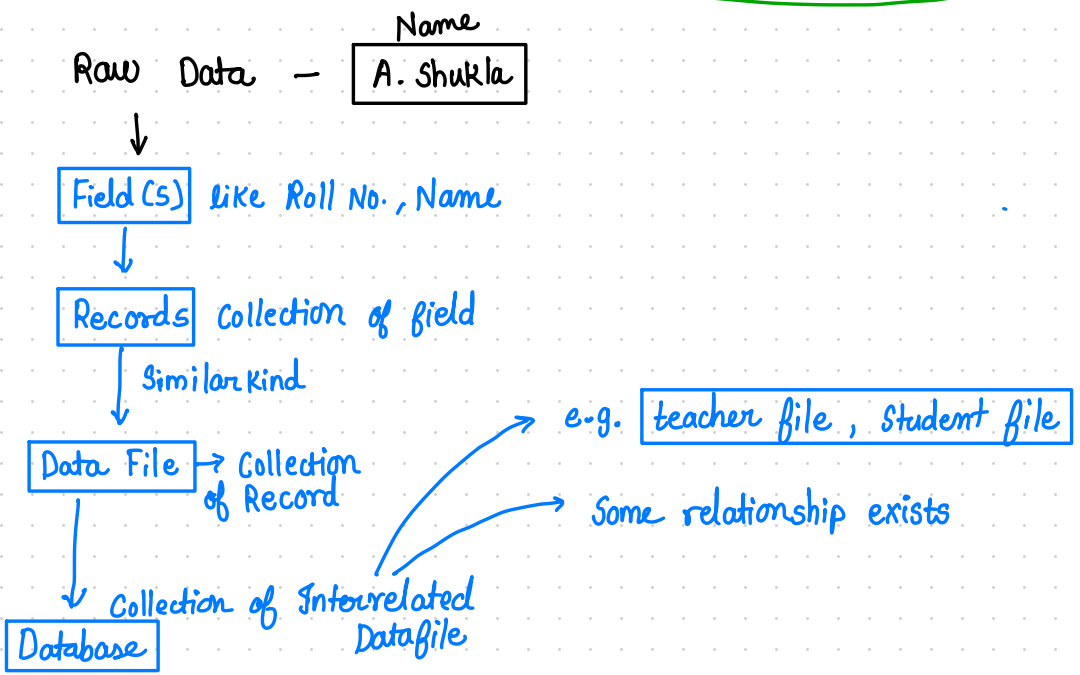
Books :

Principles of database - JD ullman

Database System Concepts - HF Korth & A Silber

Database Management Systems - Raghu Ramakrishnan



DBMS ⇒ Software (Allows us) can create Data & Manipulate

What are the advantage of creating File & Manipulation over HLL (File creation & Manipulation) ?

① Data Redundancy & Inconsistency

Roll No.	Name	HALL	Game Name	Fees

Possibility of Redundancy

(Roll No., Name, Hall) (G.Name, Fees) (Roll, G.Name)

Attributes repeated to make connections

→ When we remove redundancy, inconsistency also removed.

② Easy Access of Data

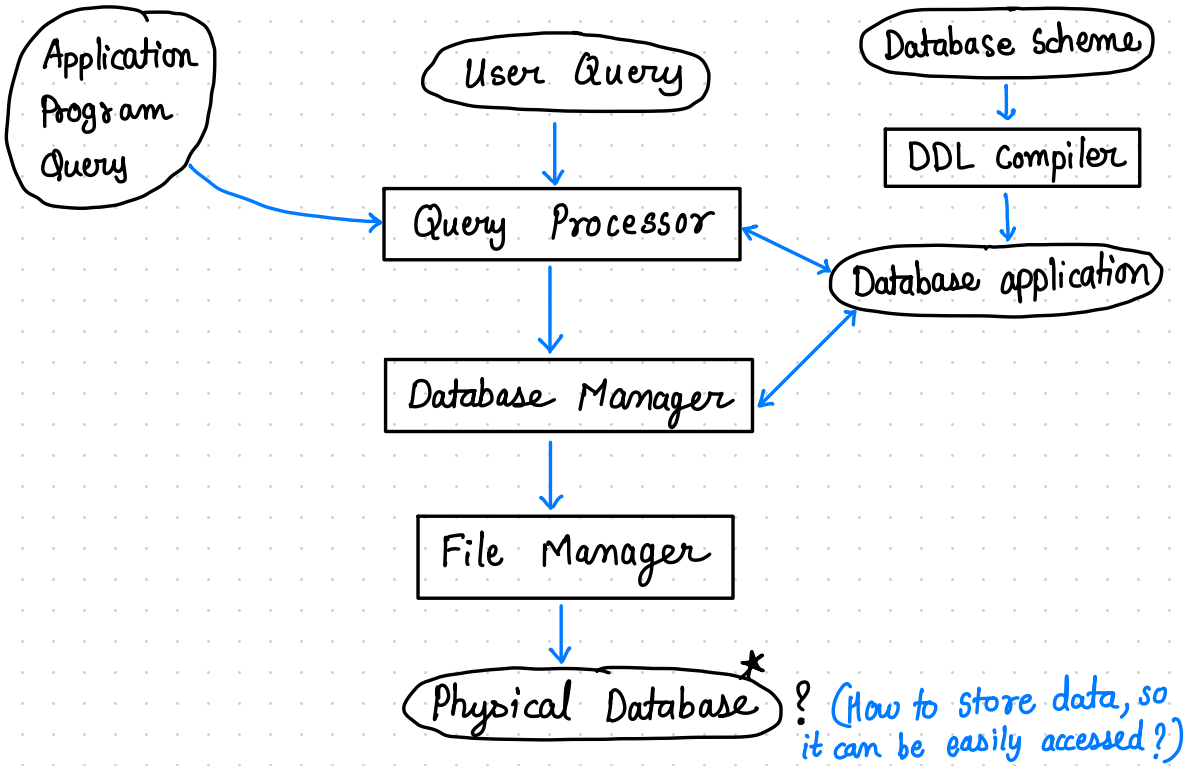
③ Multiple Users

④ Security Problems

⑤ Integrity Problems → like in Railway ___ System - Lower birth to Senior Citizen or Ladies

⑥ Synchronization
- → Multiple reader
- → But only one can manipulate at a time to avoid inconsistency.

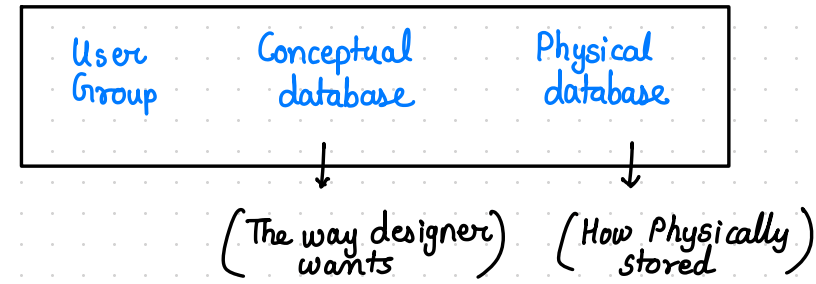
⑦ Recovery Process (To prevent Info Loss)



DDL - Data Definition Language

DML - Data Manipulation Language

* Levels of Abstraction :



Object Based Logical Models

- Entity Relationship Model
- Binary "
- Semantic "
- Infological "

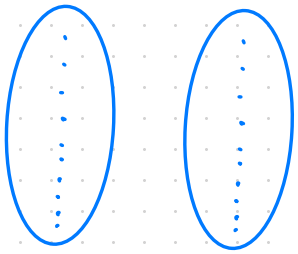
Record based logic Models

- Relational Model
- Network Model
- Hierarchical Model

Entity Relationship Model (E-R)

Entity collection → Entity Set

e.g. Customer (C.Name, P.No., C.City)
Employee (emp.name, p.No.)



- one-one
- One-many
- Many-one
- Many-many

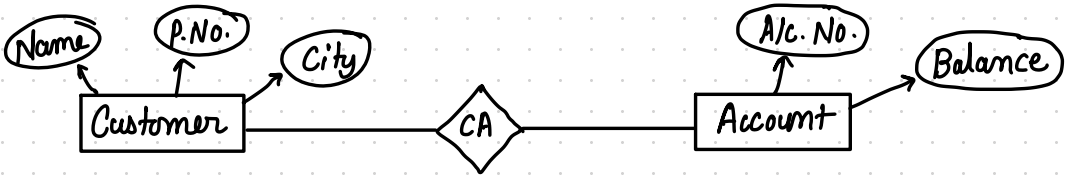
(field)

Primary Key → Collection of one or more than one attribute whose value will uniquely identify only one record.

or

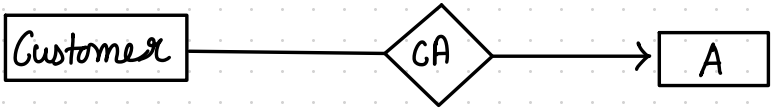
Minimum No. of attribute which uniquely identify.

Notations

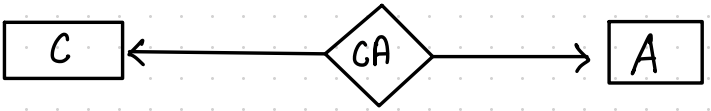


Many-Many (No arrow on both sides)

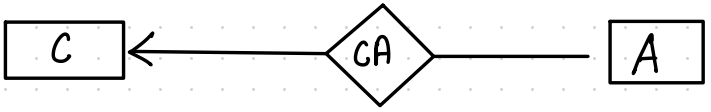
Customer have an account & an account may belong to multiple customer.



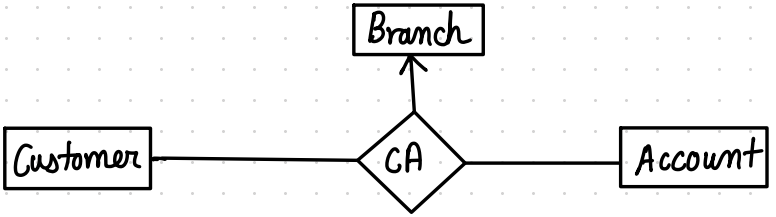
Many-one



One-one



One-many

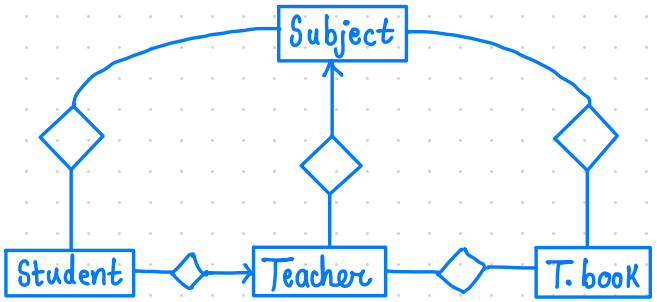


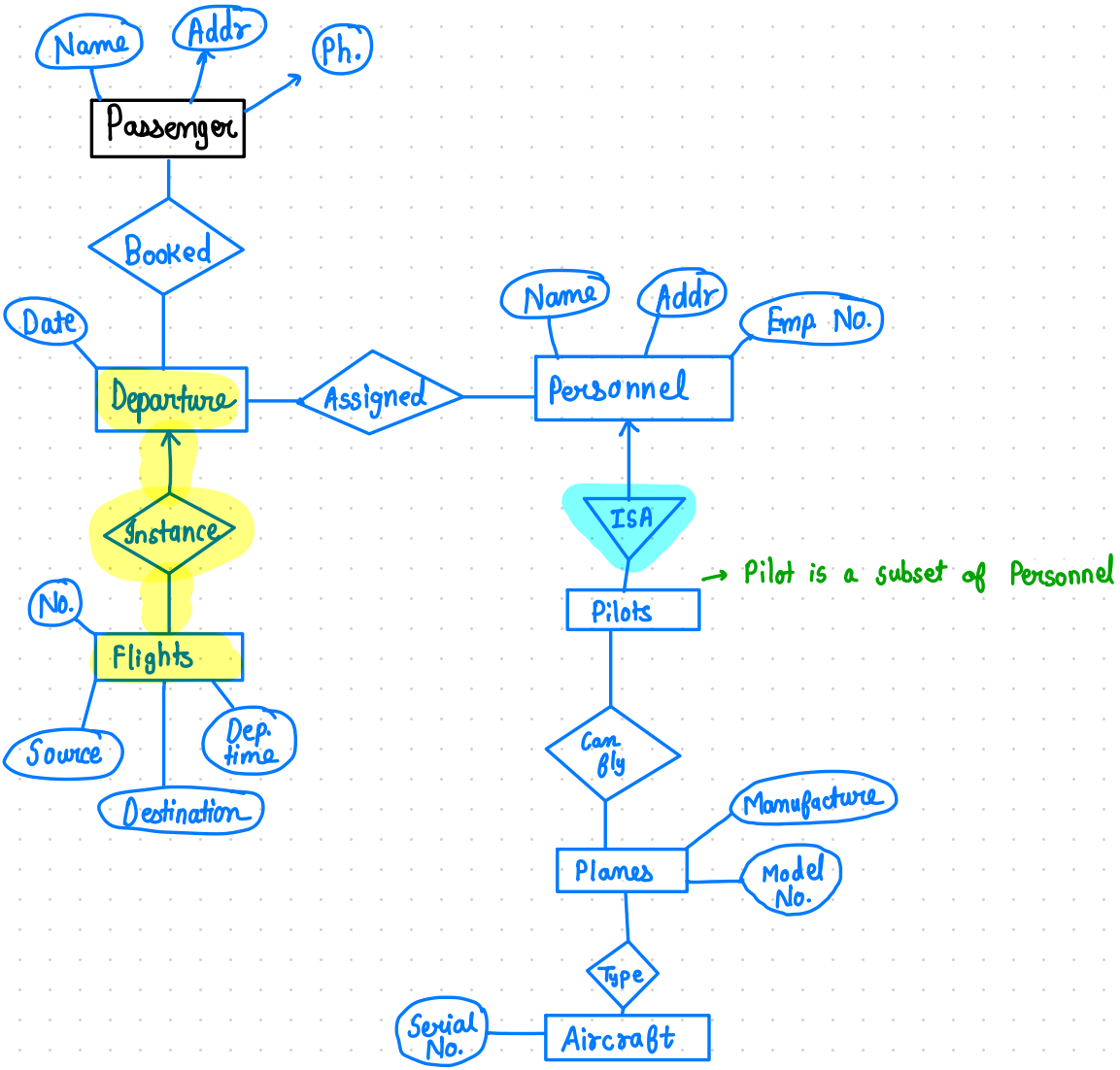
One customer may have multiple Acc., One acc. belong to many customer but in one branch.

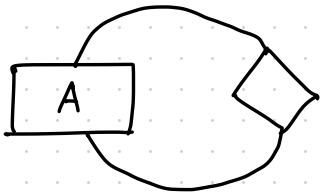
⇒ Draw one E-R diagram for Database System.
(Textbook, Subject, Student, Teacher)

① For each Subject each student is taught by a Single Teacher.
② Each teacher teaches only one subject.
③ Each Subject is taught by several teachers.
④ Teachers choose their respective text book.

Ans:







why we need to make a relationship ?

E1 1000 M1

⋮

M1 900 — —

E1 1000 M1 M1 900 - - -

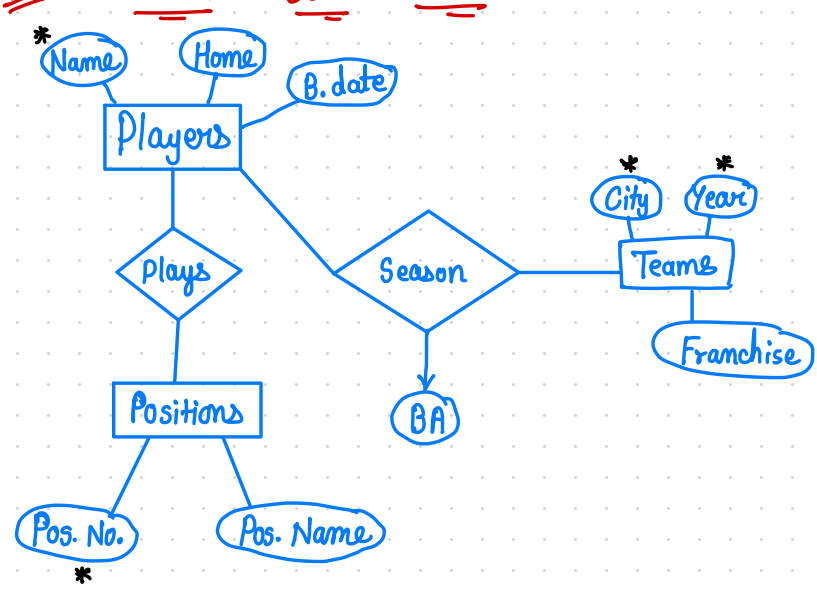
\- - - - -

Relational Data Model

Hierarchical " "

Network " "

★ Relational Data Model



Players (Name, Home, B.date)

Positions (Pos. No., Pos. Name)

Teams (City, Year, Franchise)

KEY of the Relationship

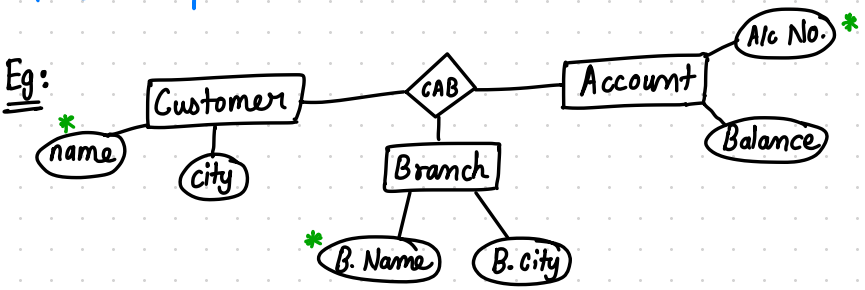
Plays (Name, Pos. No.)

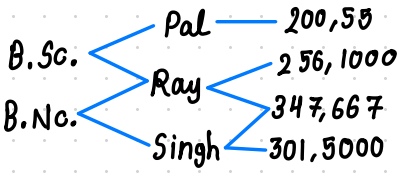
Season (Name, City, Year, BA)

Q. Player X, which position ?

⇒ Go to positions table

Eg:



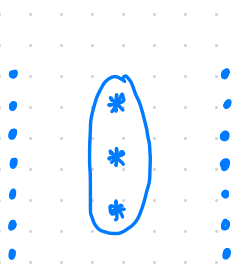


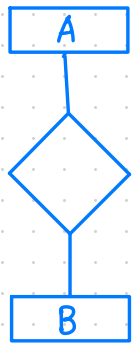
CAB

name	B. Name	Ac. No.
Pal	B.Sc.	200

Network Model

→ ER model where rel^n is restricted to binary & many-one





Create a Temporary set such that its many one with A & B.

Branch

B. name	B. City
B.Sc.	—
B. No.	—

Customer

C. Name	C. City
Pal	—
Ray	—
Singh	—

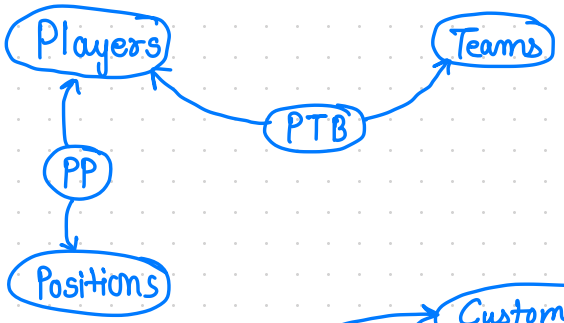
Account

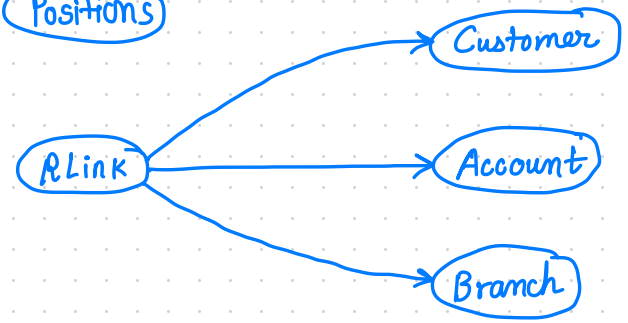
Ac. No.	Balance
200	—
256	—
347	—
301	—

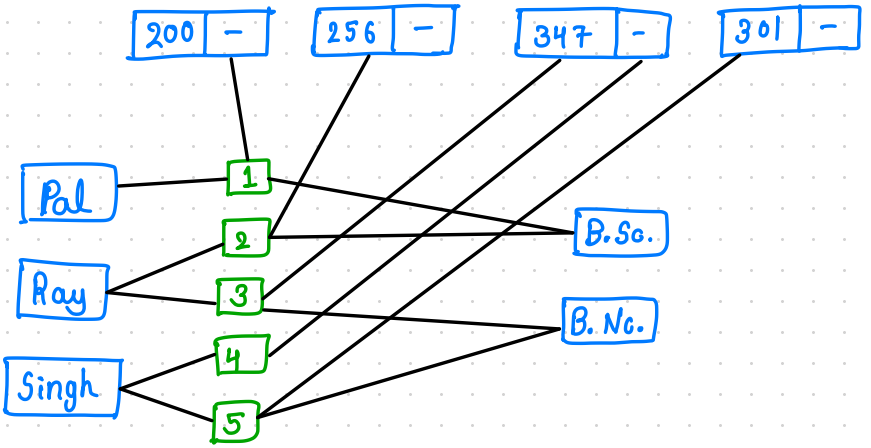
CAB

C. name	B. name	A/c No.
Pal	B Sc	200
Ray	B. Sc.	256
Ray	B. Nc.	347
Singh	B. Nc.	347
Singh	B. Nc.	301

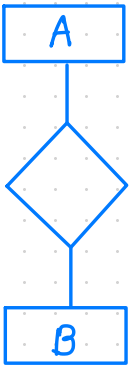
Example :

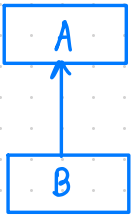

Players
Teams
PTB
PP
Positions

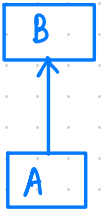

Customer
ALink
Account
Branch

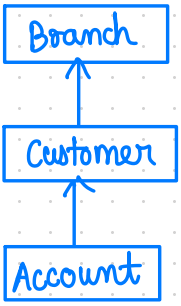

200
256
347
301
1
2
3
4
5
Pal
Ray
Singh
B.Sc.
B.Nc.

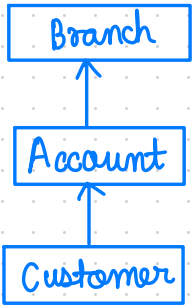
Hierarchical Model

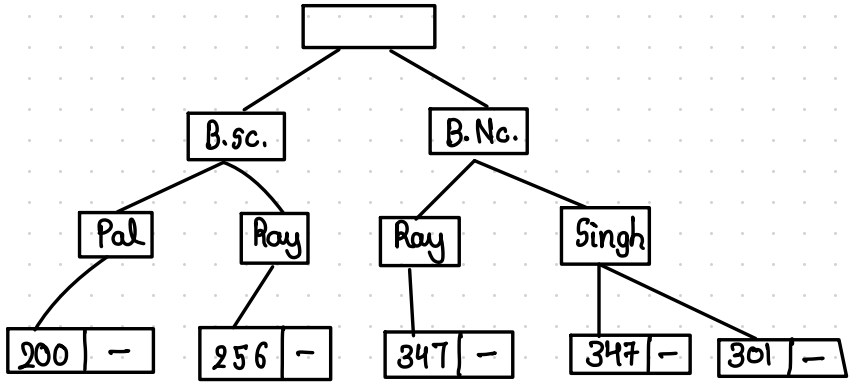


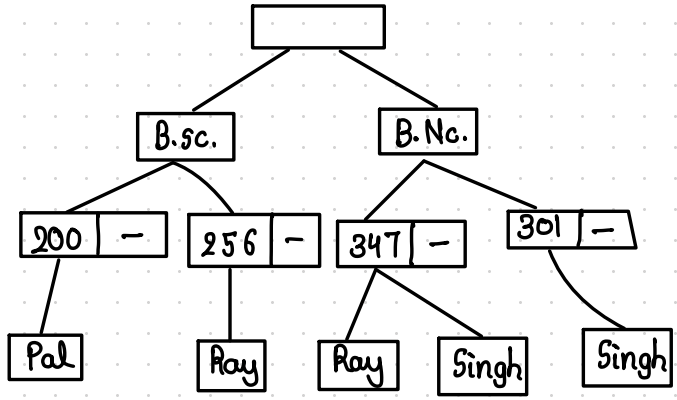












Do problems from book

⭐ Database Design :-

* Relational Database Design :-

→ Design is good if it is free from certain things:

1) Redundency
2) Updation Anamolies
3) Insertion Anamolies
4) Deletion Anamolies → only 1 student for hockey & he says NO, if you delete, your data is lost about fees of hockey.

S. Name	Roll. No. RN	HALL	GAME	FEE

(S.Name, RN, Hall)
(RN, Game)
(Game, Fee)
→ Redundancy in key only & Nowhere else.

Functional Dependency (FD)

U → Universal set of attributes
$U_j \subset U$: U_j is subset of U.
$X \rightarrow Y$; X & Y are subset of U_j

X functionaly determines Y
Y is functionaly dependent on X.

X = determinant
↓
Y = Dependent attribute

Eg: S_id → S.name (S_id ↓ student)

tuple = Record in database

Trivial FD.
$X \rightarrow Y$, where Y is a subset of X.
Eg: S_id → S_id
LHS ∩ RHS ≠ ∅
S_id Sname → S_id

Non-trivial : $X \cap Y = \emptyset$
S_id → Sname

Relation R : $X \rightarrow Y$ (X determines Y)

μ_1 & μ_2 (tuples)

	X	Y
$M_1 \rightarrow$	x	y
$\mu_2 \rightarrow$	x	y

If $\mu_1[x] = \mu_2[x] \Rightarrow \mu_1[y] = \mu_2[y]$

R(A,B,C,D) → find FDs.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_3	d_4

$A \rightarrow C$

$D \rightarrow B$ ✓ → If x val. is diff., there can be any y value.

$G \rightarrow A$

$AB \rightarrow D$

$B \rightarrow D$ ✗

$A \rightarrow A$ ✓ (Trivial FD)

$A \rightarrow B$, $B \rightarrow C$ $\overset{\text{logically}}{\Rightarrow}$ $A \rightarrow C$ (elements of a closure)

$\mu_1(a) = \mu_2(a) \rightarrow \mu_1(b) = \mu_2(b)$

& b b c c

∴ a a c c

⇝ Closure of a set of FDs

F F^+

$F^+ = \{X \rightarrow Y \mid F \Rightarrow X \rightarrow Y\}$

R(A,B,C) ; $F = \{A \rightarrow B, B \rightarrow C\}$

$F^+ = \{A \rightarrow C, A \rightarrow ABC, AB \rightarrow ABC, \ldots\}$

* Primary Key

→ Minimum No of attributes which determines one ____

$R(A_1, A_2, \ldots, A_n)$

$X \subseteq A_1, A_2, \ldots, A_n$

1) $X \to A_1, A_2, \ldots, A_n$ is in F^+

2) $Y \subseteq X$ & $Y \to A_1 A_2 \cdots A_n$ is in F^+

* Armstrong's Axioms :

1) Reflexivity : If $Y \subseteq X \subseteq U$, then $X \to Y$ (Trivial; Determing itself)

2) Augmentation : If $X \to Y$ and $Z \subseteq W$, then $XZ \to YZ$

(Augmenting Z on both side)

** 3) Transitivity : If $A \to B$, $B \to C$, then $A \to C$

$R(\text{CITY}, \overset{\text{Street}}{\text{ST}}, \text{PIN})$

$F = \{ \text{CITY ST} \to \text{PIN}, \ \text{PIN} \to \text{CITY} \}$

What are Key ? Tell logically using axioms :

Ans.

CITY ST → PIN

∴ CITY ST → PIN CITY ST (2nd ax. - Augmentation)

PIN → CITY, PIN ST → CITY ST

∴ PIN ST → CITY PIN ST

→ You may have more than 1 Key.

THUMB RULE

* → If an attribute is not in RHS of any FD then that attr. must be a part of Key.

* Inference Rules

1. Union Rule : $\{X \to Y, X \to Z\} \models X \to YZ$ (If, then)

   Logic : $X \to XZ$, $XZ \to YZ$

   $X \to YZ$

2. Pseudotransitivity Rule : $\{X \to Y, WY \to Z\} \models XW \to Z$

3. Decomposition Rule : If $X \to Y$ & $Z \subseteq Y$, then $X \to Z$

   OR: If $X \to YZ$ then $X \to Y$ & $X \to Z$

4. If $X \to YZ$, then $X \to Y$ & $X \to Z$

(3 & 4 same)

* Closure of a set of attributes :

F X^+ $X \to A$

LEMMA : $X \to Y$ follows from Arm's ax, iff $Y \subseteq X^+$

Theorem : Arm's ax are sound & complete. (You can't derive any extra FDs)

Book of Ulman → see proof of both above

* X^+ algorithm of Bernstein :

1. Let $N = 0$ and $X(N) = X$

2. If there is an FD : $A \to B$ in F
   whose LHS A is contained in $X(N)$
   but RHS B is not contained in $X(N)$
   then $X(N+1) = X(N) \cup B$
   otherwise terminate.

3. $N = N+1$, Go to step 2

$R = (A, B, C, D, E, G)$

$F = \{ AB \to C,\ C \to A,\ BC \to D,\ ACD \to B,\ D \to EG,\ BE \to C$
$CG \to BD,\ CE \to AG \}$

$(BD)^+ = ?$

Ans. $N = 0$; $X(N) = BD$

$D \to EG$; $X(1) = BDEG$

$BE \to C$; $X(2) = BDEGC$

$C \to A$; $X(3) = BDEGCA$

Eg: R (FI, MOC, SAL, SS, JOB, COR, LUCK, ED, EFF, SEN)

(Family Income, Make of Car, Salary, Spouse's salary, Job, City of Residence, Luck, Education, Effort, Sincerity)

F = { FI → MOC, SAL SS → FI, JOB FI → COR,
LUCK ED → JOB, JOB → ED, EFF SEN JOB → SAL }

(HINT : Thumb Rule)

Ans : (LUCK, SS, JOB, SEN, EFF)$^+$ → will give you all the attributes.

(LUCK, SS, ED, SEN, EFF)

Partial Dependency :

$X \to Y$ ↓ then elementary FD is present

$X' \subset X$ s.t. $X' \to Y$

$AB \to C$

$A \to C$

$F \rightsquigarrow f: X \to A$ $\quad B \in X$

X' ← $(X - \{B\}) \to A$ is in F^+

extraneous attribute

$X \to Y$ $\quad \{X - B\} \to Y$

$B \subseteq X$

Partial dependency

Eg: $AB \to C$, $A \to C$

B = extraneous attribute

$F = \{AB \to DEF, \underbrace{AC \to G}_{f}, A \to C\}$

⇒ $(A)^+ = ACG$ $\quad (AC - \{C\}) \to G$ ✓

$(AC)^+ = ACG$ $\quad$ ↓ extraneous attribute

$F' = \{AB \to DEF, A \to G, A \to C\}$

$\{AB \to DEF, A \to CG\}$

Redundant FD

f in (F) if $(F-f)^+ = F^+$

$f: X \to A$ } Redundant FD

⇝ A set of FDs F is Minimum if there is no set G with less FDs than F s.t. $G^+ = F^+$ [There is no redundant FD & it has least possible no. of FDs.]

Is this set minimum? $\{A \to B, A \to C\}$ (F ×) $\{A \to BC\}$ (G ✓ min.)

L-minimum: A set of FDs F is L-min. if

(L = Left-hand side)

1) F is minimum
2) There exist NO partial dependency between ____

Is it L-min?

$\{ABC \to D, A \to B\}$

→ 1st check is it minimum? → YES

B is redundant

⇒ $\{AC \to D, A \to B\}$

LR - Minimum: A set of FDs F is L-min. & also for each FD it is minimum on the RHS.

(L = LHS, R = RHS)

$A \to (AB)$ Not min.

R (A, B, C, D, E, F)

F = { AB → E, AC → F, AD → B, B → C, C → D }

Soln:

AB → E : NR (AB)$^+$ = ABCDF (E is not there) 1) check for minimum

AC → F : NR (AC)$^+$ = ACDBE → F is not there

AD → B : NR (AD)$^+$ = AD → B is not there

NR : Non-Redundant

No Redundancy

AB → E ; $A^+ = A$, $B^+ = BCD$

AC → F $C^+ = CD$

AD → B $D^+ = D$

No partia dependency

Can I reduce the No. of FDs ? What change we will make?

Eg: F = { AB → C, C → A, BC → D, ACD → B, D → EG, BE → C, CG → BD, CE → AG }

Find L-R Min.

⇒ (1) Rewrite the FD s.t. on the RHS of each FD there is only one attribute.

NR AB → C
NR C → A
NR BC → D
* R ACD → B
NR D → E
NR D → G

NR BE → C
NR CG → B
* R CG → D
NR CD → D
* R CE → A
NR CE → G

(2) check for minimum

1) Find Redundant FDs of this set.

ACD gives you ACDEGB ✓ Hence ACD → B is Redundant

Similarly, CG → D & CE → A

$AB \to C$	$BE \to C$
$C \to A$	$CG \to B$
$BC \to D$	$CD \to D$
$D \to E$	
$D \to G$	$CE \to G$

2) Check for each FD whether ∃ partial dependency or not.

$AB \to C \Rightarrow$ find A^+ & B^+

$A^+ \downarrow A$ $\quad$ $B^+ \downarrow B$

No partial dependency ✓

3) Is it minimum or ∃ can reduce it ?

→ If for any 2 FDs if LHS is same, merge the RHS.

→ Each FD represents one table, we will have to carry less data.

Decomposition of a Relational Scheme :

$R = (A_1, A_2, \dots, A_n)$

$P = \{R_1, R_2, \dots, R_k\}$ → k subsets

$R = R_1 \cup R_2 \cup \dots \cup R_k$

It must satisfy 2 properties when we decompose :

1) Lossless Join property → else ∃ will lose information when we join them.

$R : R_1, R_2, \dots, R_k$

D is a set of dependencies

All the tuples of Relation R $\quad r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \dots \dots \bowtie \Pi_{R_k}(r)$

$\bowtie$ ↓ Natural Joint

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

$R(A,B,C)$

$\Pi_R(A,C)$

A	C
a_1	c_1
a_2	c_2
a_3	c_3

Theorem : If $\rho = (R_1, R_2)$ is a decomposition of R & F is a set of dependencies then ρ is lossless join decomposition w.r.t F iff.

$(R_1 \cap R_2) \rightarrow (R_1 - R_2)$

or

$(R_1 \cap R_2) \rightarrow (R_2 - R_1)$

This FD exists in F or F^+.

Proof in Ulman

⇒ Their intersection determines their subtraction.

Example : $R(A,B,C)$ $\quad F = \{A \rightarrow B\}$

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

Does it satisfy lossless join property ?

$R_1 \cap R_2 = A$

$R_1 - R_2 = B$ $\quad (A \rightarrow B) \Rightarrow$ can be in F or F^+

YES

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

NO

$r = \{a_1 b_1 c_1 , a_2 b_1 c_2\}$

$\Pi_{AB}(r) = \{a_1 b_1 , a_2 b_1\}$

$\Pi_{BC}(r) = \{b_1 c_1 , b_1 c_2\}$

$\Pi_{AB}(r) \bowtie \Pi_{BC}(r) = \{a_1 b_1 c_1 , a_1 b_1 c_2 , a_2 b_1 c_1 , a_2 b_1 c_2\}$

should be equal to r, but its NOT.

2) Preservation of Dependencies :

$R \Rightarrow R_1, R_2, \ldots, R_k$

F → set of dependencies

Union $\leftarrow \bigcup_{i=1}^{k} \pi_{R_i}(F) = F$

dep. which exists in F where attributes of Ri are involved.

→ we decomposed into k - sub-relations s.t. when we join them we get original Relation.

→ If I don't preserve some FDs while decomposing WHAT WILL HAPPEN ?

I may not be able to answer all queries.

BUT

→ If loss-less join is not there, our answer to query will be wrong. So, its more imp. than preserving property.

Example : $R(C, S, P)$

$F = \{ CS \rightarrow P,\ P \rightarrow C \}$

$R_1(S, P)$ $R_2(C, P)$

⇒ 1) Is it satisfying loss-less join property ?

YES

2) Is it satisfying preservation of dependency ?

$\pi_{R_1}(F) = \{ S \rightarrow S,\ P \rightarrow P \}$

$\pi_{R_2}(F) = \{ P \rightarrow C \}$ → does not preserve dependency.

Example :

$R(A, B, C, D)$ $\quad F = \{A \to B,\ C \to D\}$

$R_1(A, B)$ $\quad R_2(C, D)$

$\Rightarrow$ Not loss-less

$\Rightarrow$ Preserve dependency ?

$\Pi_{R_1}(F) = \{A \to B\}$
$\Pi_{R_2}(F) = \{C \to D\}$
} preserve dependency ✓

1. Candidate Key : → minimum, its closure can give all attri.
→ Anything extra added, then it becomes Super Key.

These are individual columns in a table that qualifies for the uniqueness of all rows

→ Prime attri. = that which makes the candidate keys.
Non-prime attri = remaining others.

2. Primary Key or KEY : RN

Is the columns you choose to maintain uniqueness in a table.

3. Foreign KEY : RN, Hall

is a Collection of field(s) in one table that uniquely identifies a Row of another table.

primary KEY of table T2 but it is there in table T1 also.

4) Super Key : RN, S. Name

If you add any other column or attribute in a primary key then it will become a super key.

★ Normal Forms :

(Normalisation) is the process by which when you decompose it, that will be free from anamolies.

★ Boyce Codd Normal Form (BCNF)

R F : X → Y

→ If Y is not a subset of X and if X is a Key (Not just a member but whole KEY) then X → Y is in BCNF.

→ If all FDs satisfy this property then Relation R is in BCNF.

Eg: R (PAN, PI, DI, DRUG, QTY, COST)

F = { PAN → PI, PI → DI, PI DRUG → QTY, DRUG QTY → COST }

Is it BCNF ?

⇒ KEY : { PAN, DRUG } , PAN → PI is not in BCNF

Q. How to decompose relation R into subrelation s.t. each sub-rel^n is in BCNF ?

① R_1 (PAN → PI) , R_2 (PAN, DI, DRUG, QTY, COST)

R_1: sub. rel^n ; R_2: sub rel^n ; R_2 ← all remaining attributes except RHS of R_1 (PI)

FDs available in F or F^+ :

Not a KEY (DRUG QTY → COST

Find out FDs involved in attributes of R_2 only, those may be in F or F^+.

→ Decompose R_2 in sub-rel^n :

② A_3 (DRUG, QTY, COST) A_4 (PAN, DI, DRUG, QTY)

⇓ possible FD

PAN → DI } it is in F^+, not in F, we can use.

③ R_5 (PAN, DI)

④ R_6 (PAN, DRUG, QTY) → contains KEY → you can find out any attribute that satisfies the loss-less join property.

PAN DRUG → QTY

decomposition of A is R_1, R_3, R_5 & R_6

Theorem: BCNF decomposition produces loss-less join decomposition always.

→ we have lost PI → DI ⇒ BCNF may not preserve dependencies.

Example: {C → A, AE → B, BF → C, CD → EF, EF → AD}

R (A, B, C, D, E, F)

CD → KEY

{C, A} ✓

{B, C, D, E, F}

{B F C} ✓

{B F D E}

BF DE

BF CDE A

{B, D, E} ✓

{B, F, D} Also KEY

BFD → EF

BD → E

BF → C CD → EF EF → AD

AE

★ Prime Attribute

An attribute $A \Rightarrow$ prime if A is a member of a key of R, otherwise we call it as Non-prime attribute.

$R(A,B,C,D)$, $F = \{ AB \to C, B \to D, BC \to A \}$

$\Rightarrow$ Key : AB, BC

★ Third Normal Form : (3NF)

If : R $X \to A$: A is not in X

then either X is a key of R or A is prime attribute (i.e. in the key)

BCNF > 3NF

★ Bernstein's 3NF Algorithm

Input : A set of attributes

Output: decomposed sub-relation which is free from anomalies

1) Rewrite the FDs s.t. there is only 1 attribute on the RHS of each FD.
2) Rewrite the list of FDs s.t. $\nexists$ no redundant FD.
3) Rewrite the FDs s.t. No proper subset of LHS functionally determines RHS. (Removing partial dependency)
4) Combine the dependencies with same LHS using the Union Rule.

→ then if $X \to Y$ is in the list, make a decomposition with attributes $\{X, Y\}$ ⇒ This will ensure preservation of dependencies.

5) Calculate the keys of the original relation.
If NO key is included in any of the decomposed sub-relation then create a new sub-relation with the attributes of key only. ⇒ for the loss-less join property.

6) If any of the sub-relation is a sub-set of the other, remove the smaller sub-relation.

Example:

A (PAN, PI, DI, DRUG, QTY, COST)

F = { PAN → PI, PI → DI, PI DRUG → QTY, DRUG QTY → COST }

⇒

1) ✓
2) ✓
3) ✓
4) decomposed sub-relation:

R_1 (PAN, PI) R_2 (PI → DI)

R_3 (PI, DRUG, QTY)

R_4 (DRUG, QTY, COST), R_5 (PAN, DRUG)

Ex: $R(A,B,C,D,E,F,G,H,I)$

$F = \{A \to BCD,\ AE \to FG,\ F \to AEG,\ C \to HI\}$

$\Rightarrow$ 3NF decomposition ?

$\Longrightarrow$ 1) $A \to B$, $A \to C$, $A \to D$, $AE \to F$, $AE \to G$, $F \to A$, $F \to E$, $F \to G$, $C \to H$, $C \to I$

2) $A \to B$, $A \to C$, $A \to D$, $AE \to F$, $F \to A$, $F \to E$, $F \to G$, $C \to H$, $C \to I$

3) ✓

4) $A \to BCD$, $AE \to F$, $F \to AEG$, $C \to HI$

5) KEY of original :

$R_1(A,B,C,D)$, $R_2(A,E,F)$ ✗

$R_3(C,H,I)$, $R_3(A,E,F,G)$

$A \to B$, $A \to C$, $A \to D$, $AE \to F$, $AE \to G$, $F \to A$, $F \to E$, $F \to G$, $C \to H$, $C \to I$

3] No partial dependency.

$A \to BCD$, $AE \to F$, $F \to AEG$, $C \to HI$

$\Rightarrow R_1(A,B,C,D)$, $R_2(A,E,F)$

$R_3(A,E,F,G)$, $R_4(C,H,I)$

$\Rightarrow R_1(A,B,C,D)$, $R_3(A,E,F,G)$, $R_4(C,H,I)$

1 NF

2 NF - No partial dependency but transitive dependency.

★ Multivalued dependency :

MATH	T1	B1
	T2	B2
	T3	
DSA	T11	D1
	T12	D2
		D3

u	MATH 1	T1	B1
t	MATH 1	T1	B2
v	MATH 1	T2	B1
s	MATH 1	T2	B2

(c, t_1, x_1) , (c, t_2, x_2) appears

then (c, t_1, x_2) , (c, t_2, x_1) will also appear.

$X \rightarrow\rightarrow Y$ holds in R

if I have 2 tuples : t, s : $t[X] = s[X]$

then you must obtain 2 more tuples : u, v s.t. :

1) $u[X] = v[X] = t[X] = s[X]$

2) $u[Y] = t[Y]$ and $u[R-X-Y] = s[R-X-Y]$

3) $v[Y] = s[Y]$ and $v[R-X-Y] = t[R-X-Y]$

X	Y	R-X-Y
t[X]	t[Y]	t[R-X-Y]
s[X]	s[Y]	s[R-X-Y]
u[X]	u[Y]	u[R-X-Y]
v[X]	v[Y]	v[R-X-Y]

There is a Multivalued dependency between X & Y.

$R(X, Y, Z)$

If $X \to\to Y$ then $X \to\to Z$

$\{X \to\to Y,\ Y \to\to Z\} \models X \to\to (Z - Y)$

$\{X \to Y\} \models X \to\to Y$

★ 4th Normal Form :

R is a rel^n with a set of dependencies D.

R is in 4NF if there is an MDF of form $X \to\to Y$ where Y is not contained in X & XY does not include all the attributes of R.

and then X is a key of R.

$X \to\to Y$

$XY \quad Y \not\subseteq X$

X is a key of R

R, D

$R(X, Y, Z)$

Eg: $R(A, B, C, D, E, F, G)$

$D = \{A \to\to B,\ B \to\to G,\ B \to\to EF,\ CD \to E\}$

$\Rightarrow$

KEY : ACD

$A \to\to B$ not in 4NF.

✓ $R_1(A, B)$, $R_2(A, C, D, E, F, G)$

$CD \to E$ (Not in 4NF)

✓ $R_3(C, D, E)$, $R_4(A, C, D, F, G)$

$A \to\to G$

✓ $R_5(A, G)$, $R_6(A, C, D, F)$

$A \to\to F$

✓ $R_7(A, F)$, ✓ $R_8(A, C, D)$

KEY

↓

stop here

★ Relational Algebra

⇒ Operations :

1.) **Union(U)**: Set of tuples t which will be either in R or in S or in both

R U S

A	B	C
a	b	c
d	a	f
c	b	f

R

D	E	F
b	g	a
d	a	f

S

No heading

a	b	c
d	a	f
c	b	f

RUS

} first write R, then see in S what's not here, add that

2.) **Set difference (−)**: Set of tuples t which will be in R but not in S.

R − S

a	b	c
c	b	d

R − S

3.) Cartesian product (x) : $R \Rightarrow K_1$ no. of attributes (R^{K_1})

& $S \Rightarrow K_2$ (S^{K_2})

R x S

↓

$(K_1 + K_2)$ attributes

↪ first K_1 (A, B, C) components will form a tuples of R

& last K_2 (D, E, F) " " " " " " S.

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

- Take first tuple of R, join with all tuples of S.
- Take next tuple.

Number of tuples : $K_1 * K_2$

4.) Projection (π) :

$\pi_{i_1, i_2, \ldots, i_m}(R)$

↓

Integer value (col. No.) or attribute name

$\pi_{C,A}(R)$

C	A
c	a
f	d
b	c

→ Taking out from each tuple, only the attribute values in $(i_1, i_2, \ldots i_m)$

$\pi_{2,3}(S)$

E	F
g	a
a	f

5.) Selection (σ) : F is a formula $\longrightarrow$ operands

- $\Rightarrow$ attri names
- $\Rightarrow$ column names etc.
- $\Rightarrow$ Constant value

Arithmetic operators
Comparative operators
Logical operators

V ∧ ¬ < = >

$\sigma_F(R)$ } from rel^n R we will take the tuples which will satisfy the formula F.

Eg: $\sigma_{2>3}(R)$

$\sigma_{1='s' \vee 2='p'}(R)$

$\sigma_{B='b'}(R)$

A	B	C
a	b	c
c	b	d

Additional operators :

1.) Intersection ($\cap$) : Set of tuples which are in R as well as in S.

$R \cap S = R-(R-S)$

d	a	f

2.) Quotient ($\div$) : R $\longrightarrow$ No. of attributes r

S $\longrightarrow$ " s

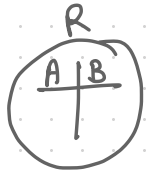
$R^{(r)}, S^{(s)}$ $\quad r > s, \; s \neq \phi$

$R \div S \Rightarrow$ set of tuples with attri $(r-s)$ such that for all tuples u in S, tu is in R

R(A,B), S(B) t is tuple of $R \div S$

$R \div S = \pi_A(R) - \pi_A(\pi_A(R) \times S - R)$

All comb^n of AB



Eg:

R

a	b	c	d
a	b	e	f
b	c	e	f
e ← c	d	c	d
e	d	e	f
a	b	d	e

S

c	d
e	f

No. of attri : $r - s = 4 - 2 = 2$

$R \div S$

a	b
e	d

b c won't come because bcef is there but bccd is not there

If c replaced by e,
Now ed can also come.

3.) θ-join :

$R \bowtie_{i \theta j} S$ → i is in some relation with j.

$\sigma_{(i \theta (r+j))}^{R \times S}$

R

A	B	C
1	2	3
4	5	6
7	8	9

S

D	E
3	1
6	2

$R \underset{B<D}{\bowtie} S$ } No. of columns = 5 (3+2)

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

4.) **Natural Join :** 1.) R × S

2.) for each attri. A which appears in both R & S, select those tuples from R × S where,

R.A = S.A

3.) Remove S.A

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

→ 2 attri same name. I have to remove.

∴ Resultant table will have 4 columns. A, B, C, D

A	R.B	R.C	S.B	S.C	D
	✓	✓	✓	✓	

R × S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Ex: Employee (Emp. id., Name, Salary, M-id)

M-id = 1 for Manager
= 0 otherwise

→ find out name of the employee whose salary is more than any manager (may not be All).

⇒ $\Pi_2 (\sigma_{4 \neq '1' \wedge 8 = '1' \wedge 3 > 7} (\text{Employee} \times \text{Employee}))$

Ex: Customer (C.name, street, C.city)

Deposit (B.name, ac. No., C.name, balance)

Borrow (B.name, loan no., c.name, amount)

Branch (B.name*, B.city, Asset)

Client (C.name*, emp.name*)

* = KEYS

(1) → find the C.names such that C.name & the emp. who handles that customer, his name is same.

$$\Pi_1 (\sigma_{1=2} (\text{client}))$$

(2) → find all clients of emp. 'X Y Z' at corresponding cities of those clients.

$$\Pi_{1,5} \left(\sigma_{2='XYZ' \wedge 1=3} (\text{Client} \times \text{Customer}) \right)$$

↓ If this condition was not there, we could have used Natural Join.

(3) → find all customers who have both account and loan at 'KGP' branch.

$$\Pi_3 \left(\sigma_{3=7 \wedge 1=5 \wedge 1='KGP'} (\text{Borrow} \times \text{Deposit}) \right)$$

OR

$3=7 \wedge 1='KGP' \wedge 5='KGP'$ (Deposit × Borrow)

OR (Natural Join)

$$\Pi_3 (\sigma_{1='Kapi'} (Borrow \bowtie Deposit))$$

(4) find all customers who have accounts in all the branches in the city 'KOLKATA'.

→ All customers & their branch where they have account

$$\Pi_{C.name,\ B.name} (Deposit) \div \Pi_{B.name} (\sigma_{B\text{-}city='KOL'} (Branch))$$

→ All the Braches in Kolkata.

Eg: Lives (p.name, street, city) — person

Works (p.name, c.name, Salary) — company

Located in (c.name, c.city)

Manages (p.name, M.name) — Manager

Do of your own.

(1) find the name & city for all employees who work for 'FCI' company.

(2) find the name, street & city of all employees who work for 'FCI' comp. & whose salary is more the 50,000.

(3) find all employees who live in the same city & the company they work for.

(4) find all employees who live in the same city & same street as their manager.

(5) find all employees whose salary is more than every employee of 'XYZ' company.

Try :

Soln : (1) $\Pi_{1,3}\left(\text{Lives} \bowtie \left(\Pi_{\text{P.name}}\left(\sigma_{\text{C.name}='FCI'}(\text{Works})\right)\right)\right)$

(2) $\Pi_{1,3}\left(\text{Lives} \bowtie \left(\Pi_{\text{P.name}}\left(\sigma_{\substack{\text{C.name}='FCI' \\ \& \\ \text{Salary} > 50000}}(\text{Works})\right)\right)\right)$

(3) $\Pi_1\left(\sigma_{3=8}(\text{Lives} \bowtie \text{Works} \bowtie \text{Located in})\right)$

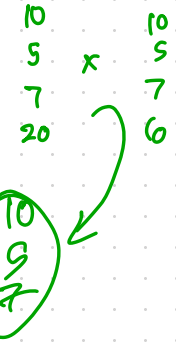
(4) $P = \text{Lives} \bowtie \text{Manages}$ { P.name, street, city, M.name.

$$\Pi_1\left(\sigma_{1=7 \wedge 2=4 \wedge 3=5 \wedge 1\neq 4}(\text{Lives} \times P)\right)$$

(5) $P = \Pi_3\left(\sigma_{\text{C.name}='XYZ'}(\text{Works})\right)$

$Q = \Pi_1\left(\sigma_{1<2}(P \times P)\right)$

$R = P - Q$



$(\text{Works} \times R)$

→ we can solve any query except sorting, etc.

Tuple Calculus

* Propositional Logic

$\neg$ (not), $\wedge$ (and), $\vee$ (or),

$\Rightarrow$ (Implication or If/then)

$\equiv$ (Equivalence or Iff)

* Proposition :

A declarative statement which will be either true or false is called a proposition.

→ Symbols used to represent prop. is called atom(s).

* Tuple Relational Calculus : ⟶ takes this form $\{t \mid \psi(t)\}$ where

→ t is a tuple variable of some fixed length.
& $\psi(t)$ is a formula built from the atoms.

$R(s)$, $s \in R$

$s[i]$ θ $u[j]$

⇒ i^{th} attribute of tuple s is in relation θ with j^{th} attri of tuple u.

- **Existential quantifier ($\exists$)**

→ There exist atleast one tuple x for which this formula will be true. $(\exists x)\ \psi(x)$

- **Universal quantifier ($\forall$)**

→

$$\left.\begin{array}{l}(\forall x)\ \psi(x) \\ (\exists s)(R(s))\end{array}\right\} \text{Formula}$$

There exist atleast one tuple s which is a tuple of the relation R.

$$(\forall s)(\psi(s)) \Big\} \text{Formula}$$

Any tuple s you take, it will be in Relation R.

$R \cup S$: $\{ t \mid R(t) \vee S(t) \}$

$R - S$: $\{ t \mid R(t) \wedge \neg S(t) \}$

$R^{(r)} \times S^{(s)}$: $\{ t^{(r+s)} \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge$ (u belongs to R, v belongs to S)

→ I have to use 3 tuple variables

$$(t[1] = u[1] \wedge t[2] = u[2] \wedge \cdots t[r] = u[r] \wedge$$
$$t[r+1] = v[1] \wedge \cdots\cdots \wedge t[r+s] = v[s]) \}$$

$\Pi_{i_1, i_2, \ldots, i_K}(R)$: $\{ t^{(K)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge \cdots\cdots$

→ I have to use 2 tuple variables.

$$\cdots\cdots \wedge t[K] = u[i_K]) \}$$

$\sigma_F(R)$: $\{t \mid R(t) \wedge F'\}$ — denoting as F' bcz F & F' will not be same, one is Relational algebra one is tuple calculus.

You are selecting tuple s.t. it will be a member of R only, so we don't need 2 tuple var, but only 1 → t

Eg:

$\Rightarrow \pi_{1,4}(\sigma_{2=3}(R^{(2)} \times S^{(2)}))$:

$$\{t^{(2)} \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge \underbrace{u[2] = v[1]}_{2=3} \wedge \underbrace{t[1] = u[1]}_{\pi_1} \wedge \underbrace{t[2] = v[2]}_{\pi_4}\}$$

F'

$\Rightarrow R^{(2)} \bowtie S^{(2)}$:

$$\{t^{(2)} \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge u[2] = v[1] \wedge t[1] = u[1] \wedge t[2] = v[2]\}$$

Q. Find the branch name, loan No., Customer Name & amount for loans over ₹ 10,000.

Customer (C.name, street, C.city) V
Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount) U
Branch (B.name*, B.city, Asset)
Client (C.name*, emp.*name)

⇒ Borrow ✓

$\{t \mid t \in Borrow \wedge t[4] > 10000\}$

Q. Find all customers who have loan more than ₹ 10,000.

⇒ $\{t^{(1)} \mid (\exists u)(u \in Borrow \wedge u[4] > 10000 \wedge t[1] = u[3])\}$

} have to use seperate tuple variable

Q. Find all customers who have loans from "IIT" branch and the cities where they live.

⇒ $\{t^{(2)} \mid (\exists u)(\exists v)(u \in Customer \wedge v \in Borrow \wedge u[1] = v[3] \wedge v[1] = \text{"IIT"} \wedge t[1] = u[1] \wedge t[2] = u[3])\}$

Without Cartesian Product

↳ $\{t \mid (\exists u)(u \in Borrow \wedge u[1] = 'IIT' \wedge t[1] = u[3] \wedge (\exists v)(v \in Customer \wedge u[3] = v[1] \wedge t[2] = v[3]))\}$

→ with cartesian product $\{t \mid (\exists u)(\exists v)(u \in Borrow \wedge v \in Customer \wedge u[1] = 'IIT' \wedge u[3] = v[1] \wedge t[1] = u[3] \wedge t[2] = v[3])\}$

Q. Find all customers having a loan or Account or Both at 'IIT' branch.

⇒

- Deposit (B.name, ac. No., C.name, balance)
- Borrow (B.name, loan no., c.name, amount)

⇒ $\{t^{(1)} \mid (\exists u)(u \in Deposit \wedge u[1] = 'IIT' \wedge t[1] = u[3])$

Logical OR → $\vee (\exists v)(v \in Borrow \wedge v[1] = 'IIT' \wedge t[1] = v[3])\}$

Instead :

Logical AND : have both account & loan at 'IIT' branch

AND NOT : have account but no loan at 'IIT' branch.

Q. Find all customers who have account at all branches in the city 'Kanpur'.

⇒ Customer (c.name, street, c.city)
Deposit (b.name, ac.no., c.name, balance) ✓

$\{t \mid (\forall u)(u \in Branch \wedge u[2] = \text{'Kanpur'} \Rightarrow (\exists v)(v \in Deposit \wedge u[1] = v[1] \wedge t[1] = v[3]))\}$

If P then Q.

$P \Rightarrow Q$

$\neg P \vee Q \longrightarrow \{t \mid (\exists u)(u \notin Branch \vee u[2] \neq \text{'Kanpur'} \vee (\exists v)(v \in Deposit \wedge u[1] = v[1] \wedge t[1] = v[3]))\}$

Till here : Syllabus for Midsem.

SQL :

```
Select  A1, A2, ...., An
from    R1, R2, ...., Rn
where   P ;
```

correspond relational algebra

$$\pi_{A_1, A_2, \ldots, A_n} (\sigma_P (R_1 \times R_2 \times \ldots \times R_m))$$

→ Union intersect minus
and or not

Q. Find the name of all branches in deposit relation.

⇒ Fill at last

```
→ Select distinct b.name
  from Deposit ;
```

(distinct → for unique selection)

Q. find all customers who have account at IIT branch.

⇒

```
Select  C.name
from    Deposit
where   b.name = 'IIT' ;
```

Q. find all customers having a loan, an account or both at IIT branch.

⇒

```
(Select c.name
 from Deposit
 where b.name = 'IIT')
union
```

```
( select  C.name
  from    Borrow
  where   b.name = 'IIT');
```

Q. find all customers having a loan at any branch & their city.

⇒

Customer (C.name, street, C.city)
Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

```
Select  C. C.name, C.city
from    Customer C, Branch B
where   C. C.name = B. C.name ;
(and    b.name = 'IIT') → for only IIT branch
```

Select
from
where

Q. Find all customers who have both loan and account at IIT branch.

⇒

```
Select  C.name
from    Deposit
where   b.name = 'IIT'
    and C.name  in  (Select  C.name
                     from    Borrow
                     where   b.name = 'IIT')
```

Q. find all customers who have account at IIT branch but no loan from IIT branch.

⇒
```
Select  C.name
from    Deposit
where   b.name = 'IIT'
and     C.name  not in
        (Select  C.name
         from    Borrow
         where   b.name = 'IIT' )
```

Q. find all branches that have more assets than any branch located in KOP.

⇒
```
Select  T.b.name
from    Branch T, Branch S
where   T.asset > S.asset
  and   S.b.city = 'KOP'
```

```
Select  b.name
from    Branch
where   asset > any

        (Select  asset
         from    Branch
         where   b.city = 'KOP' )
```

> any
< any
<> any

> all
< all

Q. find all customers with balance lies between 50000 & 60000.

⇒
```
Select c.name
from Deposit
where balance >= 50000 and balance <=60000
```

Customer (C.name, street, C.city)
Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

```
Select c.name
from Deposit
where balance between 50000 and 60000
```

not between

Q. find in alphabetical order all the customers who have loan in IIT branch.

⇒
```
Select C.name
from Borrow
where b.name = 'IIT'
order by C.name
```

For descending order
order by ___ desc;

---

avg, min, max, sum, count

Q. Find the average account balance at all branches.

⇒
```
Select   b.name, avg(balance)
from     Deposit
group by b.name
having   avg(balance > 10000)
```
If we want only avg. balance > 10000

Q. Find no. of tuples in Customer relation.

⇒
```
Select  count(*)
from    Customer
```

Q. Find all customers who have deposit at IIT branch but for whom there is no entry in Customer relation.

⇒
```
Select  c.name
from    Deposit D
where   b.name = 'IIT' and c.name not in
                          (Select c.name
                           from   Customer)
        not exists
                (select *
                 from   customer C
                 where  D.c.name = C.c.name)
```

⇒ After midsem, one problem → compulsory to submit ✓

Teacher (T.name, Dept., Tl.No., S.title)
(Telephone, subject title)

Student (S.name, course, hall)

Study (S.title, S.name, level, status, marks)
(Subject name)

Q. Find the name of teachers of Math dept. who teach 200 Level subjects.

Q. Find students of CSDP course living in VS or SN hall who study No subject taught by Prof. XYZ.

⇒
```
Select S.name
from Student
where course = 'CSDP' and
      (hall = 'VS' or hall = 'SN')
and S.name not in
    (select S.name
     from Teacher T, Study S
     where t.name = 'XYZ'
      and T.S.title = S.S.title)
```

Q. Find the avg marks in the subjects taught by Prof. XYZ.

⇒
```
Select  S.S.title , avg (marks)
from  Study S, Teacher T
where  T.S.title = S.S.title
    and  t.name = 'XYZ'
  group by  S.S.title
```

Q. Find the name of the subjects such that all students studying the subject secure more than 80 marks.

⇒

Teacher (T.name, Dept., Tl.No., S.title)
Student (S.name, course, hall)
✓Study (S.title, S.name, level, status, marks)

```
Select  S.title
from  Study
  group by  S.title
  having  min (marks) > 80
```

Q. Find the name of students who secure equal marks in 2 different subjects.

⇒
```
Select  S.name
from  Student
where  S.name in ( Select  S.S.name
                   from  Study S , Study T
                   where S.S.name = T.S.name
                   and  S.S.title ≠ T.S.title
                   and  S.marks = T.marks.
```

you can also write this ONLY

Q. Find the number of students in each subject taught by Prof. XYZ.

⇒ Teacher (T.name, Dept., Tl.No., S.title)
Student (S.name, course, hall)
Study (S.title, S.name, level, status, marks)

```
Select  S.s.title, count (S.name)
from    Study S, Teacher T
where   t.name = 'XYZ'  and
        S.s.title = T.s.title
        group by s.s.title
```

⇒ R(A,B,C) | Is it in 3NF ?
F = {A → C, B → C}

⇒ 3NF decomposition

KEY : AB

$R_1(A,C)$, $R_2(B,C)$, $R_3(A,B)$ Yes

BCNF decomposition :
$R_1(A,C)$, $R_2(A,B)$ No

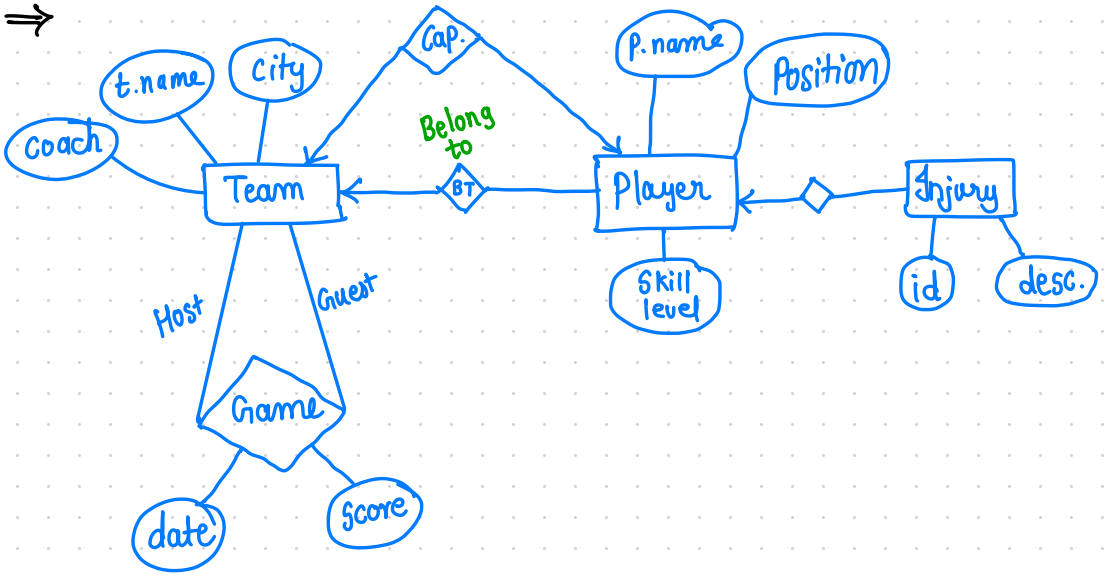
→ In SQL we can use nested queries.

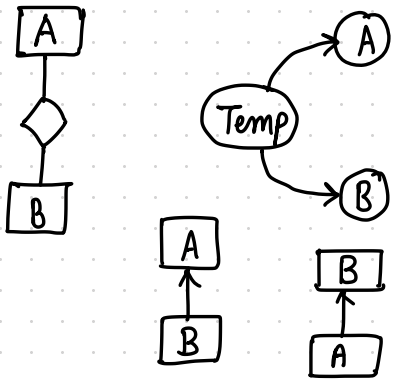
Q. Suppose you are given the following requirement for a database for IPL :

1) IPL has many teams
2) Each team has a name city, coach, captain and a set of players.
3) Each player belongs to only 1 team
4) Each player has a name, a position, skill level & the set of injury records
5) A team captain is also a player.
6) A game is played between 2 teams referred as host team and guest team & has a date and a score.

Draw the E-R diagram ?

⇒





★ QWEL

Format :

range of t_1 is R_1 → Relations
range of t_2 is R_2 → Relations
⋮
range of t_n is R_n → Relations
retrieve $(t_{i_1} \cdot A_{j_1}, t_{i_2} \cdot A_{j_2}, \ldots, t_{i_m} \cdot A_{j_m})$ → Attributes of t_is
where P ; → condition

t_1 → tuples

Relational Algebra :

$\{ \; | \; (\exists t_1)(\exists t_2)(\exists t_3)$

Q. find all customers having an account at IIT branch:

Customer (C.name, street, C.city)
Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

⇒ range of t is Deposit
retrieve (t.c.name)

where t.b.name = 'IIT'

Q. find all customers having a loan at IIT branch & their cities.

(Customer (C.name, street, C.city)
⇒ Deposit (B.name, ac. No., C.name, balance)
(Borrow (B.name, loan no., c.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

⇒ range of t is Borrow
range of u is Customer
retrieve (t.c.name, u.c.city)
where t.c.name = u.c.name
and
t.b.name = 'IIT'

Q. Find all customers who have loan & account at IIT branch.

Customer (C.name, street, C.city)
[Deposit (B.name, ac. No., C.name, balance)
[Borrow (B.name, loan no., C.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

⇒
```
range of t is Borrow
range of u is Deposit
retrieve (u.C.name)
where t.c.name = u.c.name
      and
      t.b.name = 'IIT'
      and
      u.b.name = 'IIT'
```

Q. find all customers who have account or loan or both at IIT branch. (Union operation)

Customer (C.name, street, C.city)
(Deposit (B.name, ac. No., C.name, balance)
(Borrow (B.name, loan no., C.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

⇒ Here, we can't use nested queries like SQL, its difficult.

```
range of u is Deposit
retrieve into temp (u.c.name)
where u.b.name = 'IIT'
```

Creating new file ←

Accounts at IIT

```
range of S is Borrow
append to temp (s.c.name)
where S.b.name = 'IIT'
```

(appending)

loans at IIT

Storing in existing file / will create if not exist

```
range of t is temp
retrieve unique (t.c.name)
```

Taking unique

112

distinct → In SQL

Q. Find all customers who have account at IIT branch but no loan at IIT branch.

⇒

Same

```
range of u is Deposit
retrieve into temp (u.c.name)
where u.b.name = 'IIT'
```

Accounts at IIT

```
range of S is Borrow
range of t is temp
delete (t)
where t.c.name = S.c.name
      and
      S.b.name = 'IIT'
```

Deleting those who have loans at IIT

```
range of t is temp
retrieve unique (t.c.name)
```

Q. find all customer names in alphabetical order who have accounts at IIT branch.

Customer (C.name, street, C.city)
Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount)
Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

⇒ range of u is Deposit
retrieve u.c.name
where u.b.name = 'IIT'
Sort by u.c.name

max sum
min count
avg

Q. find the average account balance for all accounts at IIT branch.
→ Condition will come inside brackets.

⇒ range of t is Deposit
retrieve avg(t.balance where
t.b.name = 'IIT')

Q. Find all account numbers whose balance is more than the average balance at the branch where the account is held.

Deposit (B.name, ac. No., C.name, balance)

⇒ range of t_1 is Deposit
range of t_2 is Deposit
retrieve ____ where ____
____ long method

≡ range of t is Deposit
retrieve (t.ac.no.)
where t.balance > avg (t.balance by t.name)

Q. find all customers who have account at all branches in the city "KGP".

⇒ Customer (C.name, street, C.city) → ?
t_1 S ✓Deposit (B.name, ac. No., C.name, balance)
Borrow (B.name, loan no., c.name, amount)
V ✓Branch (B.name, B.city, Asset)
Client (C.name, emp. name)

all such B.name, for all C.name, B.city = KGP

⟹ range of t is Deposit
range of u is Branch
range of s is Deposit
retrieve (t.c.name)
where
countu (s.b.name where
s.b.name = u.b.name and
u.b.city = 'KGP' and
s.c.name = t.c.name)

=
countu (u.b.name
where u.b.city = 'KGP')

Tomorrow : student teacher
& File organization. → not in books
± Attend classes → different than books

Teacher (T.name, Dept., Tel.No., S.title)
Student (S.name, course, hall)
Study (s.title, s.name, level, status, marks)

28th Feb 2023

Q. Find the name of subjects such that all students studying the subject secure more than 80 marks.

→ range of t is Study
retrieve (t.stitle)
where min(t.marks by t.Stitle) > 80

Q. find the students of CSDP course living in 'VS' or in 'SN' hall, who study no subject taught by prof. 'XYZ'.

⇒ range of t is student
append to temp1 (t.sname)
where (t.course = 'CSDP' and
(t.hall = 'VS' or t.hall = 'SN'))

range of u is teacher
range of s is study
append to temp2 (s.sname)
where u.stitle = s.stitle
and u.tname = 'XYZ'

range of t_1 is temp1
range of t_2 is temp2
delete (t_1)
where t_1.sname = t_2.sname

range of t_3 is temp1
retrieve unique (t_3.sname)

Q. Find the maximum marks secured in each subject of the CSDP course.

⇒ Teacher (T.name, Dept., Tel.No., S.title)
Student (S.name, course, hall)
Study (S.title, S.name, level, status, marks)

⇒ range of t is study
range of u is student
retrieve (t.stitle, max(t.marks by t.stitle))
(t.stitle: not asked but we are giving)
where (t.sname = u.sname
and u.course = 'CSDP'

Q. Find the name of students who secured equal marks in two different subjects.

⇒ range of t is study
range of u is study
retrieve (t.sname)
where (t.sname = u.sname
and t.stitle <> u.stitle
and t.marks = u.marks)

File Organization

→ How to store your data in secondary device s.t. retrieving time will be minimum.

Data to be stored in

- Main memory → Data structures → negligle time to perform the operations
- Secondary device → File organization → (retrieving from here) is time consuming

⇩

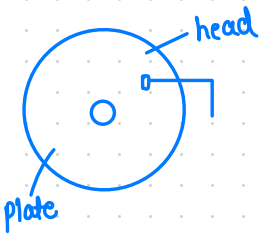
Insert
Delete
Update
Searching

For which type of operation, I should store my data in which fashion.

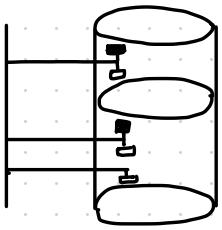
Block ↘ lowest unit by which we transfer data

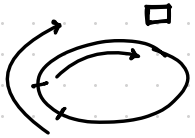
1 Block = 2400 bytes (varies from machine to machine)
or 4800 bytes

Gramophone = sequence of tracks give you a cylinder.



head has information, on which Block it has to perform operation.





Seek time = time it takes to move the arm to the correct cylinder.

disk latency time (Rotation latency time) =

Average seek time (s) = time it takes to traverse $\frac{1}{3}$rd of the cylinder.

Averge rotational latency time (r) : avg time it takes to locate the correct block on a track once head is on correct track.

$= \frac{1}{2} *$ revolution time

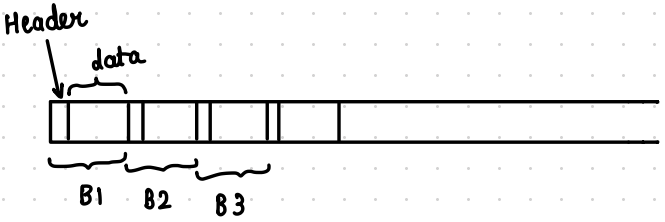
Block transfer time (btt): the time it takes to transfer the information of one block from disk to memory once the head is correctly positioned at the beginning of a block.

$= s + r + btt$

(Size of block) B bytes

(Data transfered at the rate of) t bytes/ms

$$btt = \frac{B}{t} \text{ ms}$$



Effective block transfer time (ebt) : time it takes to transfer the information of one block along with header information.

$$ebt = \frac{B}{t_1}$$

ebt will be greater

* Parameters for IBM 3380

B	2400 bytes
btt	0.8 ms
C	600 (blocks per cylinder)
ebt	0.84 ms
N	885 (No. of cylinders)
r	8.3 ms
s	16 ms
t	3000 bytes/ms
t'	2857 bytes/ms

* Read 10 adjacent blocks (same for write)

time = $s + r + 10 \times ebt$ → $btt + 9 \times ebt$

= 32.7 ms

* Random access of 10 blocks

time = $10\,(s + r + btt)$

= 251 ms

* Sequential access time $\approx b \times ebt$ (when value of b is very large)

random access time = $b \times (s + r + btt)$

* Heap File Organization

T_F = time to fetch a record

T_N = time to fetch next record

T_I = time to insert new record

T_U = time to update a record

T_D = time to delete a record

T_X = time to exhaustive reading of a file

T_Y = time to reorganize a file

b blocks

$T_F = \frac{b}{2} \times ebt$

* 100000 records of 400 bytes each

$\Rightarrow b = \frac{100000 \times 400}{2400} = \frac{100000}{6} = 16,667$ (no. of blocks)

$\therefore T_F \approx 7$ sec. → avg. time taken to access any 1 block. not considering time $(s+r)$

* fetch 10000 records (in hospital)

time = ≈ 19 hours → Not suitable

OS does not read the header part of 1 block ⇒ btt

when you access continuous blocks, you are forced to read header also = ebt

ebt > btt

T_N = time to fetch next record → new address → it can be anywhere so I have to start from beginning.

$= T_F$

T_I = time to insert new record

→ Assuming I have enough space in last block to insert new record.

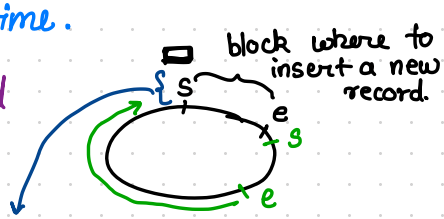
→ Assuming, I have address of last & first block.

→ Trasfer to main memory, use data structures, bring back

often information is modified & write it.

→ Main memory takes negligible time.

s & e can be interchanged



$(s + r) + btt$ (read)

$+ (2r - btt)$ (one full revolution) $= 2r$

$+ btt$ (write)

$T_I \Rightarrow \boxed{s + r + btt + 2r}$

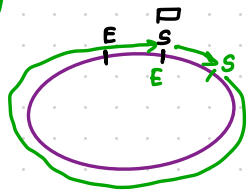
T_D = time to delete a record

→ we don't delete physically, we just put a deletion beat.

→ search the block

+ put deletion beat in main memory

$T_D = \boxed{T_F + 2r}$ includes writing time.



T_x = time to exhaustive reading of a file

$\boxed{T_x = b \times ebt}$

e.g. $\frac{100000 \times 400}{2400} = b =$

$b \times ebt = \underline{14 \text{ sec.}}$ → To access 100000 records

→ This HFO is most suitable for sequential accessing of

data from beginning to end.

→ Not suitable for individual fetching.

T_x (independent fetch of all records) (n = no. of records)

$= n \times T_F$

$= n \times \frac{b}{2} \times ebt$ = 190 hours → 8 days

→ Random accessing is not possible with this organization.

T_Y = time to reorganize a file

→ Rewrite entire file again by removing the deleted records. write only active records.

→ we do it periodically (15 d or 1 month)

→ Access all the blocks, then rewrite active records only.

$T_Y = b \times ebt + \frac{n}{Bfr} \times ebt$

(n = no. of active records; Bfr = No. of records per block; $\frac{n}{Bfr}$ = no. of blocks required to store the active record.)

★ Bucket :

→ logical unit of data structure in file organization.

→ Buffer = a portion of the memory kept aside for doing some specific jobs. Physically it doesn't exist, only logically you are visualizing.

1 Bucket = n blocks , $n = 1, 2, 3, \ldots$

→ I will access data Bucket wise, not block wise.

Eg: 1 bucket = 2 blocks.

time = $(s + r + btt) + (e\,bt)$

⇒ Random time = $bK\,(s + r + dtt)$ → $btt + n \cdot ebt$

bK = no. of buckets required → accessing randomly.

dtt = data transfer time

→ If Bucket size increases → bK will decrease

Random time = $bK\,(s + r)$ + $bK \times dtt$

$bK\,(s+r)$ ↓ Reduce

$bK \times dtt$ ↑ constant

Random time ↓ Reduce

For sequential case :

Eg: bK = 1 block = 2400 bytes

dtt = 0.8 $(= btt)$

bucket size = 2 blocks

$dtt = btt + 1\,ebt = 0.8 + 0.84$

$= 1.64$

Bucket size = 1 $bK = 1250$

$\Rightarrow 1250 \times 0.84 \approx 1050$

Bucket size = 2 $bK = 625$

$\Rightarrow 625 \times 1.64 \approx 1025$

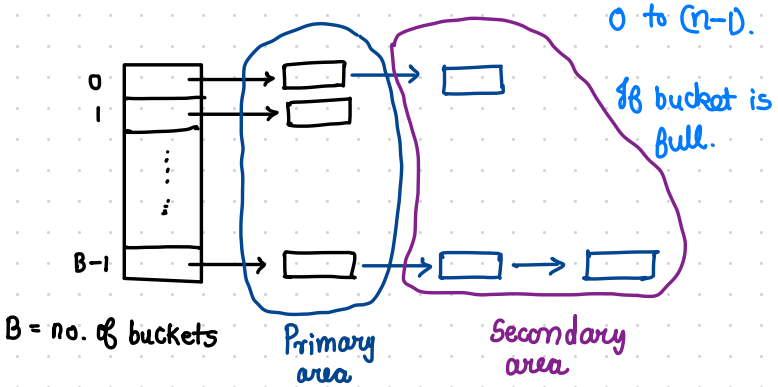
→ Not useful when we want to access sequentially. It is useful for Random access → it reduces the time.

★ Hash File Organization

→ I will store the records depending upon Key value.

no. of blocks required = n

create a hash function → will give me integer value between 0 to (n-1).



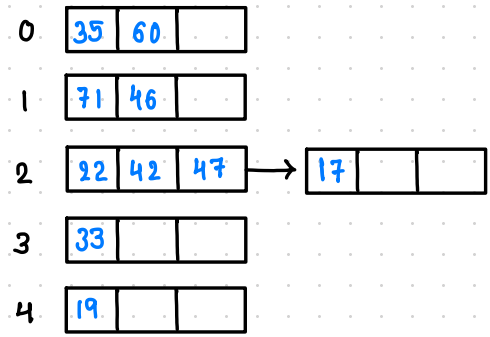
★ Data Structure

22, 35, 33, 42, 71, 60, 47, 19, 46, 17

$h(x) = x \bmod 5$ → no. of buckets

No. of buckets = 5

No. of records per bucket = 3



Load factor

$$Lf = \frac{n \text{ (No. of records)}}{M \times Bkfr}$$

M: no. of buckets in primary area

Bkfr: No. of records per bucket

$$\frac{10}{5 \times 3} = \frac{2}{3} = 0.6$$

→ If it is less than 1 then we can say that overflow area will be less.

T_F (no chain) = $s + r + dtt$

T_F(successful) = $(s + r + dtt)$ + $\frac{x}{2}(s + r + dtt)$

$(s + r + dtt)$: To access first bucket in the primary area

$\frac{x}{2}$: I have x buckets in sec. area, on average I will access $\frac{x}{2}$ buckets.

T_F (Unsuccessful) = $(s + r + dtt) + x(s + r + dtt)$

T_I = T_F(unsuccessful) + $2 \cdot r$ (seen in last class)

T_F(unsuccessful) : The information of that block / bucket is transferred to main memory

$T_D = T_F$ (successful) + $2r$

$T_U = T_D$: If key value is not changed.

(Assuming key value is not changed)

$T_U = T_D + T_I$: If key value is changed.

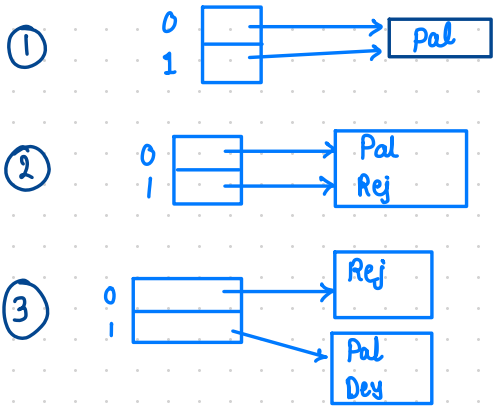
T_x = time to exhaustive reading of a file

= $n \times T_F$

Extendible hashing

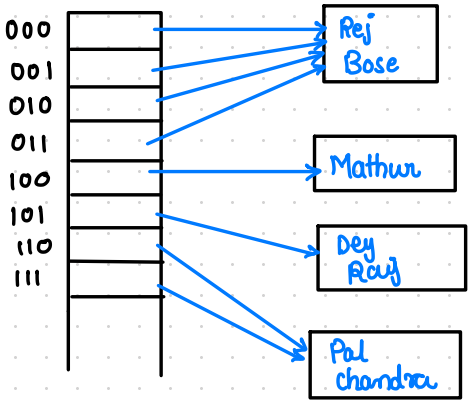
	Name	h(Name)
5	Bose	0010 1101
6	Chandra	1101 0101
3	Dey	1010 0011
7	Mathur	1000 0111
1	Pal	1111 0001
4	Ray	1011 0101
2	Rej	0101 1000

create a bucket



Buckets = 3

bits =



Next Monday

→ Pavan Kumar

B+

★ B+ Tree :

Roll No., Grade } of a student

Stored at the leaf of a tree.

→ Assume 2 or 3 records in a leaf.

Record : Roll No. Grade
39 → C
etc.

→ 2 or 3 pointer in a non-leaf

→ Non-leaf store key values

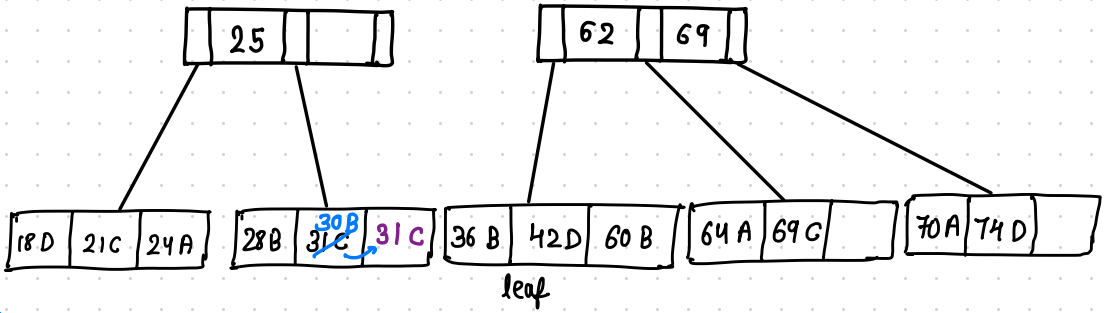
→ Policy can be strictly less than or less than equal to, anything.

B+ Tree

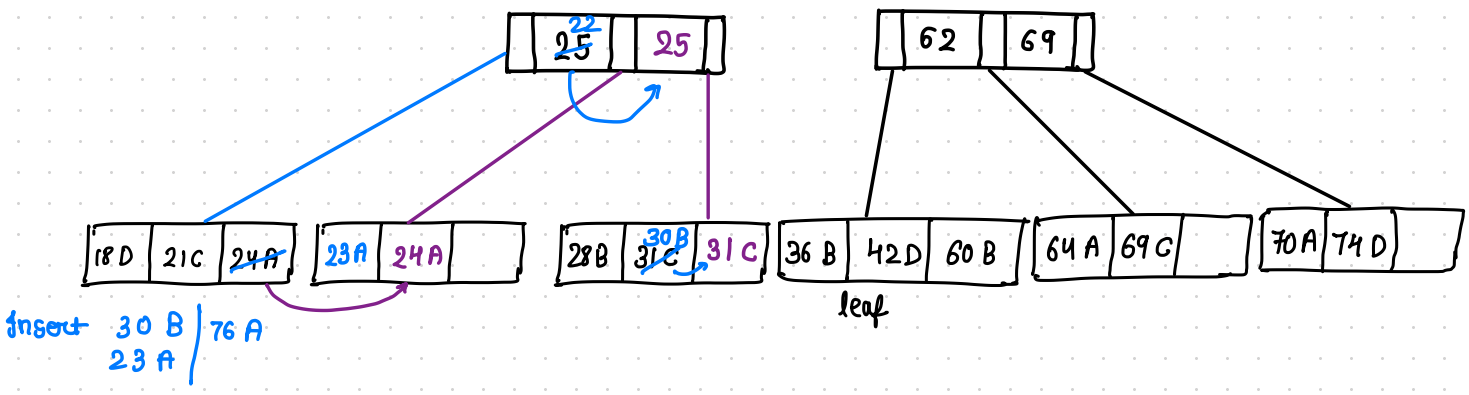
Roll No., Grade

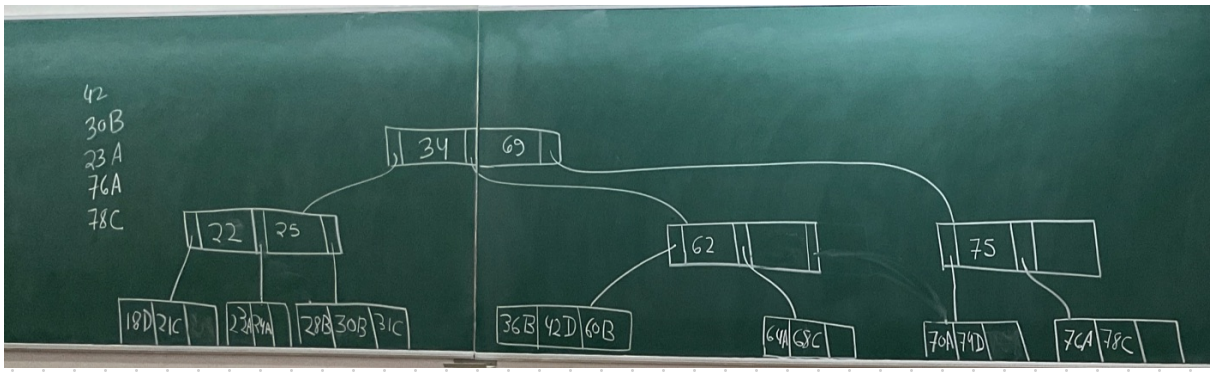
2/3 record in a leaf

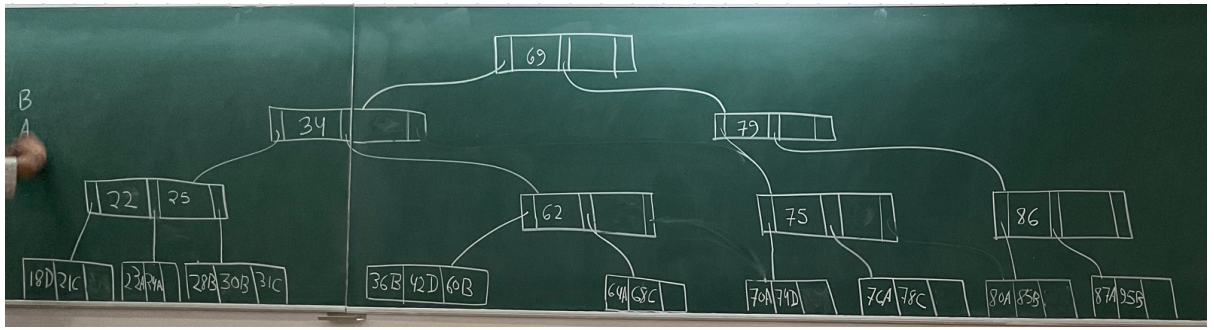
2/3 pointer in Non-leaf



Insert 30 B


22
25
25
62
69
18 D
21 C
24 A
23 A
24 A
28 B
30 B
31 C
31 C
36 B
42 D
60 B
64 A
69 C
70 A
74 D
leaf
Insert 30 B
23 A
76 A

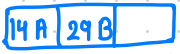




Eg:

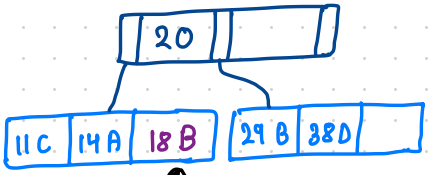
14 A

29 B



11 C

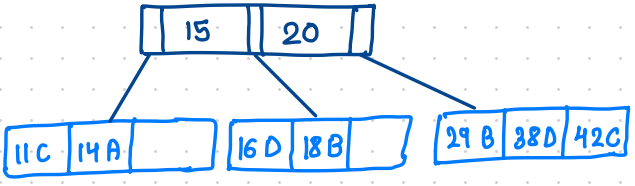
38 D



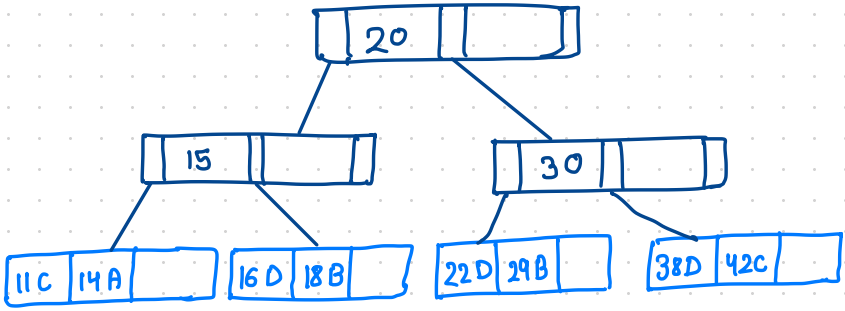
18 B

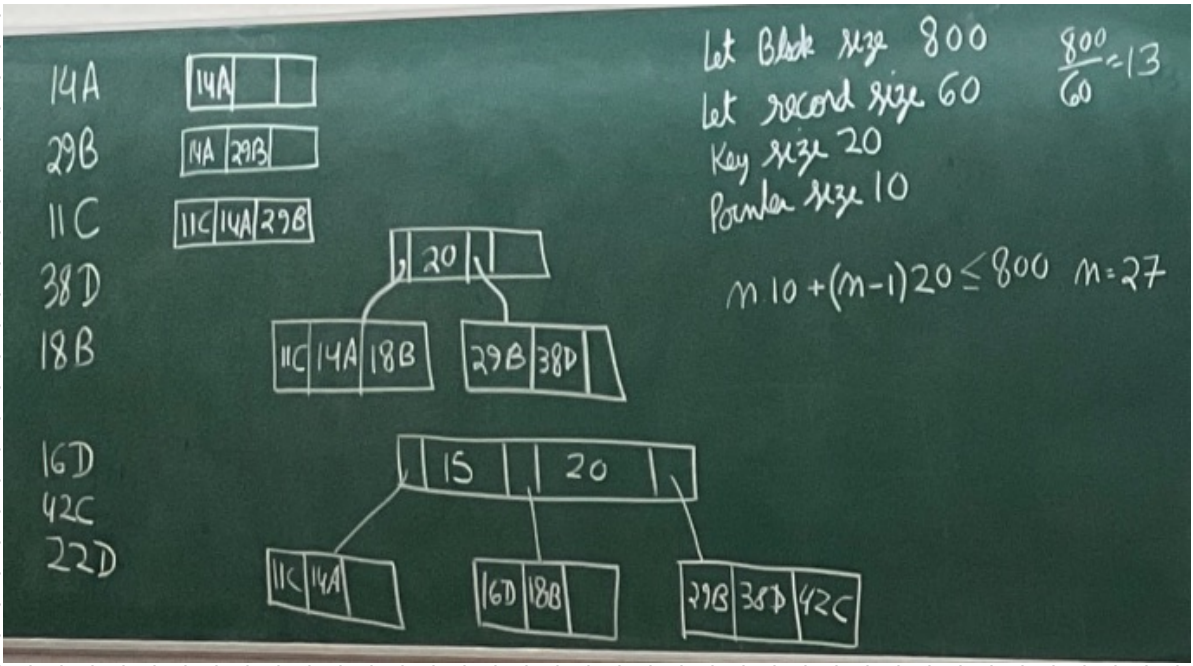
16 D

42 C



22 D


20
15
30
11C
14A
16D
18B
22D
29B
38D
42C


14A
29B
11C
38D
18B
16D
42C
22D
14A
14A 29B
11C 14A 29B
20
11C 14A 18B
29B 38D
15 20
11C 14A
16D 18B
29B 38D 42C
Let Block size 800
let record size 60
Key size 20
Pointer size 10
800/60 = 13
n·10 + (n-1)20 ≤ 800
n = 27

Let Block size 800

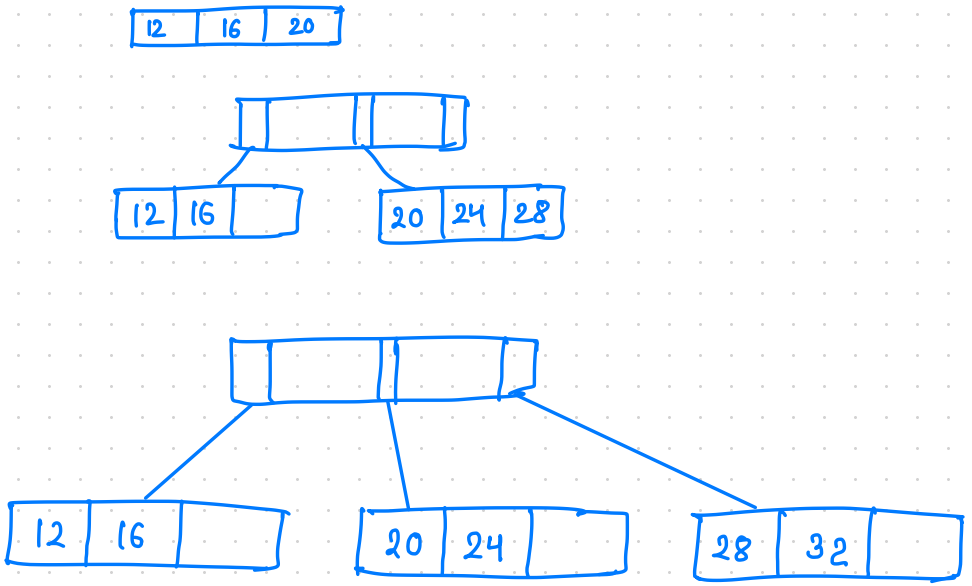
record size 60 $\quad \frac{800}{60} = 13$

Key size 20

Pointer size 10

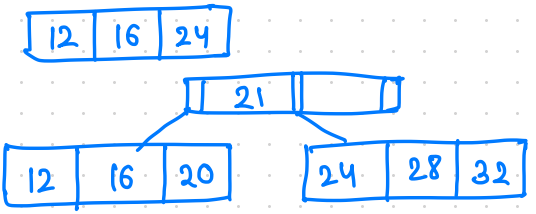
$n \times 10 + (n-1)20 \leq 800 \qquad (n = 27)$

Eg: Let 12, 16, 20, 24, 28, 32, 36, 40 are inserted



Eg: Insert : 12, 16, 24, 28, 32, 20

Orders in which records are inserted, storage changes.



2/3 record in a leaf except root

3/4/5 pointer in non-leaf except root

---

14 March 2023

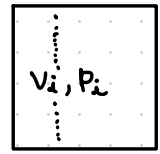
* Index files : → Key value in a sorted order
→ corresponding address where you can find the desired record

→ Original file (very large) will not change.
It will sort on key values

→ It takes only 1 block access.

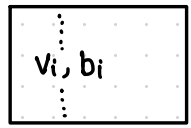
Dense Index

Original unsorted file (Heap file)



Index file
For each record, key value & address of bucket is written.

Original Sorted file



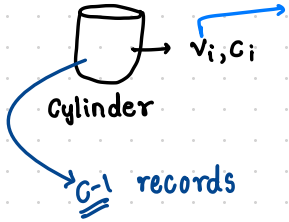
Index file
First record in the block with address b_i and key value V_i

Sparse Index

★ Indexed Sequential Access Method (ISAM)

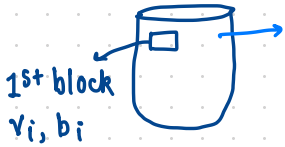
→ File is stored in many cylinders.

C cylinders



→ highest key value of the cylinder with address c_i.

sorted in cylinder



→ Contains (b+1) blocks, 1st block doesn't contain records.
∴ There will be b records.

↳ Highest key value with block address b_i.

B+ Tree of order v :

- Root has atleast 2 children unless it is a leaf.
- No internal node has more than 2v keys.
- No internal node except the root has less than v keys.

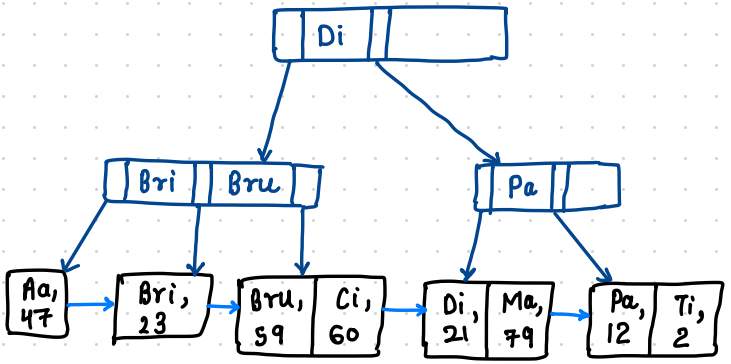
$$v \leq \leq 2v$$

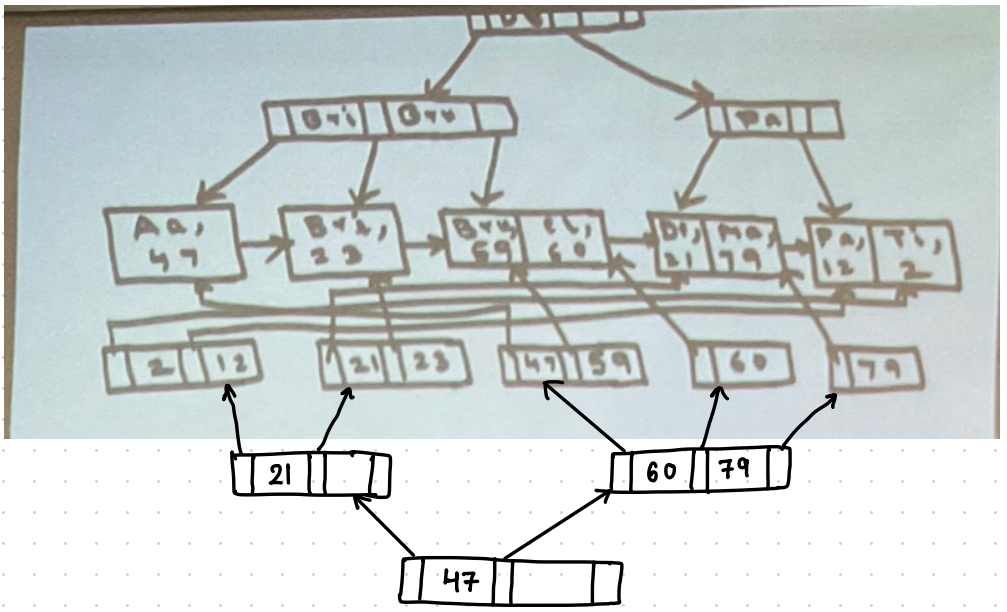
- Internal nodes contain only keys & addresses of nodes in the next level.
- An internal node with K keys has (K+1) children.
- All leafs are at the same level.
- If B+ Tree is used as primary index, the leaves of the tree contains data records.

→ If B+ Tree is used as secondary indexing, leaves contains key value & record addresses.

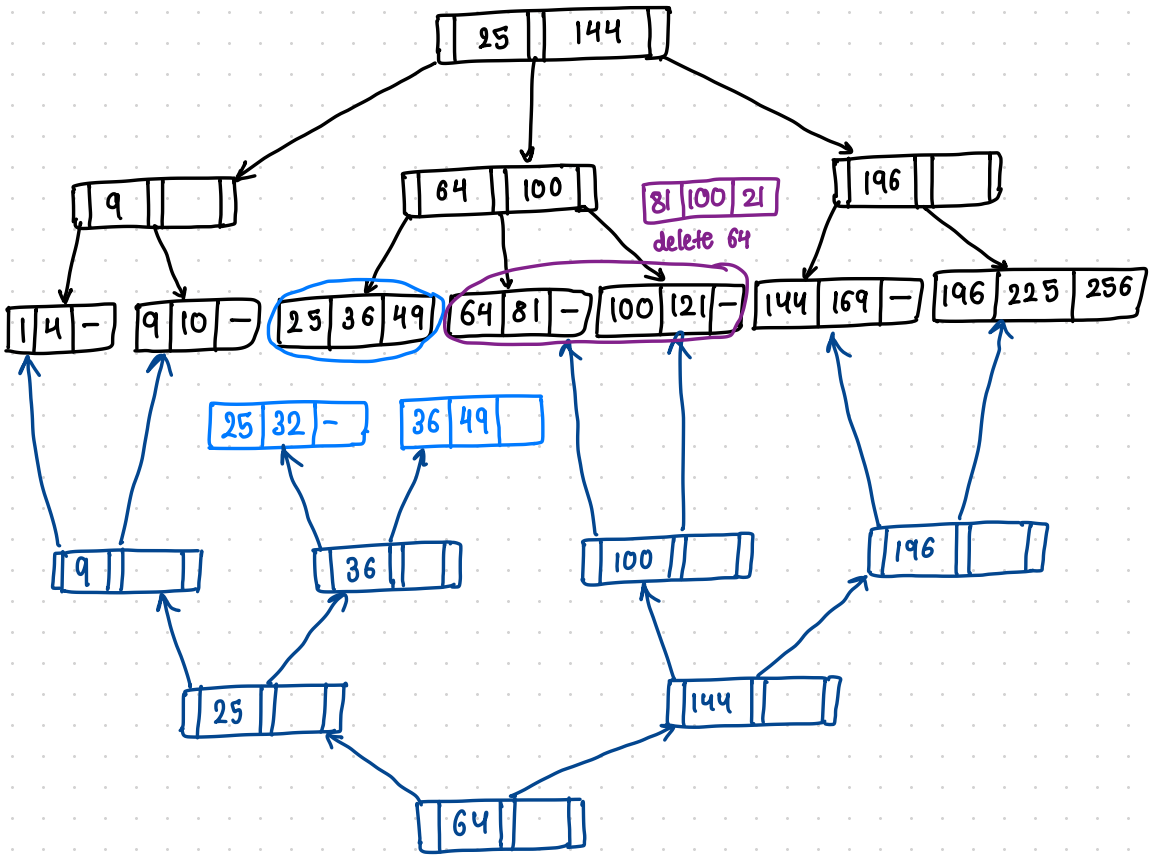
→ Leaves may also contain address of the next leaf for sequential processing.

→ If an internal node is $\frac{2}{3}$rd full then you will get optimum result.



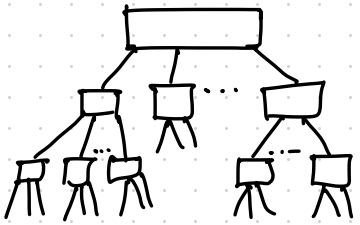


$2v = 2$
$v = 1$



1 block = 2400 bytes

Key value + address = 12 bytes (200)



140 blocks

140 x 140

140 x 140 x 140

★ Variable length records :

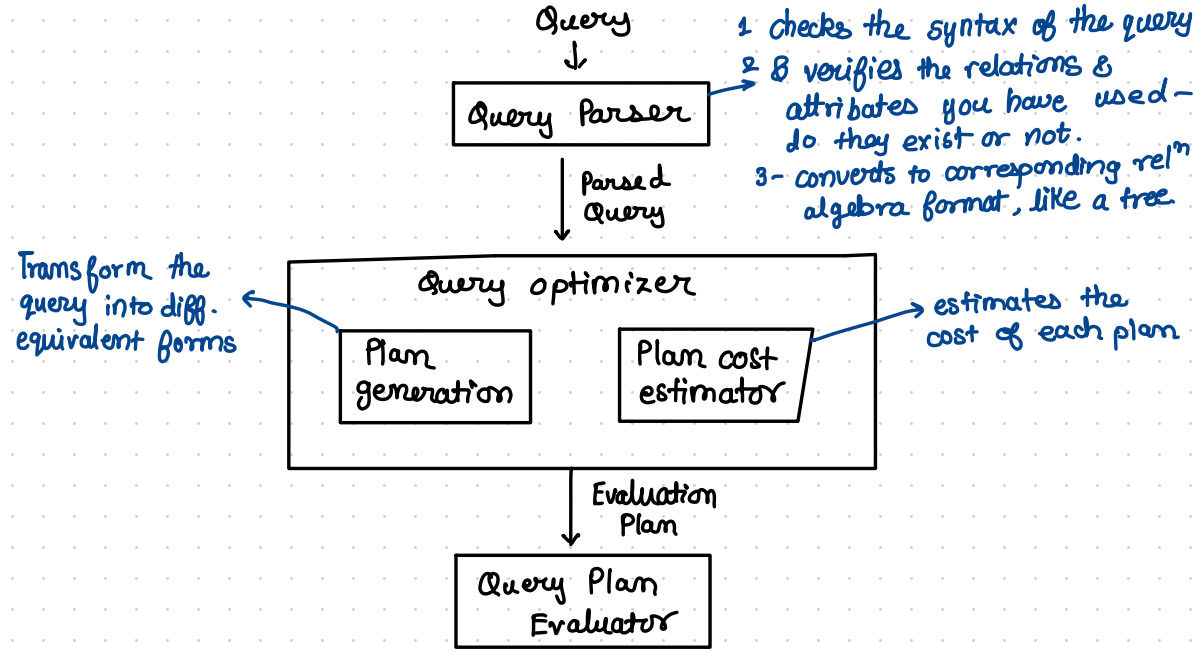
→ Convert it to a fixed length record. Take the maximum length & take it as fixed.

20th March, 2023

★ Query optimization

→ Is the SQL query unique ? No, we may write in various forms.

∴ Query is not executed in your format, it is optimized & then executed. Calculate cost of various forms & see the minimum.



★ Equivalence Rules

1.) $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

If fast then less time

- Selection operator is commutative.

2.) $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

3.) $\Pi_{L_1}(\Pi_{L_2} \cdots (\Pi_{L_n}(E)) \cdots) = \Pi_{L_1}(E)$

4.) $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$

$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

} whether cartesian product takes less time or θ-join takes less time.

5.) $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$ } θ-join is commutative

6.) $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

Same for θ-join

} Natural join & θ-join are associative.

7.) $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta_2} E_2$

} cardinality will be reducing.

↳ If θ_1 is using attributes of E_1 only.

$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$

} when θ_1 involves only E_1 & θ_2 involves only E_2.

8.) $E_1 \cup E_2 = E_2 \cup E_1$

$\cap \qquad\qquad \cap$

} ∪ & ∩ are interchangeable are associative.

9.) $\sigma_{\theta}(E_1 \; - \; E_2) = \sigma_{\theta}(E_1) \; - \; \sigma_{\theta}(E_2)$

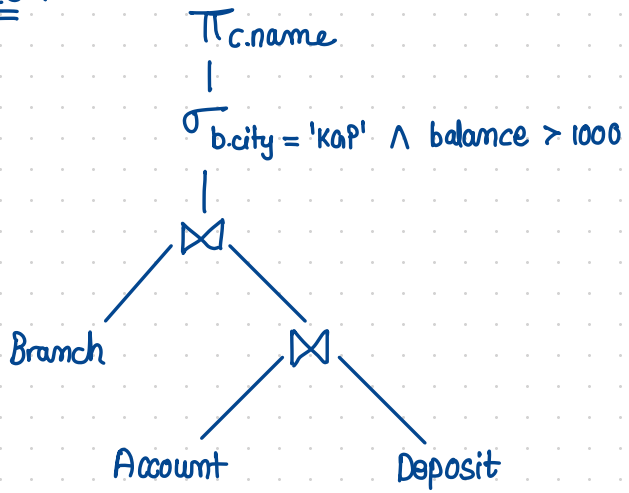
$\cup \qquad\qquad\quad \cup$

$\cap \qquad\qquad\quad \cap$

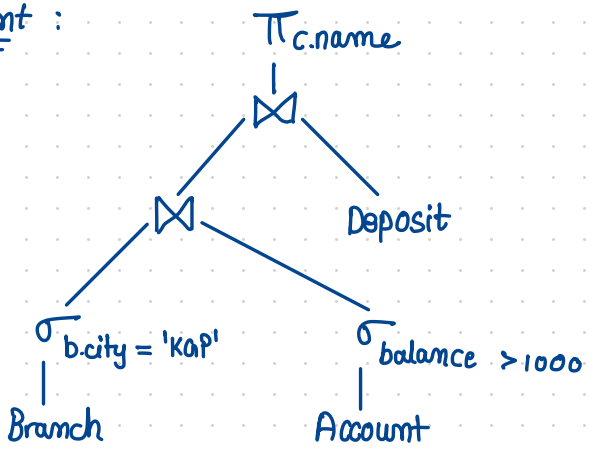
10.) $\Pi_L (E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

Eg: $\Pi_{C.name}(\sigma_{b.city = 'KAP' \wedge balance > 1000}(Branch \bowtie (Account \bowtie Deposit)))$

Tree :



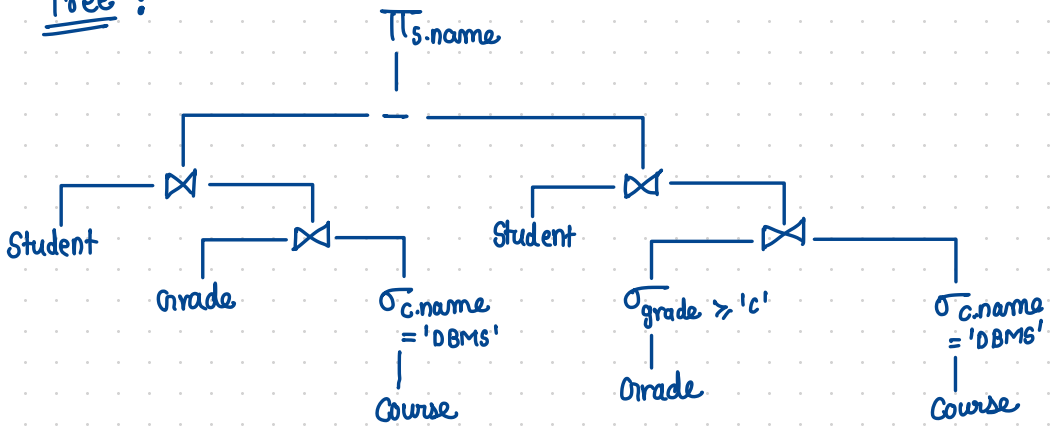
Equivalent :



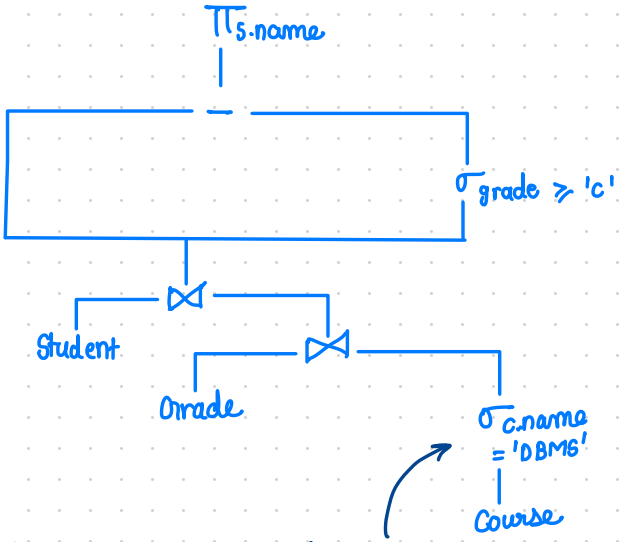
Eg: Find the list of students who have not obtained grade C or higher in DBMS course?

$$\Rightarrow \Pi_{S.name}\{Student \bowtie (Grade \bowtie \sigma_{C.name = 'DBMS'}(Course)) - (Student \bowtie (\sigma_{grade \geq 'C'}(Grade)) \bowtie (\sigma_{C.name = 'DBMS'}(Course)))\}$$

Tree:



Equivalent:

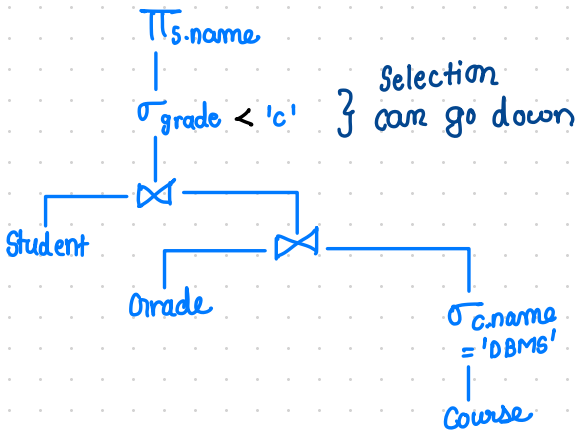


This operation we are doing only once. Earlier we were doing it twice.

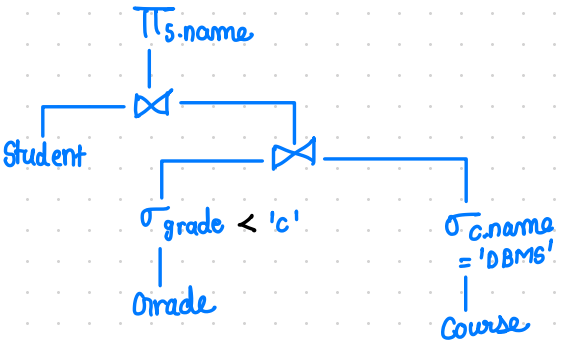
↦ ultimately I have to perform selection at the earliest. It will reduce the tuples (& hence time).

$$R - \sigma_c(R) = \sigma_{\neg c}(R)$$

Equivalent :



Equivalent :



Recovery from a failure :

→ Your system may fail when you are performing some operation.

- Periodically Backup can be done

↓

Hardware failure

Software failure

Human error

* Recovery Schemes

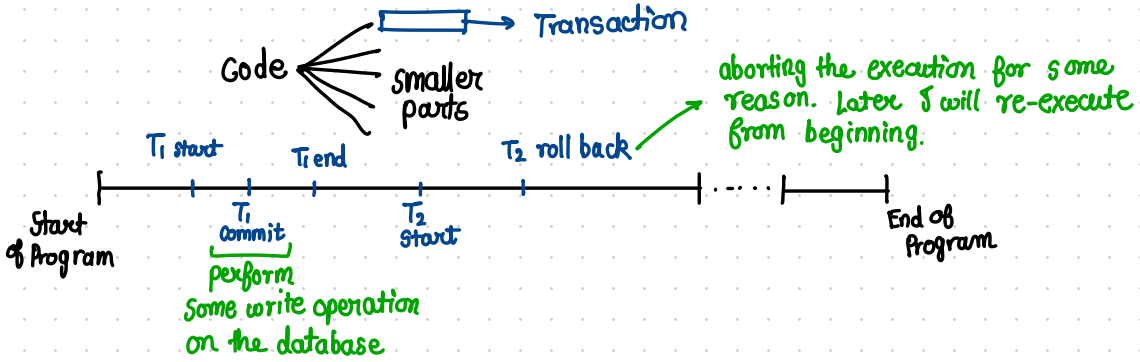
⇒ Forward error recovery : when a particular err in the system is detected, recovery system makes an accurate assessment of the state of the sys. then makes appropriate adjustment based on anticipitated ... had the system been _ _ .

Listed problem ⟶ corresponding solutions

⇒ Backward error recovery : the system is reset to some previous correct state which is free from errors.

* Transaction :

↳ A program unit whose execution may change the content of a database.



Successful Termination : It has reached the end. (after performing the operation).

Suicidal Termination : System detected some error. it forced the T to roll back.

* Properties of a Transaction :

ACID Test *

⇒ Atomicity Property : Implies that it will run to completion as an indivisible unit. At the end of which no changes have occurred to the database or the database has been changed in a consistent manner.

⇒ Consistency Property : This property implies that if the database was in a consistent state before the start of the transaction then on termination of the transaction, the database also will be in a consistent state.

⇒ Isolation Property : This property indicates that the actions performed by the T will be isolated or hidden from outside the transaction until the transaction terminates.

⇒ Durability Property : Ensures that the commit action of a Transaction on its termination will be reflected on the database.

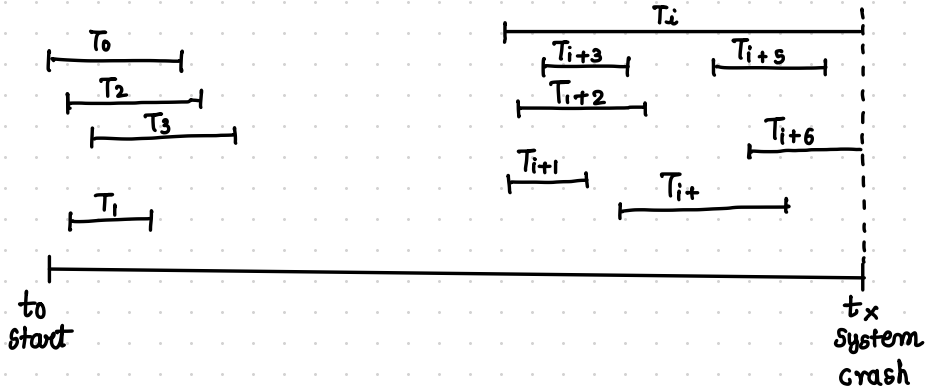
* Recovery in Centralized DBMS :

Log : a file on the secondary device which stores the info about all transaction actions, operation performed. The prev. values of the modified data items.

→ storage can be a problem.

Write ahead log strategy :

Before writing on the database, write on the log file.

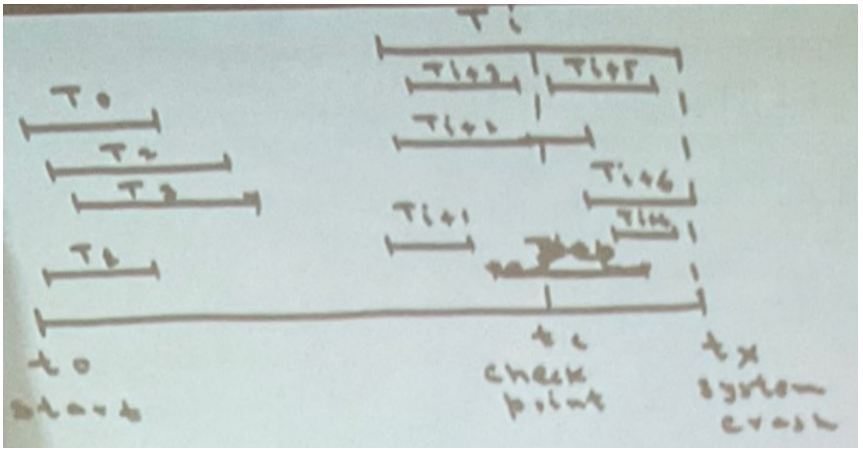


Checkpoint :

Transaction - consistent check point : Here the T that are active when system timer signals a checkpoint are allowed to complete but no new transaction are allowed to be started until the checkpoint is complete.

Action Consistent check point : Active transactions are allowed complete the current step & no new actions are allowed to be started on Database while the checkpoint complete.

T- oriented check point : Idea is to take a check-point at the end of each transaction by forcing the log of the T on stable storage.



UNDO $(T_i, T_{i+2}, T_{i+4}, T_{i+5}, T_{i+6})$

⇓

REDO $(T_{i+4}, T_{i+5}, T_{i+2})$

UNDO (T_i, T_{i+6})

✱ Do, Undo, Redo :

→ A transaction on the current database transforms it from current state to new state ⇒ Do operation

→ Undo operation comes into picture when a transaction decides to terminate itself.

→ T REDO involves performing the changes made by a T that committed before a system crash with the right ahead log strategy a commited T implies that a log for the T could be written to secondary storage but the physical database may or may not have been modified before the system failure.

✱ Concurrency Management

✱ Schedule : The order in which the statements of transactions are executed is known as schedule.

T_1

1. Read (Avg. F. sal)
2. Avg. F. sal = f_1 (Avg-F-sal)
3. Write (Avg. - F - Sal)

T_2

4. Read (Avg. S. Sal)
5. Avg. S. sal = f_2 (Avg. S. Sal.)
6. Write (Avg. S. Sal.)

S_1

1 2 3 4 5 6

S_2

4 5 6 1 2 3

S_3

1 4 2 5 3 6

↳ ↳ Data variables are independent.

AIM: Schedule should give you the correct result. (same)

T_3

1. Read (A)
2. A = A + 10
3. Write (A)

T_4

4. Read (A)
5. A = A * 1.1
6. Write (A)

A = 200

S_1

T_1 T_2

A = 231

S_2

T_2 T_1

A = 230

S_3

4 5 1 2 6 3

A = 220 (loss update)

A = 210 (finally.)

200 → 220 → write

200 → 210 → write

from T_3 → 210

	T_5		T_6
1	Read (A)	7	Sum = 0
2	A = A - 100	8	Read (A)
3	write (A)	9	Sum = Sum + A
4	Read (B)	10	Read (B)
5	B = B + 100	11	Sum = sum + B
6	write (B)	12	write (sum)

A = 500

B = 1100

S_1

1 7 8 2 3 9 4 5 6 10 11 12

A 500

A 400

B = 1100

B = 1200

Sum 0

A 500

Sum = 500

B = 1200

B = 1700

Sum = 1700

But sum should be 1600 bcz we are decreasing A & increasing B both by 100.

* Serializable Schedule :

→ Given a set of n-transactions, the schedule will be serializable if following conditions hold.

1. All transactions are correct in the sense that any one of the T is executed by itself on a consistent database, the resulting database will be consistent.

2. Any serial execution of the T is also correct & preserves the consistency of the database. The results obtained are correct.

* ⇒ Precedence Graph :

G (V, E)
- V → set of vertices → the T of the schedule
- E → directed edges

$T_i \rightarrow T_j$

Read → write

write → Read

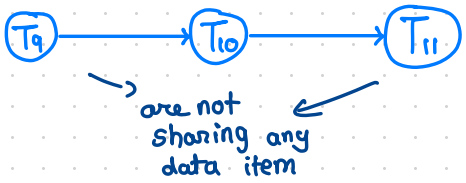
→ A directed edge from Node Ti to Node Tj indicate one of the following condition :

1. T_j performs the operation Read(A) to read the value written by T_i.

2. T_j performs write(A) after T_i performs read(A)

Schedule :	T_9	T_{10}	T_{11}
Read (A)	✓		
A = f_1 (A)	✓		
write (A)	✓		
Read (A)		✓	
A = f_2 (A)		✓	
write (A)		✓	
Read (B)		✓	
B = f_3 (B)		✓	
write (B)		✓	
Read (B)			✓
B = f_4 (B)			✓
write (B)			✓

} 3 vertices



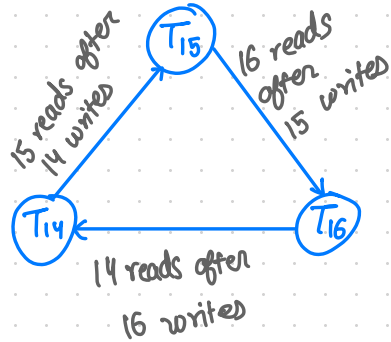
Schedule	T_{12}	T_{13}
Read (A)	✓	
A = f_1(A)	✓	
Read(A)		✓
A = f_2(A)		✓
write(A)		✓
Write (A)	✓	

13 writes after 12 reads

12 writes after 13 reads

↳ Schedule is not serializable (Loop is present)

S	T_{14}	T_{15}	T_{16}
Read (A)	✓		
Read (B)		✓	
A = f_1(A)	✓		
Read (C)			✓
B = f_2(B)		✓	
write (B)		✓	
C = f_3(C)			✓
write (C)			✓
write (A)	✓		
Read (B)			✓
Read (A)		✓	
A = f_4(A)		✓	



Read (c)	✓	
write (A)		✓
C = f_5 (c)	✓	
write (c)	✓	
B = f_6 (B)		✓
write (B)		✓

★ Locking System

Lock

(X) Exclusive Lock : T can update or write the dataitem & No other T can even read it until the T is unlocking it.

↓

Exclusive use of dataitem for 1 transaction.

(S) Shared Lock (Read lock) : can read dataitem & work with it but can't modify the dataitem

T_5	T_6
Lock x (sum)	
Sum = 0	
Lock s (A)	
Read (A)	
Sum = Sum + A	
unlock (A)	
Lock x (A)	

	T_5	T_6
Lock X (sum)		✓
Sum = 0		✓
Lock S (A)		✓
Read (A)		✓
Sum = Sum + A		✓
unlock (A)		✓
Lock x (A)	✓	
Read (A)	✓	
A = A - 100	✓	
write (A)	✓	
Unlock (A)	✓	
Lock x (B)	✓	
Read (B)	✓	
B = B + 100	✓	
write (B)	✓	
Unlock (B)	✓	
Lock x (B)		✓
Read (B)		✓
Sum = Sum + B		✓
write (Sum)		✓
Unlock (B)		✓
Unlock (Sum)		✓

★ **Two Phase Locking** : It has 2 phases

A growing phase where the no. of blocks increases from 0 to a max value for the transaction

&

A contracting phase where the no. of blocks held decreases from max value to zero.

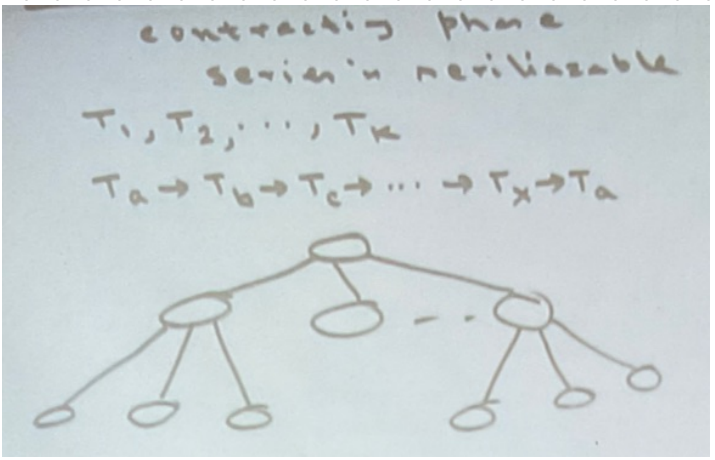
→ Series is serializable if a schedule follows 2 phase locking.

Proof by contradiction : Suppose the series is not serializable.

→ There is a cycle.

$T_a \rightarrow T_b \rightarrow T_c \rightarrow \cdots \rightarrow T_x \rightarrow T_a$

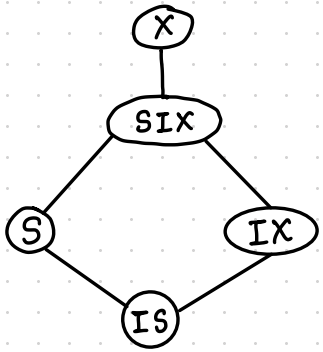
→ Once you lock & unlock then you can't lock again.

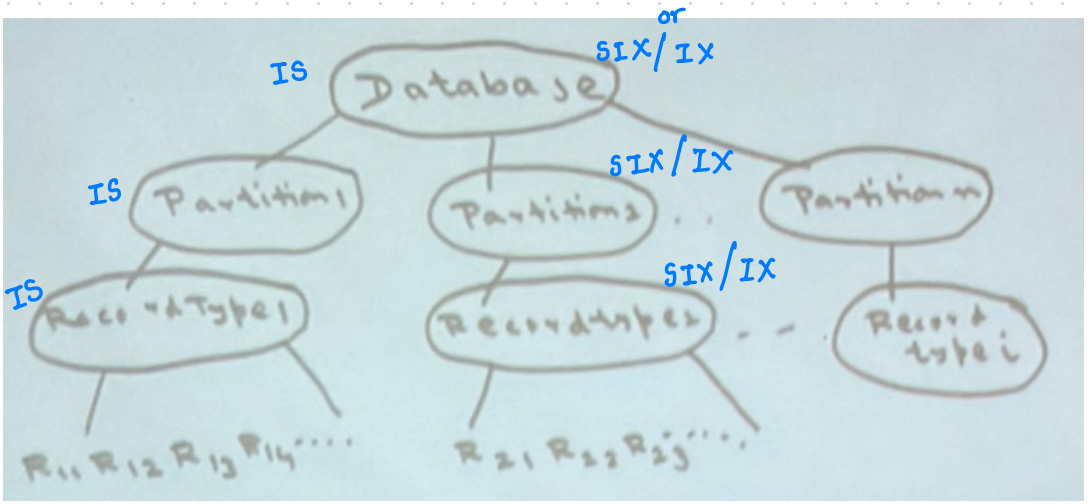


Intention share (IS): The node or its descendents cannot be exclusively locked. The descendent may be locked in S or IS mode.

Intention Exclusive (IX) : the node itself can't be exclusively locked. However the descendents can be locked in any of the locking modes.

Shared and Intention Exclusive (SIX) : Node locked in SIX mode & all the d are implicitly locked in S mode but any of the descendents can be locked in X or IX mode.





R_{13} → retrieve
R_{13} → S

R_{22} → update
R_{22} → X

⭐ Timestamp Based ordering Protocol :

$T_i \rightarrow$ time stamp value t_i

$T_i(t_i)$ & $T_j(t_j)$

s.t. $t_i < t_j$

(t_i: older, t_j: younger)

$X : \{x, W_x, R_x\}$

x : value of X

W_x : the system clock value when the dataitem was written for the last time.

R_x : Read time stamp value

→ A transaction T_a with timestamp value t_a issues a read operation for the data item X with values $\{x, W_x, R_x\}$ (R_x → t_a)

1) This request will be allowed if $t_a \geq W_x$
2) This request will fail if $t_a < W_x$

→ A transaction T_a with timestamp value t_a issues a write operation for the data item X with values $\{x, W_x, R_x\}$ (W_x → t_a)

1) The write operation will be allowed if $t_a \geq W_x$ & $t_a \geq R_x$
2) The write operation will fail if $t_a < R_x$

A younger transaction is already using the value of x. So, t_a is not allowed to modify.

3) $R_x \leq t_a < W_x$, younger T has already updated the value of x. Write operation is ignored → not allowed.

Schedule	T_{22} (t_{22})	T_{23} (t_{23})
Sum = 0	✓	
Read (A)	✓	
Sum = Sum + A	✓	
Sum = 0		✓
Read (A)		✓
A = A − 100		✓
Write (A)		✓
Read (B)	✓	
Sum = Sum + B	✓	
Show (sum)	✓	
Sum = Sum + A		✓
Read (B)		✓
B = B + 100		✓
Write (B)		✓
Sum = Sum + B		✓
show (sum)		✓

Initially ⇒ A : 400, W_a, R_a B : 500, W_b, R_b

Step 2 : A : 400, W_a, t_{22} "

Step 5 : A : 400, W_a, t_{23} "

Read write operations

Step 7 : A : 300, t_{23}, t_{23} "

Step 8 : " B : 500, w_b, t_{22}

Step 10 : Sum = 900

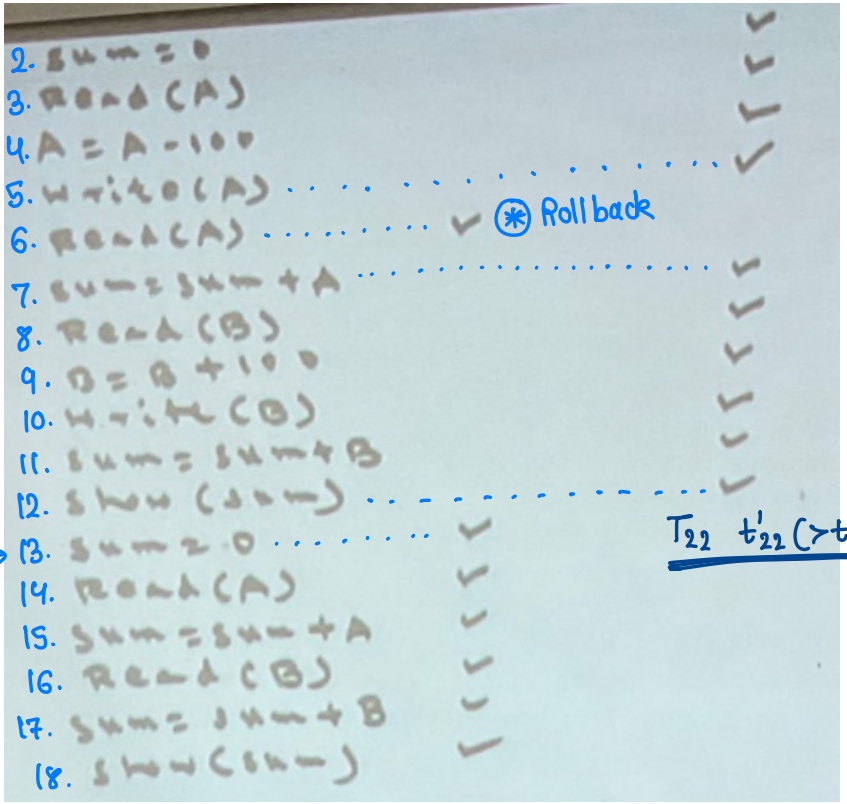
Step 12 : B : 500, w_b, t_{23}

Step 14 : B : 600, t_{23}, t_{23}

Step 16 : Sum = 900

Schedule	T_{22} (t_{22})	T_{23} (t_{23})
Sum = 0		
Read (A)		
Sum = Sum + A		
Sum = 0		
Read (A)		
A = A - 100		
write (A)		
Read (B)		
Sum = Sum + B		
Show (sum)		
Sum = Sum + A		
Read (B)		
B = B + 100		
write (B)		
Sum = Sum + B		
show (Sum)		

1. Sum = 0



	A : 400, W_a, R_a	B : 500, W_b, R_b
Step 3	A : 400, W_a, t_{23}	"
Step 5	A : 300, t_{23}, t_{23}	"
Step 6 →	Not allowed	

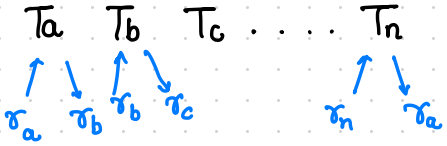
Deadlock ⇒ Next class

who missed mid-sem test on 7th April from 10-12

mail to prof, your Roll No.

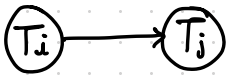
No other test.

★ Deadlock and its Resolution:

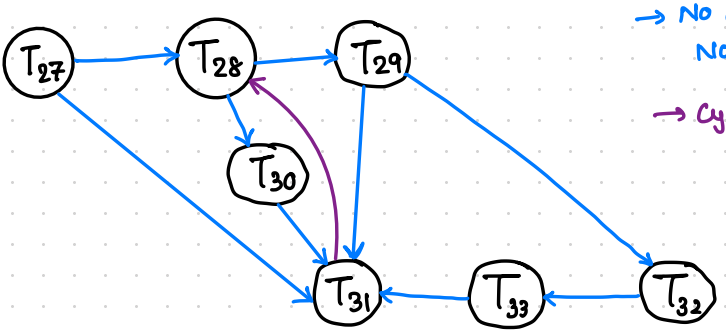


Wait for graph :

T-id	Items-locked	Items waiting for
T_{27}	B	C, A
T_{28}	C, M	H, G
T_{29}	H	D, E
T_{30}	G	A
T_{31}	A, E	C* ← added later
T_{32}	D, I	F
T_{33}	F	E



→ Ti is waiting for the data-item currently allocated to and held by Tj.



→ No cycle
No deadlock

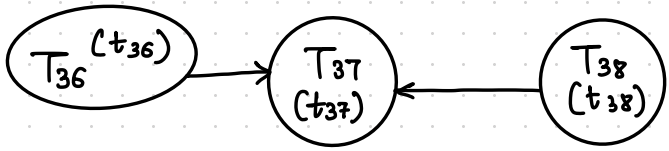
→ Cycle present
Deadlock occurs.

Wait - Die :

Wait & Die scheme for Deadlock is as follows:

1) If the requesting transaction is older than the transaction that holds the law on the requested data item. The requesting T is allowed to wait.

2.) If the requesting T is younger than the T that holds the law on the requested data-item. The requesting T is aborted and rolled back.



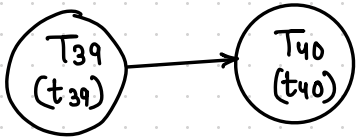
$t_{36} < t_{37} < t_{38}$

T_{36} → allowed to wait
T_{38} → Roll back

★ Wound -wait

1. If a younger T holds a data item requested by older one, younger T will be aborted and rolled back.

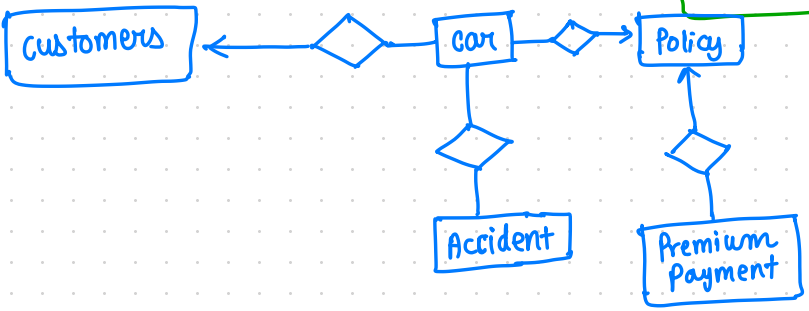
2. If a younger T requests a data item held by older T, the younger T is allowed to wait.



$t_{39} < t_{40}$ → aborted & rolled back.

midsem solution

7th → 11 to 1
MA seminar room
check midsem paper.
No other time

① 

② $F = \{AB \to C,\ AB \to D,\ C \to A,\ D \to B\}$

AB, BC, CD, AD

3NF ✓ & Not BCNF

No BCNF decomposition preserving dependencies

③ R & F

$\{A \to B,\ C \to D,\ B \to C\}$

$R_1 = \{(A,B), (C,D), (B,C)\}$

$R_2 = \{(A,B), (C,D), (A,C)\}$

$R_3 = \{(B,C), (A,D), (A,B)\}$

(ii)

A	B	C	D	E
p	2	3	4	5
2	p	3	7	5
p	2	3	4	6

$BC \to D$ violates

$\Rightarrow p = 2$

$BC \to\!\!\!\!\to D$ violates

$\Rightarrow p \neq 2$

(4) (i) YES - 1 M
proper justification for loss less join - 1 M } 2M.

(ii) BCNF → 3NF
implies
logic

(iii) $R \div S = \Pi_{A-B}(R) - \Pi_{A-B}[\Pi_{A-B}(R) \times S - R]$ 1M

example: 1 M

(5) (i) user, Borrow

(ii) Borrow, Supplier, Supply

(iii) Supply, Book

(6) (i) Loan

(ii) All

(iii) Loan, Borrower

11 AM | Apr 10

Distributed Database

Col^n of multiple logically interconned database on a network, and S/w used here is

DDBMS

Shared Memory Architecture

Share the same secondary disk & primary disk

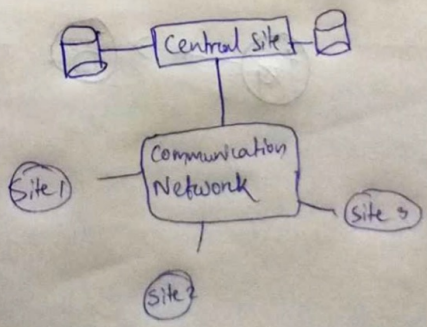
Shared disk architecture

Multiple processor share the same sec disk but they have their own primary disk

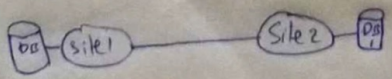
Share nothing architecture

Multiple units have individual CPU, Disk, Database

Centralised Database ~~Architecture~~ Network Architecture



Truely Distribute



Advantages of Distributed Database

↳ Managmnt of Dist. Data with different levels of transparency.

~~Where we're not disclosing where data is stored~~

we're not disclosing, where the data is stored.

↳ Dist. or net. transparency.

this refers to fillers for the user for the Operational detail.

(i) location transparency → query doesn't depend on Loc^n

(ii) Naming transparency → Once a name is specified, the named object can be accessed unambiguously without additional

↳ Replication transparency.

Makes the user unaware of existence of multiple copies.

+Better availability
+ Performance
+ Reliability

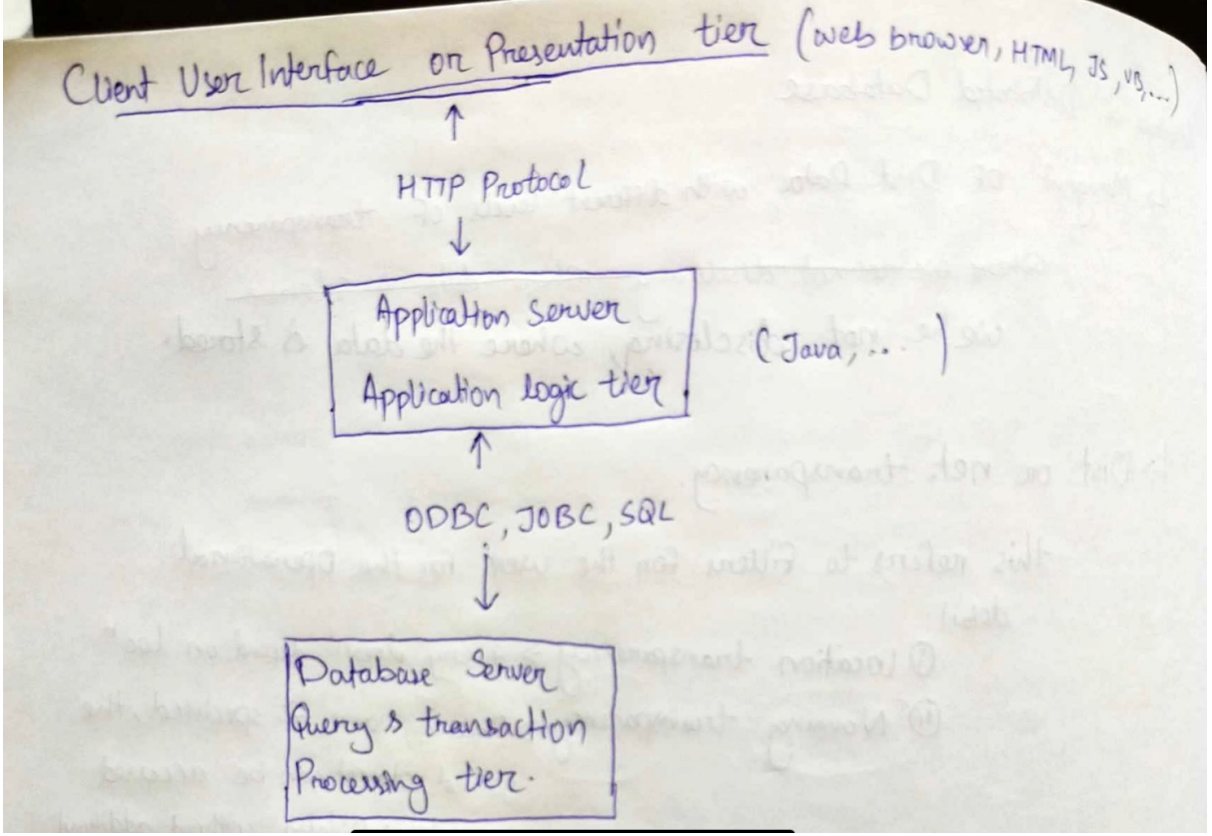
↳ Fragmentation transparency

→ Horizontal Fragmentation
→ Vertical "
→ Mixed.

↳ Increased reliability and availability.

~~↳~~ System is available on time.

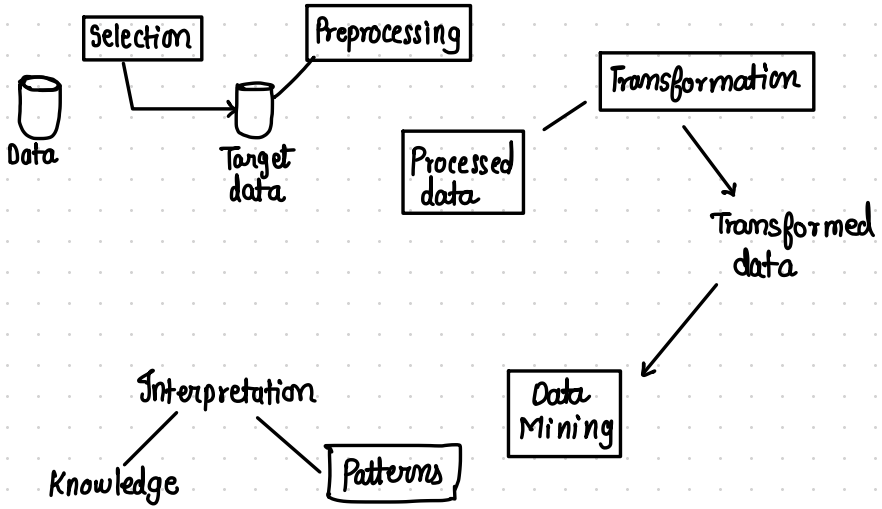
↳ Improved performance
↳ Easy expansion.



★ **Data Mining** is a part of Knowledge discovery in Databases (KDD)

→ Collect non-trivial, relevant data from a huge database.
or
discover

→ detect frauds in bank, etc.



- Association Rule Mining → Mall : you purchase 1 thing, and another put aside it.
- Classification → A person of 20yr age & having good credit points → he will buy a computer or not.
- Clustering / Similarity Matching
  - ↳ predicting where should KFC open its new outlet in KGP.

TID	Item
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Itemset

Itemset is a collection of one or more items.

↳ {Bread}, {Milk, Bread}

K- Itemset

An itemset which contain K-items.

Support Count (σ)

Frequency of occurring of an itemset in a set of Transactions.

σ {Milk, Beer, Diaper} = 2

Support : Fraction of Transactions that contain an itemset.

s {Milk, Beer, Diaper} = $\frac{2}{5}$

Frequent Item sets : An itemset whose support is $\geq$ Minimum Support Threshold

Association Rule Discovery : {Diaper} $\longrightarrow$ {Beer}

{Milk, Bread} $\longrightarrow$ {Coke}

d unique items,

Total no. of itemsets $= 2^d$

Total no. of possible association rules $= 3^d - 2^{d+1} + 1$

★ Apriori Algorithm (Late 90's)

Frequent itemset

Any subset of frequent itemset must be frequent.

L_K : Set of candidate K-itemsets

L_{K-1}

Join Step : C_K is generated by joining L_{K-1} with itself.

C_K ↓ Candidate itemset of size K

possible itemsets if you join K items together.

L_K => frequent itemset of size K.

from C_K I will obtain L_K.

Prune step : Any (K-1) itemset that is not frequent cannot be a subset of frequent K itemset.

L_1 : { frequent items } ;

for ($K=1$; $L_K \neq \emptyset$; $K++$) do begin

C_{K+1} = candidates generated from L_K ;

for each transaction t in database DO

increment the count of all candidates in C_{K+1} that are contained in t

b_{K+1} = Candidates in C_{K+1} with a min-support.

end

return $\cup_K L_K$;

TID	Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

min. support $= \frac{2}{9} = 22\%$

Minimum confidence $= 70\%$

S1

IT	SC
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_1

IT	SC
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

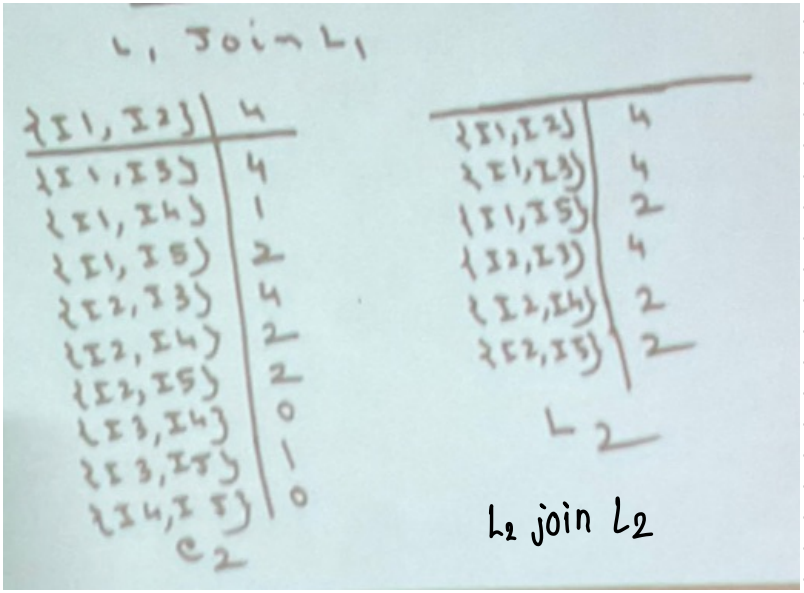
L_1

S2

L_1 join L_1

$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2
$\{I_3, I_4\}$	0
$\{I_3, I_5\}$	1
$\{I_4, I_5\}$	0

$\{I_1, I_2\}$	4
$\{I_1, I_3\}$	4
$\{I_1, I_4\}$	1
$\{I_1, I_5\}$	2
$\{I_2, I_3\}$	4
$\{I_2, I_4\}$	2
$\{I_2, I_5\}$	2
$\{I_3, I_4\}$	0
$\{I_3, I_5\}$	1
$\{I_4, I_5\}$	0



(S3)

$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

C_3

$\{I_1, I_2, I_3\}$	2
$\{I_1, I_2, I_5\}$	2

L_3

(S4) C_4 : L_3 join L_3

I_1, I_2, I_3, I_5 ✗ can't be frequent

$C_4 = \phi$

STOP

$L = \{ \{I_1\}, \{I_2\}, \{I_3\}, \{I_4\}, \{I_5\},$
$\{I_1, I_2\}, \{I_1, I_3\}, \{I_1, I_5\}, \{I_2, I_3\}, \{I_2, I_4\}, \{I_2, I_5\},$
$\{I_1, I_2, I_3\}, \{I_1, I_2, I_5\} \}$

union of all L's

Association Rule

I S is subset of I

$S \rightarrow (I - S)$ if

$$\frac{\text{Support - count } (I)}{\text{Support - count } (S)} \geq \text{min. confidence}$$

$\{I_1, I_2, I_5\}$

$\{I_1, I_2\}, \{I_1, I_5\}, \{I_2, I_5\}, \{I_1\}, \{I_2\}, \{I_5\}$ } possible S-values

R1 : $I_1, I_2 \rightarrow I_5$ $\frac{sc\{I_1, I_2, I_5\}}{sc\{I_1, I_2\}} = \frac{2}{4} = 50\%$

Reject

R2 : $I_1, I_5 \rightarrow I_2$ $\frac{sc\{I_1, I_2, I_5\}}{sc\{I_1, I_5\}} = \frac{2}{2} = 100\%$

R_2 is selected

.....