1)

Database System

Operations

Database
[Collection of Related data]

DBMS
- SQL Server → by microsoft
- Oracle 9i, 11, 12c etc
- My SQL → by oracle too.
- DBL → ibm

(10%) Structured → by RDBMS
→ we can store in particular structure
ex IRTC
University

(90%) Unstructured
- Webpages

eg photos, video, paragraphs--
eg. big data

RDBMS → Relations → table

2)

File systems Vs DBMS

CISF, NISF → file systems

DBMS is used because we are using client-server structure, means many user want to access the same data therefore we cannot use filesystem

1) Searching is fast in DBMS & less memory usage as compare to filesystem.
For ex- if we open filesystem of trains then we get all 25GB data about the trains, but at the same time by using SQL query we can took off 1KB of data from that.

2) Attributes

File location] metadata
File permissions]

to we can access data only if we know about metadata
but no need metadata in DBMS.

DBMS → Do Not Require's Attributes
Filesystem → — " "

to Searching is faster in DBMS (?)

3) Concurrency

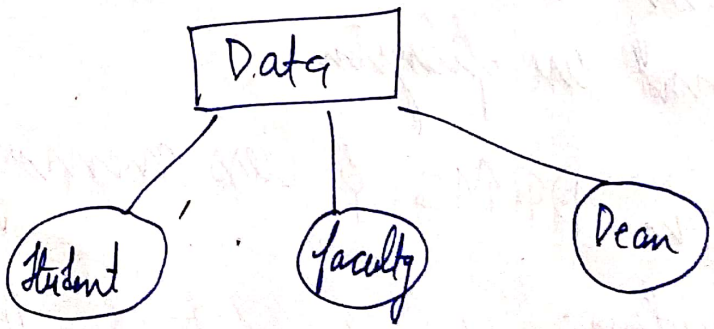
Many user's using data at the same time.

DBMS → Provide protocol for concurrency
Filesystem → Don't " " " " , hence lead to data inconsistency

4) Security

DBMS → Role based access control provides

Eg



O.S. uses controls filesystem & don't provide any role based access control (Hierarchical security)

5) Redundancy

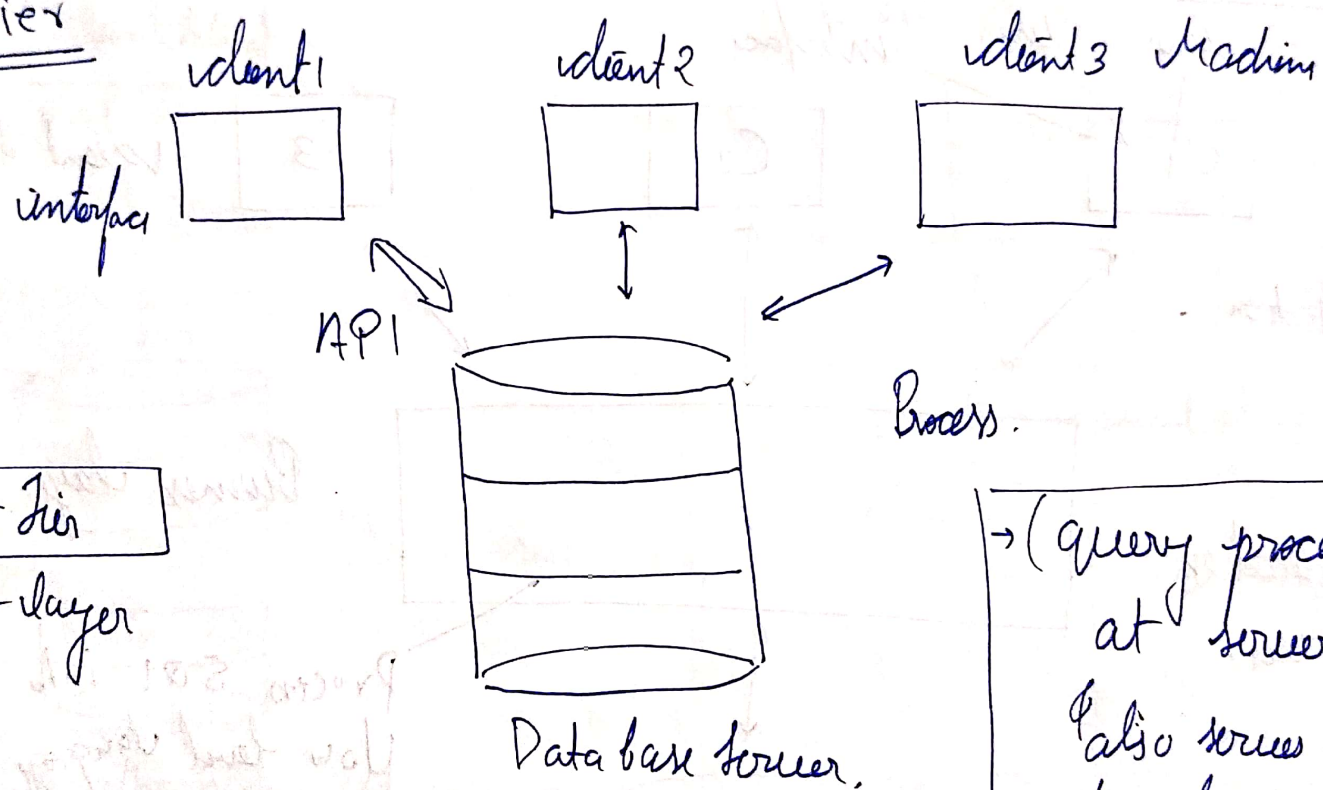
Advantage

DBMS stores unique data

→ DBMS is used as always at backend so stores the data uniquely, although at front end data may not be unique.

2-Tier And 3-Tier Architecture

2-Tier



2-Tier
(2-layer

→ (query process at server) & also serves data to client
→ So all loads on data base server.

ex Ticket Booking
Bank.

* Easy maintanance.
* As no. of clients are v. large, hence they are not helpful

Server has v. large load.

Advantage

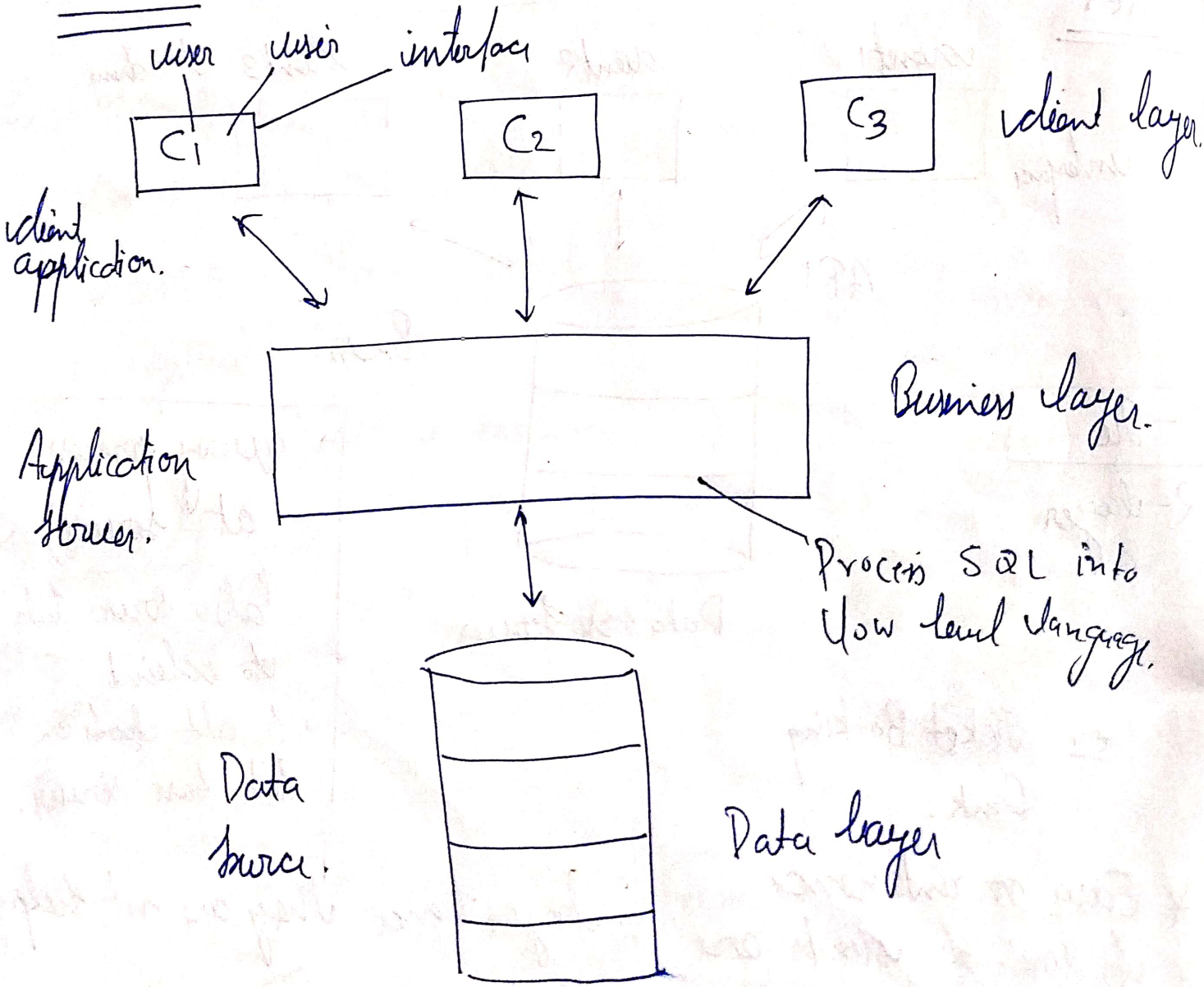
→ Easy to maintain

Disadvantage

Security → No direct link with server.

Scalability → due to large no. of client

3-Tier



Advantage

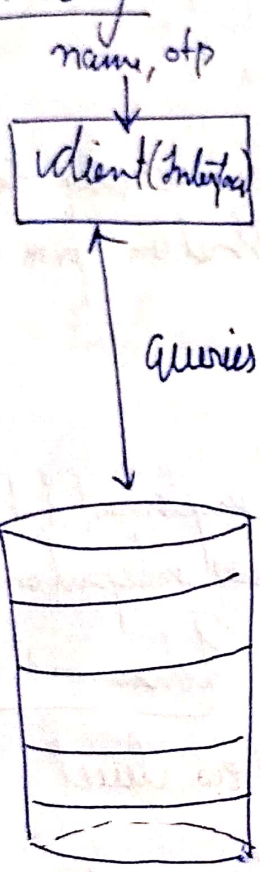
Security
Scalability
Less load on server.

Disadvantage

Maintanance
Complexity.

* Interface → are the programs that generates query.

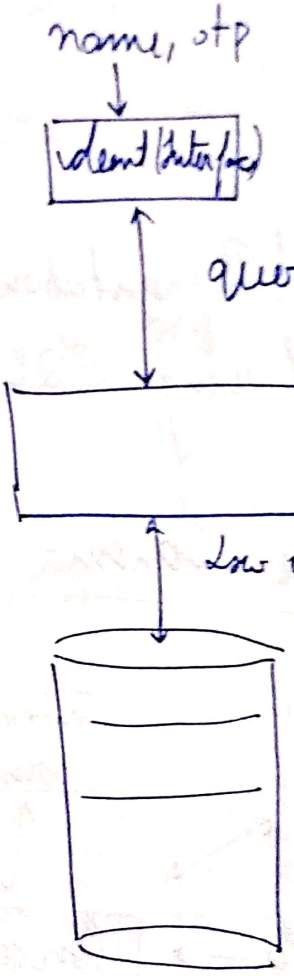
Summary



Data server.

Converts the high level language that are query into low level languages, process it & send data back to client

2 - Tier



Data server.

Server convert process the low level language code & process it & send data back.

* Business layer converts queries into low level language

3 - Tier

5) "Schema" → Logical Representation

By RDBMS

Tables

Student

Roll no.	Name	address

Course

Cid	name	Duration

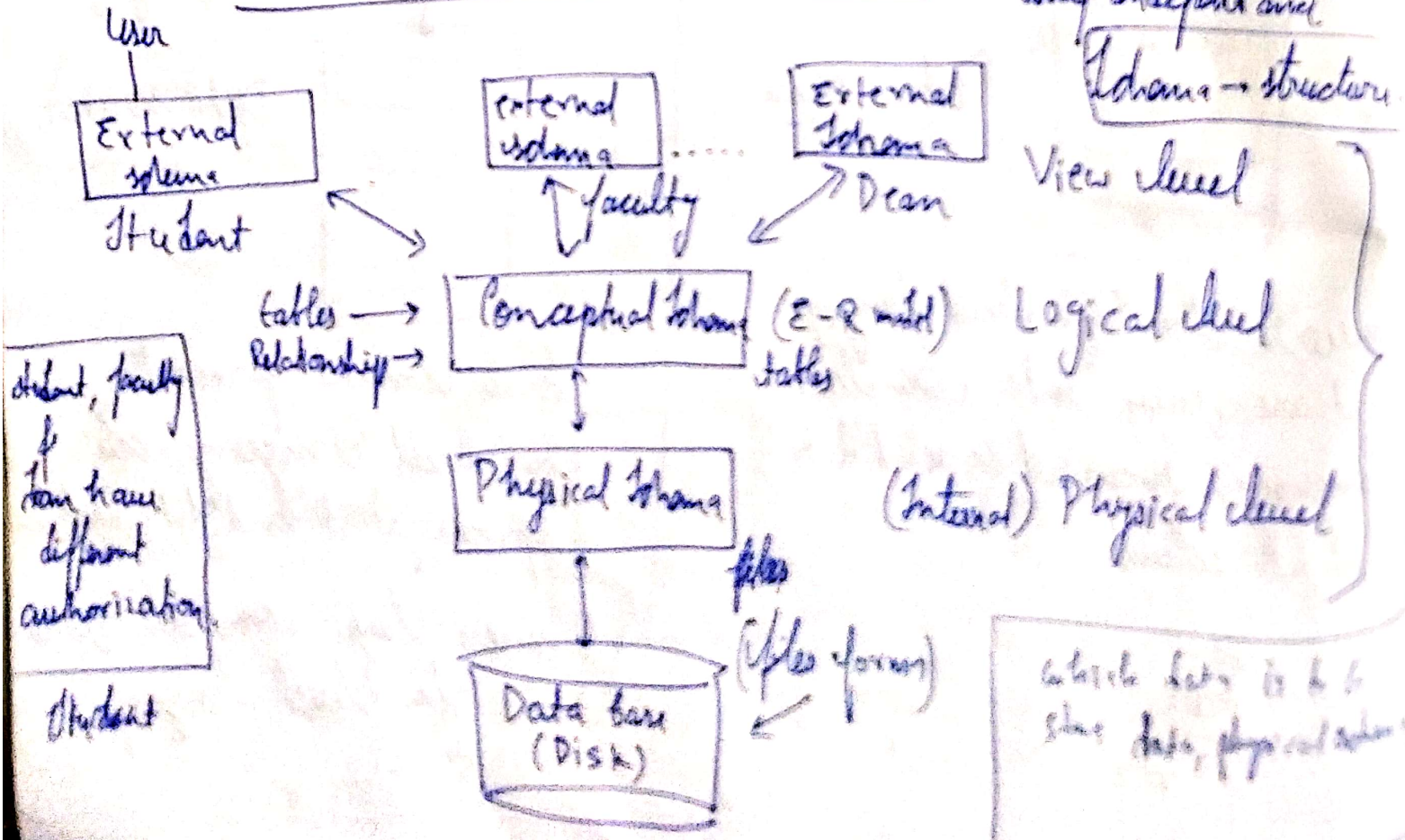
} Logical Representation.

But actually they are stored in form of file.

These Logical Representation can be implemented using SQL → DDL.

6) Three Schema Architecture → To implement data independence

Schema → structure



Tables

Student → Name, roll no, address.

Relationship

Student study courses

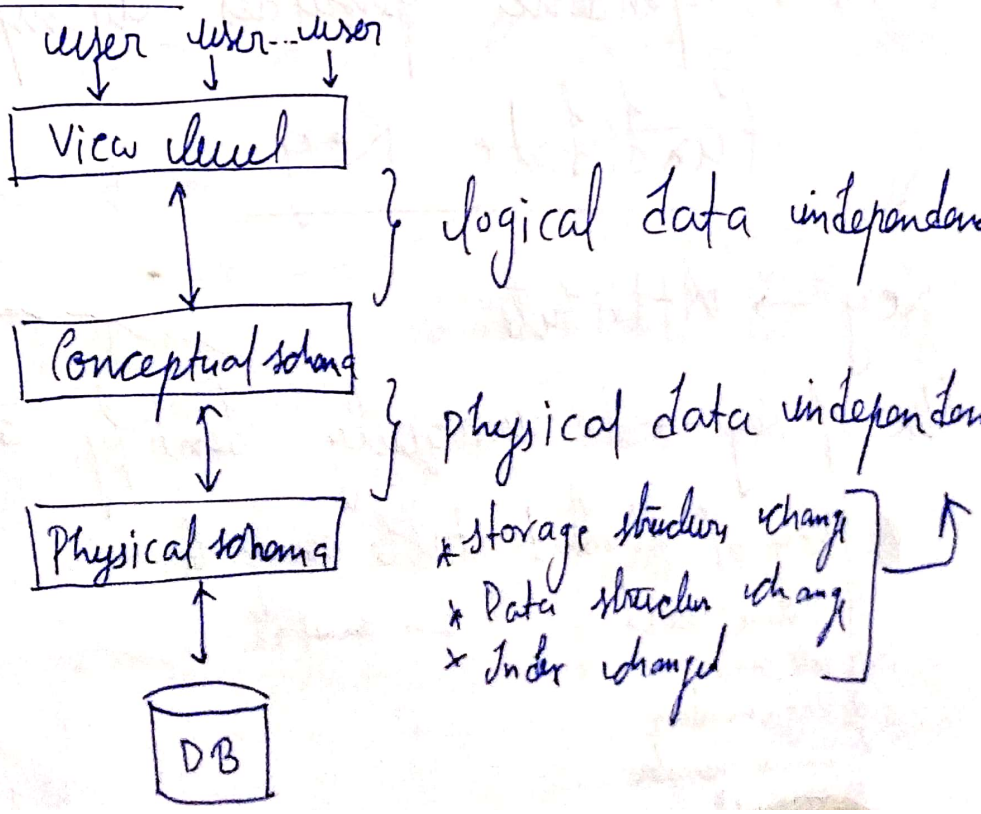
View level → PHP, java → for interface, front end design.

Logical level → They logically design → make the print

Physical level → Actually design all managements at physical level., data base administrator stays here.

Data Independance → Means there is no direct link of data with it the clients.

"Data Independence" → Provides transparency.



Changes at Conceptual schema doesn't affect appearance at view level → Logical Data independence.

Means when we don't have to change application program at the view level.

Physical Data Independence

If we change data at data base from 1 hard disk to another hard disk this is physical data independence.

Or if instead of queue, data bases changes to stack, it doesn't change conceptual level.

So changes at physical schema does not affect conceptual schema.

* Data Independence provides transparency (?)

Candidate Key

Key → Attribute

Use of Key → To uniquely identify any two tuple in the table.

Key of student table may be

1) Adhar card
2) Roll no.
3) Registration
4) License number
5) Voter ID
6) Phone number
7) email id

Roll no	Name	City	Age
1	Reddy	Allahabad	20
2	Ahad	Kanpur	21
3	Reddy	Allahabad	20

Set of all key is → candidate key.

< Athar card, Voter ID, License no, Rollno, phone no, email)

From the candidate key → we pick most suitable for unique identification.

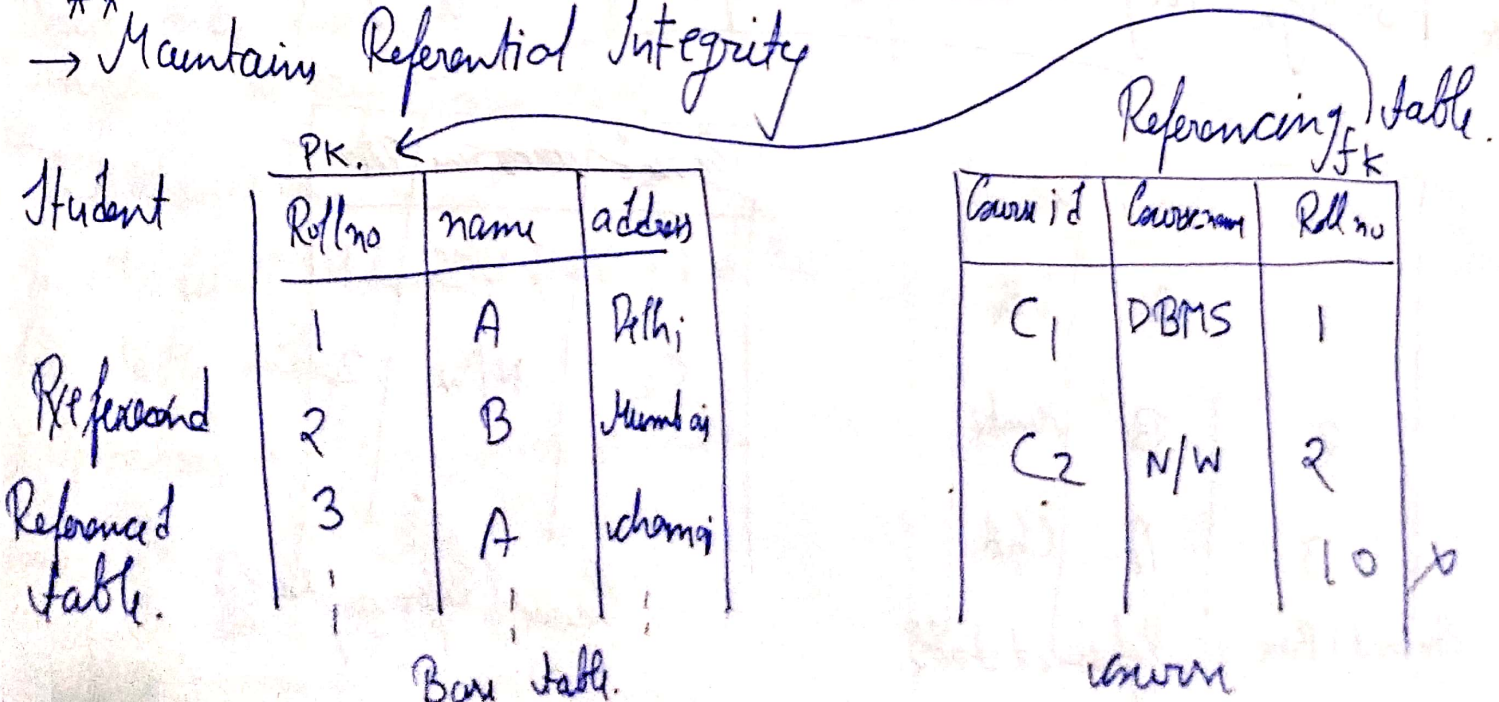
Rest of candidate key are alternate key

Foreign Key

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

**

→ Maintain Referential Integrity



during table creation

```
Create table Course
(
    course_id varchar(10)
    course_name varchar(20)
    Roll no int references Student(Roll no)
);
```

After table course is created

```
After table course.
ADD constraint fk
foreign key (Roll no)
references Student (Roll no)
```

name may be different

* There might be more than one foreign key in a table but there is only one primary key & it is not NULL.

Now

* Foreign key maintain referential integrity.

PK

Roll no.	Name	address
1	A	Delhi
2	B	Mumbai
3	A	Chd.

Student (Base or Referenced table.

course_id	course_name	Roll no
C_1	DBMS	1
C_2	N/W	2

Course (Referencing table)

Referenced table.

1) Insert – No Violation
2) * Delete → May cause violation.

Sol^{n} → i) On delete cascade.

It delete data of particular Roll no's of row in primary / referenced table as well as all it referencing table.

ii) On delete set NULL

If we delete particular row in base, then all the referencing table have Roll no as NULL that is foreign key.

* But if foreign key is acting as primary key in referencing table. Then we cannot set it as NULL

iii) On delete No action

Delete only base table row & not to anything in referencing table

Eg: But it will show error.

To 1st delete Referencing table row & then base table entry

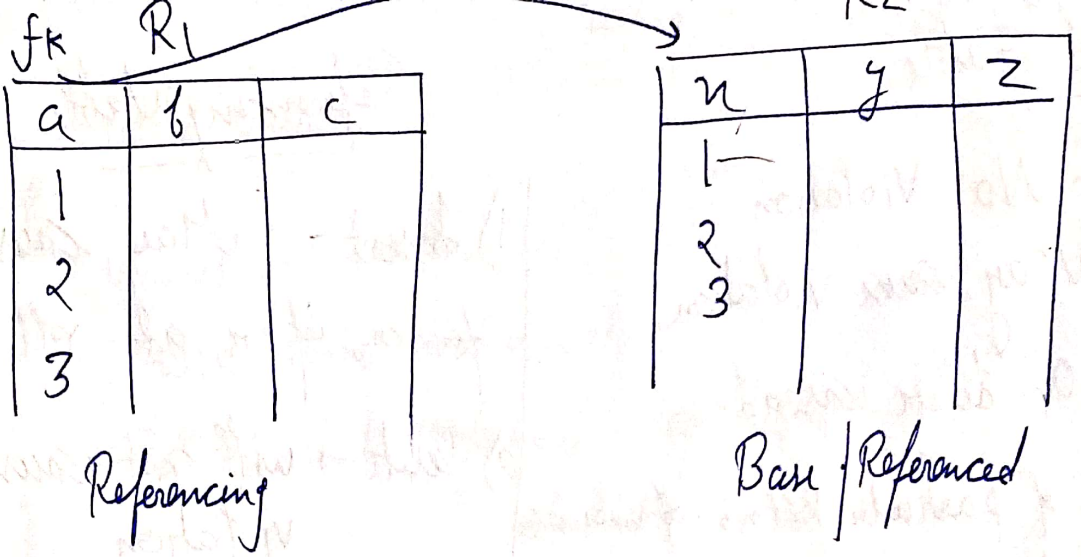
3) Updation → May cause violation

same. { On update cascade
On delete update NULL
On update no action

Referencing table.

1) Insert → May cause violation for ex. if we add roll 7 at Course
2) Delete → will not cause any violation
3) Updation → May cause violation.

Ex



Insert into R_1 → violation.
Insert into R_2 → No "
Delete from R_1 → No "
Delete from R_2 → violation.

Super Key

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

** Super set of any candidate key is super key.

Rollno	name	age

Let Candidate Key = Rollno.

Super keys { Rollno, name
Rollno, age
Rollno, name, age.
, but name, age is not super key.

Candidate key + any key = super key.

$R(A_1, A_2, \dots A_n)$ then how many super keys can possible

a) if A_1 is candidate key.

$$2^{n-1}$$ super keys

b) if A_1 & A_2 candidate key.

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

c) $\underbrace{A_1 A_2}_{CK_1}$ $\underbrace{A_3 A_1}_{CK_2}$

$$2^{n-2} + 2^{n-2} - 2^{n-4}$$

(For conceptual view) ←

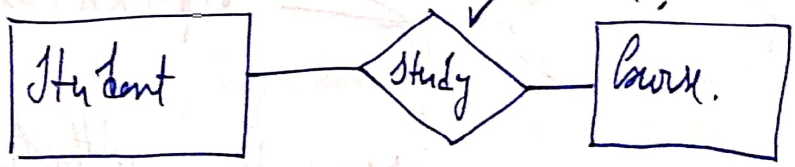
Entity - Relationship Model (E-R model)

Entity → Any thing/Object that have physical existence.

Student (Roll no., age, address)

Roll no
age
address
→ attribute

entity type or Schema



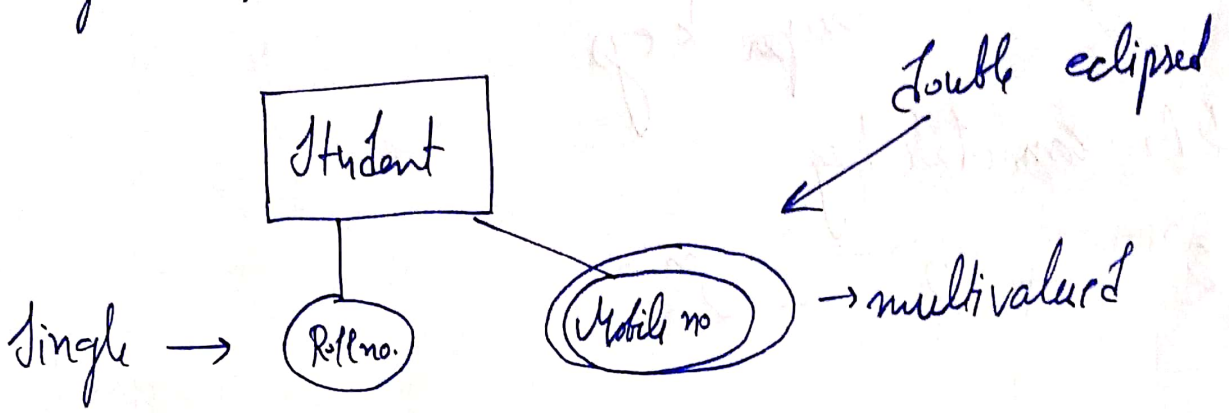
Entity → ▭

Attribute → ⬭

Relationship → ◇

Type of Attributes

1) Single v/s Multivalued Attributes



2) Simple v/s Composite Attributes

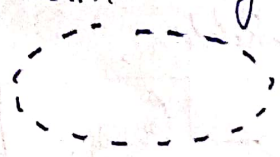
Simple → Cannot be further devided. ex. age.

Composite Attribute → Constitute of single albributes ex name
first name + middle name + last name

3) Stored or Derived attribute

Stored → normal attribute that is not derived, DOB

derived → age → it derives from DOB.

 ← representation

4) Key v/s Non key Attributes

Key → uniquely identify each row ex. Roll no. (Unique)

Non-key Attribute → can/cannot " " " " ex Name.

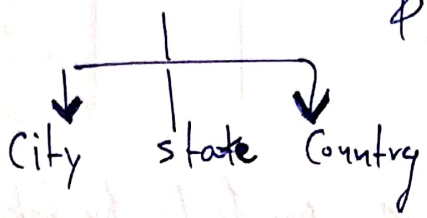
* key attribute is represented by —— ("Underline")

5) Required V/s Optional Attributes ER
(mandatory)

6) Complex attribute
(Composite + multivalued)

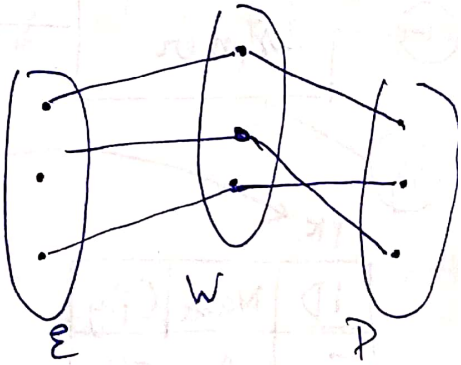
ex address

& may be have same value



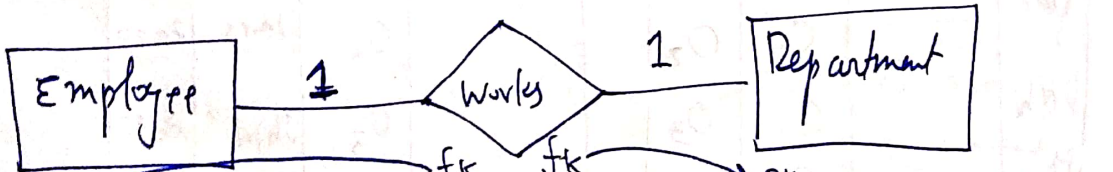
3) Degree of Relationship (Cardinality)

One to one	1–1
One to many	1–M
many to one	M–1
many to many	M–M (M–N)



One to one

?



Eid	Ename	age
E_1	A	20
E_2	B	22
E_3	C	28
E_4	A	24
E_2	B	22

EID	DID
E_1	P_1
E_3	P_2
E_2	P_3

Did	Dname	Location
P_1	IT	Bangalore
P_2	Product	Delhi
P_3	HR	Delhi

PK = either Eid or Did
but PK= E id

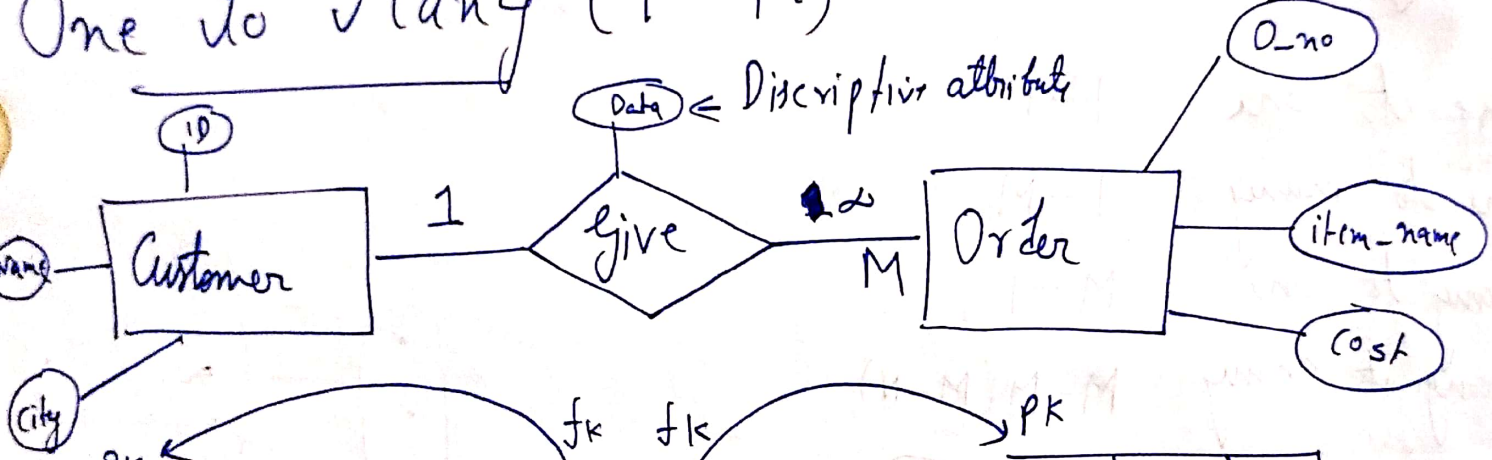
After merging

EID	Ename	age	D_ID
E_1	A	20	D_1
E_2	B	25	D_3
E_3	C	28	D_2
E_4	A	29	—
E_5	B	22	—

Did	Dname	Locn
D_1	IT	Bang.
D_2	Prod.	Delhi
D_3	HR	Delhi

I can also merge work with dept. by taking Did as PK.

One to Many (1–M)



PK

ID	Name	City
C_1	A	Tale.
C_2	B	Delhi
C_3	C	Mum.
C_4	A	Mum.

fk fk

ID	O_no	Date
C_1	O_1	
C_1	O_2	
C_2	O_3	
C_2	O_4	

PK

O_no	Item_name	Cost
O_1	Bucket	1000
O_2	Shoes	2000
O_3	Shirt	1500
O_4	Jeans	2000

PK = (O_no)

customer can give many order.

* Each table have 1 P.K.

* Attribute of Relation is descriptive attribute

After merging

ID	Name	City
C_1	A	Taly
C_2	B	Delhi
C_3	C	Mum
C_4	A	Mum

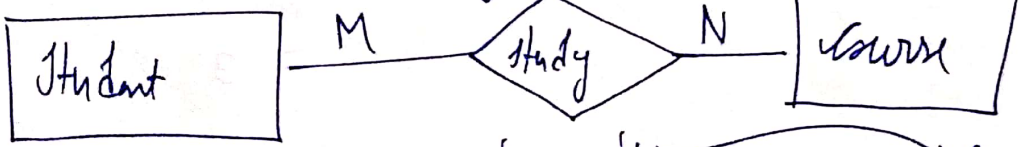
ID	O_no	item_name	Cost	Date
C_1	O_1	Bucket	1000	
C_1	O_2	shoes	2000	
C_2	O_3	shirt	1200	
C_2	O_4	Jeans	2000	

Customer cannot be merge with "Give" because value ID repeats.

So primary key of relationship table is same as M side (PK) & relationship table merge with M side table only & it is applicable for many to one only.

Many to Many

Designing



PK ← FK FK → PK

Rollno.	name	age
1	A	16
2	B	17
3	A	16
4	C	17
5	D	18

Rollno	C-id
1	C_1
2	C_2
1	C_2
2	C_1
3	C_3

Cid	name	credit
C_1	Maths	4
C_2	Phy.	4
C_3	Chem	4
C_4	Hindi	4

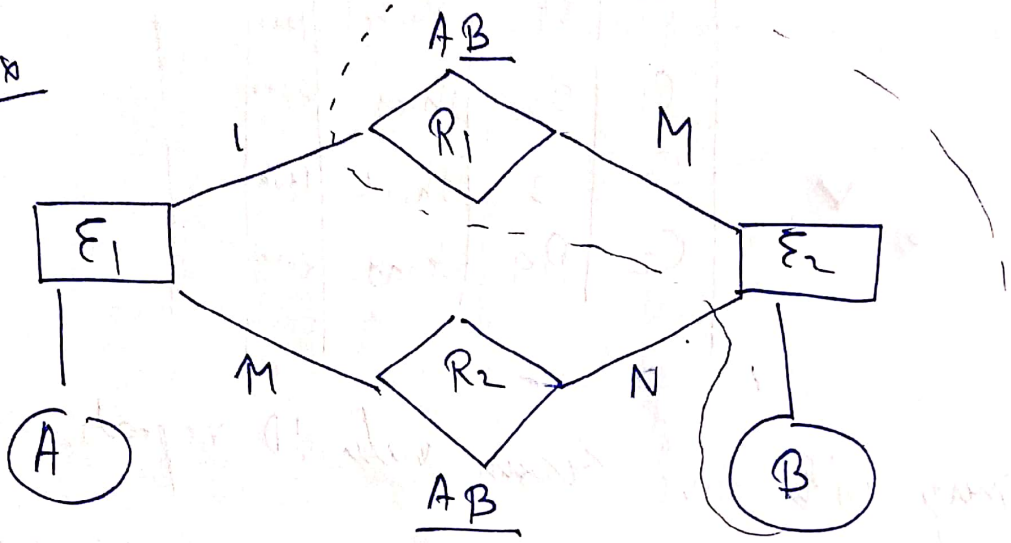
PK in Referencing table = Rollno + [-id]

Composite key

These table cannot be combine

⇒ M-N → No Reduction

Ex



What is the min. no. of tables required to represent this E-R model into Relational model?

E1
E2
R1
R2
} Table

As R1 & E2 can combine

No Total E1 R1&E2 R2

1 + 1 + 1

= 3

Normalization

It is a technique to Remove or Reduce Redundancy from a table.

Course = C Faculty = F

PK

SID	Sname	Age
1	Ram	20
2	Varun	25
1	Ram	20

Row level Redundancy

PK

SID	Sname	Cid	Cname	FID	Fname	Salary
1	Ram	C1	DBMS	F1	John	30000
2	Ravi	C2	Java	F2	Bob	10000
3	Nitin	C1	DBMS	F1	John	30000
4	Amrit	C1	DBMS	F1	John	30000
⋮						

Course level Redundancy

Ist^n

PK.

only this data is given no SSID

C10, MBBS →
- Insertion anomaly → what should be SID
- Deletion Anomaly → deletion of student also delete faculty info
- Updation Anomaly → update at faculty salary take place in each column for the same faculty!! (due to column level duplicacy)

Because it is possible that only one student studying that course.

Ist^n

SID	Sname

CID	Cname

FID	Fname	Salary

First Normal Form

Table should not contain any multivalued Attribute

Not in 1st N.F.

Student

Rollno	Name	Course
1	Jai	C/C++
2	Harsh	java
3	Omkar	C/DBMS

1st Representation

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

PK = Rollno Course

* 2nd Representation

Rollno	Name	Course1	Course2
1	Jai	C	C++
2	Harsh	java	NULL
3	Omkar	C	DBMS

PK = Roll no

* But have problem - if only 1 student that have 1 Course & remaining have 2 Course, then there is lot of NULL values

3rd representation

Roll no	Name
1	Jai
2	Harsh
3	Omkar

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

Base table

PK = Roll no

Primary key: Roll no Course

Foreign key = Roll no

Closure Method

To find all candidate key.

ex R(ABCD)

Given functional dependency → {A → B, B → C, C → D}

$A^+ = BCDA$

$B^+ = BCD$

$C^+ = CD$

$D^+ = D$

So A can determine all attributes → C.K.

* A determine C by transitive prop.

* A determine A by reflexive prop.

Candidate key, CK = {A} | PA = {A} | NPA = {BCD}

$(AB)^+ = ABCD$, but AB is not C.K. as A is already CK & CK is minimal & hence AB is superkey.

ex R(ABCD)

FD = {A → B, B → C, C → D, D → A}

$A^+ = ABCD$

$B^+ = BCDA$

$C^+ = CDAB$

$D^+ = DABC$

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

Prime Attribute → That is used in making C.K.

Prime Attribute = {A, B, C, D}

Non prime Attribute = {ϕ}

ex R(ABCDE)

FD = {A → B, BC → D, E → C, D → A}

Those attribute that not occur at Right side of FD, then that attribute should be used to make Candidate Key.

i.e. E = BDCAE

as E⁺ = EC

So AE⁺ = ABECD

BE⁺ = BECDA

CE⁺ = CE

DE⁺ = DEABC

CK = {AE, BE, DE}

PA = {A, D, B, E}

Non Prime Attribute = {C}

Or

As AE become CK, Now check if A is present at right side of any, then replace that with A, let it is D, as D → A, so DE is also CK.

Similarly BE = CK.

Functional Dependency

Gives relationship b/w attributes

$X \longrightarrow Y$

X Determines Y

or

Y Determines X

, X → Determinant
Y → Dependent attribute

ex

Sid → Sname

1 → Ranjit

2 → Ranjit

, hence FD help to remove confusion.

valid ✓

Sid	Sname
1	Ranjit
1	Ranjt

valid

Sid	Sname
1	Ranjit
2	Ranjt

Valid ✓

Sid	Sname
1	Ranjit
2	Varun

Invalid ✗

Sid	Sname
1	Ranjit
1	Varun

Functional Dependency

1) Trivial functional dependency , $X \rightarrow Y$, then Y is subset of X

→ It is always true.

→ LHS ∧ RHS ≠ ϕ

ex ~~Sid ∧ Sname ≠ ϕ~~ , as Sname is subset of Sid

SidSname → Sid

SidSname ∧ Sid ≠ ϕ

Non-Trivial FD

$X \rightarrow Y$, $X \cap Y = \phi$

$S_{id} \rightarrow S_{name}$

$S_{id} \rightarrow S_{mester}$

$S_{id} \rightarrow$ Phone no.

$E_{id} \rightarrow E_{name}$

$E_{id} \rightarrow$ location.

* here we have to make cases as prev.

Properties of F.D.

1) Reflexive: if Y is subset of X, then $X \rightarrow Y$, ex $S_{id} \rightarrow S_{id}$
→ always valid

2) Augmentation
if $X \rightarrow Y$, then $XZ \rightarrow YZ$
ex $S_{id} \rightarrow S_{name}$
S_{id} phone no $\rightarrow S_{name}$ Phone no.

3) Transitive
if $X \rightarrow Y$ & $Y \rightarrow Z$, then $X \rightarrow Z$
ex $S_{id} \rightarrow S_{name}$ & $S_{nam.} \rightarrow$ City
$S_{id} \rightarrow$ City

4) Union
if $X \rightarrow Y$ & $X \rightarrow Z$ then $X \rightarrow YZ$

5) Decomposition if $X \rightarrow YZ$ then $X \rightarrow Y$ & $X \rightarrow Z$
but $XY \rightarrow Z \Rightarrow X \rightarrow Z, Y \rightarrow Z$ X (wrong)

Pseudo Transitivity : if $X \to Y$ & $WY \to Z$, then $WX \to Z$

Composition : if $X \to Y$ & $Z \to W$, then $XZ \to YW$

2nd NF, Second Normal form

→ Table or relation must be in 1st Normal form

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

→ If partial dependency, then it is not in 2nd NF

Customer ID	Store ID	Location
1	1	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai

Candidate key = Customer ID + Store ID

Prime attributes : Customer ID
& Store ID

Non prime " : Location.

Not in 2nd NF. because only store ID determine location only. So location is not depend on customer ID & C.K = Customer ID + Store ID

Both are in 2nd NF

Store ID	Location
1	Delhi
2	Bangalore
3	Mumbai

CK = Store ID

location depends on Store ID ✓

+

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

CK = Customer ID + Store ID

Ex R(ABCDEF)

FD = {C → F, E → A, EC → D, A → B}

CK : EC = FADB

EC⁺ = ECFADB

1st step Find CK.

CK = {EC}

2nd step Prime attributes {E, C}

3rd step Non prime attributes {A, B, D, F}

Now proper subset of CK = C, E

as C determines A, & hence it is partial dependency bcoz a part of CK determines Non-prime attribute.

* LHS should be proper subset of any CK & RHS should be a Non Prime Attribute for atleast one can then there is partial dependency & hence not in 2nd NF.

⇒ {C → F, E → A, EC → D, A → B}

PD PD Not PD Not PD

hence not in 2nd NF

PD → partial dependency.

Proper subset of AB = A, B

Third Normal form

→ Table or relation must be in second normal form

&

→ There should be no transitive dependency in table

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

FD: Roll no → State
State → City

PA = {Roll no}
NPA = {State, City}

CK = {Roll no.}

Imp

* Non prime attribute cannot determine any non-prime attribute
o.w. not in 3rd N.F.

Roll no → State & State → city , Roll → city (transitive)

hence the above is not in 3rd normal form

ex R(A, B, C, D)

F.D: AB → CD, C → A

CK: $AB^+ = ABCD$

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

Not in 3rd N.F.

?

B → is candidate key ?

Ex

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

CK : {AB, DB}, $AB^+ = ABCD$

PA = A, B, D

NPA = {C}

FD : $AB \rightarrow CD$, $D \rightarrow A$

hence in 3rd N.F.

Imp Trick

For each FD

LHS must be a CK or Super key OR RHS is a prime attribute

then only it is valid FD & in 3rd N.F, o.w. not.

Special Case of 3 NF

BCNF (Boyce codd Normal form)

→ Table should be in 3 NF

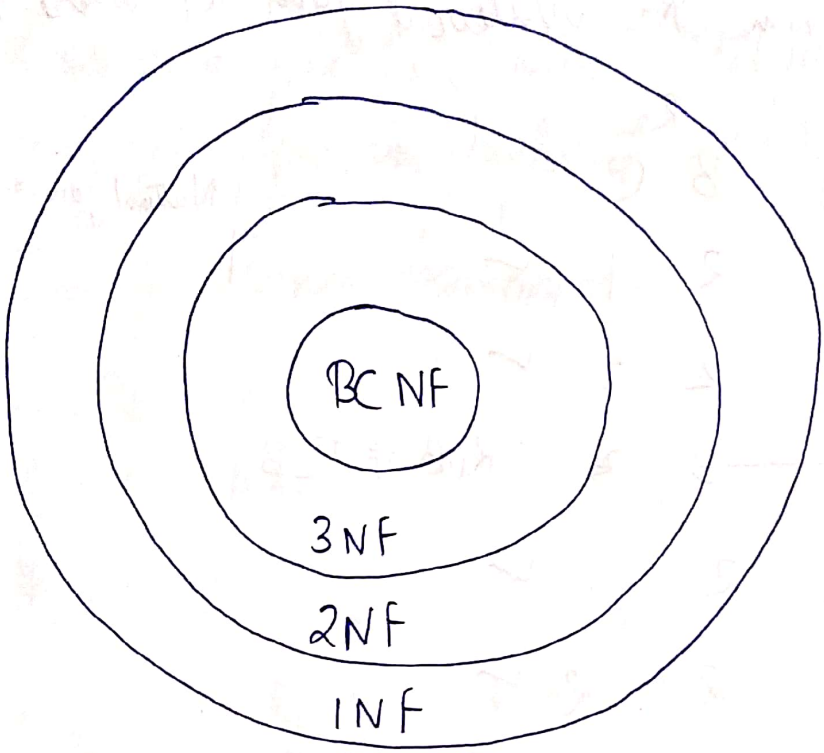
Rollno.	Name	Voterid	age
1	Ravi	K043	20
2	Varun	M034	21
3	Ravi	K786	23
4	Rahul	D286	21

CK : { Roll no, Voter id }

FD : { Rollno → name, Rollno → Voterid, Voterid → age, Voter id → Rollno. }

*LHS of each FD should be CK candidate key or super key then it is BCNF

So far



if a table is BCNF then it is 3NF, 2NF, 1NF.

Lossless / Lossy Join Decomposition (5th Normal form)

During normalization - decomposition should be lossless.

R

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

$R_1(A,B)$

$R_2(B,C)$

R_1

A	B
1	2
2	2
3	3

R_2

B	C
2	1
2	2
3	2

Atleast one attribute should be common to join the tables.
(here B is common)

Find the value of C if the value of A = 1

→ so we need to join R_1 & R_2 to give this ans.

Select $R_2.C$ from R_2 Natural join R_1 where $R_1.A = 1$

R_1		R_2		
A	B	B	C	
1	2	2	1	✓
1	2	2	2	✓
~~1~~	~~2~~	~~3~~	~~2~~	$R_1.B \neq R_2.B$
2	2	2	1	✓
2	2	2	2	✓
~~2~~	~~2~~	~~3~~	~~2~~	$R_1.B \neq R_2.B$
~~3~~	~~3~~	~~2~~	~~1~~	
~~3~~	~~3~~	~~2~~	~~2~~	
3	3	3	2	✓

Natural join = Cross product + some condition

R'

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

extra
⇒ inconsistency

sporious tuples

from R
ans = 2 1

from R'
ans = 2 1
2 2

So lossy decomposition.

So there is criteria so that we could not get error.

** Common attributes should be CK or SK of either R_1 or R_2 or both.

So we should decompose here with $R_1(AB)$ & $R_2(AC)$ then we will not get spurious tuples & lossless joins

⇒ Condition for lossless decomposition

1) $R_1 \cup R_2 = R$

$AB \cup AC \equiv ABC$

2) $R_1 \cap R_2 \neq \phi$

3) R_1 CK or R_2 CK or both

This condition is also for 5th Normal form.

Imp Ex $R(ABCDEF)$, check the hieghest Normal form.

FD : { $AB \rightarrow C$, $C \rightarrow DE$, $E \rightarrow F$, $F \rightarrow A$ }

step1: Find all CK's in Relation.

$AB^+ = ABCDEF$

$A^+ = A \rightarrow$ Not CK

$B^+ = B \rightarrow$ Not CK

$\Rightarrow AB =$ CK.

CK = { AB, FB, EB, CB }

step 2: Write all prime attributes

{ A, B, C, E, F }

step 3: Write all not prime attributes

{ D }

{ AB → C, C → DE, E → F, F → A }

1st check for BCNF

Lhs of all FD's should be CK / Super key.

Not BCNF @ C, F, F are not CK.

{ AB → C, C → DE, E → F, F → A }

✗ BCNF	✓	X	X	X
✗ 3rd NF	✓	X	✓	✓
✗ 2nd NF	✓	X	✓	✓
1st NF	✓	✓	✓	✓

Ex — R(ABCDEF)

CK = { AB, FB, EB, CB }

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

Non Prime attributes { D }

FD's { AB → C, C → D, C → E, E → F, F → A }

Assuming in ✗ 1st N.F, make it upto BCNF

For 2nd NF

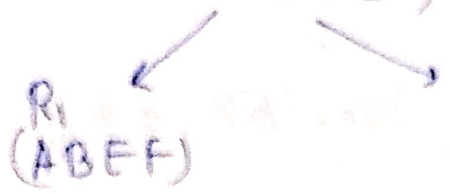
FD's { AB → C, C → D, C → E, E → F, F → A }

FD PD FD FD FD

hence not in 2nd NF.

To make 2nd NF, we have to decompose.

R(A1 R(A B C D E F)



$C^+ = C\,D$

but its join R1 & R2 there must be some common, here C is added in R_1 bcz $C = CK$ of R_2

R_1 (A B C E F) R_2 (C D)

$R_1\{AB \to C, C \to E, E \to F, F \to A\}$ $R_2\{C \to D\}$

Now, for checking 3NF.

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

3rd NF { LHS must be a CK or RHS Prime attribute }

For 3rd NF

$R_1\{AB \to C, C \to E, E \to F, F \to A\}$ ✓ ✓ ✓ ✓ $R_2\{C \to D\}$ ✓

hence in 3rd NF.

For BCNF

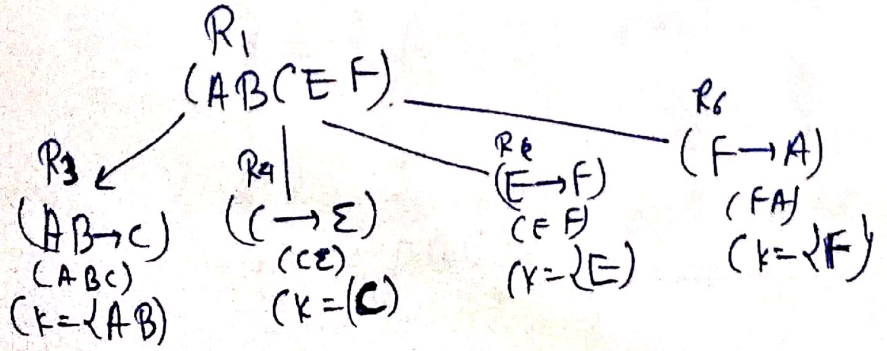
LHS must be a CK.

$R_1\{AB \to C, C \to E, E \to F, F \to A\}$ ✓ X X X

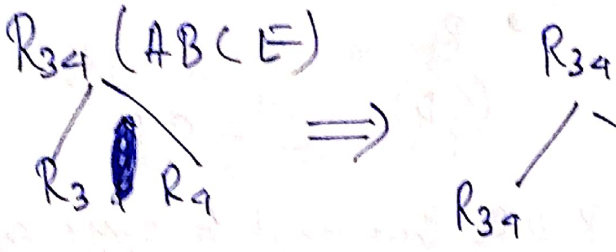
R_1 not in BCNF

$R_2\{C \to D\}$ ✓ in BCNF

⇒ 0% Redundancy



& In BCNF there is always lossless decomposition.



ex

Schema 1 : Registration (rollno, Courses)

Non-trivial FD { rollno → Courses }

BCNF ✗

hence 3 NF ✗
2 NF ✗
1 NF ✓

Schemas 2 : Registration (rollno, Courseid, email)

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

CK = { rollno Courseid }

PA = { rollno, Courseid }

NPA = { email }

BCNF ✗

3 NF ✓

Schema 3 : Registration (rollno, Courseid, marks, grade)

Non-trivial FD : { rollno, Courseid → marks, grade
marks → grade }

Prime Att : { rollno, Course id }

NPA : { marks, grade }

2nd NF ✗

Schema 4 Registration (rollno, Courseid, credit)

Non-trivial FD { rollno, Courseid → credit
Courseid → credit }

PA : { rollno, Courseid }

NPA : { credit }

BCNF ✗

3 NF ✗

2 NF ✗

1 NF ✓

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 $X \geq Y$

if Y covers X $X \leq Y$

then $\boxed{X \equiv Y}$

Now if X covers Y ✓

$A^+ = ABC$ $B^+ = BC$, $A^+ = ABC$ | Closure of LHS of Y in X

Now if Y covers X ✓

$A^+ = ABC$, $B^+ = BC$ | Closure of LHS of X in Y

$\Rightarrow X \leq Y$ & $Y \geq X$

$\Rightarrow \boxed{X \equiv Y}$

ex

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

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

Now, X covers Y ✓ $X \geq Y$

$AB^+ = ABCD$, $AB^+ = ABCD$, $C \to D$

Y Covers X, $Y \subseteq X$, $Y \not\supseteq X$

$AB^+ = ABCD$, $B^+ \rightarrow B$, $C \rightarrow D$

find
$B \rightarrow C$ in X
to it not covers

$X \not\equiv Y$

$X \geq Y$, $Y \not\geq X$

Joins

- Cross Join (Cross product) — Possible if there is common attribute
- * Natural Join
- Conditional
- Equi Join
- * Self Join
- Outer join — left, Right, ~~either~~ full

Select Address from employee where cname = 'Varun'

o/p: Chd

fk

E_No	E_name	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amrit	Delhi
5	Nitin	Wordq

'Employee'

Dep No	Name	Eno
D_1	HR	1
D_2	IT	2
D_3	MRKT	4
D_4	Finance	5

'Department')

Find E_name of emp whose working in HR dept.

* If more than one ques table is used, then joins is required.
* There must be atleast one common attribute to use join.

Join = Cross product + Select Statement (Condition)

'Natural Join'

"find the emp names who is working in a department"

emp

E_no	E_name	Dept No	E_no
1	Ram	D_1	1
1	Ram	D_2	2
1	Ram	D_3	3
2	Varun	D_1	1
2	Varun	D_2	2
2	Varun	D_3	3
3	Ravi	D_1	1
3	Ravi	D_2	2
3	Ravi	D_3	3
4	Amrit	D_1	1
4	Amrit	D_2	2
4	Amrit	D_3	3

Dept.

E_No	E_name	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amrit	Delhi

Emp

Dept No.	Name	E_no
D_1	HR	1
D_2	IT	2
D_3	MKTG	3

Dept

Select E_name from Emp where Address = 'Delhi';

* If we have to set common attribute is same then we use natural join.

Select E_name from (Emp, Dept) (cross product.) where emp.e_no = Dept.e_no;

(query)

Emp

Eno	Ename
1	Ram
2	Varun
4	Amrit

Dept

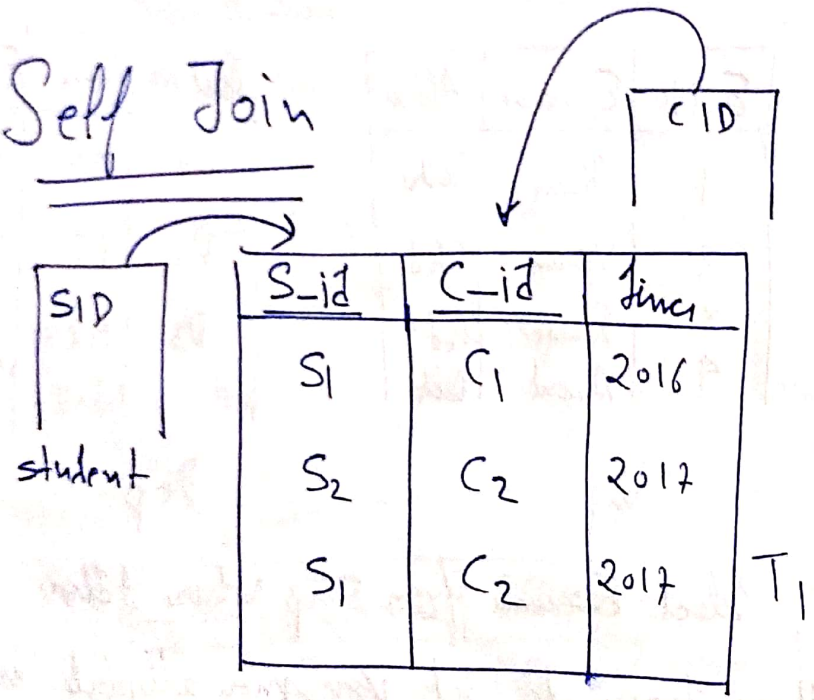
Dep_No	Eno
D_1	1
D_2	2
D_3	4

o/p: Ram
Varun
Amrit.

Actual way: Select E-name from emp Natural join Dept

(in two table but!! column name should be same.

Self Join



PK = Sid, C_id.

Find student id who is enrolled in at least two courses.

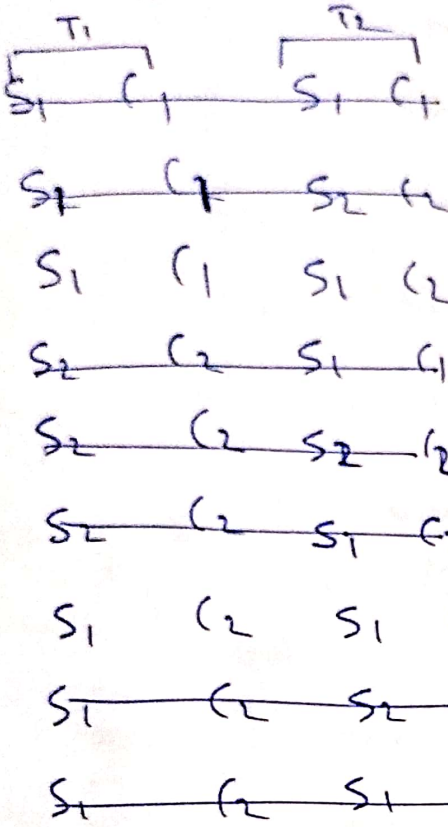
o/p: S_1

Select T_1.Sid from Study as T_1, Study as T_2 where T_1.S_id = T_2.Sid and T_1.C_id <> T_2.C_id

Sid	Cid	Time
S_1	C_1	2pm
S_2	C_2	2pm
S_1	C_2	2pm

(query)

T_2



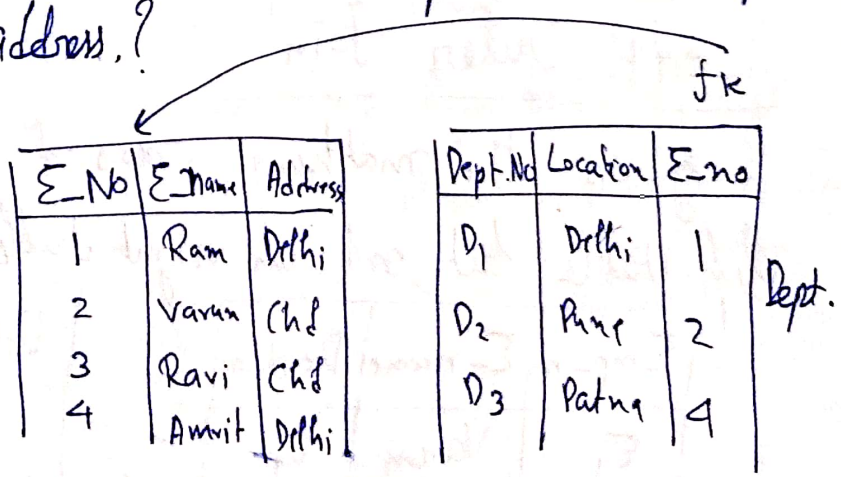
$\Rightarrow$

T_1	T_2
S_1 C_1	S_1 C_2
S_1 C_2	S_1 C_1

(Why not two times S_1 will be printed ?)

'Equi Join'

Find the emp name who worked in a department having location same as their address.?



Select E_name from Emp, Dept. where
Emp.E_no = Dept.E_no and
Emp. Address = Dept. location;

	Ram	Delhi	D1	Delhi	1
1	Ram	Delhi	D2	Pune	2
1	Ram	Delhi	D3	Patna	4
2	Varun	Chd	D1	Delhi	1
2	Varun	Chd	D2	Pune	2
2	Varun	Chd	D3	Patna	4
3	Ravi	Chd	D1	Delhi	1
3	Ravi	Chd	D2	Pune	2
3	Ravi	Chd	D3	Patna	4
4	Amrit	Delhi	D1	Delhi	1
4	Amrit	Delhi	D2	Pune	2
4	Amrit	Delhi	D2	Pune	4

o/p : Ram

Left Outer Join

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

Emp

Emp_no	E_name	Dept_no
E1	Varun	D1
E2	Amrit	D2
E3	Ravi	D1
E4	Nitin	—

dept.

Dept_No.	D_name	Loc
D1	IT	Delhi
D2	HR	hyd
D3	Finance	Pune.

(qualify)

- Select emp-no, E-name, D-name, LOC from emp left outer join dept On(emp.dept_no = dep.dept_no)

o/p

Emp-no	E-name	D-name	LOC
E_1	Varun	IT	Delhi
E_2	Amrit	HR	Hyd.
E_3	Ravi	IT	Delhi
E_4	Nitin	—	—

Right Outer Join

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

emp

Emp-no	E-name	Dept-no
E_1	Varun	D_1
E_2	Amrit	D_2
E_3	Ravi	D_3

Dept-No	D-name	LOC
D_1	IT	Delhi
D_2	HR	Hyd.
D_3	Finance	Pune
D_4	Testing	Noida

Select emp-no, E-name, D-name, LOC from emp Right outer join Dept On(emp.dept-no = dept.dept-no)

natural join

o/p

Empno	E-name	D-name	LOC
E_1	Varun	IT	Delhi
E_2	Amrit	HR	Hyd
E_3	Ravi	Finance	Pune
—	—	Testing	Noida

Left outer join ∪ Right outer join

= Full outer join

Relational Algebra (1970)

→ Procedural Query language.

OR

Formal Query language

"Operators"

Basic operator

- → Projection (π)
- → Selection (σ)
- → Cross Product (×)
- → Union (∪)
- → Rename (ρ)
- → Set Difference (−)

Derived Operator

- → Join (⋈)
- → Intersect (∩)

$X \cap Y = X - (X - Y)$

- → Division (/, ÷)

Projection (Π)

RDBMS

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

$\Pi_{Rollno}(Student)$

o/p: 1
2
3

Query : Retrieve the rollno from table (Student)

ex $\Pi_{Rollno, Name}(Student)$

o/p

Rollno	Name
1	A
2	B
3	A

ex $\Pi_{name}(Student)$

o/p. A
B

Imp

* Projection removes redundant data

Selection (σ)

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

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

o/p B

- $\sigma_{Rollno='2'}(Student)$

o/p: 2, B, 21

but to get name only

$\Pi_{name}\left(\sigma_{Rollno='2'}(Student)\right)$ o/p: B

but if we did

$\sigma_{Rollno=2}\left(\Pi_{name}(Student)\right)$ // wrong.

A, B.

Projection works on column

Selection works on rows

Cross Product

R_1

A	B	C
1	2	3
2	1	4

R_2

C	D	E
3	4	5
2	1	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

$R_1 \times R_2$

Columns $3 + 3 = 6$

$m + n$

Tuples $2 \times 2 = 4$

$n \times y$

To Join

if Atleast one column should same.

Set Difference (−)

$(A - B)$ = A but not B

$= A \wedge B'$

* Not Commutative, $A - B \neq B - A$

Roll no.	Name
1	A
2	B
3	C

(Student)

Emp. No	Name
7	E
1	A

(Employee)

Rollno	Name
2	B
3	C

(Student − Employee)

condition

* No. of column must be same in no.
* Domain of every column must be same

Name of person who is a student but not emp.

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

Name
B
C

← // name of o/p column is same as 1st table

Union

Roll no.	Name
1	A
2	B
3	C

(Student)

Emp. No	Name
7	E
1	A

(Employee)

Rollno.	Name
1	A
2	B
3	C
7	E

Student ∪ Employee

Condition

1) No. of column is same in no.
2) Domain of every column of both the table must be same.

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

Name
A
B
C
E

Name of either employee & student or both.

Q.

RollNo	Name
⋮	⋮

$\cup$

Name	EmpNo
⋮	⋮

$=$ NULL, bcoz order is different

Division Operation in Relation algebra

→ derived operator

$A(X,Y) / B(Y)$ = it results X values for that there should be tuple $<X,Y>$ for every Y value of relation B.

ex. $E(Sid, Cid) / C(Cid)$

~~for~~ for all, every → go for division operator.

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

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'

$E(Sid, Cid) / C(Cid) = S_1$

$(\Pi_{Sid}(Enrolled)) \times \Pi_{Cid}(Course) - Enrolled \Rightarrow$

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

$-$

S_1	C_1
S_2	C_2
S_1	C_2
S_3	C_2

$S_1 \times C_1$

$S_2 \quad C_2$

S_3

$$= \begin{vmatrix} S_2 & C_2 \\ S_3 & C_1 \end{vmatrix}$$ o/p:

$$\Pi_{Sid}\left(\left(\left(\Pi_{Sid}(Enrolled)\right) \times \Pi_{Cid}(Course)\right) - Enrolled\right)$$

o/p: S_2
S_3

$$\Pi_{Sid}(Enrolled) - \left(\Pi_{Sid}\left(\left(\left(\Pi_{Sid}(Enrolled)\right) \times \Pi_{Cid}(Course)\right) - (Enrolled\right)\right)$$

o/p: $\begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix} - \begin{pmatrix} S_2 \\ S_3 \end{pmatrix} = S_1$

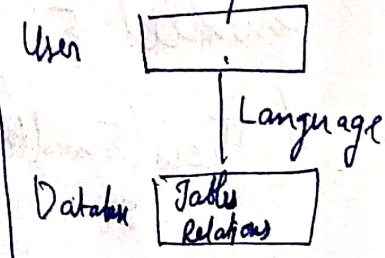
* From the table, which is used to get data is at numerator & that table which is helping in getting data is placed at denomenator.

Structured Query Language (SQL)

→ SQL is domain - specific language
→ SQL is declarative language
→ DDL, DML, DCL, TCL
→ Keys and constraints
→ Operations (Like, between, In, Not in, Conditional) exist, not exist.

* Language
- Declarative: what to do
- Procedural: what to do, how to do.

- User fetch data from database using SQL



→ clauses (distinct, order by, group by, from, having)

→ aggregate functions

→ Joins and Nested queries

→ PL SQL (Trigger, cursor, procedures) // Procedural language.
function

Father of data base : EF Codd.

1st implemented using SQL by ibm.

SE QUEL → Structured english query language

ibm → dB2
Microsoft
oracle
Microsoft → SEQUEL server

SQL Commands

Data Definition language

Structure: Table

- Create
- Alter
- Drop
- Truncate
- Rename

Data Manipulation language

- Select
- Insert
- Update
- Delete.

Data Control Language

- Grant
- Revoke.

Transaction Control language

- Commit
- Rollback.
- Save point.

Constraints

- Primary key
- Foreign key
- check
- unique
- Default
- Not Null.

CREATE

Syntax: CREATE TABLE <table name> (<col1> data type (width),
<col2> data type (width), ---);

Ex: CREATE TABLE Customer (Name varchar(30),
cust_id Number,
Address varchar2(50));

DATA Types:

Numeric	Character	Date	BOOLean
NUMBER	CHAR	DATE	BOOLEAN
FLOAT	VARCHAR2	TIME	
INT	NCHAR	TIMESTAMP	
REAL	NVARCHAR2	INTERVAL	
DECIMAL	LONG		
BINARY FLOAT etc	RAW		
	LONG RAW		

After Command

→ Add Column/s
→ Remove Column/s
→ Modify data type
→ Modify data type length
→ Add Constraints
→ Remove constraints
→ Rename column/Table

int

ID	Name

varchar

(query)

Alter table Student Add (address varchar(30));

ID	name	addr

eg Create table employee
(
id int,
name varchar(10)
);

Alter table employee add address varchar(10);
alter table employee drop column address;
alter table employee modify id varchar(10); // change data type
alter table employee modify id varchar(newlength) 10,20...
alter table employee rename column id to roll-no; // rename column
alter table employee rename to employee_new // rename table.
alter table employee add primary key (roll-no)

eg Write a SQL to display maximum salary from table.

Write a SQL Query to display employee name who is taking maximum salary.

Emp

E-id	E_name	Dept	Salary
1	Ram	HR	10000
2	Amrit	MRKT	20000
3	Ravi	HR	30000
4	Nitin	MRKT	40000
5	Varun	IT	50000

select max(Salary) from Emp;
o/p: 50000

select E_name from emp where Salary = (select Max(Salary) from emp);
o/p: varun

Execution

10000 = 50000 false
20000 = 50000 false
30000 = 50000 false
40000 = 50000 false
50000 = 50000 true.

→ Write SQL Query to display second highest salary from emp table?

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

o/p = 40000

→ Write SQL to display emp_name of second highest salary from emp table.

select E_name from emp where salary =
select max(salary) from emp where salary <> (select max(salary) from emp);

o/p: Nitin

2=2 | True.
2 In (1,2,3,4) | True

Ex: Write a query to display all the dept names along with no. of emps working in that

o/p: HR 2
MRKT 2
IT 1

Select Dept from emp group by Dept;

o/p: HR
HR
MRKT
MRKT
IT

or dept

Select Dept, count(*) from emp group by Dept;

o/p: HR 2
MRKT 2
IT 1

ex Write a query to display all the dept names where no of emps are less than 2

Select dept from emp group by dept having count(*)<2

o/p: IT

Select e-name from emp where dept In Select dept from emp group by dept having count(*) < 2

| Where we cannot use with grp, bcoz where is for the whole table

o/p: Varun.

ex Write the query to display highest salary department with & name of emp who is taking the highest salary.

select max(salary) from emp group by dept;

o/p: HR 3000
MRKT 40000
IT 50000

select dept, max(salary) from emp group by dept

HR 3000
MRKT 40000
IT 50000

select c_name from emp where salary In (select Max(Salary) from emp group by dept);

o/p: Ravi
Nitin
Varun.

In / Not In

Emp

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

Project

fk	pk		
Eid	Pid	Pname	Location
1	P1	IOT	Banglore
5	P2	BIG DATA	Delhi
3	P3	Retail	Mumbai
4	P4	Android	Hyderabad

Query: Detail of emp whose address is either Delhi or chd or Pune

* For more than one comparison go with In / Not In

select * from emp where Address In ('Delhi', 'chd', 'Pune');

o/p: 1 Ravi chd
2 Varun Delhi
3 Nitin Pune.
5 Ummy chd

select * from emp where Address Not In ('Delhi', 'chd', 'Pune');

o/p: 4 Robin Bangalore

Now, Use of In & Not In in sub query

Find the name of the employee who are working on a project

select ename from emp where Eid In (select Distinct (Eid) from Project);

inner query

* Distinct is used for removal of data redundancy

o/p:

Ravi	Ravi
Ammy	Nitin
Nitin	Robin
Robin	Ammy

Select ename from emp where Eid Not In (Select distinct (Eid) from Project);

o/p: Varun

Exists / Not Exists → returns true/false.

Used in correlated query.

query: Find the detail of the employee who is working on at least one project?

Select * from emp where Eid exists (Select Eid from project where emp.Eid = project.Eid);

o/p:

1	Ravi	chd
~~3~~	~~Ratul~~	.
3	Nitin	Pune
4	Robin	Bangalore
5	Ammy	uchd.

Select * from emp where Eid Not exists (Select Eid from project where emp.Eid = project.Eid);

o/p: 2 Varun Delhi

"Aggregate function"

Max, Min, Count, Avg, Sum.

emp.

E_id	E_name	Dept	Salary
1	Ram	HR	10000
2	Amrit	MRKT	20000
3	Ravi	HR	30000
4	Nitin	MRKT	30000
5	Varun	IT	50000
6	Sanay	Testing	NULL

select Max(Salary) from emp;

o/p: 50,000

select Min(Salary) from emp;

o/p: 10,000

select Count(*) from emp;

→ give total tuples, o/p: 6

select Count(Salary) from emp;

→ o/p: 5.

select Distinct(Count(Salary)) from emp;

o/p: 4

select Sum(Salary) from emp;

o/p: 140,000

select Distinct(Sum(Salary)) from emp;

o/p: 11000

select Avg(Salary) from emp;

$$\text{Avg(Salary)} = \frac{\text{Sum(Salary)}}{\text{Count(Salary)}} = \frac{140,000}{5}$$

Select Distinct (Avg(Salary)) from emp;

$$\text{Distinct}(\text{Avg}(\text{Salary})) = \frac{\text{Distinct}(\text{Sum}(\text{Salary}))}{\text{Distinct}(\text{Count}(\text{Salary}))} = \frac{11000}{4}$$

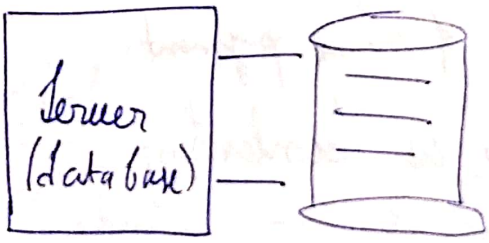
Transaction

It is a set of operations used to perform a logical unit of work.

A ~~transaction~~ transaction generally represent change in database.

Two Operations

1) Read – Data base access
2) Write – Change
3) Commit



Transfer

R(A) → 1000
A = A − 500
W(A) → 500
R(B) → 2000
B = B + 500
W(B) → 2500

all at RAM

R() → Read
W → Write.

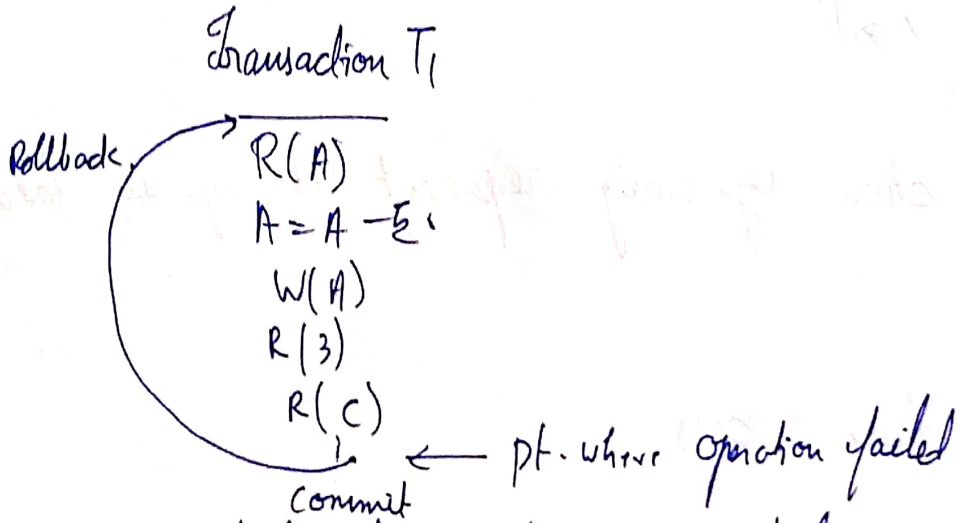
Commit → saves all the ~~data~~ operations. value in HD

ACID Properties

Atomicity Consistency Isolation Durability
A C I D

Atomicity

→ either all or ~~NULL~~ None.



means a failed transaction cannot be resumed, it will restart

Ex transaction of online payment.

→ This is not same as downloading

Consistency

Before the transaction start & after the transaction completed, the sum of money should be same.

for ex $A \xrightarrow{1000} B$, A = 2000
B = 3000

T_1

R(A) → 2000
A = A - 1000
W(A) 1000
R(B) → 3000
B = B + 1000
W(B) 4000

T_1

R(A)
A = A − 1000
W(A)
R(B) ⇒ 3000
B = B + 3000
W(B) ⇒ 4000
Commit.

A = ~~2000~~ 1000
B = ~~3000~~ 4000

Before T_1 sum = 2000 + 3000 = 5000
After commit, sum = 4000 + 1000 = 5000
hence consistent.

Isolation

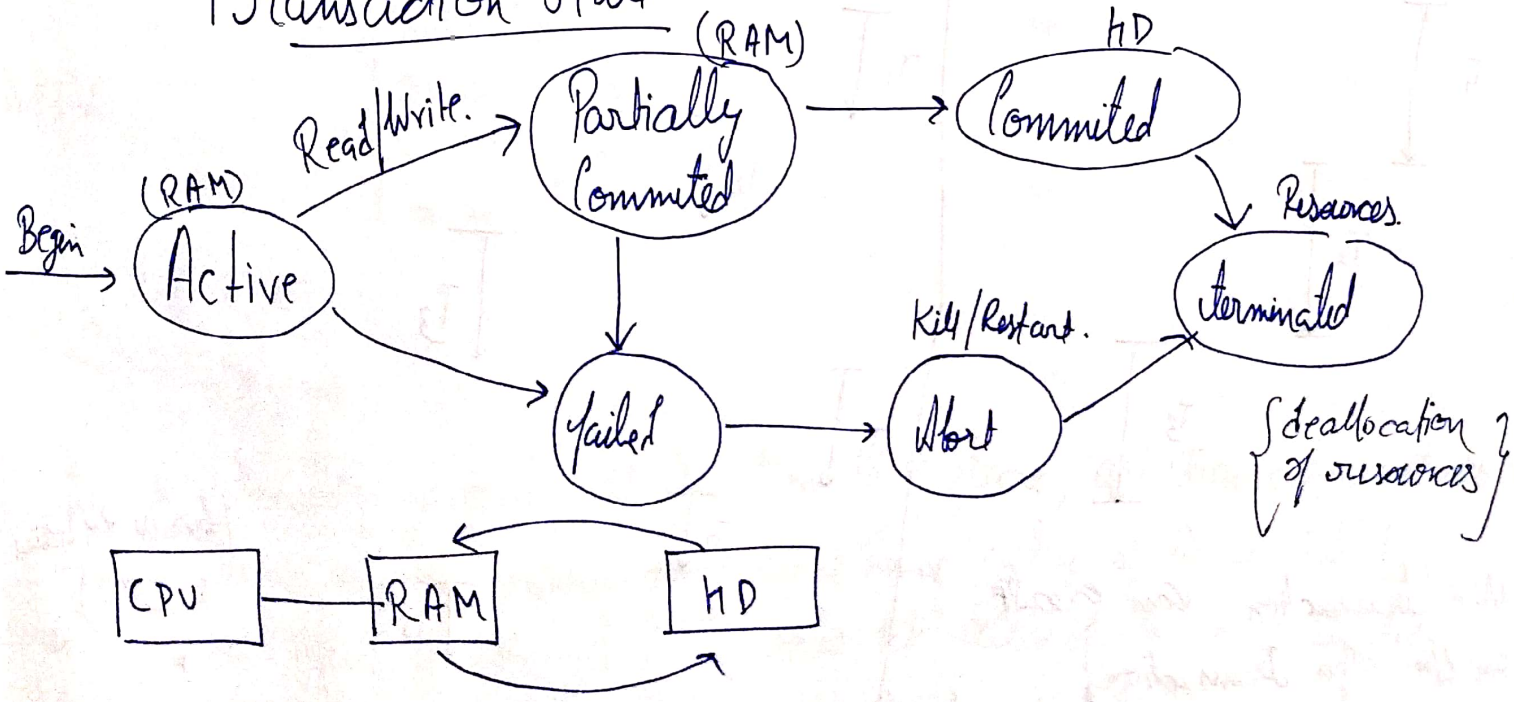
Convert parallel schedule to serial schedule because serial schedule is always consistent.

Durability

All the changes after commit is permanent.
(after transaction)

Therefore after transaction is done, save all changes at H.D.

Transaction State



active → Now data is in RAM from HD.

Partially → Read/Write operations is done at RAM committed

Committed → Changes are save to harddisk.

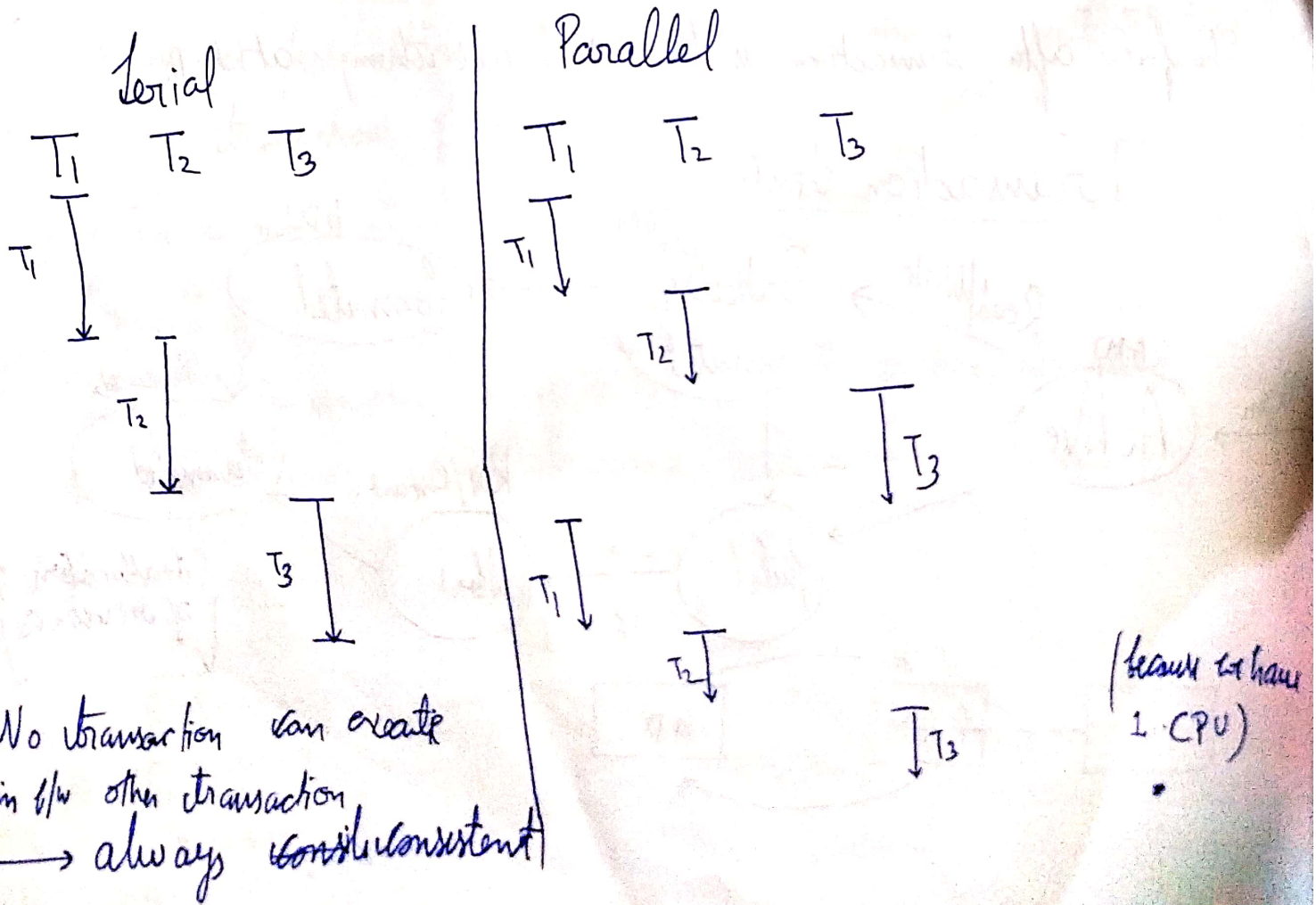
Terminated → Ressorce are going back to O.S.

Abort → Kill & Restart

Schedule

It is chronological execution sequence of multiple transaction

T_1 T_2 T_3 --- T_n



No transaction can execute in b/w other transaction

→ always consistent

(because we have 1 CPU)

Serial transaction will have more waiting time

Ex ATM transaction

→ secure.

→ Performance degradation

→ less waiting time

→ higher performance

Performance : Throughput → No. of transactions executed per unit time.

Recoverable Schedule

Recoverable → Ability to get recover

Ex A=10 B=20

T_1	T_2
10 ← R(A)	
5 ← A=A-5	
5 ← W(A)	
⋮	R(A) // 5
Roll back	A=A-2 // 3
	W(A) // 3
	Commit
20 // R(B)	
* fail	

due to atomicity property, at the time of failure, all transaction starts again from very beginning point & hence value of A at data base is 10 (initial value), so changes of T_2, that set value of A as 3 in data base is lost, this is known as irrecoverable schedule.

Serializable schedule Problems

Read – Write Conflict or unrepeatable read

Same data

R(A)	R(A)] No problem.
R(A)	W(A)
W(A)	R(A)] → Problems.
W(A)	W(A)

1st Scenario

U_1	U_2
T_1	T_2
2 || R(A)	
⋮	R(A) || 2
⋮	A=A-2
⋮	W(A) || 0
⋮	Commit
!!! 0 || R(A)	
W(A)	
Commit	

A = ~~2~~ 0

So T_1 get abort
Bcoz initially it reads 2
, now Read 0 without any
operation of T_1

2nd Scenario

T_1	T_2
10 R(A)	
9 A=A-1	
⋮	R(A) || 10
⋮	A = A-1
⋮	W(A) || 9
9 || W(A)	Commit
Commit	

A = ~~10~~ ~~9~~ 9

A value should be 9, but actual value is 9 only in database

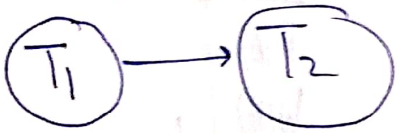
Serializability — Conflict / View

Ability to become serializable

[We have to find the serial schedule, which is clone of given parallel schedule

S

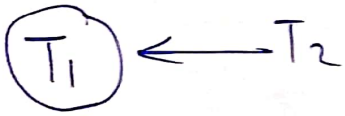
T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)



Both are serial schedule. , so both need not to be to be serialize.

S

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

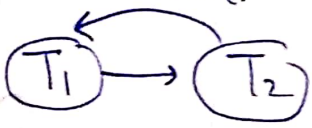


Now

S_1

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

→ Parallel schedule.



Now we have to check whether this can be converted into serial schedule.

$T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$

S

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

$T_1 \rightarrow T_2 \rightarrow T_3$
$T_1 \rightarrow T_3 \rightarrow T_2$
$T_2 \rightarrow T_3 \rightarrow T_1$
$T_2 \rightarrow T_1 \rightarrow T_3$
$T_3 \rightarrow T_1 \rightarrow T_2$
$T_3 \rightarrow T_2 \rightarrow T_1$

$3! = 6$ possibility but out of which only One is correct

Conflict Equivalent

R(A) R(A) } Non-Conflict Pair

R(A) W(A)
W(A) R(A)
W(A) W(A)
} Conflict Pair

R(B) R(A)
W(B) R(A)
R(B) W(A)
W(A) W(B)
} Non-Conflict pair

S

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

S'

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

S

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

→

S

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

→

S

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

$\equiv S'$

* Adjacent non-conflict pair position can change.

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

No change in position.

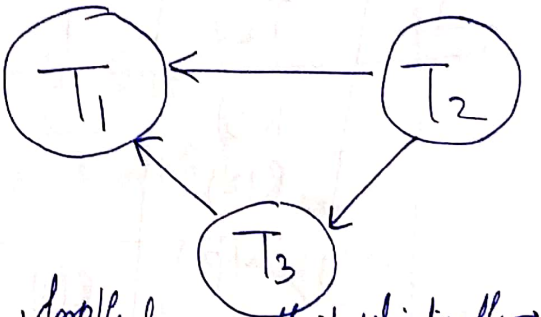
* $S \xrightarrow{CE} S'$, then S' is serializable. , CE → Conflict equivalent.

Conflict Serializability

* check conflict pairs in the other transactions & draw edge.

T_1	T_2	T_3
① R(x)		
		② R(y)
		③ R(x)
	④ R(y)	
	⑤ R(z)	
		⑥ W(y)
	⑦ W(z)	
R(z)		
W(x)		
W(z)		

Precedence Graph



No loop/cycle → conflict serializable → consistent

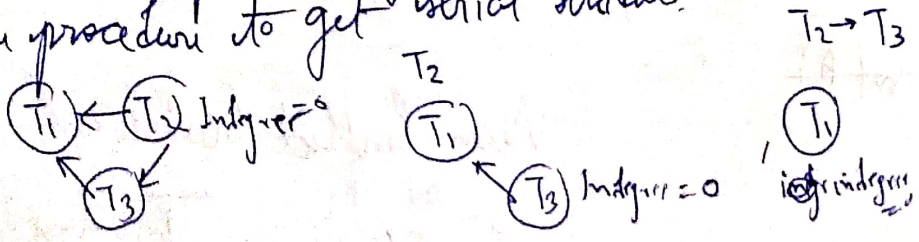
Conflict of W(y) → R(y), W(y)

* If there is no loop/cycle then the schedule is conflict serializable.

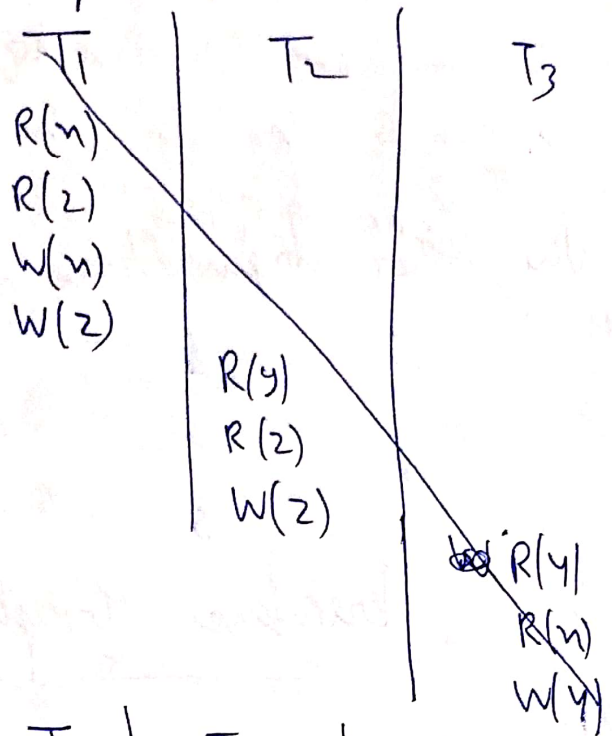
& If conflict serializable, then it can be converted into serial schedule & hence consistent schedule.

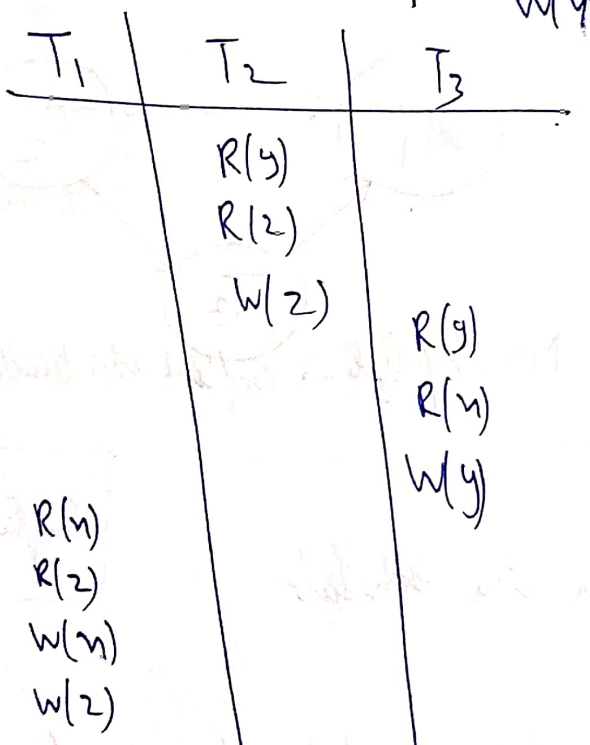
Now if schedule is conflict serializable, then check for the no node with indegree = 0, indegree = no. of input edges. Remove that node from the precedence graph & set it with ① 1st priority, then ② apply the same procedure to get serial schedule.

Ex



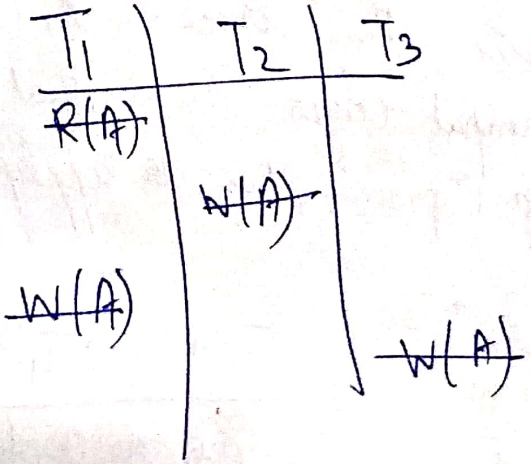
So final serial order is

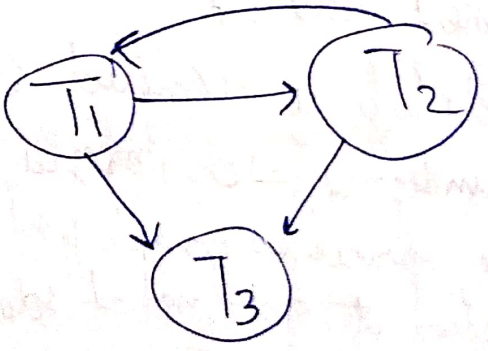




ex Check whether schedule is conflict serializable or not?

S





Non conflict serializable

or

C.S

I don't know!!

* No loop/cycle → Conflict Serializable → Serial → Consistent

but if loop/cycle → don't know → view serializable

So we use view serialization.

Lets A = 100

S

T_1	T_2	T_3
R(A)	*A = A-40	
	W(A) → 10	
*A = A-40		
W(A) → 20		*A = A-20
60-40=20		W(A) → 0

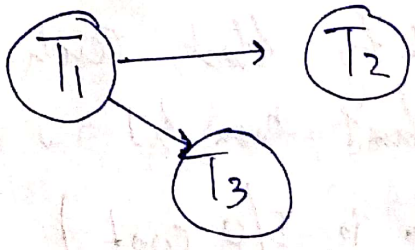
S'

A = 100

T_1	T_2	T_3
100 R(A)		
A=A-40 → 60 // W(A)		
A=A-40	→ W(A) 20	
A=A-20		→ W(A) 0

- As the final value of A in each table same hence $S \equiv S'$

& S' is serializable, Now graph of S'



$\Rightarrow T_1 \rightarrow T_2 \rightarrow T_3$

Why Indexing is used ?

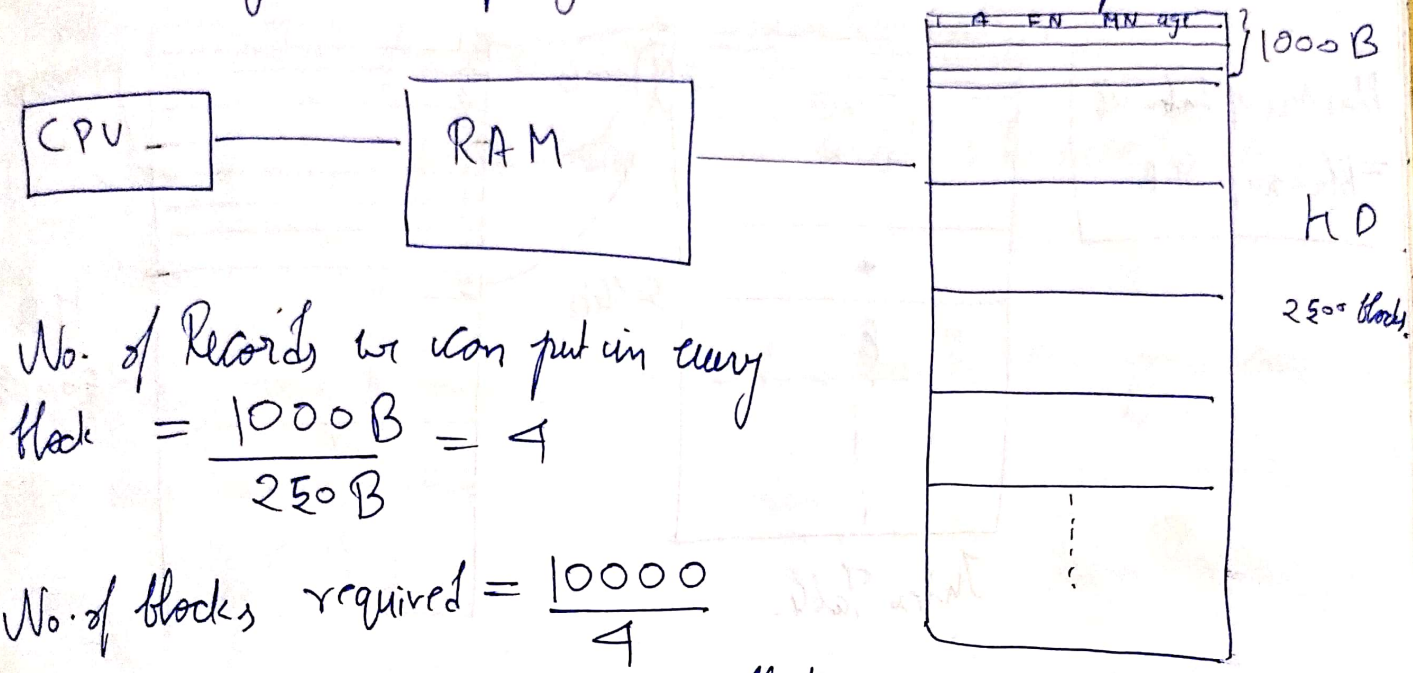
* CPU works with faster memory (RAM) because CPU execute million instructions per sec.

* If each block stored 100 Records, then two is store 10000 records, 100 blocks are required.

* RAM searches every data
CPU searches data block by block, each time a block is fetched into RAM, if data found, then hit, o.w miss & next block is searched [Block transfer is I/O cost]

** I/O cost is Reduced by indexing

Q. Consider a Harddisk in which block size = 1000 bytes, each record is of size = 250 Bytes. If total no. of records are 10000 & the data entered in Harddisk without any order/Unordered/ What is avg time complexity to search a record from HD?



No. of Records we can put in every block $= \frac{1000B}{250B} = 4$

No. of blocks required $= \frac{10000}{4}$

$= 2500$ blocks

I/O Cost → Cost in calling each block in RAM.

Best case = 1

Worst case = 2500 = N

Avg. I/O cost $= \frac{2500}{2} = 1250 = N/2$

$O(N)$

but if data is sorted ordered, then search by binary search,

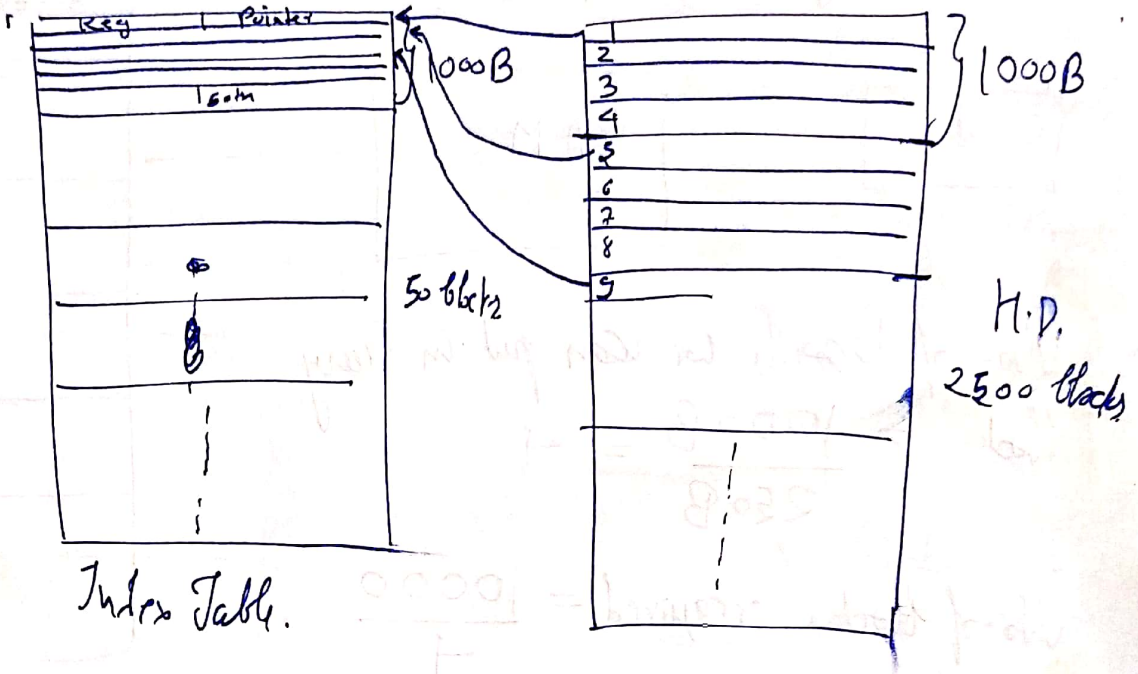
$\log_2(N) = \log_2(2500) = 12$

Now we check for the case of entering

Consider a Hard Disk in which the [illegible] =

What is the avg time complexity to search a record from Index table if Index table entry = 20B (Key + Pointer) → (Page no.)
10B 10B

Block size of Index Table = Block size of H.D.



each block in Index Table contains = $\frac{1000B}{20B} = 50$

No. of blocks in index Table = $\frac{2500}{50} = 50$ (For the case of sparse, if the data is ordered in H.D.)

each / entry in
No. of blocks in index Table = $\frac{10000}{50} = 200$ (For the dense case, if the data is unordered in H.D.)

Q6 (Total records contain
Key means the value by which each record is uniquely identify &
in case of sparse → pointer is the start of that particular block in H.D.
& in case of dense → pointer is the pointer to that particular record in H.D.

searches [redacted] in sparse: $\log_2 50 + 1 = 7$

↑ in searching key ↑ searching data in block.

searches [redacted] in dense $\log_2 200 + 1 = 9$.

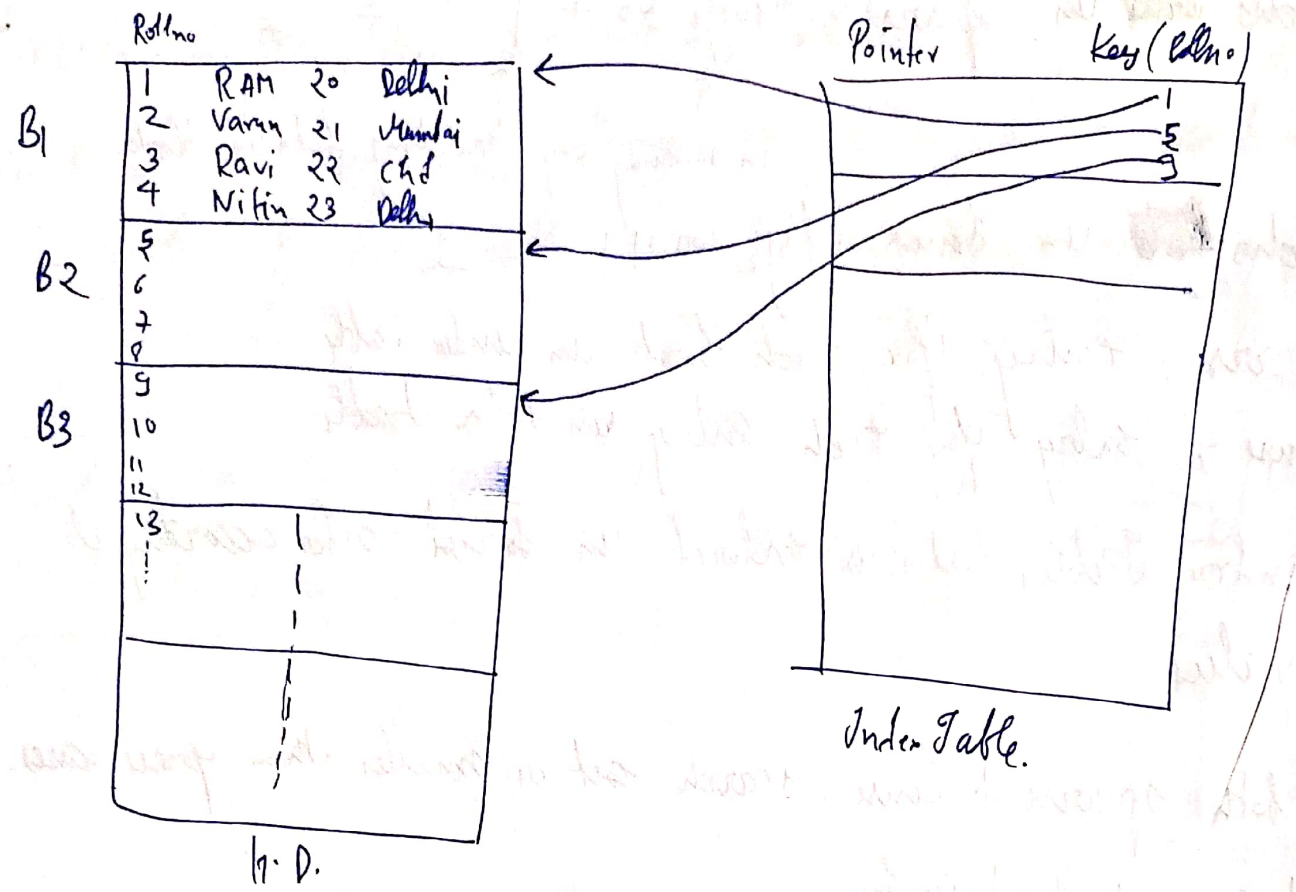
* In sparse, entry for each block in index table.
* In dense, entry for each entry in index table
* In index table, data is entered in sorted order according to key value.

In both sparse & dense, search cost is smaller than previous cases that are without index.

Primary Index

To apply primary index
→ ordered file (sorted)
→ key value (unique)

	Key	Non-key
Ordered file	Primary Index (sparse)	Clustered Index (sparse)
Unordered file	Secondary Index	Secondary index

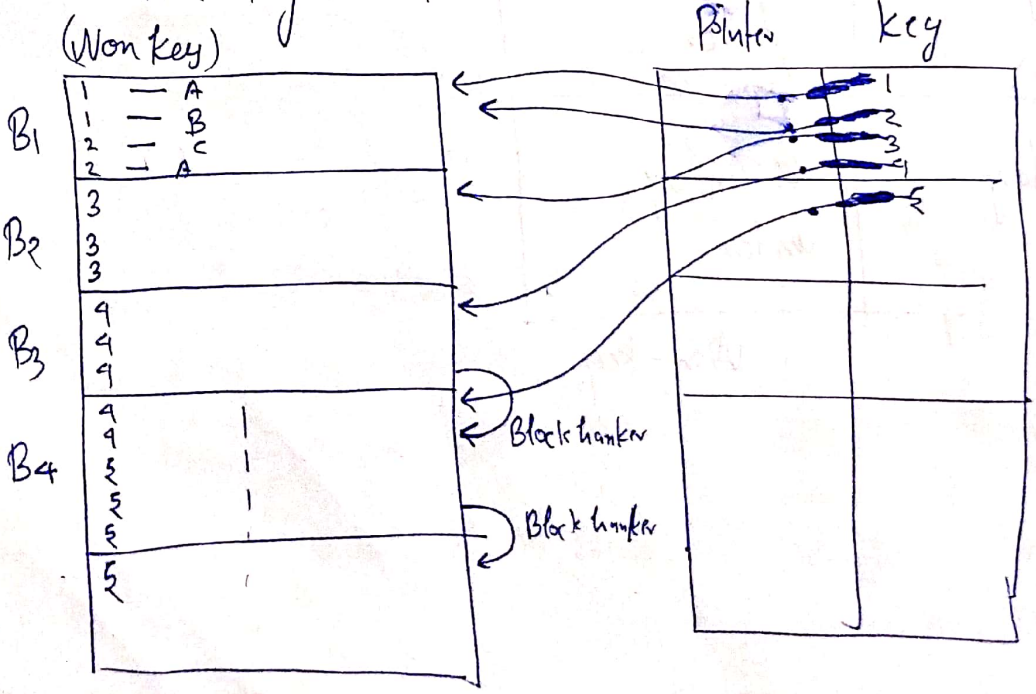


No. of entries in Index Table = No. of blocks in HD (for sparse)

Search time $\log_2(N)+1$, N = No. of blocks in the index table.

Clustered Index (Sparse)

- To apply clustered index
  - → ordered file
  - → Non-key value.



$\log_2 N + 1 + 1$

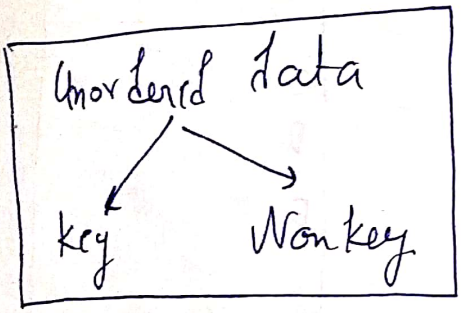
↓ No. of blocks in index table

↑ due to block anchor.

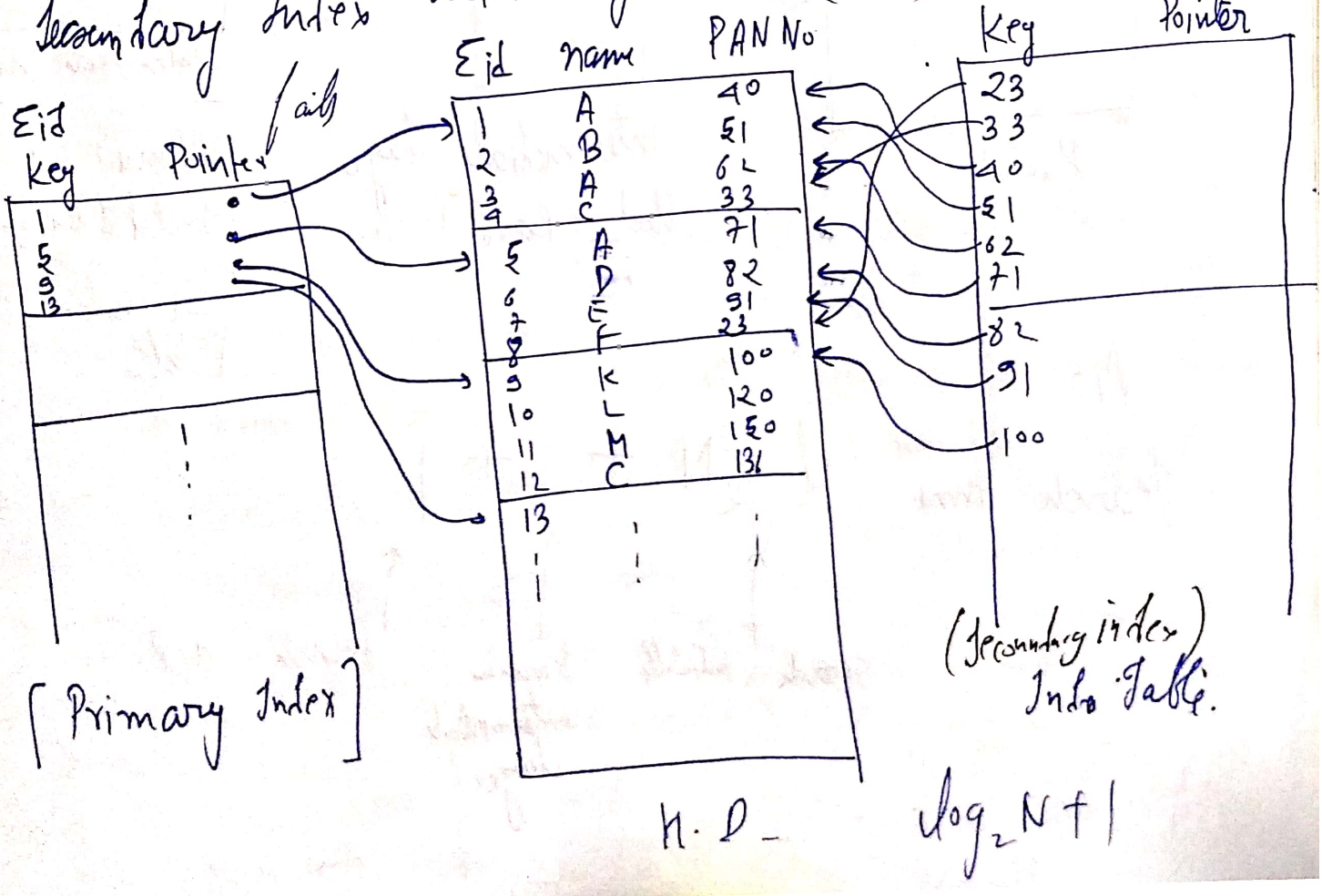
As some of the 4's are in two blocks because they cannot fit in the one block of H.D., hence we use the concept of block anchor so that we can know that 4's (or any other value) present in more than one blocks

→ Cluster index → atmost 1

Secondary Index



1) Secondary Index with key value (Unique) PAN No.

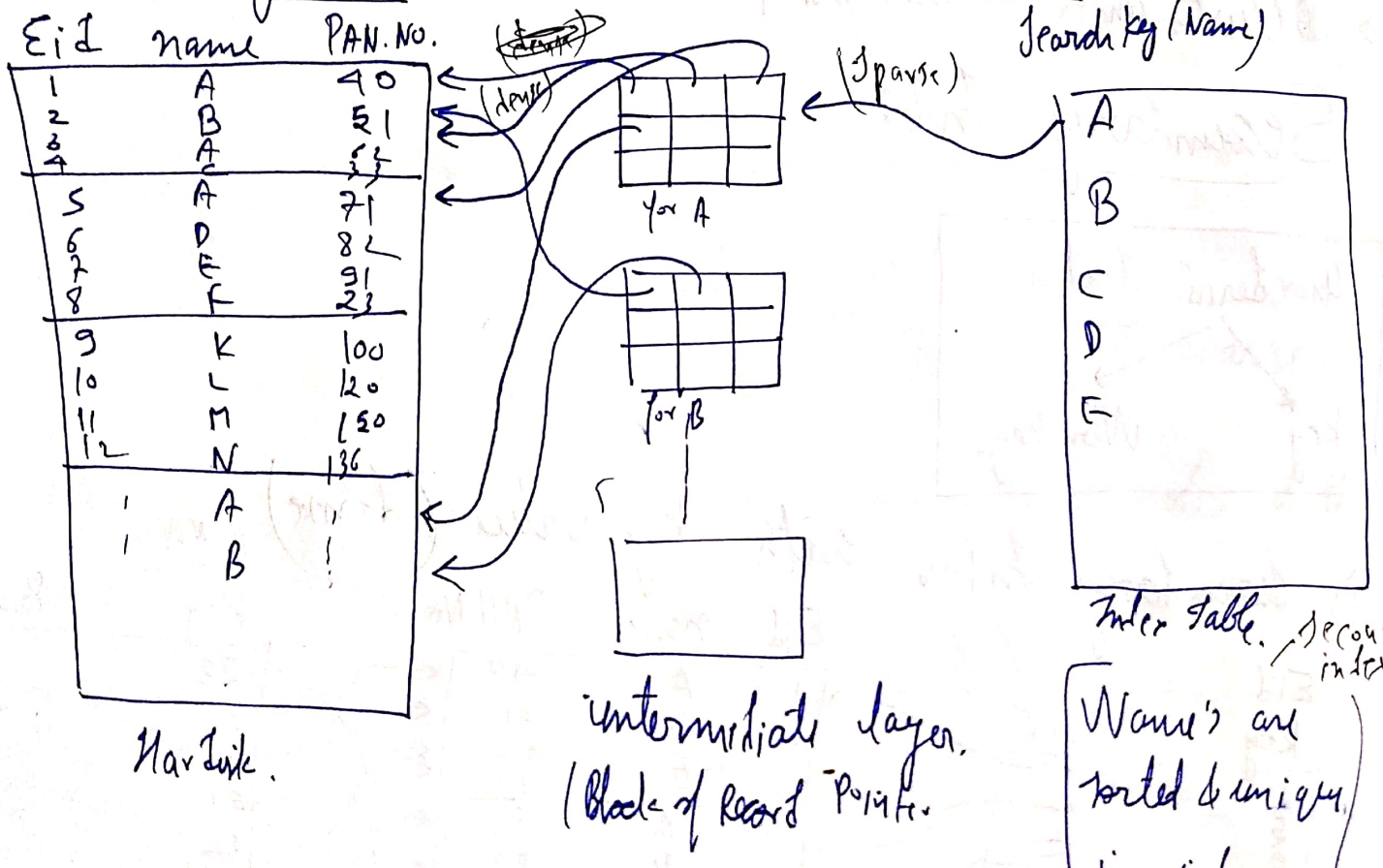


$\log_2 N + 1$

If searching is based on PAN No. (which is unordered) in Hard disk, then primary index is fails.

& if PAN No. is unique (but unsorted), then we search based on PAN No. & sort them acc^n to PAN No. in the index Table. & searching is dense type

Secondary Index with non-key value.



No. of IOs

Search Time $\log_2 N + 1 + 1$

- $\log_2 N$: search in Index Table
- $+1$: search in intermediate layer
- $+1$: search in H.D.

Immediate Database Modification (Log based Recovery)

(Undo/Redo) strategy.

A = 100
B = 200

D.B.

T_1
R(A)
A = A + 100
W(A) | 200
R(B)
B = B + 200
W(B) | 400
Commit.

A = 200
B = 200

A = 200
B = 200

$< T_1$, start >	Redo
$< T_1, A, \underset{\text{old}}{100}, \underset{\text{new}}{200} >$	
$< T_1, B, 200, 400 >$	
$< T_1$, Commit >	

Redo
$< T_1$, start)
$< T_1, A, 100, 200)$
$< T_1, B, 5000, 6000)$
$< T_1$, Commit)
undo
$< T_2$, start)
$< T_2, C, 700, 800)$

$T_1 \rightarrow$ redo
$T_2 \rightarrow$ undo

→ In Immediate Data base modification, when we write W(A) then Allocation new value of A is updated in D.B. even before commit.

→ When data base administrator look at log table, if T_1 is both start & commited as the case above, then update the A & B new value in data base. but if T_1 starts & fails before commit → then older value of A & B is stored in the Database, as older value of A & B is saved in log table. But not in differed

A = 100
B = 200

A = 200
B = 200

A = 200
B = 400

T_1
R(A)
A = A + 100
W(A)
R(B)
B = B + 200
W(A)
fail

$< T_1$ start>
$< T_1, A, 100, 200)$
$< T_1, B, 200, 400)$
→ undo

A = 100
B = 200

for fail before commit

Important Points

The view of the total data base content is Conceptual view, i.e. how many tables, how they are related etc.

Cardinality → No. of Tuples (rows)

Degree → No. of attributes (columns)

Foreign key maintains referential integrity which leads to consistency.

Concurrency Control Protocol

→ To achieve serializability & Recoverable. By locking protocol.

Shared - Exclusive Locking

→ Shared-lock(S) → if Transaction locked data item in shared mode then allowed to read only.

→ Exclusive Lock(X) → If Transaction locked data item in exclusive mode then allowed to Read & write both.

Eg:

T_1	T_2
S(A) // Shared lock on A	X(A) // Exclusive lock on A
R(A)	R(A)
U(A) // Unlock A	W(A)
	U(A) // Unlock A.

grant ↓ \ Request →	S	X
S	Yes	No
X	No	No

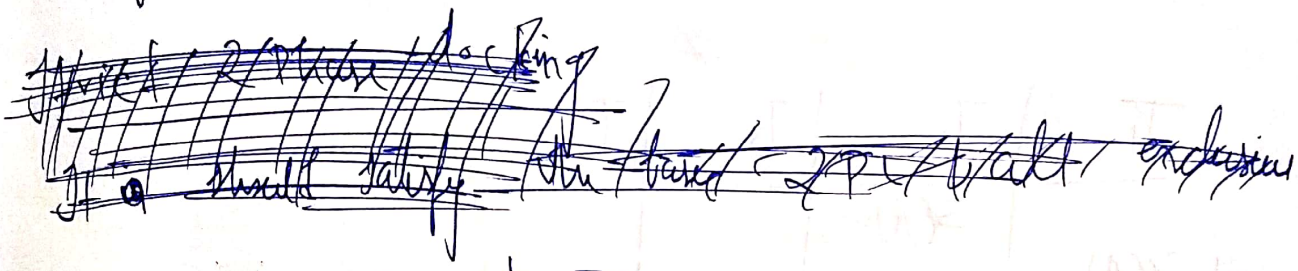
SS → no conflict

SX, XS, XX → Conflict occur R/W.

To achieve consistency by not allowing conflict pair

Problems in S/X locking

1) May not sufficient to produce only serializable schedule
2) May not free from irrecoverability
3) May not free from deadlock.
4) May not free from starvation.



1) ex

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
X(B)	
R(A)	
W(B)	
U(B)	

@ the given schedule cannot convert it into $T_1 \rightarrow T_2$ / $T_2 \rightarrow T_1$ hence S/X locking may not produce serializable schedule.

2) ex

A = 100 90

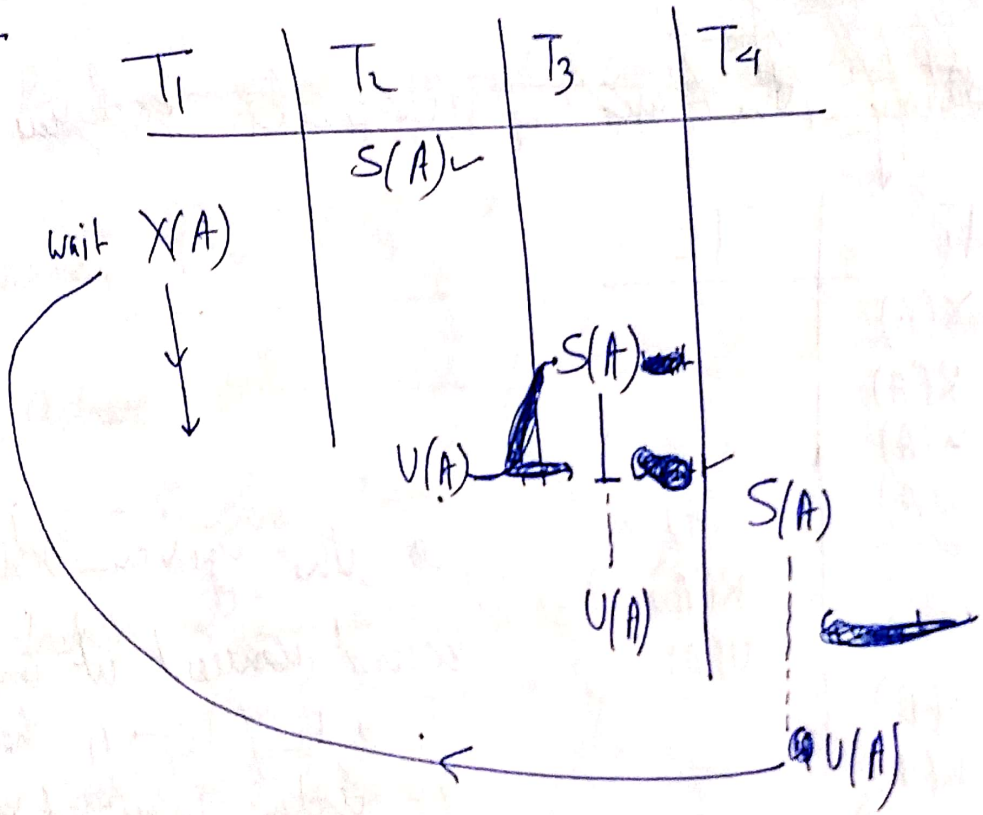
T_1	T_2
X(A)	
R(A)	
A = A - 10	
W(A)	
U(A)	
	X(A) S(A)
	R(A) R(A)
	Commit // A = 90
Roll back	
fail	

but after commit, no roll back is possible, hence garbage value is committed at T_2 because T_1 fails, hence irrecoverable.

3 ex →

T_1	T_2
X(A) ←	
	X(B) ← , deadlock !!!
wait X(B)	
	X(A) wait

4th ex →



→ So starvation is deadlock but not for infinite time.

→ Starvation for longer duration of time

2-phase locking (2PL)

Growing phase: locks are required & no locks are released

Shrinking phase: locks are released & no locks are acquired

S/X
T_1
X(A)
S(B)
R(B)
W(A)
R(B)
S(A)
R(C)
S(C)
R(C)
U(A)

growing phase

shrinking phase.

** growing phase & shrinking phase → ensures serializability & hence consistent

ex

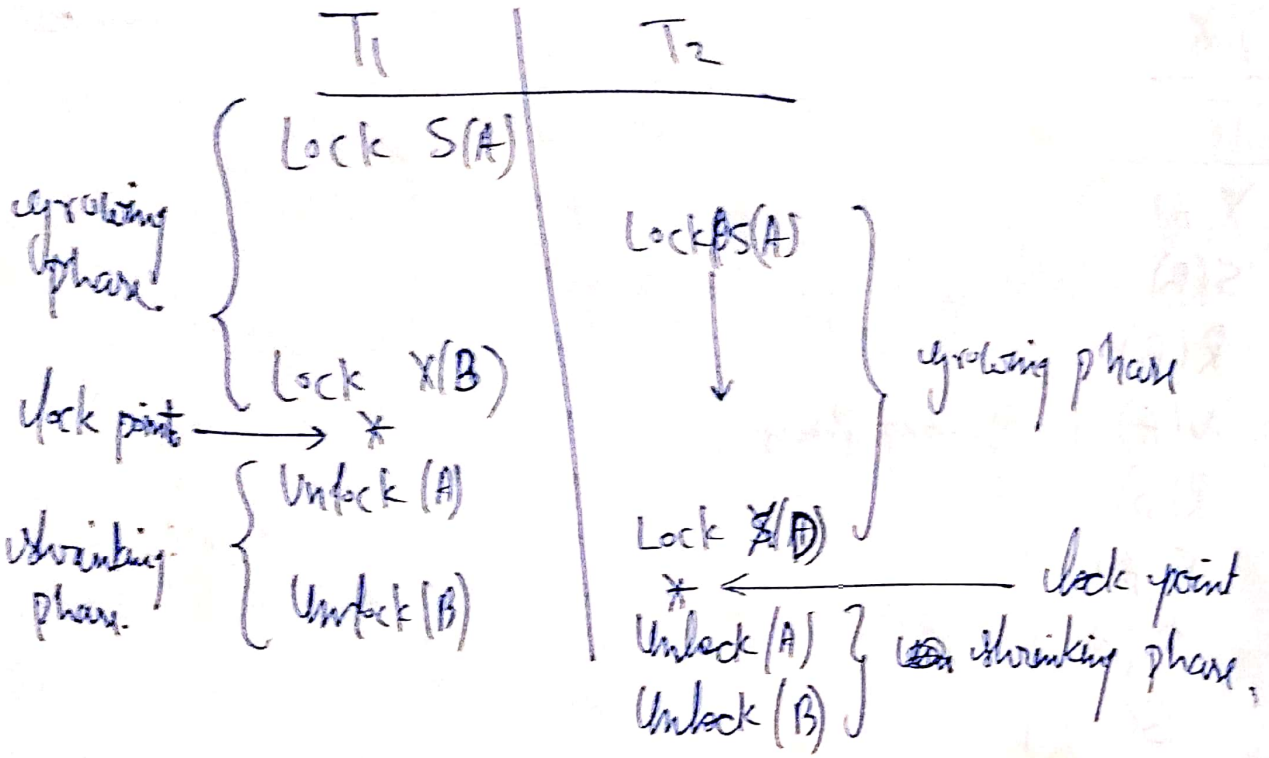
T_1	T_2
X(A)	
R(A)	
W(A)	
	*S(A)
	R(A)
S(B)	
R(B)	
U(A) *	
⋮	
U(B)	

growing phase

shrinking phase

$\Rightarrow T_1 \rightarrow T_2$

hence 2PL implement serializability



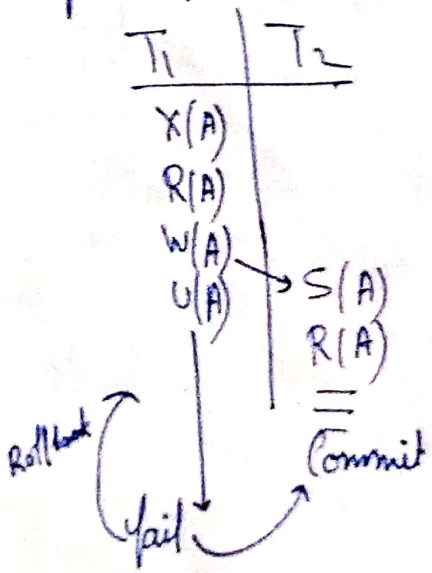
Lock point → Point where transaction goes from growing to shrinking phase.

2PL always ensures serializability & the one who had lock point more upper will execute 1st. So in the above schedule $T_1 \rightarrow T_2$

Drawbacks of 2PL

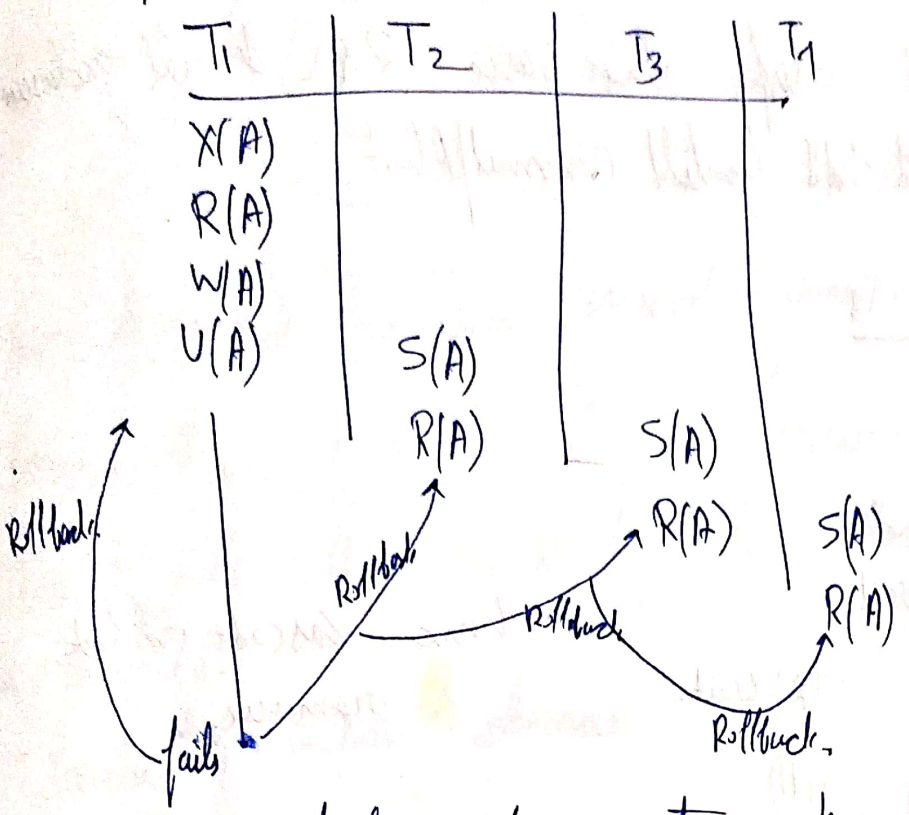
* Advantage of 2PL is → It ensures serializability always.

1) May not free from irrecoverability



as T_2 reads T_1 data so T_2 should also to rollback but it is already committed before.

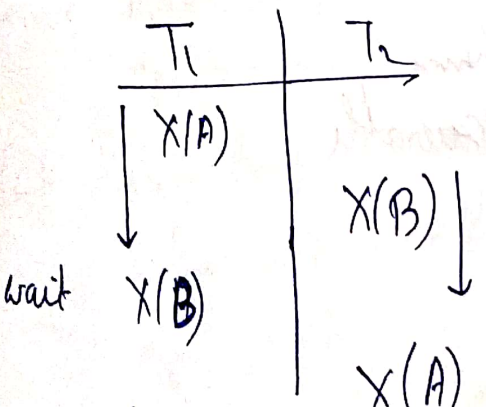
2) Not free from cascading Rollback.



So No Cascadless.

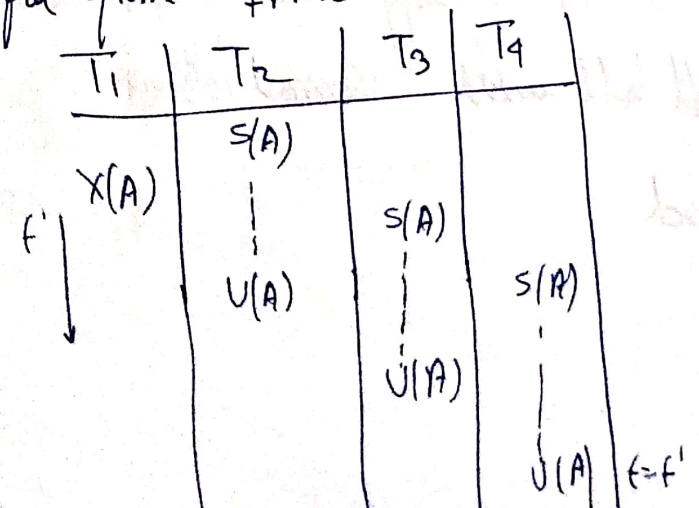
So due to failure of one transaction, multiple rollback occurs, this is known as cascaded rollback, which is a problem!!

3) Not free from Deadlock



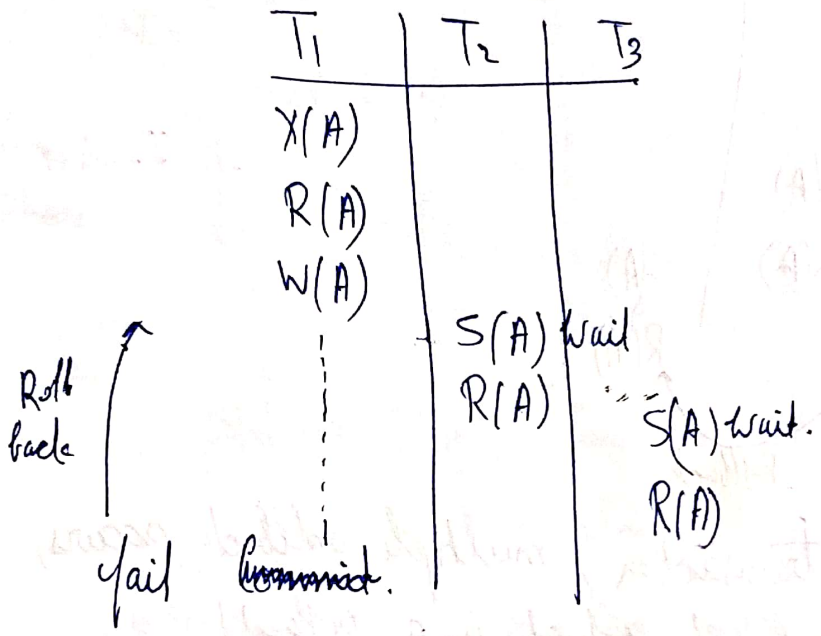
Deadlock !!! (Circular Waiting)

4) Not free from Starvation:

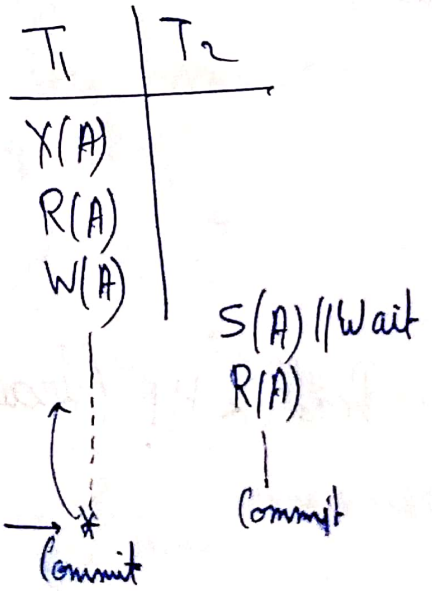


Strict & Rigorous 2PL

Strict 2PL → It should satisfy the basic 2PL & all exclusive locks should hold until commit/Abort.



hence cascade rollback remove.



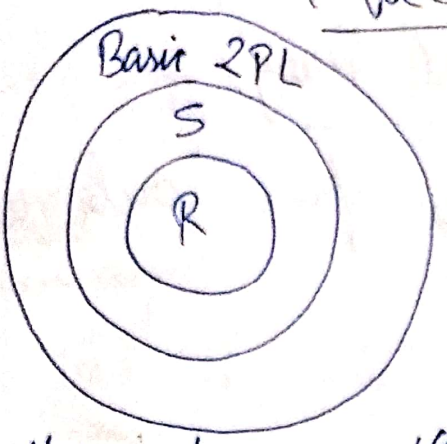
hence. Recoverable

Rigorous 2PL → It should satisfy the basic 2PL & all shared, exclusive locks should hold until commit/Abort.

→ Removes cascaded rollback

→ Removes Irrecoverability

(query)



S → strict
R → Rigorous.

but deadlock & starvation cannot be removed by S & R 2PL.

Conservative 2PL (that are required)

→ It will take all the locks before start, hence deadlock & starvation is removed.

→ Practically impossible

*** Find Nth Highest Salary using SQL

ID	Salary
1	10000
2	20000
3	30,000
4	40,000
5	40,000
6	50,000

Select Max(Salary) from emp;
o/p: 50,000

Select Max(Salary) from emp where
Salary Not in (Select Max(Salary) from emp);
o/p: 40,000

Select id, salary from emp e1 where N-1 = (Select Count(Distinct Salary)
from emp e2
where e2.salary > e1.salary)

for N=4, o/p: 2, 20000
for N=3, o/p: 4, 30000

ID	Salary
1	10,000
2	20000
3	20000
4	30000
5	40000
6	60000

ID	Salary
1	10,000
2	20,000
3	30,000
4	30,000
5	40,000
6	50,000

Correlated Subquery ('Synchronized Query')

→ It is subquery that uses values from outer query

→ Top to down approach

→ Like nested loops : OQ(IQ)

Find all employee details who works in a department.

```
Select * from emp where
exists (select * from
        dept where
        dept.Eid = emp.Eid);
```

emp

Eid	name	address
1	A	Delhi
2	B	Pune
3	A	Chd
4	B	Delhi
5	C	Pune
6	D	Mumbai
7	E	Hyd.

Dept

Did	Dname	Eid
D_1	HR	1
D_2	IT	2
D_3	MKTG	3
D_4	Testing	4

O/P

1	A	Delhi
2	B	Pune
3	A	Chd
4	B	Delhi

Nested Subquery

→ Bottom up

```
select * from emp
where eid in (select e_id from dept)
```

1, 2, 3

o/p: 1 A
2 B
3 C

Correlated Subquery

→ Top to down approach

```
select * from emp where
exist( select id from dept
  where emp.e_id = dept.e_id)
```

o/p: 1 A
2 A
3 C

Joins

→ Cross product + condition

```
select attributes from
emp, dept where
emp.e_id = dept.e_id;
```

o/p: 1 A
2 B
3 C

fast ↑↑
more space

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

Dept

dept.no	name	E_id
D_1	LT	1
D_2	HR	2
D_3	MRKT	3

Select

inner query executes only once

1 row of outer, compare with all inner rows.

Time stamp Ordering Protocol

→ unique value assign to every transaction.
→ It tells order (when they enter into system)

10:00 — T_1 — 100 — older
10:10 — T_2 — 200 — younger
10:15 — T_3 — 300 — youngest.

→ TS(T_i) → Time stamp of the transaction
→ Read_TS (RTS) → Latest transaction no. which performed Read successfully.
→ Write_TS (WTS) → " " " " " Write "

T.S. | 10 | 20 | 30

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

$\Rightarrow RTS(A) = 30$

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

$\Rightarrow WTS(A) = 20$

RTS & WTS is for each variable.

* Earlier Transaction will finish 1st.

Rule

1) Transaction T_i issues a Read(A) operation
   a) if $WTS(A) > TS(T_i)$, Rollback T_i
   b) Otherwise execute R(A) operation
      Set $RTS(A) = Max(RTS(A), TS(T_i))$
2) Transaction T_i issues write(A) operation.
   a) if $RTS(A) > TS(T_i)$, then Rollback T_i
   b) if $WTS(A) > TS(T_i)$, then Rollback T_i
   c) Otherwise execute write(A) operation
      Set $WTS(A) = TS(T_i)$

No problems !!

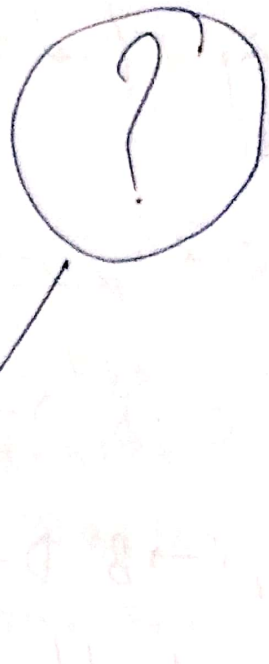
100 T_1	200 T_2 young.
R(A)	
	R(A)

Or

R(A)	
	W(A)

Or

W(A)	
	R(A)



Problems

100 T_1	200 T_2
	R(A)
W(A)	
Rollback	

|| As R(A) may read garbage.

T_1	T_2
	W(A)
Rollback ↑ R(A)	
commit	
cannot rollback	* fail

hence T_1 should Rollback. at R(A) itself it not allow error.

It follows

$T_1 \rightarrow T_2$

Serializability.

100 T_1	200 T_2
	W(A)
Rollback ↑ W(A)	
commit	
	fail

Lost updation problem

For the following functional dependencies, find the correct minimal cover.

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

step 1

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

step 2

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

step 3

Now, we have to take care of left,

$AC \to D$

if we remove A,

$C^+ = CB$, so in C^+, A did not come hence A cannot remove from AC

if we remove B,

$A^+ = A, B$, so in A^+, B did not come hence C cannot remove from AC

So AC cannot be further reduced.

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

now combine Rhs.

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

cannot remove

$A^+ = A$ $A \to B$

$B^+ = B$, $C \to B$

$D^+ = DBC$, $D \to A$

but

$D^+ = DBC$

hence $D \to B$ can be remove.

$AC^+ = ACB$

hence $AC \to D$ cannot remove.

Structured / Query / Lang.

SQL

→ from SEQUEL

→ based on RA & TRC

Creation of ~~Table~~ Schema

SCHEMA: Groups logical object together.

Syntax: CREATE SCHEMA schema_name [AUTHORIZATION owner_name];

ex CREATE SCHEMA FINANCE AUTHORIZATION RAVI;

GRANT SELECT ON FINANCE.* to user1;

Creation of Table

CREATE TABLE <table name> (<col1> datatype (width),
<col2> datatype (width), ...);

ex CREATE TABLE Customer (Name Varchar2(30),
Cust_id Number
Address varchar2(50));

~~DATATYPE~~

Data-Type

Numeric	character	Date	Boolean
NUMBER	CHAR	DATE	BOOLEAN
FLOAT	VARCHAR2	TIME	
INT	NCHAR	TIMESTAMP	
REAL	NVARCHAR2	INTERVAL	
DECIMAL	LONG		
BINARY FLOAT etc	RAW		
	LONG RAW		

Constraints

1) NOT NULL → Null values allowed.
2) UNIQUE
3) Primary key → UNIQUE + NOT NULL
4) Foreign key
5) CHECK
6) DEFAULT
7) REF. Constraints

ex- Requirements:

Dept: dept-id (P.K.)
dept-name (UNIQUE)
location

emp: emp-id (P.K.), Emp-name (NOT NULL)
job, dept-id (REF dept-id of Dept)
mgr, salary (≥5000)
comm (If not given default 0)
email, phone → unique

Constraint can be defined at two level
a) Column level b) table level

* NOT NULL constraint should be defined at column level.
* Composite key should be defined at table level.

SQL: CREATE TABLE Dept (dept_id NUMBER PRIMARY KEY,
dept_name varchar2(30),
location varchar2(100),
UNIQUE (dept_name));

CREATE TABLE emp (emp_id NUMBER PRIMARY KEY,
emp_name varchar2(30) NOT NULL,
dept_id NUMBER REFERENCES Dept(
dept_id),
job varchar2(30), mgr varchar2(30)
salary NUMBER CHECK (salary ≥ 5000)
comm NUMBER DEFAULT 100,
email varchar2(30)
phone varchar2(13),
* UNIQUE (email, phone));

We can represent no. using string of characters using varchar2.

Insert

Syntax1

INSERT INTO <tablename> values (attr_value1, attr_value2 ...);

ex INSERT INTO Dept values (1, 'CSE', 'Hyd');

* Attribute value should be in the order of attr. list
* No. of attributes should be equal to no. of attr. in the attr. list.

Syntax2

INSERT INTO <tablname) (attr1, attr2, attr3...)
values (att-value1, att-value2, ...)

ex INSERT INTO Dept (dept-id, dept_name, location)
values (2, 'ece', 'kap'),

INSERT INTO Dept (dept-id, dep-name)
values (3, 'eee');

Dept-id	Dep-name	location
1	CSE	HYD
2	ECE	KAP
3	EEE	NULL

DELETE : Deletes the data from existing table.

Syntax :

DELETE FROM <table name> [WHERE<condition>];

Dept

dept-id	dept-name	location
1	cse	hyd
2	ece	kcap
3	eee	hyd

Emp

emp id	dep-id	e-name
1	3	abc
2	2	def
3	1	ghi

DELETE FROM emp where emp-id = 1 ;

DELETE * FROM emp;

DELETE FROM Dept where dept-id = 1 ;

Referential integrity constraint get violated.

DROP → Delete both data & table.

Syntax: DROP <TABLE <table name>;

DROP TABLE emp;

DROP TABLE Dept;

After drop, we cannot insert, 1st we create table again & then we can insert.

Referential integrity constraint get voilated here too.

UPDATE: Used for updating the data

Syntax

UPDATE <table name> SET <attr> = <value>
... [WHERE <condition>];

Ex: UPDATE Dept SET dept_name = 'IT'
WHERE location = 'KGP';

UPDATE Emp SET dept_id = 1
WHERE emp_id = 2;

UPDATE Dept SET dept_id = 4
WHERE dept_name = 'ece';

→ Update may lead to Referential integrity get violated.

Referential Integrity Constraint

Dept:

dept_id	dept_name
1	CS
2	ECE
3	ECC
4	IT
5 9	CE
6 7	ME

Emp:

emp_id	dept_id
1	3
2	3
3	4
4	6 NULL
5	5 9
6	NULL

Constraint to maintain Referential integrity

Three triggered action:

* ON DELETE * ON UPDATE

→ SET NULL
→ SET DEFAULT
→ SET CASCADE

CREATE TABLE emp (emp-id NUMBER, REFERENCES Dept(dept-id));

Set NULL

dept-id NUMBER REFERENCES Dept (dept-id)
ON DELETE SET NULL ON UPDATE SET NULL

ex. DELETE FROM Dept WHERE dept-id=1;
UPDATE Dept SET dept-id = 7 WHERE dept-id = 6;

Set Default

dept-id NUMBER DEFAULT 3 REFERENCES Dept(dept-id)
ON DELETE SET DEFAULT ON UPDATE SET DEFAULT

ex. DELETE FROM Dept WHERE dept-id = 2;
XUPDATE Dept SET dept-id = 8 WHERE dept-id = 3;

wrong → (as DEFAULT ID = 3)

It is also valid → ON DELETE SET NULL ON UPDATE SET CASCADE

SET CASCADE

dep-id NUMBER REFERENCES Dept(dept-id)

ON DELETE SET CASCADE ON UPDATE SET CASCA

ex. DELETE FROM Dept WHERE dept-id = 4;

UPDATE Dept SET dept-id = 9 WHERE dept-id = 5

ALTER:

⇒ It is used to add, delete or modify columns in an existing table.

⇒ Also used to add & drop various constraints on an existing table.

ex. CREATE TABLE Emp (emp-id NUMBER,
emp-name char(10) UNIQUE,
Salary NUMBER CHECK (Salary ≥ 5000),
Phone varchar2(13),
Pincode varchar2(6));

* ALTER TABLE Emp ADD PRIMARY KEY (emp-id);
* ALTER TABLE Emp DROP UNIQUE (emp-name);
* ALTER TABLE Emp MODIFY CHECK (Salary ≥ 10000);
* ALTER TABLE Emp ADD Addr varchar2(30);
* ALTER TABLE Emp MODIFY phone varchar2(20);
* ALTER TABLE Emp DROP COLUMN Pincode;

SELECT

Dept

dept-id	dept-name
1	CSE
2	IT
3	ECE
4	EEE
5	ME

emp

emp-id	dept.	emp-name	Job
1	5	Sindhu	9
2	2	Srinivas	6
3	1	Santosh	9
4	2	Sanjay	9
5	4	Sindu	6
6	3	Sanjay	C

Syntax:

SELECT < attr. list >

FROM <table_list>,

WHERE < conditions>

TV (projection)

cartesian product

T (selection)

ex. SELECT emp_name from emp WHERE dept=1

o/p : Santosh

ex. names, emp-id of the employee who works for IT dept.

Select emp-id, emp_name from Emp, Dept

WHERE dept.= dept-id AND dept-name='IT';

o/p: 2 Srinivas

4 Sanjay

ex. Retrieve the jobs at all the employees

⇒ Select job from emp;

o/p: {9, 6, 9, 9, 6, 3}

⇒ Select distinct job from emp;

{9, 6, C}

Relation Algebra — TV → No duplicate
SQL — Select → Duplicates

VIEW

student

stu_id	stu_name	dept
1	Jindu	2
2	Jaumya	1
3	Srinavas	3
4	Santosh	2
5	Subbu	4
6	Sindhani	2
7	Satheesh	2

dept

dept_id	dept_name
1	CSE
2	ECE
3	ME
4	EEE
5	CE

Create table stu_ece as (select stu_name,

Now if want to create ~~normal~~ table that shows only ece stu-id & dept, then

Create table stu_ece as (select stu_name, stu_id
from student, dept
where dept = dept_id AND
dept_name = 'ece');

stu_ece

stu_name	stu_id
Jindhu	1
Santosh	4
Sindhani	5
Satheesh	2

But this is static table, as if some values of student or dept changes then stu_ece shows wrong data, so need of "VIEW".

VIEW:- It is a virtual table based on the result set of a SQL statement.

Syntax: CREATE VIEW <viewname> AS

SELECT <cd names>

FROM <cd_names>

WHERE 'condition'

Create VIEW stu_ece as(select stu_name, stu_id
from student, Dept
where dept=dept_id AND
dept_name='ECE');

ALIASING

Dept

dept_id	dept_name
1	CSE
2	ECE
3	EEE
4	IT

Emp

emp_id	dept_id	sup_id	salary	emp_name
1	2	7	15k	a
2	3	4	20k	b
3	1	7	20k	c
4	4	7	30k	d
5	2	4	20k	e
6	1	7	20k	f
7	1	-	50k	g

For each employee, retrieve employee id, name, salary and the name of his or her immediate supervisor.

```
select E.Emp_id As "Employee Id",
E.Salary As "Salary",
E.Emp_name As "Employee name", S.Sup_id As "Supervisor ID",

FROM Emp E,
     Emp S
WHERE E.Sup_id = S.Emp_id;
```

```
Select E.emp_id As "Employee ID", D.dept_name As "Department name"
from Emp E,
     Dept. D,
where E.dept_id = D.dept_id;
```

Pattern matching

LIKE:

i) %, represent any sequence of 0 or more characters
ii) _, used to replace a single characters.

a% → all name starts with a
_a% → 2nd character is a in name
%a → ends with a
"%a_"

name	% marks	rank	email
Jindhu	78.8%	1234	abc-def@ghi.klm
Jaanya	78.8%	1235	abc.def
Rani	90%	66	abc-d@ghi.gt

Select * from student where name like 'R%';
→ starts with R.

- Select name from students where rank like '----';

Select name from student where marks like '90\%' escape '\'
and email like 'abc_%@%.%' escape '\';

Select name from student where marks like '90\%' escape '\' and
email like 'abc_%@%.%' escape '\';

o/p: Ravi

if not like is used

o/p Indu
Samyag

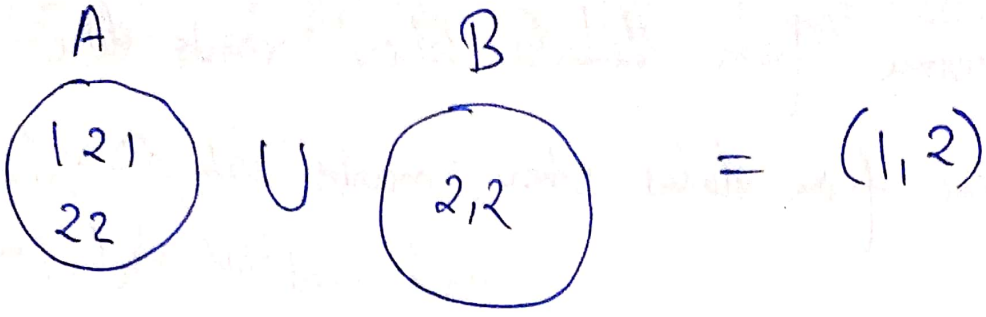
Set-Operation

UNION
INTERSECT
EXCEPT
} (U, ∩, −)

UNION ALL
INTERSECTALL
EXCEPT ALL
} to allow duplicate as well.

Retrieve the list of stuid's who got either grade 'A' or grade 'B'

```
SELECT stuId FROM ENROLL
    WHERE grade = 'A'
SELECT stuId FROM ENROLL
    WHERE grade = 'B'
```

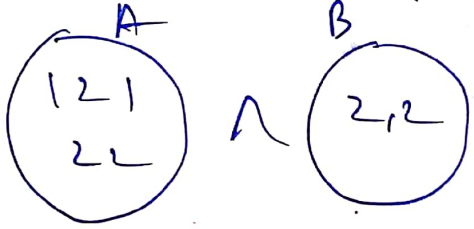


if we write UNION ALL

= (1, 2, 1, 2, 2, 2, 2)

ex List of stud Ids who got Both grade 'A' & grade 'B'

```
SELECT stuId FROM ENROLL
    WHERE grade = 'A'
INTERSECT
SELECT stuId FROM ENROLL
    WHERE grade = 'B'
```

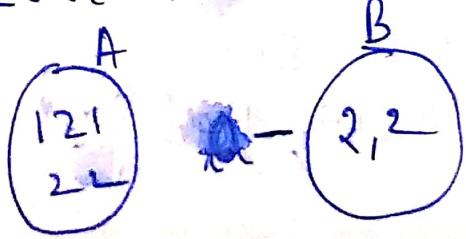


Intersect = (2)

Intersect All = (2, 2)

ex Got grade 'A' but not grade 'B'

```
SELECT stuId FROM ENROLL
    WHERE grade = 'A'
Except
SELECT stuId FROM ENROLL WHERE grade = 'B'
```



except = {1}

except all = {1, 1, 2}

Arithmetic Operator

+, -, *, / (For numeric only)

Increas the salary of the employee by 10% & display the salary & employeename

Select EmpName, Sal * 0.1 FROM EMPLOYEE;

CONCATENATE

SELECT FNAME || LNAME AS "FULL NAME"
FROM EMPLOYEE;

EMPLOYEE.

FNAME	LNAME
ravi	raula

FULLNAME
raviraula

BETWEEN

SELECT * FROM EMPLOYEE WHERE Salary
BETWEEN 30000 AND 50000;

NOT BETWEEN can be used; 30000, 50000 are included.

* SELECT * FROM EMPLOYEE WHERE Emp_Name
BETWEEN 'A' AND 'D';

A
B
C
D
Da

if Not B/w

B ✗
C ✗
D ✗ } other than this all display

SELECT * FROM EMPLOYEE WHERE HIRE-date BETWEEN '01-JAN-2010' AND '31-DEC-2010'

ORDER BY:

Order by is used to order/sort the result of the SQL query in ascending (or) descending order.

By default: Ascending

Select A,B,C from R ORDER BY A, B, C;

1 2 1
1 2 3
2 1 1
2 1 3
3 4 3
3 5 4

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

Select A, B, C from R ORDER BY A DESC, B, C;

3 4 3
3 5 4
2 1 1
2 1 3
1 2 1

* Select A, B, C from R ORDER BY A DESC, B, C DESC

3	4	3
3	5	4
2	1	3
2	1	1
1	2	3
1	2	1

Dealing with NULL Values in various Contexts

Arithmetic expression (+, −, *, /)

Logical expressions

A
5+1 = 6
4+1 = 5
NULL+1 = NULL
(1+1) = 1

(1 < NULL) AND (1<2)
unknown

A	B	A AND B	A OR B
T	T	T	T
T	F	F	T
F	T	F	F
F	F	F	F
T	UK	UK	T
F	UK	F	UK
UK	UK	UK	UK

NOT(T) = F
NOT(F) = T
NOT(UK) = UK.

#

A	B
a	1
b	2
c	NULL
d	NULL

Select A FROM R
WHERE B = NULL

* Two NULLS ARE always different

o/p: empty

SELECT A FROM R WHERE B IS NULL

c

d

IS NOT NULL (in above query) ↑

a

b

c.Name = d.name.

a
b
c
NULL
NULL

a
b
c
NULL
NULL

o/p a a
b b
c c

R

A	B
a	1
b	2
c	NULL
d	NULL

Select B FROM R

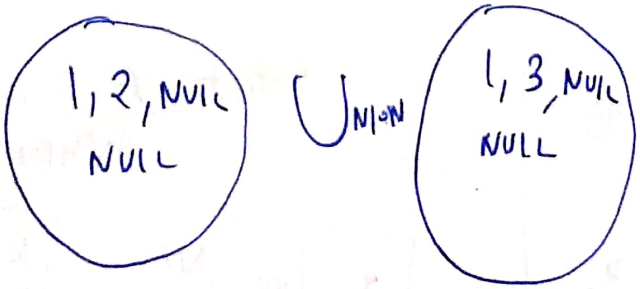
1, 2, NULL, NULL

~~Select B~~

Select DISTINCT B FROM R

1, 2, NULL

#



o/p: 1, 2, 3, NULL

UNION ALL

o/p: 1, 1, 2, 3, NULL, NULL, NULL
NULL

∩ o/p: 1, NULL

INTERSECT ALL:

o/p. 1, NULL, NULL.

IN OPERATOR

eg: Retrieve the Fname of employee who works for departments 2, 3, 4, 5.

```
SELECT Fnames FROM employees WHERE DNO
                    IN (2, 3, 4, 5)
```

eg: Find Fname, Address of the employee who works for the department located in Newyork

```
Select Fname, Address FROM employee WHERE DNO in
( Select DNO FROM Dept_locations WHERE
                    location = 'NEWYORK');
```

non "correlated query"

eg: Retrieve the Fname, Lnames of all the managers

```
Select Fname, Lname from employee e
        WHERE SSN IN (Select Mgr_ssn from Department
                                                d);
```

eg: Retrieve the FNAME of each employee who has a dependent with the same fname & same sex as the employee

```
Select e.Fname from employee e where e.ssn in (Select essn from
Dependent d WHERE e.fname = d.Dependent_name AND
        e.sex = d.sex);
```

"correlated" query

ex

Find out all the employee id's who worked on same (or) any project on which employee 10 has worked for same number of hours.

Select distinct essn from WORKS_ON WHERE (pno, hours) IN
(Select pno, hours from WORKS_ON WHERE essn = 10);

ANY/SOME, ALL (>, <, >=, <=, <>, =)

* Find out fnames of all employee whose salary is greater than all employee in department No. 5

Select fnames from EMPLOYEE WHERE
salary > ALL (Select salary from EMPLOYEE
WHERE Dno = 5)

if all → any, then if any salary greater than the salary of emp of dno = 5 will be printed.

to all 11K✓ (10K, 20K, 30K)
31K✓

to use any 31K X (10K, 20K, 30K)
11K✓

* = any or = some or in

* this can't be used to compare more than one values but in can

<> ⇒ Not equal to

<> any ≡ <> some ≡ not in

But <> any, <> some cannot be used to compare more than one row value but not in can.

EXIST / NOT EXISTS

e

ssn	Fname	sex

d

essn	sex	Dname

Ex Retrieve the fname of the employee who have dependents with same fname, sex as that of the employee.

Select e.fname from employee e WHERE EXISTS (Select *
FROM Dependent d WHERE e.ssn = d.essn AND e.sex
= d.sex AND e.fname = d.Dependent_name) ;

Ex Retrieve the fname of the employee who have no dependents.

Select fname from Employee WHERE NOT EXISTS (Select *
from Dependent where ssn = essn);

* Both of the above query are correlated query.

Aggregate functions

set → no duplicate
multiset → have "

AVG, MIN, MAX, SUM, COUNT

Select AVG(Salary) From emp;
Select COUNT(Salary) From emp

AVG, SUM
FOR NUMERIC only

Select DISTINCT AVG(Salary) From emp;

Select AVG(MARKS_A + MARKS_B) from emp;

Impt. Aggregate func^n cannot be used with WHERE clause.

Dealing with NULL Value in Aggregate func^n

* All aggregate func^n except count(*) ignore NULL values in their input collection.
* The count of an empty collection is defined to be zero(0) & all other aggregate operation return a value of NULL when applied on an empty set.

A	B
1	NULL
2	NULL
NULL	NULL
3	NULL

Count(*) = 4
Count(A) = 3
Count(B) = 0

SUM(A) = 6
AVG(A) = 2
MAX(A) = 3
MIN(A) = 1

SUM(B) = NULL
AVG(B) = NULL
MAX(B) = NULL
MIN(B) = NULL

Group by

Ex:
For each department that has more than 5 employers retrieve the department name & the number of its employees who are making more than 40000

Select Dname, count(*) [redacted]
FROM Department, Employee. WHERE
Dnumber = Dno AND Salary > 40000 AND
Dno IN (Select Dno from EMPLOYEE GROUP BY Dno
Having count(*) > 5) Group by Dname;

* Having is used with Group by only.

[redacted]

Select, from, WHERE, Group, having.

JOINS

#Nat
Normal JOIN
Natural JOIN // No condition
LEFT OUTER JOIN
RIGHT " "
FULL " "
NATURAL LEFT OUTER JOIN
NATURAL RIGHT OUTER JOIN
NATURAL FULL OUTER JOIN

A	B
1	a
2	b
3	c
4	d
5	e
6	f

A	D
1	g
2	h
3	i
7	j
8	k
9	l

select * FROM (R JOIN S ON A=C)

A	B	A	D
1	a	1	g
2	b	2	h
3	c	3	i

~~Natural~~ Normal join

LEF OUTER JOIN

A	B	A	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	NULL	NULL
6	f	NULL	NULL

RIGHT OUTER JOIN

A	B	A	D
1	a	1	g
2	b	2	h
3	c	3	i
NULL	NULL	7	j
NULL	NULL	8	k
"	"	9	l

FULL OUTER JOIN

A	B	A	D
1	a	1	g
2	b	2	h
3	c	3	i
4	d	NULL	NULL
5	e	"	"
6	f	"	"
NULL	NULL	7	j
"	"	8	k
"	"	9	l

Natural LEFT OUTER JOIN
Natural RIGHT OUTER JOIN
Natural FULL OUTER JOIN
} All are same as above concepts but possible only if two table have matching attributes.

Ex
Select Fname, Lname, Address from (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber) WHERE Dname = 'Research';

→ info of employee, working in Research department.

Ex Select Fname, Lname, Address from
(EMPLOYEE NATURAL JOIN DEPARTMENT AS DEPT(Dname, Dno, Mgrssn, MgrstartDate))) WHERE Dname = 'RESEARCH';

```
Select e.Lname As Emp_name, s.Lname AS SUP_NAME
FROM (EMPLOYEE AS e LEFT OUTER JOIN EMPLOYEE AS S
      ON E.Superssn = S.Ssn);
```