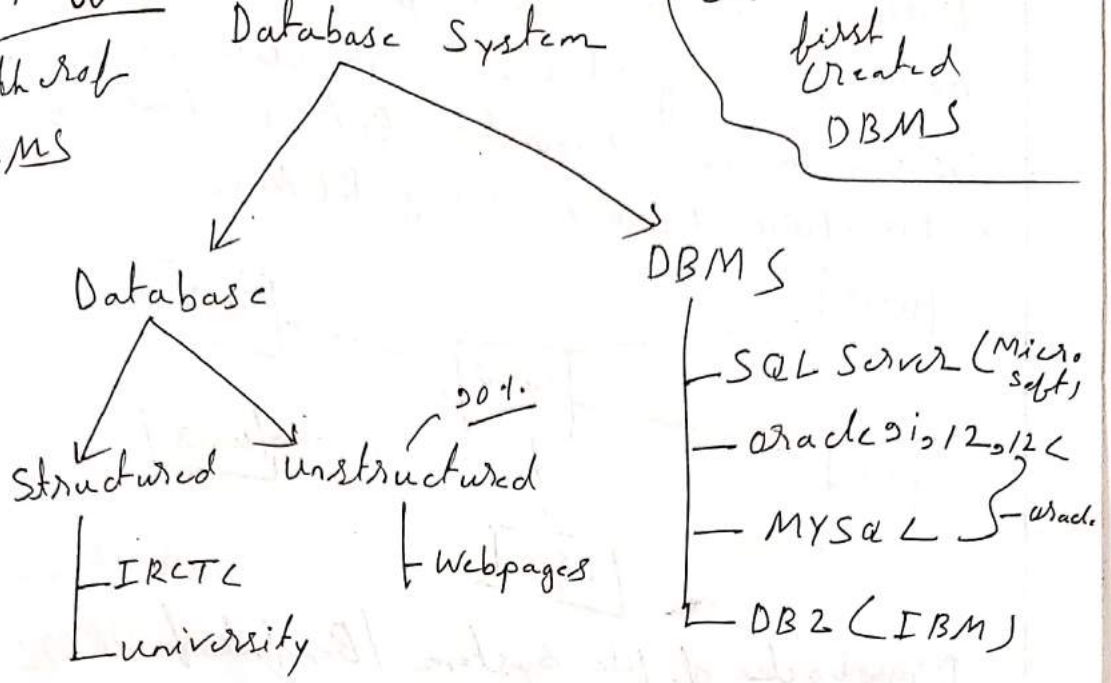
oldest network DBMS

DBMS From Gatesmashers

E. F. Codd
Father of DBMS

Charls Bachman first created DBMS



Database: a collection of related data which represents some aspect of real world.

DBMS (Database Management System): is a software for storing and retrieving data. It is used to do the operation like (insertion, updation, deletion).

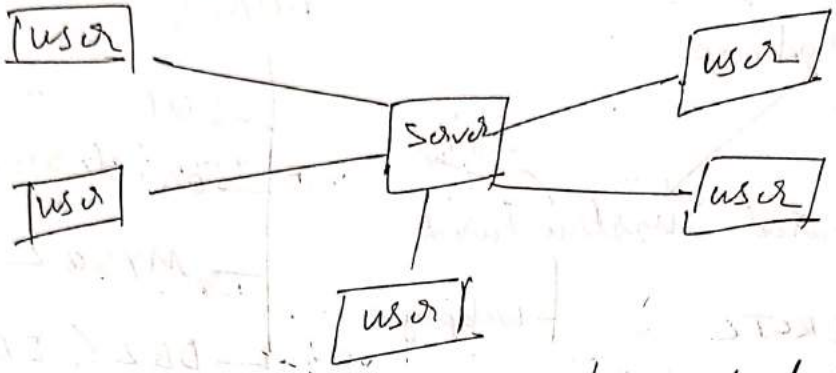
- Structure data is stored in a particular structure. Most used is Relation DBMS. It stores the data in form of table.
- Unstructure data has no structure. Web pages has no specific structure like there may be different types of photos, videos. Suppose there may be photos, videos, text in webpage first.

Nowadays 90% data are unstructure.

Ex- Big data, Hadoop.

File System & DBMS ⇒

Earlier we use ~~so on~~ file system for storing and accessing data. Though still we use file system in our computer. But in client ~~struct~~ server architecture we are using DBMS.



Drawback of file system / Benifit of DBMS:

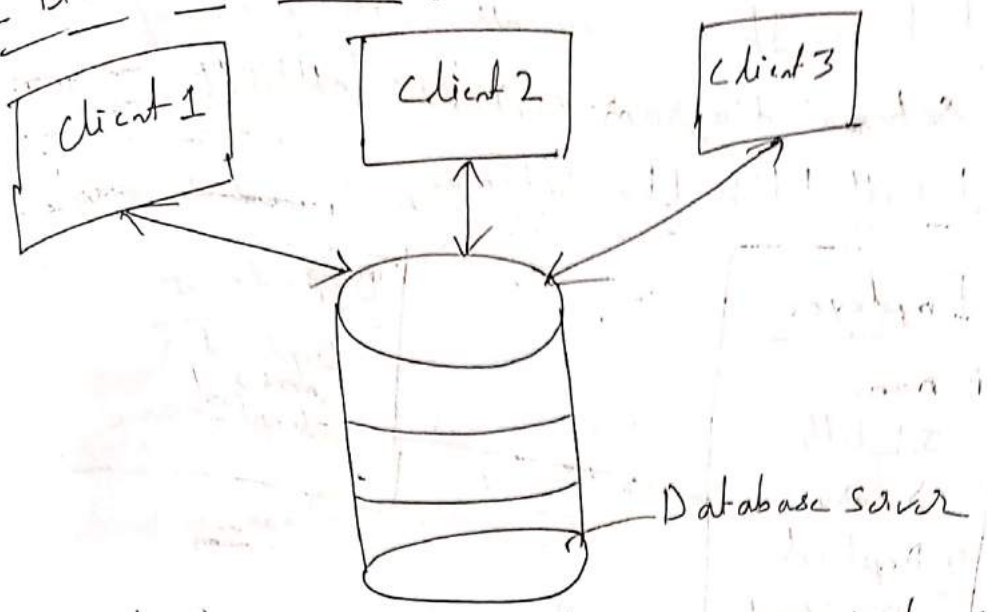
(i) **Difficulty**: If we want just 1KB data (train detail) but it was stored in a big file. In file system we will get the whole file (5 GB) (whole train details). It will take a lot of time as well as memory.

(ii) **Attributes**: In file system we have to know file name as well as location. DBMS, we just have to write a query.

(iii) **Concurrency**: In file system concurrent access is difficult. But DBMS helps concurrent access by multiple users. DBMS has protocals for it. Multiple user booking ticket

(iv) **Security**: File system does not provide roll based system. DBMS gives roll based access control. It enhances the security (user, admin, student, faculty)

2 tier Architecture :



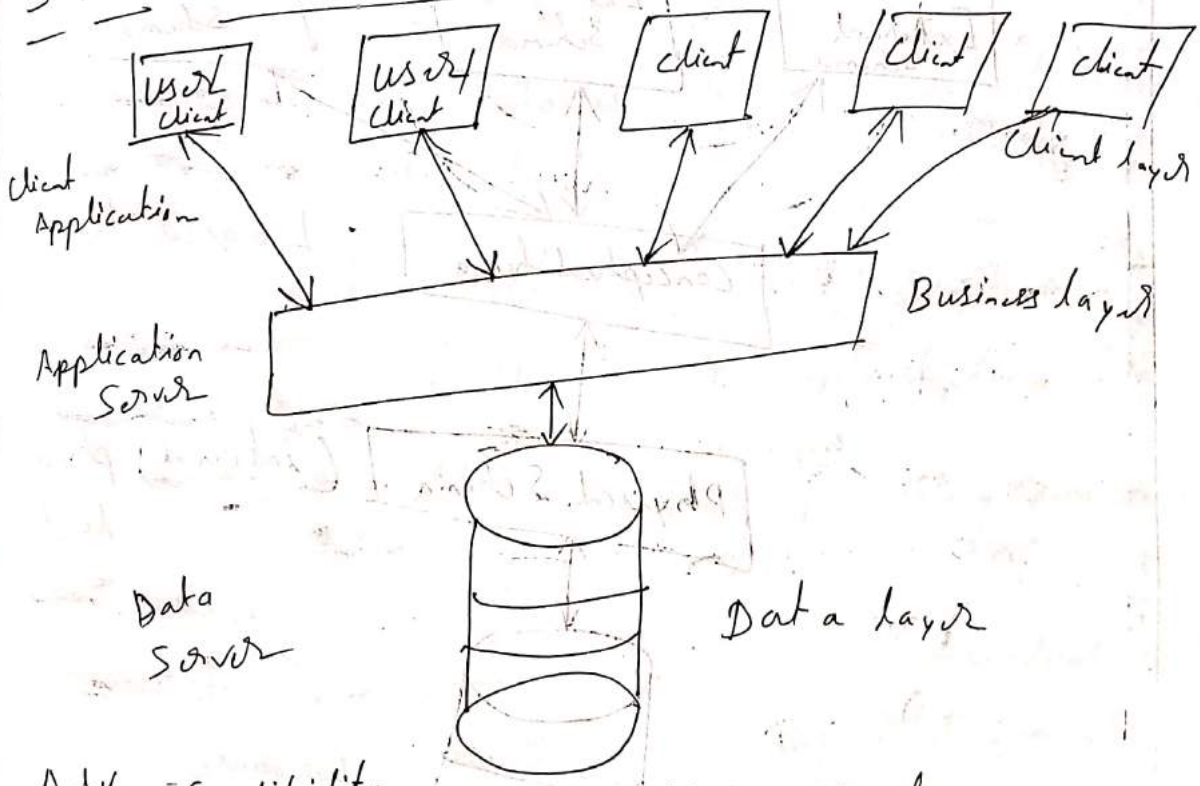
client have an API (JDBC, ODBC) to connect to the database
Interface

Adv ⇒ Maintanence low.

Dis :- (i) Scalability not possible
(ii) Security compromised

uses → • Reservation booking in counter
• Bank Physically money deposit or withdraw via form.

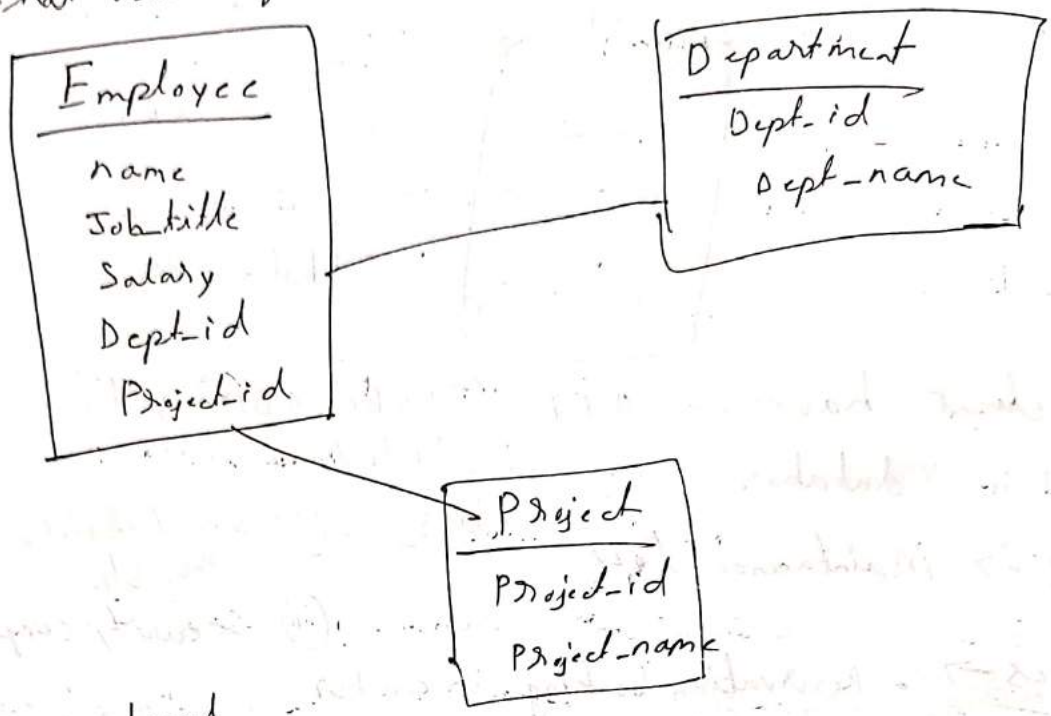
3 tier Architecture :



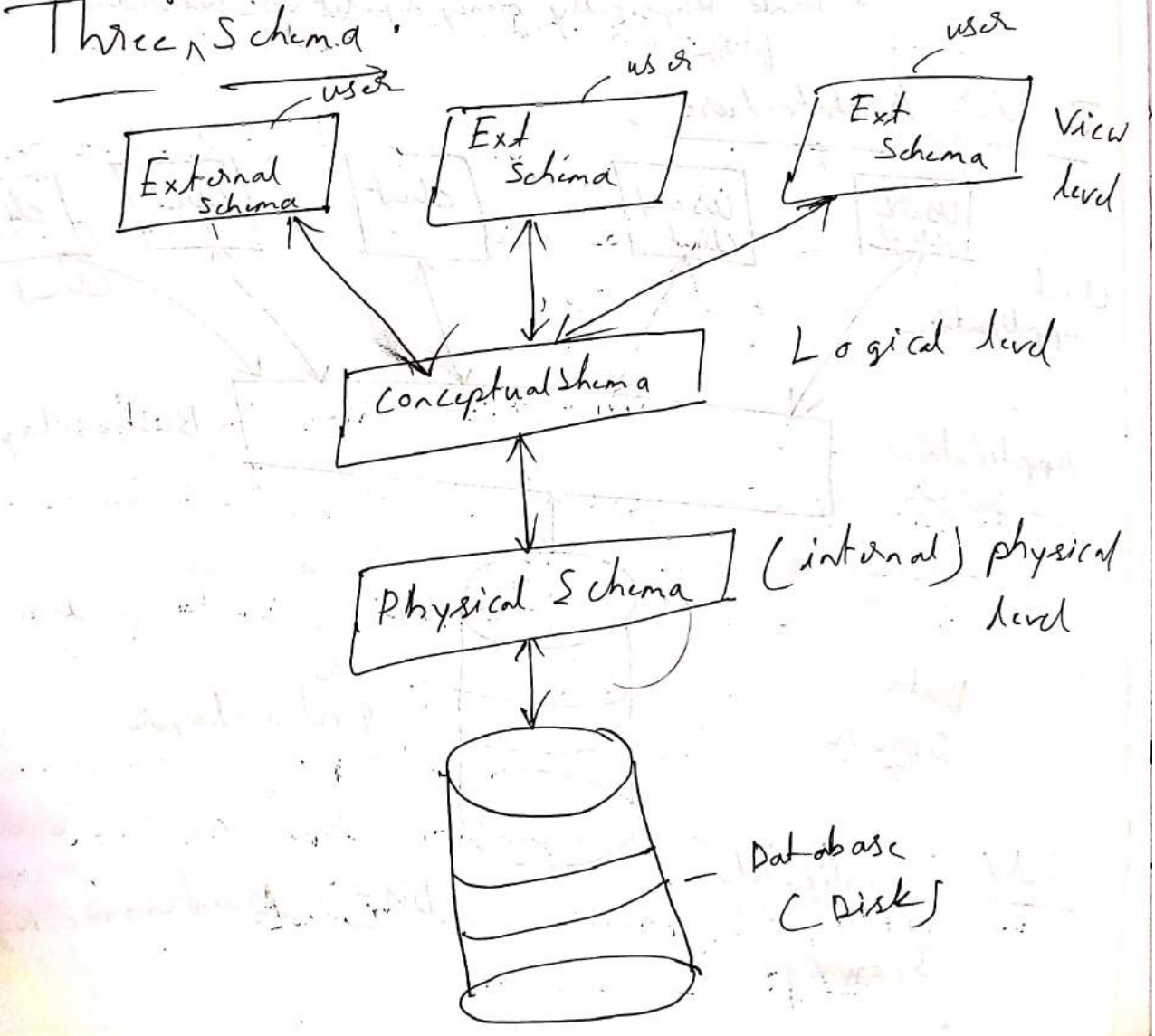
Adv - Scalibility
Security

Dis- Maintenance high

Schema: It is a logical Representation of database. It gives overall description of database. A schema diagram contains attributes, entities that will define the schema.



Three Schema level



main motive is to data independence (user and database should not connect directly). user does not know their data where is stored or how

- External schema is like how user will view their data. Student, Faculty, Dean has separate view.
- Conceptual/logical schema define the design of database. We define entities, attributes & their relationship. ER diagram is its example
- Physical schema tells how the data will be stored on storage device. The data can be stored in form of files.

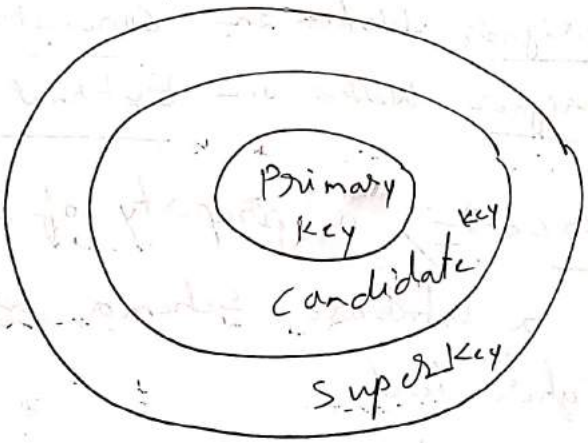
(*) Database administrater handle physical schema
Database designer Works on conceptual schema
Interface designer Works on External schema.

Data independence :⇒ a property of DBMS that helps to change a database schema without changing a schema at a higher level.

- Physical data independence: changing physical structure will not affect on conceptual level. This is achieved by mapping.
  Ex - . using new storage device . Switching data structure
  . change location of data.
- Logical data independence: changing conceptual schema without any effect on view level. achieved by mapping.
  Ex- . Add/modify/delete new attribute
  . Merging two record.

Keys :⇒ Key is a set of attributes that can identify each tuple uniquely in a table.

- **Candidate Key**: A set of minimal attribute that can identify each tuple uniquely in the table.
  A & A B both identify tuple uniquely than A will be candidate key, A B will be super key.
- **Primary Key**: A candidate key that the database designer selects while designing the database.
  - Primary keys are unique and not null.
  - (*) Only 1 primary key is possible made of attribute or set of attribute
- **Alternate key**: Candidate keys that are left unused after implementing the primary key are called Alternate key.



Candidate key in student table:

- Adhar card
- Roll no] — Primary key
- Reg no]
- Voter id
- Phone no
- Pan no
- Email id

Alternate key

- Primary key always provided by the system.

Foreign key :- It is an attribute or set of attributes that references to primary key of same table or another table (relation)

- It maintain referential integrity

Student

Roll_no	name	address
1	Ani	kolkata
2	Ani	Delhi
3	Jhili	Andul

Course

course id	course name	Roll no
C_1	CS	1
C_2	maths	2
C_3	nurse	3

foreign key

referencing table

Referenced table

primary key

name may not be same

- There may be more than one foreign key.

creation - Create table course

{ course_id varchar(10),
course_name varchar(20),
Roll_no int references
Student(Roll_no)

}

if we make foreign key in already created table

Alter table course
ADD constraint fk
foreign key (Roll_no)
References Student(Roll_no)

Referential Integrity of Foreign key :=>

integrity - being honest

- Referential integrity is a constraint applied on foreign key which requires the foreign key to have a matching primary key which makes sure reference from a row in one table to another table is valid.
- We will get error if we violate referential integrity.

Reference table

(i) insert : no violation

(ii) Delete : May cause ~~deletion~~ violation
(if we delete a data which is not available in foreign key so no problem. but if it is in foreign key then it will violate)

Solution :=>

- on delete cascade (automatically all the rows which were taking referenced will also deleted)
- on delete set NULL (automatically all throws which were taking reference will be null)

The problem occurs when the foreign key is also primary key of table then nullify is not possible

- On delete no action : It is by default

It will give error.

(iii) update : may cause violation

Solution – on ~~d~~update cascade
on update reaction

Referencing table

(i) insert – may cause violation

(ii) deletion – no violation

(iii) updation → may cause violation (if we update foreign key value)

Q | Let $R_1(a, b, c)$ and $R_2(x, y, z)$ be two relation in which 'a' is foreign key in R_1 that refers to the primary key of R_2. Consider a option – (a) Insert into R_1 (b) Insert into R_2 (c) Delete from R_1 (d) Delete from R_2 Which is true about referential integrity

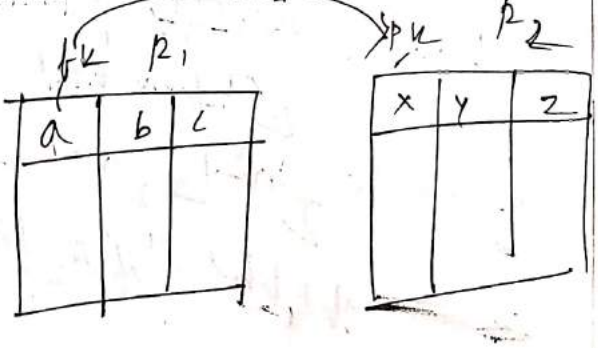
(a) option a & b will cause violation

(b) b & c will cause violation

(c) c & d will cause violation

(d) a & d will cause violation

⇒ (d) ✓



Super key ⇒ Super key is a combination of all possible attribute which can uniquely identify two tuple in a table.

Super set of any candidate key is super key.

Rollno	name	age

- Super key must have one candidate key

Rollno → candidate key — super key

Rollno, name → Super key

Rollno, age → Super key

Rollno, name, age — Super key

name, age — ✗ Super key

(*) if R($A_1, A_2 \dots A_n$) then how many Super key

→ if A^1 is candidate key then Super key possible 2^{n-1} (A_1 A_2 - - A_n ; fixed 2 ... 2 possibility

→ if A_1, A_2 both candidate key

A_1 A_2 A_3 - - A_n

fixed fixed

$2^{n-1} + 2^{n-1} - 2^{n-2}$

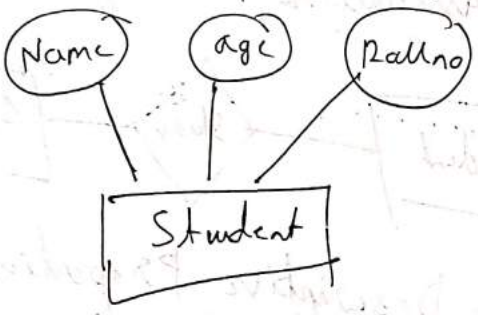
if A_1 fixed, if A_2 fixed, common part

→ if $A_1 A_2$ & $A_3 A_4$ candidate key then
$2^{n-2} + 2^{n-2} - 2^{n-4}$

Introduction to ER Model :=>

ER diagram or Entity relationship model is a conceptual model that gives the graphical representation of the logical structure of database.

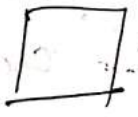
- Before doing house we make design
- It shows all the constraints and relationship that exist among the different components
- ER diagram mainly composed of Entity sets



Attribute set
Relationship set

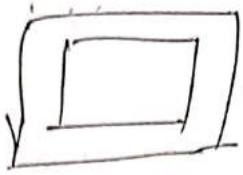
Entity :=> a real world object with an existance
In a college database student, Professor, course is an entity

- Strong entity : if it has a primary key.
  Represented by single rectangle

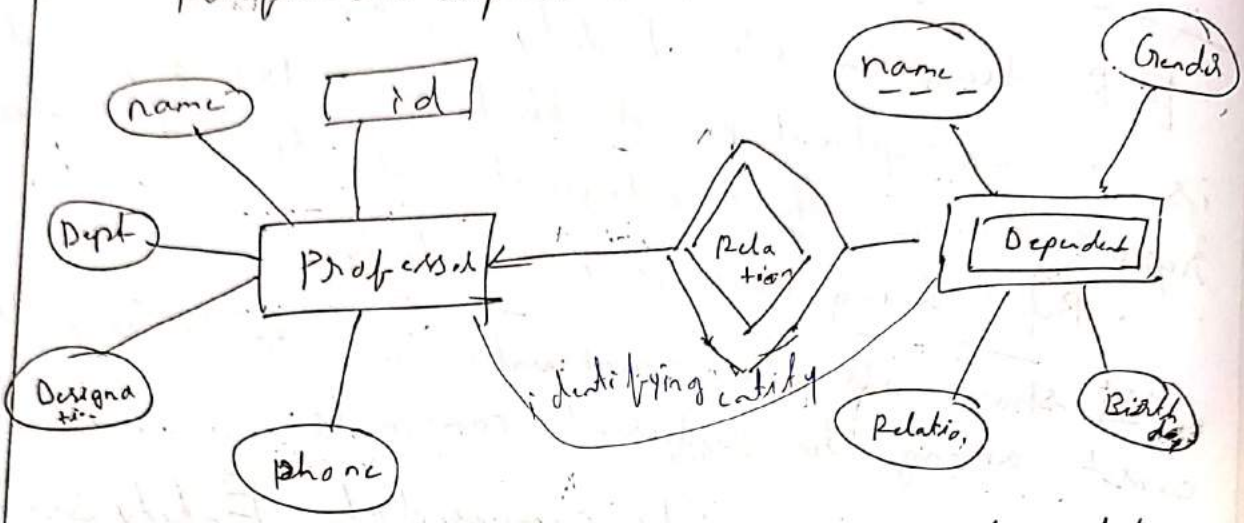


Professor is a strong entity, it has a professor id as primary key.

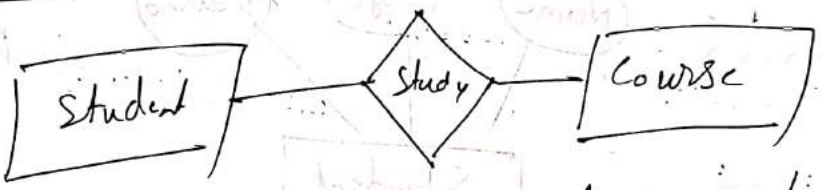
- Weak entity : Do not have Primary key. Dependent on strong entity. Represented by double rectangle



professor- dependent is weak entity.



Relationship :⇒ Association among several entities



Attributes :⇒ Descriptive properties which are owned by entity.

like- Age, name, class is attribute of students

- Single attribute : attribute which can take only one value

ex- Age is single valued (we can't write two different age.

- Multivalued : attribute which can take multiple value. ex- Mobile no can take multiple values

- Simple: attribute which can not be divide
  Ex- Age is simple
- Composite: attribute which can be divided further
  Ex- Name can be divided first-name, middle name, last name.
- Stored: attribute which can not be derived.
  ex- DOB is stored
- Derived: attribute which can be derived by other attribute
  ex- Age can be derived from DOB.
  dotted ellipse
- Key: attribute which can identify an entity in unique way.
  ex- Rollno is key attribute.
  Rollno - underline
- non key: attribute which is not key attribute.
- Required: mandatory value *Mobile / *Name
- optional: - Not mandatory
- complex - Composite + multivalued
  2 Residential address with 2 diff number

Degree of Relationship (cardinality) :=>

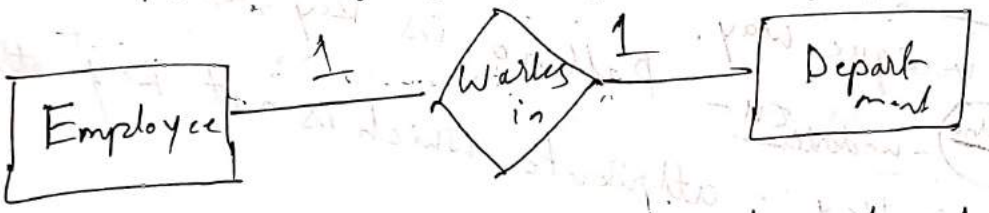
maximum number of Relationship instances in which an entity can participate.

(i) one to one (ii) one to many
(iii) Many to one (iv) Many to many

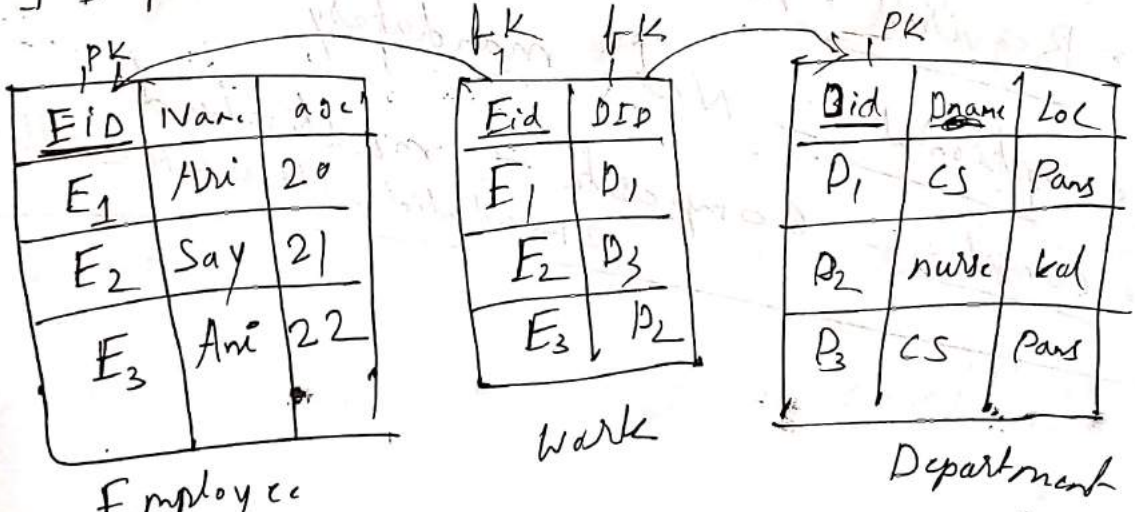
(i) one to one :=> An entity in Set A can be associated with atmost 1 entity in Set B and vice versa.

A record in one entity is associated with a record in other entity and vice versa.

ex- country – capital
person – fingerprint



1 Employee can works in only 1 department
1 Department can have 1 Employee only



- In work there will be minimum 2 attribute.
- Both can be primary key.

PK = Either E_ID or D_ID

• Can we reduce the tables

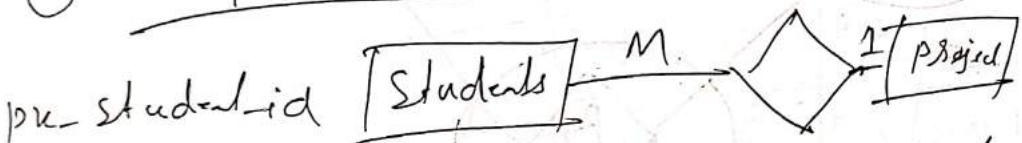
=> Yes we can make 2 table from 3.

If Eid is primary key then we will just merge the Employee & work.

(ii) One to many :=> PK will be project_id

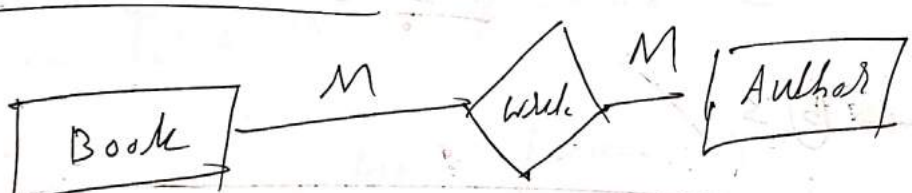
• can Reduce from 3 to 2 table also 1 student many project

(iii) Many to one



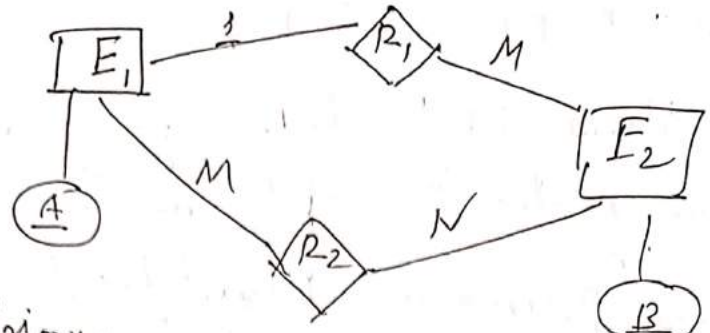
many students 1 project

(iv) Many to many



PK will be combinid of bookid & Authorid

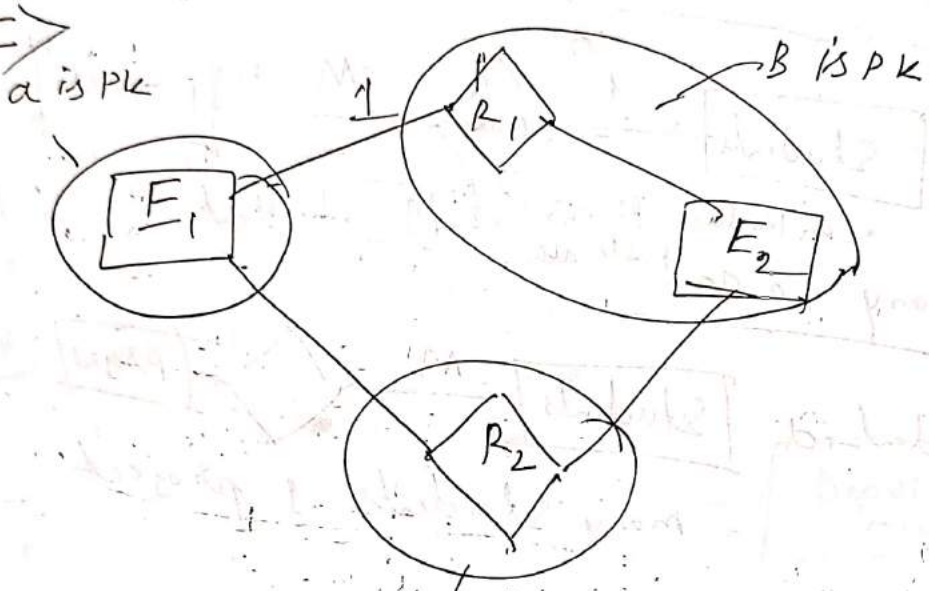
Q1



minimum no of tables required to represent this E-R model into Relational model?

(a) 2 (b) 3 (c) 4 (d) 5

⇒



→ (b) 3 ✓

Normalization :⇒ Process for making the data consistent by

- Reducing ~~Red~~ as removing Redundancies (similar value)
- Ensuring the integrity of data.

• Row level duplicacy:

Roll	name	Age
1	Ari	20
2	Ani	22
1	Ari	20

We remove row level duplicacy with the help of Primary key, we can make Roll primary key

• Column level Duplicacy:

Sid	Sname	Cid	Cname	FID	Fname	Salary
1	Ari	C_1	DBMS	F_1	Bob	30000
2	Ani	C_2	Java	F_2	Don	40000
3	Paltu	C_1	DBMS	F_1	Bob	3000
4	Hari	C_1	DBMS	F_1	Bob	30000

— just a small example

Colum 1, 3, 4 is copy of each other except Sid 4 Sname.

This type of duplicacy form major 3 Problem

(i) insertion anomly (problem occured on specific case)

We can easily insert details of Student (Sid, Sname) but we can't just only add any details of faculty & course as we have no student id (but as it is PK it can't be null)

(ii) Deletion anomly: We can delete details of 1 or 3 or 4 but if we delete 2 (Sid) then we will lost the details of that particular course & faculty details. as it was available only 1 time in database.

(iii) updation anomly :=> We can update the sname but if we try to salary of bob then it will update the salary 3 times. (as the data is written 3 times.) time consuming

Simple Solution : divide the table into 3 tables

Sid	Sname

Cid	Cname

Eid	Fname	Salary

First Normal Form (1NF) :=>

- Each cell of the table contains only an atomic value. (either single or a null value)

Roll (PK)	Name	course
1	Ari	C/C++
2	Ani	Java
3	Shili	C/DBMS

is not in 1NF as it contain multiple value

Solution - We can represent this in 3 way to make it 1NF.

Solution 1 :=>

Roll	name	course
1	Ari	C
1	Ari	C++
2	Ani	java
3	shili	C
3	Shili	DBMS

Roll + course / combined PK

Solution 2

Roll (PK)	name	Course1	course2
1	Ari	C	C++
2	Ani	Java	null (empty)
3	jhili	C	DBMS

null (not 0) — 0 is different

But the problem is when only a student is enrolled a lot of course & other are enrolled only 1. So There will be so much null. (a not good Representation)

Solution 3 : Divide table into 2

Base table / Reference table

Roll no (PK)	name
1	Ari
2	Ani
3	jhili

Roll no (FK)	course
1	C
1	C++
2	java
3	C
3	DBMS

~~no PK~~

Roll no + course — PK

Closure Method

If we want to find candidate key we use closure method

(i) $R(ABCD)$

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

meaning of A closure means what A can determine

$A^+ = ABCD$

As B also determine C then we can say A also determine C

C again determine D

A will determine itself Also

∴ So A is determining all the attribute then A is the candidate key

$B^+ = BCD$ ✗

$C^+ = CD$ ✗

$D^+ = D$ ✗

$AB^+ = ABCD$

AB determine AB

B also determine C

$CK = \{A\}$

(ii) Here AB is not candidate key as it is not minimal. AB is Super key.

prime attribute = {A}

non prime attribute = {B, C, D}

(ii) $R(ABCD)$

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

$\Rightarrow A^+ = ABCD$

$B^+ = BCDA$

$C^+ = CDAB$

$D^+ = ABCD$

$CK = \{A, B, C, D\}$

Prime attribute are the attribute that makes candidate key

Here prime attributes are $\{A, B, C, D\}$

(iii) $R(ABCDE)$

$FD = \{A \to B, BC \to D, E \to C, D \to A\}$

⇒ Short trick - the attributes that are on right sides is determined

Here B, D, C, A is being determined

So we can conclude that each candidate key must contain E (as E can be determined by itself)

✓ $(AE)^+ = ABE = ABEC = ABCDE$

✓ $(BE)^+ = BE = BEC = BECD = BECDA$

↳ Suppose we get AE is candidate then we can check if A is in right hand side of any FD

Here $D \to A$ - right hand

$\therefore (DE)^+ = ABCDE$ ✓

$(CE)^+ = CE = EC$ ✗

$CK = \{A, E, DE, BE\}$

∴ Prime attribute = $\{A, B, D, E\}$

non prime attribute = $\{C\}$

Function Dependency :=>

Determinate

$X \longrightarrow Y$ — dependend

it says X determines Y

or

Y is dependent / determined by X

Ex-

Sid	name
1	Anisha
2	Anisha

We have a confusion that whether there is 2 name are same people, we will check Sid. If Sid different then both Anisha different else if Sid same both are same person

Sid	name
1	Ani
1	Ami

✗ not valid as Sid is determinate and it says same student but name says there is 2 student.

Two type of FD

(i) trivial FD :- $X \longrightarrow Y$ is said to be trivial if $Y \subseteq X$

$\{Sid, Sname\} \longrightarrow Sid$

LHS ∩ RHS $\neq \phi$

(ii) non trivial: $X \to Y$ is said to be non trivial

if $Y \not\subseteq X$

$LHS \cap RHS = \phi$

Properties of FD

(i) Reflexivity: if $Y \subseteq X$ then $X \to Y$

also. $X \to Y$ $\quad$ $Sid \to Sid$

(ii) Augmentation: if $X \to Y$ then $XZ \to YZ$

if $Sid \to Sname$ then $\{Sid, phone\} \to \{Sname, phone\}$

(iii) Transitive: if $X \to Y$ & $Y \to Z$ then

$X \to Z$

Ex- $Sid \to Sname$ and $Sname \to Scity$ then

$Sid \to Scity$

(iv) Union :- if $X \to Y$ & $X \to Z$ then $X \to YZ$

(v) Decomposition :- if $X \to YZ$ then $X \to Y$ & $X \to Z$

(x) $XY \to Z$ then $X \to Z$ & $Y \to Z$ XX not valid

(vi) pseudo transitive :- $X \to Y$ & $WY \to Z$ then

$WX \to Z$

(vii) Composition - $X \to Y$ & $Z \to W$ then.

$XZ \to YW$

Second normal form (2NF) ⇒

- Relation / table must be in First normal form (1NF)
- No partial dependency should not be there

mean —

∴ (All the non prime attribute fully only functional dependent

Proper subset of candidate key → non prime attribute

↑ partial dependency)

Eliminates hidden dependency

$R = \{A, B, C, D\}$

Suppose $CK = \{AB\}$

Prime attribute = {A, B}

non prime attribute = {C, D}

if $AB \to D$, $AB \to C$ } ✓ fully dependent

$A \to C$
$B \to D$
$A \to D$
$B \to C$ } — partial dependency

a] $R = (A, B, C, D, E, F)$

$FD = \{C \to F, E \to A, EC \to D, A \to B\}$

⇒ In order to check CK we will check right hand side, we notice EC is not determined

So in CK there must EC

✓ $EC^+ = EC = ECA = ECAD = ECADB$

$= ECADBF$

Now we check whether E/C is available in right hand or not. In this case there is not available.

∴ So $CK = \{EC\}$

∴ Prime attribute $= \{E, C\}$

non prime attribute $= \{A, B, D, F\}$

Now we will check whether partial dependency exist or not

$C \rightarrow F$ ← non prime attribute

// Subset of candidate key

partial dependency ✓

So this is not in 2NF

if first is not partial dependency then

Third Normal Form (3NF)

- table must be in ~~2nd~~ 2NF
- No transitive Dependency

(*) to be in 3NF LHS should be super key / candidate key Or RHS is prime attribute

(non prime → non prime is transitive Dependency)

a] $R[ABCD]$ FD: $AB \rightarrow C, C \rightarrow D$

⇒ $CK = \{AB\}$

Prime attri $= \{A, B\}$ non prime $= \{C, D\}$

$AB \rightarrow C$ ✓ $C \rightarrow D$ ✗ it is transitive Dependency.

a) $R(ABCD)$ FD: $AB \to CD$, $D \to A$

$\Rightarrow CK = \{AB, BD\}$

$AB^+ = ABCD$ ✓

As A is dependent on D

$DB^+ = ABD = ABCD$ ✓

Prime attribute = {A, B, D}

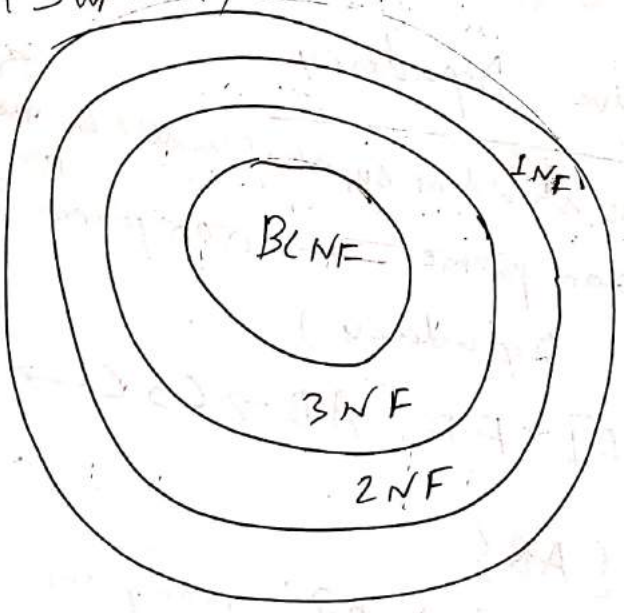
non prime attribute = {C}

$AB \to CD$ ✓ $\quad D \to A$ ✓

∴ This is in 3NF.

BCNF (Boyce Codd Normal Form)

- to be in 3NF
- Left Hand Side (LHS) should be always Candidate key or Super key



- Third Normal (3NF) always ensure Dependency preserving decomposition but not in BCNF

⇒ Dependency Preservation ensures none of the FD that hold on the original relation are lost
- the sub relation still hold or satisfy FD of the original relation.

- Both 1NF, 2NF, 3NF & BCNF ensures lossless decomposition.

Types of Decomposition ⇒

(i) Lossless Join Decomposition ⇒ • if a Relation R is decomposed into sub relation $R_1, R_2 \dots R_n$.
• This decomposition is called lossless join Decomposition when the join of the sub relation results the same relation R that was decomposed.

$$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n = R$$

$\bowtie$ - is a natural join Operator

Natural join ⊆ Cross product + Common attribute (Value same)

R

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

R_1

A	B
1	2
2	2
3	3

R_2

B	C
2	1
2	2
3	2

R_1 R_2

A	B	B	C	
1	2	2	1	✓
1	2	2	2	✓
1	2	3	2	×
2	2	2	1	✓
2	2	2	2	✓
2	2	3	2	×
3	3	2	1	×
3	3	2	2	×
3	3	3	2	✓

Natural join table

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

Lossy in terms of duplicacy / inconsistent

The above join is called lossy join as the common attribute was not primary key of main table.

Lossy Join 'Decomposition' :⇒ if Relation R is decomposed into sub relation $R_1, R_2 \dots R_n$

This decomposition is called lossy join decomposition. When the join of relation does not result the same relation R.

(*) if common attribute ~~is CK or SK of other~~ Should be candidate key or superkey of main table of R_1 or R_2 then there will be lossless join decomposition.

Ex:- $R_1(A\ B)$ $R_2(A\ C)$

(i) $R_1 \cup R_2 = R$

(ii) $R_1 \cap R_2 \neq \phi$

(iii) common should be R_1 CK or R_2 CK or both

1NF :=>
- no multivalued attribute
- only single attribute

2NF :=>
- In 1st NF must
+
- No partial dependency, only full dependency

if AB-CK & C D non prime attribute

A → C ✗ Partial dependency

AB → C ✓

3NF :=>
- In 2nd NF
+
- no transitive Dependency

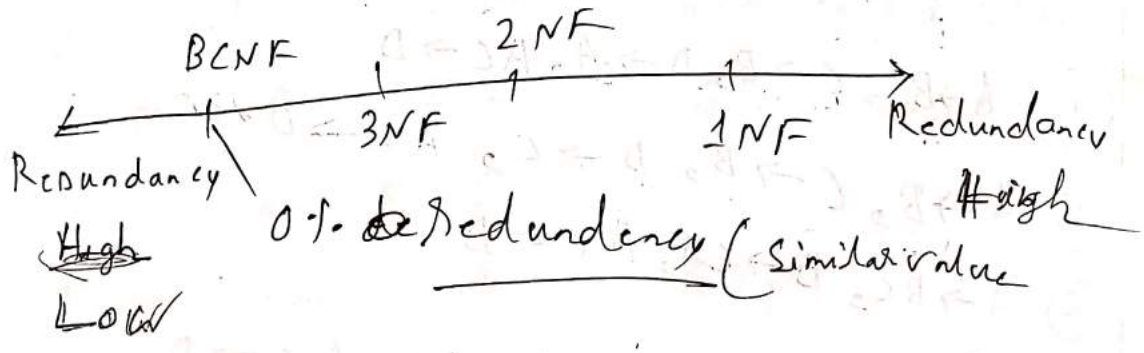
& Either LHS candidate key / Super key or RHS prime attribute

BCNF :=>
- In 3rd NF
+
- LHS must be CK or SK

4th NF :=>
- In BCNF
+
- No multivalued Dependency

(X →→ Y) Arijit → 3 Phone no. → 3 Email

5th NF :=> In 4th NF
+
- Lossless Decomposition



Minimal Cover :⇒ A minimal cover is a simplified and reduced version of the given set of functional dependency.

It is also called irreducible set.

It is also called as canonical cover.

Steps to find all FDs

(i) split right hand attribute of

$X \to YZ$

$\downarrow$

$X \to Y$, $X \to Z$

(*) $AB \to C$ then we can't write $A \to C$, $B \to C$ ✗

(ii) Remove all redundant (same FDs)

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

Here $A \to C$ can be removed as it can be achieved with $A \to B, B \to C$

(iii) Find the extraneous attribute in remove it (Left side)

$AB \to C$

if A^+ contain B then we can remove B

if B^+ contain A then we can remove A

Ex – Find minimal cover

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

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

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

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

(d)

⇒ Step 1 –

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

(D → B is crossed out)

Step 2 :⇒ We will see closure of every LHS omitting this FD

$A^+ = A\phi$ So $A \to B$ can't be removed

$C^+ \Rightarrow C$ accepted

$D^+ = DBC$ accepted

$D^+ = DABC$ So D is determining B so it will be removed

Now further we will not include this for further consideration

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

$D^+ = DAB$

$AC^+ = ACB$

Step 3 – $A^+ = AB$ So. we can't remove any of AC

$C^+ = CB$

Then the minimal cover,

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

⟶ (A) ✓

Q] $R(ABCDEF)$ check the highest Normal Form of FD: $\{AB \to C,\ C \to DE,\ E \to F,\ F \to A\}$

⇒ First we will have to find candidate Key

$AB^+ = ABCDEF$ $CB^+ = CEDFAB$

$CK = \{AB, FB, EB, CB\}$

Prime attribute $= \{A, C, E, F, B\}$

non prime $= \{D\}$

First We will check upper to lower

BCNF → AB → C ✓ AB is CK

C → DE ✗ C is not CK / SK

3NF → AB → C ✓ AB is SK/CK

C → DE ✗ C is not CK/SK

DE is not also prime

(E is prime but DE not prime)

2NF → AB → C ✓ AB is CK/SK full dependency

C → DE ✗ C is subset of CK & DE is non prime

By default every table is 1NF in this case as there is no table to check Wheather it contain multivalue or not.

Making a Relation into 2NF, 3NF, BCNF. ⇒

- R (ABCDEF)

FDs { AB → C, C → D, C → E, E → F, F → A }

⇒ Candidate Keys are = { AB, FB, EB, CB }

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

non prime attribute = { D }

The table itself is in 1NF (As their is no value)

Now we will check if it is in 2NF

AB → C ✓ C → D ✗ C → E ✓ E → F ✓ F → A ✓

Full Partial Full Full Full

So it is not in 2NF

To convert the table in 2NF we have to divide the table

And we have to make sure
- Lossless decomposition
- Dependency Preserving decomposition

We will divide the table (Full dependency together & partial dependency)

R (A B C D E F)

R1 (A B E F)

R2 (C D)

To make it lossless decomposition we have to keep some common attribute on R1 & R2. The common attribute should be CK / SK of R1 & or R2 & main in this case R2 has CD it is comparatively little So finding candidate key is quick easy.

R

R1 (A B C E F)

R2 (C D)

{AB → C, C → E, E → F, F → A}

All FD (Full dependent)

C → D

CK

FD

CK = { AB, FB, EB, CB }

CK = { C }

(2NF ✓)

Now we will check whether it is in 3NF or not. If it is in 3NF ok otherwise we will convert

AB → C ✓ C → E ✓ E → F ✓ F → A ✓

PA (prime attribute) PA PA PA

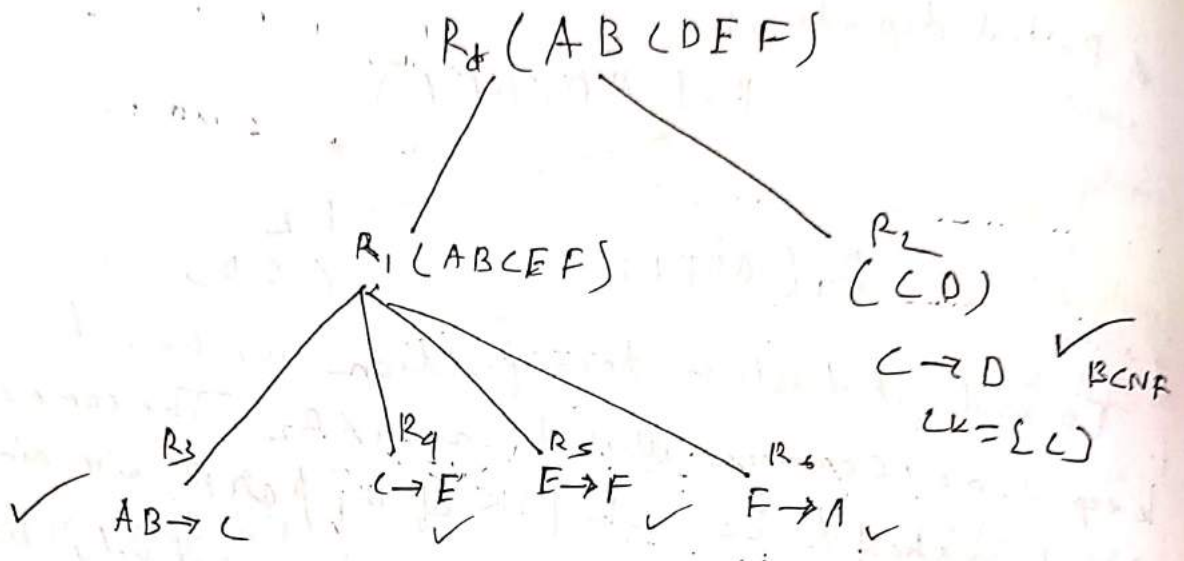
C → D ✓

The table is in 3NF also

Now We will check Whether it is in BCNF, If not then we will convert

AB → C ✓ (BCNF) C → E ✗ EF ✗ , F → A ✗

C → D ✓ (BCNF)



CK {A B} {C} {E} {F}

Pros :- Redundency 0 %.

Cons → High number of table
Problem in joining two table

• We can easily join R3 & R4 as it has a common attribute C & it is CK of R4.

Same for (R4 & R5) , (R5, R6)

But if we want to join R3 & R5 Then we can't do directly as their is no common attribute we have to take help of other attribute (R4 in this case)

Q.1 (1) Which of the following statement is true

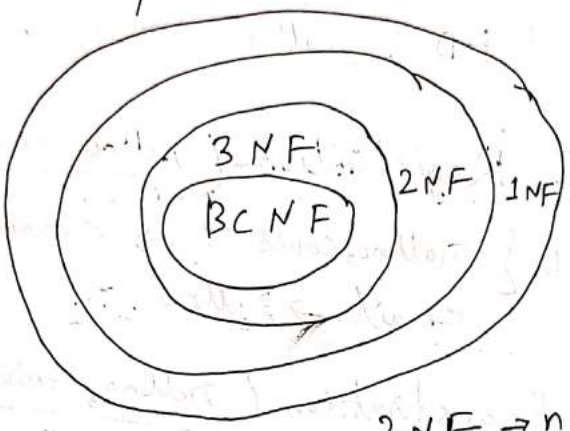
(a) A relation is in 3rd NF then it is always in BCNF

(b) A relation is in 2nd NF then it is not in 1st NF

(c) A relation is in BCNF then it is in 2NF

(d) A relation is in 2NF then it contains partial dependency.

⇒



2NF → not contain any Partial dependency

(c) ✓

Q. A relation R has eight attribute ABCDEFGH

$F = \{CH \to G, A \to BC, B \to CFH, E \to A, F \to EG\}$

How many candidate keys in R

(a) 3 (b) 4 (c) 5 (d) 6

⇒ A B C D E F G H

D is not in RHS in any FD

So D must be in Candidate Key $D^+ = D$ so it is a CK alone

$BD^+ = BDCFHGEA$ $CK = \{BD, AD, ED, FD\}$

Now we will check if B is available in RHS of any FD

Yes in $A \to BC$ it is available So we will check AD

$AD^+ = ADBCEFGH$

Now we will check Whether B is available in any other (in this case there is none). Next we will check if A is

in any RHS $E \rightarrow A$

So $ED^+ = AEDBCFGH$

Now we will check Wheather E is available or not.

$F \rightarrow EG$ in this E is in RHS

$FD^+ = EGDABCFH$

→ (b) 4 ✓

Q) Schema-1 : Registration (rollno, courses)

non trivial FD { roll no → course }

Schema-2 : Registration (rollno, courseid, email)

Non trivial FD { rollno, courseid → email
email → rollno }

Schema-3 : Registration (rollno, courseid, marks, grade)

non trivial FD { rollno, courseid → marks, grade
marks → grade

Schema-4 : Registration (rollno, courseid, credit)

non trivial FD { rollno, courseid → credit
courseid → credit }

Which in 3NF but not in BCNF ?

→ Schema-1 : BCNF ✓ means 3NF, 2NF, 1NF ✓

Schema-2 : BCNF ✗ as email is not CK

3NF ✓ rollno, courseid is CK
rollno is Prime attribute

So obvious it is in 2NF 1NF

Schema-3 : BCNF ✗
3NF ✗ "marks is not CK, grade is not Prime
2NF ✗ attribute

Schema 4 : BCNF ✗
3NF ✗
2NF ✗

→ (Schema 2) (Ans)

Equivalence of Functional Dependency :=>

$X = \{A \to B, B \to C\}$ | $Y = \{A \to B, B \to C, A \to C\}$

if X covers Y $X \supseteq Y$
Y covers X $X \subseteq Y$
then $X \equiv Y$

① X covers Y : we will take dependency from Y and check according to X

$A^+ = ABC$ — $A \xrightarrow{Y} B$ ✓ $A \to C$ ✓
$B^+ = BC$ — $B \to C$ ✓

② Y covers X : we will take LHS of X and take closure from Y.

$A^+ = ABC$ — $A \to B$ ✓
$B^+ = BC$ — $B \to C$ ✓

$\therefore X \equiv Y$

a) $X = \{AB \to CD,\ B \to C,\ C \to D\}$

$Y = \{AB \to C,\ AB \to D,\ C \to D\}$

=> X covers Y: $AB^+ = ABCD$ — $AB \to C$ ✓, $AB \to D$ ✓

$C^+ = CD$ — $C \to C$ ✓

Y covers X

$AB^+ = ABCD$ — $AB \to CD$

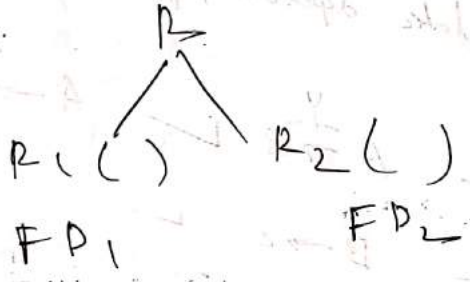
$B^+ = B$ — $B \to C$ ✗

So $U \not\equiv Y$

Dependency Preserving Decomposition =>

R(A B C D) FD → A → B, A → C

FD^+ — A → C



$FD_1 \cup FD_2 = FD^+/FD$

a) Let R(ABCD) with Functional dependency $\{A \to B, B \to C, C \to D, D \to B\}$.

R is decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(BD)$ will dependency will preserved

=>

First we will determine the possible dependencies (non trivials) and will check if it satisfy the Dependies of R

$R_1(AB)$	$R_2(BC)$	$R_3(BD)$
$A \to B$ ✓	$B \to C$ ✓	$B \to D$ ✓
$B \to A$ ✗	$C \to B$ ✓	$D \to B$ ✓
We have to check if B can determine A in main dependency R	$C^+ = CDB$ (in R)	@ $B^+ = BCD$ (in R)
in this case $B^+ = BCD$		

Now we will write the union of the possible dependencies -

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

Now we will check Whether the dependency in R can be possible while taking closer from the possible dependencies

$A \to B$ ✓ $B \to C$ ✓, $C \to D$ ✓, $D \to B$ ✓

∴ Dependency is preserved.

Q. Let $R(ABCD)$ with Functional Dependics $\{AB \to CD, D \to A\}$ Decompose $R_1(AD)$ & $R_2(BCD)$.

⇒

$R_1(AD)$	$R_2(BCD)$
$A \to D$ ✗	We have to write only non trivial dependencies
$D \to A$ ✓	$B \to CD$ ✗ | $BC \to D$ ✗
it is not in R or can be derived from R	$C \to BD$ ✗ | $CD \to B$ ✗
	$D \to BC$ ✗ | $BD \to C$ ✓

Now we will check LHS of Dependies R and take closes from possible combination

Possible combination are - $D \to A$, $BD \to C$

$AB^+ = AB$ So we cant achieve $AB \to CD$

$D \to A$ achieved

So Dependency is not preserved.

Join ⇒

- Cross join — only cross product
- Natural join
- Conditional join → Theta join, inner join (Default)
- Equal join
- self join
- outer join ← Left, Right, full

inner join is used to return rows from both table which satisfy the given condition.

E_no	E_name	Address
1	Arijit	Tamluk
2	Sayan	Ghatal
3	Anisha	Balurghat
4	Saranghae	Korea

Deptno	Name	Eno
D_1	CS	1
D_2	IT	2
D_3	Health	3
D_4	BTS	4

Suppose we just want to know the address of a Employee of named Arijit then we can get it only from Employee table

(Select Address from Employee Where Ename = "Arijit");

- But When We Want to display Employee name Who is Working in Health department.

=> We cant get answer with just only 1 table but we have to use two table.

Here we use join easily. We can also do with subquery but it is complex

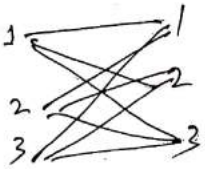
⊛ Atleast one common attribute needed for Join

- Join = cross product + condition.

Natural Join =>

⊛ Whenever there is any query firstly we have to find How many table we need to give the result

- if there is only one table we use simple command like select
- If there is two table needed we will use cross product
- crossproduct is done using table1, table2



- cross product

(X) A natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same name and data type.

- There should be one common attribute.
- There is no need to specify the column names to join
- The resultant table always contains unique columns
- It is possible to perform a natural join on more than two table.
- Do not use on clause

Syntax: Actual

```
Select columnnames/* from
table1 Natural Join table2;
```

Explanation :=> Actually what happen.

> Select E_name from employee, Department Where Employee.E_no = Department_E.no

cross product

⇓

it will first cross multiply 2 table there will be 16 rows but will select only where E_no are same.

There is only 4 column where E_no are same.

We only print E_name as select command only need to print E_name.

Self Join :=>

Used to join a table to itself as if the table were two tables temporally. Renaming at least one table in the Sql statement

Syntax: Select a.column_name, b.column_name ....
From table1 a, table2 b
Where a.common_field = b.common_field;

Referencing a Student table's primary key

fk

fk

referencing a course table's primary key

S_id	C_id	Since
S1	C1	2016
S2	C2	2017
S1	C2	2017

Study

— it is an intermediate table

We have to find student id who is enrolled in atleast two courses → 2, 3, 4 ... n

We will use Self Join to solve this

```
Select s_id from study as T1, study as T2
Where T1.sid = T2.sid and T1.cid <> T2.cid
```

<> — not equal to

Equal Join :⇒ Combine table based on (Equi) matching values in specified columns.

- The column names do not need to be same.
- The resultant table contains repeated column
- can perform on more than 2 table.

Emp

E_no	E_name	Addrs
1	Ram	Delhi
2	Axi	WB
3	Ani	Karca
4	Priya	Japan

Dept

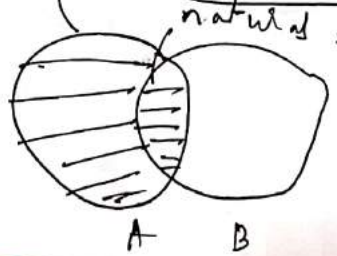
Dept_no	Locats.	Eno
D_1	Delhi	1
D_2	Karca	2
D_3	Bangalore	4

Suppose we want to find emp_name who worked in a department having location same as their address

⇒ Select Ename from Emp, Dept where
Emp.Eno = Dept.Eno and Empaddrs
= Dept.location

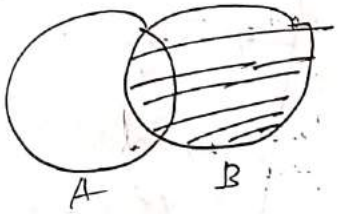
Left outer Join :⇒ it gives the matching rows and the rows which are in left table but not in right table

Extra Any outer Join = Natural join + Something



Syntax — Select column_names from table1 (Left table)
left outer join table2 On (table1.common attri = table2.common attri)
Condition.

Right outer Join :- ⇒ it gives the matching rows and the rows which are in right table but not in left table.



Right

Select columns from table1 Right outer join table2
on (table1.common = table2.common)

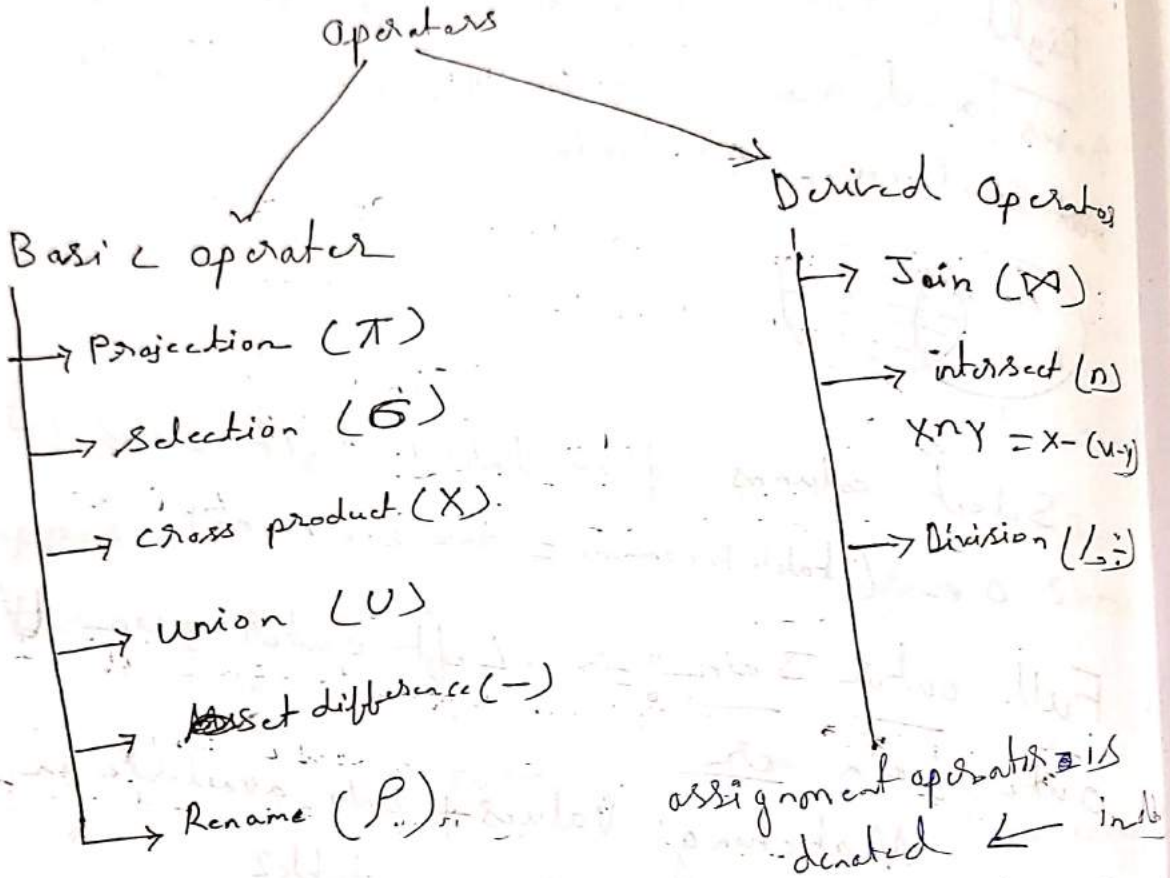
Full outer Join :- ⇒ Left outer join Right outer join or
Matching values + only available on table1
+ only available on table2.

Relational Algebra :=> E.F Codd 1970

It is also called Procedural query Language.
(What to do / How to do it)

Formal query Language

Which takes relation as input & give relation as output



Projection :=> Project some columns.

Student

Rollno	Name	Age
1	Arijit	18
2	Anisha	19
3	Sayan	22

if we want to fetch rollno then we will write Π_{rollno} (Student) → Show only rollno column
(table name)

$\Pi_{rollno,name}$ (Student) → Show only rollno, name.

- Projection always give distinct (unique results)

Name
A
B
A

Π_{Name} (student)

it will only give — A distinct
B elements

Selection (σ) :=> Select rows based on given condition

If we want to retrieve the name of student whose roll no is 2

=> Retrieve the name means Projection Π
Whose roll no is 2 — selection σ

Π_{name} ($\sigma_{rollno=2}$ (Student))

2	Anisha	

Anisha

We always use Π as last because Π always show distinct values so we can loose any value. if name is same then we may loose desired data,

Projection = column

Selection = rows / tuples

Cross Product (X) :=> cross product of relations returns $m * n$ rows where m and n are numbers of rows in R_1 & R_2

$m+n$ - number of columns

A	B	C
1	2	3
2	1	4

R_1

C	D	E
3	4	5
2	1	2

R_2

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

Union (U) ⇒ tuples which are in R_1 or in R_2.

Max no of rows returned = m+n

min no of rows returned = max(m,n)

- No of columns should be same on both table in order to do union
- Domain of every column should be same.

Roll	Name
1	A
2	B
3	C

(Student)

Emp no	Name
7	E
1	A

(Employee)

Roll	Name
1	A
2	B
3	C
7	E

(Student) U (Employee)

By default the attribute name of union will be first table attribute name

- There will no duplicate value.

$$(\Pi_{name}(Student) \cup \Pi_{name}(Employee))$$

$$\Downarrow$$

Name attribute is

Name
A
B
C
E

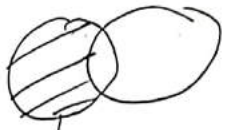
⊛ (Student) U (Employee) give the result age order of data is matched

numerical, char ∪ numerical, char

else it will give error

Set Difference :=>

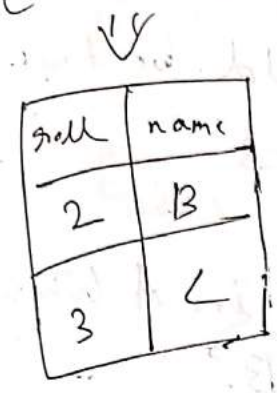
A - B = available in A but not in B



A - B

- no of columns must be same
- Domain of every column must be same

(Student) — (Employee)



roll	name
2	B
3	C

Suppose we want to find student names who are not employee

$$(\Pi_{name}(Student) - \Pi_{name}(Employee))$$

Rename (ρ) :=> Renaming a relation to another relation.

intersection ($\cap$) :=> Returns those tuple which are in both R_1 & R_2

Max rows = min (m, n)

Min rows = 0

Conditional Join ($\bowtie_c$) : Selection from two or more tables based on some condition

Cross product + condition.

Equi Join ($\bowtie$) :⇒ Special case of condition join when only equality condition are applied.

Left outer join → ⟕

Right outer Join - ⟖

Full outer join - ⟗

Division Operator :⇒ A (x,y) / B (y) returns x values for that there should be tuple (x,y) for every y value of relation B

or

A/B returns those tuple in A which are associated with every tuple of B.

note → Attribute of B should be a proper subset of attributes of A.

Sid	cid
S_1	C_1
S_2	C_1
S_1	C_2
S_3	C_2

Enrolled (E)

cid
C_1
C_2

course (C)

Query - Retrieve Sid of students who enrolled in every / all course

⇒

$E(sid, cid) / C(cid) = S_1$ (as S_1 is enrolled in every course of cid)

- It is an derived operator

$$\Pi_{sid}(E) \times \Pi_{cid}(C) \Rightarrow \begin{matrix} S_1 & C_1 \\ S_2 & \times \ C_2 \\ S_3 \end{matrix}$$

it means every student is enrolled in every course

now we will use set difference (−) to know the students who are not enrolled on all courses.

$$\Pi_{sid}\left(\Pi_{sid}(E) \times \Pi_{cid}(C)\right) - (E)$$

$S_2 \quad C_2$

$S_3 \quad C_1$

now we get the students who are not enrolled in every course now we will just minus from students of Enrolled

$$\Pi_{sid}(E) - \Pi_{sid}\left(\left(\Pi_{sid}(E) \times \Pi_{cid}(C)\right) - E\right)$$

$\Downarrow$

S_1

Relational Calculas :=>

E.F Codd 1972

Relational calculas — Tuple Relational calculus
Domain Relational calculus
Lacroix 1977

Tuple Relation calculas is a non Procedural language. (We only say what to do)

Procedural language < What to do / How to do (< Language)

- A query is expressed as

$\{t \mid P(t)\}$ Where t = resulting tuples & $P(t)$ is known as predicate and these are condition that are used to fetch it. it gives all tuples if $P(t)$ is true

Operations: AND — $\wedge$
OR — $\vee$
NOT — $(\neg)$

It also use quantifiers

- $\exists t \in r(Q(t))$ => there exist a tuple in t in relation r such that Predicate $Q(t)$ is true.

ex — $\exists t\ (age > 18)$

if there is any tuple where age > 18 then it will give true

• $\forall t \in r(Q(t))$ - gives true if all tuple follows $a(t)$

ex- $\forall t (age > 20)$

if all ages > 20 then it will give true.

Unsafe expression ⇒

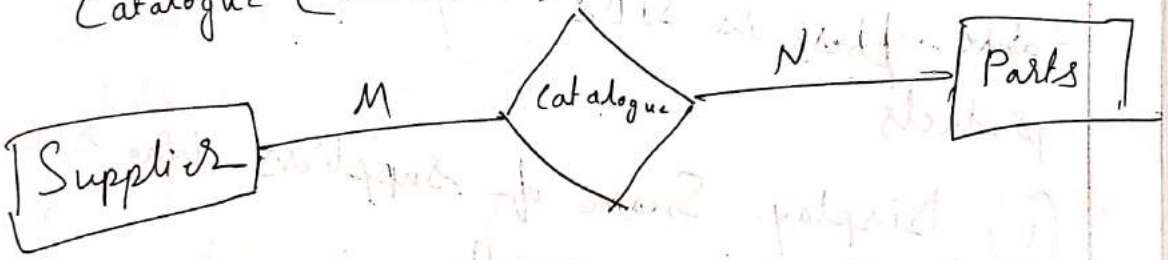
$\{ S.name \mid \neg Supplier(S) \}$

it is unsafe as it is telling to print names which are not in supplier table. There is infinite name so it is unsafe.

(*) Supplier (SID, Sname, address)

Parts (PID, Pname, color)

Catalogue (SID, PID)



(1) Write a query in SQL to display Sname of suppliers?

⇒ $\{ s.Sname \mid Supplier(s) \}$ ← variable

We make variable s of supplier table now we will check all the tuples and print Sname.

(ii) Display Pname of parts whose color is Red

$\Rightarrow \{ \$p.pname \mid Parts(p) \wedge p.color = \text{"Red"} \}$

(iii) to display SID of suppliers whose name is "Arijit" and address is "tamluk"

$\Rightarrow \{ S.SID \mid Supplier(S) \wedge S.Sname = \text{"Arijit"} \wedge S.address = \text{"tamluk"} \}$

(iv) Display SID of Suppliers who supplied Some parts

$\Rightarrow \{ c.SID \mid Catalogue(C) \}$

We can get this with the help of catalogue table. there is SIDs only which supply some products

(v) Display Sname of Suppliers who Supplied Some parts

free variable — Bounded variable

$\Rightarrow \{ S.Sname \mid Supplier(S) \wedge (\exists C)(Catalogue(C) \wedge S.SID = C.SID) \}$

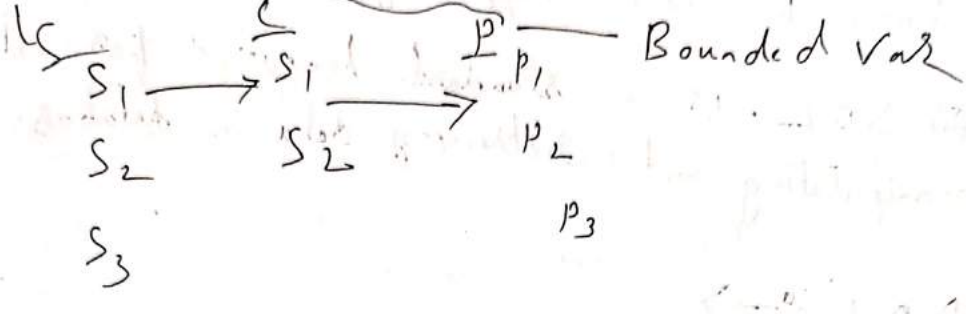
Here we need two table (Suppliers, catalogue)

We use there exist which check if Supplier exist in catalogue where S.SID = C.SID

(vi) Display Sname of Suppliers who supplied some Red color parts

$\Rightarrow \{ S.Sname \mid Supplier(S) \wedge (\exists C)(Catalogue(C) \wedge (\exists P)(parts(P)) \wedge S.sid = C.sid \wedge P.Pid = C.Pid \wedge P.color = 'Red' \}$

free variable



SQL (Structure Query Language) $\Rightarrow$

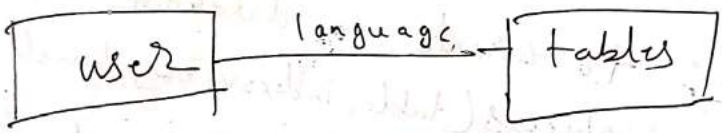


table is stored in database in a structured way thats why we called it structure query Language.

EF codd publish Relational model in 1970 with Relational Algebra and Tuple relational Calculas (TRC)

Previous it is called SEQUEL (Simple English Query Language)

Later it is named SQL

- SQL is a domain specific language as it is only used in dbms / C is used in various place.
- SQL is declarative language or non procedural language
  We just say what to do, we dont know how it is happening inside.

⊛ SQL is a standard language for storing, manipulating and retrieving data in database.

DDL :=>

Data Defination language which deals with database Schemas (tables)

- Create: to create a database and its objects (table, indexs views, function..)
- Alter: Alters the structure of the existing database
- DROP: Delete the whole structure
- Truncate: Remove all records from a table including all the spaces.
- Rename: Rename the table / objects

DML :=> Data Manipulation language used to store, modify, retrieve, delete and update data

- Select: retrieve data from a database
- Insert: insert data into table
- update: update existing data within the table.
- Delete: Delete record
- Merge: combining all three main statement (insert, update, delete)

DCL :=>
Data control Language, mostly concerned with rights, permission

- Grant — allow users access privillages
- Revoke — withdraw users access given by using grant

TCL :=>
Transaction Control language deals with a transactions

- Commit: commit a transaction
- Rollback: Rollback a transaction in case of error
- Savepoint: Save the points in between transaction

Constraints :- ⇒ Condition in attribute value

- Unique : All the values in a attribute is unique. No two data can't match
- Not NULL : Mandatory field. It can't be empty *
- Primary key : unique + not null
- check : checking values in time of entering

check (age > 18). if user enters -10 or 14 it will show a error message instantly and will say to enter the correct age

- Foreign key : key which is taking reference from primary key
- Default : if user does not enter any value then by default it will take a value.

Suppose i does not enter dob it will take defaultly 01.01.1990.

(*) We can use multiple combination of constraints.

Create Database :=>

used to create new SQL database

Syntax

```
Create database database_name;
```

DROP Database :=>

used to drop an existing SQL database

```
Drop database database_name;
```

CReate Table :=>

used to create a new table in a database

Syntax:

```
Create table table_name
(
    column_1_name   datatype,
    column2name     datatype,
    column3name     datatype
    - - - -
);
```

like -

```
Create table emp
(
    id int,
    name varchar2(20),
    Salary number(10)
);
```

To know the details of the column we use.

description -

```
desc tablename;
```

desc emp;

Drop table :=> used to drop (delete) an existing table in a database.

```
Drop table table_name;
```

Truncate table :=> used to delete the data inside a table but not the table itself.

```
Truncate table table_name;
```

Alter table :=> used to adds, delete or modify column_names in an existing table
- used to add & drop various constraints on an existing table
  - Add column
  - Modify datatype
  - Add constraints
  - Rename column/table.
  - Remove Column
  - Modify datatype length
  - Remove constraint

Ex-

- Alter table tablename
  Add columnname datatype;
- Alter table tablename
  Drop column column column_name;
- Alter table table_name
  modify Column_name datatype;

- Alter table tablename
  rename column oldname to newname;

Alter table tablename rename to newname;

Alter table tablename
  add primary key (column_name);

Alter VS Update ;=>

Alter is Data Definition command
update is Data Manipulation command

Alter can change, add, remove the table structure (column's name, constraints, data type)

update can change only the data inside columns.

like update emp
  set salary = salary * 2;

Delete vs Drop vs Truncate ;=>

Delete is DML command but Drop, Truncate is DDL command
Delete is used to delete rows.
Drop is used to delete the entire table including schemas.
Truncate just clear the record / data in the schema but the structure remains unchanged.

Delete can delete specific rows according to condition but truncate delete rows entire table locking the data.

There is option of rollback in delete before committing as it keeps a log file. Slower

There is no option in rollback in truncate.

It is faster.

Select :=> Used to select data (columns) from database.

Syntax: Select column1, column2, ...
from tablename;

if we want to select all fields then we use asterisk (*)

Select * from emp;

Select distinct :=> used to return only distinct (different) values

Syntax: select distinct column1, column2, ...
from table_name;

Where :=> used to filter records in terms of condition

Syntax: Select column1, column2, ...
from tablename
Where condition;

operators:

= : Equal < : Less than

> : greater than >= : greater than or equal

<= : Less than or equal <> : Not equal

!= : Not equal.

AND → display record if all conditions are true

OR → if any of the conditions is true

Not → if the condition is not true.

Roll	Name	Section
1	Arijit	A
2	Sayan	B
3	Anisha	C
4	Priyanka	A
5	Soumen	B

– student

- Select Section from student;

O/P – Section
A
B
C
A
B

O/P – Section
A
B
C

- Select distinct Section from student;

- Select Name from student
  Where Roll = 3;

O/P – Name
Anisha

Order by :=> used to sort the result set in Ascending or Descending order. By default it sort Ascending order.

Syntax - Select column1, column2 ...
from table_name
Order by column1, column2, .. Asc/Desc

Ex- Select * from student
order by name ; // Asc by default
Select * from student
order by name Desc ;

Insert into :=> used to insert new records in a table

Syntax : • Insert into table_name
Values (value1, value2, ...) ;
// We have to maintain the order of column and its value)

• Insert into table_name (column1, column2 ...)
Values (value1, value2, ...) ;

Ex: Insert into student
Values (6, 'Sabir', 'A') ;

Null Value :=> It is not possible to test for null values with comparison operators such as =, ≠, <>

We will use IS NULL or IS NOT NULL

Syntax - Select column_names
from tablename
Where column_name IS NULL;

Ex - Select * from student
Where section IS NOT NULL;

Update :=> used to modify existing table's record

Syntax: update tablename
Set column1 = value1, column2 = value2 ....
Where condition;

ex - update student
set name = 'Suranghae'
Where roll = 3;

Delete :=> used to delete existing record in a table

Syntax: Delete from tablename; // delete all rows.

Delete from tablename
Where condition; // delete rows that matched the condition.

Select top :=> used to specify the numbers of record to return.

Syntax -- • Select top number column-na
from table_name
Where condition;

• Select column_names
from table_name
Where condition
limit number;

• Select column_names
from table_name
Fetch First number rows only;

• Select column_names
from table_name
Where rownum <= number;

ex • Select top 3 * from student;

• Select * from student
limit 3;

• Select * from student
Fetch first 3 rows only;

Aggregate Function :=> NULL is not counted in aggregate function

(i) Min() :=> Return the smallest value of selected column (numeric)

Syntax: Select min(column_name)
from table_name;

ex- Select min(Salary)
from emp;

(ii) Max() :=> Return the largest value of the selected column (numeric)

Syntax: Select max(column_name)
from table_name;

ex- Select max(Salary)
from emp;

(iii) Avg() :=> Return the average value of the selected column (numeric)

Syntax: Select Avg(column_name)
from table_name;

ex- Select Avg(Salary)
from emp;

(iv) Sum() :=> return total sum of a numeric column

Syntax Select sum(column_name) from table_name;

⑤ Count() :=> returns the number of rows that matched the specified criteria.

Syntax: Select count(column_name) from table_name;

ex- Select count(name) from student;

IN :=> allows you to specify multiple values in a where clause.
. shorthand for multiple OR.

Syntax: • Select column_names from tablename where columns_name IN (Value1, Value2, ...);

• Select column_names from table_name where column_names IN (Select statement);

ex- Select * from students where section IN ('A', 'B');

Between :=> Selects values within given range
Value can be numbers, text / dates
• it is inclusive: both start and end value is included

Syntax - Select column_names from table_name where column_name between value1 and

Like :⇒ Used in a where clause to search for specified pattern in a column.

- The percentage sign (%) represent zero, one or multiple characters
- The underscore (_) represent one or ~~multiple~~ single character

Syntax:
```
Select column_names
from tablename
Where columnname like pattern;
```

Patterns —

'a%' → Values start with a
'%a' → Values end with a
'%ac%' → values that have ac in any position
'_a%' → Values that have a in 2nd position
'a_%' → values start with a & at least 2 characters in length
'a__%' → values start with a & at least 3 character in length
'a%o' — starts with a and ends with o.

Join :⇒ Used to combine rows from two or more table based on related column between them.

Types :- Inner Join :⇒ (by default) Select records that have matching values on both table

Syntax - Select column_names
from table1
Inner join table2
On table1.column_name = table2.column_name;

Don't preserve non-matched tuples

Ex- Select orders.orderID, Customers.customername
from orders
inner join customers
on orders.customerid = customers.customerid;

Left / Right / Full outer Join :⇒

Left = common rows + left in left table rows (table1)

Right = Common rows + left in right table rows (table2)

full = Common rows + left in both table

Select column.names
from table1
Left / Right / full join table2
on table1.columnname = table2.column.name

Union :=> used to combine the result set of select command

- there should be same number of columns
- Columns must have similar data type.
- must in same order

Syntax- Select column_names
from table1
Union
Select column_names from table2;

(*) Union select only distinct value
- Union all select duplicate value.

Group By :=> Group rows that have same value (identical data)

Syntax - Select column_name (same)
from table_name
group by column_name;

(*) We can only use the same column name in select. but can use aggregate function of other column.

(*) Where is used in every row but in group by we use having

Having :⇒ Where clause place condition on select statement but having clause places on groups created by group by clause

Syntax - Select column_names
from table
where condition
group by column_names
Having condition; → aggregate function

Ex- Select count(customerid), country
from customers
group by country
having count(customer_id)>5;

Aliases :⇒ used to give a table temporary name

AS - keyword is used.

Syntax - • Select column_name as alias_name
from tablename;

• Select column_names
from tablename as alias_name;

Subqueries

⇒ Emp

Eid	Ename	Dept	Salary
1	Ram	HR	10000
2	Amrit	MRKT	20000
3	Ravi	HR	30000
4	Nitin	MRKT	40000
5	Varun	IT	50000

(1) Display Max salary from emp table

⇒ first we have to know for find the result which table is needed

We will use aggregate function

```
Select max(salary) from emp;
```

⇓

50000 (O/P)

(2) Display employee name who is taking maximum salary.

⇒ We will used nested query

outer query

```
Select Ename from emp where salary
= (select max(Salary) from emp)
```

inner query

In this first inner query is executed then the outer query will execute.

Where salary = 50000 (inner query result)

(3) Display 2nd highest salary from emp table

=> Suppose first we get the highest Salary, then we will exclude it then again will find the highest Salary

(*) Always use dummy table

Select max(salary) from emp where salary <> (select max(salary) from emp);

First inner query will execute then the outer

(4) Display name who is taking 2nd highest salary

=> We will just have to write a outer query for name

Select Ename from emp where Salary = (select max(salary) from emp where Salary <> (select max(salary) from emp));

(5) Display all the departments name along with number of employees working with them?

=> first we will use group by for togethering identical data

then use count() aggregate function

```
select dept, count(dept)
  from emp
  group by dept;
```

HR	2
MRKT	2
IT	1

(6) Display all the dept names where no of employees less than 2

=>
```
select dept
  from emp
  group by dept
  having count(*) < 2;
```

(7) Display employee name where no of employees less than 2

=>
```
select Ename from emp where
dept in ( select dept from emp
group by dept having count(*) < 2);
```

(8) Display highest salary department wise and name of emp who is taking that salary

=>
```
select Ename from emp where
salary in (select max(salary) from emp
group by dept);
```

Eid	Ename	Address
1	Ravi	Chd
2	Varun	Delhi
3	Nitin	Pune
4	Robin	Bangalore
5	Ammy	chd

Emp

Eid (fk)	Pid	Pname	Location
1	P_1	IoT	Bangalore
5	P_2	Big data	Delhi
3	P_3	Retail	Mumbai
4	P_4	Android	Hyd

Project

(9) Details of employee whose address is either Delhi or chd or Pune

⇒ Select * from emp where Address in ('Delhi', 'chd', 'Pune');

(10) find the name of employee who are working on a project

⇒ Select ename from emp where eid in (select distinct eid from project);

(11) Find the details of emp who is working on at least one project?

⇒ Select * from emp where exists (select eid from project where emp id = project eid);

(*) In nested query we do first inner query then compare with outer query but in corelated nested query first we execute outer query then search inner query. it is top to down approach.

In nested query inner query just executed one time but in corelated nested query inner query checked several times.

- Exist / not exist always return true or false checking condition.

Corrdated Subquery :=>

- It is a subquery that uses values from outer query
- Top to down approach

In (11) we use corrdated subquery.
Here we can use exist, not exist, Any, all operator
for every outer row the inner query will execute every time.

1 Ravi chd - it will selected by outer query then it will be checked with inner query then 2nd row will be selected and so on.

(*) corrdated Delete update also happen.

Nested Subquery Vs Correlated Subquery Vs Joins

- **Nested subquery**: Bottom up approach

  ex- Select * from emp where cid in
  (select cid from dept)

  inner query execute first only one time.

- **Correlated Subquery**: Top down approach
  - comparison number high

  Select * from emp where exists
  (Select eid from dept where emp.cid=dept.cid);

- **Joins**: cross product + condition

  Select * from emp, dept where
  emp.cid = dept.cid;

cid	name
1	Arijit
2	Sayan
3	Anisha
4	Priyanka
5	Suvamay

emp

dno	dname	cid
D1	HR	1
D2	IT	2
D3	MRKT	3
D4	IT	5

dept

Find the detail of all employee who works in any department

Find Nth highest Salary :=>

We can solve it with the help of correlated query

```
Select id, salary from emp e1
Where N-1 = ( select count (distinct salary)
              from emp e2 Where e2.salary
              > e1.salary );
```

id	Salary
1	10000
2	20000
3	20000
4	30000
5	40000
6	50000

emp

id	Salary	
1	10,000	> 30,000 ✗
2	20,000	> 30000 ✗
3	20000	
4	30000	> 30000 ✗
5	40000	> 30000 ✓
6	50,000	> 30000 ✓

e2 count = 2

Suppose we want 3rd highest → 30000 ✓

N-1 = 3-1 = 2

(*) We recognize it to be correlated query by we use outer query value (e1) in inner query.

Q | You need to display last name of employees who have 'A' as second character in their name.

(a) Select last_name from emp where last_name like '_A%';

(b) Select last_name from emp where last_name = '*A%';

(c) Select last_name from emp where last_name like '%A%';

⇒ (a) ✓

Q | A command to remove relation from SQL Database

(a) Delete table tablename

(b) Drop table tablename

(c) Erase table tablename

(d) Alter table tablename

⇒ Remove relation means remove Sql database table

We use Drop command (DDL)

4) In the following schema R(a, b)

q_1: select * from R

q_2: (select * from R) intersect (select * from R)

q_3: select distinct * from R

(a) q_1 q_2 q_3 produce same result

(b) only q_1 q_2 produce same result

(c) only q_2 q_3 produce same result

(d) q_1 q_2 q_3 produce different result

$\Rightarrow q_1$ select * from R $\Rightarrow$

a	b
1	1
2	2
3	3
2	2

q_2 —

a	b
1	1
2	2
3	3
2	2

$\cap$

a	b
1	1
2	3
3	3

$\Rightarrow$

a	b
1	1
2	2
3	3

$=$

q_3 —

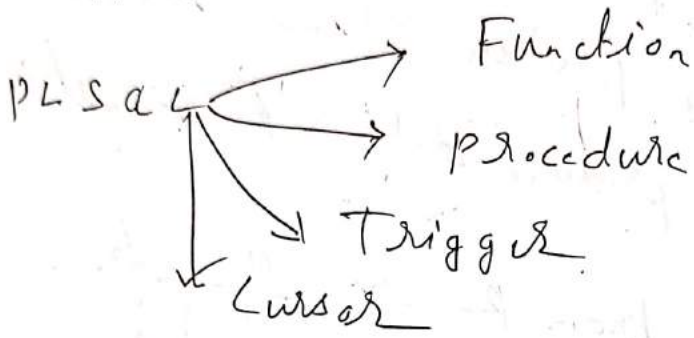
a	b
1	1
2	2
3	3

(c) ✓

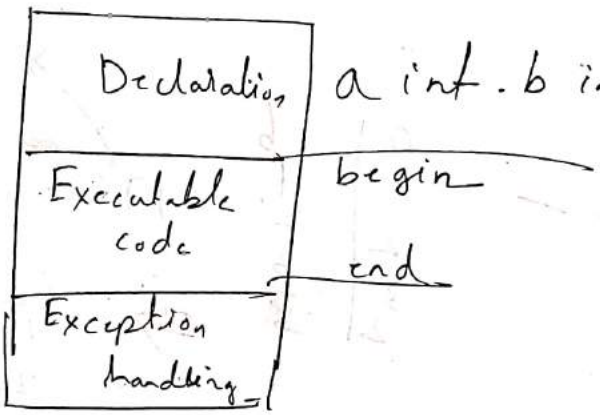
PL SQL (Procedural SQL) ⇒

SQL is declarative / non procedural — only What to do is said.

but PL SQL is Procedural Sql (What to do, How to do) . We use function here



PL SQL Block code



adding number

```
declare
a int
b int
c int
begin
a:=10
b:=20
c:=a+b
end
```

Transaction :=>

Transaction is a set of operation used to perform a logical unit of work.

. It generally represent change in database.

Operations :=>

. Read operation: Read(A) instruction will read value of 'A' from the database and will store it in the buffer in main memory.

. Write operation: Write(A) will write the updated value 'A' from the buffer to the database

Ex- Suppose we want to withdraw money then we use ATM transaction

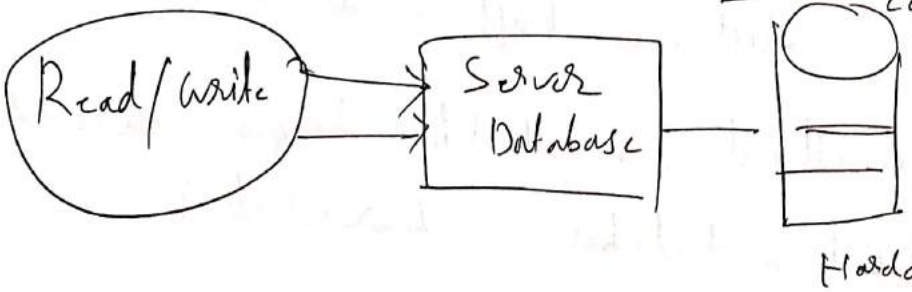
Work → Withdraw money

Set of operation - Card inserted
Reading information of card
language select
Account type
money
Pin
money
completed

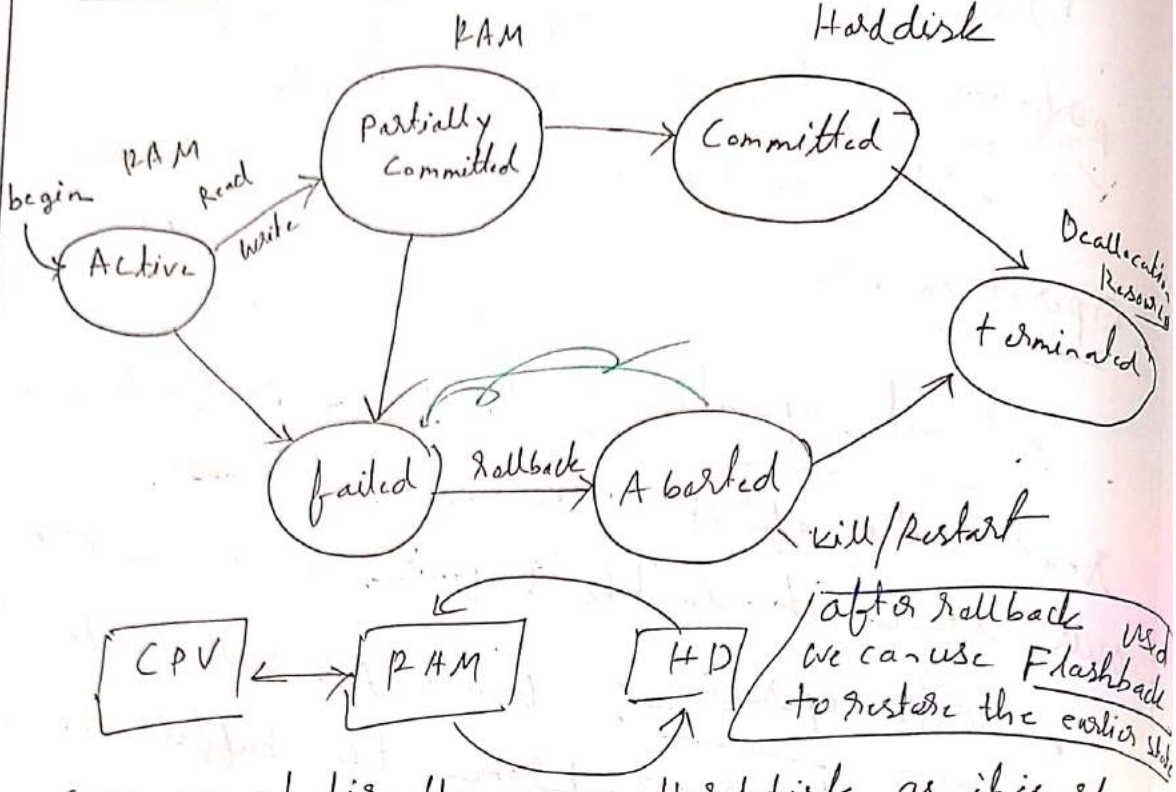
transaction

R(A) → 1000 | A = 1000, B = 2000
A = A - 500 // buffer
W(A) → 500 // DB
R(B) → 2000
B = B + 500 // buffer
W(B) → 2500 // DB

Commit



Transaction States :=>



CPU can't directly access Harddisk as it is slow Thats why we use buffer in main memory

- **Active state :**
  - Ts in active state as long as instructions are executed.
  - All the changes are in buffer of main memory
- **Partially committed :**
  - A transaction execute its last instruction it is in Partially committed
  - Still in buffer of main memory
- **Committed :** . All the changes successfully Stored in database (harddisk)

• Failed : • When transaction is in active or partially committed state and some failure occurs (power off, back button etc) then it is in failed state.

• Aborted : • if transaction failed then Recovery manager roll back all of its write operations to the database and bring the database to original state. Recovery module select one of two operation
(i) Re start the transaction
(ii) kill the transaction

ACID Properties of transaction ⇒

To maintain Concistency of database certain properties are followed during transaction

ACID — Isolation — Durability — Consistency — Atomicity

Atomicity : • Either the transaction occurs completely or it does not occur at all.
• No transaction occurs partially

Consistency : • The database must be consistent before and after the transaction.
• Before and after the transaction the sum of money should be same.

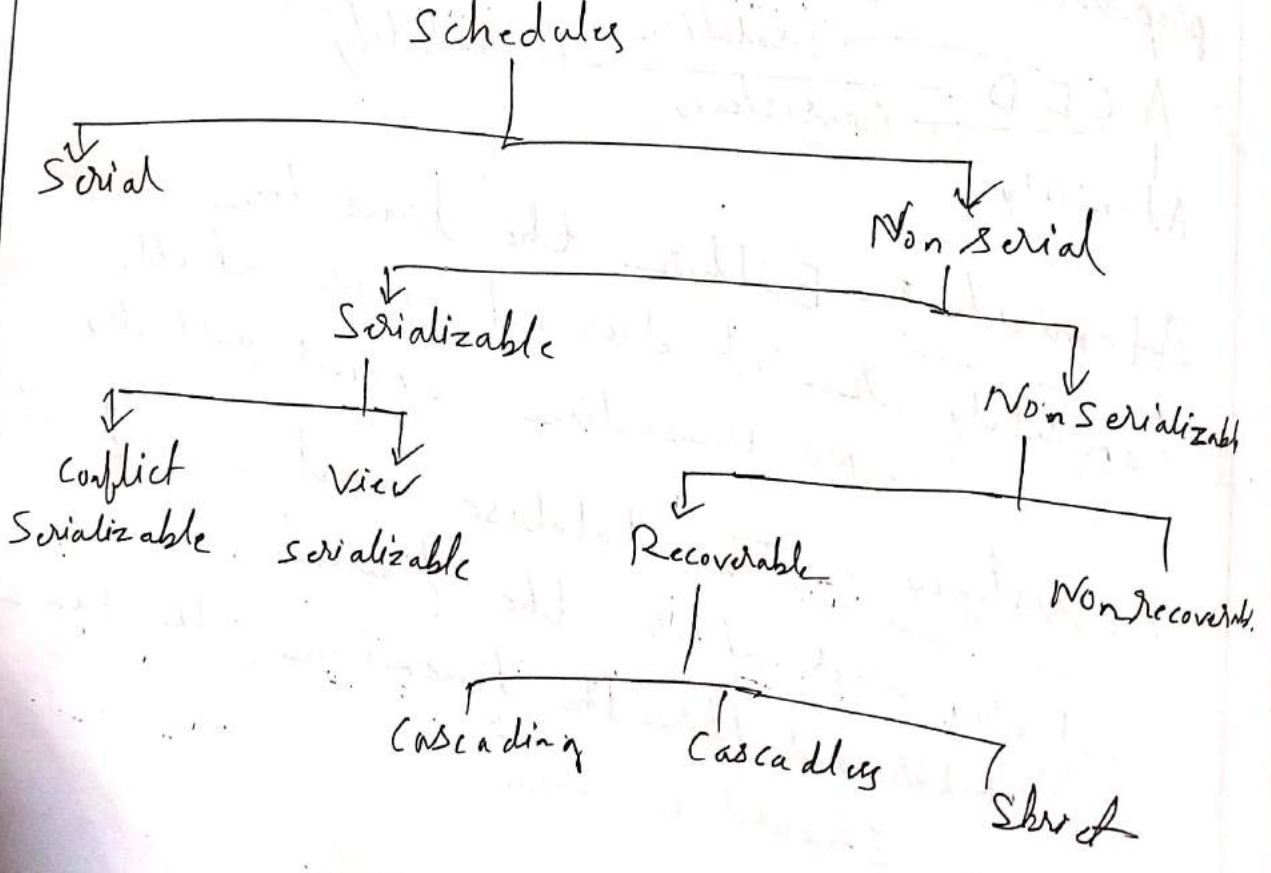
* Isolation :=>

- Multiple transaction occurs independently without interference
- changes occuring in a particular transaction will not be visible to other until it committed/written to memory.

. Durability :=>

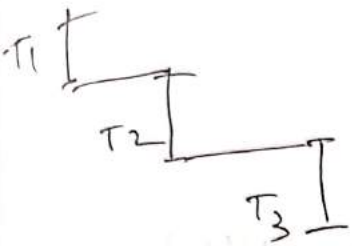
- if a transaction committed then it can not be lost no matter what.
- if there is failure but changes will not be lost if committed.

Schedules :=> The order in which the operation of multiple transaction appear for execution



Serial Schedule:

- All the transaction execute serially one after the other
- When one transaction executes, no other transaction is allowed to execute.



T1 T2 T3

Adv :- Consistent result

Disadv :- High Waiting time. Throughput decrease

Ex- In atm machine at a time only one card can inserted

Non Serial / parallel :-

- Multiple transaction execute concurrently.

Adv → High throughput & performance

Disadv — not consistent

ex— Online banking . IRCTC ticket booking

Types of problem in Parallel

(1) Dirty read :

T_1	T_2
R(A) 60	
W(A) 50-10	
Rollback ↑ failure	R(A) 50 — Dirty read
	W(A) 50-10 — it reads
	Commit wrong value

as transaction is roll backed

(ii) incorrect Summary :- in between a transaction a other transaction come, read, commit an aggragate function

(iii) Lost update : older changes are overwritten by newer changes by transaction.

(iv) Unrepeatable Read :

T1	T2
R(x) 100	
	R(x) 100
W(x) 50	
	R(x) 50

different read

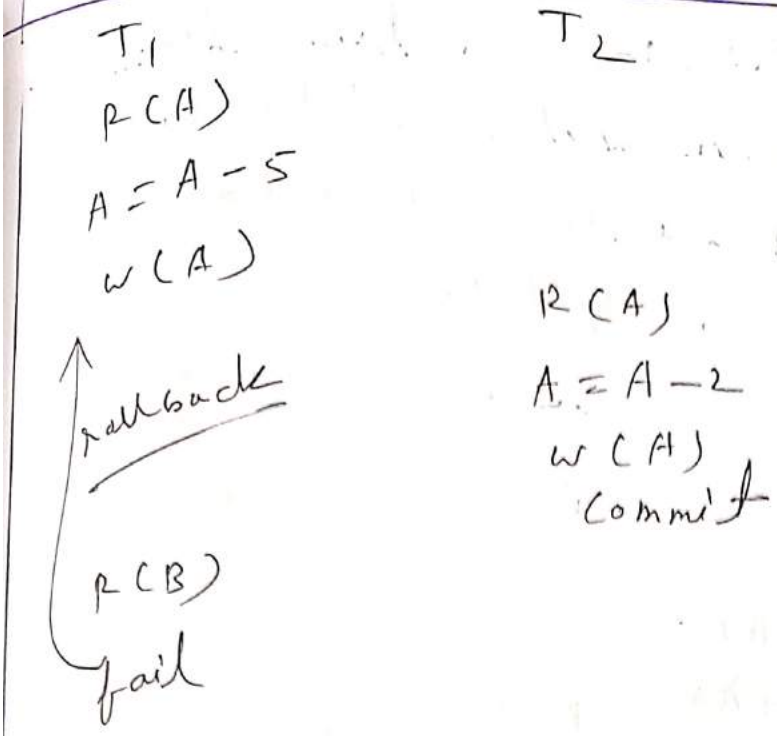
(v) Phantom Read

T1	T2
R(x)	
	R(x)
Delete(x)	
	R(x)

Non Recoverable Schedule :

If in a schedule

A transaction perform dirty read operation from an uncommitted transaction & commit before the transaction from which it read dirty read.



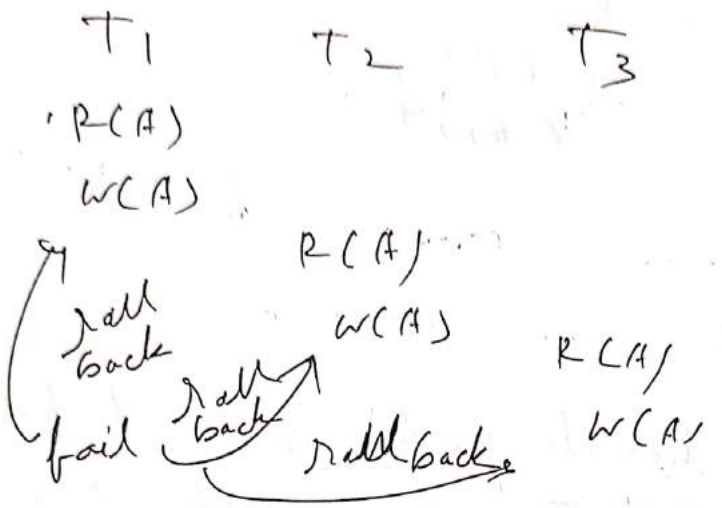
Recoverable Schedule :⇒ If in a Schedule A transaction perform a dirty read operation from an uncommitted transaction. Its commit operation is delayed till the uncommitted transaction either commit or roll back. It is called Recoverable Schedule

. Cascading Schedule: failure of one transaction cause several dependent transaction to roll back or abort. Dis- CPU Performance low no helpful

. Cascadeless Schedule: transaction is not allowed to read a data untill the last transaction that has written it is committed or aborted. (but we can write) this is also problem

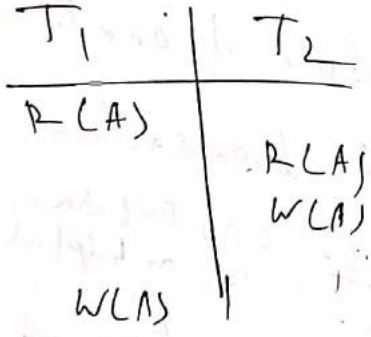
Analogy- Copying others answer in exam hall

Strict Schedule :=> A transaction can either read or write a data until the last transaction that has written is committed or aborted.



Serializability :=>

helps to identify whether a non serial schedule can be made serial or not to maintain consistency of data.



This is non-serial. To make it Serial we have to do $T_1 \longrightarrow T_2$ or $T_2 \longrightarrow T_1$

We have to find a clone schedule which is Serial

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

$T_1 \rightarrow T_2$

This is done transaction of previous

Types –

(i) Conflict Serializability :

non serial can be converted into serial by swapping its nonconflicting operation.

R(A) R(A) – Non conflict

R(A) W(A)
W(A) R(A) } conflict
W(A) W(A)

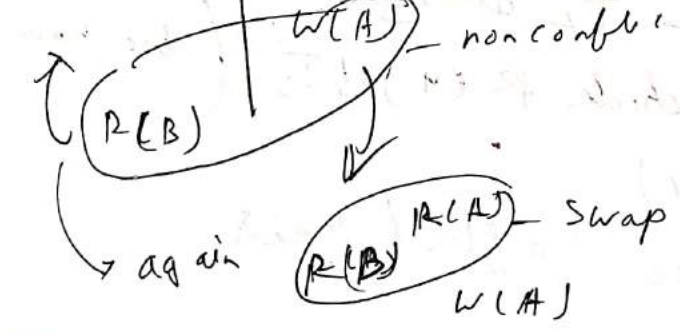
R(A)
R(B) R(A)
W(B) R(A) } non conflict
R(B) W(A)
W(A) W(B)

S

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

S'

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



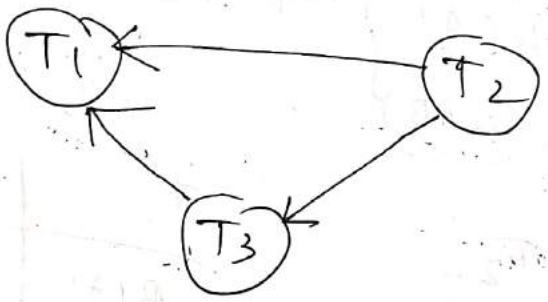
Precedence graph to check Serializability

S is a schedule which has 3 transaction

T_1	T_2	T_3
$R_1(X)$		
		$R(Y)$
		$R(X)$
	$R(Y)$	
	$R(Z)$	
		$W(Y)$
	$W(Z)$	
$R(Z)$		
$W(X)$		
$W(Z)$		

Conflict pair
R — W
W — R
W — W
on same var

- Number of vertices = number of transaction
- check conflict pair in other transaction and draw edges



- R(X) has a conflict pair W(X) but their is no W(X) in T_2 or T_3
- now we will check R(Y) of T_3 (we have to check sequentially)

R(Y) also has no conflict pair

- Next R(x) of T_3 has a conflict pair of W(x) of T_1
  So, we will draw an edge $T_3 \rightarrow T_1$
- Next R(y) of T_2 has a conflict pair with W(y) of T_3 $T_2 \rightarrow T_3$
- next R(z) of T_2 has a conflict pair with W(z) of T_1
- next W(y) of T_3 has no conflict pair
- next W(z) T_2 has a pair with R(z) of T_1

$T_2 \rightarrow T_1$ already drawn so no need to draw.

note - dont check conflict pair in own Transaction.

(*) We have to check any loop or cycle. If there is no loop or cycle then its conflict serializable

if there is any loop then it is not serializable

loop / cycle - if we start from T_1 & whether we can return to T_1

- No need to cover all nodes but we have to back starting point.

Serializable Possibility Sequence

$T_1 \to T_2 \to T_3$ $T_3 \to T_1 \to T_2$

$T_1 \to T_3 \to T_2$ $T_3 \to T_2 \to T_1$

$T_2 \to T_1 \to T_3$

$T_2 \to T_3 \to T_1$

but we will choose from the graph on the basis of in degree = 0

no → is going to that vertices.

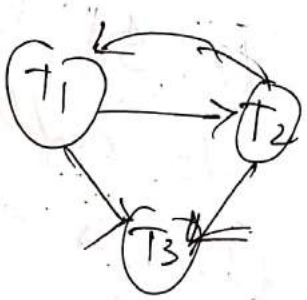
in this case T_2 in degree = 0 then we will delete T_2

$T_2 \to T_3 \to T_1$

– now T_3 in degree = 0

Check Whether Schedule is conflict serializable

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

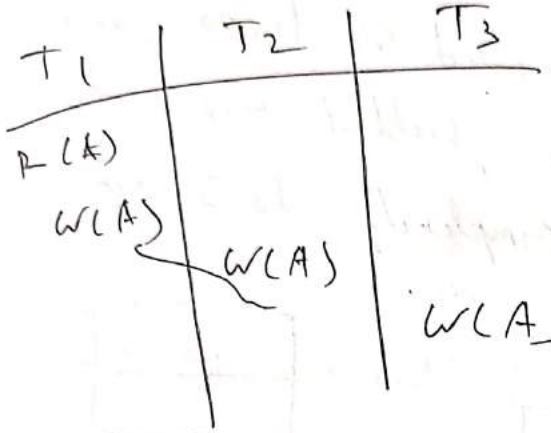


So there is not loop or cycle

if there is no loop then we can say this is serializable.

But if there is loop then we can't say that it is not serializable. This is not conflict serializable. But it can be serializable

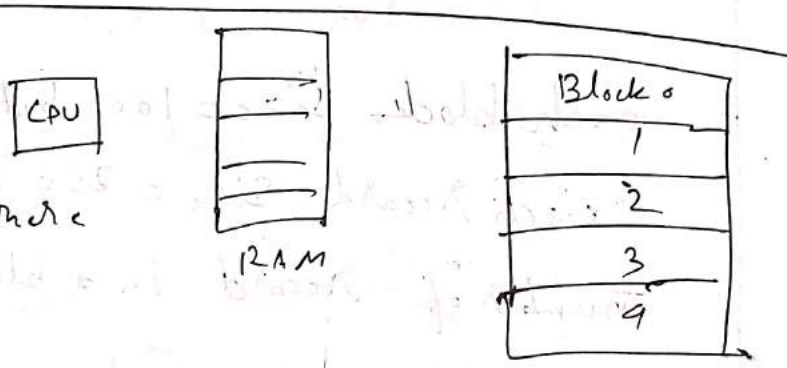
Thats why we use View Serializibility to check wheather it is serializable or not



- this is Serializable
- this view Serializability

In previous case W(A) of T_3 do the final write and here also.

Indexing :=>



Select * from stu where roll no = 1;

CPU can't directly communicate with Harddisk as it is slow (Harddisk)

Slow Harddisk

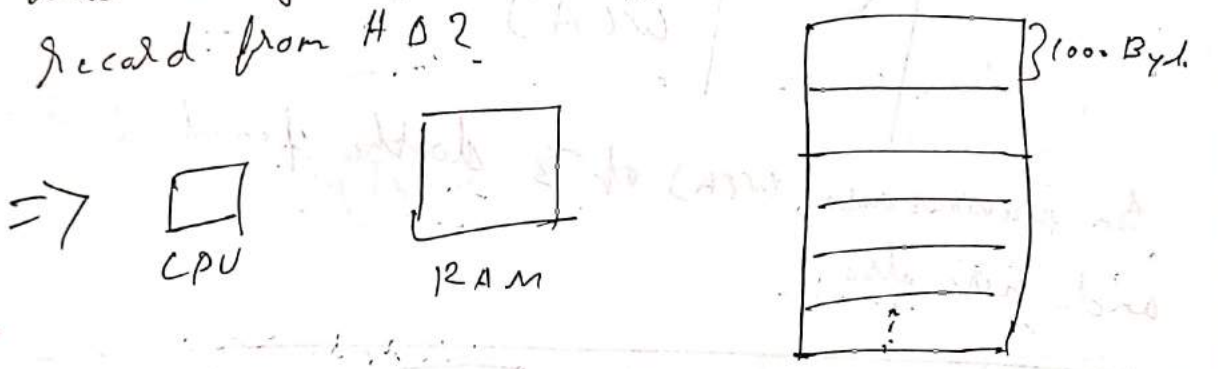
First the stu file will come to RAM then CPU will check for roll no = 1 data.

So we need to maintain index so that we can take the file of stu at very minimum time. else we have to take each block and search wheather our required data is in that block

* Indexing reduce the cost (as number of transferring block from harddisk to Ram is minimum)

ex → searching topic on book without index vs with index.

Q] Consider a harddisk in which block size = 1000 bytes. each record is of size = 250 byte, if total no of record is 10,000 and data entered in harddisk without any order. What is avg time complexity to search a record from HD?



each block size = 1000 byte

each record size = 250 byte

number of record in a block = $\frac{1000}{250} = 4$

∴ Total need of block = $\frac{10000}{4} = 2500$

Best case = 1 time (we get req rec in first)

worst case = 2500 time (linear search)

∴ Avg time = $\frac{2500}{2} = 1250$ search

if it was ordered then $\log_2 n = \log_2 2500 = 12$ (binary search)

• What is avg time complexity to search a record from Index table if index table entry of 20 byte (key + pointer)
10B 10B

⇒ index also stored in secondary table.
in index table there is key (rollno/ age/name)
and pointer (Page no/block no)

each block of harddisk can have = $\frac{1000 \text{ byte}}{20 \text{ byte}} = 50$ index entry

Dense — Same number of entry in index with number of entry in harddisk.

Sparsh ⇒ One entry for one block.
→ 2500 entry
→ 10000 entry

(*) index is ordered (binary search) ≠ unique(key)

So we need to take block in ram in case of Dense

total block needed = $\frac{10000}{50} = 200$

$\log_2 200 \leq 8$ (upper value) + 1 (for page taking to ram)

⇒ 9 search

in case of Sparsh

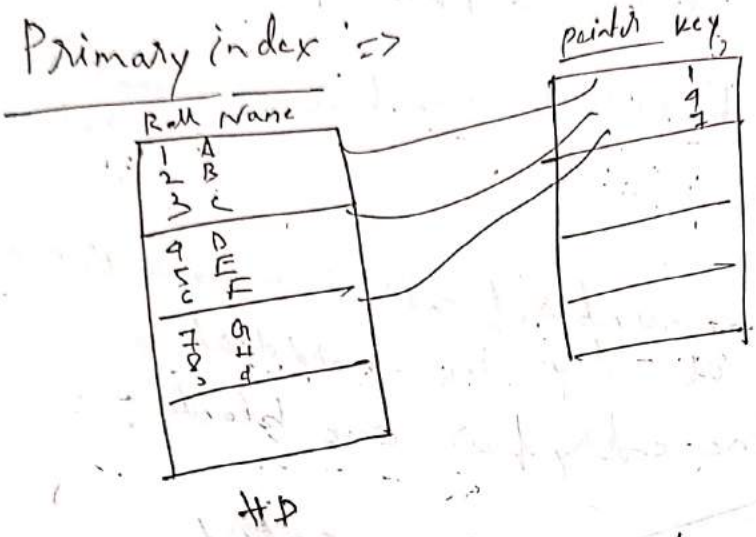
$\log_2 50 + 1 = 6 + 1 = 7$ search

Types of indexing :

(i) Primary index
(ii) clustering index
(iii) Secondary index

	key	non key
ordered	Primary (Sparse)	Clustered (Sparse)
unordered	Secondary (Dense)	Secondary

Primary index =>

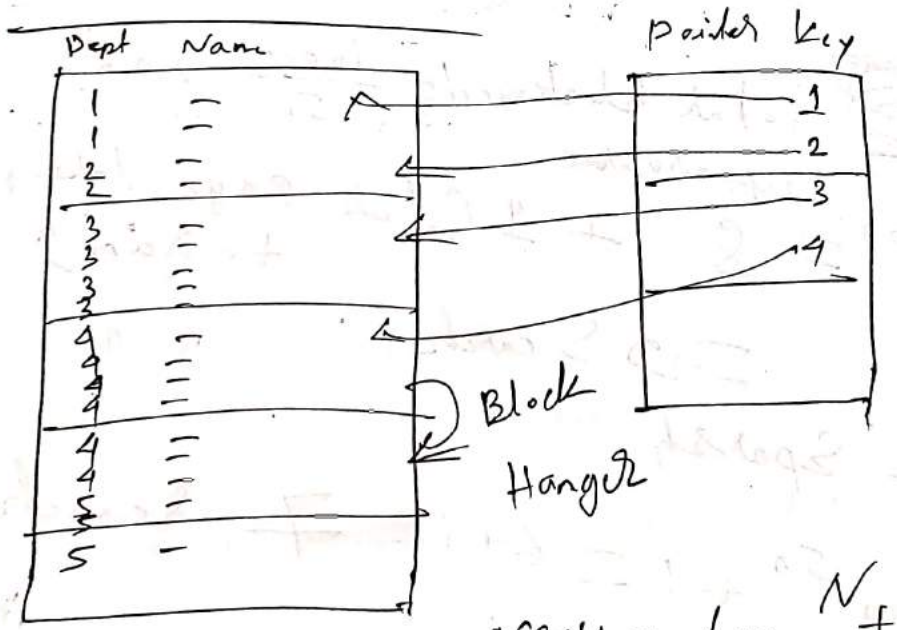


- it is sparse (ordered) one entry for one block

Average number of block access using index $= \log_2^{N+1}$

N = number of block in index table

Clustered index

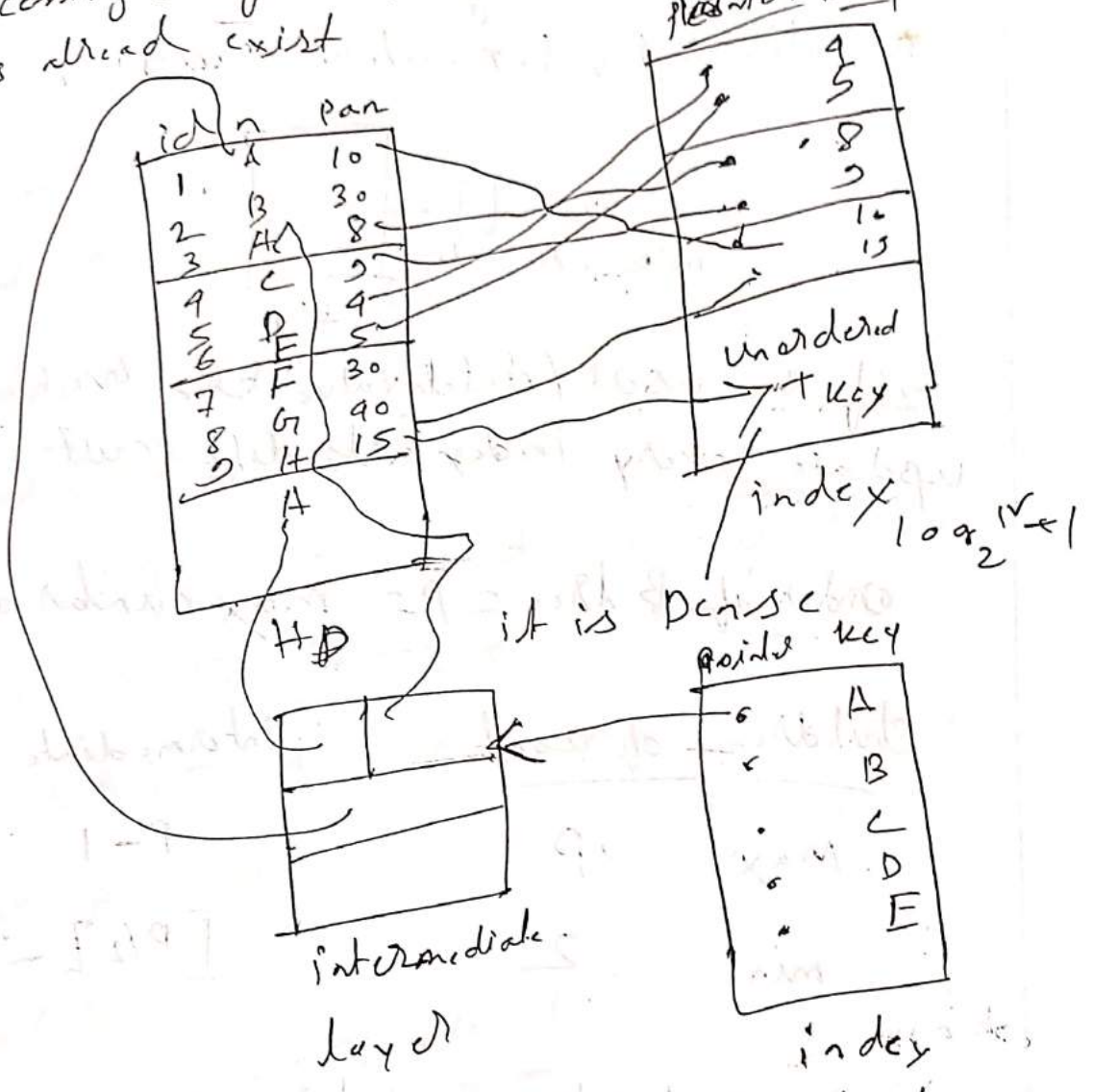


- it is sparse (ordered) one entry for one block

access $= \log_2^{N+1} +$ no of block header

Secondary index :=>

Secondary index provides secondary means of accessing a file for which primary access already exist

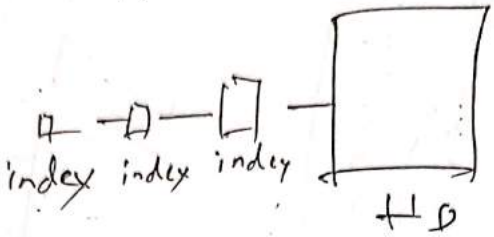


Sparse + Dense

$\log_2 N$ + 1+ other intermediate block pointer

B tree :⇒ We use balance tree for multilevel index

if index is big then we maintain an other index to maintain index



if we insert / delete value then we have to update every index. its difficult

order of B tree = P = max number of childs

Children of Root		intermediate node
Max	P	P − 1
min	2	$\lceil P/2 \rceil - 1$

max children = order = P

∴ key = P − 1

Record Pointer = P − 1

A	B
1	Null
2	Null
3	Null
4	Null
Null	Null

count(*) = 5 (number of rows (it count null))

count(A) = 4 (it does not count Null)

count(B) = 0 (count of null value is 0)

Sum(A) = 10 (does not include null)

Avg(A) = $\frac{10}{4}$ = 2.5

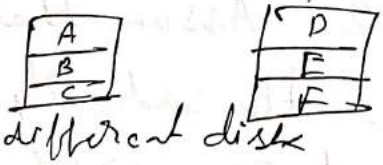
Min(A) = 1 (as null means no value)

Max(A) = 4

Sum(B)
Min(B)
Max(B)
} = Null

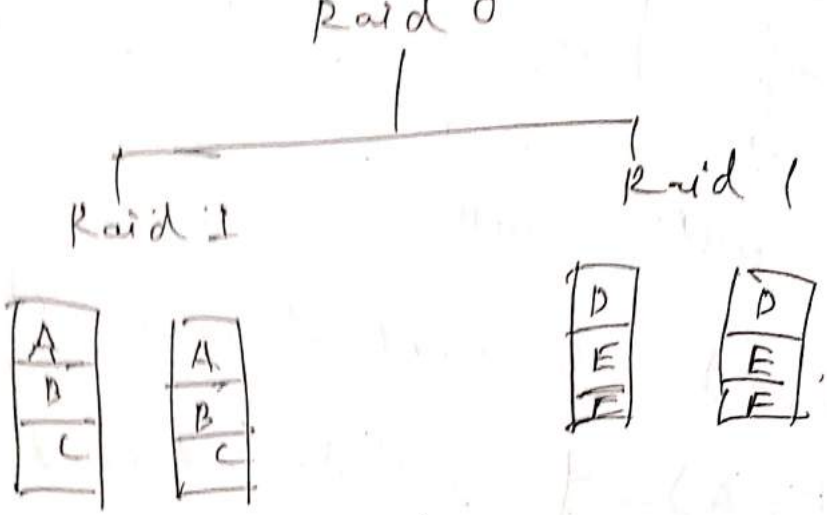
RAID - Redundant array of independent disks

Raid 0 - different data in different disk

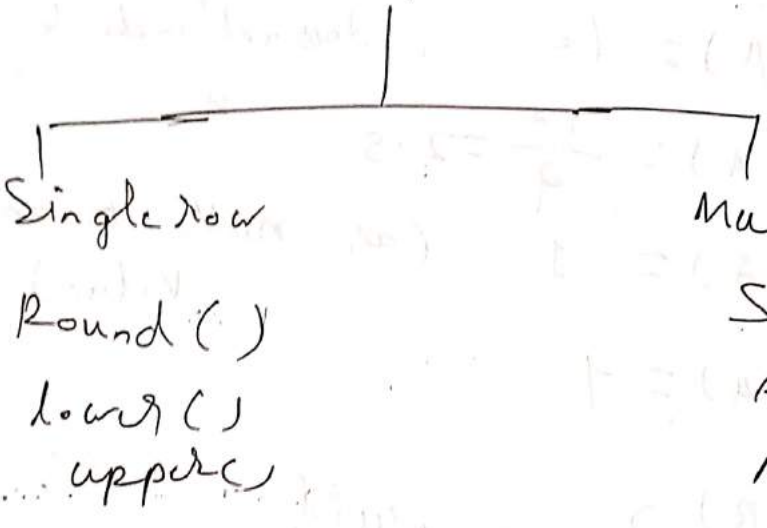


Raid-1 : mirror data. copy data

Raid 0+1 : we Splitting + mirror



SQL function



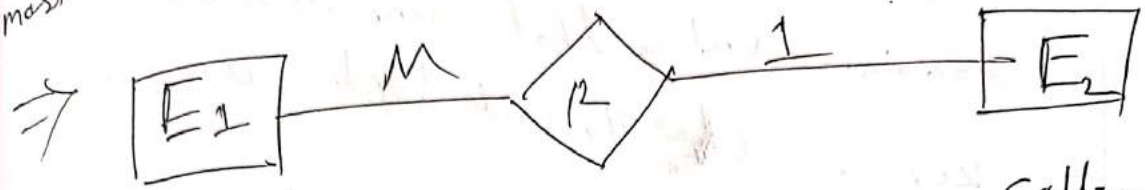
Q) In an ER model, suppose R is many to one relationship from entity set E_1 to entity set E2. Assume that $E_1 \neq E_2$, participate totally in R. the cardinality of E_1 is greater than the cardinality of E2. Which one is true about R?

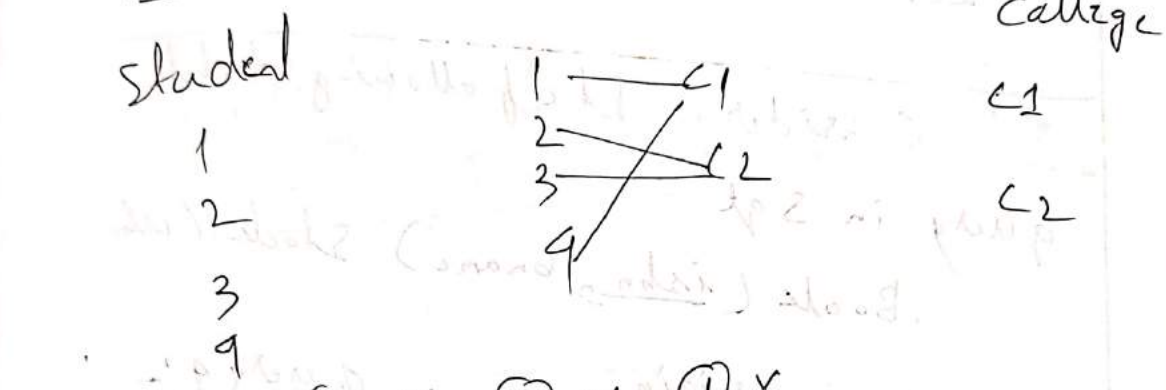
(a) Every entity in E1 is associated with exactly one entity in E2.

(b) Some entity in E1 is associated with more than one entity in E2

(c) Every entity in E2 is associated with exactly one entity in E1.

(d) Every entity in E2 is associated with at most one entity in E1.





(a) ✓ (b) ✗ (c) ✗ (d) ✗

Q. The view of total database content is

(a) coceptual view (b) internal view

(c) external view (d) Physical view

→ (a) ✓

Q. In relational model, cardinality is

(a) no of tuples (b) No of attribute

(c) No of tables (d) No of constraints

→ (a) ✓ ~~(b)~~ degree = no of constraints

Q) ___ constraint is used to maintain Consistency among tuples in two relation
(a) key (b) domain (c) referential integrity
(d) entity integrity

→ (c) ✓

domain - value that one attribute take
key - classify each tuple uniquely

Q) Consider the following table and query in Sql

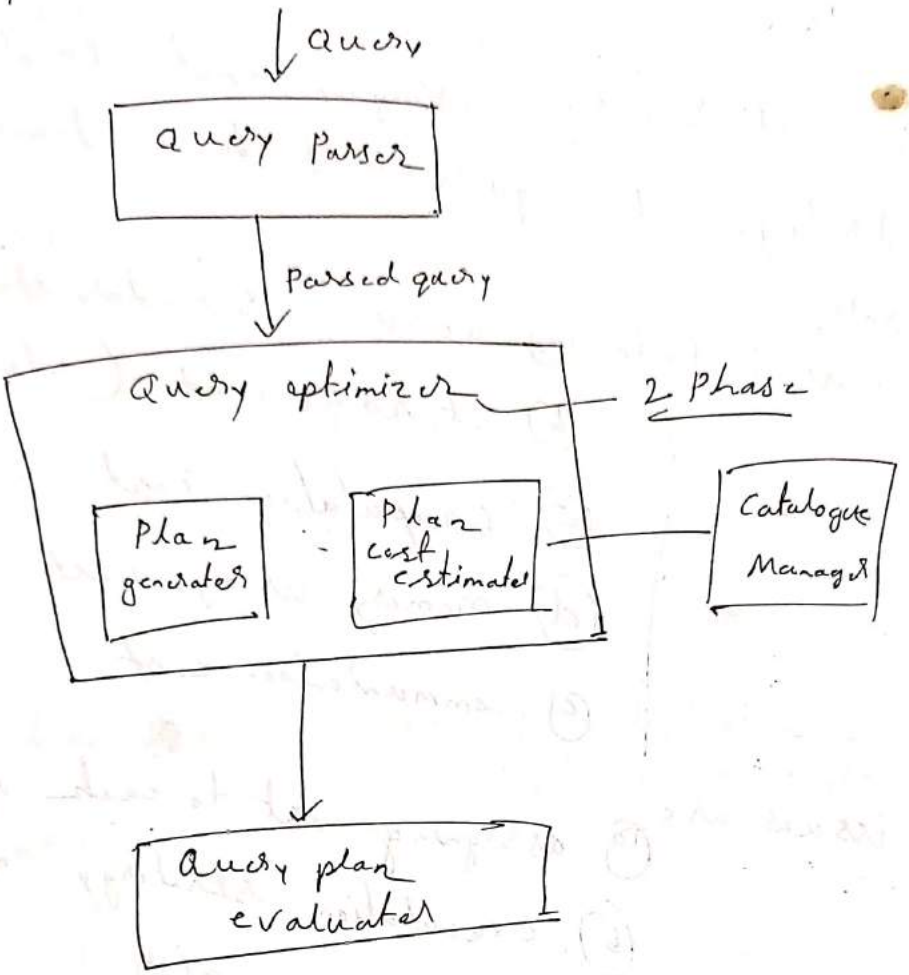
Book (isbn → bname) Stock (isbn copies)

query 1 - inner join
query 2 - Left outer
query 3 - Right outer
query 4 :- full outer

Which query is certain to have an output that is a superset of the o/p of the other three queries?
(a) Q1 (b) Q2 (c) Q3 (d) Q4

⇒ inner join = natural join
Lo j = inner join + left table left
Ro j = inner join + Right table left
Full oj = inner join + left + Right table

Query Optimization => It is the process of selecting an efficient execution plan for evaluating the query.



optimization is used for accessing the database in an efficient manner.

Ex-> Select Sno from Student where Sno = '101'

This is translated into

$\sigma_{Sno='101'}(\pi_{Sno}(Student))$

$\pi_{Sno}(\sigma_{Sno='101'}(Student))$ ← efficient

Two techniques are

(i) Heuristic (logical) optimization.

(ii) Cost based optimization

(ii) cost based :=> assign cost to different strategy of a query and choose lowest cost strategy.

Cost include (a) accessing secondary storage (disk block)

(b) storage cost (intermediate result)

(c) computation cost

(d) memory usage cost

(e) communication cost

issues are (a) assigning cost to each operation

(b) execution strategy not fixed

(c) time consuming

(d) do not gurrantee best optimized

(i) Logical :=> We use some predefined rules. It usually help to optimize cost but not always. It is based on query tree

(a) apply select operation before project, join or other binary operation

(b) apply project operation before join or other binary operation

It reduces size of files.

(*) Field means Columns, Records mean Rows } keys are constraints

(*) A relational database consist of a collection of table.

(*) column has only set of values.
row has different set of values.

(*) Column = attribute = Field
Row = Record = tuple

(*) Relational Algebra = Procedural language
Domain relational & tuple relational = non Procedural

(*) varchar (20) = 'hi' it will take 3 space as it changes length according to it.
char (20) - has fixed length

(*) Delete from R — Delete command remove the entries in the table.

(*) Trim is used to remove spaces at the end of the string

|| — Concatenation operator (append two string)

(*) Union eliminate duplicate
union all - Retain duplicate

(*) Degree = number of attribute in a relation

• inner join are SQL Server default join

- aggregate functions take a collection of values as input
- Avg () = mean value
- We use distinct to represent unique value
- count (*) only include null value, others dont include it (aggregate function)

(X) With clause: It is used for temporary relation.

(X) To include aggregate function having clause must be included after Where we cant use only in Where only.

(X) lateral ⇒ used to access attribute of preceding table.

(X) Subqueries can not join table.

(X) do not match some specified condition — double use of Not exists

---

ER diagram : Set of entity = entity set

Relationships are — unary, binary, Ternary
(Relation between same set) (two set) (Three set)

Participation constraint ⇒ Specify the presence of an entity when it is related to another entity

(i) Total — all entity is in relationship (double line)

(ii) Partial :- Some entity involve in relationship —