Supplier(t) AND (∃d)(Catalog(d)) AND t.Sid = d.Sid}
(t: free variable, d: bonded variable)

5) Write a query in SQL to display Sname of supplier who supplied som Red colon parts? {t.Sname | Supplier(t) AND (∃d)(Catalog(d)) AND (∃e)(Parts(e)) AND t.Sid = d.Sid AND e.color = 'Red' AND d.Pid = e.Pid}

⇒ Joins (Cross-Product + condition):

atleast one attribute / column must be common

1) Natural Join:

Employee

Eno.	Ename	Address
1	Ram	Delhi
2	Varun	Pune
3	Ravi	Pune
4	Amit	Delhi

Department → FK

Deptno.	Name	Eno.
D_1	HR	1
D_2	IT	2
D_3	MRKT	4

Query: Find the employee names who is working in a department

$\Pi_{Ename}(Employee \bowtie_{Eno = Eno} Department)$

Ename
Ram
Varun
Amit

Can have different names

Natural Join → by default joins attributes which are common to both the tables

2) Equi Join:

Employee

Eno.	Ename	Address
1	Ram	Delhi
2	Varun	Mumbai
3	Ravi	Mumbai
4	Amit	Delhi

Department

Deptno.	Location	Eno.
D_1	Delhi	1
D_2	Pune	2
D_3	Patna	4

Query: Find the name of employees who worked in a department having location same as their address?

$\Pi_{Ename}(Employee \bowtie_{Employee.Eno = Dept.Eno \text{ AND } Address = Location} Department)$

Ename
Ram

3) Self Join:

Study

Sid	Cid	Since
S_1	C_1	2016
S_2	C_2	2017
S_1	C_2	2017

Query: Find id of student who is enrolled in atleast two courses.

$\Pi_{S_1.Sid}(Study\ S_1 \bowtie_{S_1.Sid = S_2.Sid \text{ AND } S_1.Cid \neq S_2.Cid} Study\ S_2) \Rightarrow$

Sid
S_1

4) Left Outer Join: (Inner Join + Remaining tuples of left table)

It gives the matching rows and the rows which are in left table but not in right table.

Employee

Eno.	Ename	Deptno.
E_1	Varun	D_1
E_2	Amrit	D_2
E_3	Ravi	D_3
E_4	Nitin	–

Department

Deptno.	Dname	Location
D_1	IT	Delhi
D_2	HR	Hyderabad
D_3	Finance	Pune

$\Pi_{Eno., Ename, Dname, Location}$ (Employee ⟕ $_{Emp.Deptno = Dept.Deptno.}$ Department)

⇓

Eno.	Ename	Dname	Location
E_1	Varun	IT	Delhi
E_2	Amrit	HR	Hyderabad
E_3	Ravi	IT	Delhi
E_4	Nitin	–	–

5) Right Outer Join (Inner Join + Remaining tuples of right table):

Employee

Eno.	Ename	Dept no.
E_1	Varun	D_1
E_2	Amrt	D_2
E_3	Ravi	D_3

Department

Deptno.	Dname	Location
D_1	IT	Delhi
D_2	HR	Mumbai
D_3	Financ	Pune
D_4	MRKT	Noida

$\Pi_{Eno, Ename, Dname, Locati.}$ (Employee ⟖ $_{Emp.Dno = Dept.Dno.}$ Department)

⇓

Eno.	Ename	Dname	Locati..
E_1	Varun	IT	Delhi
E_2	Amrit	HR	Mumbai
E_3	Ravi	Financ	Pune
–	–	MRKT	Noida

6) Full Outer Join

Full Outer Join (⟗) = Left Outer Join (⟕) ∪ Right Outer Join (⟖)

→ These files are organized logically as a sequence of records and reside permanently on disks. Each file is divided into fixed-length storage units known as Blocks.
→ These blocks are the units of storage allocation as well as data transfer.
→ Usually, the record size is smaller than the block size, but for large data items such as images, the size can vary. For accessing the data quickly, it is required that one complete record should reside in one block only.

★ File Structures and Organization :-

→ A database consist of a huge amount of data. The data is grouped within a table in RDBMS, and each table have related records. A user can see that the data is stored in form of tables, but in actual this huge amount of data is stored in physical memory in form of files.

→ A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.

→ Storing the files in certain order is called file organization.
→ File Structure refers to the format of the label and data blocks and of any logical record.

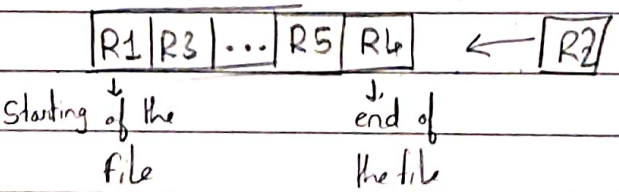
⇒ Types of File Organization:

1) Sequential File Organization:

→ The easiest method for file organization is Sequential method.
→ In this method the file are stored one after another in a sequential manner.

a) Pile File Method

→ This method is quite simple, in which we store the records in a sequence i.e. one after other in the order in which they are inserted into tables.

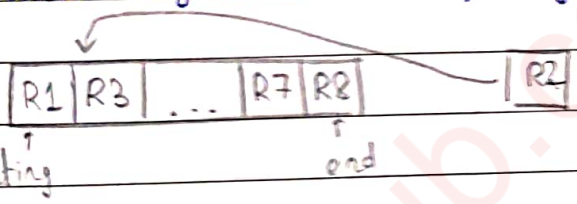


→ Let the R1, R3 and so on upto R5 and R6 be four records in the sequence. Here, rows are nothing but a rows in any table. Suppose a new record R2 has to be inserted in the sequence, then it is simply placed at the end of the file.

b) Sorted File Method

→ In this method, as the name itself suggest wherever a new record has to be inserted, it is always inserted in a sorted (ascending or descending) manner.

→ Sorting of records may be based on primary key or any other key.



→ Let us assume that there is a preexisting sorted sequence of four records R1, R3 and so on upto R7 and R8. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file and then it will sort the sequence.

Advantages:

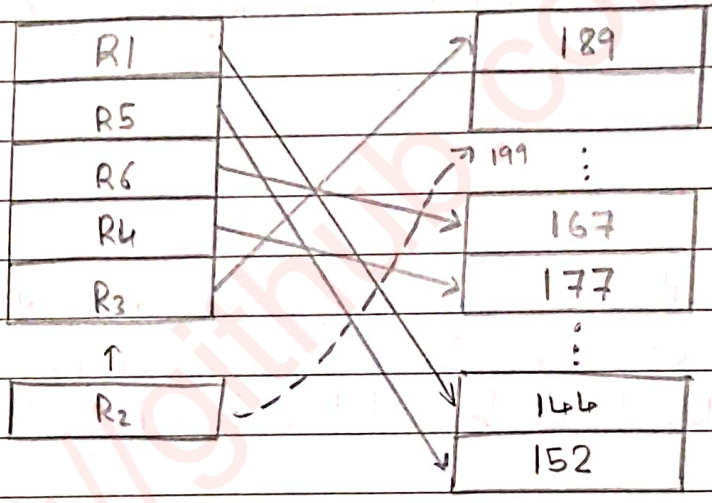
→ Fast and efficient method for huge amount of data.
→ Simple design.
→ Files can be easily stored in magnetic tapes i.e. cheaper storage mechanism.

Disadvantages:

→ Time wastage as we cannot jump on a particular record that is required, but we have to move in a sequential manner.
→ Sorted file method is inefficient as it takes time and space for sorting records.

2) Heap File Organization:

→ Heap File Organization works with data blocks.
→ In this method records are inserted at the end of the file, into the data blocks.
→ No sorting or ordering is required in this method.
→ If a data block is full, the new record is stored in some other block, here the other block need not be the very next data block, but it can be any block in the memory.
→ It is the responsibility of DBMS to store and manage the new records.



→ Suppose we have five records in the heap R1, R5, R6, R4 and R3 and suppose a new record R2 has to be inserted in the heap then, since the last data block i.e. data block 3 is full it will be inserted in any of the data blocks selected by DBMS, lets say data block 1.

→ If we want to search, delete or update data in heap file organization then we will have to traverse the data from the beginning of the file till we get the requested record. Thus if the database is very huge, searching, deleting or updating the record will take a lot of time.

Advantages:

→ Fetching and retrieving records is faster than sequential record but only in case of small databases.

→ When there is a huge number of data that needs to be loaded into the database at a time, then this method of file organization is best suited.

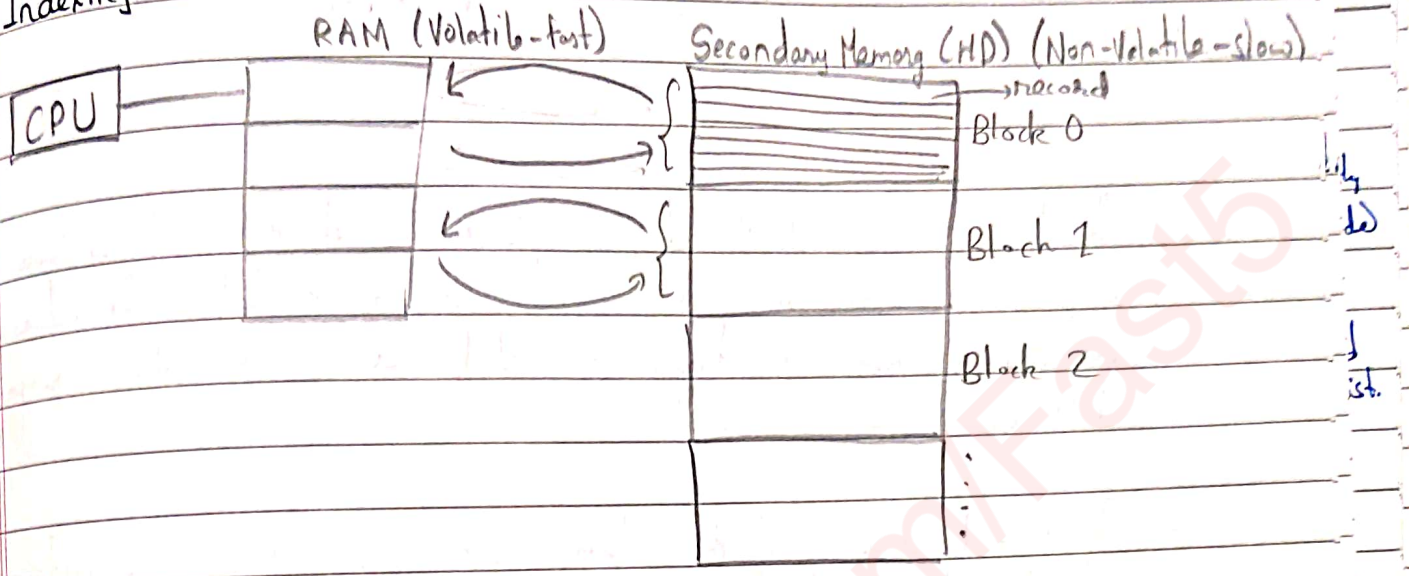
Disadvantages:

→ Problem of unused memory blocks.

→ Inefficient for large databases.

3) Hash File Organization

⇒ Indexing :



→ Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

→ The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Q Consider a Hard disk in which block size = 1000 bytes, each record is of size 250 bytes. If the total no. of records are 10000 and the data is entered in Hard disk without any order (unordered). What is avg time complexity to search a record from HD? ⇒ Without Indexing

avg No. of records in a block (Blocking Factor) $= \left\lfloor \frac{1000}{250} \right\rfloor = 4$

Total no. of blocks required = 4 records → 1 block

10000 records → $\frac{10000}{4} = 2500$ blocks

Since it is unordered → linear search : Best = 1

Worst = 2500

If ordered → binary search : Avg $= \lceil \log_2(2500) \rceil$ Avg $= \frac{2500}{2} = 1250$

$= 12$

Index Structure:

anchor attribute
or
search attribute — Block pointer
or — or

Search Key	Data Reference

→ The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed quickly.

→ The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

→ Data Reference is also called Block Pointer because it uses block-based addressing.

Q Consider a Hard disk in which block size = 1000 bytes, each record is of size = 250 bytes. If total no. of records are 10000 and the data is entered in Hard disk without any order (Unordered). What is Avg time complexity to search a record from Index table if size of entry in index table = 20 bytes (10 bytes (Keys) + 10 bytes (Pointers)).

No. of records in a block = $\left\lfloor \frac{1000}{250} \right\rfloor = 4$

Total no. of blocks required : 4 records → 1 block

10000 records → $\frac{10000}{4} = 2500$ blocks

Note: Block Size in HD = Block size in index table = 1000 bytes

∴ No. of records in a block in index table = $\left\lfloor \frac{1000}{20} \right\rfloor = 50$

If Ordered ⇒ Sparse Indexing ⇒ 50 records → 1 block

2500 records → $\frac{2500}{50}$ = 50 blocks required

∴ Avg time Complexity of Searching = $\lceil \log_2 50 \rceil + 1 = 6 + 1 = 7$

+1 → to search record in the block

If unordered ⇒ Dense Indexing ⇒ 50 records → 1 block

10000 records → $\frac{10000}{50}$ = 200 blocks required

∴ Avg time Complexity of searching = $\lceil \log_2 200 \rceil + 1 = 8 + 1 = 9$

Types of Indexing:

ordered file	Primary Index (sparse)	Clustered Index (sparse & dense)
unordered file	Secondary Index (dense)	Secondary Index (dense)

Indexing → Primary (Sparse), Secondary (Dense), Clustering

Key (✗ repeat) Non-Key (✓ repeat)

So, to have ordered index file mapped to unordered file (of same size) → BS

1) Primary Indexing:

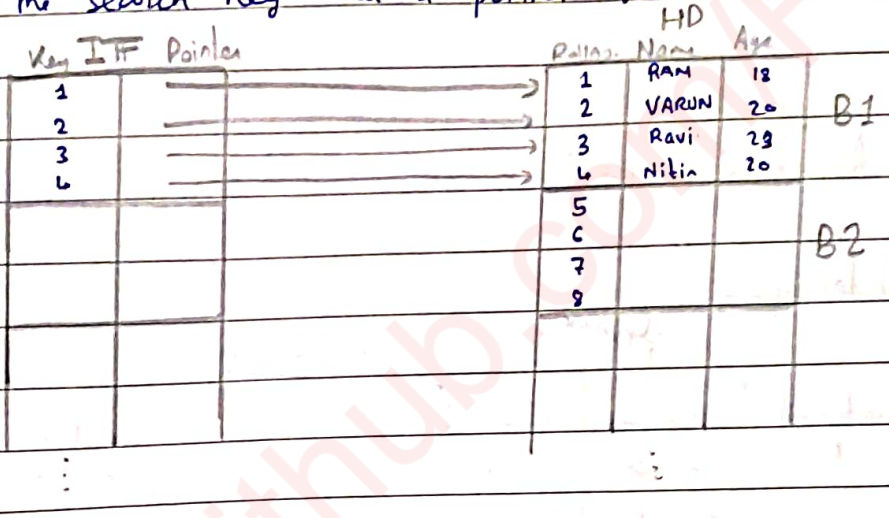
→ If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.

→ As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.

It can be further classified into two:

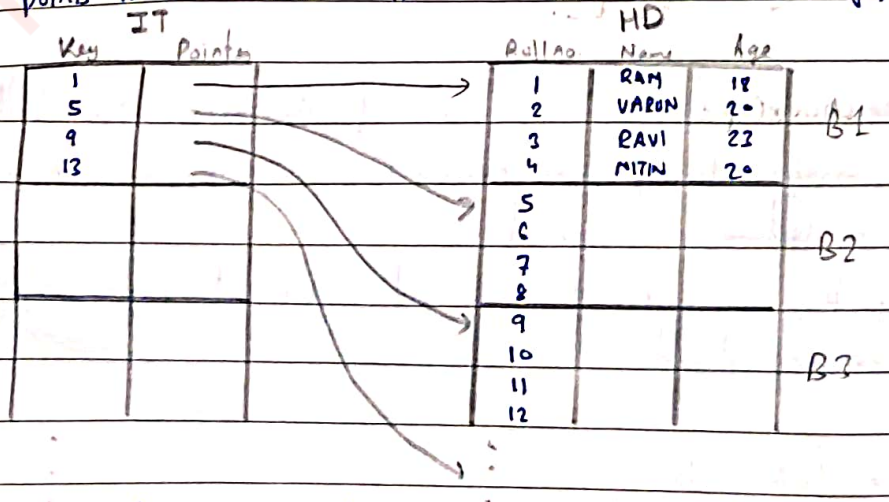
a) Dense Index (comes in

→ The dense index contains an index record for every search key value in the data file. It makes searching faster.
→ In this, the number of records in the index table is same as the number of records in the main table.
→ It needs more space to store index records itself. The index records have the search key and a pointer to the actual record on the disk.



b) Sparse Index

→ In the data file, index record appears only for a few items. Each item points to a block.
→ In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



∴ No. of entries in index table = no. of blocks in Hard disk

Search → $(\log_2 N + 1)$ (+1 to scan the block in the index table) N → no. of blocks in index table

dense indexing as has value index file mai aaraha

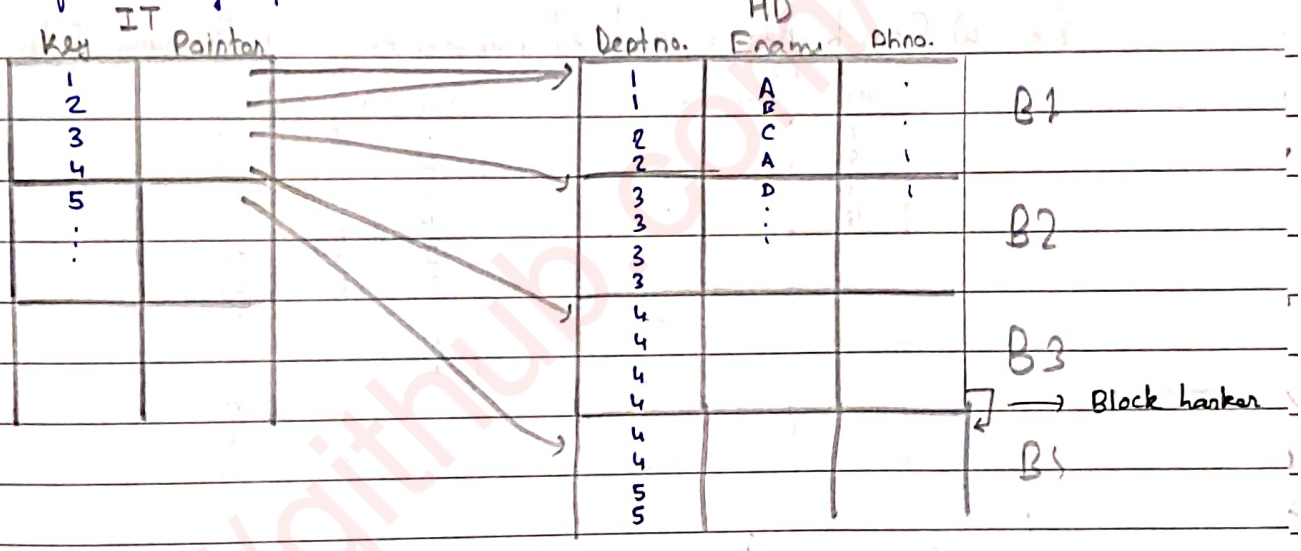
2) Clustering Indexing: (always sparse index)

↳ har ek record in HD is not present in index table

→ A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.

→ In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.

→ The records which have similar characteristics are grouped, and indexes are created for these group.



Worst Case Search → $(\log_2 N + 1 + 1)$, N → no. of blocks in index table

↓ to search the record in the block

↘ if it is present in the next block

3) Secondary Indexing:

→ In the sparse indexing, as the size of the table grows, the size of mapping also grows.

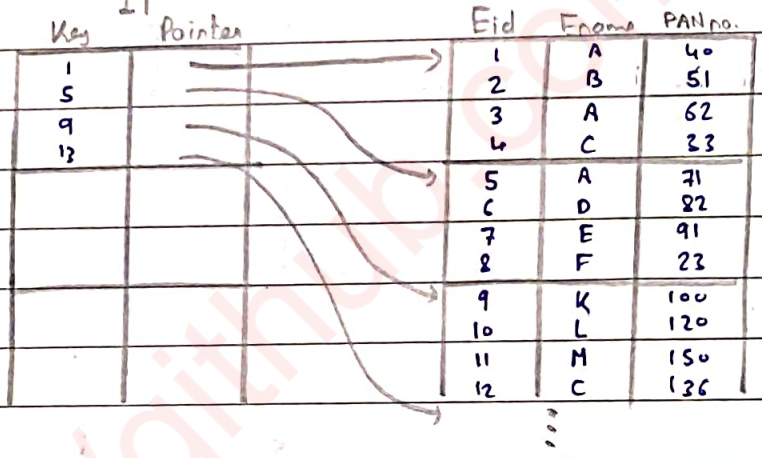
→ These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping.

→ If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. so, secondary indexing is introduced.

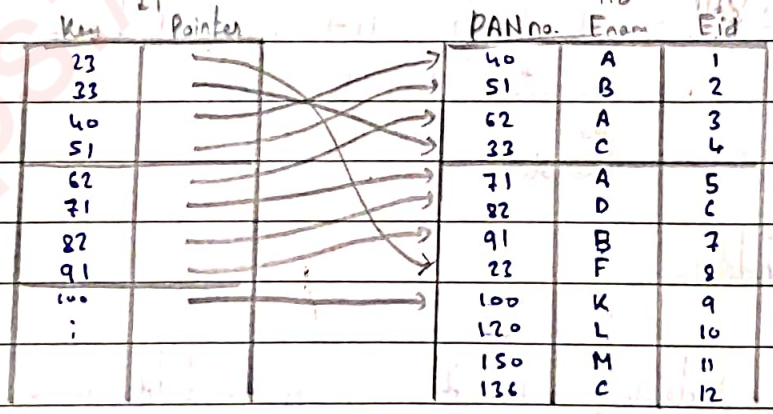
Also ordered ←

→ In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
→ In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small.
→ Then each range is further divided into smaller ranges.
→ The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).

Ex: a) Search key is ordered primary key ⇒ Primary Index (Sparse)



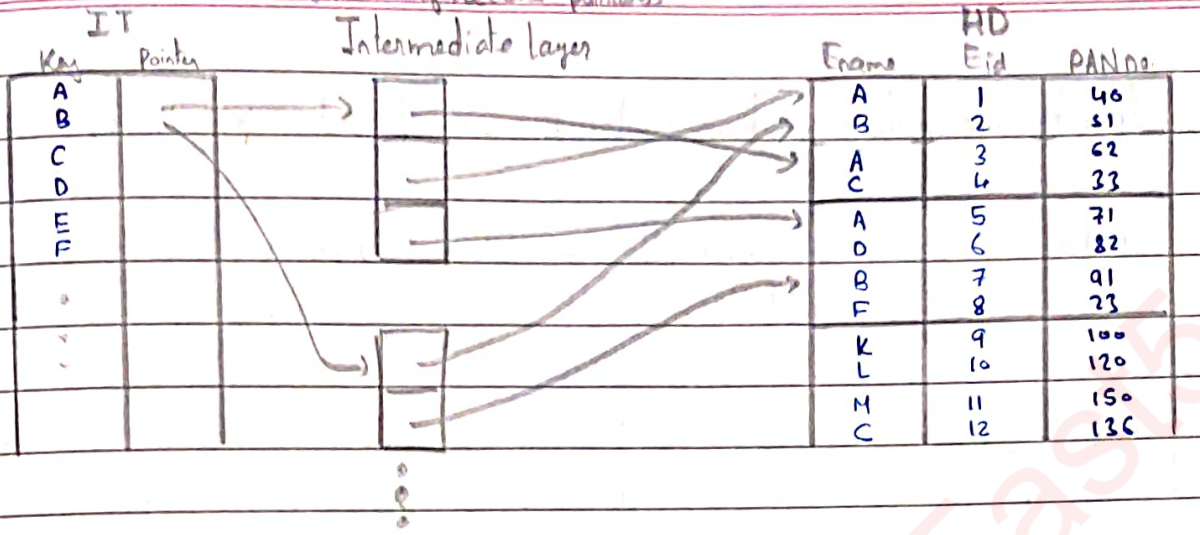
b) Search key is unordered primary key ⇒ Secondary Index (Dense)



c) Search key is unordered non-primary key ⇒ Secondary Index (Dense + Sparse)

P.T.O

Dense index → unordered
(Blocks of record pointers)

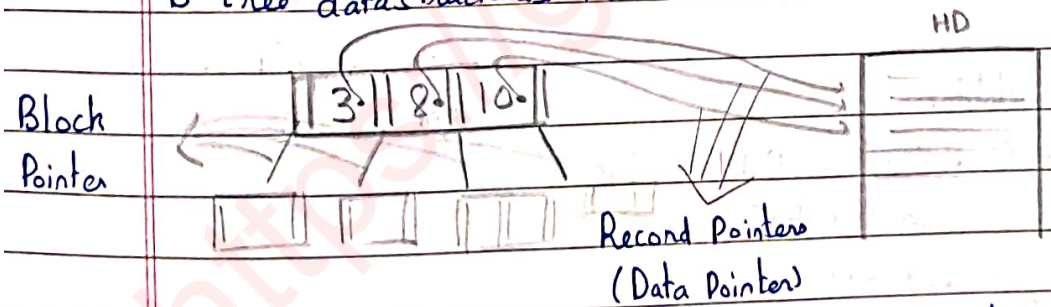


Search → $(\log_2 N_1 + N_2 + 1)$ N_1 → no. of blocks in index table
N_2 → no. of blocks in intermediate table

⇒ B-Tree (Dynamic Multilevel Index):

→ Assume an architecture where we have a Secondary memory and many intermediate tables joined by the index table. So, here if we want to delete, update or insert a new record then it will become very difficult to alter all the values in different tables. So, to reduce this complexity we use B-tree datastructure i.e. we create indexes based on B-tree.

↳ derived from M-way Tree



∴ No. of Children = No. of Block Pointers ∴ No. of Keys = No. of Record Pointers

Suppose Order = n	Root	Intermediate
Max child	n	n
Min child	2	$\lceil \frac{n}{2} \rceil$
Max keys	n-1	n-1
Min keys	1	$\lceil \frac{n}{2} \rceil - 1$

Q Insert the following keys into B-tree if order = 4 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

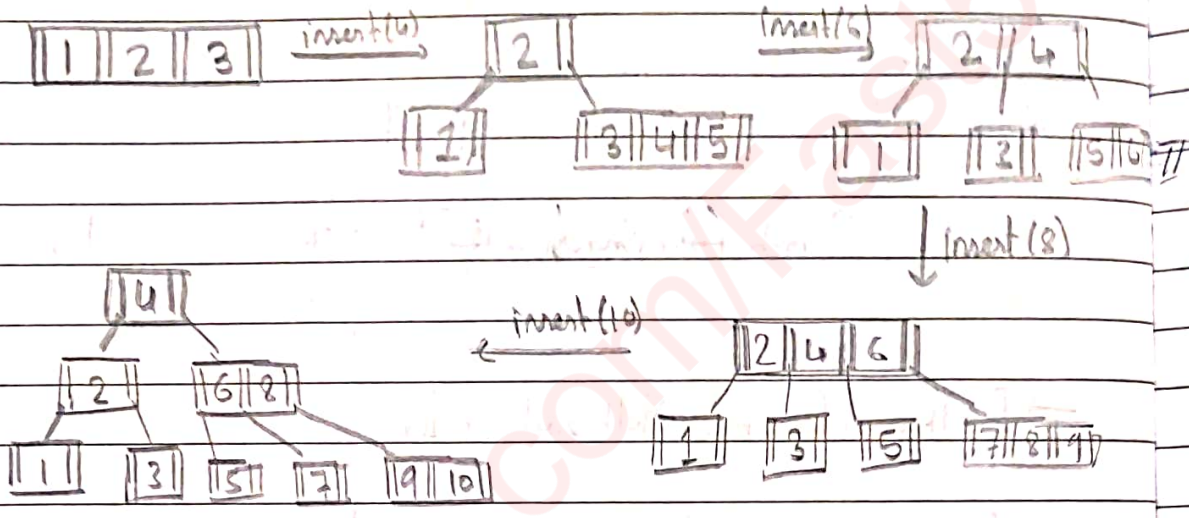
Order = 4 max children : 1) Root = 4 2) Intermediate = 4

min children : 1) Root = 2 2) Intermediate = $\lceil \frac{4}{2} \rceil = 2$

mid-key

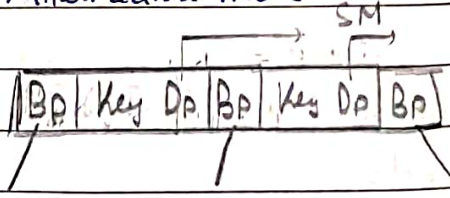
$\lceil \frac{m-1}{2} \rceil$ m → even

$\lceil \frac{m}{2} \rceil$ m → odd



Q Consider a B-Tree with Key Size = 10 bytes, block size = 512 bytes, data pointer size = 8 bytes and block pointer size = 5 bytes. Find the order of B-Tree.

Order of B-tree = max no. of children of root / intermediate nodes



Suppose there n children

$n \times BP + (n-1) Key + (n-1) DP \leq$ Block Size

$5n + 10n - 10 + 8n - 8 \leq 512$

$23n - 18 \leq 512$

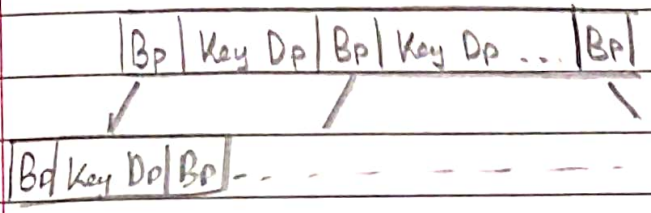
$23n \leq 530$

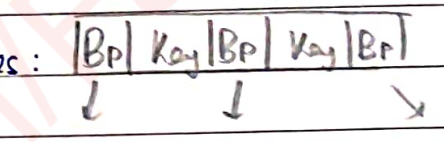
$n \leq 23.04$

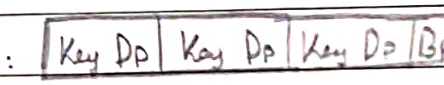
∴ Order of B-tree = 23

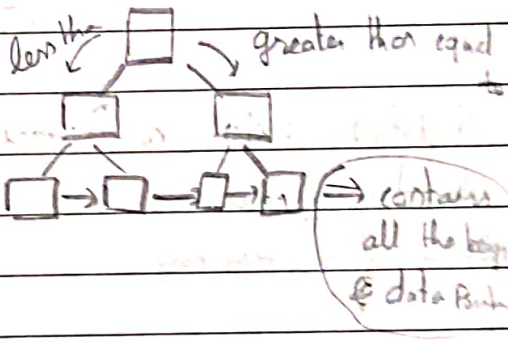
⇒ Difference between B-tree and B+ tree :

B- Tree	B+ Tree
→ Data is stored in leaf as well as internal nodes.	→ Data is stored only in leaf nodes.
→ Searching is slower, deletion complex	→ Searching is faster, deletion easy (directly from leaf node)
→ No redundant search key present.	→ Redundant keys may be present.
→ leaf nodes not linked together.	→ leaf nodes are linked together like linked list.









⇒ B+ Tree:

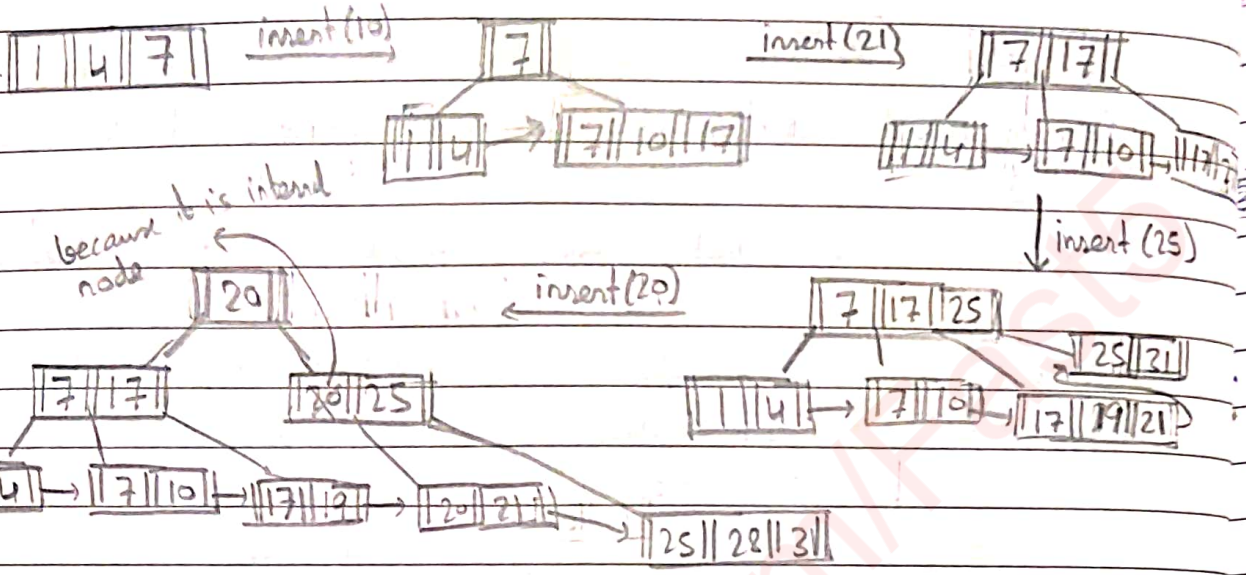
		root node	intermediate node
Order = n	max children	n	m
	min children	2	$\lceil m/2 \rceil$

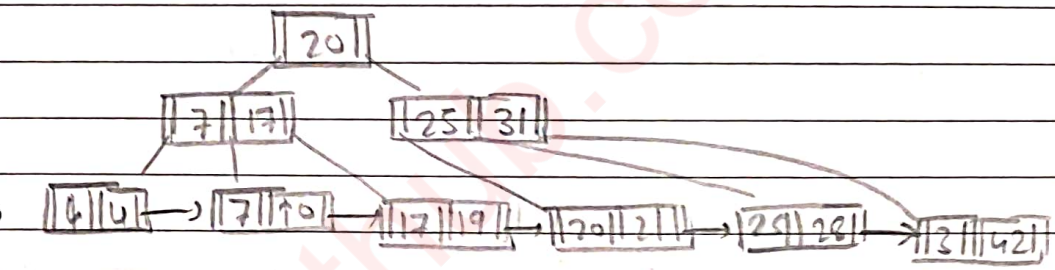
Q Insert the following keys into B+ tree if order = 4 : 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

Order = 4 max children = 4 max keys = 3
min children = $\lceil 4/2 \rceil = 2$ min keys = 2 - 1 = 1

mid-Key

$\frac{m+1}{2}$ max

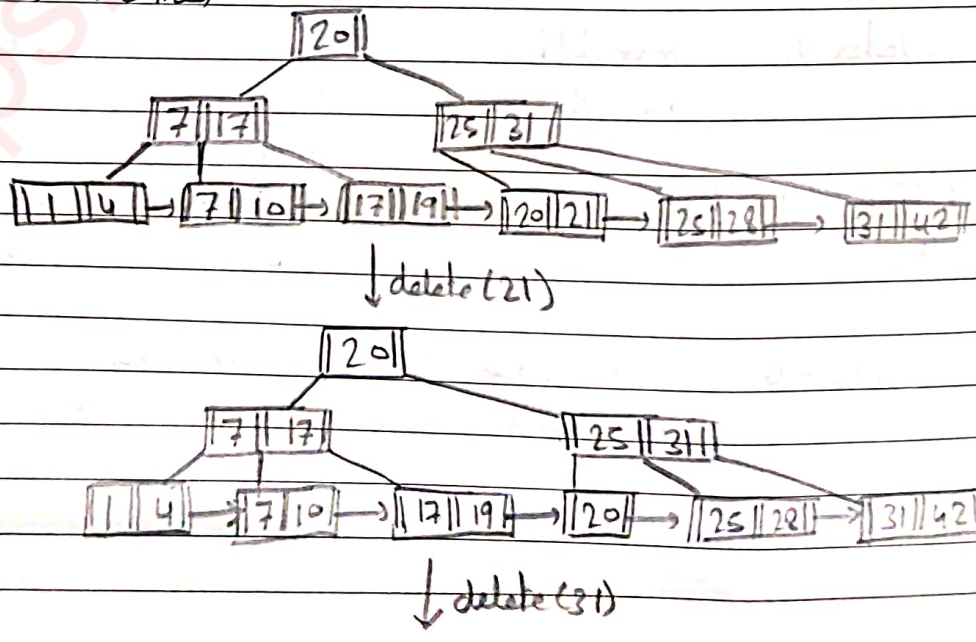


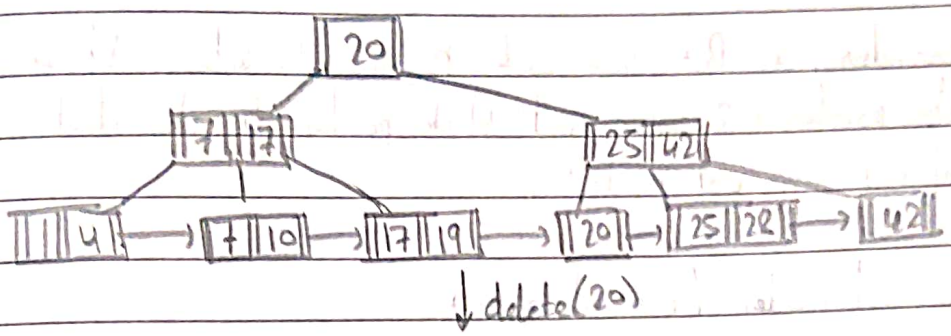


all data is stored in leaf node

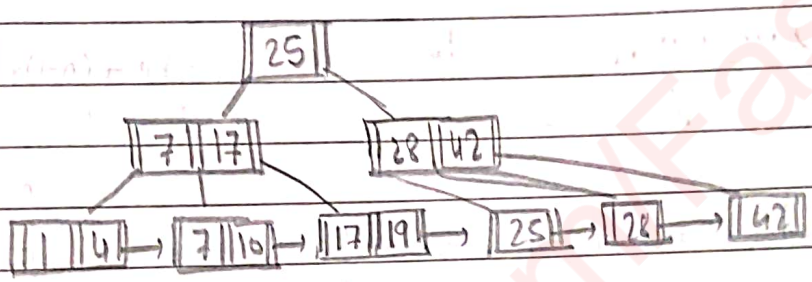
So, searching is very easy ⇒

Q From the above B+ tree create delete the following nodes: 21, 31, 20, 10, 7, 25, 42
(Rules same as in B tree)

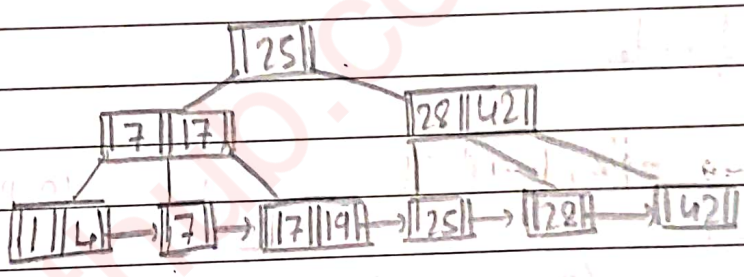




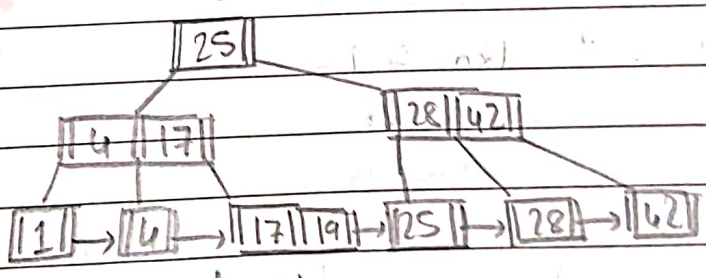
↓ delete(20)



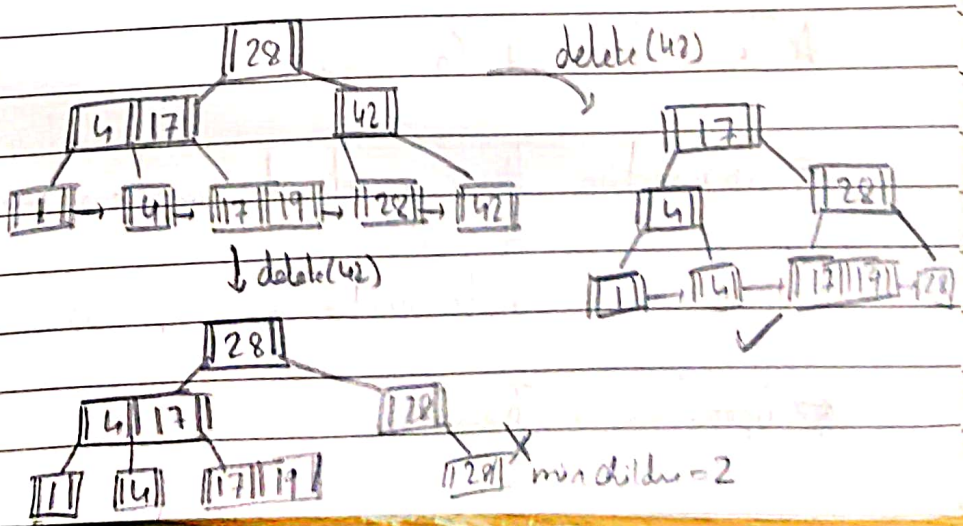
↓ delete(10)



↓ delete(7)

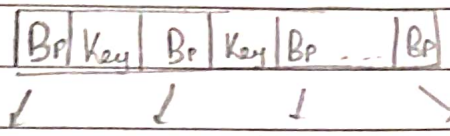


↓ delete(25)



Q Consider a B+ Tree with key size = 10 bytes, block size = 512 bytes, data pointer = 8 bytes and block pointer = 5 bytes. What is the order of leaf and non-leaf node?

i) Non-leaf nodes (internal nodes)



$n \times BP + (n-1) \times key \leq block\ size$

$5n + 10(n-1) \leq 512$

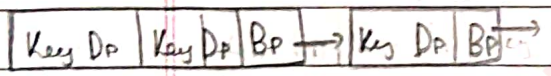
$15n \leq 522$

$n \leq 34.8$

$n_{max} = 34$

∴ Order = max no. of children = 34

ii) Leaf nodes



$(n-1)\,Key + (n-1)\,DP \leq block\ size$

$(n-1)\,10 + (n-1)\,8 \leq 512$

$18n \leq 530$

$n \leq$

$n(Key + DP) + BP \leq block\ size$

$18n + 5 \leq 512$

$18n \leq 507$

$n \leq 28.$

$n_{max} = 28$

∴ Order = max no. of (Key + DP) pairs = 28

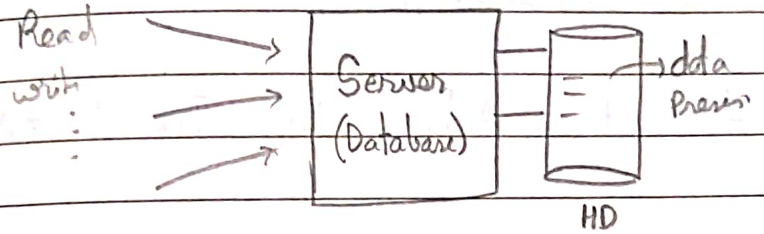
★ Transaction and Concurrency :-

⇒ Transaction: It is a set of operations used to perform a logical unit of work.
A transaction generally represent change in database.

Operations in Transaction:

1) Read Operation (Read(A)) → this instruction will read the value of 'A' from the database and will store it in the buffer in main memory.

2) Write Operation (Write(A)) → this instruction will write the updated value of 'A' from the buffer to the database.



Ex: Transfer ₹500 from A (₹1000) to B (₹2000)

R(A)(-1000)
A = A-500
W(A)(-500) } in RAM not in HD
R(B) (-2000)
B = B+500
W(B) (-2500) } in RAM not in HD
Commit → now permanently change in HD and database

→ ACID properties of Transaction:

To ensure the consistency of the database, certain properties are followed by all the transactions occurring in the system.

1) Atomicity:

→ This property ensures that either the transaction occurs completely or it does not occur at all.
→ In other words, it ensures that no transactions occurs partially.

2) Consistency:

→ This property ensures that the database remains consistent before and after transaction.
→ In other words, Before the transaction starts and After the transaction is completed, the sum of money should be same.

In example of Transfer ₹500 from A to B : Before Transaction = A+B
= 1000+2000
= 3000

After Transaction = A+B
= 500+2500
= 3000

∴ Data base remains consistent

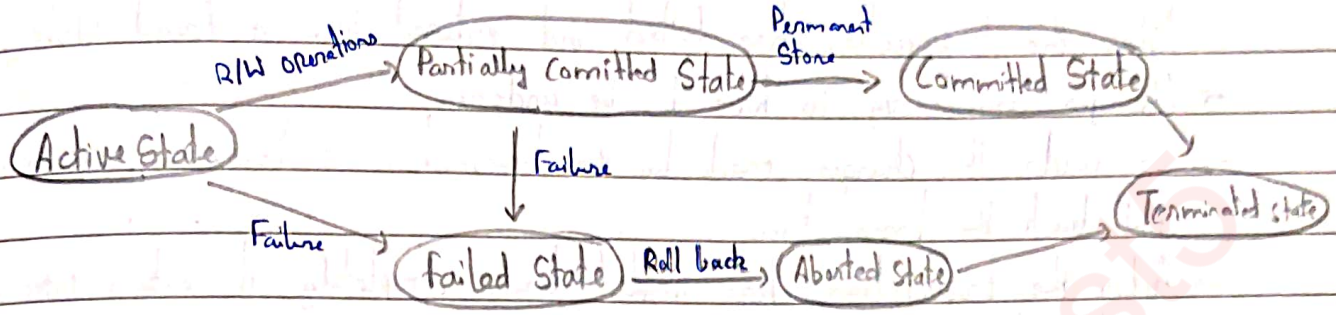
3) Isolation:

→ This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
→ The resultant state of the system after executing all the transactions (Parallel Transaction) is the same as the state that would be achieved if the transactions were executed serially one after the other.

4) Durability:

→ This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
→ It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.

→ Transaction States :



1) Active State

→ This is the first state in the life cycle of a transaction.
→ A transaction is called in an active state as long as its instructions are getting executed.
→ All the changes made by the transaction now are stored in the buffer in main memory.

2) Partially Committed State

→ After the last instruction of the transcation has been executed, it enters into a partially committed state (Before commit).
→ It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

3) Committed State

→ After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state (After Commit).
→ Now, the transaction is considered to be fully committed.

4) Failed State

→ When a transaction is getting executed in the active state or partially committed state and some failures occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

5) Aborted State

→ After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.

→ To undo the changes made by the transaction, it becomes necessary to roll back the transaction.

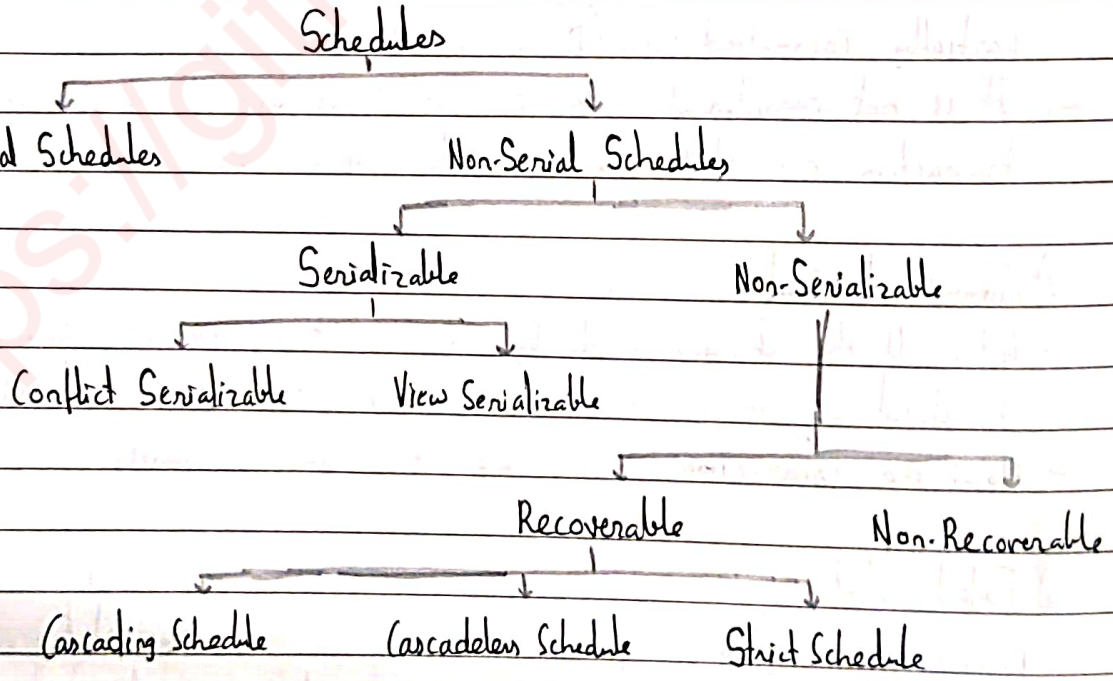
→ After the transaction has rolled back completely, it enters into an aborted state.

6) Terminated State

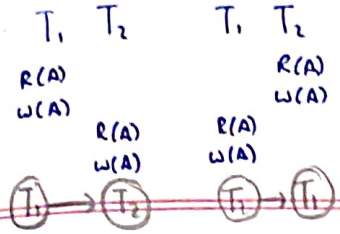
→ This is the last state in the life cycle of a transaction.

→ After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

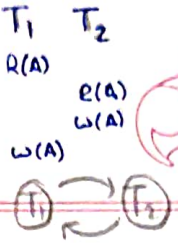
⇒ Schedules:



→ Ex



Ex



→ Schedules: It is chronological execution sequence of multiple transactions.

1) Serial Schedules:

→ All the transactions execute serially one after the other.
→ When one transaction executes, no other transaction is allowed to execute.
→ Serial Schedules are always - Consistent, Recoverable, Cascadeless and Strict.

2) Non-Serial Schedules:

→ Multiple transaction executes concurrently.
→ Operation of all the transactions are inter leaved or mixed with each other.
→ Non-serial Schedules are <u>not</u> always - Consistent, Recoverable, cascadeless and strict.

⇒ Serializability:

→ Some non-serial schedules may lead to inconsistency of the database.
→ Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

In above example of non-serial schedule if $T_1 \rightleftarrows T_2$ can be converted to $T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$ then the non-serial schedules can be converted into serializable schedule.

n transaction → n! serial schedules possible

→ Serializable Schedules:

→ If a given non-serial schedule of 'n' transactions is equivalent to some serial of 'n' transactions, then it is called as a serializable schedule.
→ Serializable schedules are always - Consistent, Recoverable, Cascadeless and Strict.

Types of Serializability:

1) Conflict Serializability:

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called a conflict serializable schedule.

Conflict Equivalent:

R(A) R(A) → non-conflict pair

R(A) W(A)
W(A) R(A)
W(A) W(A)
→ Conflict pairs

R(B) R(A)
W(B) R(A)
R(B) W(A)
W(A) W(B)
→ non-conflict pairs

Q Are S and S' are conflict equivalent ?

S: T_1	T_2	S': T_1	T_2
R(A)		R(A)	
W(A)		W(A)	
	R(A)	R(B)	
	W(A)		R(A)
R(B)			W(A)

$T_1 \rightarrow T_2$

Swap the positions of adjacent non-conflict pairs

S_1: T_1	T_2		S_2: T_1	T_2
R(A)			R(A)	
W(A)		⟶	W(A)	
	R(A)		R(B)	
R(B)				R(A)
	W(A)			W(A)

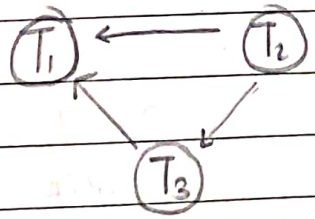
$\therefore S_1 \equiv S_1'$ i.e. S and S' are conflict equivalent schedules

Other type of problem is to check whether a given schedule is conflict serializable schedule or not. ⇒ Use Precedence Graph

Precedence Graph : $G(V, E)$
- V: no. of vertices = no. of transactions
- E: check conflict pairs in other transactions and draw the edges

Q Check whether the given schedule S is conflict serializable. If yes, then also determine which serial schedule it is equivalent to.

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



$\therefore$ No loop / Cycle

$\therefore$ Conflict Serializable ⟶ serializable ⟶ consistent

↓ of which type?

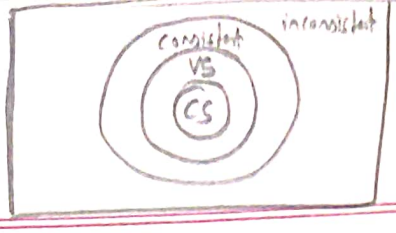
6 possibilities (n!)

↓

indegree = 0 ⇒ write & remove from graph

↓

$T_2 \rightarrow T_3 \rightarrow T_1$



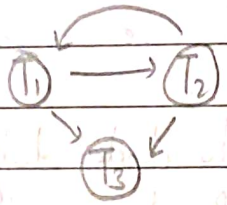
2) View Serializability:

If a given schedule is found to be viewed as equivalent to some serial schedule, then it is called a view serializable schedule.

Q. Check whether the given schedule is conflict serializable or not?

S:

T_1	T_2	T_3
R(A)		
	w(A)	
w(A)		
		w(A)



∴ loop/cycle exist

∴ ~~we~~ cannot Non-conflict serializable → But we cannot say anything about serializability

↓

to know that we use view serializability

View Serializability:

S':

	T_1	T_2	T_3
A=100	R(A)		
A=A-40 A=60	w(A)		
A=A-20 A=40		w(A)	
A=A-40 A=0			w(A)

S:

	T_1	T_2	T_3
A=100	R(A)		
A=A-20 A=80		w(A)	
A=A-40 A=40	w(A)		
A=A-40 A=0			w(A)

$T_1 \rightarrow T_2 \rightarrow T_3$

(Assume A=100)

Same

∴ $S \equiv S'$ i.e, S and S' are view equivalent

→ Non-Serializable Schedules:

→ A non-serial schedule which is not serializable is called a non-serializable schedule.
→ A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.
→ Non-serializable schedules - may or may not be consistent and may or may not be recoverable.

1) Irrecoverable Schedules:

If in a transaction schedule,
→ A transaction performs a dirty read operation from an uncommitted transaction
→ And commits before the transaction from which it has read the value then such a schedule is known as an Irrecoverable Schedule.

Ex: S:

T_1	T_2
R(A)	
W(A)	
	R(A) = 50 | x100
	commit
failure	

Roll Back

Ex: S':

T_1	T_2
R(A)	
W(A)	
	R(A)
commit/Roll back	
	commit

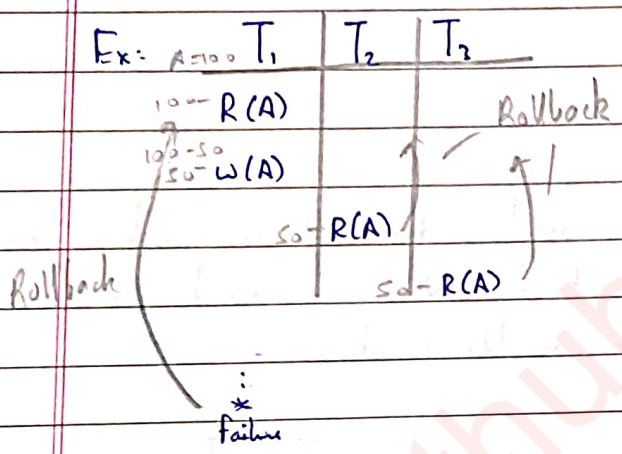
2) Recoverable Schedules:

If in a schedule,
→ A transaction performs a dirty read operation from an uncommitted transaction
→ And its commit operation is delayed till the uncommitted transaction either commits or roll backs
then such a schedule is known as a Recoverable Schedule.

Types of Recoverable Schedules:

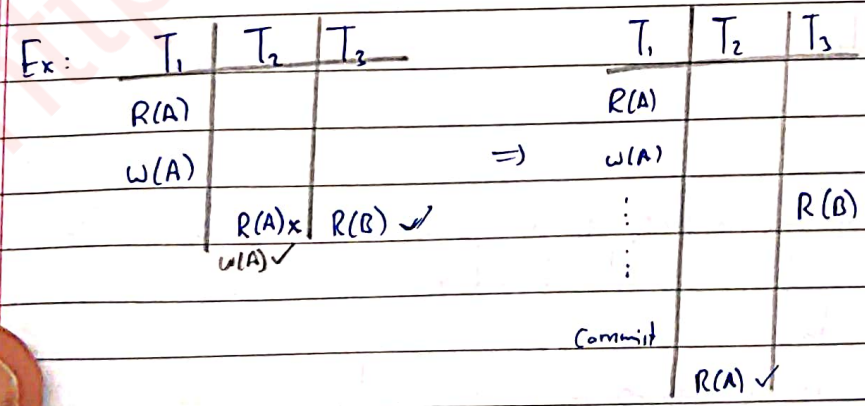
1) Cascading Schedule

→ If in a schedule, failure of one transaction causes several others dependent transaction to rollback or abort, then such a schedule is called as a cascading schedule.



2) Cascadeless Schedule

→ If in a schedule, a transaction is not allowed to read a data item until the last dependent transaction that has written is committed / aborted, then such a schedule is called Cascadeless Schedule.



3) Strict Schedule

→ If in a schedule, a transaction is neither allowed to read nor write a data item until the last dependent transaction that has written is committed / aborted, then such a schedule is called as a Strict Schedule.

Ex:

T_1	T_2	T_3
R(A)		
w(A)		
	R(A)✗	w(A)✗

⇒

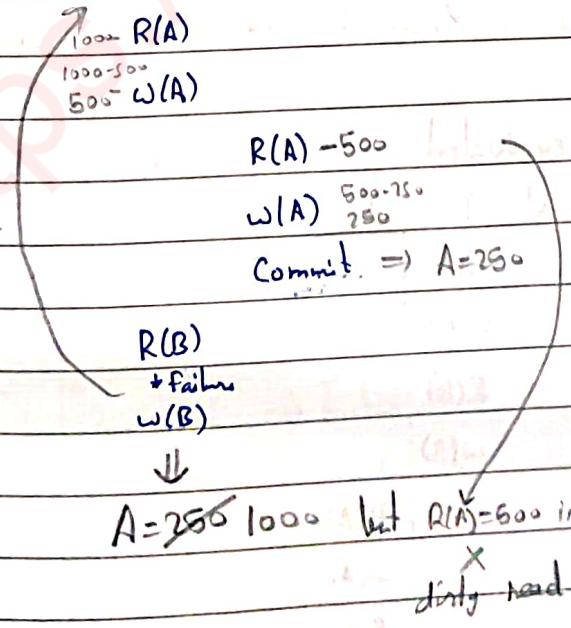
T_1	T_2	T_3
R(A)		
w(A)		
⋮		
commit		
	R(A)✓	w(A)✓

⇒ Problems due to Concurrent Execution:

1) Write-Read Problem (Dirty Read Problem)

A=1000

S: T_1 T_2



2) Read - Write Problem (Unrepeatable Read Problem)

A=100 T_1 T_2

100 ← R(A)

R(A) → 100

w(A) → 50 (A-10)

Commit ⇒ A = ~~100~~ 50

50 ← R(A)

w(A)

Commit

how (user 1)

3) Write-Write Problem (Lost Update Problem)

3 A=100 T_1 T_2

100 ← R(A)

100-20 = 80 ← w(A)

w(A) → 90 (80+10)

Commit ⇒ A = ~~100~~ 90

Commit ⇒ A = 90.

But User 1 thought it would be 80 ∴ Upade is lost

Q Are S and S' are view equivalent

S: T_1	T_2	S': T_1	T_2
R(A)		R(A)	
w(A)		w(A)	
	R(A)	R(B)	
	w(A)	w(B)	
R(B)			R(A)
w(B)			w(A)
	R(B)		R(B)
	w(B)		w(B)

i) Initial Read

	S	S'
A:	T_1	T_1
B:	T_1	T_1

✓

ii) Final Write

	S	S'
A:	T_2	T_2
B:	T_2	T_2

✓

iii) W-R ...(intermediate)

	S	S'
A:	$T_1 \to T_2$	$T_1 \to T_2$
B:	$T_1 \to T_2$	$T_1 \to T_2$

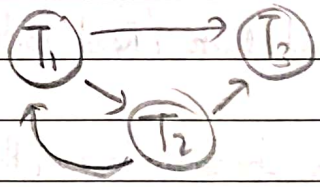
✓

$\therefore S \equiv S'$ i.e. view equivalent

Q Check view serializability for S.

S:	T_1	T_2	T_3
	R(A)		
		w(A)	
	w(A)		
			w(A)

First check if it is conflict serializable



$\therefore$ loop exists

$\therefore$ Non-conflict serializable

So, check view serializability

possible S' : 3! = 6 possibilities ($T_1 \to T_2 \to T_3$, $T_1 \to T_3 \to T_2$, $T_2 \to T_3 \to T_1$ ...)

S':	T_1	T_2	T_3
	R(A)		
	w(A)		
		w(A)	
			w(A)

i) Initial Read

	S	S'
A:	T_1	T_1

✓

ii) Final write

	S	S'
A:	T_3	T_3

✓

iii) W-R

	S	S'
A:	-	-

✓

$\therefore S \equiv S'$ i.e. views equivalent

$\therefore$ view serializability

⇒ Concurrency Control Protocols:

→ The concurrency control protocols ensures the atomicity, consistency, isolation, durability and serializability of the concurrent execution of the database transactions.

→ Basic Aim is to achieve Serializability and Recoverability.

1) Lock-based Protocol:

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it.

a) Shared-Exclusive Locking

i) Shared lock / Read lock (S)

→ In a shared lock, the data item can only be read by the transaction.

→ It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

Ex:

T_1	T_2
S(A)	
R(A)	
W(A) x	S(A) R(A) ✓
:	W(A) x
U(A)	U(A)

ii) Exclusive lock (X)

→ In the exclusive lock, the data item can be both read as well as written by the transaction.

→ It cannot be shared between the transaction.

Ex: T_1 T_2

X(A)
R(A)
W(A)
R(A)x
W(A)x

Lock Compatible Table:

grant ↓ / Request →	S	X
S	✓	x
X	x	x

Q Apply Shared-Exclusive protocol on below schedules:

S_1: T_1 T_2

R(A)
W(A)
R(A)
R(B)
W(B)
R(B)

↓
Serializable

S_1: T_1 T_2

X(A)
R(A)
W(A)
U(A)
S(A)
R(A)
U(A)
X(B)
R(B)
W(B)
U(B)
S(B)
R(B)
U(B)

S_2: T_1 T_2

R(A)
W(A)
R(A)
R(B)
R(B)
W(B)

↓
Non-Serializable

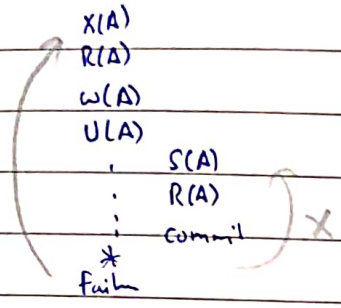
S_2: T_1 T_2

X(A)
R(A)
W(A)
U(A)
S(A)
R(A)
U(A)
S(B)
R(B)
U(B)
X(B)
R(B)
W(B)
U(B)

Problems in Shared-Exclusive locking:

→ May not be sufficient to produce only serializable schedule. (See Previous Q)
→ May not free from Irrecoverability.

Ex: T_1 T_2



→ May not be free from dead lock

Ex: T_1	T_2
G X(A)	
	X(B) G
wait for ← X(B)	
	X(A) → wait for

Both waiting → dead loch state (Infinite waiting)

→ May not be free from starvation

Ex: T_1	T_2	T_3	T_4 ...
	G S(A)		
waiting X(A)	⋮		
	⋮	G S(A)	
Starvation	U(A)	⋮	G S(A)
		⋮	⋮
		U(A)	⋮ ...

b) 2-Phase Locking:

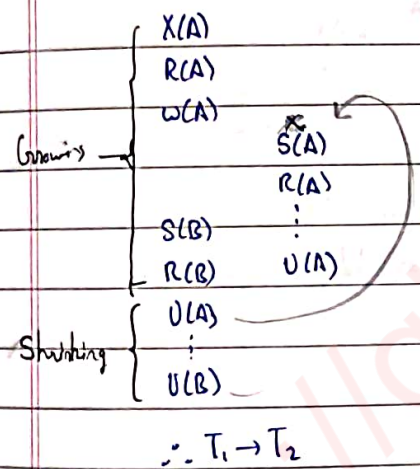
→ The 2-phase locking protocol divides the execution phase of the transaction into two parts: i) Growing Phase ii) Shrinking Phase

i) Growing Phase: locks can be acquired but cannot be released

ii) Shrinking Phase: locks can be released but cannot be acquired

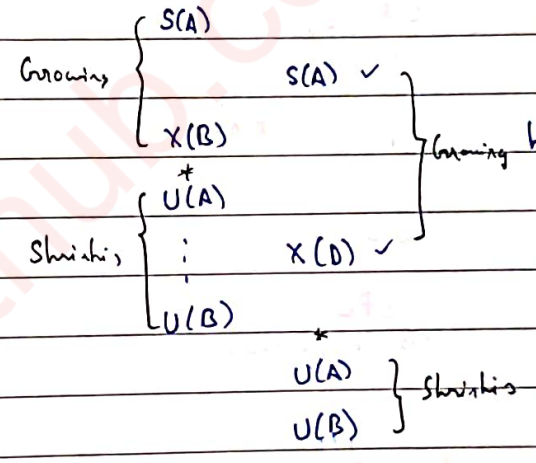
→ Here Serializability is achieved.

Ex: i)



∴ $T_1 \rightarrow T_2$

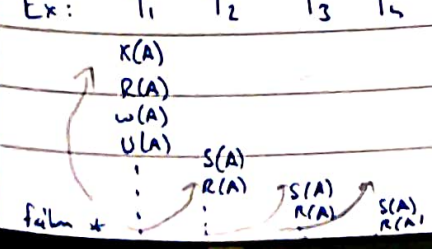
ii)



Advantages: always ensures serializability

Drawbacks: → May not be free from irrecoverability (same as shared-Exclusive)
→ May not be free from dead lock (same as shared-Exclusive)
→ May not be free from starvation (same as shared-Exclusive)
→ May not be free from cascading rollback

Ex:

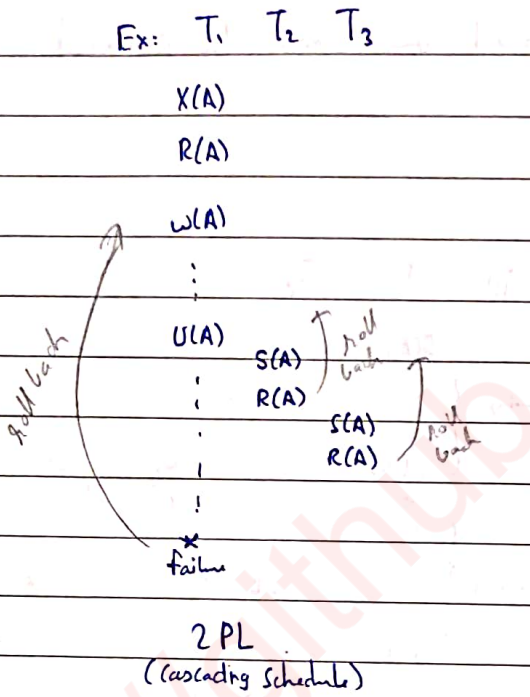


c) Strict 2PL:

It should satisfy the basic 2PL and also all exclusive locks should hold until commit / roll back.

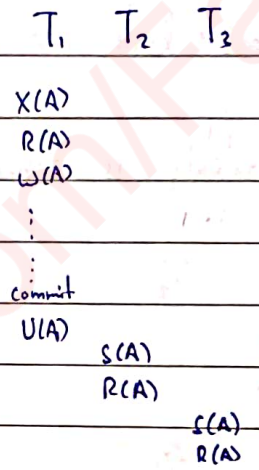
Advantages: → free from cascading schedule i.e. cascadeless schedule

Ex:



2PL
(Cascading Schedule)

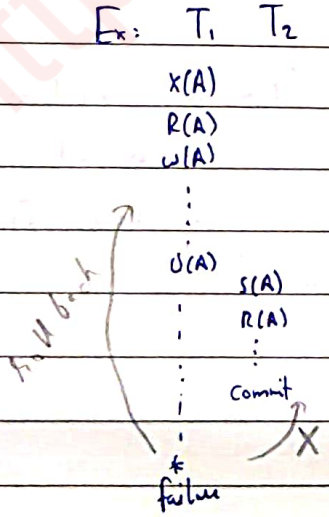
⇒



Strict 2PL
(Cascadeless schedule)

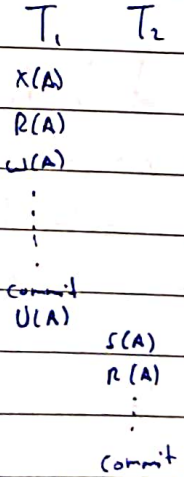
→ free from irrecoverability

Ex:



2PL
(Irrecoverable)

⇒



Strict 2PL
(Recoverable)
always

2) Time Stamp Ordering Protocol

→ Time Stamp TS(Ti) is a unique value assigned to every transaction by DBMS.
→ It tells the order of transaction (when they enter into system).

Ex: T_1^{20} T_2^{15} T_3^{50}
↑ oldest ↑ youngest

Read_TS (RTS): Last (latest) transaction no. which performed Read successfully.

Write_TS (WTS): Last (latest) transaction no. which performed Write successfully.

Ex: T_1^{10} T_2^{20} T_3^{30} ⇒ RTS(A) = 30
R(A) WTS(A) = 20
W(A)
R(A)
W(A) R(A)

Conflicting Pairs: RW, WR, WW (To pehle aayega vo pehle jayega)
↙ ↘
allowed if in increasing sequence else not allowed

T_1^{10} T_2^{20}
R(A) 10→20
W(A)
W(B) 10→20
R(B)
W(C) 10→20
W(C)

a) Basic Time stamp Ordering Protocol:

i) Transaction T_i → after a transact issues a Read (A) operation.

→ If WTS(A) > TS(T_i), then roll back T_i (R(A) ✗)
→ otherwise R(A) ✓ and set RTS(A) = max {RTS(A), TS(T_i)}

older → 100 200 ← younger

Ex. T_1 T_2

R(A)
W(A)
W(B)
R(B)
W(C)
W(C)

WTS(A) = 200

Ex: T_1 (100) T_2 (200)

W(A) ✓
R(A)
~~R(B)~~ W(B) ✗ WTS(B) = 200
R(B) ~~R(B)~~

roll back

ii) Transaction T_i issues a Write (A) operation

→ If RTS(A) > TS(T_i), then roll back T_i (W(A) ✗)
→ If WTS(A) > TS(T_i), then roll back T_i (W(A) ✗)
→ otherwise W(A) ✓ and set WTS(A) = TS(T_i)

Ex: T_1 (100) T_2 (200)

R(A) ✗ RTS(A) = 200
roll back W(A)
W(B) ✗ WTS(B) = 200
roll back W(B)
R(C) ✓
W(C)
W(D) ✓
W(D)

Q In the given schedule check whether it is allowed to execute by using BTMO.

oldest ← (100) (200) (300) → youngest

T_1	T_2	T_3
✓ R(A)		
	R(B) ✓	
W(C) ✓		
	R(B) ✓	
R(C) ✓	↑ roll back	
W(B)		
		W(A)

Initially:

	A	B	C
RTS	0	0	0
WTS	0	0	0

i) $R(A) \rightarrow T_1$: $0 < 100 \Rightarrow$ set $RTS(A) = 100$

ii) $R(B) \rightarrow T_2$: $0 < 200 \Rightarrow$ set $RTS(B) = 200$

iii) $W(C) \rightarrow T_1$: $0 \overset{R(C)}{<} 100 \overset{0<100}{\Rightarrow}$ set $WTS(C) = 100$

iv) $R(B) \rightarrow T_3$: $200 < 300 \Rightarrow$ set $RTS(B) = 300$

v) $R(C) \rightarrow T_2$: $100 \overset{?}{<} 100 \overset{\times}{\Rightarrow}$ set $RTS(C) = 100$

vi) $W(B) \rightarrow T_2$: $300 \overset{R(B)}{>} 200 \Rightarrow$ roll back T_2

vii) $W(A) \rightarrow T_3$: $100 \overset{\times}{>} 30 \rightarrow / 0 > 300 \overset{\times}{\Rightarrow}$ set $WTS(A) = 300$

Advantages: → ensures serializability (CSS) Disadvantages: → not free from starvation
→ free from dead lock → may be Irrecoverable

b) Thomas Write Time-stamp

i) Transaction T_i issues a Read (A) operation (Same as BTMO)

→ If $WTS(A) > TS(T_i)$, then roll back T_i (R(A) ✗)
→ otherwise R(A) ✓ and set $RTS(A) = \max\{RTS(A), TS(T_i)\}$

ii) Transaction T_i issues a Write (A) operation

→ If $RTS(A) > TS(T_i)$, then roll back T_i (W(A) ✗)
→ If $WTS(A) > TS(T_i)$, then ignore W(A) by T_i & continue the execution of T_i
→ otherwise W(A) ✓ and set $WTS(A) = TS(T_i)$

Advantages: → ensures serializability (VSS) Disadvantages: → not free from starvation
→ free from dead lock → may be Irrecoverable

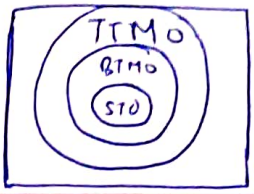
Q. Which of the following is correct ?

a) 2PL → optimistic protocol

b) 2PL → Pessimistic protocol

c) Time-stamp → optimistic protocol

d) Time-stamp → pessimistic protocol



c) Strict Time-stamp Ordering Protocol:

BTMO / TTMO + Strict Recoverability ⇒ also ensures Recoverability

⇒ Deadlock handling:

→ A system is in dead lock state if there exists a set of transaction such that every transaction in the set is waiting for another transaction in the set.

Ex: T_1 T_2

X(A)

X(B)

wait kar raha ← X(B)

X(A) → wait kar raha

→ If there exists a set of waiting transaction $T_0, T_1, \ldots, T_{n-1}$ such that $T_0 \to T_1, T_1 \to T_2, \ldots, T_{n-1} \to T_0$, so none of the transaction can progress in such situation.

There are two principle for dealing with deadlock problem:

1) Prevention : which ensure that system will never enter into a deadlock state.

2) Detection and Recovery: allow system to enter into deadlock, then try to recover.

I) Deadlock Prevention

→ To ensure no hold & wait, each transaction locks all the data items before it begins execution.

→ To ensure no cyclic wait, impose an ordering of all data items. $T_1 \to A(1)$, $T_2 \to B(2)$

a) Wait-Die Protocols:

→ non preemptive → don't forcefully snatch resources from other transactions.
→ older transaction may wait for younger transaction to release data item.
→ younger transaction never waits for older transaction and roll backs.

$TS(T_i) < TS(T_j) \Rightarrow T_i \rightarrow$ older, $T_j \rightarrow$ younger

i) If T_i requires shared resource holded by $T_j \Rightarrow (T_i) \rightarrow (T_j)$

ii) If T_j requires shared resource holded by $T_i \Rightarrow (T_i) \leftarrow (T_j)$ ↑roll back

b) Wound-Wait Protocol:

→ Preemptive → forcefully snatch resources from other transactions
→ older transaction wounds younger transaction (roll back) to release data item
→ younger transaction waits for older transaction

$TS(T_i) < TS(T_j) \Rightarrow T_i \rightarrow$ older, $T_j \rightarrow$ younger

i) If T_i requires shared resource holded by $T_j \Rightarrow (T_i) \rightarrow (T_j)$ ↑roll back

ii) If T_j requires shared resource holded by $T_i \Rightarrow (T_i) \leftarrow (T_j)$

Both of these protocols consists of roll backs which reduces the efficiency.

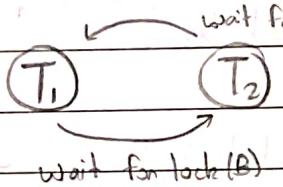
c) Lock-Timeouts

→ In above methods wait can be infinite (Starvation) but here we give a specified time for wait.

2) Deadlock Detection and Recovery

→ In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for graph to detect the deadlock cycle in the database.

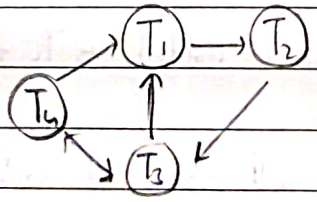
Wait for Graph: If cycle or loop is present, then there is a deadlock



⇒ Cycle / loop present
∴ in a deadlock state

→ When a detection algorithm determines that a deadlock exists, the system must recover from the dead lock.
→ The most common solution is to roll back one or more transactions to break the deadlock.
→ Choosing which transaction to abort is known as Victim Selection.

Ex: In the below wait-for-graph transactions T_1, T_2 and T_3 are deadlocked. In order to remove dead lock one of the three transactions must be rolled back. (upto to some check point → Partial roll back)



We should roll back those trasactions that will incur the minimum cost.
→ The trasaction which have the fewest locks
→ The transaction that has done the least work
→ The transaction that is farthest from completion.

Recovery Management System:-

→ Wherever a system fails to function according to its specification and doesn't deliver the expected output, that situation is called the failure of the system.
→ Database Recovery Management helps to recover this type of failures in DBMS.
→ It ensures the atomicity and durability.

⇒ Log Based Recovery:

→ A log is a sequence of records that contains the history of all updates made to the database.
→ Sometimes log records is also known as System log.

log record has the following fields: 1) Transaction identifier → to get the transaction that is executing
2) Data item identifier → to get the data item of the transaction that is running.
3) The old value of the data item (before write)
4) The new value of the data item (after write)

Ex:

T_i	log record
R(A)	$< T_i, start >$
A=A-50	$< R, T_i, 1000 >$
W(A)	$< W, T_i, 1000, 950 >$
B=B+50	
W(B)	$< W, T_i, 1000, 1050 >$
commit	$< T_i, commit >$

* log files are written before the actual operations

→ Wherever a transaction performs a write, it is essential that the log record for that write is to be created before the DB is modified.
→ Once a log record exists, we can output the modification in DB if required. Also, we have the ability to undo the modification that has already been updated in DB.

Log Based Recovery work in two modes: 1) Immediate mode 2) Deferred mode

1) Immediate Mode:

→ In immediate mode of log-based recovery, database modification is performed while transaction is in Active State.
→ It means as soon as transaction is performed or executes its write operation, then immediately these changes are saved in Database also.
→ In immediate mode, there is no need to wait for the execution of the COMIT statement to update the database.

If the system is crashed or failed:

a) Case-1: If the system crashes after the transaction has executed the commit state

In this case, when transaction executed commit statement, then corresponding commit entry will also be made to the log file immediately.

To recover the database, recovery manager will check the log file to recover the database, then the recovery manager will find both <T, start> and <T, commit> in the log file which represents that transaction T has been completed successfully before the system failed, so REDO(T) operation will be performed and updated values of Data item A and B will be set in DB.

b) Case-2: If transaction failed