Database Management systems :

Relational DBMS :

→ It is a DBMS in which data is stored in the form of relations.

*) Data (Raw fact) → Information (Meaningful data) → record (Logically related data) → database (similar records) → DB Mgmt (set of pgms) → DBMS (records + pgms)

*)

Datastructure	DBMS	
Graphs	Network DBMS	→ Not feasible
Trees	Hierarchial DBMS	→ Not feasible
Tables | set	Relational DBMS	→ present in use
objects	Object oriented DBMS	→ future.
objects + Table	object relational DBMS	→ future.

*) DBMS is used when there is large volume of data to store and many users are present
Ex:- E-commerce
Ex:- Banking reservation.

Goal of DBMS :

EFFICIENT STORAGE AND EFFICIENT RETREIVAL

Conceptual design :

① ER Model : (Top-down approach)

Components → (11)

$(ER)^2$ HACK AWS DP

- Entity
- Entity set
- Relationship
- Relationship set
- Hierarchial relationship
- Aggregation
- Cardinality ratio
- Key Constrains
- Attributes
- weak entity set
- strong Entity set
- Degree of rel'ship
- participation Constraints.

1) Entity : Noun
object in real world

2) Entity set : Collection of similar entities.
rep :

3) Relationship : verb
Association among Entity sets

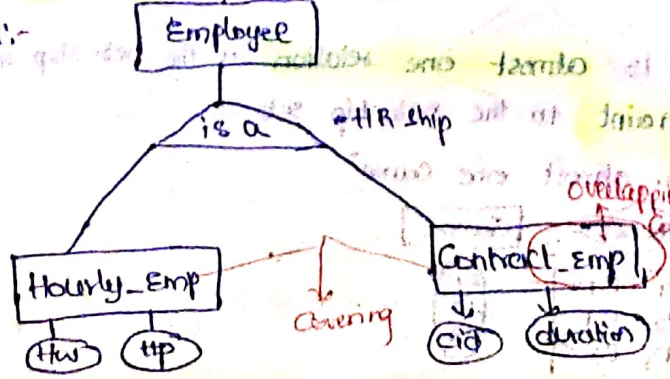
4) Relationship set : Collection of similar relationships
rep :

5) Hierarchial relationship :

Used to show the relationship among the generalized and specialized Entity

rep : 

Ex:-



Overlapping Constraint :

It determines whether two Entity sets in the sub class can be rep with single Entity set

Ex:- Contract Emp overlaps Senior Emp

Covering Constraint :

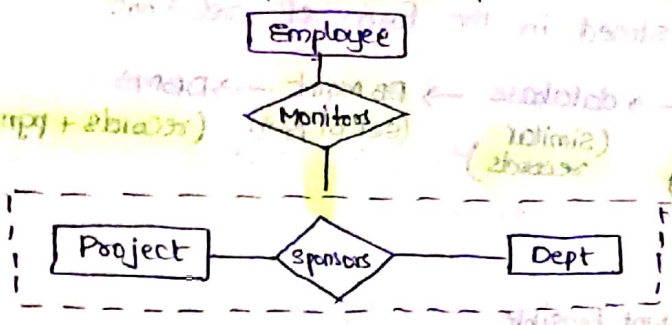
It determines whether all Entities in the sub class collectively includes every Entity in the super class

Ex:- Hourly Emp & Contract Emp Covers Employee

⑥ Aggregation :

It allows relationship to participate in another relationship set

Ex:-



→ Aggregation box.

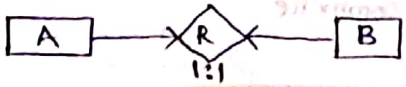
⑦ Cardinality ratio : (Mapping cardinalities)

It Express num of entities to which another entity can be associated via relnships

a) 1:1 cardinality :

Each entity in A is associated with atmost one entity in B
Each entity in B is associated with atmost one entity in A

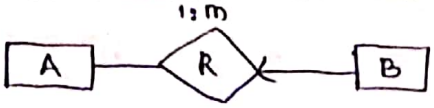
Ex:-



b) 1:m cardinality :

Each entity in A is associated with 0 or many entities in B
Each entity in B is associated with atmost one entity in A.

Ex:-



c) m:1 cardinality :

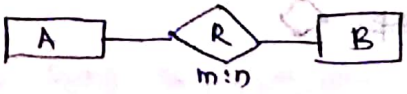
Each entity in A is associated with atmost one entity in B
Each entity in B is associated with 0 or many entities in A

Ex:-

d) m:n cardinality (default cardinality)

Each entity in A is associated with 0 or many entities in B
Each entity in B is associated with 0 or many entities in A

Ex:-



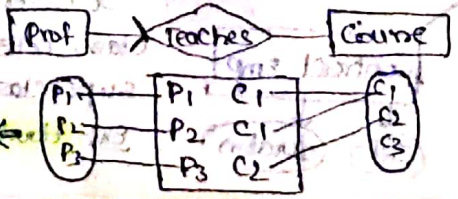
Note : In 1:m and m:1, the key constraint is from m side entity set to the relnship set

⑧ Key Constraints :

If for each entity in the entity set there is atmost one relation in the relnship set then the entity set is said to be key constraint to the relnship set

rep :→

Ex:- Each prof teaches atmost one course



⑨ Attributes :

a) Simple Attribute : which cannot be divided further

Rep : (Sal)

b) Composite Attribute : Composed of set of simple Attributes.

Rep :

c) single valued Attribute : Almost one value for each entity in the entity set

Rep : (Contact)

d) Multivalued Attribute : More than one value for each entity in the entity set

Rep : ((Contact))

e) Stored Attribute : An Attribute whose value need not to be changed.

Rep : (DoB) (Gender)

f) Derived Attribute : whose value derived from another attribute.

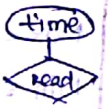
Rep : (Age)

g) Key Attribute : which uniquely identifies each entity in the entity set

Rep : (RNO)

h) Descriptive Attribute : which describes relationship set

Rep :



⑩ strong Entity set :

An Entity set which has a key Attribute to distinguish each entity in the entity set

Ex:-

→ strong

⑪ weak Entity set :

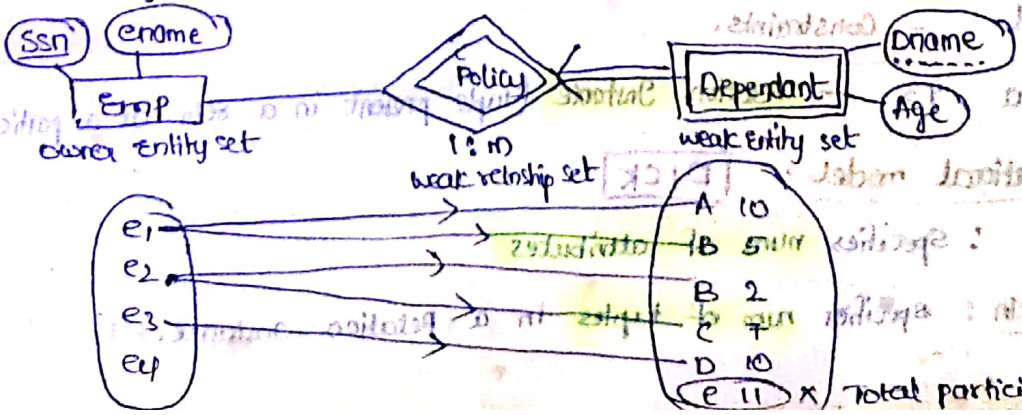
An Entity set which doesn't has a key Attribute to distinguish each Entity in the Entity set.

→ weak Entity set must always be in total participation with weak relationship set.

→ The cardinality of owner Entity set with weak relationship set is 1 : m.

→ weak Entity set is uniquely identifiable using partial key and key of owner Entity set

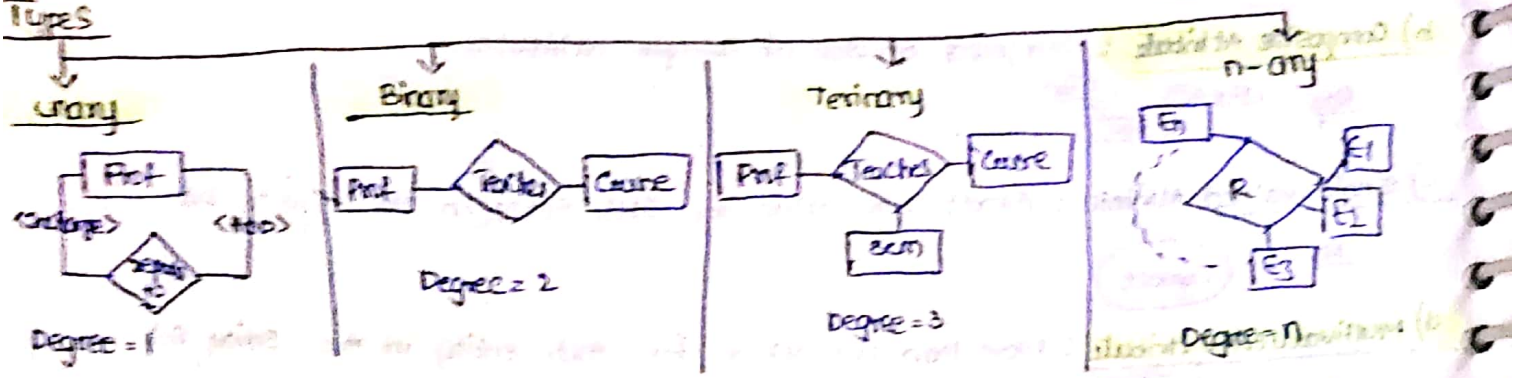
Ex:-



(12) Degree of a relationship set:

It specifies num of Entity sets associated with the relationship set

Types

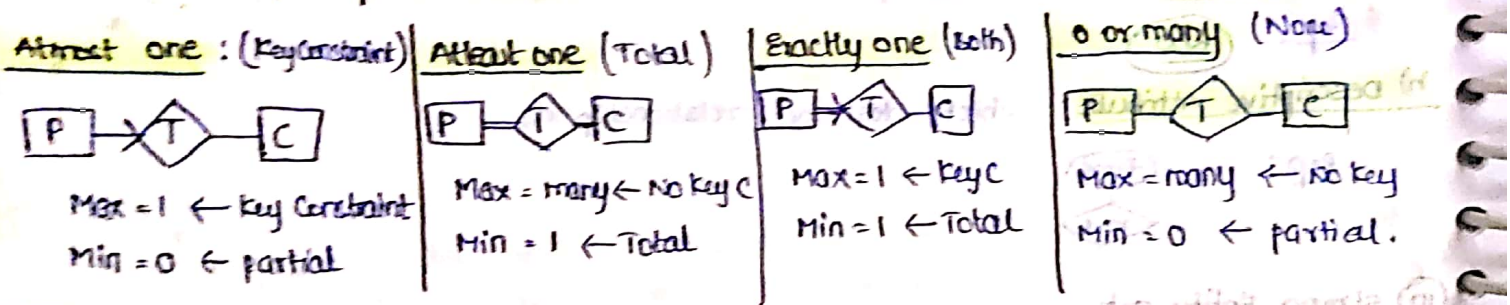


(13) Participation Constsaints:

If Every Entity in the Entity set participates in a relationship set then the Participation is said to be Total participation otherwise the participation is said to be partial participation

Total participation ⇒ ═══

Partial participation ⇒ ———



logical database design using (Relational model):

Database is rep as collection of relations (tables)

Consists of rows & cols / Records / Tuples / Attrs

*) Relations are described using Relation schema

Structure of a rel

Name | Attr | Domain of Attr | Conditions / Constraints.

Ex:- student — Name

Rno	Name	Marks	— Attr
Number(4)	string(20)	Number(3)	— Domain of Attr
Primary key	Not null		— Constraints.
10	Krishna	93	— Relation Instance (Tuple present in a rel at a particular time)

Components in Relational model: DICK

(1) Degree of a rel : Specifies num of attributes

(2) Cardinality of a rel : specifies num of tuples in a Relation instance.

Integrity Constraint :

It is a condition on a database schema and restricts the data that can be stored as an instance of a database.

- primary key – (Unique + Not Null)
- unique – (No dups allowed)
- Not NULL – (Dup allowed + Nulls not allowed)
- Check – (user defined conditions)

} Table constraints

- Foreign key (Referential Integrity Constraint) :

→ It involves two relations (parent and child) & it states that its references should be valid. ← To ensure it

*) Foreign key may contain both duplicates and NULL.

*) Every value present in foreign key must present in primary key of the referenced relation but need not viceversa

*) Deleting from parent (referenced reln) and insertion from child (referencing reln) may violate foreign key constraint.

*) Foreign key → ON DELETE / ON UPDATE → NO ACTION, CASCADE, SET NULL, SET DEFAULT

*) Ex:-

prof (parent)

ssn (Pk)	Name
1	A
2	B
3	C
4	D
5	E

delete (x) ; Insert (✓)

Teaches (child)

cid	cname	ssn (FK)
98	DB	1
99	DS	3
100	CD	1
101	CN	NULL
102	PL	6

delete (✓) ; Insert (x)

*) A relation may contain both primary key and foreign key that refers to the same relation. (self-referential relation)

Key constraint :

Set of attr which uniquely identifies each tuple in a relation

① Candidate key :

It is a minimal set of Attributes which uniquely identifies each tuple in a relation.

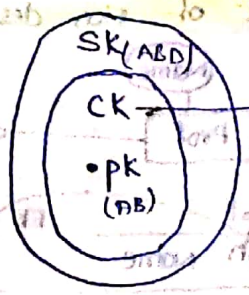
② primary key :

Select any one candidate key as primary key.

③ super key :

It is a set of Attributes that contain candidate key.

R(ABCD)



Note :

Every CK is a SK but Every SK need not be CK.

*)

$$Pk \subseteq CK \subseteq Sk$$

Ex:- student (Rno, name, father, panno)

Rno ← key : PK, CK, SK

(Name, Father) ← key : CK, SK

Pno ← key : CK, SK

(Rno, name) ← key : SK

(Rno, pno) ← key : SK

2nf

ER to Relational model:

① Entity set:

→ Each entity set is represented with one relation

→ The Attributes of the relation includes all the simple Attributes of the Entity set

→ The key Attribute of an Entity set will become primary key for the relation.

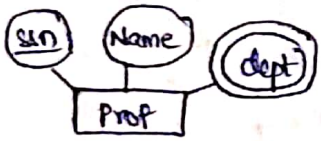
⇒

Prof

SSN	Name	Basic	TA	DA	HRA

② Entity set with multivalued Attribute:

→ If an Entity set contains multivalued Attributes. It can be represented with two relations → All simple Attributes
→ Key and multivalued Attr



→

Prof

SSN	Name	dept
1	A	{R, S}
2	B	{P, Q}

ACID violation

⇒

Prof

ssn	Name	dept
1	A	R
1	A	S
2	B	P
2	B	Q

Primary key problem

decompose

Prof

PK ssn	Name
1	A
2	B

dept

FK ssn	dept
1	R
1	S
2	P
2	Q

③ Relationship set:

→ The Attributes of the relation includes:

(i) The primary keys of participating relations and those are declared as foreign keys to their respective relation

(ii) Descriptive Attr if any

(iii) set of non descriptive Attr will become primary key for the relation.

Prof

PK SSN	Name
1	A
2	B
3	C

Teaches

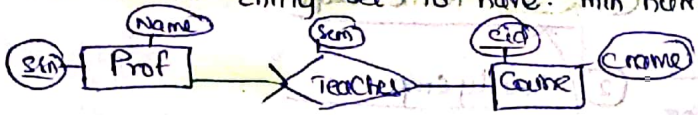
FK SSN	FK Cid	Sem
1	98	3
2	99	5
3	98	3

Course

PK Cid	Cname
98	DB
99	DS

(4) Relationship set with key constraints :

If there is any key constraint then merge the table of relationship set with the table of an entity set to have min num of relations in the relational model



Prof teaches

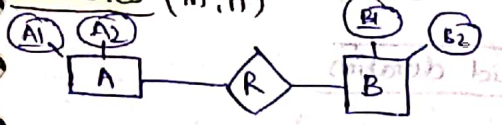
ssn	Name	sem	cid
1	A	3	98
2	B	5	99
3	C	3	98

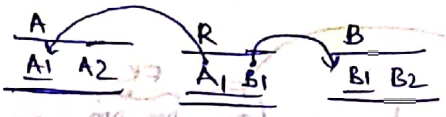
FK → Course

PK cid	Cname
98	DB
99	DS

(5) Cardinality ratios :

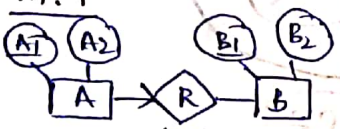
3 tables (m:n)





2 tables

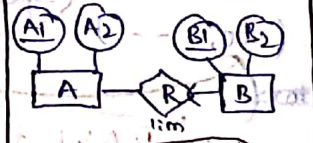
m:1



AR			B	
A1	A2	B1	B1	B2

R(AR) R(B)

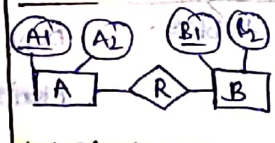
1:m



A		BR		
A1	A2	B1	B2	A1

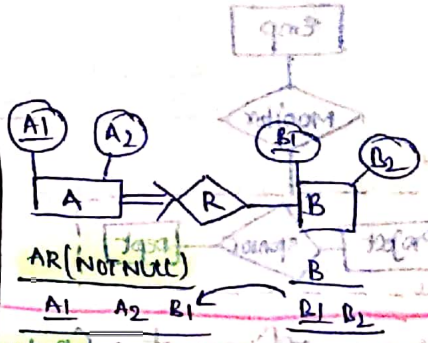
R(A) R(BR)

1:1



(i) R(AR) R(B)

(ii) R(A) R(BR) ✓



AR(NOT NULL)

A1	A2	B1	B	
			B1	B2

R(AR)

NOT NULL

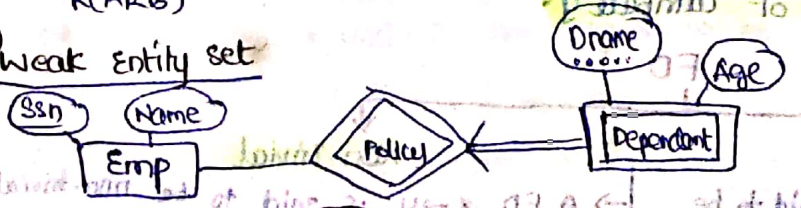
* It is also possible with one table but recommended with 2 tables to save space.

1 table

ARB			
A1	A2	B1	B2

R(ARB)

(6) Weak Entity set



Emp

ssn	Name

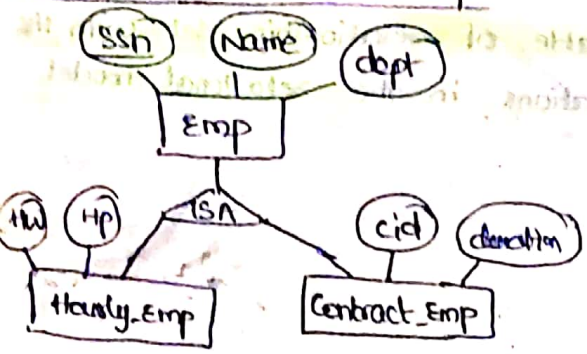
FK | ODS

dep_policy

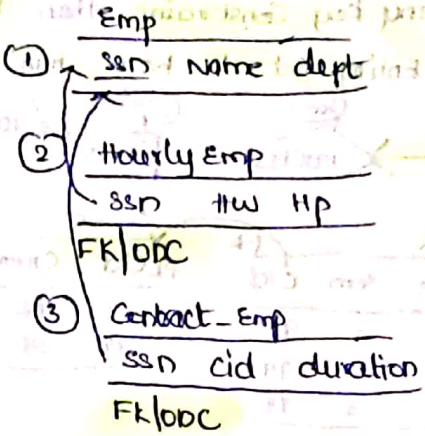
ssn	Dname	Age

→ bcoz if Emp is removed then the policies regarding the emp should be removed.

⑥ Hierarchial relationship



→ 3 relations



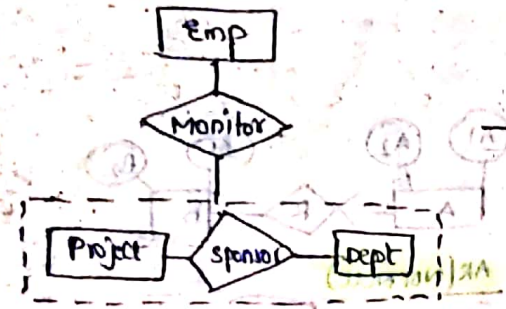
Covering Constraint :

2 tables are used bcoz Either Emp present in Hourly_Emp or Contract_Emp.

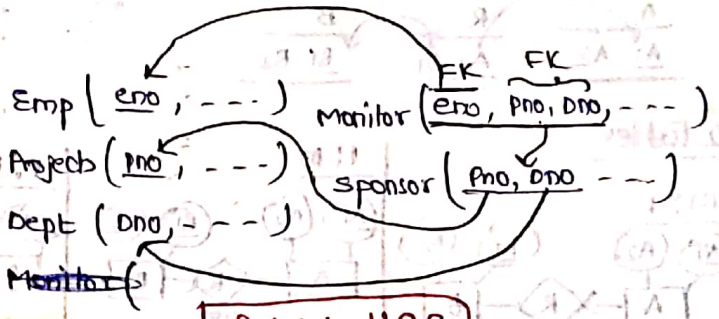
Hourly_Emp				
SSn	Name	dept	hw	Hp

Contract_Emp				
SSn	Name	dept	cid	duration

⑦ **Aggregation :**



→ 5 tables.



RAW HCE

Schema refinement : (Redundancy)

Problems due to dependency:

	Prob	sol
① Repeated storage	✓	✓
② Insertion problems	✓	x
③ Updation problems	✓	✓
④ Deletion problem	✓	✓

→ Decomposition

→ Due to FD's.

Functional dependency :

A FD $x \to y$ says that of any two tuples agree on the values on attribute x then they must also agree on the value of attribute y.

*)

x	y
x_1	y_1
x_2	y_3
x_3	y_2
x_4	y_1

x ✓ ✓ x

$x \to y$

FD

Trivial

→ A FD $x \to y$ is said to be trivial iff $y \subseteq x$

Ex:- $AB \to A$, $AB \to B$, $AB \to AB$

*) Trivial FD will always satisfy FD's in a relation

Non-Trivial

→ A FD $x \to y$ is said to be non-trivial iff there is atleast one attr on RHS which is not the subset of LHS.

Ex:- $AB \to BC$

$AB \to C$

Properties of FD: (RATUD)

① **Reflexivity**: If $X \supseteq Y$ then $X \to Y$ is a FD (Trivial)

Ex:- Hero, Title → Hero

② **Augmentation**: If $X \to Y$ is a FD then $XZ \to YZ$ is also a FD but if $XZ \to YZ$ is a FD then $X \to Y$ may or maynot be possible.

Ex:- Title → Hero

Title, prod → Hero, prod.

③ **Transitivity**: If $X \to Y$ and $Y \to Z$ is a FD then $X \to Z$ is also FD.

④ **Union**: If $X \to Y$ and $X \to Z$ is a FD then $X \to YZ$ is also FD.

⑤ **Decomposition**: If $X \to YZ$ is a FD then $X \to Y$ and $X \to Z$ is also FDs.

Closure set of FD's (F^+):

It is a set of all FD's that can be determined using the given set (F) of FD's.

Ex:- R(ABC)

$F: \{A \to B, B \to C\} \Rightarrow F^+ = \{A \to B, B \to C, \underbrace{AC \to BC}_{\text{Aug}}, \underbrace{A \to C}_{\text{Trans}}, \underbrace{AC \to B}_{\text{Decomp}}, \underbrace{AC \to C}_{\text{Reflex.}} \dots\dots\}$

Not feasible bcoz of too many possibilities.

$F \subset F^+$

Attribute closure (x^+):

It is a set of all attributes that can be determined using the given set of attributes.

Ex:- R(ABC)

$F: \{A \to B, B \to C\}$

$A^+ = \{A, B, C\}$

$B^+ = \{B, C\}$

$C^+ = \{C\}$

$AC^+ = \{A, C, B\}$

$ABC^+ = \{A, B, C\}$

Applications of Attribute Closures

finds

- Additional FD's
- Equivalence of FD's
- minimal set of FD's
- Keys of a relation.

① Finding Additional FD's:

A FD $X \to A$ is possible iff X^+ contains A.

② Equivalence of FD's:

The two FD's F and G are said to be equivalent iff

$F^+ = G^+$ → Not feasible.

So, If Every dependancy of F is determined by G (G Covers F) and Every dependancy of G is determined by F (F Covers G) then $F = G$.

⇒ Feasible

Steps to Check F = G:

F Covers G ⇒ (i) write G's FD's $(X_1 \to Y_1, X_2 \to Y_2)$

(ii) find X_1^+ in F, X_2^+ in F

(iii) If X_1^+ Covers Y_1 & X_2^+ Covers Y_2 then the FD's satisfies and F Covers G

Finding minimal set of FD's : (canonical / irreducable set)

A minimal cover for the set of FD's (F) is a set of FD's (G) such that every dependancy of F is in the closure of G.

Step(1) : Minimize LHS of each FD's

$AB \rightarrow C$

A can be deleted if B^+ contains A
B can be deleted if A^+ contains B

Step2 : Minimize RHs of each FD

verify each FD such that whether it can be determined from the remaining set of dependencies if possible consider such dependancy as redundant and remove from the minimal set

*) The given set of dependencies may have more than one minimal set and all the minimal sets are logically equivalent.

Ex:- Find minimal set

$F: \{A \rightarrow C \quad AC \rightarrow D \quad E \rightarrow AD \quad E \rightarrow H\}$ $\Rightarrow A^+ - \{A, C\} \quad C^+ - \{C\}$

$A \rightarrow C$ $A \rightarrow D$ $E \rightarrow AD$ $E \rightarrow H$

$E \rightarrow A$ $E \rightarrow D$ (2nd)

$G: \{A \rightarrow C \quad A \rightarrow D \quad E \rightarrow A \quad E \rightarrow H\}$

$G' = \{A \rightarrow CD, E \rightarrow AH\}$

(4) Finding keys of a relation :

If x^+ contains all the attributes of a relation then x is called superkey of a relation
If x^+ is minimal set then x is candidate key of a relations. ie

x^+ contains all Attr → superkey
↓ minimization
x → candidate key
↓ select one
Primary key.

Max possible superkeys $= 2^n - 1$

Ex:- $R: \{R, A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Properties of Decomposition :

a) Loseless Join decomposition (must & should) :

The decomposition of R into R_1 and R_2 is said to be loseless iff

$$R_1 \bowtie R_2 = R$$

Ex:-

R

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_2

R_1

A	B
a_1	b_1
a_2	b_1
a_1	b_2

$\bowtie$

R_2

A	C
a_1	c_1
a_2	c_1
a_1	c_2

$R_1 \times R_2 = S$

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_1	b_2	c_1
a_1	b_2	c_2

$S \neq R$

Spurious tuple (Lossy Join)

R_1

A	B
a_1	b_1
a_2	b_1
a_1	b_2

$\bowtie$

R_2

B	C
b_1	c_1
b_2	c_2

$R_1 \times R_2 = S$

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_2

$S = R$ (Loseless Join)

Note:

The decomposition of R into R_1 and R_2 is said to be loseless iff the attribute common must be a key in either R_1 or R_2 or both.

Ex:- R(ABC) F: {A → B}

R_1(AB) R_2(BC) ⇒ A is a key

$AB \cap BC \Rightarrow B$

B is not a key

~~lossless~~ lossy join

R_1(AB) R_2(AC)

$AB \cap AC = A$

A is a key

loseless join.

(2) Dependancy preserving decomposition (desirable):

The decomposition of R with dependencies F into R_1 and R_2 with the dependencies F_1 and F_2 respectively is said to be dependancy preserving iff $F^+ = F_1^+ \cup F_2^+$

Ex:- R(ABC)

F: {A → B, B → C, C → A}

R_1(AB) R_2(BC)

R_1(AB)	R_2(BC)
F_1: A → B, B → A	F_2: B → C, C → B
$B^+ = \{B, C, A\}$	$C^+ = \{C, A, B\}$

Now, C → A Compute C^+ in $F_1 \cup F_2$

$F_1 \cup F_2 = \{A \to B, B \to A, B \to C, C \to B\}$

$C^+ = \{A, B, C\}$

C → A is possible.

∴ Decomposition is dependancy preserving

Normal Forms: To Eliminate data redundancy, Ins, del, update anamolies.

1. Unnormalized relation
   ↓ Remove MVA, Composit Attr
   1NF (only Atomic values)
   ↓ Remove PFD
   2NF (only FFD)
   ↓ Remove TD's
   3NF ($X \to A$) (SK (or) PA)
   ↓ Remove prime transitivity
   BCNF ($\frac{X}{SK} \to A$)

~~Imp~~

BCNF : SuperKey → any

3NF : superkey → prime Attr

2NF : Non-Key → Non-Key

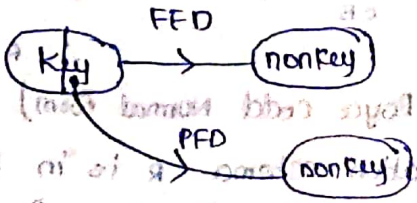
1NF : part of Key / Prime Attr → non-Key

*) Every relation in the relational model must be in 1NF

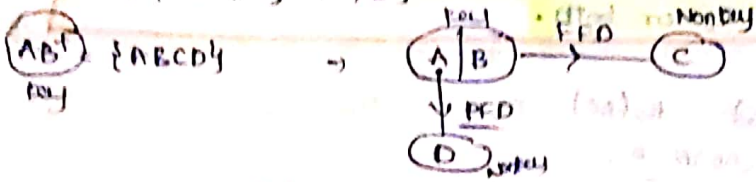
*) Partial Functional dependencies:

PFD: part of key → Non key

FFD: Key → non key

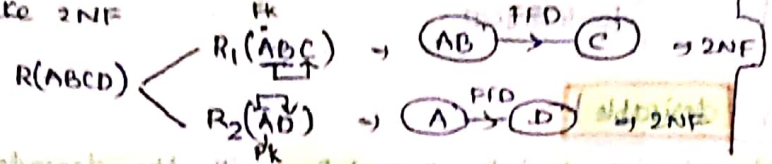


Ex:- R(ABCD)
F: {AB→C, A→D}

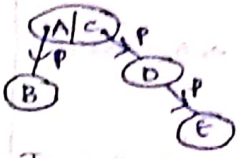


R is in 1NF

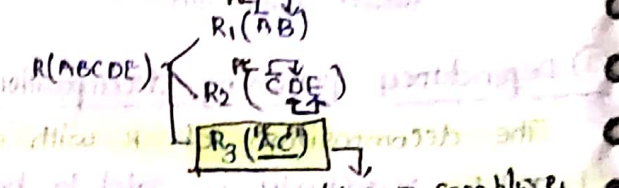
To make 2NF



Ex:- R(ABCDE)
F: {A→B, C→D, D→E}
AC+ = {A B C D E}
R is in 1NF

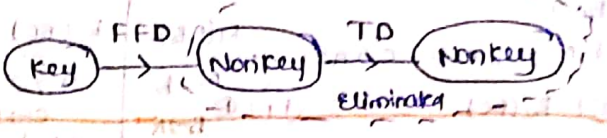


To make 2NF



*) Transtive dependencies :

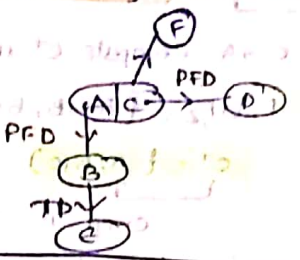
TD: Nonkey → Nonkey

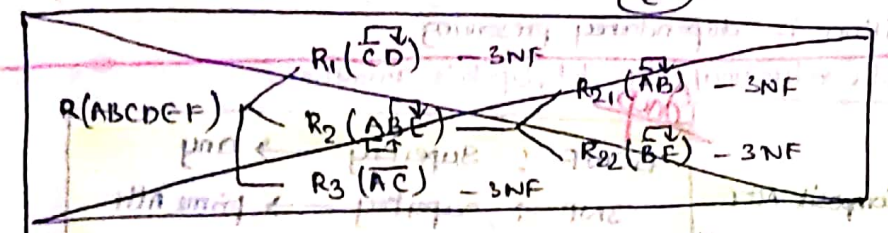


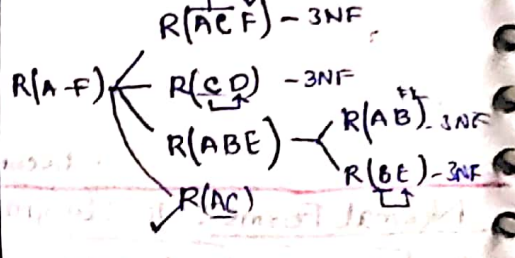
*) If a relation contains a key with only one attribute is always in 2NF

Ex: R(ABCDEF)
F: {AC→F, A→B, B→E, C→D}

AC+ = ACFBED ✓
CK







*) A relation schema R is in 3NF if whenever a nontrivial FD of the form X→A holds X is a superkey or A is a prime Attribute

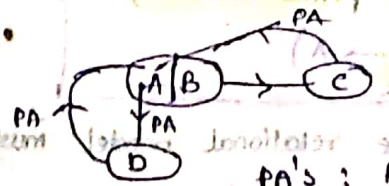
X → A
(SK) or (PA)

Prime Attribute : An Attribute which is a part of candidate key :

*) If all the attributes are prime Attributes then the relation is in 3NF

Ex:- R(ABCD)
F: {AB→C, A→D, D→A, C→A}

CK : AB
DB
CB



PA's : A, B, C, D

R is in 3NF

BCNF (Boyce Codd Normal form)

A relation schema R is in BCNF iff whenever a non-trivial FD of the form X→A holds X must be superkey of a relation.

X → A
(SK)

Ex:- R(ABC)

F: { AB → C (SK) C → A (PA) }

CK: AB
CB

R is in 3NF but not in BCNF

PA's → ABC

To make BCNF

R(ABC) < R1(AC) PK
R2(BC) FK

But AB → C is lost

Not dependency preserving

If it is reqd then go to 3NF or move to BCNF

*) If all the determinants are superkeys then the relation is in BCNF.

*) A rel with two Attributes is always in BCNF

SQL (Structured Query language)

Forward to Hadoo

Order of Evaluation of SQL Clauses (FWG HEDSO)

From
where
Groupby
Having
Expressions
Distinct
Set Operns
order by

Conditions for selecting rows

- and
- or
- not
- in
- not in
- between and
- not between and
- is NULL
- is not NULL
- like '%'
- like '_'
- <, <=, >, >=, =
- <> => Not.

pattern matching operator

'R%' → start with R
'%R' → End with R
'%R%' → Contains R
'___' → length 3
'__%' → min length = 2

NULL : not a zero, not specified, unknown, Inapplicable, known but not specified.

*) The operator 'in' is a set comparision operator in which it compares a value with a set of values and returns true if the value is in the set

*) Select *
From students.
where F 1 = 2 → 0 rows ret
T 1<2 → All rows ret
F Null = 0 → 0 rows ret
F null <> 0 → 0 rows ret

*) select *
From student
where F null = null → 0 rows ret
F null <> null → 0 rows ret
T null is null → All rows ret

*) select *
From Companies
order by name desc, Turnover asc

⇒ First it sorts by name in desc order
If conflict is present in name then among the conflicts it sorts by Turnover in asc order

Aggregate functions:

*) Aggregate functions should not be used in where clause

Number, Text, Date ← min, max
Number ← sum, Avg
any datatype ← Count

return 1 row always
It ignores NULL

Ex:-
* student

Name	M1	M2
A	10	5
B	Null	10
C	20	15
D	Null	20
E	30	null
F	Null	25

⇒ select sum(M1+M2)
From student ;

→ 50 [10+5, 20+15]

Nulls are ignored

*) Select Avg(M1) From student → 20 $\left[\frac{10+20+30}{3}\right]$ ⇒ Nulls are ignored. | Select sum(M1)/count(M1) From student → 10

*) Count(Constant) ⇒ Num of rows in a relation

Ex:- Count(10), Count('sai')

*) select Name From student where M1 = max(M1); → Error

- where → select rows based on cond
- MAX → col in consideration. ↓ 30

Difficult for query processor to consider

*) Aggregate functions cannot be used in orderby clause

Select Name From student orderby Max(M2); → Error.

*) An Attribute and aggregate fn can't be used together in the select clause without the group by clause because it violates 1NF

Select Name, Max(M1) From student → violates 1NF ⇒ Error

Set Manipulation operators

- Union
- Intersect
- Except

→ applied among two Compatable relational instance
- → same num of Attr
- → Consider From left to right the domain should be same.

Ex:- (select cname From Depositor) Union / Intersect / Except / MINUS (select custname From borrower)

ABC

Union – A B C D E

Intersect – C ← set open Eliminates duplicate values

except – ABDE

MINUS – AB

Groupby and Having :

Groupby → used to compute aggregate fns in a group of rows

Having → It is a group qualifier; used for selection of groups

*) Every attribute used in the select clause must appear in the group by clause to have one value per group. and viceversa need not be true.

Ex:- select branch, year, count(*) From student groupby branch; (Error) → Soln

> select branch, year, count(*) From students (✓) Group by branch, year;

=> Violates 1NF

branch	year	Count(*)
CSE	II	2
CSE	I	1

=>

branch	year	Count(*)
CSE	{I, II}	3

*) Finding num of students other than cse in each branch.

> select branch, Count(*)
From student
Group by branch
Having branch <> 'cse';

↓
Correct but not Efficient

Efficient Query →

> select branch, Count(*)
From students
where branch <'cse';
Group by branch;

*) The having clause contains either an aggregate function or an attribute used in the groupby clause to have one value for the group in the group selection.

*) In the absence of Groupby, the having clause consider the whole rel^n as one group

Ex:- > select Count(*)
From student
having Count(*) ≥ 3;
⇒ It ret o/p

*) Nulls are not ignored in Groupby Clause.

Sol:-

T	
A	B
1	a
1	b
2	a
null	b

> select A, Count(*)
From T
Groupby A;
⇒

o/p →

A	Count(*)
1	2
2	1
null	1

Join:
used to Join related tuples from 2 relns into a single tuple

> select name
From student, course
where student.cid = Course.cid. and
Join condition.

Cname = 'DB'	
A	
A	

*) Join

11 Natural Join (Matching rows)
Implicit equality on all attr having same name

Ex:- > select name
From student Natural join course
where Cname = 'DB';

10 Left outer Join
→ It ret all the rows from left side reln even if there is no matching row in the right side reln

01 Right outer Join
→ It ret all rows from right side relation even if there is no matching row in the left side reln.

00 Outer Join (No matching rows)
Full outer Join
→ It rets all the rows from both the relns even if there is no matching row with other relation.

FOJ = LOJ ∪ ROJ

Examples on outer Join

> select name, Cname
From professor natural left Join Course_Incharge;

→ Left outer Join

(or)

select name, Cname
From Professor P, Course_Incharge C
where P.ssn = C.ssn(+);

*) Select cname, Name
From professor Natural right {outer} Join Course_Incharge;

(or)

Select cname, Name
From Course_Incharge Natural left {outer} Join Professor;

(or) ⇒ Right outer Join.

Select cname, Name
From professor P, Course_Incharge C
where P.ssn(+) = C.ssn

*) select Name, cname
From professor Natural full {outer} Join Course_Incharge;

(or)

Select cname, Name ⇒ Full outer Join.
From professor P, Course_Incharge C
where P.ssn(+) = C.ssn(+);

Ex:-

Professor

ssn	name
1	A
2	B
3	C

Course_Incharge

cid	cname	ssn
98	DB	1
99	DS	3
100	CD	1
101	PL	Null

⇓

left outer Join :

Name	cname
A	DB
A	CD
B	Null
C	DS

Right outer Join :

Name	cname
A	DB
C	DS
A	CD
Null	PL

Full outer Join

Name	cname
A	DB
A	CD
B	Null
C	DS
Null	PL

*) Consider two rels's R and S with m and n rows respectively then the min and max num of tuples in R Natural Join S is

Min → 0 (If FK contains Null)
Max → m×n (If no attr is common in R and S)

Cartesian product
⇓
when there are no attr in common

*) Consider the following rels student(Rno, name, marks) Enroll(Rno, Cno)
Assume that there is no violation of Ref Int Constraint, let student, Enroll had 120, 8 tuples
find max, min in student Natural Join Enroll 8, 8

⇒ Max case

Rno	Rno, Cno
1	(1,1)
	(1,2)
	(1,8)
2	
120	

⇓
8 tuples

Min case:

Rno	Rno, Cno
1	(1,1)
2	(2,2)
8	(8,8)
120	

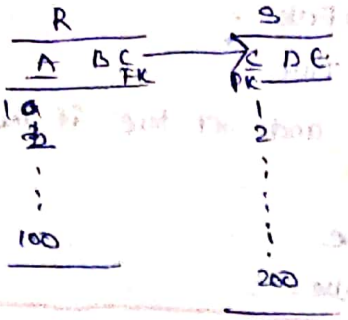
⇓
8 tuples.

*) Consider a rel R(A B C) S(C D E)

F: {A→BC, C→DE}

R: 100 tuples S: 200 tuples

find max, min R natural join S.



Max case → If all c's in R are 1 → 100

Min case → If all c's in R are NULL → 0

Nested Query: (Query within Query)

→ The Inner query is evaluated first and its result is used by the outer query.

outer query → dependent on Inner query

Inner query → Independent

→ Execution approach: Bottom to top.

Q Name of the student with second highest marks.

```
> select Name
  from student
  where marks = ( select max(marks)
                  From student
                  where marks not in (select max(marks)
                                      From student)
```

not in / < : both are used for 2nd highest, but < is used for 3rd, 4th - - -)

but num of Nested Queries increases.

↓ sol

Using Correlated Query

Correlated SubQuery:

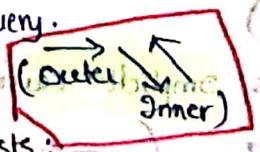
→ The Inner query is evaluated once for each row of outer query.

→ The Inner query depends on the row of outer query.

→ The outer query depends on the result of Inner query.

→ Execution approach: outer to Inner to outer

top to bottom to top



Q name of the student with 3rd highest marks.

```
> select name
  From student s1
  where ( select Count (distinct s2.marks))
          From student s2
          where (s1.marks <= s2.marks) = 3 ;
```

Exists:

→ It is used to compare with an Emptyset

→ Exist (Result) → True

Exist (Emptyset) → False.

Set Comparision operators :

① **operator Any**: It compares a value with each value in the set and set true if the given condition satisfied for atleast one value in the set.

Ex:- $x < \text{any}(10, 20, 30)$
$x < 10$ or $x < 20$ or $x < 30$

$= \text{any}(\text{empty}) \Rightarrow$ False.
$= \text{any}(\text{Result}) \Rightarrow$ True

② **operator All**: It compares a value with each value in the set and set true if the given condition satisfied for all values in the set.

Ex:- $x < \text{all}(10, 20, 30)$
$x < 10$ and $x < 20$ and $x < 30$

$= \text{all}(\text{empty}) \Rightarrow$ True
$= \text{all}(\text{Result}) \Rightarrow$ False.

Relational Algebra : (Rep query Evaluation plan)

→ procedural language
→ It describes a step by step procedure for computing desired answer.

① **selection operator** (σ_{cond}) : select the rows

② **projection operator** (Π_{Attr}) : projection of cols.

*) Elimination of duplicates is implicit in both Relational Algebra and Calculus.

③ **set operations** :

$R \{ \cup, \cap, - \} S$ → Compatable
→ same num of attr
→ Domain of attr should be same

④ **cartesian product (x)** :

R x S contains all the pairs of R and S tuples.

⑤ **Join operator** ($\bowtie_{cond}$)

It is defined as cartesian product followed by selection and then projection

(A) $\Pi_{cust\_name}\left(\sigma_{branch='xyz'}(Borrower \bowtie_{Borrower.loanno = loan.loanno} loan)\right)$

→ Select distinct cust_name
→ From Borrower join loan ON Borrower.loanno = loan.loanno
where branch = 'xyz'.

⑥ **Natural Join** ($\bowtie$)

Join condition is implicit Equality on all attributes having the same name.

$R \bowtie S = \Pi_{A,B,C,D}\left(\sigma_{R.B=S.B \wedge R.C=S.C}(R \times S)\right)$

R		S		R ⋈ S			
A	B	B	C	A	B	C	D

⑦ **Rename operator** (ρ)

used to represent the result of a relational algebra expr with symbolic name.

∴ Readability ↑

(A) → $\rho(X, Borrower \bowtie loan)$
$\rho(Y, \sigma_{branch='xyz'} X) \rightarrow \Pi_{cust\_name} Y$.

(8) Division operator:

Consider two relations A and B where A has two attributes. x and y and B has one attribute y with same domain as in A. then the set of x values such that for every y value in B there is a tuple in Relation.:

Ex:-

A		B1		A ÷ B1
x	y	y	⇒	x
x_1	y_1	y_1		x_1
x_2	y_1	y_2		
x_3	y_1	y_3		
x_1	y_2			
x_2	y_2	B2		A ÷ B2
x_1	y_3	y	⇒	x
		y_1		x_1
		y_2		x_2

Query:

$$\Pi_{cust\_name}\left(\sigma_{branch='Hy3'}(Borrower \bowtie loan)\right) \Rightarrow \Pi_{cust\_name}\left(Borrower \bowtie \left(\sigma_{branch='Hy3'}(loan)\right)\right)$$

optimized query :

Relational calculus :

- → non-procedural language
- → Declarative language

Types of RC

Tuple relational Calculus (TRC)

- → result is described as set of tuples
- → The set of tuples of a relation instance are rep symbolically using a name called Tuple variable
- → Form of TRC Query

$\{ T \mid P(T) \}$ — Tuple var; P(T) → formula which describes T

Domain Relational Calculus

- → result is described as set of values of an attribute domain.
- → The set of values of an attribute domain is represented symbolically by a name called Domain variable.
- → Form of DRC query

$\{ \langle x_1, x_2, \ldots x_n \rangle \mid P(x_1, x_2, \ldots x_n) \}$ — $x_1 \ldots x_n$: Domain var; P(...): formula which describes domain var.

Find name of the student whose marks is above 10

TRC : $\{ T \mid \exists S \in student\ (S.marks > 10 \wedge T.name = S.name) \}$ — T.name = S.name: Compulsary.

DRC : $\{ \langle N \rangle \mid \exists R, M\ (\langle R, N, M \rangle \in student\ (M > 10)) \}$

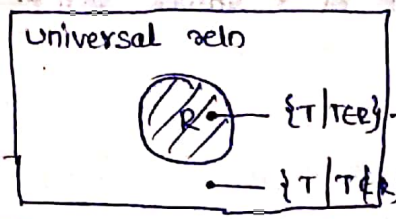
Find name of the student and the fee paid for the Cno = 99

TRC : $\{ T \mid \exists S \in student\ (T.name = S.name \wedge \exists E \in Enroll\ (E.Cno = 99 \wedge E.Fee = T.Fee)) \}$

DRC : $\{ \langle N \rangle \mid \exists_{R,M}\ (\langle R, N, M \rangle \in student) \wedge \exists_{R,C}\ (\langle R, C, F \rangle \in Enroll \wedge (C = 99)) \}$

*) A query that returns infinite set of rows is called unsafe query.

*) A query that returns finite set of rows is called safe query.



Transaction Management :

It is a collection of opers that performs a single logical unit of work.

Transaction properties (ACID)

① **Atomicity :** All or none → Transaction mgr
Keep track on

② **Consistency :** Correctness ensured by User, App prgrer
responsible for.

③ **Isolation :** Each Transaction (T_x) is unaware of other T_x's → Concurrency control mgr
Keep track on. Uses certain Protocols

④ **Durability :** All updates done by T_x should become permanent → Recovery mgr
responsible for

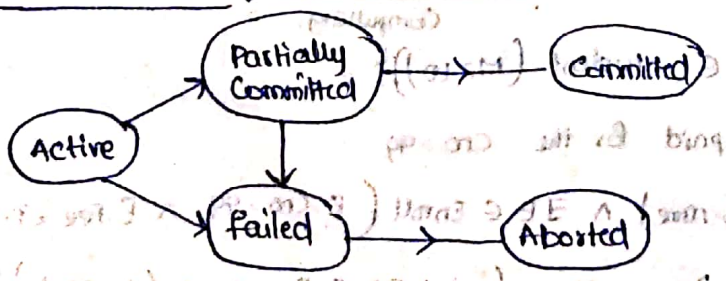
Ex:-

	Father +1000	A 1000	Brother +500
		2000	
10:01	Read(A) 1000		1000 Read(A) 10:02
10:03	A = A+1000 2000		1500 A = A+500 10:04
10:05	Write(A) 2000	2000	1500 Write(A) 10:06
	↓ All done (Atomicity)	↓ Correctness (Consistency)	

→ overall → Inconsistent

Problem : Concurrent Execution.

Transaction states :



Restart : s/w (or h/w Failure)

Kill : logical Error.

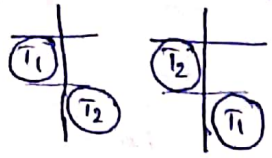
Schedule :

→ It rep the Chronological order in which the operns of a transaction is executing

① Serial schedule :

A serial schedule consists of sequence of Ins from various T_x's, where the Ins belonging to each transaction appears together in the schedule.

Ex:-



2 ways

Note :

→ A serial schedule always ensures Consistency

→ **Num of serial schedules = n!**

n = num of T_x's

② Concurrent schedule :

If two T_x's are running concurrently the OS may execute first T_x for a while then switch to second T_x and executes for a little while and then switch to first T_x and so on, is called Concurrent schedule.

*) Num of Concurrent schedules possible over n T_x's with $n_1\ n_2 \dots n_n$ operns respectively, are $= \frac{(n_1+n_2+n_3+\dots+n_n)!}{n_1!\ n_2!\ \dots\ n_n!}$

**) Num of Non serial schedules $= \frac{(n_1+n_2+\dots+n_n)!}{n_1!\ n_2!\ \dots\ n_n!} - n!$

Ex:-

T_1	T_2
$R_1(A)$	$R_2(A)$
$W_1(A)$	$W_2(A)$

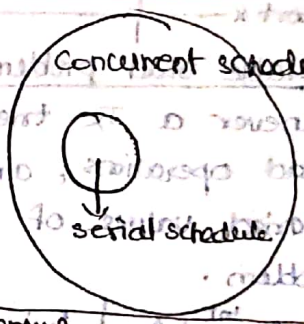
Schedules :

Serial schedules :
- $R_1\ W_1\ R_2\ W_2$
- $R_2\ W_2\ R_1\ W_1$

Concurrent schedules :

non-serial schedule :
- $R_1\ R_2\ W_1\ W_2$
- $R_1\ R_2\ W_2\ W_1$
- $R_2\ R_1\ W_2\ W_1$
- $R_2\ R_1\ W_1\ W_2$

Num of ways = $\frac{4!}{2!2!}$ = 6 ways



Why concurrency ?

→ Minimize longer waiting time

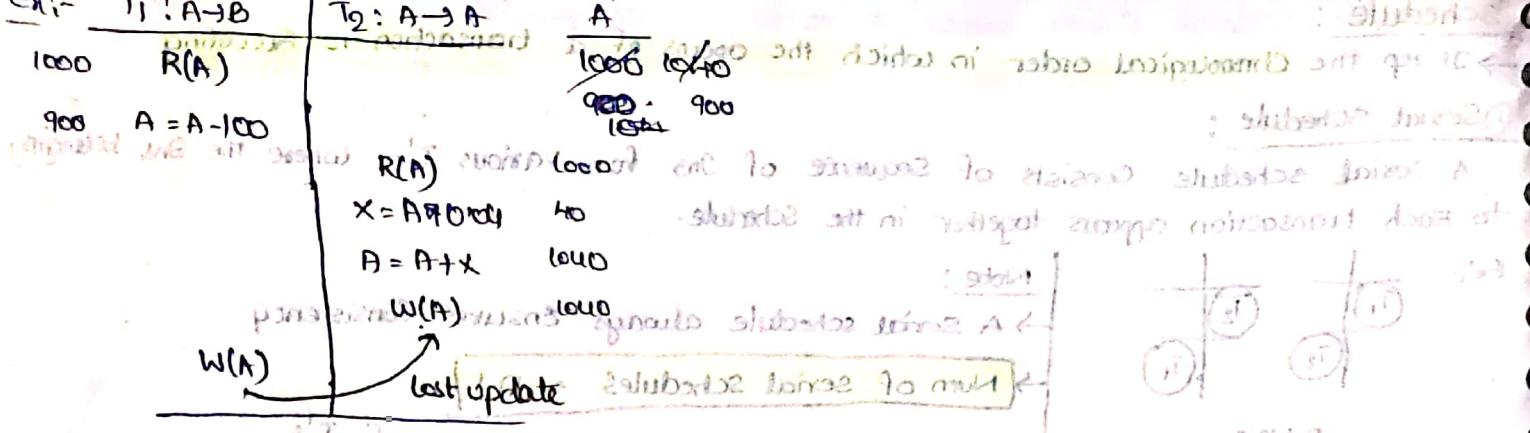
→ effective utilization of resources.

(So, every Concurrent schedule that when executed must be equivalent some serial schedule to ensure Consistency) Imp

Problems due to Concurrent schedules :

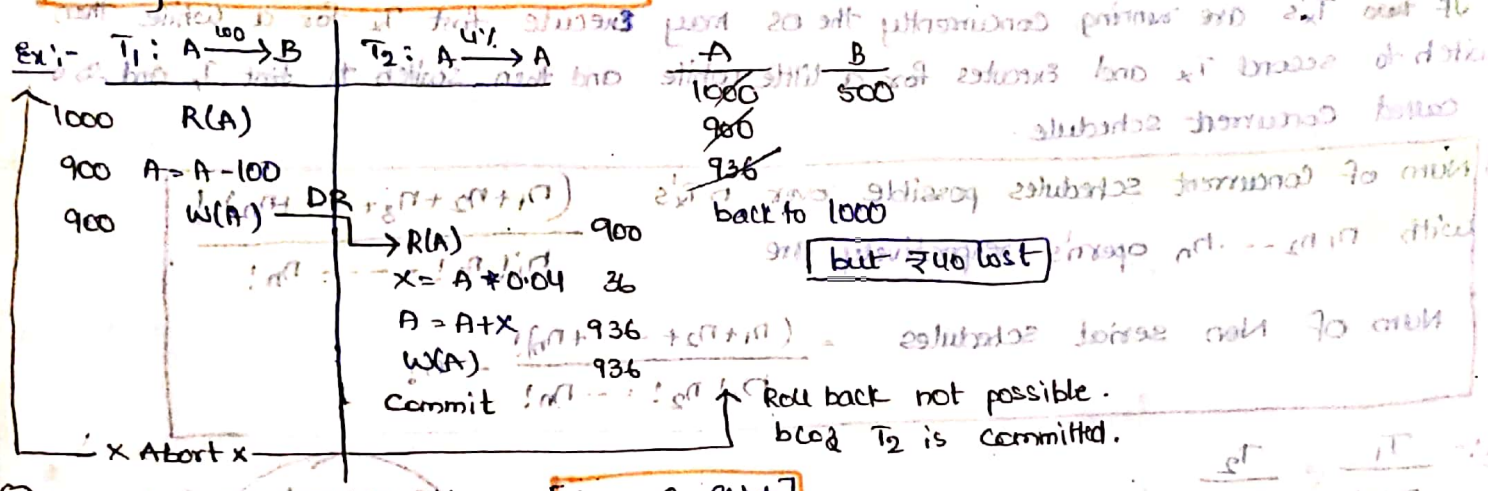
① lost update problem (W-W Conflict)

If the update of one T_x is overwritten by the update of another T_x without reading it results in the first update to become lost.



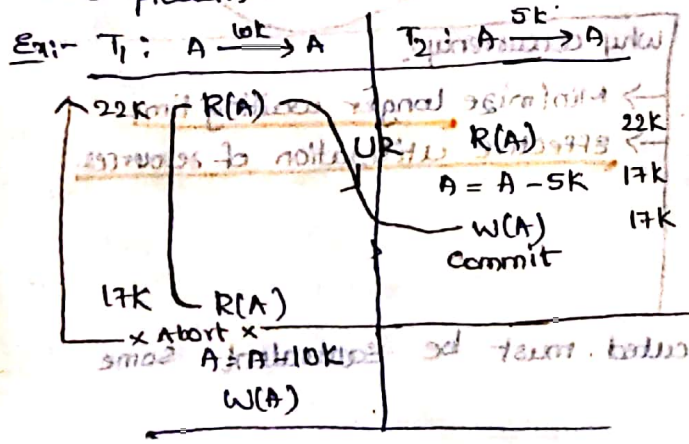
② Dirty read problem [W-R Conflict] :

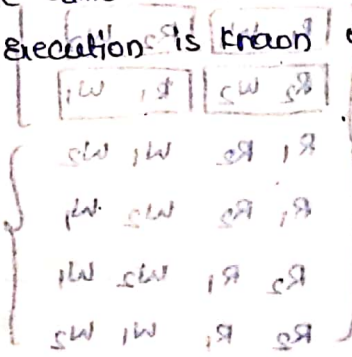
→ Reading of an uncommitted data is called dirty read



③ unrepeatable read problem : [R-W Conflict]

→ If whenever a Tx tries to read the values of a data item twice if between the two read operations, another transaction updates the same result in first Tx to read varied values of same data item during its execution is known as unrepeatable read problem.





How to identify the problem :

① Lost update : 2 writes simultaneously without any read

② Dirty read : Read of uncommitted write

③ unrepeatable read : b/w 2 reads of there is a write

④ Repeatable reads : No problem.

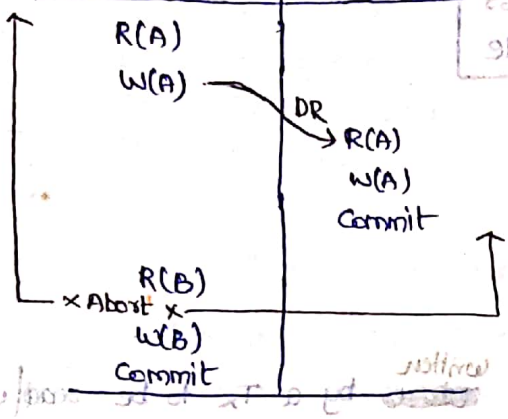
Schedules based on Recoverability :

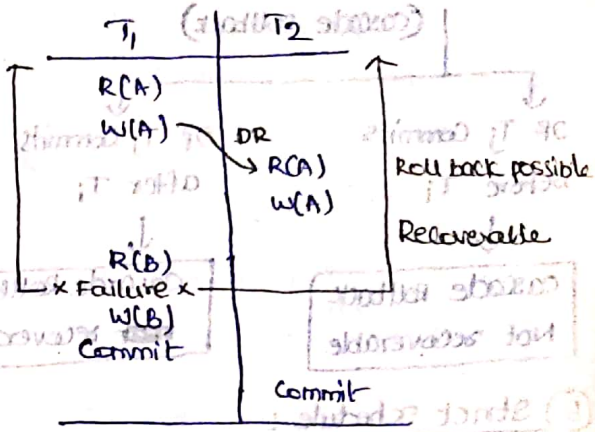
① Non recoverable schedule :

A schedule is said to be non-recoverable . if T_2 commits before the commit of T_1

→ If T_i fails after T_j commits , it requires to roll back both T_i and T_j but T_j of roll back is not possible Hence the schedule is said to be non-recoverable

Ex:- $T_1 : A \rightarrow B$ $T_2 : A \rightarrow A$





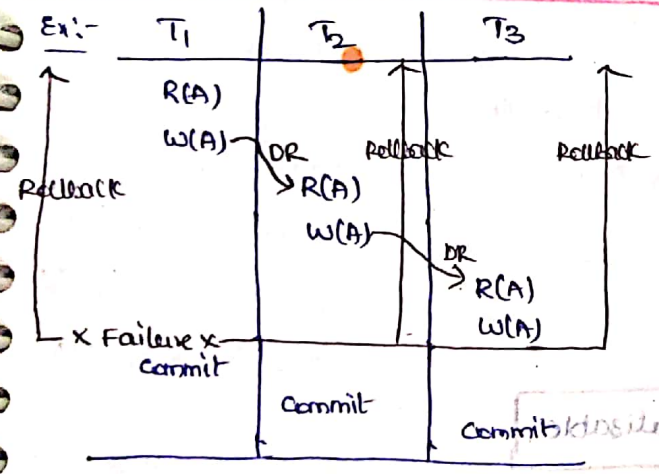
② Recoverable schedule :

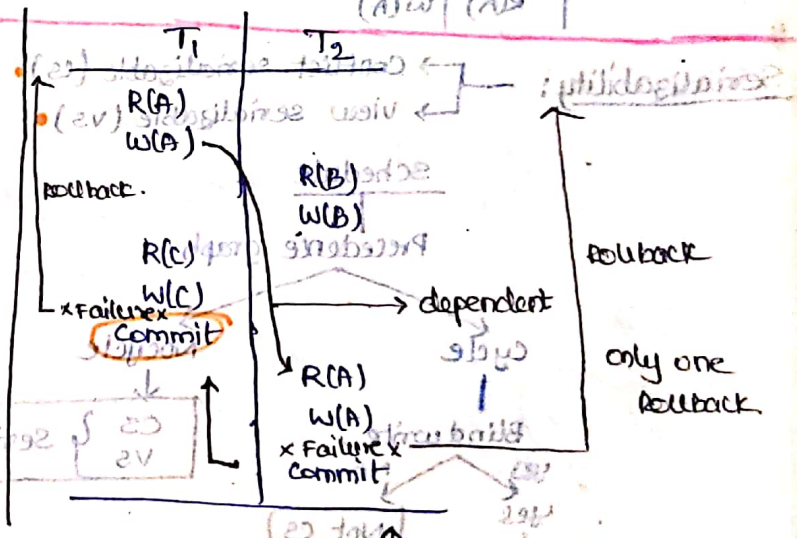
A schedule is said to be Recoverable . of Commit of T_2 is after the Commit of T_1

③ Cascading Rollbacks / aborts :

A single T_x failure if leads to more than one T_x to abort is called a schedule with cascading Rollbacks / aborts .

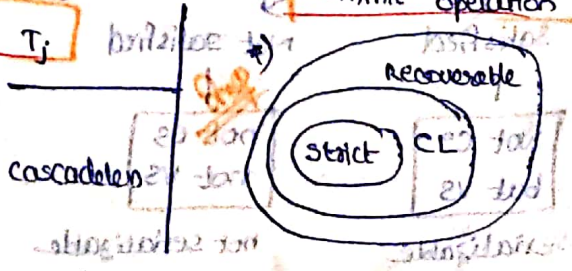
Ex:-





④ Cascadeless schedule . :

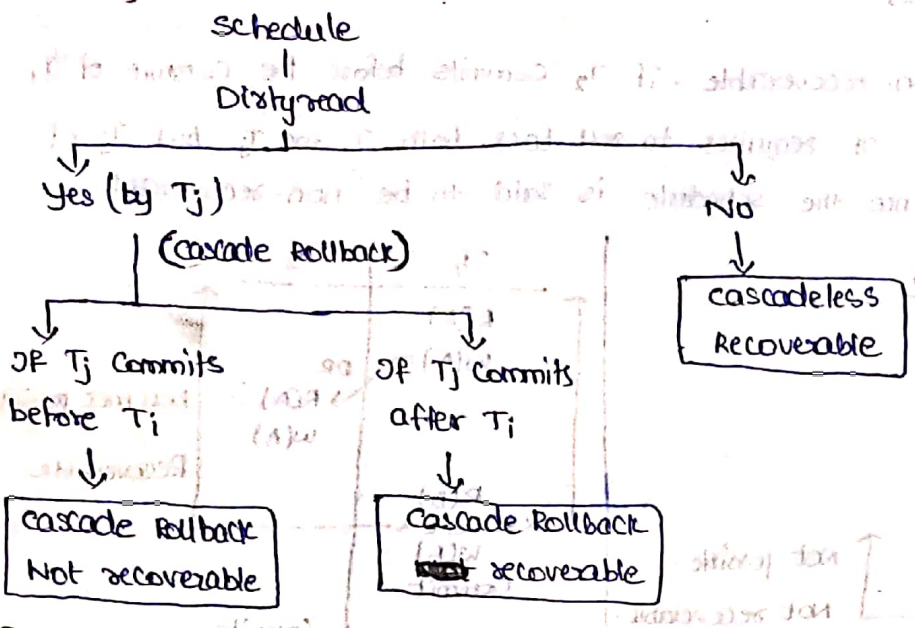
A schedule is said to be cascadeless . let for each pair of T_x's T_i and T_j such that T_j reads the value of the data item previously written by T_i then the Commit operation of T_i should appear before the read operation of T_j



*) Every cascade less schedule is recoverable but every recoverable schedule need not be cascadeless

*) Every strict schedule is both cascadeless and recoverable

Summary :



(5) Strict schedule :

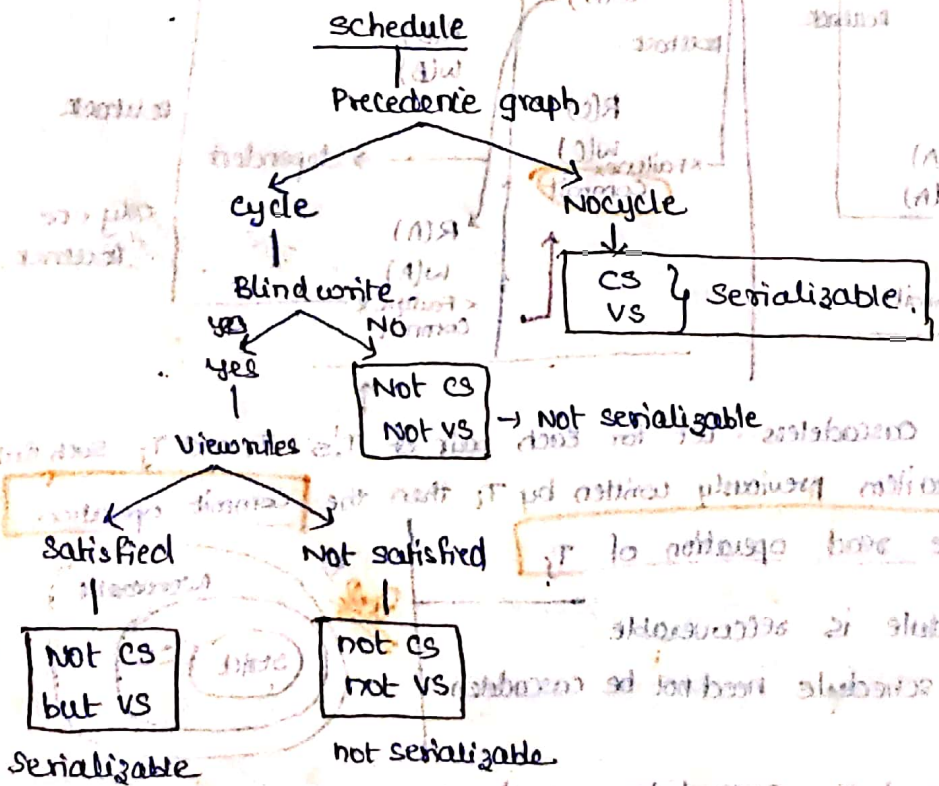
A schedule is said to be strict if a value written by a T_x to be read/written by another transaction only after T commits or aborts. is called strict schedule.

Ex:-

T_1	T_2
R(A)	
W(A)	
Commit/Abort	
	R(A) / W(A)

⇒ Both: Cascadeless Recoverable.

Serializability : → Conflict serializable (cs) → View serializable (vs)



Def of serializability:

A Concurrent schedule if Equivalent to serial schedule is said to be serializable schedule

Types of serializability

Conflict serializability

Conflict operations:

$W_1(A)$	$R_2(A)$
$W_1(A)$	$W_2(A)$
$R_1(A)$	$W_1(A)$

Two schedules are said to be Conflict Equivalent iff all the conflicts in S_1 and S_2 are same

$S_1 \stackrel{C}{=} S_2$

↓

Type & num of Conflicts same

*) A Concurrent schedule if Conflict Equivalent to serial schedule is said to be Conflict serializable schedule

Test for Conflict serializability:

① Construct precedence graph

② If there is a conflict draw an edge from T_i to T_j

③ If the directed graph has no cycle then it is Conflict serializable.

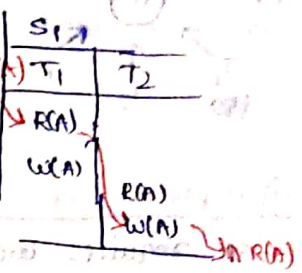
④ The order of serializability is determined based on 'Topological sort' of directed graph

View serializability

*) Two schedules are said to be view Equivalent if all the views (write(Q) – Read(Q)) must be same for Each data item.

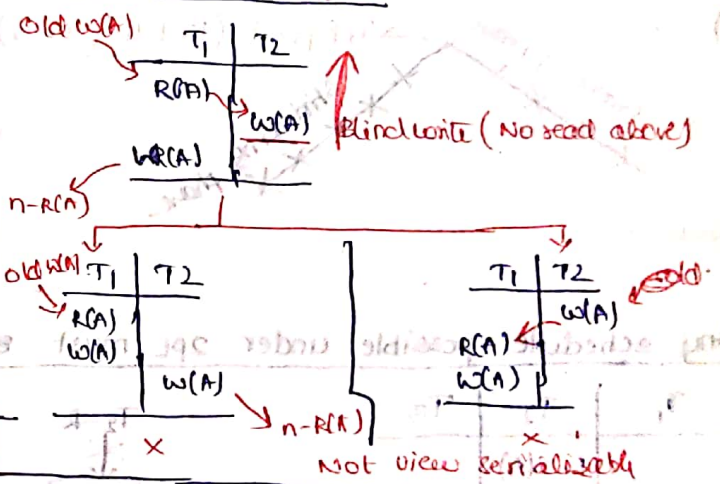
$S_1 \stackrel{V}{=} S_2$

(write(Q) – Read(Q))

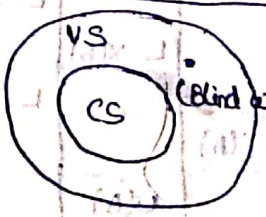


* A Concurrent schedule if view Equivalent to a serial schedule is said to be view serializable schedule.

Test for view serializable:



*) A schedule ie view serializable but not Conflict serializable then there must be atleast one Blind write oper.



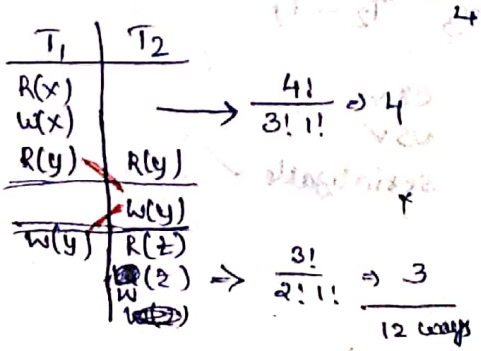
Q) Consider the following T_x's

T_1 : R(x) W(x) R(y) W(y)

T_2 : R(y) W(y) R(z) W(z)

Find num of Concurrent schedules that are Conflict serializable using T_1 & T_2.

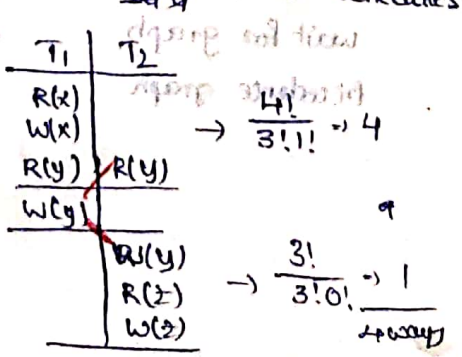
Sol) Concurrent schedules ⇒ $\frac{(4+4)!}{4!4!}$ ⇒ $\frac{\overset{2}{8}\times 7\times 6\times 5}{24\,4}$ ⇒ 70 schedules.



→ $\frac{4!}{3!1!}$ ⇒ 4

→ $\frac{3!}{2!1!}$ ⇒ 3

12 ways



→ $\frac{4!}{3!1!}$ ⇒ 4

→ $\frac{3!}{3!0!}$ ⇒ 1

4 ways

Not CS schedules ⇒ 12+4 ⇒ 16.

CS schedules ⇒ 70 − 16 ⇒ 54 ways

Concurrency Control :

① lock based protocols :

This protocol requires the data items be accessed in a mutually exclusive manner

lock modes :

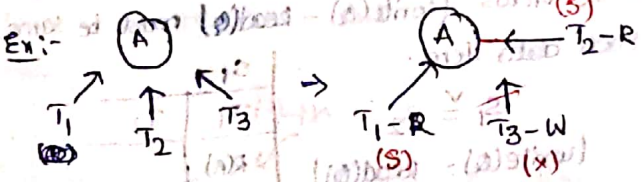
① shared lock (Lock-S) : only read

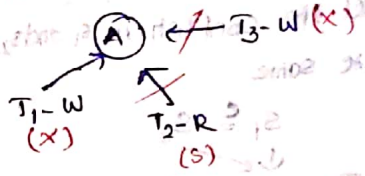
② Exclusive lock (Lock-x) : only write

Compatibility of lock modes :

	S	Ex
S	✓	✗ (unrepeatable read)
Ex	✗ (dirty read)	✗ (lost update)

Ex :-





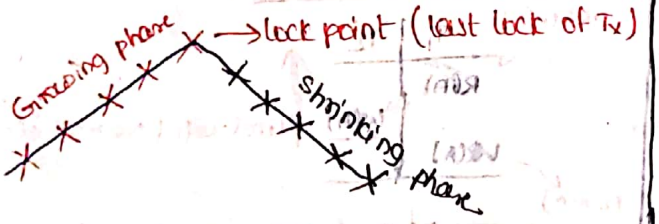
a) 2 phase locking protocol :

It requires locking and unlocking being done in two phases

Growing phase : T_x can obtain locks but can't release it

shrinking phase : T_x can release locks but can't obtain it

*)



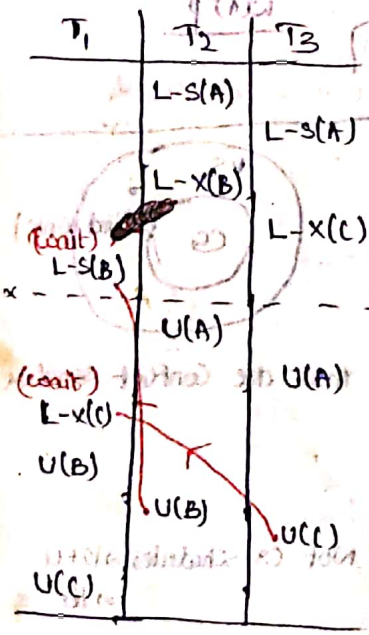
Points to remember :

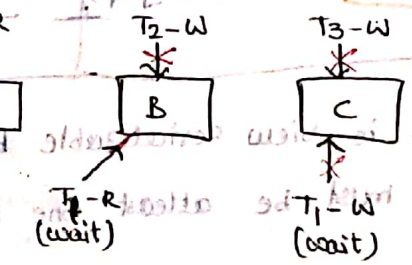
*) Any schedule possible under 2PL must ensure conflict serializability

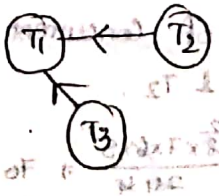
*) Cascading Rollbacks may occur under 2PL

*) Deadlocks may occur under 2PL

Any schedule possible under 2PL must ensure Conflict serializability :







wait for graph.

Precedence graph

Topological sort

$T_2 - T_3 - T_1$

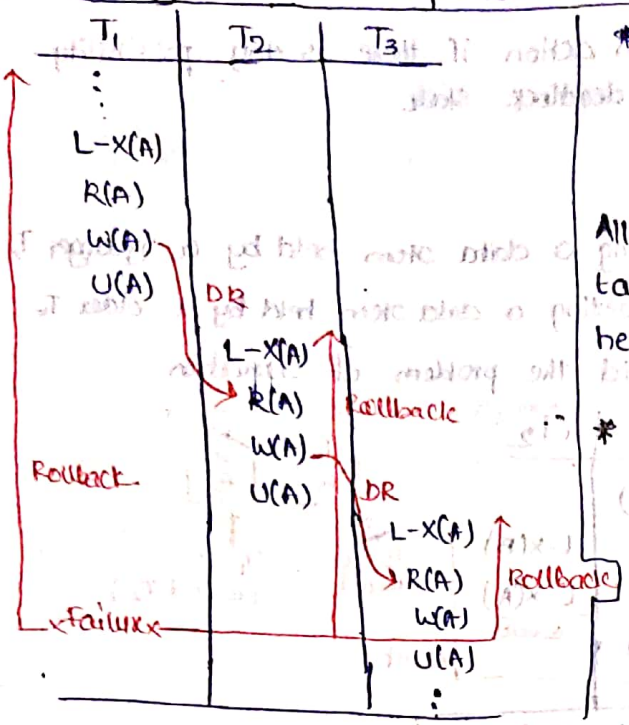
$T_3 - T_2 - T_1$ order

CS ✓

VS ✓

Serializable ✓

Cascading Rollbacks may occur under 2PL:

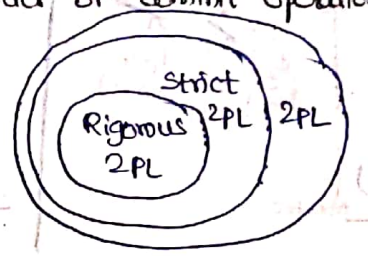


*) cascading Rollbacks can be avoided by a modification

- Strict 2PL: All the Exclusive mode locks taken by a T_x must be held until it Commits.
- Rigorous 2PL: All the locks taken by a T_x must be held until it Commits.

* In Rigorous 2PL the order of serializability is in the order of Commit operations

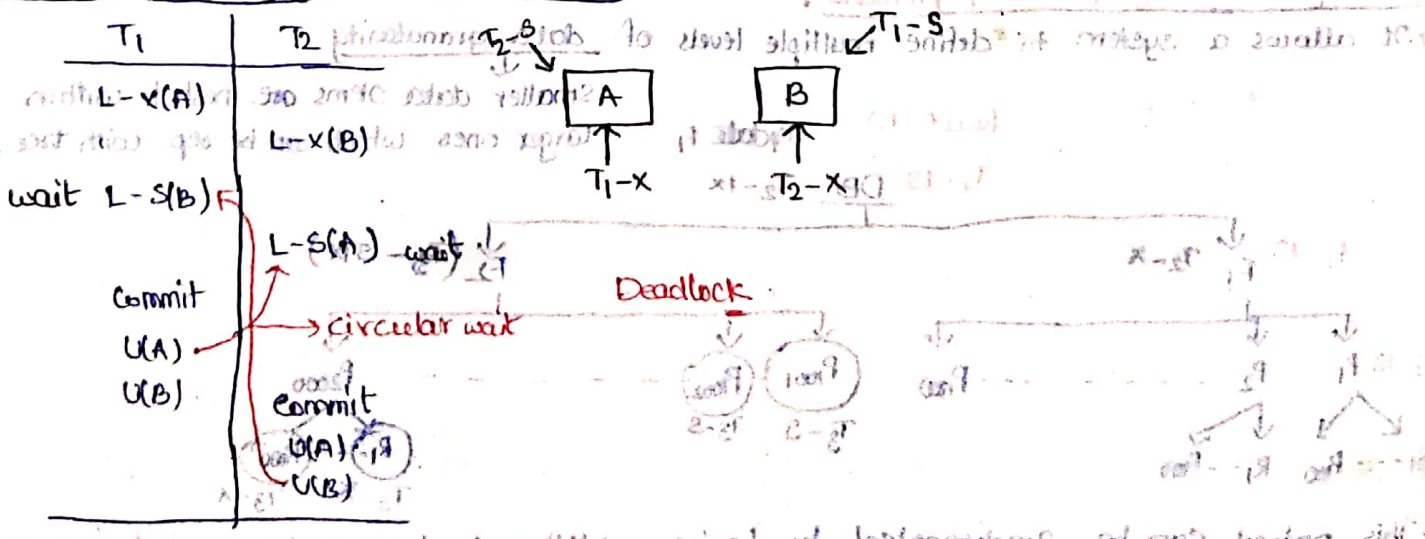
*)



*)

T_1	T_2	T_3	T_4
Not 2PL	2PL	Strict 2PL	Rigorous 2PL
L-X(A)	L-S(A)	L-X(A)	L-S(A)
R(A)	Gp R(A)	Gp R(A)	Gp R(A)
Gp W(A)	L-X(B)	W(A)	L-X(B)
U(A)	R(B)	L-S(B)	R(B)
L-S(B)	W(B)	R(B)	W(B)
R(B)	Sp U(A)	Sp U(B)	Commit
W(B)	U(B)	Commit	Sp U(A)
U(B)	Commit	Sp U(A)	U(B)
Commit		X(A) released after Commit	All locks released after Commit

Dead locks may occur under 2PL:



Deadlock handling :

Deadlock prevention : In this approach it takes an action if there is any possibility for the system to enter into deadlock state.

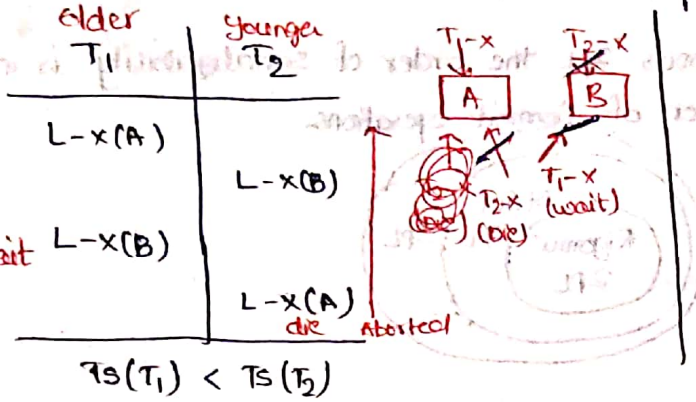
① wait-die deadlock prevention strategy :

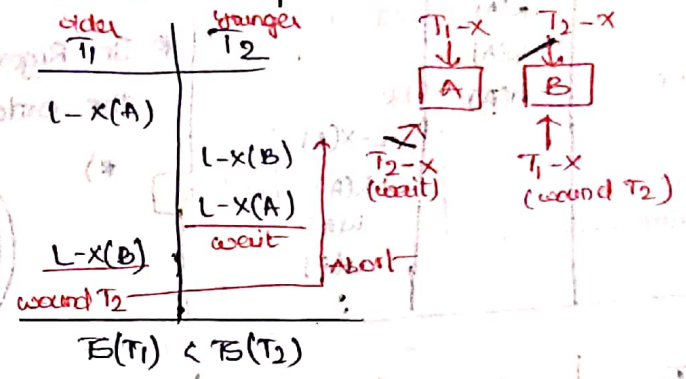
(older) (younger)

An elder T_x is allowed to wait if requesting a data item held by a younger T_x

An younger T_x is aborted and restarted if requesting a data item held by a older T_x.

with the same timestamp to avoid the problem of starvation





② Wound - wait deadlock prevention strategy :

(older) (younger)

A younger T_x is allowed to wait for an older T_x

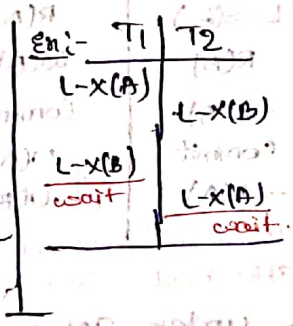
A older T_x requesting an item held by younger T_x prempts the younger T_x by aborting it.

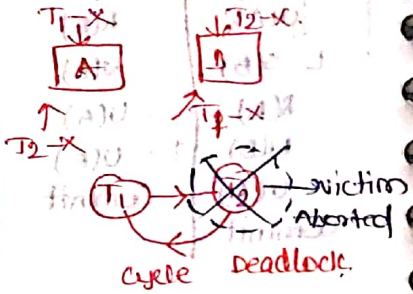
Deadlock detection :

① Construct wait for graph

② we have a state of deadlock iff the wait for graph has a cycle

③ select a victim T_x and abort it to release the state of deadlock

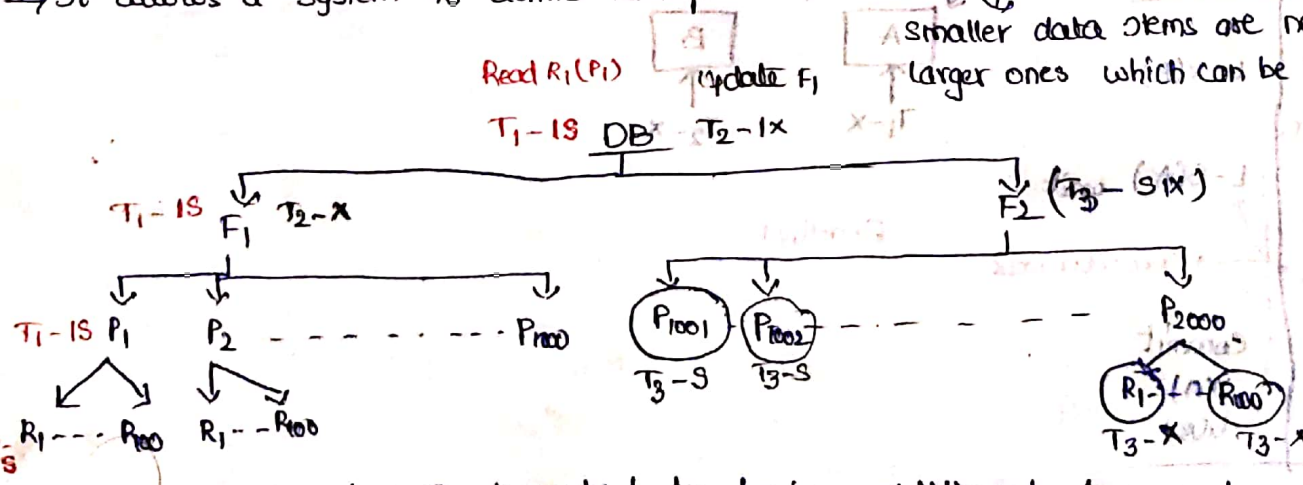




② Multiple Granularity locking protocols :

→ It allows a system to define multiple levels of data granularity

smaller data items are nested within larger ones which can be rep with tree.



→ This protocol can be implemented by having additional lock modes called intension lock modes.

Intension lock mode.

Intension lock mode :

1) Intension shared mode (Is) :

An Explicit locking is being done at lower level of tree but with only shared lock mode

2) Intension Exclusive mode (Ix) :

An Explict locking is being done at lower level with exclusive lock mode

3) shared and intension exclusive mode (six) :

The subtree rooted by the node is locked Explicitly in shared mode and that Explicit locking is done at lower levels with Exclusive mode

Points to remember :

→ locking being done in top to bottom manner
→ unlocking being done in bottom to top manner
→ since locking and unlocking being done in 2 phases ⟶ multiple Granularity 2PL

③ Time stamp ordering protocol : (Top)

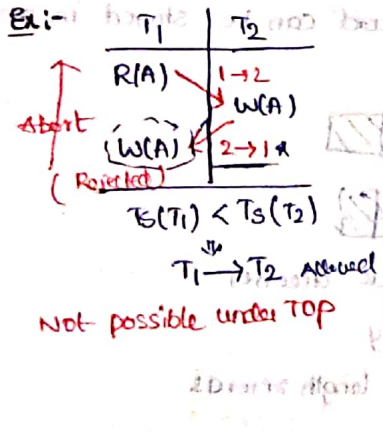
(It is a **deadlock free protocol**)

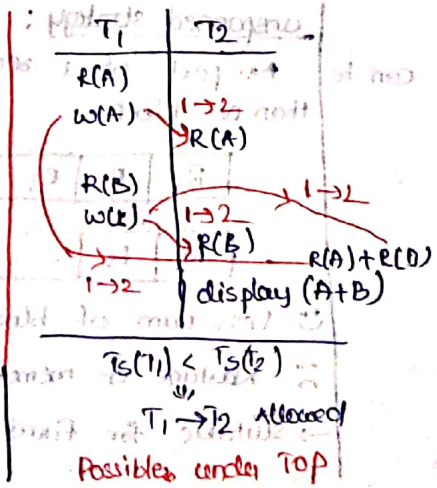
(The order of serializability is determined in the **order of their Timestamps**)

→ With each T_x we associate a fixed unique Timestamp ie $Ts(T_i)$
→ If T_j entered into a system after T_i then $Ts(T_i) < Ts(T_j)$
→ If $Ts(T_i) < Ts(T_j)$ then the resultant schedule must be equivalent to serial schedule $T_i \to T_j$

*) Time stamp ordering protocol requires that all the conflicting read and write operns must be Executed in the order of their Timestamps.
*) If any T_x is not Executed its Operation in the order of its Timestamp such Operation is rejected and the T_x is aborted.
*) The aborted T_x is restarted in a system with **new Timestamp** with the problem of starvation

Ex:-





☺ Ensure (conflict) serializability.
Deadlock free

☹ starvation may occur due to
Continously abort and restart of T_x.

To minimize
↓
sol
Thomas write rule (TWR)

Thomas write rule (TWR) :

Modification of TOP that minimizes the prob of starvation by **Ignoring obsolete write operations** ie **Useless write opers.**

Ex:-

T_1	T_2
R(A)	
	W(A) 1→2
W(A)	2→1

obsolete. → So Constrained conte

Not possible under Top but possible under TWR.

Ex:-

T_1	T_2
W(A)	
	W(A) 1→2
W(A)	2→1

Abort Reject

Not Constrained conte.

Not possible under Top and TWR.

Points to remember:

$T_i \to T_j$	Allowed	Not Allowed
Top	$R_1(A)$ $W_2(A)$ $W_1(A)$ $W_2(A)$ $W_1(A)$ $R_2(A)$	$W_2(A)$ $R_1(A)$ $W_2(A)$ $W_1(A)$ $R_2(A)$ $W_1(A)$
TWR	$W_2(A)$ $W_1(A)$ → If obsolete	→ Constrained conte.

File organization:

Database → Files → Records → data; Blocks

Blocking factor: $B_f = \frac{B}{R}$

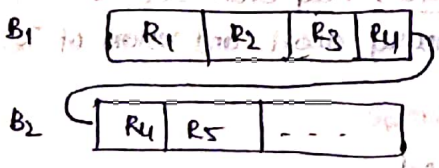
Avg num of records that can be stored in a block

Ex:- B = 100B, R = 30B

$B_f = \frac{100}{30}$ → 3.33 rec/block (spanned); 3 rec/block (unspanned)

Spanned strategy

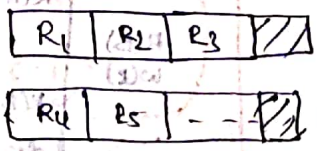
It allows partial part of a record can be stored in a block.



☺ No wastage of memory
☹ More num of block acenes
→ suitable for variable length records

unspanned strategy:

No part of a record can be stored in more than one block.



☺ Less num of block accesses
☹ Wastage of memory
→ suitable for fixed length records

Organization of records in a file:

Unordered file organization

Records are stored at the end of the file

Searching: Linear search

☺ Insertion is easy
☹ Searching is inefficient.

Ordered file organization

→ records are stored in the order of some key value

Searching: Binary search

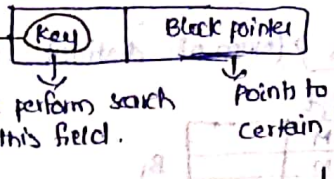
☺ Searching is efficient
☹ Insertion was difficult

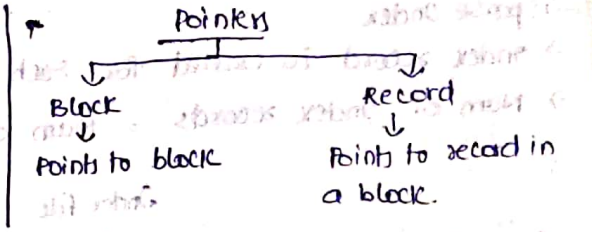
Ordered file organization (uses Indexes)

Index: It is used to speed up the retrieval of records in response to certain search conds

↳ It is an ordered file with two fields.

PI – On Primary key
CI – On Non key
SI – On Candidate key / Non key

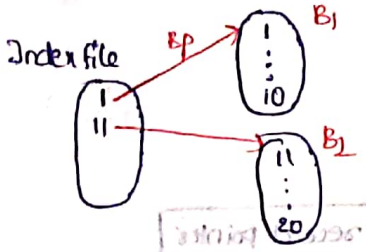




Classification of Index:

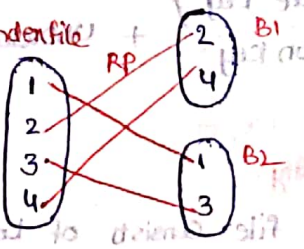
① Sparse Index:

→ It has Index Entry for some search key value
→ An Index record is created for each block in data file
→ Num of Index records = Num of blocks in the datafile
→ uses block pointers



② Dense Index:

→ It has Index Entry for every search key value
→ An Index record is created for each record in the data file
→ Num of Index records = Num of records in data file.
→ uses record pointers
→ provides logical ordering of data file.



Another classification:

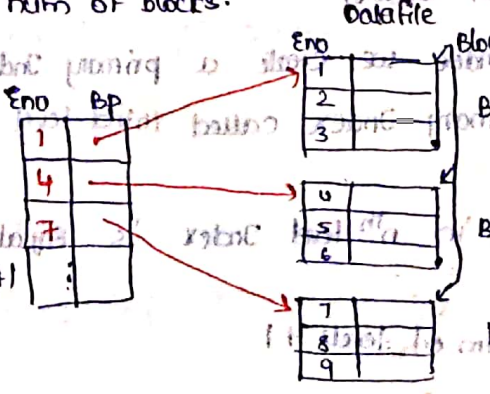
- **Index**
  - **Single level Index**
    - **Primary Index** → File is ordered on PK (PK + ordered)
    - **Clustered Index** → File is ordered on non key (NK + ordered)
    - **Secondary Index** → File is unordered on NK/CK (NK/CK + unordered)
  - **Multilevel Index**
    - **B trees**
    - **B+ trees**

Primary Index:

→ It is created on primary key of the data file, & the file is physically ordered on it
→ It is an ordered file with two fields

Primary key	Block pointer

→ Index record is created for the first record of each block called Block anchor
→ Sparse Index
→ num of Index records = num of blocks.



- Binary search
- B_i – Index file
- Avg num of block acces = $\lceil \log_2 B_i \rceil + 1$

- Binary search
- B – data blocks
- Avg num of block acces = $\log_2 B$

Clustered Index:

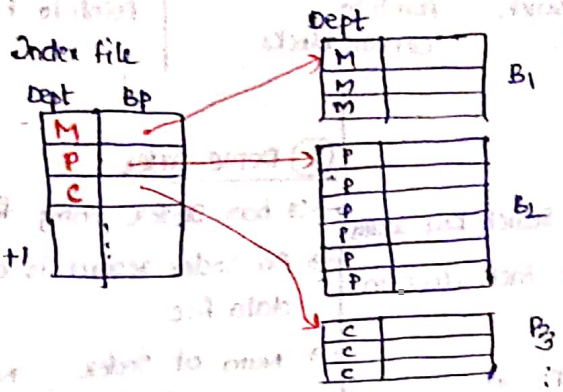
→ The index file is ordered on a non-key with two fields

Clustering field (Non key field)	Block pointer

→ Sparse Index

→ Index record is created for each cluster

→ Num of index records = Num of clusters → (Num of distinct non key values.



- Binary search
- B_i - Index blocks
- Avg num of block access = $\lceil \log_2 B_i \rceil + 1$

- Binary search
- B - data blocks
- Avg num of block access = $\log_2 B$

Secondary Index:

→ It provides secondary means of accessing a file on which some primary acces already exists

→ Candidate key / Non key + Unordered → secondary index.

Eliminated bcoz of infeasibility.

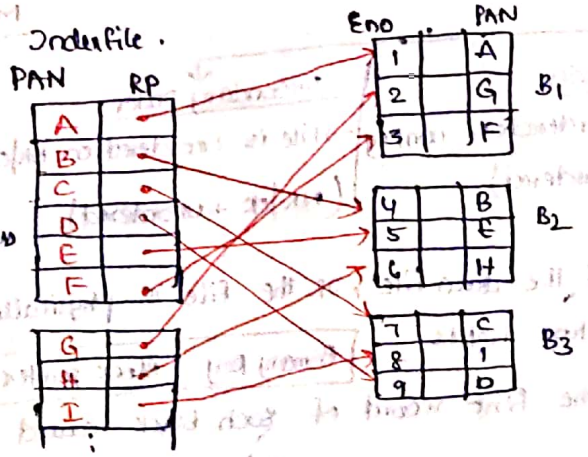
→ The index file consists of two fields

candidate key	record pointer

→ Dense Index

→ Index record is created for each record in data file

→ Num of index records = Num of records in data file.



- Binary search
- B_i index files.
- Avg num of Block acces
  - $\lceil \log_2 B_i \rceil + 1$ /
  - $\lceil \log_2 B \rceil + 1$

sort PAN
- Linear search
- B data blocks
- Avg num of Block acces = $\frac{B}{2}$

Multilevel Index: (Index over an Index)

→ on any of the first level index we create a primary index called secondary level index on which we create a primary index called third level index and so on until it reaches to one block.

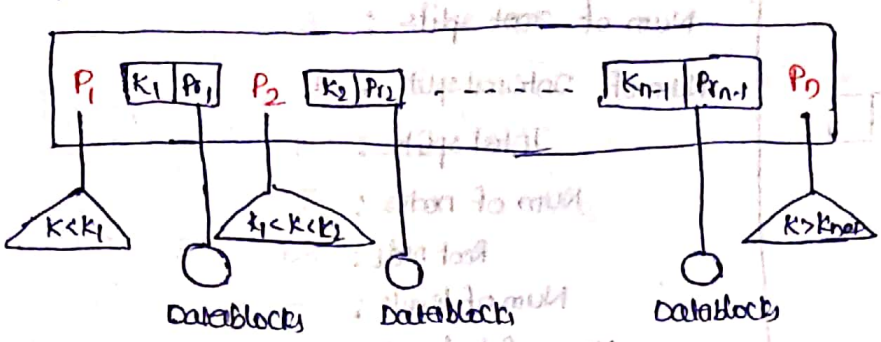
→ Num of index records in n^{th} level index is equal to the num of blocks in $n-1^{th}$ level index

→ Num of Block Acceses = Num of levels + 1

B-Tree : (Height balanced search tree)

Structure of BTree node :

→ It provide direct access



$K_1 < K_2 < \cdots < K_{n-1}$ → Keys

$Pr_1, Pr_2 \cdots Pr_{n-1}$ → Record | data ptrs.

$P_1, P_2 \cdots P_n$ → Tree | Block | Index ptr.

*) n tree ptrs, n-1 rec ptrs + keys } → B-Tree

Order of B-Tree :

It is the max num of tree ptrs that can be stored in a Block

order of BTree ⇒ $n*p + (n-1)(Pr+K) \leq B$

*) levels - m

order - n

Num of records = $n^{m+1} - 1$

Ex:- B Tree order = 24, 69% full, num of Index records in 3 level BTree.

m = 24

69% of m = 24 * 0.69 = 16

Root level : 1 node, 16 tree ptrs, 15 Index records

level 1 : 16 nodes, 16*16 tree ptrs, 16*15 rec ⇒ 240

level 2 : 256 nodes, 256*16 tree ptrs, 16*16*15 rec ⇒ 3840

level 3 : 4096 nodes, 4096*16 tree ptrs, 16*16*16*15 rec ⇒ 61440

↓ 65,536 tree ptrs

65,535 rec.

B-Tree Insertion :

→ A new element is always inserted at the leaf node.

→ If the node is full we split the node to two nodes by moving middle element to one level above and all the keys less than middle element goes to left subtree and that are greater than middle goes to right subtree

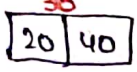
Points to remember :

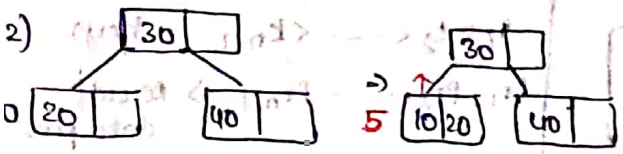
If a Btree of order : n

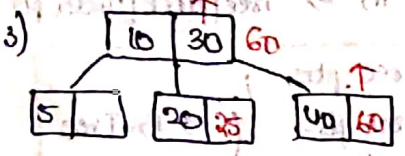
→ Max : n pointers, n-1 keys

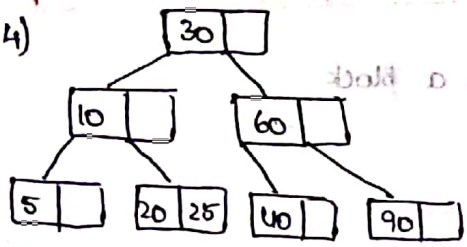
→ min : $\lceil \frac{n}{2} \rceil$ pointers, $\lceil \frac{n}{2} \rceil - 1$ keys except for root node.

Construct a B-Tree of order 3 with keys: 20, 40, 30, 10, 5, 25, 60, 90

1) 

2) 

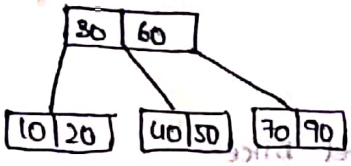
3) 

4) 

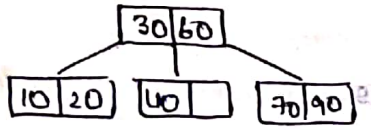
Root
L1
L2

Num of leaf splits : 3
Num of root splits : 2
Num of Internal splits : 1
Total splits : 4
Num of nodes : 7
Root node : 30
Num of levels : 2
Num of leaf nodes : 4
Num of full nodes : 1

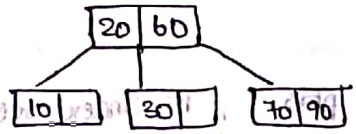
B-tree deletion



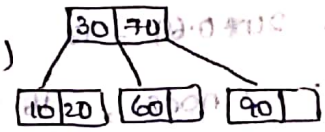
(i) Delete 50



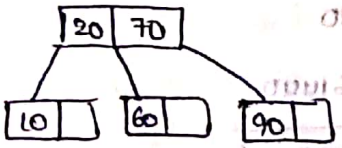
(ii) Delete 40



(or)

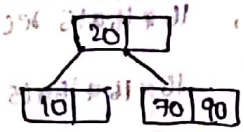


(iii) Delete 30 :

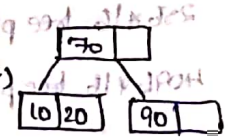


(or)

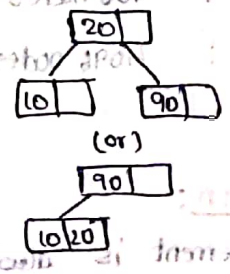
(iv) Delete 60



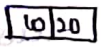
(or)



Delete 70



(vi) Delete 90



B+ tree :

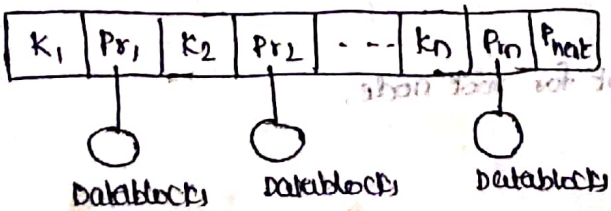
It is a modification to B-Tree data structure | *) It provides ordered access

Modifications :

① Leaf node : Remove Tree pointers, so contains only (Key + Record ptrs) $n-1$ $n-1$

② Internal node : Remove Record ptrs, so contains only (Tree ptrs + keys) n $n-1$

B+ tree structure of leaf node :



Pnext → Block | order | tree ptr.

*) Keys → n

Record ptrs → n

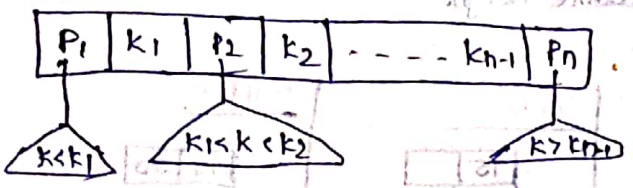
Pnext → 1

Order of leaf node :

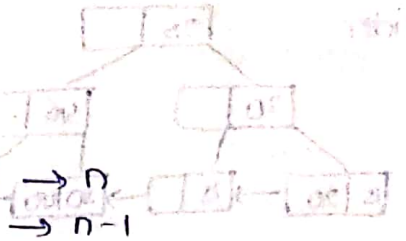
let a B^+ tree of order n

order of leaf node → $n*(P_r+k)+P_{next} \le B$

B^+ tree structure of internal node :



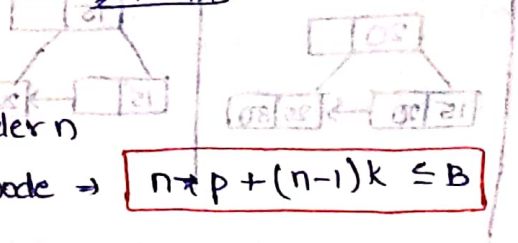
*) Tree ptrs → n
keys → n-1



Order of internal node :

Let a B^+ tree of order n

Order of internal node → $n*p+(n-1)k \le B$



Note : n_i → order of internal node

n_L → order of leaf nodes.

Number of index records = $(n_i)^{level} * n_L$

Ex : $n_i = 34$ $n_L = 31$, 69% full, num of index records = level = ?

$n_i = 34 * 0.69 = 23$

$n_L = 31 * 0.69 = 21$

Root level : 1 node, 23 ptrs, 22 keys → 22

level 1 : 23 nodes, 23*23 ptrs, 23*22 keys ⇒ 506

level 2 : 23*23 nodes, 23*23*23 ptrs, 23*23*22 keys ⇒ 11,638

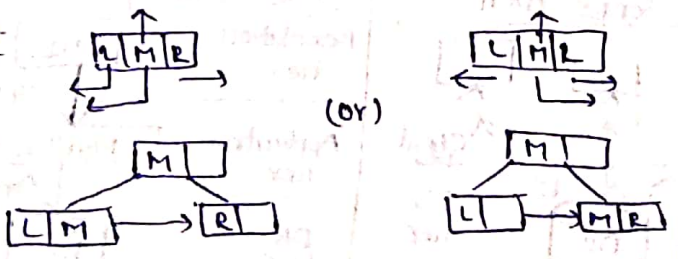
level 3 : 23*23*23 nodes, 0 ptrs, 23*23*22 keys → 11,638 keys

23*23*23*21 rec → $(23)^3 * 21$

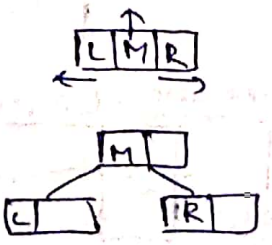
B^+ tree Insertion :

Property : ≤ (or) < ≥

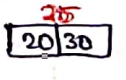
Leaf split

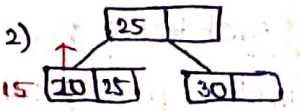


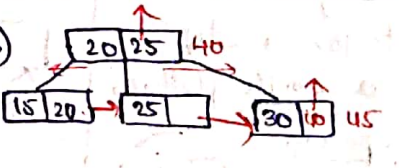
Internal node split :

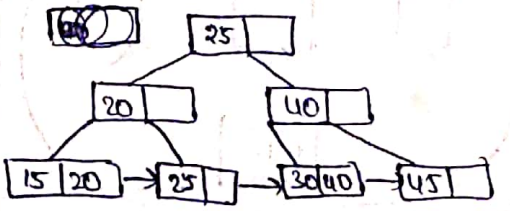


Ex:- Insert 20, 30, 25, 15, 40, 45 in B^+ tree

1) 

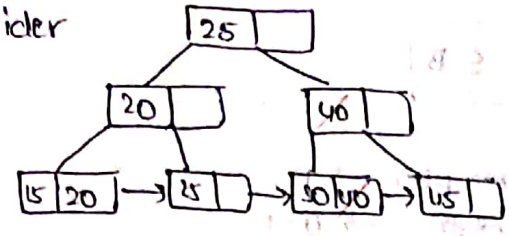
2) 

3) 

4) 

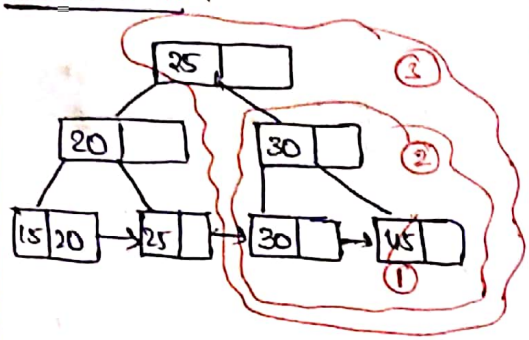
B+ tree deletion :

Consider

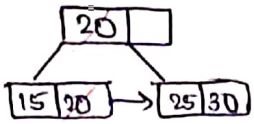


Steps to follow
1. Property
2. min order to be satisfied
3. Balance height

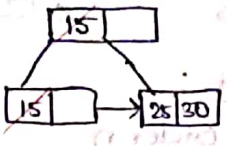
(1) Delete 40 :



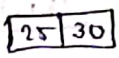
Delete 45



Delete 20



Delete 15



× — The End — ×

Min num of keys in B+ tree = $\frac{n}{2} - 1$

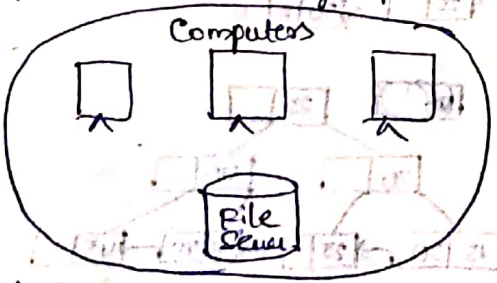
- subquery is more Efficient than join

Database Architecture :

- A DBMS is not always directly available for users. It may be centralized, decentralized, hierachical.
- DB Architecture mainly focuses on the design, development, implementation, maintanance of computer programs that store & organize information.
- Design of DBMS depends on its Architecture

1-tier Architecture :

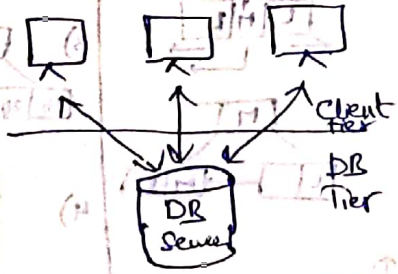
- It involves putting all of the need components for a sw application on a single platform



- It includes Interface, Middleware, Backend data in one place

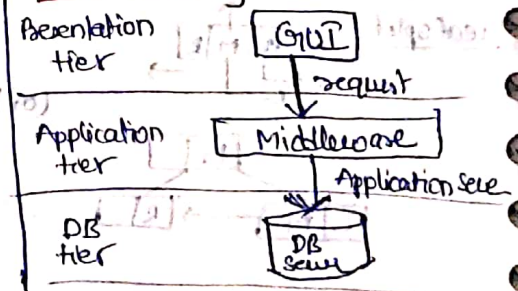
2-tier Architecture :

- client server Arch



3-tier Architecture :

- most widely used Arch



(4) N-tier Architecture : Logic tier (MVC framework)