DBMS

* Syllabus :

⇒ Functional dependency and Normalization.
⇒ Transaction and concurrency control
⇒ Relational Algebra, TRC and SQL
⇒ File organization and indexing
⇒ ER Model and Integrity Constraints.

* Data → Raw material / facts.

Database → collection of logically related Data

* RDBMS

↳ Relation → Table → Row → Tuple / Record
↳ Column → field / Attribute

1] Arity, Degree → No. of attributes
2] Cardinality → No. of Tuples
3] Relational Schema → Table Heading
[Student (sid, Name, Marks, Branch)]

4] Relation Instance : set of records at particular moment.

5] Degree of relation → No. of Attributes

6] Domain → Values allowed for the Attribute

* Functional Dependency (FD)

$x \rightarrow y$ — Relation schema R in which x & y be the attribute set of R, t_1 & t_2 Any two tuple such that

(x: Determinant, y: Dependent)

If $t_1.x = t_2.x$ then $t_1.y = t_2.y$ must be same

Note: In $x \rightarrow y$, whenever x value repeat, corresponding y value must be same.

Eg:

x	y
1	5
2	6
3	7
4	8
2	5
4	9

✗

x	y
1	5
2	6
3	5
4	6
5	7

* Types of FD

1. Trivial FD
2. Non Trivial FD
3. Semi Non Trivial FD → Not in Syllabus

1] Trivial FD [Always Valid]

$x \rightarrow y$ is a Trivial FD

if $x \supseteq y$

↳ RHS Attribute equal or part of LHS Attribute.

Eg: $AB \rightarrow A$
$AB \rightarrow B$
$AB \rightarrow AB$

2] Non Trivial FD

$x \rightarrow y$ is Non Trivial

if $x \cap y = \phi$ && must satisfy FD Definition

Eg: $A \rightarrow B$
$A \rightarrow C$

3] semi Non Trivial FD

~~$x \cap y \neq \phi$~~ $x \cap y \neq \phi$
~~$x \not\supseteq y$~~ $x \not\supseteq y$

Eg: $AB \rightarrow BC$

How to make cases.

R(A B C)

A	B	C

Non-Trivial FD case

$A \to B$ $B \to A$ $C \to A$

$A \to C$ $B \to C$ $C \to B$

$A \to BC$ $B \to AC$ $C \to AB$

$AB \to C$

$BC \to A$

$AC \to B$

Q1) ~~Non Trivial~~ Which Non-Trivial FD satisfied by the ~~Following~~ Instance.

A	B	C
2	2	4
2	3	4
3	2	4
3	3	4
3	2	4

✗ $A \to B$ ✗ $B \to A$ ✗ $C \to A$ ✓ $AB \to C$

✓ $A \to C$ ✓ $B \to C$ ✗ $C \to B$ ✗ $BC \to A$

✗ $A \to BC$ ✗ $B \to AC$ ✗ $C \to AB$ ✗ $AC \to B$

Ans :- $A \to C$

$B \to C$

$AB \to C$

* Attribute closure $[x]^+$:

⇒ R be the Relational Schema X be the attribute set of R.
The set of all possible Attributes determined from Attribute x is called Attribute closure of x $(x)^+$.

Eg: $R(A, B, C, D, E)$ $[A \to B, B \to C, C \to D, D \to E]$

$[A]^+ = [ABCDE]$
$[B]^+ = [BCDE]$
$[C]^+ = [CDE]$
$[D]^+ = [DE]$
$[E]^+ = [E]$

$[BE]^+ = [BECD]$
$[CD]^+ = [CDE]$

* Keys concept

Key - Set of Attribute which uniquely determine each tuple in the relation.

Super key - If Set of all Attribute determined by the Attribute closure of x $(x)^+$ then x is a super key.

Eg: $R(A, B, C, D, E)$ $[A \to B, B \to C, C \to D, D \to E]$

$[A]^+ = [ABCDE]$

'A' is the super key
Every key is a super key
Any superset of super key is also super key

* Candidate key :

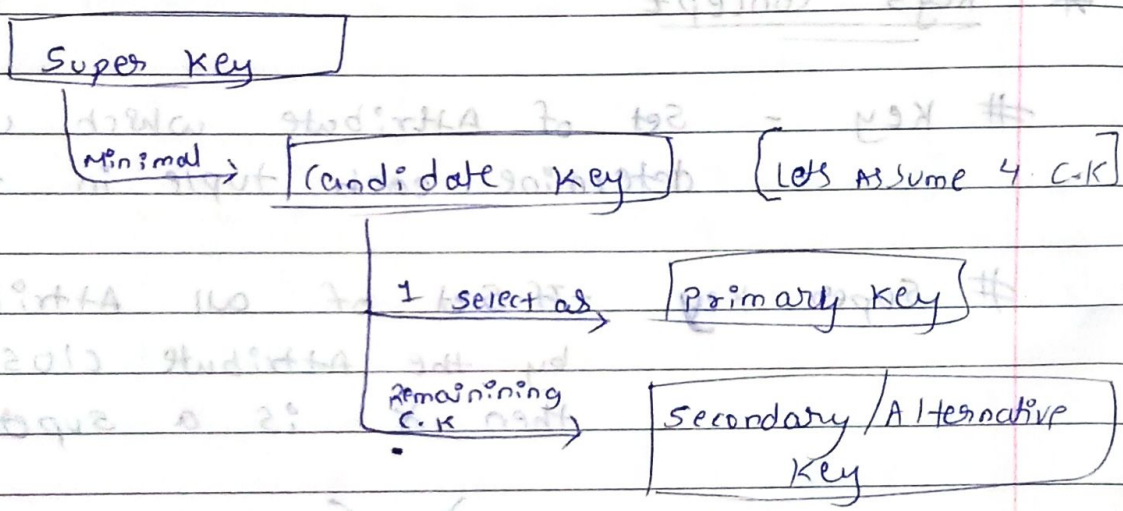
⇒ Minimal of super key

⇒ If any proper subset of super key is also super key then that proper subset is called candidate key.

Eg: R(ABCDE) [AB→C, C→D, B→EA]

⇒ AB is super key
⇒ B is candidate key

* Keys concept



* Prime Attribute / Key Attribute

⇒ Attribute that belongs (present) in any candidate key

* Non prime / Non key Attribute

⇒ Attribute that Not belongs (Not present) in any candidate key

Eg: ~~$[AB]^+ = A$~~

1] $R[ABCDE] \quad [AB \to C \;,\; C \to D \;,\; B \to EA]$

Prime / key Attribute : [B]

Non prime / Non key attribute : [ACDE]

* Finding multiple candidate keys

⇒ First find any one candidate key then that Attribute (present in C.K) is prime / key Attribute.

If $X_{Attribute} \longrightarrow$ [Prime / key Attribute]

then multiple candidate keys are there

⇒ Let Assume D is candidate key

X Attribute → [Prime Attribute] {Prime / key Attribute = [D]

C is
CK
$\begin{cases} C \rightarrow D \\ C \rightarrow DE \end{cases}$

(B) (C) → $BC \rightarrow D$
$BC \rightarrow DE$

Q] R[ABCDE] {AB → C, C → D, D → E, B → A, C → B}
find candidate keys for the relation R ?

⇒ $[AB]^+ = [ABCDE]$
AB is Super key

$[A]^+ = [A]$
$[B]^+ = [BACDE]$
B is candidate key — (1)

Prime / key Attribute = [B, C]

If X Attribute → [Prime Attribute]

C → B
$[C]^+ = [CBADE]$
C is candidate key — (2)

~~DBMS~~

$AB \rightarrow C$ → Process is recursive

Ans = 2 C.K [B, C]

* Armstrong's Axioms / Inference Rules

1] Reflexivity rule :

⇒ $\alpha \rightarrow \beta$ is Trivial (Reflexive) iff $\alpha \supseteq \beta$

2] Augmentation Rule :

⇒ $\alpha \rightarrow \beta$ ⇒ $\alpha\gamma \rightarrow \beta\gamma$

3] Transitive Rule :

⇒ $\alpha \rightarrow \beta$ & $\beta \rightarrow \gamma$ ⇒ $\alpha \rightarrow \gamma$

Additional rules :

Union
⇒ $\alpha \rightarrow \beta$ & $\alpha \rightarrow \gamma$ then $\alpha \rightarrow \beta\gamma$

Decomposition
⇒ $\alpha \rightarrow \beta\gamma$ then $\alpha \rightarrow \beta$ & $\alpha \rightarrow \gamma$

Pseudo transitivity
⇒ $\alpha \rightarrow \beta$ & $\gamma\beta \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$

* Membership set :

⇒ Let F be the given FD. Any $X \to y$ FD is a member of FD set F, iff $x \to y$ logically implied in F.

$x \to y$ logically implied means from the closure of x determine y.

$$[x]^+ = [\ldots y]$$

Q] F : $[A \to B \quad , \quad B \to C]$

check $A \to C$ Member / valid FD / implied or NOT?

⇒

$$[A]^+ = [ABC]$$

$A \to C$ is member of FD set F.

* Equality btween 2 FD set.

⇒ Let there are 2 FD set (F & G).

F : [_____] G : [_____]

F & G are equals only if,

iff ↳ F covers G : True
G covers F : True

F cover G :	True	True	False	False
G cover F :	True	False	True	False
	F ≡ G	F ⊃ G	F ⊂ G	unlom-parable

Page No.
Date: / /

Q) $F : [AB \to CD \quad , B \to C \quad , C \to D]$

$G : [AB \to C \quad , AB \to D \quad , C \to D]$

⇒ **F covers G :**

Check if FD's of G implies on F.

$AB \to C$ to prove $[AB]^+ = [ABCD]$

$AB \to D$ $[AB]^+ = [ABCD]$

$C \to D$ $[C]^+ = [CD]$

True

G covers F ,

$AB \to CD$ i.e. $[AB]^+ = [ABCD]$

$B \to C$ (Given) $[B]^+ = [BC]$

$C \to D$ (Given) $[C]^+ = [CD]$

False

$\therefore$ **F ⊃ G**

* Minimal cover

⇒ objective of the minimal is eliminate / Reduce the redundant FD [Extra FD]

⇒ Redundant FD (RFD) is a FD, if we Delete that FD from the original FD set, then after Deletion does not effect the power of FD set.

Procedure

1] Split the FD such that RHS contain single attribute

$A \to BC \Rightarrow A \to B, A \to C$

2] Find the Redundant Attribute on LHS & Delete them

$AB \to C$; A is extra if $[B]^+$ contains A
B is extra if $[A]^+$ contains B

Keath { A is extra if $B \to C$; B is extra if $A \to C$ } $AB \to C$

3) Find the redundant FD & delete them.

Eg] $\{A \to C, AC \to D, E \to AD, E \to H\}$

Step 1 : $A \to C$, $AC \to D$, $E \to A$, $E \to D$, $E \to H$

Step 2 : $AC \to D$ } $[A]^+ = [A\ C\ D]$
$[E]^+ = [E\ A\ C\ D\ H]$

C is extra

Step 3 : ① $A \to C$, ② $A \to D$, ③ $E \to A$, ④ $E \to D$ ⑤ $E \to H$

⇒ Hide the FD working on and take closure with [A] other FD if [A] is in closure than FD is extra.

$A^+ = (AD)$ $(A)^+ = (AC)$ $(E)^+ = (EDH)$ $E^+ = EAHCD$ $(E)^+ = EAHD$

Minimal cover : $A \to C$, $A \to D$, $E \to A$, $E \to H$

or

$A \to CD$, $E \to AH$

NOTE : Minimal cover may or may not be unique

* Properties of Decomposition

1. Lossless Join Decomposition
2. ~~Decom~~ Dependency preserving Decomposition.

Lossless Join Decomposition

1. Basic concept
2. Binary method
3. Chase Test

* Lossless Join Decomposition :

If $R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n \equiv R$

Lossless Join Decomposition

If $R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n \supset R$

Lossy Join Decomposition

* Natural Join ($\bowtie$) [$R \bullet\!\bowtie S$]

Step 1 : Cross Product of R and S

R	S
n_1 Tuple	n_2 Tuple
C_1 Attribute	C_2 Attribute

$R \times S$ = $n_1 \times n_2$ Tuple

$C_1 + C_2$ Attribute

Step 2 : select the tuples which satisfy equality condition on all common attribute.

Step 3 : projection of distinct Attribute.

Q] R(A B C)

A	B	C
1	5	6
2	5	8
3	8	8

⇒

Step 1 : $R_1 \times R_2$

R_1 → 3 tuples, 2 Attributes

R_2 → 3 tuples, 2 Attributes

Step 2 :

$R_1.A$	$R_1.B$	$R_2.B$	$R_2.C$
1	5	5	5
1	5	8	8
2	5	5	5
2	5	8	8
3	8	5	5
3	8	8	8

Step 3 :

A	B	C
1	5	5
1	5	8
2	5	5
2	5	8
3	8	8

lossy join

[∵ $R_1 \bowtie R_2 \supset R$]

* $R_1 \bowtie R_2$ is a lossy Join :

i) $R_1 \cup R_2 \neq R$

ii) If common attribute of R_1 & R_2 Neither a super key of R_1 Nor R_2

$[R_1 \cap R_2]^+ \not\rightarrow R_1$

$[R_1 \cap R_2]^+ \not\rightarrow R_2$

* Chase Test :

⇒ In chase Test we create a matrix in which column represent the attribute & Tuple represent the subrelations.

- Fill all the cells / value with any variable (a) in corresponding Attribute of respective sub relation.
- Now Fill the Table Entries with the help of Given FD's
- If we get any one Tuple with all the entries 'a' then Lossless Join

* Dependency preserving Decomposition :

⇒ Let R be the relational schema with FD set F is decomposed into sub relations $R_1 R_2 R_3 \ldots R_n$ with FD set $F_1 F_2 \ldots F_n$ respectively

If $F_1 \cup F_2 \cup F_3 \ldots \cup F_n \equiv F$ } dependency preserving decomposition

If $F_1 \cup F_2 \cup F_3 \ldots \cup F_n \subset F$ } dependency not preserved

Eg) Let $R(A, B, C, D, E)$ be a relational schema with FD's $A \to B$, $B \to C$, $C \to D$ and $D \to BE$ decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(CD)$, $R_4(DE)$

$(A)^+ = [ABCDE]$
$(B)^+ = [BCDE]$
$(C)^+ = [CDBE]$
$(D)^+ = [DBEC]$
$(E)^+ = (E)$

$R_1(AB)$	$R_2(BC)$	$R_3(CD)$	$R_4(DE)$
$A \to B$	$B \to C$	$C \to D$	$D \to E$
	$C \to B$	$D \to C$	

$A \to B$ $\quad$ $B \to C$ $\quad$ $C \to B$ $\quad$ $C \to D$ $\quad$ $D \to C$ $\quad$ $D \to E$

⇒ $A \to B$, $B \to C$, $C \to D$, $D \to E$ $\Longleftrightarrow$ $D \to C$, $C \to B$

$D \to B$

⇒ $A \to B$, $B \to C$, $C \to D$, $D \to BE$ → dependency preserved

* Closure of FD set $[F]^+$: set of all possible FD's which is determined by given FD set is called closure of FD set.

case 1 : when FD set is NOT given

Eg] R(AB) then find $[F]^+$

No. of component				
ϕ	$\phi \to \phi$	$A \to \phi$	$B \to \phi$	$AB \to \phi$
A	$\phi \to A$	$A \to A$	$B \to A$	$AB \to A$
B	$\phi \to B$	$A \to B$	$B \to B$	$AB \to B$
AB	$\phi \to AB$	$A \to AB$	$B \to AB$	$AB \to AB$

$n = 4$

$[F]^+ = 13$

$[F]^+ = n(n+1) + 1$

case 2 : when FD set is given

Eg] R(AB) $[A \to B]$

$\Rightarrow$ 2^n

ϕ 0 Attribute $2^0 = 1$

A 1 attribute $[A]^+ = [AB] = 2^2 = 4$

B 1 attribute $[B]^+ = [B] = 2^1 = 2$

AB 2 attribute $[AB]^+ = [AB] = 2^2 = 4$

$\Rightarrow$ //

* Normal forms

⇒ Normal forms is a set of rule, used to reduce/eliminate the redundancy.

↳ unnecessary repetition of data

⇒ Normal forms

- 1NF
- 2NF
- 3NF
- BCNF

⇒ Every higher Normal form contain the lower Normal form.

First Normal form [1NF]

⇒ A Relation R is in 1NF iff R does not contain Any multivalued Attribute

(OR)

All attribute of R are atomic.

Eg:

Roll	Name	course
1	vinod	C/Java

↓ Multivalued

Not in 1NF

→

Roll	Name	course
1	vinod	C
1	vinod	Java

R is in 1NF

NOTE :- In INF Redundancy level is too high

Redundancy level : INF > 2NF > 3NF > BCNF

:- Default RDBMS is in 1NF.

* Case 1 :-

Proper subset of candidate key → Non prime / Non key Attribute

Eliminated by 2NF

Case 2 :-

Non key Attribute → Non key Attribute

Eliminated by 3NF.

Case 3 :-

Proper Subset of one CK → Proper Subset of Another CK

Eliminated by BCNF

1] Case 1 : Proper subset of CK ⟶ Non key / Non prime Attribute

Eg: R(A B C D E F) [AB → C, C → DF, B → E]

CK = [A B]

Non prime / Non key Attribute = [C, D, E, F]

B → C

B: Proper subset of CK

C: Non key attribute

} Not in 2NF.

2] Case 2 : Non key Attribute ⟶ Non key Attribute

Eg: R(A B C) [A → B, B → C]

CK = [A]

Non key Attribute = [B C]

B → C } Not in 3NF

3] Case 3 : Proper subset of one CK ⟶ Proper subset of another CK

Eg: R(A B C D) ⇒ [AB → CD, D → A]

CK = [AB, DB]

Non key Attribute = [C]

D → A } Not in BCNF

* Partial Dependency :

⇒ $X \to Y$ is partial FD

If $\exists A \in X$,

$(X - A) \longrightarrow Y$

Eg: $AB \to C$ is partial FD

if $A \to C$

(or) $B \to C$

Second Normal form :

⇒ A relational schema R is in 2NF if every non prime attribute A in R is fully functional dependent on the primary key

$AB \to C$ is fully functionally Dependent if

$A \not\to C$

$B \not\to C$

Third Normal form

⇒ A Relational Schema R is in 3NF if every $X \to Y$ non trivial FD must satisfy the following condition

$X \to Y$

X : super key

(or)

Y : Prime key attribute

BCNF

⇒ A Relational schema R is in BCNF if every $x \to y$ Non Trivial FD must satisfy the following condition

$x \to y$
x : super key

* Important points :

⇒ If a Relation R has only one candidate key then R always in 1NF But may/may not in 2NF / 3NF / BCNF

⇒ If in a relation R, all candidate keys are simple [single Attribute] candidate key then R always is in 2NF But may or may not in 3NF / BCNF

⇒ If in a relation R, all attributes are key / Prime attribute then R always is in 3NF but may or may not be in BCNF

⇒ If a Relation R is in 3NF & all candidate keys are simple candidate key then R always is in BCNF

⇒ Binary Relation (Relation with 2 Attribute) is always is in BCNF.

⇒ A Relation R with no Non Trivial FD is always is in BCNF

OR

Trivial FD is in BCNF

$xyz \rightarrow xy$

↓

Super key

* Design Goal	1NF	2NF	3NF	BCNF
0% redundancy	X	X	X	~~✓~~
Lossless Join	✓	✓	✓	~~✓~~
Dependency Preserving	✓	✓	✓	~~✓~~ may or may not

* 2 NF Decomposition

R(ABCDEFGH)

{ AB → C , C → D , B → E , E → FG , G → H }

⇒ candidate Key = [AB]

Non Key Attribute = [C D E F G H]

[B → E] } Not in 2NF

$[B]^+ = [B\ E\ F\ G\ H]$ R(AB C D E F G H)

R1 (A B C D) R2 (B E F G H) } 2NF + Lossless join + Dep. Preserved

Eg 2 : R(ABCDE) F : [A → B , B → E , C → D]

CK = [AC]

Non key attribute = [B D E]

A → B
C → D } NOT in 2NF

$[A^+] = [ABE]$ $[C]^+ = [CD]$ R(ABCDE)

R1 (AC) R2 (ABE) R3 (CD)

CK=AC CK=A CK=C

* 3NF Decomposition

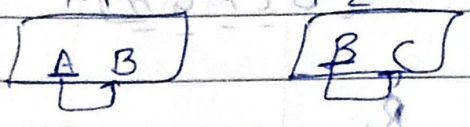
1] R(ABC) [A → B , B → C]

CK = [A]

Non key = [B, C]

Not in 3NF bcoz → { B → C }

3NF Decomposition :

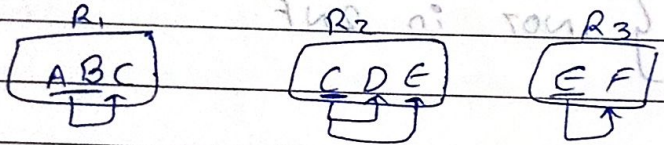


2] R(ABCDEF) [AB → C , C → D , C → E , E → F]

CK = [AB] , Non key = [C, D, EF]

C → D
C → E
E → F
} Not in 3NF

∴



CK = AB CK = C CK = E

ABCDEF

$C^+ = \{C, D, E, F\}$

$E^+ = \{E, F\}$

ABC EF CDE

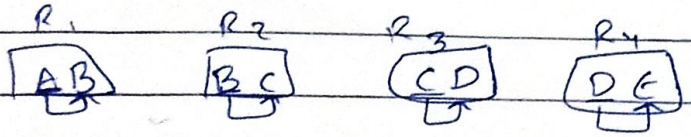
* BCNF Decomposition:

1] R(ABCDE) [A → B, B → C, C → D, D → E]

CK = [A]

Non key = [B, C, D, E]

R is in 2NF, But not in 3NF



} Now in 3NF & BCNF

There can be multiple BCNF Decomposition,
~~For example:~~ In this example

1] (ABD) (BC) (DE) ⟶ ① B → C, ② D → E, ×③ C → D — R(AB/DE)

2] (ABE) (BC) (CD) ① C → D, ② B → C, ×③ D → E — R(AB/DE)

3] (AB) (BC) (CD) (DE) ① D → E, ② C → D, ③ B → C — R(AB/DE)

In BCNF Dependency may/may not be preserved But lossless join ~~Gu~~ Guarenteed

Till 3NF lossless join & Dependency preserving must be satisfied.

* Multivalued functional dependency

$X \rightarrow\rightarrow Y$

If $t_1.X = t_2.X = t_3.X = t_4.X$ &
$t_1.Y = t_2.Y$ && $t_3.Y = t_4.Y$
&
$t_1.Z = t_3.Z$ && $t_2.Z = t_4.Z$

* Summary :

1) Database Term & RDBMS concept
2) FD concept
   FD Types <- Trivial, Non-Trivial, semi Non Trivial
   Properties of FD.
3) Attribute closure
4) Key concep — Super key, candidate key, finding multiple candidate key
5) Membership set
6) Equality b/w 2 FD set
7) Finding # of Super keys & candidate keys
8) Minimal cover
9) Properties of Decomposition
   - lossless Join <- Basic concept, Binary method, Chase Test
   - Dependency Preserving
10) Closure of FD set.
11) Normal Form — 1NF, 2NF, 3NF, BCNF

Transaction & concurrency control

⇒ A transaction is a unit of program execution that accesses and possibly updates various data items.

A	—	Atomicity	ACID [Maintain Integrity]
C	—	Consistency	
I	—	Isolation	
D	—	Durability	

1) Atomicity :

⇒ Either Execute all operation of the transaction successfully or None of them

- If transaction is failed Recovery management component are there, it rollBack the transaction & undo all modification.
- Log's [Transaction log] : log contains all the activity [modification] of the transaction.

2) Consistency :

⇒ Before & After the transaction Database must be consistent.

Eg:

Before		After
A : 4000	$A \xrightarrow{500} B$	A : 3500
B : 2000		B : 2500
6000//		6000//

3] Isolation :

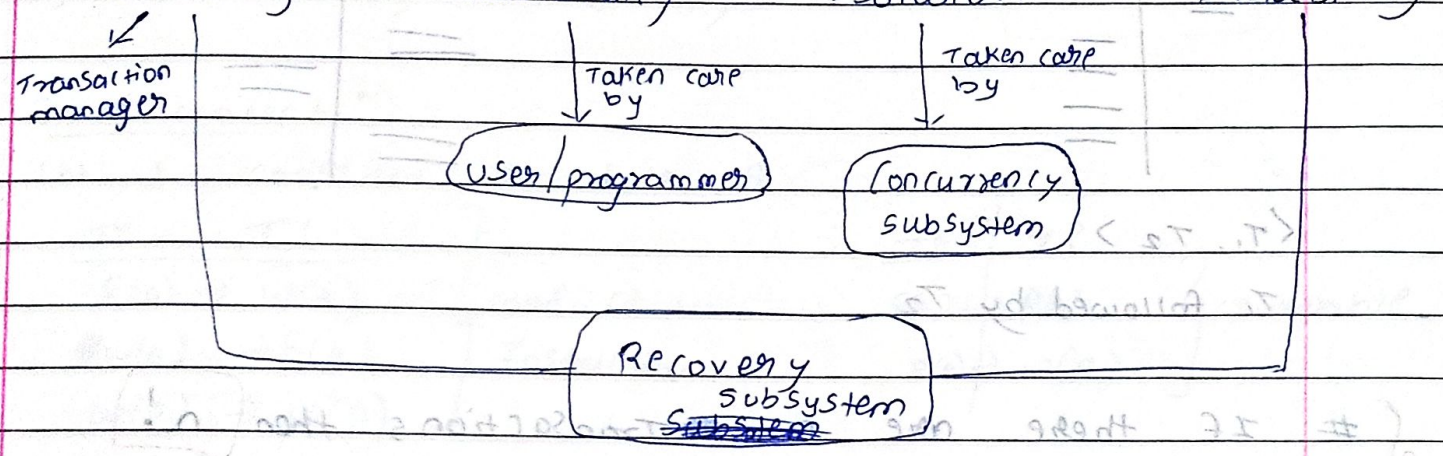
=> When two or more transaction execute concurently then isolation comes on picture

4] Durability :

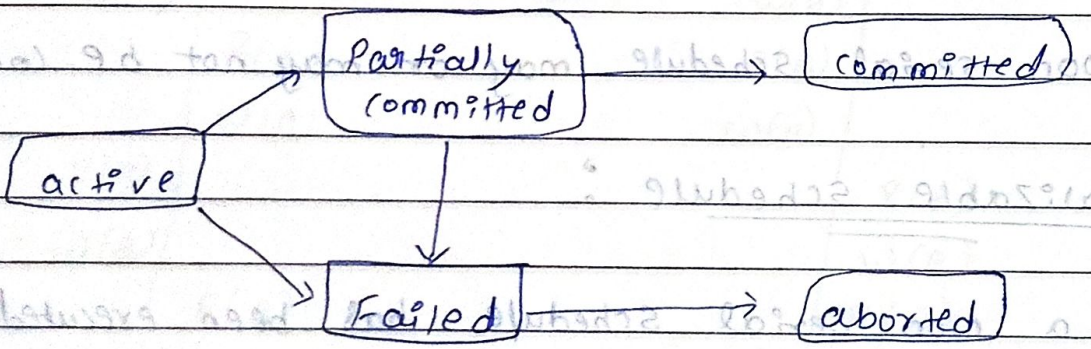
=> Any change in the Database must persist for long period of time.

- DB must be able to recover under any case of failure.

* Atomicity — Transaction manager
  Consistency — Taken care by — User/programmer
  Isolation — Taken care by — Concurrency subsystem
  Durability
  Recovery subsystem

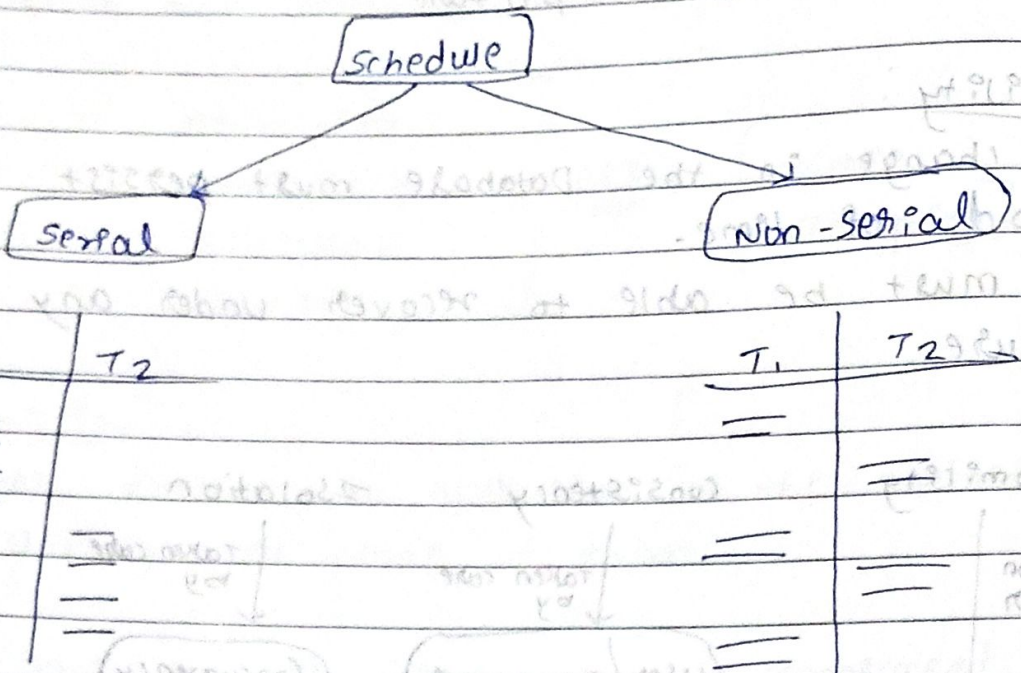


* Transaction state :



* Schedule :

⇒ Time order sequence of two or more Transactions



$\langle T_1, T_2 \rangle$

T_1 followed by T_2

Imp # If there are n Transactions then n! serial schedule.

Imp # All n! serial schedule are always consistent.

Non serial schedule may or may not be consistent.

* Serializable schedule :

- If a Non Serial schedule has been executed, that could have same effect on the database, as any serial schedule, then it is called serializable Schedule.
- The process is called serializability.

How to achieve Serializable Schedule

- Conflict Serializable
- View Serializable

* **Conflict serializable :**

1] Basic concept
* 2] Testing method [Precedence graph]
3] Conflict Equal to any serial schedule

1] **Basic concept :**

⇒ Let us consider schedule S,

T_i	T_j
R(A)	W(A)
W(A)	R(A)
W(A)	W(A)

} Conflict Instruction

T_i	T_j
R(A)	R(A)
R(A)	W(B)

} swapable

Eg :

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

⟹

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

$\langle T_1, T_2 \rangle$

2] Testing Method / ~~Exapt~~ Precedence Graph method.

G(V, E)

V : set of Transactions

E : $T_i \rightarrow T_j$ edge occurs iff any one condition hold.

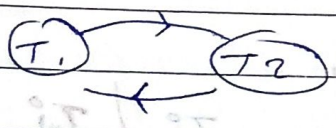
R(A) → W(A)

W(A) → R(A)

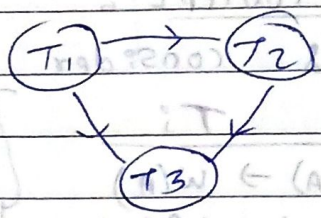
W(A) → W(A)

CNC → cycle Not conflict

Eg :



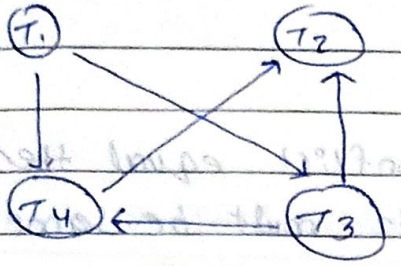
CNC → Not consistent



✓ Serializable.

* Topological Sorting

⇒ If schedule is conflict serializable [Acyclic precedence graph] then serializability order indicate that Non serial schedule is equivalent to which serial schedule.



⇒ $\langle T_1, T_3, T_4, T_2 \rangle$

* Conflict Equivalent Schedule

⇒ Two schedule are said to be conflict equivalent, if all conflicting operations in both the schedules must be executed in the same order.

Q) S_1 : $R_1(A)$ $W_1(A)$ $R_2(A)$ $W_2(A)$ $R_1(B)$ $W_1(B)$
S_2 : $R_1(A)$ $W_1(A)$ $R_2(A)$ $R_1(B)$ $W_2(A)$ $W_1(B)$

S_1:

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$
	$W_2(A)$
$R(B)$	
$W(B)$	

$R(A) - W(A) : T_1 \to T_2$
$W(A) - R(A) : T_1 \to T_2$
$W(A) - W(A) : T_1 \to T_2$

S_2:

T_1	T_2
$R(A)$	
$W(A)$	
	$R(A)$
$R(B)$	
	$W(A)$
$W(B)$	

$R(A) - W(A)$ $T_1 \to T_2$
$W(A) - R$ $T_1 \to T_2$
$W(A) - W$ $T_1 \to T_2$

S_1 is conflict Equivalent to S_2

* Conflict Serializable.

i) A schedule is said to be conflict serializable, if it is conflict Equivalent to a serial schedule.

* Imp NOTE:

i) If S_1, S_2 schedule are conflict equal then precedence graph of S_1 and S_2 must be same.

ii) If S_1 and S_2 have same precedence graph then S_1 & S_2 may or may not conflict equal.

* Serializable

⇒ Either conflict (or) view (or) Both

⇒ If conflict then by default view

⇒ If not conflict then check view if not view then not serializable.

Equivalent schedule

1) Result Equivalent → if they produce same final result for some initial value of data.

2) view Equivalent

3) conflict Equivalent

Serializable

- Conflict serializable
- View serializable

must be same for each data item:
1. Initial Read
2. Final write
3. Updated Read [write read sequence]

* View Serializability:

①

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

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

Initial read on A : T_1, B : T_1 } $S_1 \neq S_2$

②

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

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

final write: A → T_3, B → T_3 } A → T_2, B → T_3 }

$S_1 \neq S_2$

③ write - read sequence

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

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

$S_1 \neq S_2$

* Problem due to concurrent Execution:

1) WR [write - Read] / Uncommitted Read / Dirty Read problem
2) RW [Read - Write] / Non Repeatable Read problem
3) WW [write - write] / Lost update problem
4) Phantom tuple problem
   Incorrect Summary problem

* Finding total no. of ~~sched~~ Schedule

Total no. of Schedule = Serial + Non serial

Non serial = Total - Serial (n!)

Total = $\frac{(n_1 + n_2)!}{n_1!\, n_2!}$ { $T_1 \rightarrow n_1$ operation, $T_2 \rightarrow n_2$ operation }

Total no. of Non Serial schedule $= \frac{n_1 + n_2 + \dots n_m}{(n_1)!(n_2)!\dots(n_m)!} - m!$

* Phantom Tuple Problem

T_1	T_2
Select * from employee where salary > 4700	
	insert into employee value $< e_4, D, 6700 >$
Select * from employee where salary > 4700	

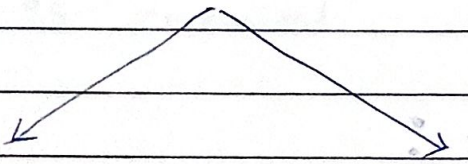
Employee

eno.	ename	salary
e_1	A	5000
e_2	B	6000
e_3	C	4500
e_4	D	6700

e_1	A	5000
e_2	B	6000
e_4	D	6700

Phantom Tuple

*



Serializability

↓

consistent

→ Conflict serializable

→ View serializable

Recoverability

↓

Recover if any case of fa

→ Recoverable schedule

→ Cascadeless schedule

→ Strict Recoverable sched

* Recoverable Schedule :

⇒ A Recoverable schedule is one, for each pair of transaction T_i & T_j such that, T_j reads a data item that was previously written by T_i then commit of T_i appear before commit of T_j.

T_1	T_2
W(A)	
	R(A)
C/R	
	commit

⇒ Recoverable schedule may/may not free from
1. WR/uncommitted Read
2. RW Problem
3. WW problem

⇒ Cascading Rollback are possible in Recoverable schedule.

* Cascading Rollback :

T_1	T_2	T_3	T_4
W(A)			
	R(A)		
↑ Rollback		R(A)	
			R(A)

⇒ T_2, T_3 & T_4 depends on T_1

⇒ If T_1 fails, Due to dependency T_2, T_3 & T_4 also Rollback

* Cascadeless Schedule :

⇒ A cascadeless schedule is one, where for each pair of transaction T_i & T_j such that T_j read a data item that was previously written by T_i, then commit of T_i appear before Read of T_j.

T_1	T_2
W(A)	
C/R	
	R(A)

⇒ No uncommited read / No WR problem

⇒ No cascading Rollback

⇒ Cascadeless schedule may or may not be free from
- RW problem
- WW problem

* Strict Recoverable Schedule :

⇒ can't read or write after T_1 is commited

T_1	T_2
W(A)	
C/R	
	R(A)/W(A)

*

① Recoverable schedule

T_1	T_2
W(A)	
	R(A)
C/R	
	↑ commit

② Cascadeless Schedule

T_1	T_2
W(A)	
C/R	
	R(A)

- WW Problem
- RW Problem

③ Strict cascadeless Schedule.

T_1	T_2
W(A)	
C/R	
	R(A)/W(A)

- RW problem

* Find No. of conflict serializable

Q) $T_1 : r_1(x)\ w_1(x)\ r_1(y)\ w_1(y)$
$T_2 : r_2(y)\ w_2(y)\ r_2(z)\ w_2(z)$

⇒
$T_1 \rightarrow T_2$
$T_2 \rightarrow T_1$

① $T_1 \rightarrow T_2$

$r_2(y)$ — $w_2(y)$ — $r_2(z)$ — $w_2(z)$
$w_1(y)$ $w_1(y)$ $w_1(y)$ $w_1(y)$

∴ $T_1 \rightarrow T_2$ ⟶ ① //

② $T_2 \rightarrow T_1$

$w_2(y)$ $w_2(y)$
$r_1(x)$ — $w_1(x)$ — $r_1(y)$ — $w_1(y)$

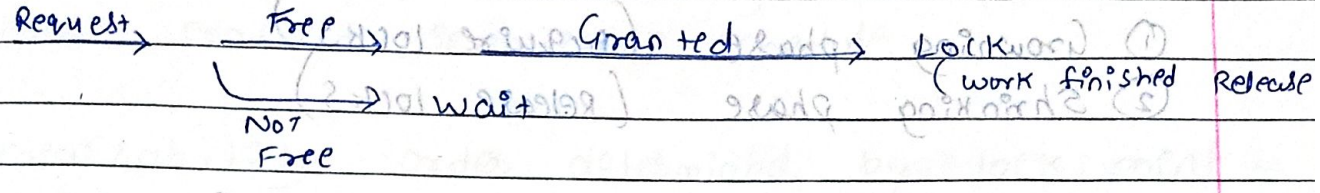
⇒ $r_2(z)$ & $w_2(z)$ can be placed anywhere, but after $w_2(y)$

⇒ we get total 53 ways of $T_1 \rightarrow T_2$

∴ 54 //

* Implementation of concurrency control

* Lock Based protocol



① Shared lock [S] [only read]

② Exclusive lock [X] [write / write(A) Read(A) | Read(A) write(A)]

Shared Lock (S)

```
Lock - S(A)
Read (A)
Unlock - S(A)
```

Exclusive lock [X]

```
Lock - X(A)
W(A) or R(A) or W(A) or W(A) R(A)
Unlock - X(A)
```

Eg:

T_1	T_2	Lock-manager
lock-X(A)		Grant - X(A, T_1)
Read(A)		
write(A)		
unlock-X(A)		Release - X(A, T_2)
	Lock-S(A)	Grant_S(A, T_2)
	Read(A)	
	unlock S(A)	Revoke - S(A, T_2)

* Two phase locking protocol [2PL]

⇒ Lock & unlock requests done in 2 phases

① Growning phase (Acquire locks)
② Shrinking phase (Release locks)

⇒ Each Transaction first finish its Growning phase then start shrinking phase.

① Growning phase :

⇒ Transaction may obtain the lock But must not release any lock.

② Shrinking phase :

⇒ In shrinking phase, Transaction Release the lock But must not ask for any new lock.

* lock point : [serializability order]

⇒ Position of last lock operation (or) first unlock operation.
(or)
Position where shrinking phase of the transaction start.

Eg: * → lock point
* [first unlocking]

T_1	T_2	T_3
	*	
*		
		*

serializability : $\langle T_2, T_1, T_3 \rangle$

Important point about 2PL

(1) 2PL Ensures conflict serializability.
If a schedule is allowed by 2pl then its Ensure conflict serializable schedule.

(2) Serializability order determined by lock point

(3) 2PL Not Ensure recoverability

(4) 2PL may suffer from Deadlock

(5) 2PL Suffers from Starvation

* Strict 2PL : 2PL + All exclusive lock taken by transaction must be hold until commit/Rollback

⇒ conflict serializable.
⇒ Recoverable schedule
⇒ Cascadeless
⇒ Strict Recoverable
⇒ Suffers from • Deadlock
• Starvation

* Rigorous 2PL : 2PL + All locks must be held back until C/R.

⇒ Suffers from • Deadlock
• Starvation

* Conservative 2PL :

⇒ Each transaction Acquire All the lock in the beginning of Execution & release after commit.

⇒ Conflict serializability
⇒ Recoverability
⇒ Cascadeless
⇒ Strict Recoverable
⇒ No Deadlock

⇒ Suffers from Starvation.

* Time stamp protocol : [TSP]

⇒ A unique Time stamp value assigned to each transaction when they arrive in the System.

⇒ Based on this timestamp, serializability order is determined.

T_1 : 10
T_2 : 20

T_1 → T_2

T_1 ↓ older Transaction

T_2 ↓ yonger transaction.

* Types of Time-stamp:

(1) Transaction Time stamp

(2) Data Item Time stamp.
   i) Read-Time stamp : RTS(Q)
   ii) Write-Time stamp : WTS(Q)

i) Read-Time stamp:

⇒ Denotes the highest [youngest] Transaction Time Stamp that perform Read(Q) operation succesfully.

Eg:

10	20	30
T_1	T_2	T_3
R(A)		
		R(A)
	R(A)	

RTS(A) = 30

ii) Write-Time stamp:

⇒ Denotes the highest [youngest] Transaction Time stamp that perform write(Q) operation succesfully.

* Data item Stamp:

T_i : Read	T_i : Write
$TS(T_i) < WTS(Q)$ not allowed & T_i Rollback	$TS(T_i) < RTS(Q)$ NOT allowed $TS(T_i) < WTS(Q)$

* Important point about TSP :

1) If schedule followed by TSP then ensure conflict serializable.

2) TSP not ensure recoverability & cascading Rollback possible.

3) Free from deadlock

4) ~~Starve~~ Starvation may occur.

* Thomas Write Rule

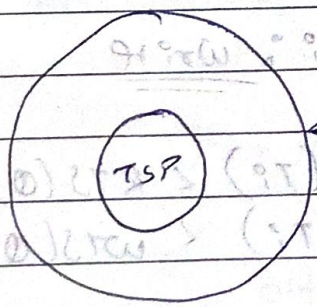
⇒ T_i : Read (Q)

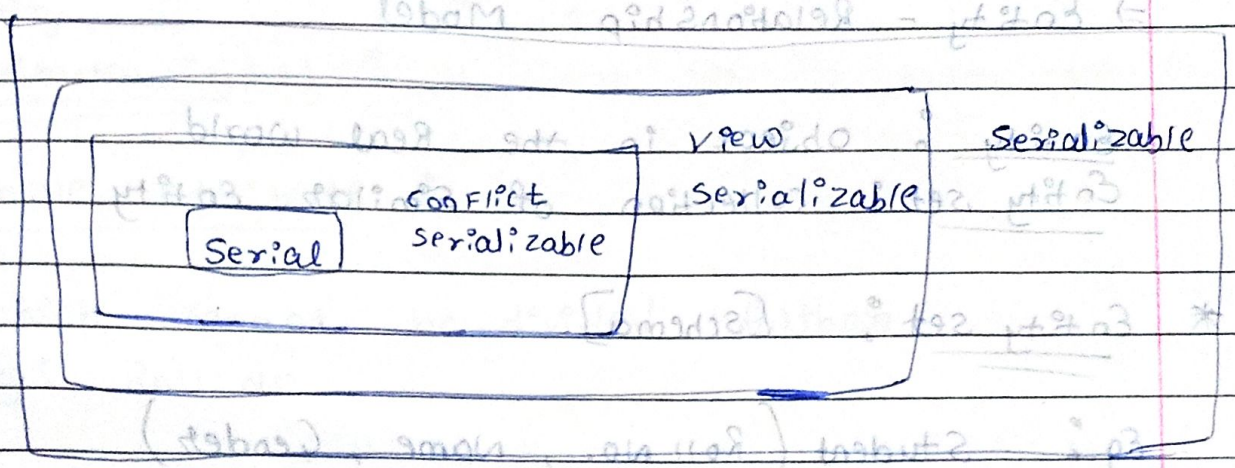
$TS(T_i) < WTS(Q)$: Read operation reject & T_i Rollback.

⇒ T_i : write (Q)

$TS(T_i) < WTS(Q)$; write operation ignored & No Rollback.

$TS(T_i) < RTS(Q)$; Reject & T_i Rollback




Serializable
View
Serializable
Conflict
Serializable
Serial

ER MODEL

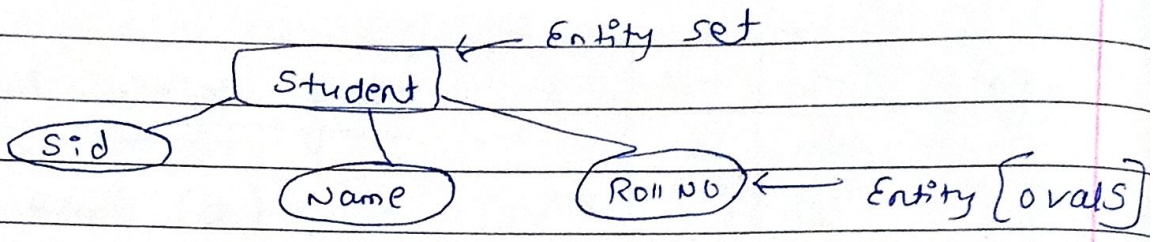
⇒ Entity - Relationship Model

Entity : object in the Real world.
Entity set : collection of similar Entity

* Entity set : [schema]

Eg : Student (Roll No. , Name , Gender)

⇒ Rectangles ⟶ Student



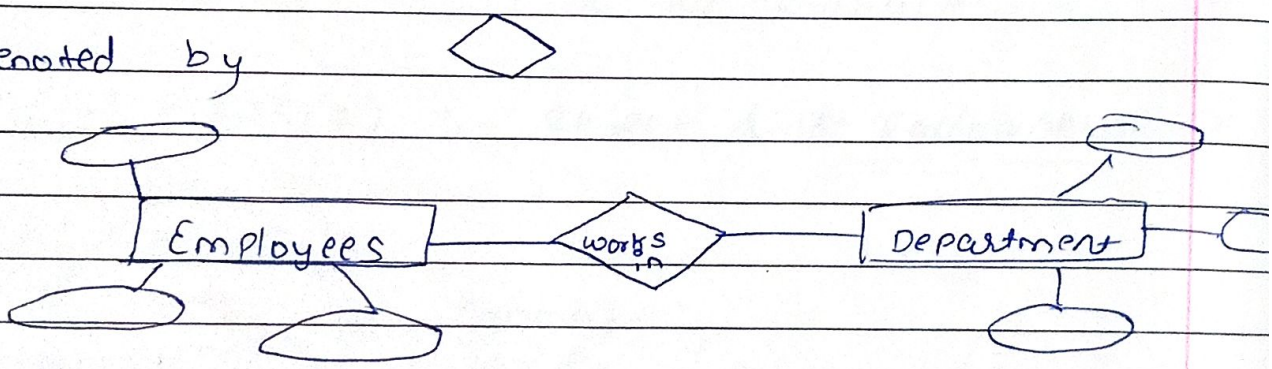
* Relationship sets :

⇒ A relationship is an association among several Entities.

⇒ A relationship set ⟶ collection

⇒ Denoted by

Eg :



* Attributes are properties used to describe an entity.

Attribute Types :

1] Simple attribute :

⇒ which cannot be divided further.

Eg: Roll No

2] Composite attribute :

⇒ which can be divided further

Eg: Name → First Name
→ Middle Name
→ Last Name

3] Single valued attribute :

⇒ which takes one value per entity.

Eg: Gender, Roll No., Result } Gender

4] Multi valued attribute :

⇒ Which takes more than one value per entity

Eg: Mobile No → Mobile No.

5] Stored attribute :

⇒ which does not require any updation.

Eg: DOB

6] Derived attributes :

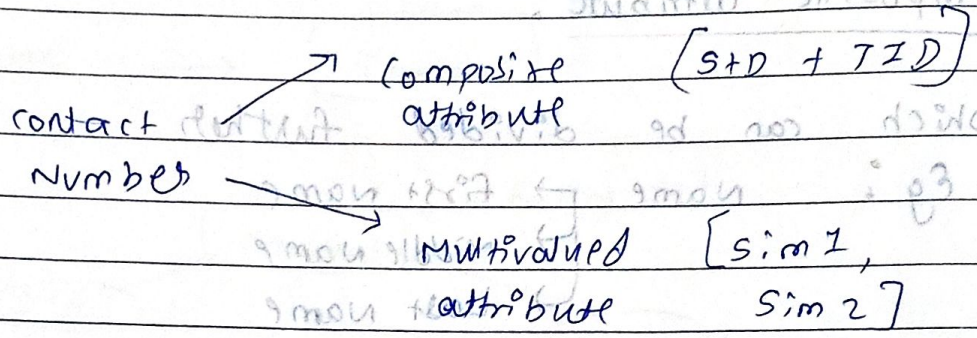
⇒ The value of the attribute derived from other attribute.

Eg: (Age) , (year of service)

7] Complex attribute :

⇒ [Multivalued + composite] attribute

Eg:



8] Key attribute :

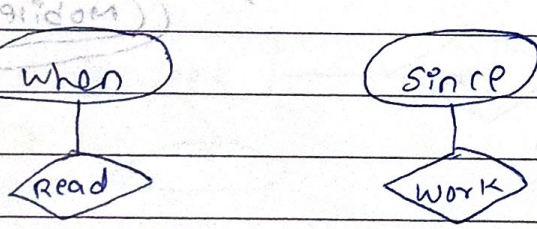
⇒ which uniquely identify an Entity in the Entity set

Eg: (Roll No.)

9] Descriptive attribute :

⇒ which gives information about the relationship set

Eg:



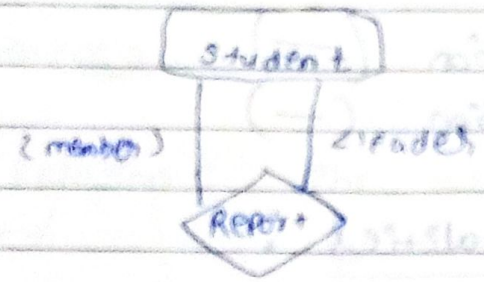
* Degree of Relationship set :

⇒ No. of Entity set participate in a relationship set

1. unary
2. Binary
3. Ternary
4. n-ary

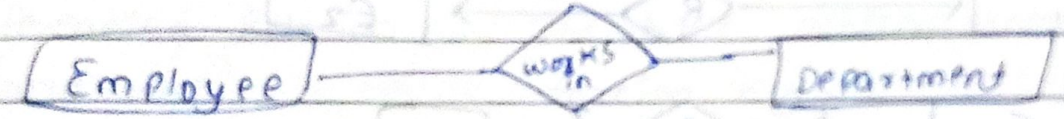
① unary

⇒



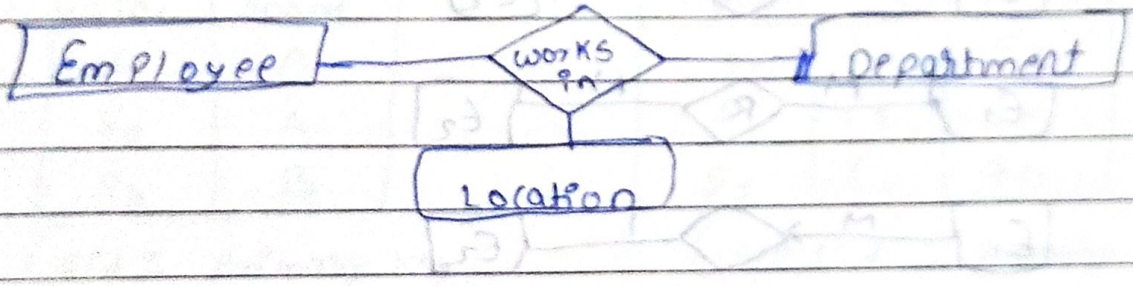
② Binary

⇒

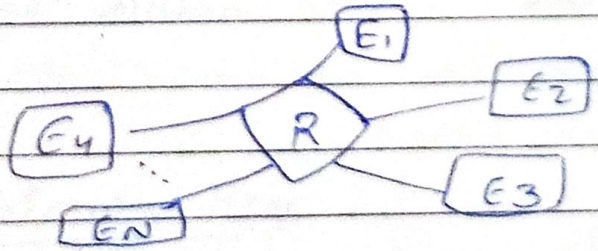


③ Ternary

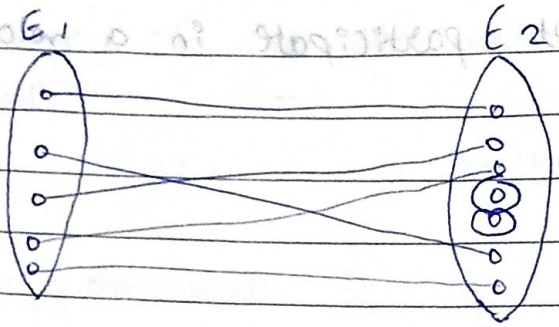
⇒



④ N-ary.



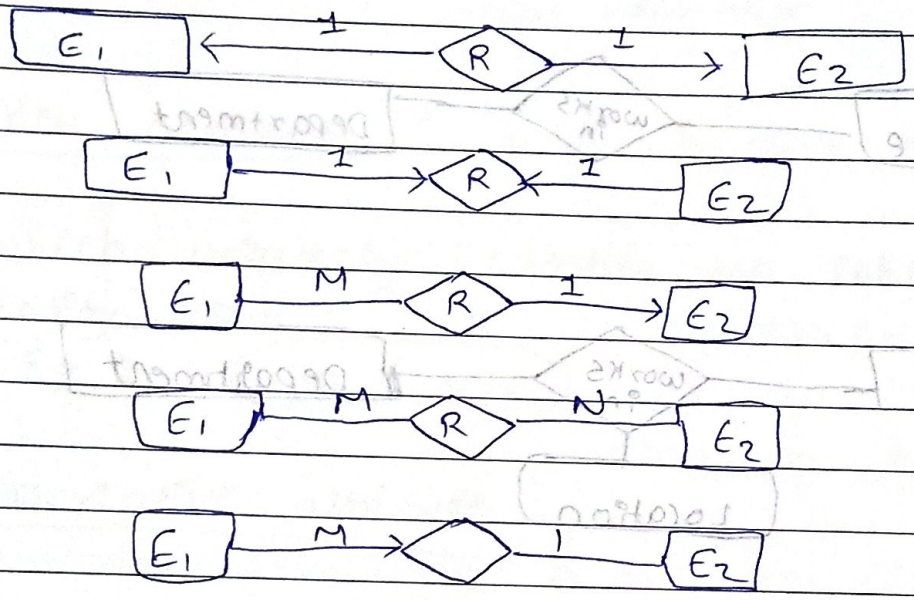
* Participation



Full participation Partial participation

⇒ Total participation (=)
⇒ Partial participation (—)

* Mapping cardinalities :



* Foreign key :

⇒ Foreign key is a set of attributes that referenced primary key or alternative key of the same relation or other relation

foreign key

going ow of India
verb

Referencing

Referenced Relation : The table which is referenced by foreign key.
[Parent table / Relation]

Referencing Relation : The table which contains the foreign key.
[Child table]

Eg :

Student	
sid	sname
S_1	A
S_2	A
S_3	B

[sid : primary key]
Referenced relation

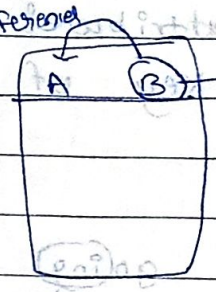
Enrolled		
sid	cid	Fee
S_1	C_1	5K
S_1	C_2	6K
S_2	C_1	7K

[sid, cid : primary key]
Referencing relation

NOTE : Primary key
↓
Unique + NOT NULL

} EnP

NOTE : ① By Default foreign key referenced to primary key of Referenced relation.



② • PK unique & Not Null.
• The value present in FK must be present in primary key of Referenced Relation.
• FK may contain Duplicate & Null values

* Student table → Referenced table
Insert ✓
Delete ✗

Child table → Referencing table
Insert ✗
Delete ✓

* Referenced Relation :
⇒ Insertion : No violation
⇒ Deletion : May cause violation.

i) on delete no action :
⇒ If cause problem on delete then deletion is not allowed on table.

ii) on delete cascade :
⇒ If we want to delete primary key value from referenced table then it will delete that value referencing table also.

on delete set null :

⇒ If we want to delete primary key value from referenced table then it will try to set the null values in place of that value in referencing table

on delete cascade : whenever primary key Deleted, then corresponding Tuple (value) from the Referencing relation Deleted cascadely.

Eg: If we want to delete (E2, Null) then No. of additional Deleted to preserve Referential Integrity

eid	ename ...	SID
E1		E2
E2		Null
E3		E2
E4		E3
E5		E4
E6		E5
E7		E6

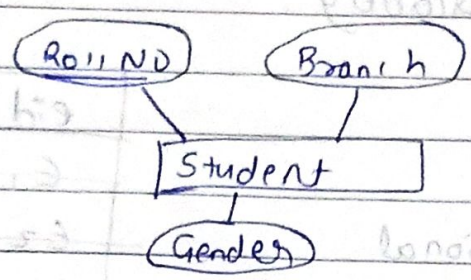
⇒ All tuples Deleted.

* ER Model → RDBMS

ER Model	RDBMS
Multivalued attribute	No multivalued
Composite Attribute	No composite
Weak Entity concept	No weak entity concept

1] Entity set → Relation (Table)
Key attribute → Primary key
Entity → Tuple.

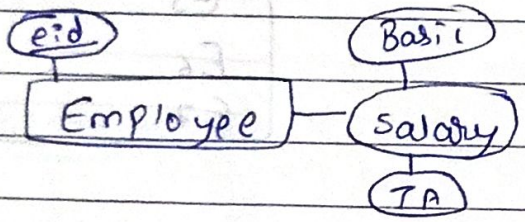
Eg:



→

Student		
RollNO	Branch	Gend

2] Composite attribute → set of simple component.

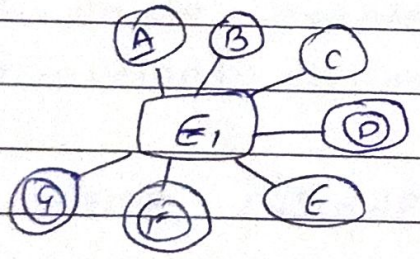


→

Employee		
eid	Basic	TA

3] Multivalued Attribute → 2 Table
→ Key Attribute + All other attribute
→ Key Attribute + All multi-valued attribute

Eg:



→

E1
A B C D E

E1'
A D F G

* Partial participation :

1] 

} 2 table

$E_1(\underline{a_1}\ a_2)$ $E_2(\underline{b_1}\ b_2\ a_1)$ F.K

2] 

} 2 table

$E_1(\underline{a_1}\ a_2\ b_1)$ $E_2(\underline{b_1}\ b_2)$

3] 

} 2 table

$E_1(\underline{a_1}\ a_2\ b_1)$ FK $E_2(\underline{b_1}\ b_2)$

(or)

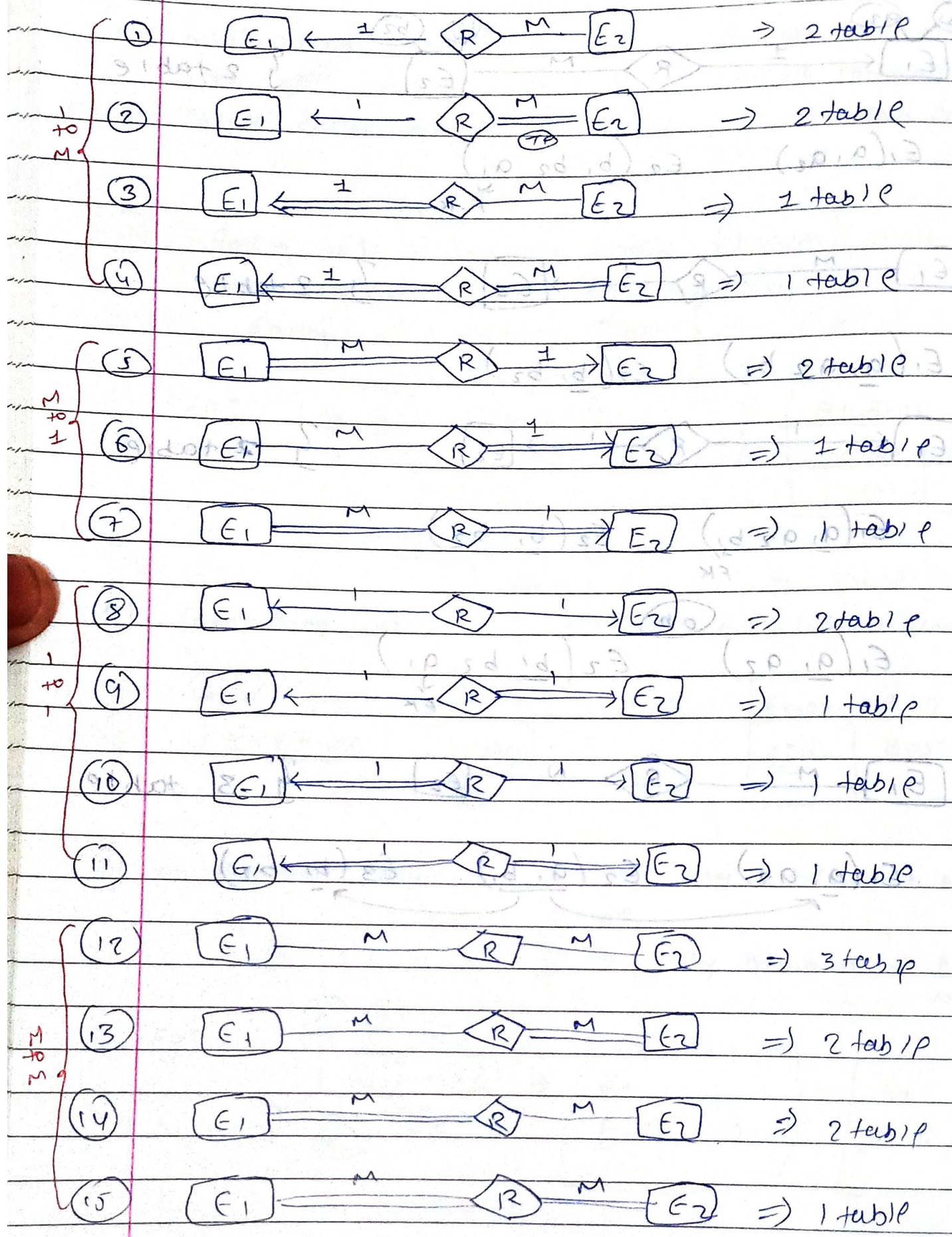
$E_1(\underline{a_1}\ a_2)$ $E_2(\underline{b_1}\ b_2\ a_1)$ FK

4] 

} 3 table

$E_1(\underline{a_1}\ a_2)$ $E_2(\underline{a_1}\ \underline{b_1})$ $E_3(\underline{b_1}\ b_2)$

* Total participation :



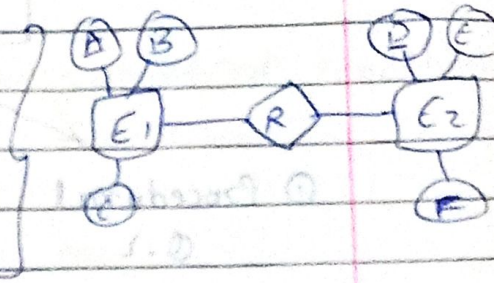
* Mapping

1 : 1 → CK : A or D

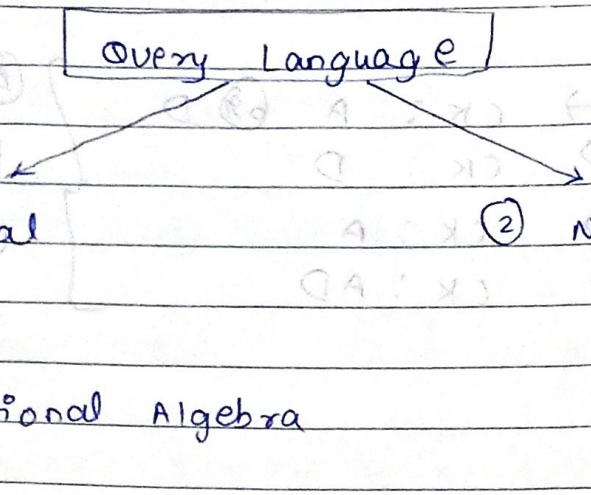
1 : M → CK : D

M : 1 → CK : A

M : N → CK : AD



Query Language



⇒ The basic idea of query language is query executed on a DB tuple by tuple one tuple at a time

⇒ Relational algebra	SQL
⇒ By default distinct output	⇒ By default Duplicate Retain

* Relational Algebra :

Basic operators :

① selection [σ]
② Projection [π]
③ cross product [×]
④ union [∪]
⑤ set difference [−]
⑥ Rename [ρ]

Derived operators :

① Join (⋈) & its type
② Division [/]
③ Intersection (∩)

1] Selection :

⇒ It select the tuples Based on specified condition

Eg: $\sigma_{condition}(Relation)$

2] Projection :

⇒ It select field / attributes from Relation

Eg: $\Pi_{attributes}(Relation)$

NOTE : ① $\sigma_{C3}(\sigma_{C2}(\sigma_{C1}(R))) \equiv \sigma_{C2}(\sigma_{C1}(\sigma_{C3}(R)))$

Imp

② $\Pi_{A1}[\sigma_{C1}(R)] \neq \sigma_{C1}[\Pi_{A1}(R)]$

③ $\sigma_{C1}(\Pi_{A1}(R)) \longrightarrow \Pi_{A1}[\sigma_{C1}(R)]$

↳ If condition is applied on A_1 attribute

④ $\Pi_A[\Pi_{AB}(R)] = \Pi_A(R)$

Imp ⑤ $\Pi_a(R) \equiv \Pi_a(\Pi_b(R))$

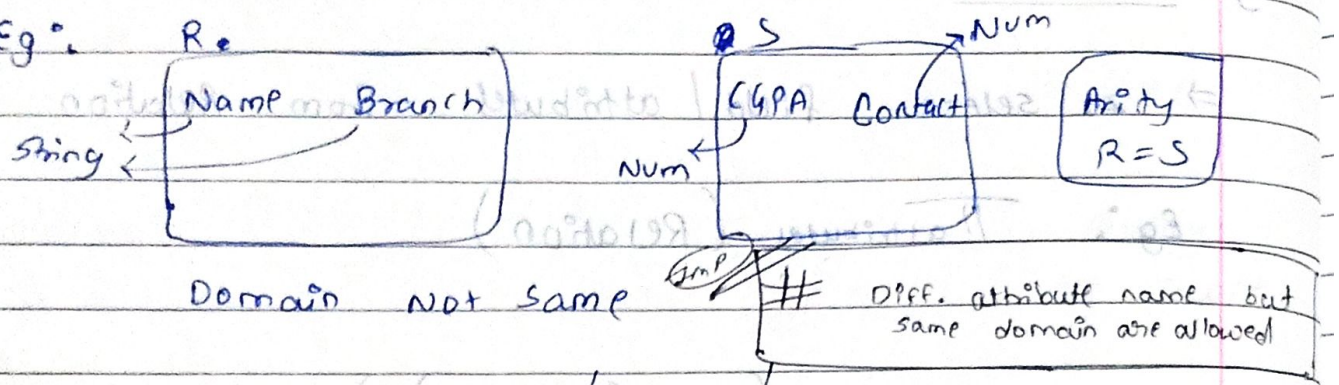
iff $a \subseteq b$

* Set operators [∪, ∩, −]

⇒ To apply set operators Relation must be union compatible. [Type compatible]

① Arity [# of attributes] of R & S must be same
② Range of attribute must be similar

Eg:



* Set difference operation / Minus / Except

(R − S) ⇒ Notation

* CROSS product

R S

n1 Tuple n2 Tuple
c1 attribute c2 attribute

⇒ R×S = n1×n2 Tuple
c1+c2 attribute

* Join operation

1] Conditional
2] Equi
3] Natural
} Inner Join

4] Left outer
5] Right outer
6] Full outer
} outer Join.

1] Natural Join [R ⋈ S]

Step 1: cross product of R x S

Step 2: select Tuple which satisfy Equality condition on all common attribute of R & S.

Step 3: Projection of Distinct attribute.

$$R \bowtie S = \Pi_{\text{Distinct Attribute}} \left[\sigma_{\text{equality condition on all common attribute}} (R \times S) \right]$$

2] Conditional Join [R ⋈c S]

$$R \bowtie_c S = \Pi_{\text{All attributes}} \left[\sigma_{\text{given condition}} (R \times S) \right]$$

Step 3 — Step 2 — Step 1

Eg: $R \bowtie_{R.sid > S_1.sid} S = \Pi_{\text{all attributes}} \left[\sigma_{R.sid > S_1.sid} (R \times S) \right]$

NOTE: Dangling Tuple → is a tuple that fail to match any Tuple of any other relation in common attribute

Spurious Tuple → Extra tuple

3] Equi - Join $[R \bowtie_\theta S]$:

⇒ It is a subset of conditional Join (or) similar to conditional Join But here 'only equality condition' applied.

$$R \bowtie_\theta S = \Pi_{attributes}\left[\sigma_{equality\ condition}(R \times S)\right]$$

⇒ Natural Join → Equi Join on common fields

* Outer Join :

⇒ Because in Inner Join some Tuple failed to satisfying the Join condition [Dangling Tuples] so loss of Data.

Outer Join = Natural Join + Dangling Tuples

= Inner Join + Dangling Tuples

4] Left outer Join :

⇒ $R \bowtie S$ or $R \bowtie_L S$:= $R \bowtie S$ && include the Tuples from left side Relation R those failed to satisfy condition.

⇒ $R \overset{\circ}{\bowtie}_c S$ = $R \bowtie_c S$ + Include the tuples from Left side relation R those failed condition

Eg:

R

A	B	C
1	2	4
3	2	6

S

B	C	D
2	4	8
2	7	4

$(R \bowtie S)$ =

A	B	C	D
1	2	4	8

$R \overset{\circ}{\bowtie} S$ =

A	B	C	D
1	2	4	8
3	2	6	NW1

5] Right outer Join

$R \overset{\circ}{\bowtie} S$ = $R \bowtie S$ && Include right side of dangling tuples.
(or)
$R \overset{\circ}{\bowtie}_R S$

6] Full outer Join :

$R \overset{\circ}{\bowtie} S$ = $R \overset{\circ}{\bowtie} S \cup R \overset{\circ}{\bowtie} S$
or
$R \overset{\circ}{\bowtie} S$

Eg:

A	B	C	D
1	2	4	8
3	2	6	NW1

$R \overset{\circ}{\bowtie} S$

∪

A	B	C	D
1	2	4	8
NW1	2	7	4

$R \overset{\circ}{\bowtie} S$

⇒

A	B	C	D
1	2	4	8
3	2	6	NW1
NW1	2	7	4

$R \overset{\circ}{\bowtie} S$

* Rename operator :

1] Table rename : $\rho(Temp, stud)$

2] Renaming all attribute : $\rho_{(S,N,A)}\ stud$

3] Renaming some particular attribute : $\rho_{1\to S,\ 2\to N}(stud)$

* Division operator [/]

$$\frac{\Pi_{AB}(R)}{\Pi_{B}(S)} = Quot.(A)$$

↙

It will retrieve values of attribute 'A' from R for which there must be pairing 'B' value for every 'B' of S.

→ $\Pi_{sid}(Enrolled) - \Pi_{sid}\left[\Pi_{sid}(Enrolled) \times \Pi_{cid}(course) - Enrolled\right]$

NOT enrolled every course

* SQL

SELECT ≡ Projection
FROM ≡ Cross product
WHERE ≡ Selection

⇒ No. of SQL clause mandatory = 2 // ⇒ From, select

Execution sequence

From
↓
where
↓
Group By
↓
Having
↓
select
↓
Distinct
↓
Order By

Group By clause :

⇒ It groups the table based on the specified attributes

$R \xrightarrow[\text{Branch}]{\text{GroupBy}} R'$

* Rename operator / Alias

Keyword → AS, (space)

eg: FROM supplier AS s, Parts - P

* Aggregate function :

⇒ functions that takes collection of values as i/p and produce single output.

1) ~~Count~~

NOTE : ① NULL + 100
NULL - 100
NULL x 100
NULL ÷ NULL
} NULL.

② comparison operation with NULL gives result 'UNKNOWN'.

③ Aggregate operator always discard / ignores the NULL value.

sid	Branch	Marks
S_1	CS	90
S_2	IT	70
S_3	CS	70
S_4	EC	56
S_5	CS	NULL

1] count [Marks] = 4
2] count (*) = 5
3] count ([Distinct] Marks) = 3
4] sum (Marks) = 286
5] Sum ([Distinct] marks) = 216
6] Avg [Marks] = $\frac{286}{4}$
7] Min [Attribute] = 56.
8] Max [Marks] = 90

* Having :

=) It is used to select the group which satisfy the condition.
[condition is for each group]

Eg:

Sid	Branch	Marks	Avg
S_1	CS	60	75
S_2	CS	90	
S_3	IT	70	65
S_4	IT	60	
S_5	EC	55	55
S_6	EC	NULL	

=>

Sid	Branch	marks
S_1	CS	60
S_2	CS	90
S_3	IT	70
S_4	IT	60

* SQL set operators :

1. UNION & union all
2. Intersect & Intersect all
3. Minus [Except] & Minus all [Except all]

Followed by RA
↓
Duplicates not allowed

Not followed by RA
↓
Duplicates allowed.

Nested Queries

- Normal (Independent) Nested Query
- Correlated Nested Query

↓

Execution sequence

⇓

Bottom → Top

Inner query → outer query.

* Other set operator

1. IN / NOT IN
2. ANY
3. ALL
4. EXISTS / NOT EXISTS

Comparision operator

<, >, <=, <>

<> ↑ Not equal

ANY (OR)

⇒ x > Any (10, 20, 40)

↓

o/p → 11, 12, 13, ...

ALL (AND)

⇒ x > All (10, 20, 40)

o/p → 41, 42, 43, ...

Eg: Select Sname From Supplier
where [Turnover > Any (select turnover from Suppliers where city = 'Delhi')]

IN : Membership set

⇒ x IN R , if R contain x then True.

Eg: x = 5 , R$_{old}$: (1,2,3,4,5)

⇒ x NOT IN R, if R does not contain x then True.

Corelated nested Query :

Execution sequence : Top → Bottom → Top
Outer → Inner → outer

Exists :

⇒ Return True if inner query result Non empty.

Not Exist :

⇒ Return True if inner query result empty

IS / IS NOT

⇒ For comparision with NULL. { Imp

Like / Not like.

⇒ used to compare to specify certain search condition for a pattern in a column.

A% → starts with A
%J → Ends with J
%I% → contains I
'_____' → All 4 length name
'S___' → Start with S & 4 length.

'S___%' → starts with S & at least 4 length.

Order By :

⇒ order by clause is used to sort the Rows.

By Default : Ascending [Asc]
Descending [Desc]

* Join with Foreign key concept :

⇒ Whenever two table (Relation) are Joined (Natural Join) with respect to primary key & Foreign key then maximum Number of Tuples in the resulting relation is equal to number of tuples in the referencing relation.

Eg: R(A B C) S(B C D)

Referencing rel^n (B is foreign key) → 1000 Tuples

referenced relation → 5000 Tuples

⇒ R ⋈ S = 1000 → Max tuples

Q. Supply (SupplierID, Itemcode) — 1000 Tuples
Inventory (Itemcode, Colour) — 2500 Tuples

⇒ Maximum # of Tuples = 1000
Minimum # of Tuples = 1000

⇒ The value present in foreign key must be present in primary key
⇒ Foreign key may contain duplicate values & Null values
⇒ Item code is f.k. in Supply table but Here Itemcode is also key. Itemcode can not be null.

Supply (Not Null)

SID	Itemcode
S_1	P_1
S_1	P_2
S_1	P_3
S_2	P_1
⋮	⋮

Inventory

Itemcode	colour
P_1	Red
P_2	Green
P_3	⋮
⋮	⋮
P2500	⋮

* Imp NOTE :

1) Select S.age From Sailor s order By s.age ASC limit 1;

→ limit 1 tells you that you only want 1st row } Prints minimum age of sailor

2] You cannot use 'equal operation (=)' with NULL
↳ Instead we use 'IS' keyword

3] i) A constraint is Enforced only for an Insert operation on a table → False, we can also enforce for an update operation on a table.

ii) A Foreign can contain NULL values → True

iii) A column with unique constraint can store NULL but not duplicate

4] ALL (NULL) = NO ROW
ALL (Empty set) = All Row of outer table
ANY (NULL) = NO ROW
ANY (Empty set) = NO Row
} Selected

5] In 'on update cascade' → If we alter in the child table and that value is not present in parent table then update is rejected

If we update in parent table we need to update in child table.

File organization & Indexing

Database
|
DB Files
|
Records
|
Fields

Blocks

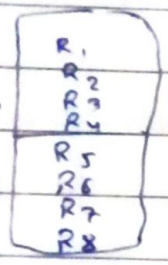
⇒ DB is collection of files
⇒ Each file is collection of records
⇒ Each record is a sequence of fields

⇒ DB is divided into number of blocks.
⇒ Each block is divided into records
⇒ Record can be stored in blocks.

* Blocking Factor [BF] :

⇒ Average number of ~~blocks~~ Records per block.

Eg:



Blocking Factor = 4

Blocking Factor = $\frac{\text{Block Size}}{\text{Record Size}}$

Blocking factor

① Spanned org

⇒ A record can be stored more than one block [Partially can be stored]

② Unspanned strategy org

⇒ A Record can be stored / belongs to a particular Block.

* Spanned org :

Advantage → No memory wastage

Disadvantage → Block acces Increase

Suitable for variable length record

* Unspanned org :

Advantage → Block access reduced

Disadvantage → Wastage of memory [Internal fragmentation]

Suitable for fixed length record

NOTE : i] Default organization → unspanned org.

ii] search key → attribute used to access data from DB

* organization of records in a file → i] ordered file org
→ ii] unordered file org

* ORDERED FILE	UNORDERED [HEAP] FILE
i] searching Easy	i] Insertion easy
ii] Insertion Expensive	ii] searching Expensive
iii] Binary search	iii] Linear search
↳ To access a record avg. no. of Block access $= \lceil \log_2 B \rceil$	↳ Average case $= \frac{B}{2}$
	worst case $= B$.

B : # Data block

* Indexing :

⇒ used to improve searching Efficiency
⇒ To reduce I/O cost.

one record of Index file contain 2 fields.

Search key	Block Pointer

one index record size = size of search key + size of Pointer

Q. Block size = 100 Byte.
Record size = 40 B
Block factor = 2 R/B

Key = 16 B , B_p = 4 B
one index record size = 20 B
Block factor of Index file $= \left\lfloor \frac{100}{20} \right\rfloor = 5$ //

Two basic kind of indices:

i] ordered indices → search key are stored in sorted order

ii] Hash indices → search keys are distributed uniformly across buckets using a hash function

NOTE: To access a record average no. of ~~access~~ blocks access.

$$= \lceil \log_2 B \rceil + 1$$

$\lceil \log_2 B \rceil$ → Index Block access
$+1$ → Data Block access

* category of Indexes

Dense Index

⇒ Index entry created for every search key value.

Sparse Index

⇒ Index entry created for some search key valu

Dense Index files:

Number of index Entries = Number of DB records.

Eg:

10101	→	10101	S	CS
12121	→	12121	W	F
.	→			
.	.			
.	.			
98345	→	98345	IK	EE

Sparse Index files:

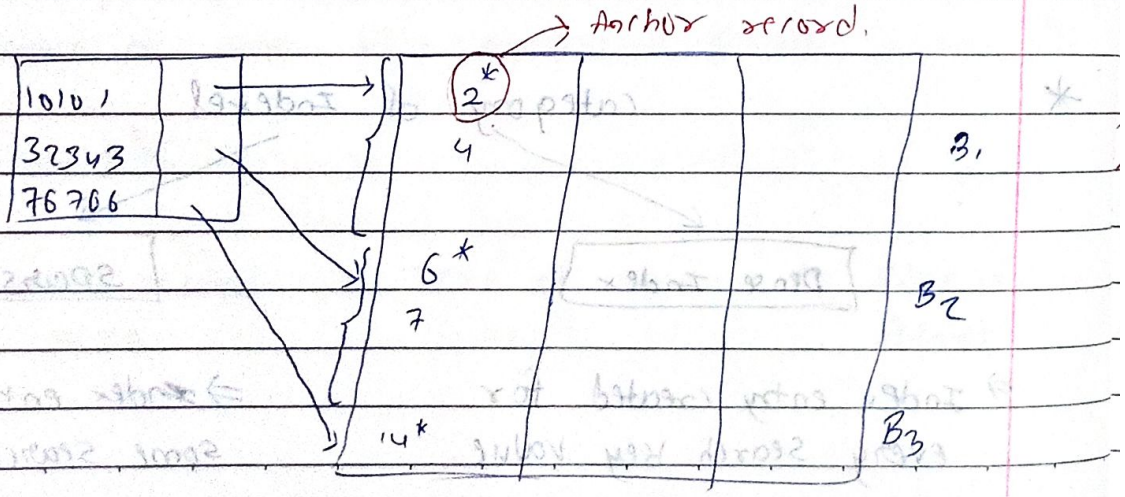
⇒ Applicable when records are sequentially ordered on search key.

⇒ To locate a record with search key value K we:

i) find index record with largest search key value < K

ii) search file sequentially starting at the record to which the index record points.

Eg:



Anchor record : First record of BLOCK

# of index Entries = # of DB BLOCKS

* Index

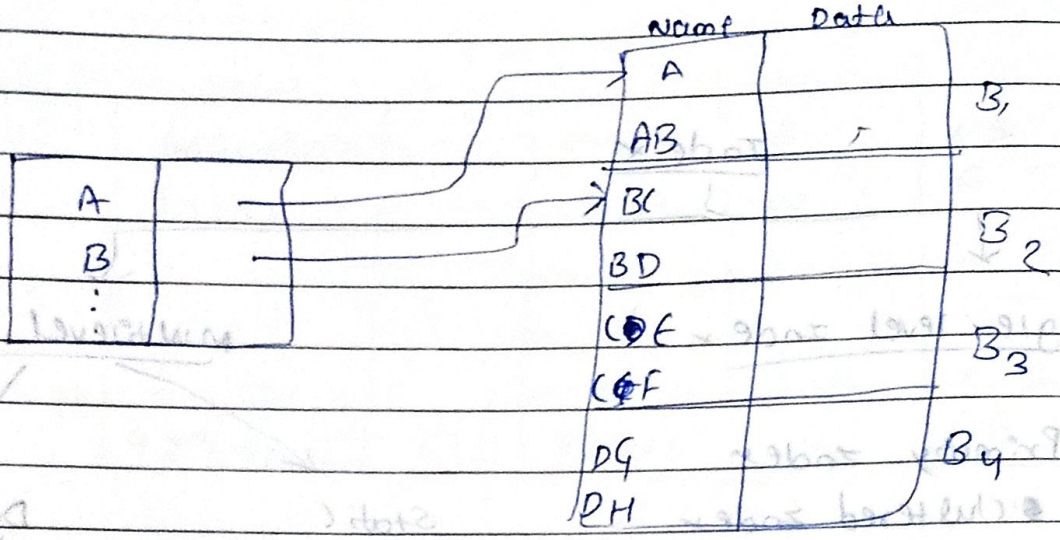
- Single level Index
  1. Primary Index
  2. Clustered Index
  3. Secondary Index
- Multilevel Indexing
  - Static multilevel Indexing
  - Dynamic multilevel indexing
    - B tree
    - B^+ tree

1. Primary Index [key + ordered DB file]
2. Clustering Index [Non key + ordered file]
3. Secondary Index [Non key/ + unordered file]
   (-K

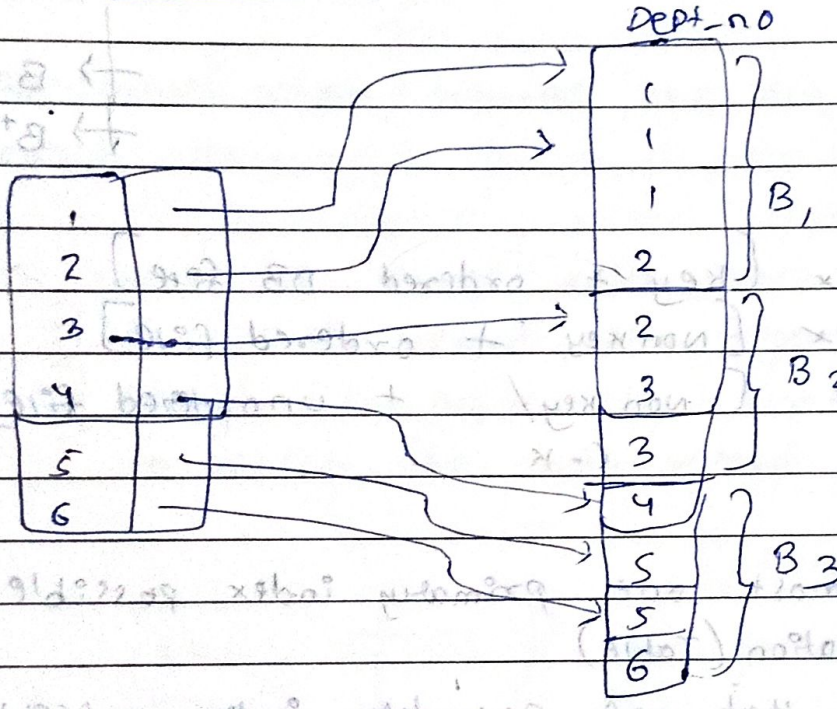
NOTE :
i) At most one primary index possible per Relation (Table)
ii) More that one secondary index possible
iii) In a relation either CI or PI, any one possible, not both

* Primary Index

↳ Dense or sparse
↳ Most Apply 'sparse' Index



* clustering index:



An index whose search key is different from sequential order of the file - Clustering Index

* Secondary index :

Case 1 : Non key + unordered

=> Name [SI] => Non key + unordered

primary index

Ac. No.	Name
1	R
2	A
3	Y
4	R

case 2 : Key + Unordered

=> Ac.No [SI] => key + unordered.

* Multilevel Indexing :

=> index to index file until 1 block index at last level

=> Idle access cost to access record using multilevel index is (n+1) blocks, n is the number of level in index.

=> outer index : a sparse index to basic index
inner index : the basic index file

Problem with static multilevel indexing

=> costly and time consuming

Soln : Dynamic multilevel indexing

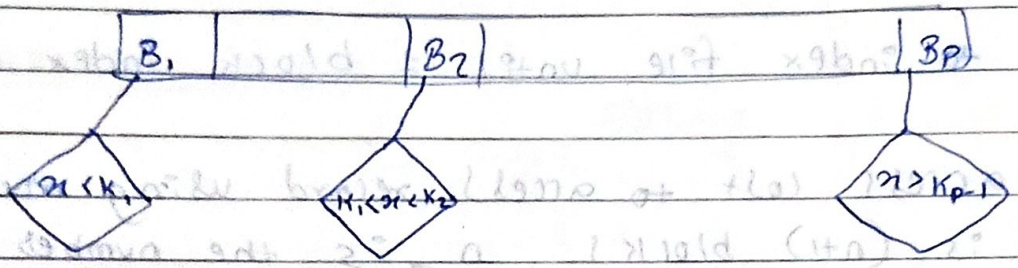
* B Tree :

order p → maximum number of block pointer is p.

B_1	$K_1 R_1$	B_2	$K_2 R_2$	...	B_{p-1}	$K_{p-1} R_{p-1}$	B_p

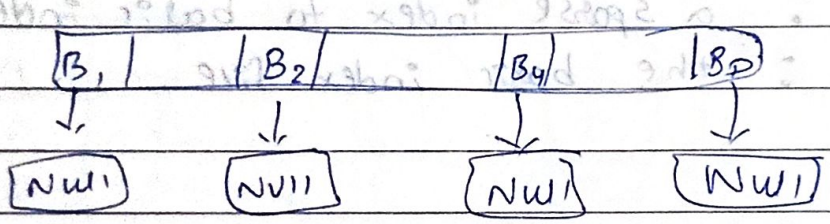
B_p : p → Block pointer
K_p key : p - 1 → search key
R_p : p - 1 → Read / Data / Record pointer

⇒ Structure of internal node



Imp

⇒ Structure of leaf node.



⇒ Every Internal Node except the Root node contain at least (min) $\lceil p/2 \rceil$ Block pointer and min $(\lceil p/2 \rceil - 1)$ keys

max p Block Pointer & p-1 keys

⇒ Root can contain 2 Block pointer min & p at max.

⇒ Keys within the node should be in asc. order

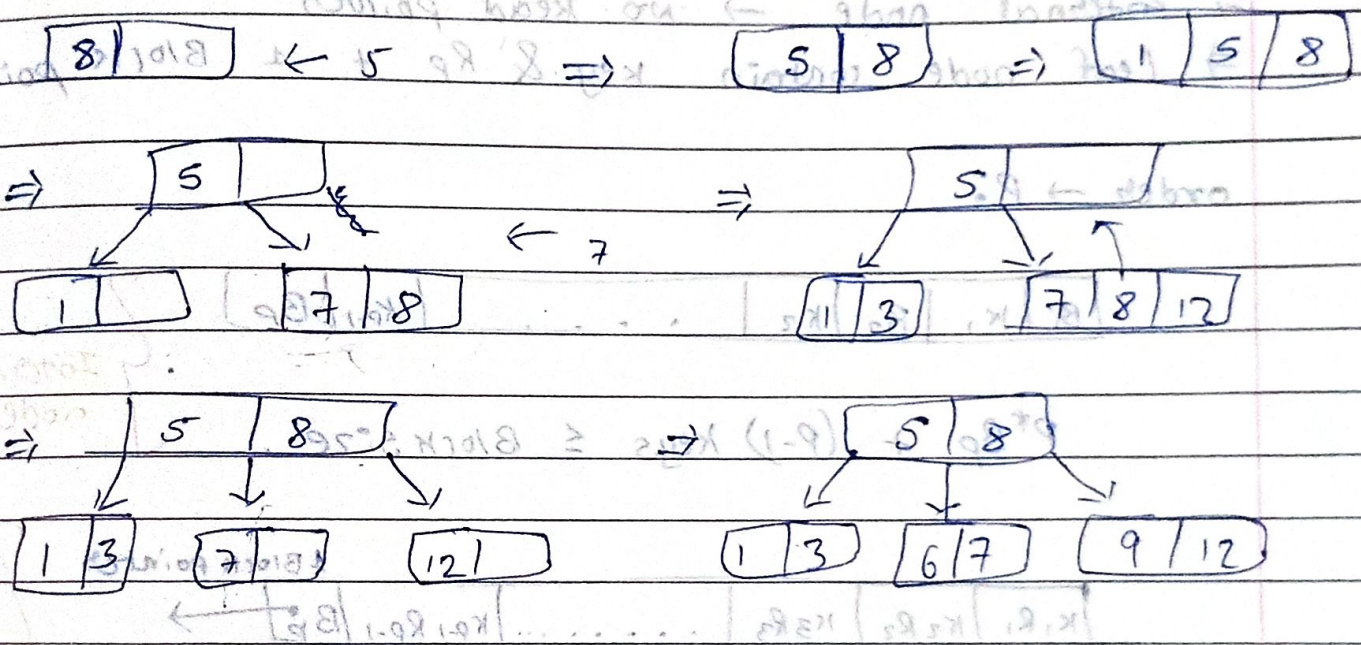
⇒ Every leaf node should be at same level

* order : p

	Min	Max	
Root	2	p	} Block Pointers
Non Root	$\lceil p/2 \rceil$	p	
Root	1	p-1	} keys
Non Root	$\lceil p/2 \rceil - 1$	p-1	

* Draw B Tree : [8, 5, 1, 7, 3, 12, 9, 6]

⇒



NOTE : Order → P

$P * B_p + (P-1)(Keys + R_p) \leq$ Block size

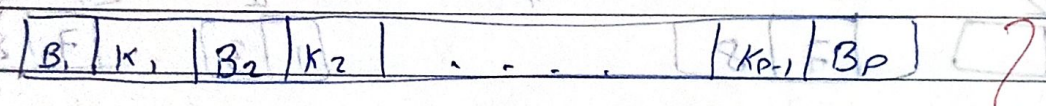
* (Level = height + 1) → Imp

Height/Level	min # Nodes	Min # Bp	Min # Keys
0/1	1	2	1
1/2	2	$2\lceil p/2 \rceil$	$2\lceil p/2 \rceil - 1$
2/3	$2\lceil p/2 \rceil$	$2\lceil p/2 \rceil^2$	$2\lceil p/2 \rceil (\lceil p/2 \rceil - 1)$
3/4			
4/5			
⋮			
n/n+1			

* B^+ tree:

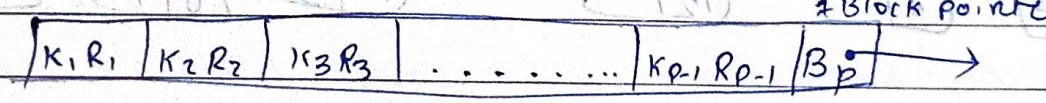
⇒ All keys are available at Leaf node.
⇒ Internal node → No Read pointer
⇒ Leaf node contain key & Rp + 1 Block pointer

order → P:



Internal nodes

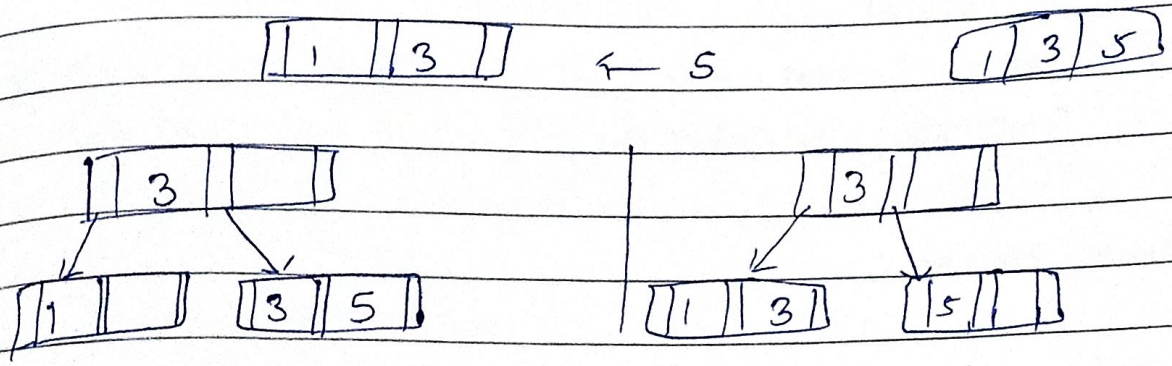
$P * B_p + (P-1)$ keys ≤ Block size.



Leaf node

$(P-1)[\text{keys} + R_p] + 1B_p \leq$ Block size

order 3: [1, 3, 5]



Right biasing

Left Biasing

FD's and Normalization

* Types of FD
  - Trivial ⇒ $x \to y \Rightarrow x \supseteq y$
  - Non Trivial FD ⇒ $x \to y \Rightarrow x \cap y = \phi$
  - Semi Non Trivial FD ⇒ $x \to y \Rightarrow x \cap y \neq \phi$ && $x \not\supset y$

* Attribute closure.

* Prime / key attribute : Attribute that belongs in any CK.

* Finding multiple candidate keys.

* Membership set : $A \to B$ logically implied then it is a member

$$[A]^+ = [\ldots, B]$$

* Equality btween two FD set :

$F : [\ldots]$ $\quad$ $G : [\ldots]$

F & G are equal if F covers G and G covers F.

F cover G :	True	True	False	False
G cover F :	True	False	True	False
	$F \equiv G$	$F \supset G$	$G \supset F$	Uncomparable

⇒ F cover G : If FD's of G can be implied by FD's of F.

* Finding # of super keys.

* Minimal covers : To eliminate the redundant FDs

$\{ A \to C , AC \to D , E \to AD , E \to H \}$

Step 1 : $A \to C , AC \to D , E \to A , E \to D , E \to H$

Step 2 : If $[A]^+ = [\ldots C]$ then C is extra ...

$\therefore A \to C , A \to D , E \to A , E \to D , E \to H$

Step 3 : Hide the FD working on & take closure,

$[E]^+ = [ACDEH]$ $\therefore$ it is redundant

* Properties of Decomposition :

① Lossless Join Decomposition → ① Basic concept
→ ② Binary method
→ ③ Chase method (Test)

→ Basic concept → Natural Join method

⇒ i] $R_1 \cup R_2 \neq R$
ii] If a common attribute of R_1 & R_2 neither a super key of R_1 Nor R_2

Lossy Join

$(R_1 \cap R_2)^+ \not\to R_1$
$(R_1 \cap R_2) \not\to R_2$

⇒ $R_1 \bowtie R_2 \ldots \bowtie R_n \equiv R$ → Lossless Join
$R_1 \bowtie R_2 \ldots \bowtie R_n \supset R$ → Lossy Join

② Dependency Preserving Decomposition

→ $F_1 \cup F_2 \cup \dots F_n \equiv F$ → Dependency preserved

→ $F_1 \cup F_2 \cup \dots F_n \not\equiv CF$ → NOT preserved.

* Closure of FD set :

Case I : when FD set is not given

ϕ
A
B
⋮

$\phi \to \phi$
$\phi \to A$
⋮

$A \to \phi$
$A \to A$
$A \to B$
⋮

no. of components $= n$

$$[F]^+ = n(n-1) + 1$$

Case II : when FD set is given

R : [A → B] ⇒ R(AB) → 2^n

ϕ	0 attribute	$2^0 = 1$
A	1 attribute	$(A)^+ = AB \Rightarrow 2^2 = 4$
B	1 attribute	$(B)^+ = B \Rightarrow 2^1 = 2$
AB	2 attribute	$(AB)^+ = AB \Rightarrow 2^2 = 4$

* Normal Form :

Redundancy level : 1NF > 2NF > 3NF > BCNF.

 # Partial dependency : $x \to y$

$(x - A) \to y$

$(A \in x)$

$x \to y$ is paltial FD.

Eg: AB → C is partial FD if

A → C (or)

B → C

2NF : [Proper subset of CK] → [Non prime attribute]

Eliminated by 2NF

⇒ It is in 2NF if every non prime attribute in R is fully functionally dependent on primary key.

3NF : $x \rightarrow y$

x : Superkey

(or)

y : Prime / key attribute

BCNF : $x \rightarrow y$ & $x \rightarrow$ Super key

NOTE : Dependency may or may not be preserved in BCNF

2NF Decomposition : Get the FD that violate 2NF & Divide the table

Eg: $(B \rightarrow E)$ $R(ABCDEFGH)$

$[B]^+ = BEFGH$

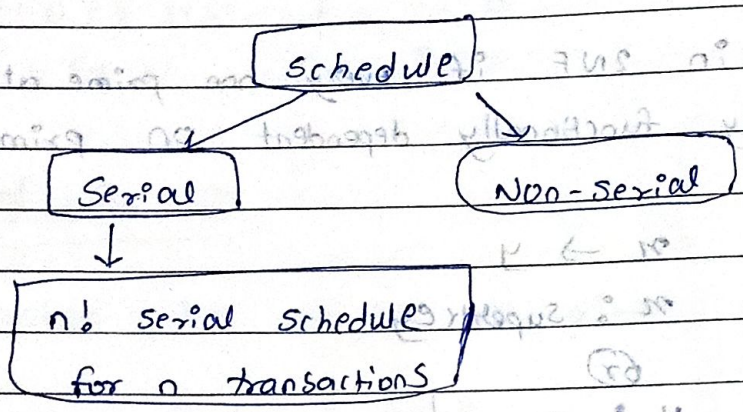
→ $R_1(ABCD)$ $R_2(BEFGH)$

⇒ If two FDs does not follow 2NF rule the it will be divided in 3 table.

Same Decomposition method for 3NF & BCNF.

Transaction & Concurrency Control

* **Schedule** : Time order sequence of two or more transactions.



* Serializable schedule → Conflict serializable
  → View serializable

* Conflict Serializable → Basic concept
  → Testing method [Precedence graph]
  → Conflict Equal to any serial schedule

* Topological sorting → Non serial schedule equivalent to which serial schedule.

* Conflict Equivalent : Two schedules are said to be conflict Equivalent if all the conflict operations in both the transactions are in same order.

* Conflict Serializable : Schedule is said to be conflict serializable if it is Conflict Equivalent to the serial schedule

NOTE : If S_1 & S_2 schedule are conflict equal then precedence graph must be same for the two.
Vice versa is not true.

* View Serializability : ① Initial read (Must be same)
② final write
③ updated read (write read sequence)

* Problems due to concurrent Execution :
⇒ WR, RW, WW, Phantom Tuple problem

* Total no. of schedule : Total = serial + non-serial

∴ Non serial = Total - serial

$$\text{Non-serial} = \frac{n_1 + n_2 + n_3 \ldots + n_m}{(n_1)!\,(n_2)!\ldots(n_m)!} - m!$$

* Recoverable

T_1	T_2
W(A)	
	R(A)
C/R	
	commit

Dependent transaction commits after the other one.

Problems : WR / RW / WW & cascading rollbacks

cascadeless

T_1	T_2
W(A)	
C/R	
	R(A)

commit before read

Problems : WW / RW

strict cascadeless

T_1	T_2
W(A)	
C/R	
	R(A)/W(A)

commit before read/write

Problem : RW

* Find no. of conflict serializable.

* Implementation of ~~~~ concurrency control

Lock based protocol → shared lock [S] [only read]
↳ Exclusive lock [X] [w/wR/RW]

Two phase locking protocol : [2PL]

=> 1) Growing phase → acquire but must not release locks
2) Shrinking phase → Release but cannot request locks

Lock point : Position where shrinking phase start.
* → first unlocking

T_1	T_2	T_3
	*	
*		
		*

Serializability : $\langle T_2, T_1, T_3 \rangle$

2PL : Suffers from deadlock & starvation

Strict 2PL : 2PL + All exclusive locks must be hold until commit / Rollback

Strict 2PL suffers from Deadlock & starvation

Rigorous 2PL : 2PL + All locks must be held until C/R

suffers from deadlock & starvation

Conservative 2PL : Locks acquired at the beginning of execution & release after commit

suffers from starvation

* Time stamp protocol : A unique time stamp is assigned to the transaction when they arrive

- Based on Time Stamp serializability order is determined

* Types of Time-stamp :

i) Transaction time stamp
ii) Data Item time stamp → Read Time stamp (RTS)
                        → Write Time stamp (WTS)

RTS : Denotes the highest [youngest] transaction time stamp that performs read operation.

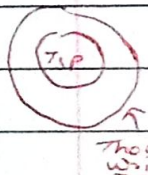
WTS : Denotes the highest [youngest] transaction time stamp that performs write operation.

* Data item time stamp :

T_i : Read	T_i : write
$TS(T_i) < WTS(Q)$	$TS(T_i) < RTS(Q)$ } NOT allowed.
NOT allowed & T_i rollback	$TS(T_i) < WTS(Q)$ } NOT allowed.

NOTE : Free from deadlock but starvation may occur

* Thomas write Rule :



T_i : Read (Q)	T_i : write (Q)
$TS(T_i) < WTS(Q)$ read operation reject & T_i Rollback	$TS(T_i) < WTS(Q)$ } write operation ignored & no rollback
	$TS(T_i) < RTS(Q)$ } Reject & T_i Rollback

Static ← Indexes → Dynamic
↳ B Tree & B+ Tree

Modification is costly
- Primary
- Clustering
- Secondary
- Multilevel
- ISAM

If rarely modification happens then use this.

↳ These are dynamic multilevel tree based indexing
↳ Modification friendly

BST, AVL Tree, Heap → In-memory DS. (i.e. MM) but not for disk

Parameters (Block ≡ Node)

(1) Order → Max no of child ptrs / tree ptrs / block ptr / node ptr.

Pointers → Block ptr BP
↳ Record ptr RP = BP + offset

Rules for nodes
1) Must be in increasing order
2) All leaves must be on same level
3) Space utilization Rule (≥ 50% utilization of order)

Root node : 2 BP to P BP
Non Root : $\lceil \frac{P}{2} \rceil$ to P BP -- Here P is the order

4) Node structure

with every value/key we put record ptr.

• <20 RP> • <40 RP> • <50 RP> •

Block ptr

Data ptr is By default Record ptr, but if not given then we can take Block ptr.

1) Non-leaf Node structure

• BP <SK, RP> BP <SK, RP> BP

2) leaf Node

Here we don't need BP, still place for BP will be there.

• <SK, RP> • <SK, RP> •

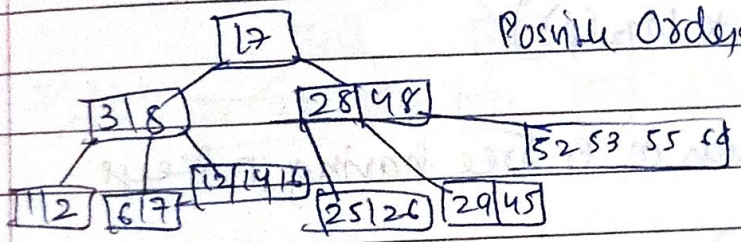
NULL ptr

* key = Data

① If a node has n BP then no of keys = n-1

② key = BP - 1

Q.



Possible Orders?

$\left\lceil \frac{p}{2} \right\rceil - 1 \leq \frac{\text{no of BPs}}{\text{Root node}} \leq p - 1 = \text{keys}$

Root node 1 to p-1

∴ Ans = 5, 6

∴ 4 ≤ p-1 keys

∴ p ≥ 5

$\left\lceil \frac{p}{2} \right\rceil - 1 \leq 2$

∴ p ≤ 6

Page No.
Date:

512
1024
4096 = 512 blocks
2

Q: Block size = 4096 B
Ptr size = 8B
Order of B Tree = ?
Int = 4B

$\frac{p}{2} - 1 \leq 512$ (p-1)

Max = $p(8) + (p-1)(4+8) \leq 4096$
Max no of block ptr = p
Max <key, RP> = p-1

BP <K, RP> BP

$\therefore 8p + 12p - 12 \leq 4096$

$p \leq \frac{4108}{20} = 205.4$

$\therefore p \leq 205.4 \Rightarrow$ max p = 205

Searching
★ worst case $\Rightarrow O(\text{Height} * O(p))$
(go to leaf node; within node)

Search within node :→ can do Binary search

★ Worst case $\Rightarrow O(\text{Height} * \log_2(p))$

Time complexity of search in B Tree having n keys:

$O(\log_{p/2} n * \log_2 p)$
- $\log_{p/2} n$ → max height
- $\log_2 p$ → within node Binary search

Search TC = $O(\log_{p/2} n * \log_2 p/2)$

= $O(\log_2 n)$

Insertion

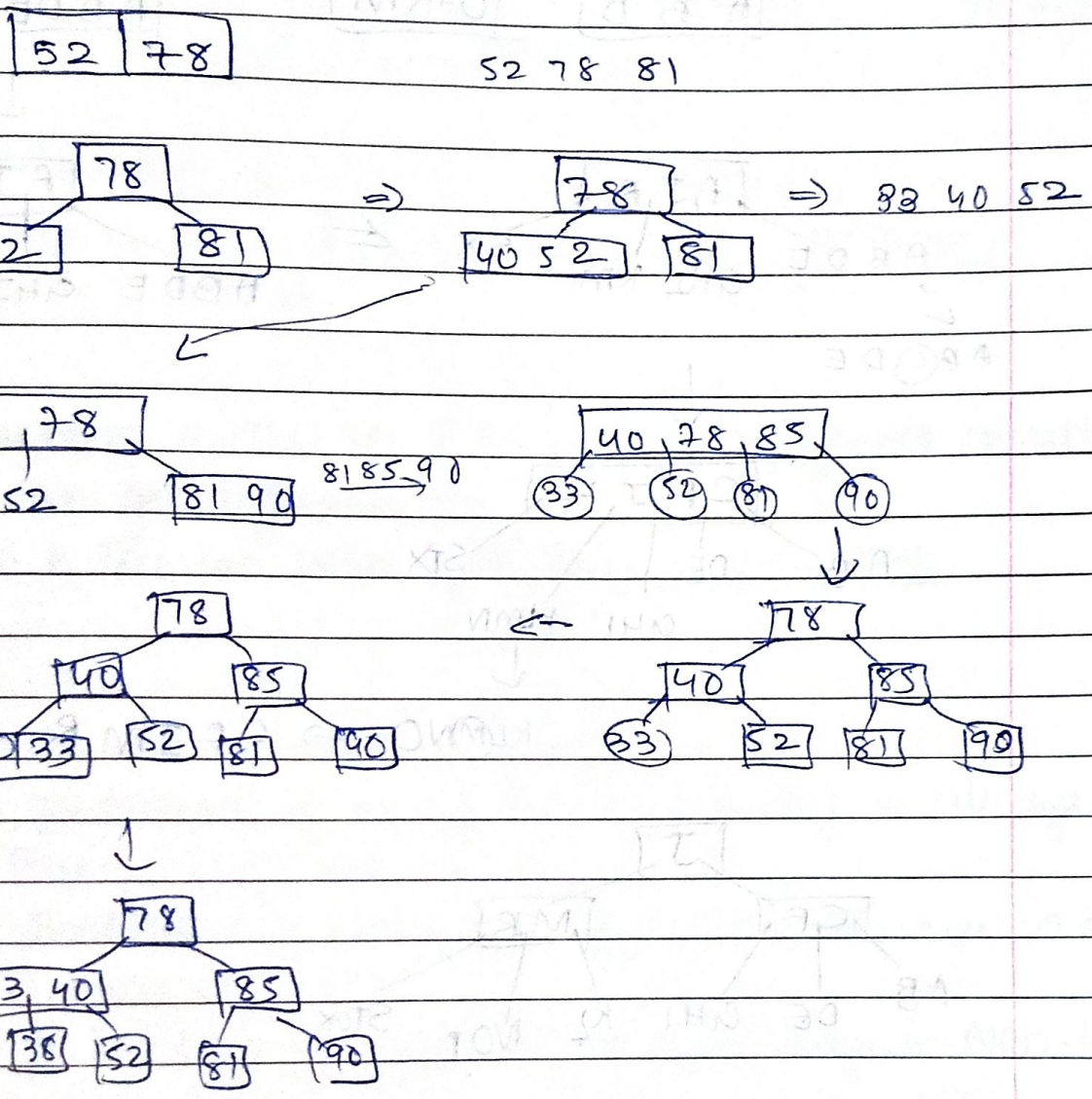
Insertion always happens on leaf node

① order = 3

78, 52, 81, 40, 33, 90, 85, 20, 38

Root = 1 to 2 key

other node = $\lceil \frac{p}{2} \rceil - 1$ to $(p-1)$ i.e 1 to 2



Order is Odd

A G F B K D H M I E S L R X C L N T U P

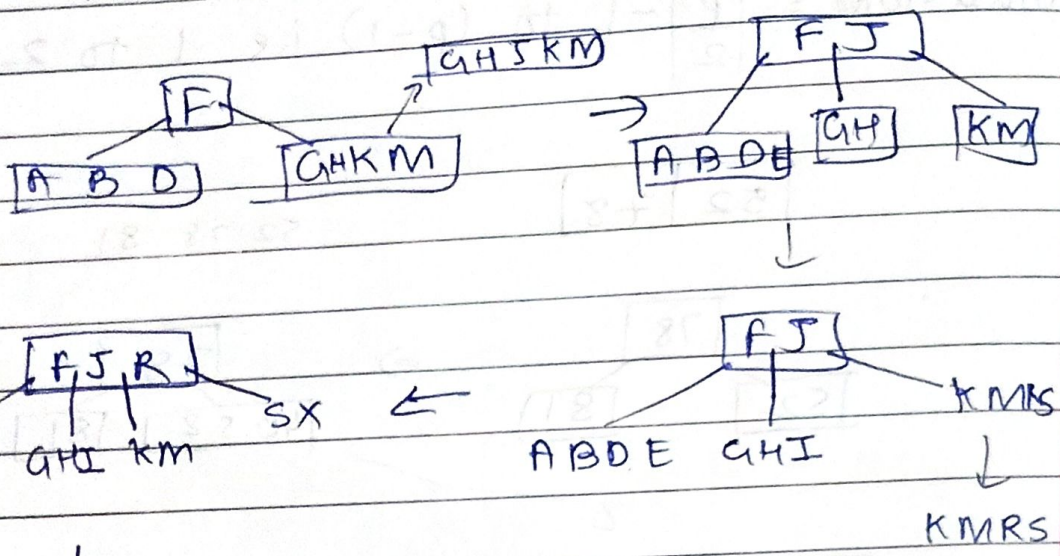
(2) A G F B K D H M I E S L R X C L N T U P

order = 5

∴ Root = 1 to 4 keys

Other node = 2 to 4 keys

A B F G (H)



* Always stick to left OR Right Biased But only one of them consistently.

Order is Even

Left Biased → More on left

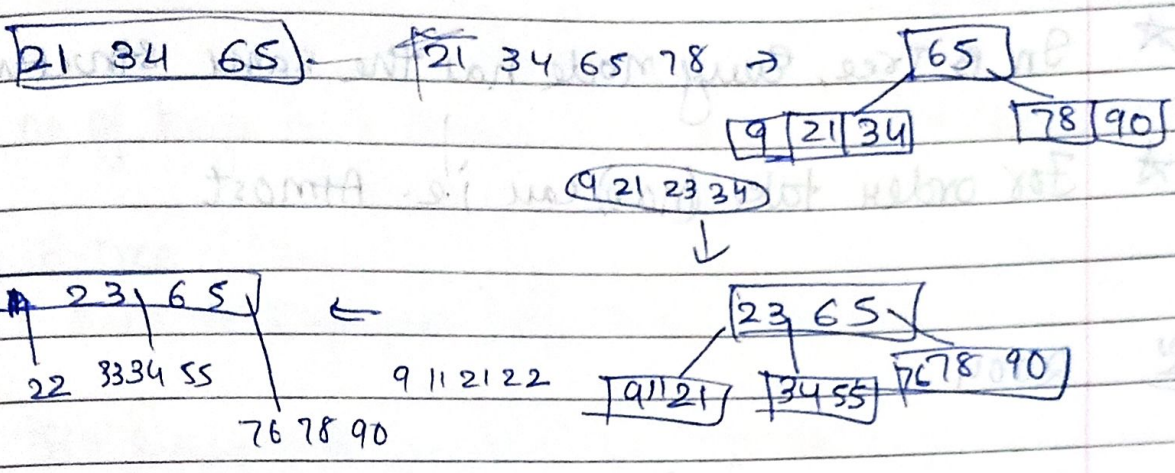
Order = 4

21, 34, 65, 78, 90, 09, 23, 11, 55, 76, 22, 33

Left-biased — 1st split → 2nd split → 3rd split

Root = 1 to 3

Non Root = 1 to 3



* Order of insertion matter in B Tree. With different insertion order we can get different tree.
* Insertion in B Tree can cause split from leaf to root

Q1 A single node will be called as leaf node

Q2 Immediate predecessor/successor of every key of non leaf is always in a leaf → True.

Q3 Immediate pred/succ of every key of leaf is always in a non leaf ⇒ False

Q4 Relationship betw no of leaf nodes & no of keys in non-leaf nodes??

→

No of leaf nodes = 1 + No of keys in non leaf nodes

* Find the order

p: max no of BP in a Node (Block)

$$P(BP) + (P-1)(Key + DP) \leq Block\ size$$

P	BP
P-1	Keys
P-1	DP

Disk block → RP or BP (B Tree node) → By Defaul

★ In B Tree, Every node has the same structure.

★ For order take (max) case i.e. Atmost

Q' 2004

Variation of B Tree

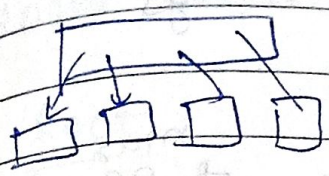
1) In B-Tree, All the nodes have the same structure

2) In leaf nodes, All tree ptrs are NULL
(But tree ptrs are present in the leaf node, even though they all are NULL.)

Variation:
Since leaf nodes require no ptr to children, they could conceivably store a different (larger) no of keys than interal nodes for the same disk page size.
Considering this variation of B tree, we can store more key in leaf as compared to internal node.

3) Subtract Block header from Block size.

Questions on B-Tree structure



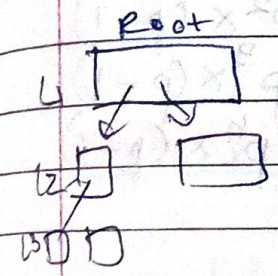
# Nodes	# keys	# BP
1	61	62
62	62×61	62×62
62×62	62×62×61	62×62×62

↳ All NULL ptrs

∴ Max no of keys in 3 levels = $61 + (62 \times 61) + (62 \times 62 \times 61)$

Q. Order of B-Tree : 62
3 levels; min # keys we can store?

IDEA : Fill every node as min as possible
∴ # Keys - $\left\lceil \frac{p}{2} \right\rceil - 1 = \left\lceil \frac{62}{2} \right\rceil - 1 = 30$ for Non Root
1 for Root



# Nodes	# Keys	# BP
1	1	2
2	2×30	2×31
2×31	2×31×30	2×31×31

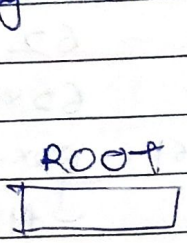
↳ All NULL

The variation

Bs = 2*B, Max no of BP per internal node = 62
Ar = 12B " " "keys keys" leaf node = 99
BH = 56B
D = 8B Max no of key per internal node = 61

Max # keys in Leaf node : y
$y(8+12) \leq 2048 - 56$
∴ $y = 99$

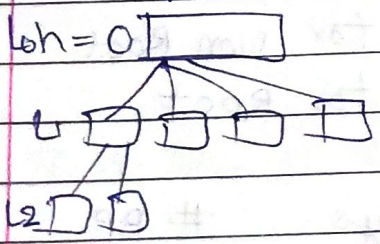
Root : 1 key to 61 keys
Non Root, Non leaf node : 30 keys to 61 keys
Leaf Node : $\left\lceil \frac{99}{2} \right\rceil = 50$ to 99 keys



# Node (max)	# keys (max)	# BP (max)
1	61	62
62	62×61	62×62
62×62	62×62×99	—

Question on Heights

Q:1 Given Height h; max # keys in B-Tree?



# node	# BP	# keys
1	p	p − 1
p	p × p	p × (p − 1)
p^2	$p^2 \times p$	$p^2 \times (p-1)$
p^3	$p^3 \times p$	$p^3 \times (p-1)$
p^h	$p^h \times p$	$p^h \times (p-1)$

① Height h; max # keys in BTree

$$(p-1) + p(p-1) + (p-1)p^2 + \cdots + (p-1)p^h$$
$$= (p-1)\left[1 + p + p^2 + \cdots p^h\right]$$
$$= (p-1)\left[\frac{p^{h+1}-1}{p-1}\right] = \boxed{p^{h+1}-1}$$

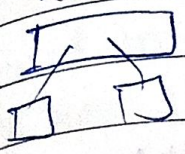
② Max # Nodes $= 1 + p + p^2 + \cdots p^h$
$$= \frac{(p^{h+1}-1)}{p-1}$$

③ Min # keys in BTree

levels = height + 1

* Height h; order p; min # keys? $\left\lceil \frac{p}{2} \right\rceil = t$

Root



# nodes	# BP	# keys
1	2	1
2	2 × t	2 × (t − 1)
2t	2t × t	2t × (t − 1)
$2t^2$	$2t^2 \times t$	$2t^2 \times (t-1)$
$2t^{h-1}$	$2t^{h-1} \times t$	$2t^{h-1} \times (t-1)$

Height h; order p: $t = \left\lceil \frac{p}{2} \right\rceil$

Min # keys = $1 + 2(t^h - 1)$

Min no of nodes $= 1 + 2[1 + t + \cdots t^{h-1}]$

$= 1 + 2\left(\frac{t^h - 1}{t - 1}\right)$

Max height: = (Min no of keys per node); n = # keys

$1 + 2(t^h - 1) = n$

where $t = \left\lceil \frac{p}{2} \right\rceil$ ← ceil

$t^h = \left(\frac{n-1}{2} + 1\right)$

$\therefore h = \left\lfloor \log_t \left(\frac{n-1}{2} + 1\right) \right\rfloor = \left\lfloor \log_t \left(\frac{n+1}{2}\right) \right\rfloor$

Min height: $h = \lceil \log_p (n+1) \rceil - 1$

↳ ceil

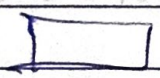
Conceptual based coming

keys = 400; p = 5
Root # keys = 1 to 4
Non Root = 2 to 4
Max no of levels :- fill nodes as less as possible.

	# Nodes	# BP	# Keys
4	1	2	1
	2	$2 \times 3^{\lceil p/2 \rceil}$	$2 \times 2 = 4$
	6	6×3	$6 \times 2 = 12$
	18	18×3	$18 \times 2 = 36$
	54	54×3	$54 \times 2 = 108$
	162	162×3	$162 \times 2 = 324$

∴ Max level = 5

Find min height = fill every node as much as possible



# Nodes	# BP	# Keys
1	5	4
5	5×5	5×4
25	25×5	25×4
~~125~~	~~125~~ $\times 5$	~~125~~ $\times 4$

till level 3 : # keys = 124 max
till level 4 : # keys = 624

keys we have : 400 ⇒ level 3 or (or max 124 keys we can put) level 4 ✓

B+ Tree

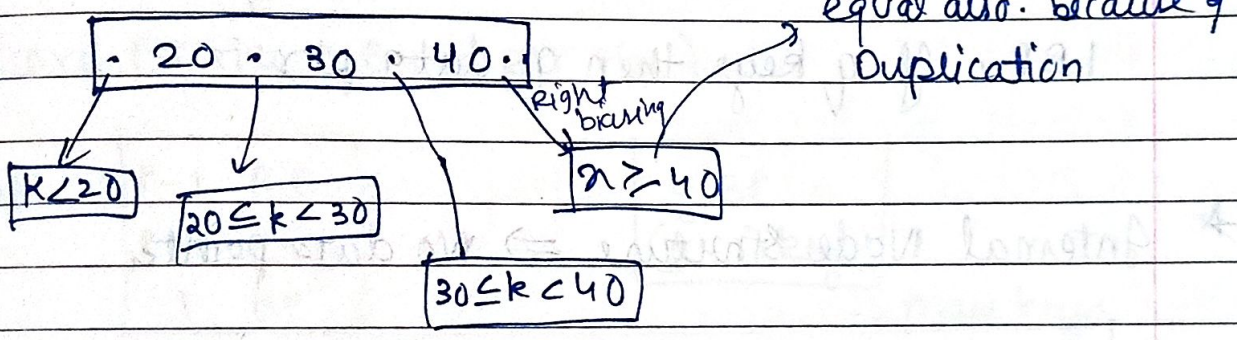
1) All the values (keys) will be present in leaf node
2) Some will be present in non-leaves node.
3) B+ tree has repetition, so in B+ Treee, Overall some value may be stored in more than one node.
4) In Non-leaf nodes, No Duplication happens
5) Every leaf node ptr to next leaf node (on Right)
6) All data ptrs → in leaf
7) No data ptr in Non leaf
8) Only tree ptrs in Non leaf
9) One Tree pt in every leaf

* # leaf Nodes = 1 + # keys in Non leaf nodes

Order of B+ Tree :- Max no of Total ptr per node.

B+ Tree Rules :-

1) Obvious search tree rules :-



$k<20$ $20<k\leq 30$ $30<k\leq 40$ $x\geq 40$ Left biasing

Order p : max # total ptr in a node

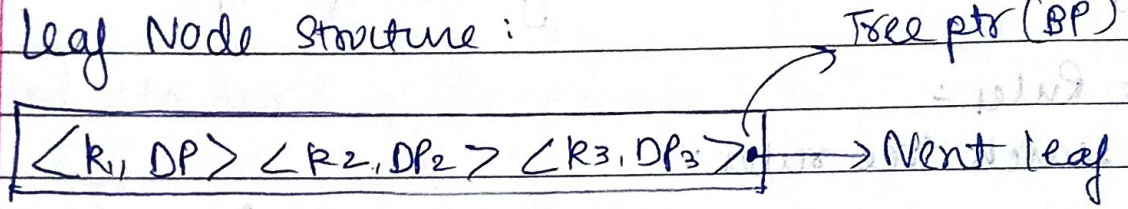
→ Root :- 2 BP to p BP
→ Non Root - Non leaf Node → $\lceil p/2 \rceil$ to p BP
→ leaf node : 1 Tree ptr means 1 BP

min
leaf : # keys : $\lceil \frac{p-1}{2} \rceil$ to $p-1$ keys

B+ Tree order p

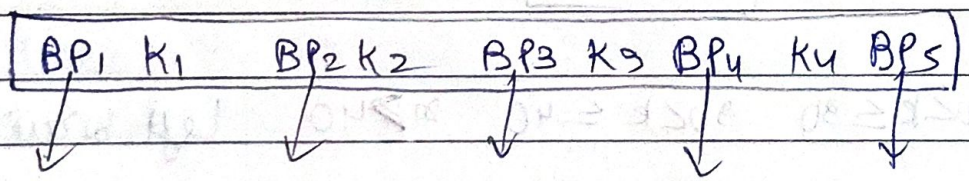
1) Root :	2 BP to p BP	1 key to p-1 keys
2) Leaf :	$\lceil \frac{p-1}{2} \rceil + 1$ to p Total ptr (↳ Tree ptr)	$\lceil \frac{p-1}{2} \rceil$ keys to p-1 keys
3) Remaining Internal Node :	$\lceil \frac{p}{2} \rceil$ to p BP	$\lceil \frac{p}{2} \rceil - 1$ keys to p-1 keys

★ Leaf Node Structure :



1 BP ; if q keys then q data ptr

★ Internal Node Structure ⇒ No data pointer



★ Root Node Structure
→ Leaf if only one Node
→ Internal if more than 1 level

① Leaf Node — order y

y − 1 keys
y − 1 DP
1 BP

$(y-1)(DP+K) + BP \leq$ Block size

② Internal node : order p

p − 1 Keys
p BP

$(p-1)(K) + P(BP) \leq$ Block size

By solving the above 2 eqn we will get the same ans.

$\because DP = BP$

① Note:- If DP is BP.

Leaf, Internal $\Rightarrow$ | p − 1 keys / p BP | $\Rightarrow$ order of leaf node = order of internal node

② If DP is NOT Block pointer:
↳ Record ptr

(Record ptr size > Block ptr size)

Leaf:

P − 1 RP
P − 1 Keys
1 BP

y BP
(y − 1) keys

$\therefore$ more keys

$\therefore P < y$

③ If B+ Tree has order P: means All nodes have order p & RP > BP then which blocks are not fully utilized?
→ Internal Blocks

Q) BSize = 4096 B,
RP = 9B
Key = 4B
BP = 8B
Order of leaf node = ?
Internal " = ?

Soln:- Assuming order = p
Leaf node = p(BP) + p(Key + RP) ≤ 4096
= 8 + (p)(4+9) ≤ 4096
= 8 + 13p − 13 ≤ 4096
= y ≤ 314.69 ∴ 314

Internal node = (p−1) key + p(BP) ≤ 4096
= (p−1)4 + p(8) ≤ 4096
= p = 341

Searching a B+ Tree

(1) For exact key values:
a) Start at the root
b) Proceed down, to the leaf

(2) For range queries:
1) As above
2) Then sequential traversal

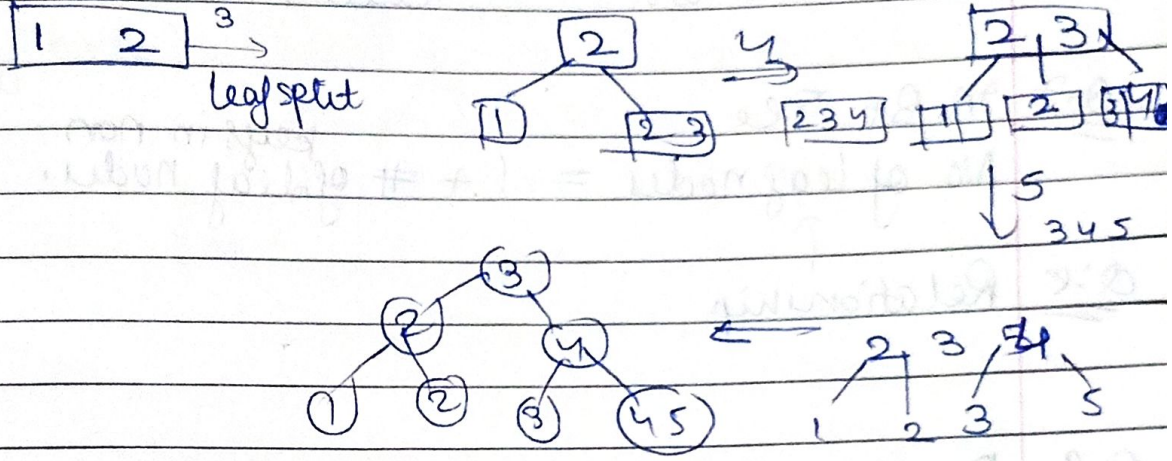
① Disk IO
② Index IO

B+ Tree Insertion

① Split of Internal node : same as B Tree
② Split of leaf Node : Different

Insertion always starts at leaf node

Order = 3
# Keys : 1 to 2
Leaf & Non Leaf both



Q1 Assume a B+ Tree has single node i.e. Only the root node.
→ Is this ~~Root~~ Leaf Node or Internal node
Ans Leaf Node.

Q2 Every value of leaf nodes is also present in non-leaf node in B+ tree → False

Q3 Some values of leaf nodes are also present in non-leaf nodes in B+ tree
→ True (B + Tree with atleast 2 nodes)

Q:4 Every value of non leaf nodes is also present in leaf nodes in B+ tree. → True

Q:5 Values in Non-leaf nodes are all unique in B+ tree.
→ True

Q:6 All searches (successful or unsuccessful) in B+ tree take nearly same amount of time (Index I/O cost)
→ True $\Theta(h)$
Index I/O cost = # levels in B+ Tree

Q:7 In B+ Tree
No of leaf nodes = 1 + # of keys in non leaf nodes.
↑
Q:8 Relationship

Q:9 B+ tree
Key = 9
BS = 512
RP = 7B
BP = 6B

An internal node can have up to p tree pointers & p-1 search field values; these must fit into a single block.

∴ Order of Internal node = ?

$$P(6) + (p-1)(9) \leq 512$$
$$p \leq \frac{521}{15} = 34.73 \quad \therefore \boxed{p = 34}$$

Order of leaf Node = Order y

$$(y-1)(\text{key} + RP) + 1\,BP \leq 512$$
$$(y-1)(9+7) + 6 \leq 512$$
$$y \leq \frac{502}{16} \quad \therefore \boxed{y \leq 32}$$

B+ Tree

Key = 16 B

RP = 6B

BP = 8B

BS = 1024B

Max values of order of internal node ? order of leaf nodes ?

Soln :-

Order of internal node = $(p-1)(16) + p(8) \leq 1024$

$= 43$

Order of leaf node = $(p-1)(16+6) + 8 \leq 1024$

$= 22p - 22 + 8 \leq 1024$

$\boxed{p = 46}$

B+ Tree

Key = 8B

BS = 512B

Index ptr = Node ptr = Block ptr = 4B

(No of ptrs per node) i.e order = ? DP = BP

Order = $(p-1)(\text{Key}) + p(BP) \leq 512$

$= (p-1)(8) + p(4) \leq 512$

$(p-1)(8+4) + 4 \leq 512$ $\Rightarrow \boxed{p = 43}$

B+ Tree, Height : h

Max # keys = ?

Max # keys : $L(I+1)^h$ → height
↳ max # keys per internal node
↑ max # keys per leaf node

Root	# Nodes	# BP	# keys
L0	1	I+1	I
L1	I+1	(I+1)(I+1)	(I+1)(I)
L2	$(I+1)^2$	$(I+1)^2(I+1)$	$(I+1)^2(I)$
⋮	⋮	⋮	
Lh	$(I+1)^h$	—	$(I+1)^h(L)$

Min # keys

	# Nodes	# BP	# keys
L0 (root)	1	2	1
L1	2	2×(i+1)	2×i
L2	2(i+1)	2(i+1)(i+1)	2(i+1)i
⋮			
Lh	$2(i+1)^{h-1}$	—	$2(i+1)^{h-1}(l)$

B+Tree : height h

min # keys = $2(i+1)^{h-1}(l)$
↳ min # keys per internal
↑ min # keys / leaf

Given keys (Given # keys)

Max Height = ?
Min Height = ?
} Use Bulk loading concept

Use → Bulk loading B+ Tree Concept (Bottom Up)

Q. Create a B+ tree of order 5 for the following set of keys with min height ?

92 24 6 7 11 8 22 4 5 16 19 20 78

# keys = 13 keys ⇒ Bulk loading
leaf = 2 to 4 keys

Bottom up Approach:

∵ min height → put max # keys per node

∴ $\left\lceil \frac{13}{4} \right\rceil$ = 4 nodes

4 BP
|
Level 1: 4 BP

Internal node : 3 to 5 ptr

∴ Height = 1 ; # levels = 2 # Nodes = 5

Max Height = ?
Fill every node as less as possible

$\left\lfloor \frac{13}{2} \right\rfloor$ = 6 leaf nodes

∴ level 1 BP = 6 2 Nodes
level 0 BP = 2 1 Nodes

Nodes = 6 + 2 + 1 = 9
levels = 3

Comparison of B & B+ Tree

	BTree	B+ Tree

(1) For the same system, same file:
i.e. Block size same, for n keys:

(1) Which will take less no of levels & less no of nodes?
→ B+ Tree
Coz we can put more child ptrs coz we save space of Record ptr.

(2) For n keys,
→ B Tree will take less coz B+ Tree is taking duplicates also.

(3) Range Query
→ B+ Tree better