Page No.
Date 2/1/2020

① Raw Data — A. Goswami 1962

↓

Field(s) — A. Goswami 1962

Name YOB

↓

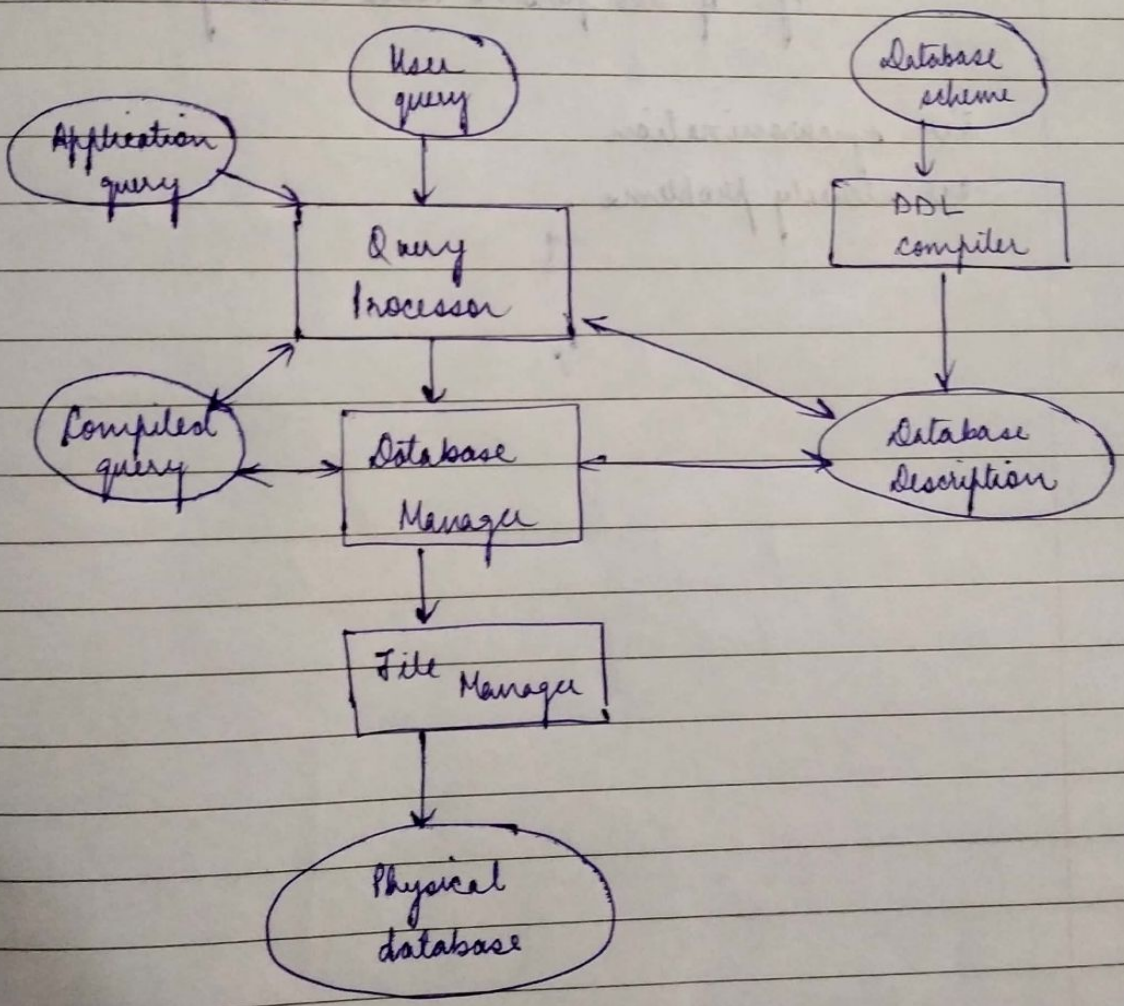
Record

↓

Data file

↓

Database — formed by data files or tables and those files/tables are inter-related i.e some relationship exits among them.



1) Database design
2) Query processing - for manipulating data
3) Physical database

For organisation of data in main memory - data structures
For organisation of data in secondary mem - database management

We use DBMS to remove the following:
i) Data redundancy and inconsistency
ii) Multiple users
eg: if two persons book a single ticket available simultaneously
iii) synchronization
iv) security problems.

Page No. | Date 6 | 1 | 2020

N/w Representation

Player	Match	Age	Run	Nature
Ram	Delhi	22	40	Medium
Hari	Delhi	27	80	Good
Dipu	Delhi	22	30	Bad
Pankaj	Delhi	30	~~76~~	Good
Hari	Bombay	27	30	Bad
Somu	Bombay	31	42	Medium
Kapil	Bombay	30	36	Bad
Ram	Pune	22	60	Medium
Dipu	Pune	22	76	Good
Somu	Pune	31	61	~~good~~ Medium
Hari	Pune	27	77	good.

1) Update — Dipu → Delhi → 22 → 30 to Somu

Now age also changes i.e 22 → 31
i.e Somu → Delhi → 31 → 30

⇒ Simple update needs multiple modification.

2) Insert

Pankaj at Pune made 45 runs

∴ Pankaj | Pune | | 45

⇒ Find age of Pankaj as 30

⇒ Insertion requires more work.

3) Potential Inconsistency

eg: error in age ie say there is another entry of
Hari | Pune | 21 | 77

⇒ The age of Hari from table before is 27

∴ the error is caught.

4) Inability to store certain information.

Reason: Duplicate information.

eg: Let Gyan is started but not played age 23

Player	Match	Runs
Ram	Delhi	40
Hari	Delhi	80
Dipu	Delhi	30
Pankaj	Delhi	76
Hari	Bombay	30
Sonu	Bombay	42
Kapil	Bombay	36
Ram	Pune	60
Dipu	Pune	76
Sonu	Pune	60
Hari	Pune	77

Player	Age
Ram	22
Hari	27
Dipu	22
Pankaj	30
Sonu	31
Kapil	30
Gyan	23

Runs	Nature
40	Med
80	good
30	Bad
76	good
42	Med
36	Bad
60	Med
77	good

* Key = (Player, Match) (Prime attributes)

Partial Dependency → not permitted in a good database design. ⇒ 2nd Normal Form.

i.e part of key (Player) determines age ⇒ Player → Age

Transitive dependency → not permitted ⇒ 3rd normal form.

Q i) What is the age of players who played at Delhi?

⇒ 22, 27, 30

⇒ Join operation: A × C

⇒ Who played good inning?

→ Hari, Pankaj, Dipu, Sonu

3) Age of players who played Bad inning?

⇒ 22, 27, 30

7/1/2020

Who played where

Nature	Player	Place	Age	Runs
good c	Anil	Delhi	22	75
med h	Gyaan	Delhi	24	46
bad g	Ram	Delhi	25	22
med a	gyan	Kanpur	24	46
good i	Yogesh	Kanpur	19	80
bad b	Ram	Mumbai	25	22
med d	Anil	Mumbai	22	46
good e	Gopal	Mumbai	30	80
bad f	Gyan	Ranchi	24	6

(Many - many relation)

Binary Relation

(many - one)

Anil	Delhi
Anil	Mumbai
Gopal	Mumbai
Gyan	Delhi
Gyan	Kanpur
Gyan	Ranchi
Ram	Delhi
Ram	Mumbai
Yogesh	Kanpur

Many - one :

P = {a, b, ... j}

Q = {x, y, z, w, t}

P	Q
a	w
b	w
c	w
d	x
e	z
f	t
g	x
h	z
i	x
j	w
k	z

Owner - set

x	{d, g, i}
y	{}
z	{e, h}
w	{c, a, b, j}
t	{f}

Page No.
Date

~~Storing many-one binary relation:~~

a	b	c	d	e	f	g	h	i	j	k
b	j	a,j	i	h	t	d	z	x	w	e

t	w	x	y	z
f	c	g	y	k

Q (?, z)

z → e → h → z

ans.

i.e. to whom z is related?

Q (a, ?)

a → b → j → w

ans

Q (g, ?)

g → d → i → x

ans

Q (?, y)

y → y

No ans.

Q Now, (?, z)

z → k → e → h → z

ans.

Q (k, ?)

k → e → h → z

ans.

Q (e, ?)

e → h → z

ans.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Data	a	b	c	d	e	f	g	h	i	j			t	w	x	y	z	
Next	1	9	0	8	7	12	3	16	14	13			5	2	6	15	4	

Q (?, x)

x(14) → 6 → 3 → 8 → 14

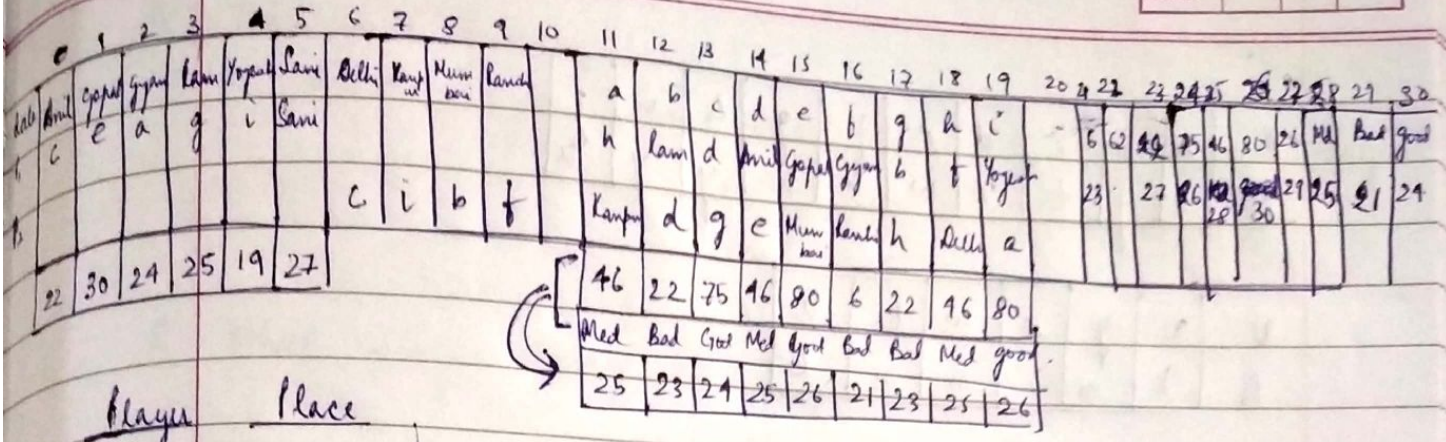
g d i

Q (e, ?)

e(4) → 7 → 16

z

Page No.	
Date	



Player — Place

{c,d} Anil — Delhi {c g h}

{e} Gopal — Kanpur {i a}

{a h f} Gyan — Mumbai {b d e}

{g b} Ram — Ranchi {}

{i} Yogesh

Good = {75, 80}

Med = {46}

Bad {6, 22, 26}

Q Anil played where? & how many runs?

Anil → c (75) → d (46) → Anil

c → g → h → Delhi

d → e → Mumbai

Q Who played at Mumbai?

Mumbai → b → d → e → Mumbai

b → Ram

d → Anil

e → Gopal

Q Who played at Kanpur

Kanpur → i → a → Kanpur

i → Yogesh

a → h → f → Gyan

Q Age of player who played at Mumbai? & how many runs?

Mumbai → b (22) → d (46) → e (80) → Mumbai

b → Ram 25

d → Anil 22

e → Gopal 30

Q What is good?

Good → 24 → 26 → 30

(75) (80)

Q What is Bad?

Bad → 21 → 23 → 27 → 29

(6) (22) (26)

Q Who played & how at Delhi?

Delhi → c (24 → 26 → 30 (Gd)) → g (23 → 27 → 29 (Bad)) → h (25 → 23 (Med)) → Delhi

c → a

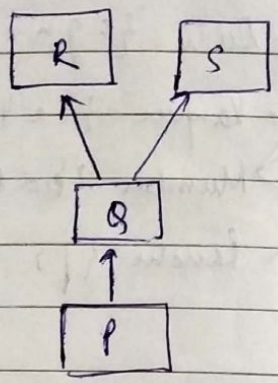
g → b

h → f

P	a	b	c	d	e	f	g	h	i	j
Q	t	y	t	x	w	b	w	y	y	u

Q	x	y	z	w	t	u
R	k	m	k	m	m	l

Q	x	y	z	w	t	u
S	v	x	x	v	x	v



9/1/20

R1

Player	Age
Anil	28
Gyan	26
Hari	22
Jalaj	26
Lalit	22
Mani	24
Dipu	19

R2

Player	Place	Runs
Lalit	Delhi	29
Anil	Delhi	86
Hari	Delhi	56
Hari	Mumbai	32
Jalaj	Mumbai	86
Anil	Kanpur	32
Lalit	Kanpur	76
Anil	Kanpur	12
Jalaj	Kanpur	58

combination of project & select.

Q. Who has age 22?

⇒ Select player from R1 where age = 22 → Hari, Lalit

Q. Who played where?

⇒ Select player, place from R2

Page No.
Date

Q Find runs of every player.

⇒ Select player, sum(runs) from R2 groupwise player.

Lalit 105
Anil 124

Q Where player of age 22 played?

Create Table R1 (Player, Age) Key Player

Create Table R2 (Player, Place, Runs) Key Player, Place, Foreign key player references R1 (player)

Insert into R1 ("Dipu", 19) Correct

Insert into R1 ("Hari", 23) Wrong (two rows same player)

Insert into R2 ("Lalit", "Mumbai", 30) Correct

Insert into R2 ("Jalaj", "Mumbai", 40) Wrong

Insert into R2 ("Anil", "Chennai", 32) Correct

Insert into R2 ("Kapil", "Delhi", 17) Wrong Foreign key player Kapil absent in R1

Q Find age of Hari?

⇒ Select age from R1 where player = "Hari"

Q Who made maximum & how many runs.

⇒ Select player, max(runs) from R2

Q What are maximum runs at all places?

⇒ Select place, max(runs) from R2 groupwise Place.

Q Find ages of players played at Delhi

⇒ Select age from R1, R2 where place = "Delhi" & R1 player = R2 player

Who played where
→ Select Player, place from R2

Project : Who made how many runs → Select Player, Runs from R2

→ Select Player, runs from R2 order by Runs

→ Select Player, runs from R2, order by Runs, Player (Player → Place)

Anil	12
Lalit	29
Hari	32
Anil	32
Hari	56
Jalaj	58
Lalit	76
Anil	80
Jalaj	86

Select: What were half centuries ! (Who, where & how much) → Select * from R2 where Runs ≥ 50

Natural join: Who made how many runs & where & what was his age?

× join → Select * from R1 and R2

Natural join → Select * from R1, R2 where R1.player = R2.player

R1.player	Age	R2.player	Place	Runs
Anil	28	Lalit	Delhi	29
Anil	28	Anil	Delhi	80

54 entries

R1 player	Age	R2 player	Place	Runs
Anil	28	Anil	Delhi	80

Page No.
Date

Q Find max (runs) of every player.

→ Select Max(Runs) from R2 → 86

Select max (runs) from R2 groupwise place

80
86
32
76

Q Who played at age 22?

Select place from R2 where player in (select player from R1, where age = 22)

⇒ Hari Rahil

DATABASE SYSTEMS

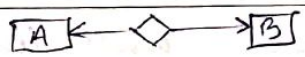
Date 13/1/20
Page

Entity-Relationship Model :- (ER)

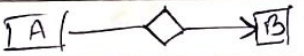
→ Entity : An object that is distinguishable from other objects. Eg : Roll

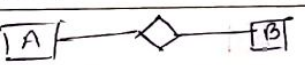
→ Entity set : A group/set of entities. Eg : {Roll, Name, Dept, Hall}

Each entity set has attributes which are the entities

→ Relationships : One-One 

One-Many 

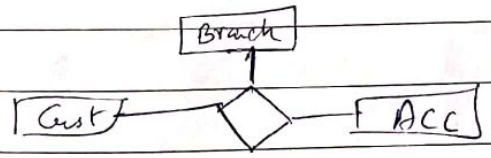
Many-One 

Many-Many 

→ Table : Column : Attributes : Key - ~~attributes~~ uniquely identifies a row.

Key ~~is~~ is a set of attributes.

Row : record.

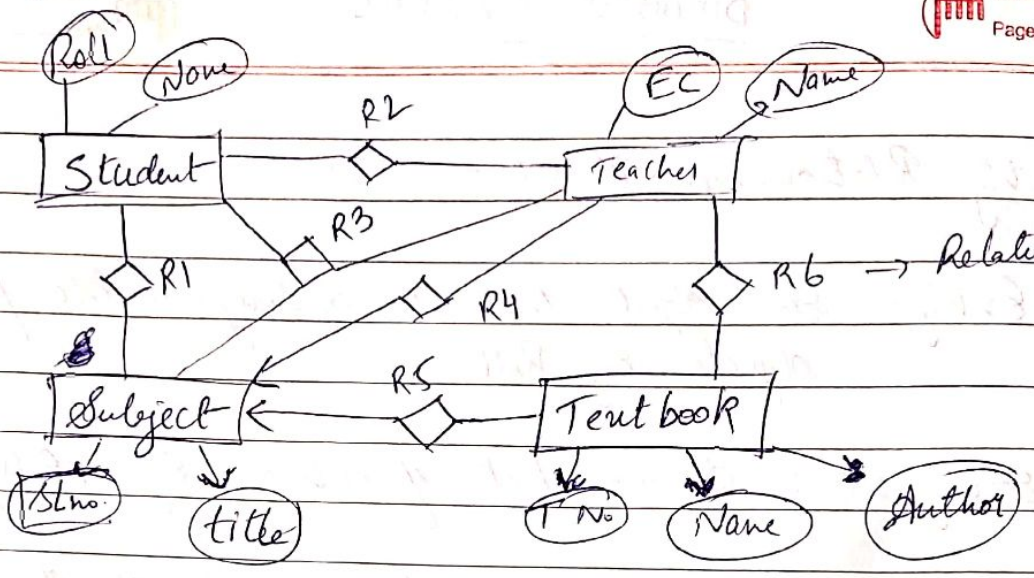
Eg 

Eg: Draw an ER diagram for a database representing textbook, subject, student teacher in an institute where 1) For each subject, each student is taught by single teacher.

2) Each teacher teaches only 1 subject

3) Each subject is taught by several teachers.

4) Teachers choose their respective textbooks.



Student ~~table~~, Teacher, Subject, Text book tables cannot answer which teachers ~~table~~ teach about ~~the~~ a subject.

Hence Relationship tables need to be written.

~~R1~~

R1

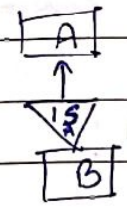
Roll	Sno

R3

Roll	Sno.	EC

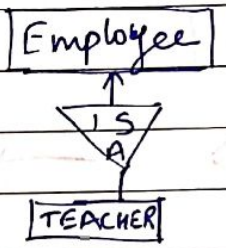
Relational Model : Using tables as entity sets & relations as additional tables.

→ Subset Relation :-

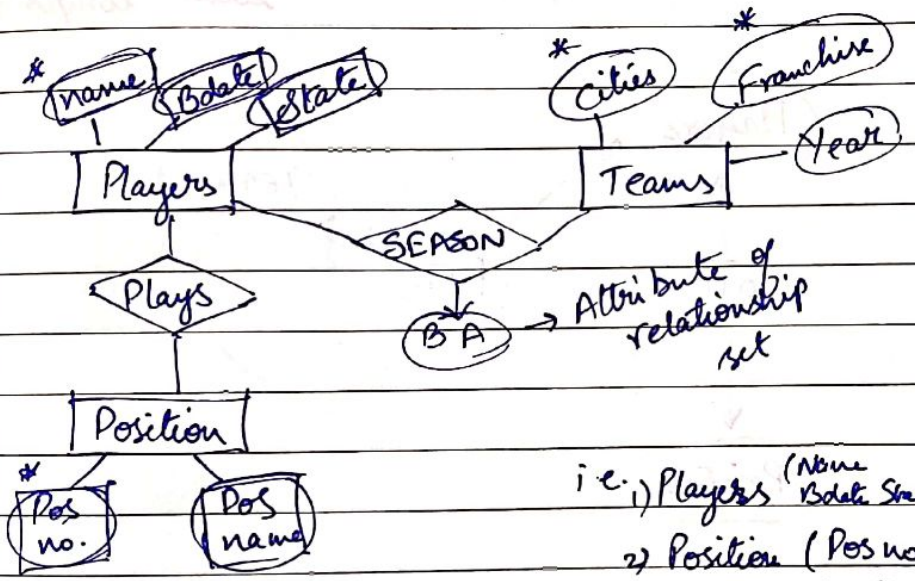


⇒ BCA

Eg:



→



Total no. of tables in corresponding relational model = 5

i.e. 1) Players (Name Bdate State) 3 + 2 relational

2) Position (Pos no., Pos name)

3) Teams (city, franchise, Year)

4) Plays (Name, Pos. No)

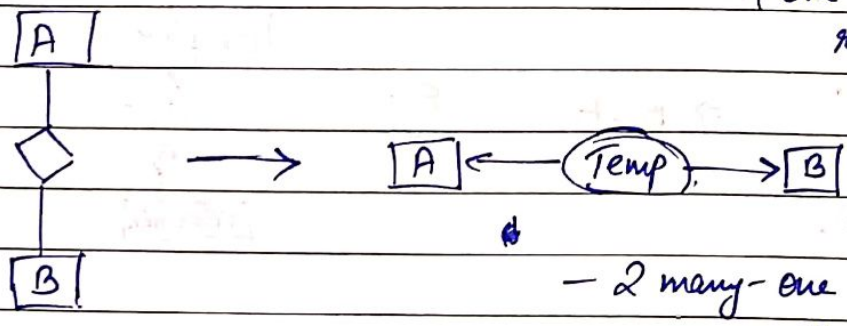
5) Season (Name, city, Franchise, BA)

* Network Model :-

CER model restricted to Binary, Many-one relationship

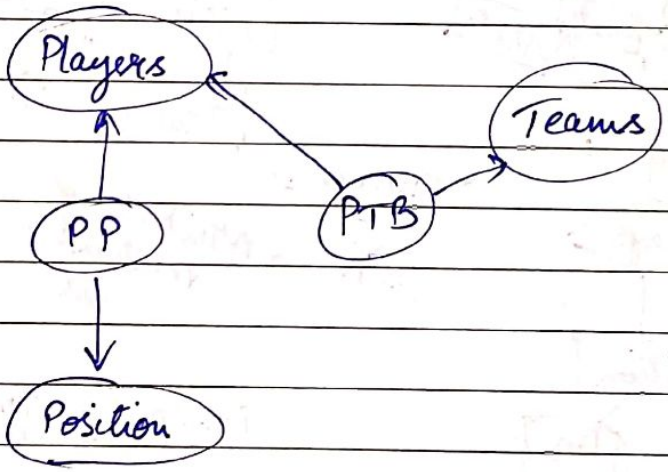
To describe ~~convert~~ many-many relationship in terms of many-one (or) one-one relationships.

(one-one is also a many-one relationship)

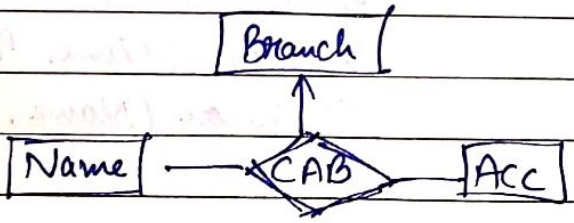


- 2 many-one relationships with temporary set.

Eg:



Eg:



→ 4 tables

~~CAB table~~ :-

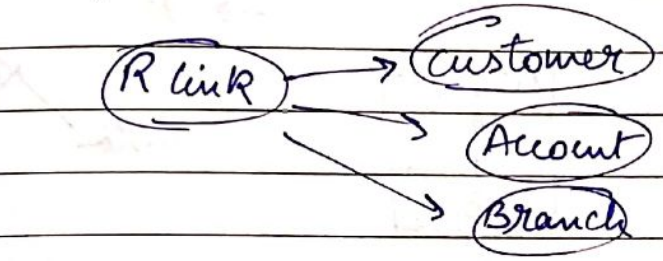
~~B-SC~~

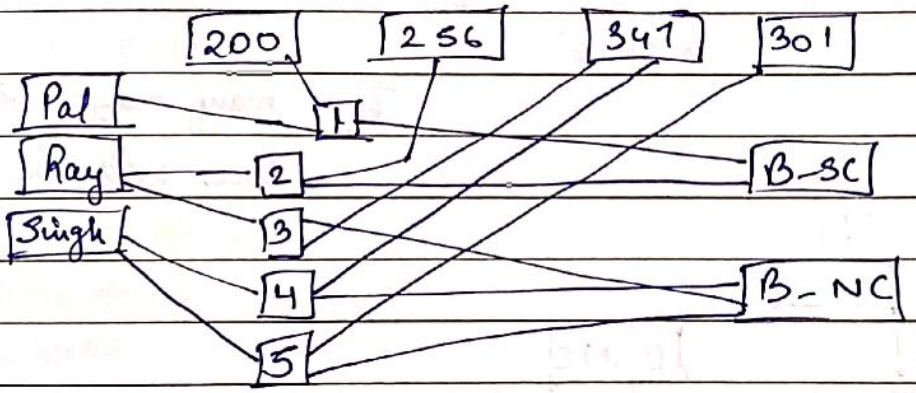
Branch	Cust	Acc
B-SC	Pal	200, 55
B-NC	Ray	256, 1000
	Sigh	347, 667
		301, 5000

CAB table :-

B-SC	Pal	200
B-SC	Ray	256
B-NC	Ray	347
B-NC	Singh	347
B-NC	Singh	301

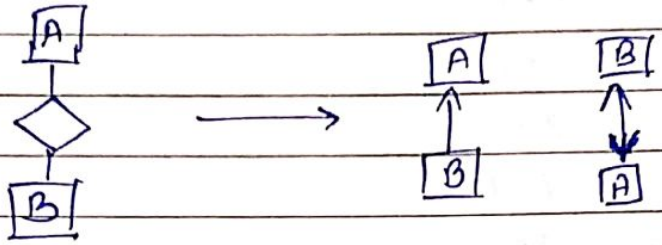
Description in Network Model :-



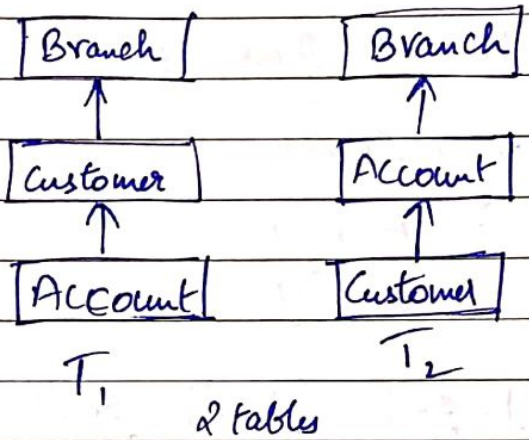


Data used only once so no redundancy. In implementation address of data or pointer to it is stored which is cumbersome

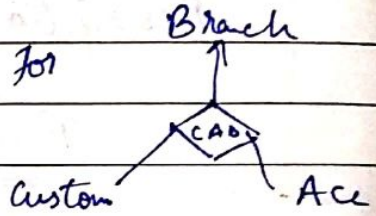
* HIERARCHICAL MODEL :-



Eg:

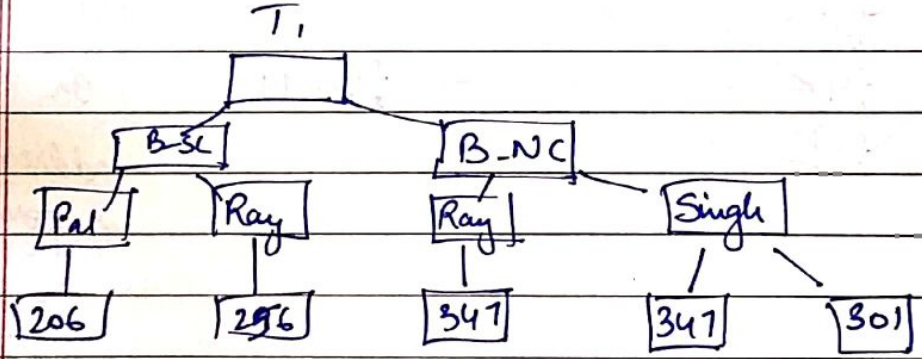


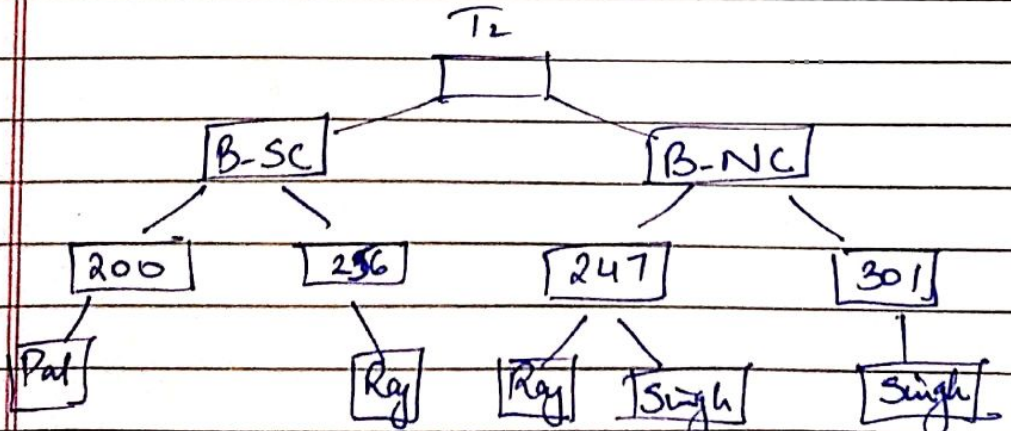
→ for



2 tables

For many-many with Branch there will be 6 tables





→ Query time is lesser than ER model but more than network model.
→ Redundancy exists.

* best
Fixed Database → Network model ~~best~~
Dynamic Database → ER model
↓
Repetition is only in key.

* RELATIONAL DATABASE DESIGN :-

A Good design if :-
Redundancy Anamolies
Updation Anamolies
Insertion Anamolies
Deletion Anamolies

SGP(RN, SNAME, HALL, GAME, FEE)
Redundancy Anamoly : Repetition ~~in~~ of unnecessary data. SNAME, HALL not required when RN is used.
Updation Anamoly : When updation costs are not minimum. missing (or) incorrect updation causes inconsistency
Insertion Anamoly : For a new game, if entry of fee is to be inserted, it is required for a student to play. This should not be necessary.

DBS

Date 16/1/20
Page

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

→ Closure of a set of FDs :-

Let F be a set of FDs the closure of F, denoted by F^+ is the set of FDs That can be logically derived from F.

$$F^+ = \{ X \to Y \mid F \models X \to Y \}$$

RHS logically derived from LHS

Eg: $R(A,B,C)$, $F = \{A \to B, B \to C\}$

$AB \to AB$, $AB \to ABC$

$F^+ = \{ A \to C, AB \to ABC, B \to BC, A \to AB, A \to AC \ldots \}$ (union operation)

→ Armstrongs Axioms :-

1. If $Y \subseteq X \subseteq U$ Then $X \to Y$ - Reflexive property | $X = AB$ $Y = A$ $AB \to A$, $X \to Y$

2. If $X \to Y$ & $Z \subseteq U$ then $XZ \to YZ$ - Augmentation property

3. If $X \to Y$ & $Y \to Z$ then $X \to Z$ - transitivity property

Eg: $R(CITY, ST, PIN)$ (ST → street)

$CITY\ ST \to PIN$

$PIN \to CITY$

~~Whether PIN → ST ?~~

→ $CITY\ ST\ CITY\ ST \to PIN\ CITY\ ST$

→ $CITY\ ST \to PIN\ CITY\ ST$

$CITY\ ST \to$ determines all attributes ∴ Key

→ ~~(illegible)~~ $PIN \to CITY\ PIN$

$PIN\ ST \to CITY\ PIN\ ST$ } all determined → Key

→ Inference Rules :-

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

$X \to YX$ ~~$\to YZ$~~ $XY \to YZ$

$\Rightarrow X \to YZ$

2. $\{X \to Y, WY \to Z\} \models XW \to Z$ (pseudo-transitivity rule)

$XW \to WY \to Z$

3. If $X \to Y$ & $Z \subseteq Y$ then $X \to Z$

4. $\{X \to YZ\} \models X \to Y, X \to Z$

Eg: $R(A, B, C, G, H, I)$

$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}$

$F^+ = \{A \to BC, A \to H, CG \to HI, AG \to HI \ldots\}$

→ Closure of a set of Attributes :-

Let 'F' be a set of FDs. The closure of attribute 'X' denoted by X^+ is the set of attributes 'A' such that $X \to A$ maybe derived from 'F' using Armstrong's axioms.

Lemma: $X \to Y$ follows from Armstrong's axioms if & only if $Y \subseteq X^+$

→ Theorem: Armstrong's axioms are sound & complete

sound: $X \to Y$ from F then if F is true, $X \to Y$ is always true

complete: Cannot derive any extra FD which you cannot derive from Armstrong axioms.

* X^+ Algorithm of Bernstein :

1. Let $N = 0$ & $X(N) = X$
2. $A \to B$ in F where L.H.S 'A' is contained in $X(N)$ and RHS 'B' is not contained in $X(N)$, then make
   $X(N+1) = X(N) \cup B$
   otherwise terminate
3. $N = N+1$, goto step 2

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

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

$(BD)^+ = ?$

$X(0) = BD$

$D \to EG$: $X(1) = BD \cup EG = BDEG$ → (AB not in BDEG)

$BE \to C$: $X(2) = BDEG \cup C = BCDEG$

$C \to A$: $X(3) = BCDEG \cup A = ABCDEG$

$\therefore (BD)^+ = ABCDEG$

↓

Key

classmate
Date 20/1/20
Page

Eg:

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

$A \to C$, $D \to B$ } Non-Trivial FDs

$A \to A$, $B \to B$, $C \to C$, $D \to D$ — Trivial FDs

Ex: $R\{$ FI (Family of Insurance), MOC (Make of car), SAL (salary), SS (spouse salary), FI (city & residence), JOB, COR, LUCK, ED (Education & multifield), EFF (Effort), SEN (sincerity) $\}$ 20/1/20

$F = \{ FI \to MOC,\ SAL\,SS \to FI,\ JOB\,FI \to COR,\ LUCK\,ED \to JOB,\ JOB \to ED,\ EFF\,SEN\,JOB \to SAL \}$

Attributes not appearing at all on right side must be part of Key :

EFF, SEN, LUCK, SS, JOB/ED

* Simplification of FD's :-

- In Non-trivial cases -

→ $X \to Y$, if $\nexists (X' \subset X \ \&\ X' \to Y)$ then it is called elementary FD.

Eg: $AB \to C$
$B \to C$
$A \not\to C$
then $AB \to C$ is an elementary F.D.

in $X \to Y$

→ If $\exists (X' \subset X \ \&\ X' \to Y)$, then there exist partial dependency on the FD $X \to Y$.

Eg: $A(B) \to C$
$A \to C$
→ not necessary
(Extraneous) Extra attribute to determine
This is called partial dependency

Extraneous Attribute :-

if $X \to A$ $\quad B \in X$

if $\exists\ (X - \{B\}) \to A$ is in F^+ $\Rightarrow$ B is extraneous

Eg: $F = \{ AB \to DEF,\ AC \to G,\ A \to C \}$

$(AC)^+ = ACG$

$(A)^+ = ACG$

↓ on removing partial dependency in $AC \to G$

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

$= \{ AB \to DEF,\ A \to GC \}$

→ A FD 'f' is redundant in a set of FD's 'F' if

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

→ Minimum Cover :-

A set of FD's 'F' is minimum is ∄ a set G with lesser no. of FD's to F s.t $G^+ = F^+$.

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

minimum set: $\{A \to BC\}$

→ A set of FD's 'F' is 2-Minimum if :-

1) F is minimum
2) There doesn't exist partial dependency on each FD.

→ A set of FD's is LR-Minimum:-

1) 'F' is α-Minimum
2) No redundancy of the R.H.S.

Eg: $A \to AB$

$A \to B$ — minimum form

Ex: $F = \{ AB \to E,\ AC \to F,\ AD \to B,\ B \to C,\ C \to D \}$

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

$AB \to E$: $(AB)^+$ from all other FD's.

$(AB)^+ = ABCDF$

$\therefore$ Not Redundant

$A^+ = A$

$B^+ = BCD$

$\therefore$ No part of dependency

$AC \to F$: $(AC)^+ = ABCDF$

$\therefore$ Not Redundant

$A^+ = A$

$C^+ = CD$ $\therefore$ No partial dependency

$AD \to B$: $(AD)^+ = AD$

$\therefore$ Not Redundant

$B \to C$: $(B)^+ = B$

$\therefore$ Not Redundant

$C \to D$: $(C)^+ = C$

$\therefore$ Not Redundant

Combining $B \to C,\ C \to D$ to $B \to D$ → loss of Information.

classmate
Date 21/1/20
Page

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

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

21/1/20

1) Rewrite FD's s.t R.H.S has minimum attributes

$AB \to C$: $(AB)^+ = AB$

$C \to A$: $(C)^+ = CA$

$BC \to D$: $(BC)^+ = ABCD$

$ACD \to B$: $(ACD)^+ = ABCDEFG$ Redundant ✗ missing

$D \to EF$: $(D)^+ = D$

$BE \to C$: $(BE)^+ = BE$

$CG \to BD$: $(CG)^+ = A\ C\ \cdot\ \cdot\ G$

$CE \to AG$: $(CE)^+ = A\ C\ E$

2. R min $= \{AB \to C, C \to A, BC \to D, BE \to C, CG \to B, CE \to G\}$

- This is minimum cover.

$D \to E, D \to G$ is not min cover.

$AB \to C$ N.R	$BE \to C$ N.R		
$C \to A$ N.R	$CG \to B$ N.R	$(CG)^+ = CGADE$	
$BC \to D$ N.R	$CG \to D$ N.R	$(CG)^+ = CGBADE$	
$ACD \to B$ R	$CE \to A$ N.R	$(CE)^+ = ACEGBD$	
$D \to E$ N.R	$CE \to G$ N.R	$(CE)^+ = ACE$	
$D \to G$ N.R			

No partial dependency

For $AB \to C$

if $(A)^+ \to C$ or $(B)^+ \to C$ they are extraneous attribute

closure checks $(AB)^+$

classmate
Date
Page

* Decomposition of a Relational scheme:

$R(A_1, A_2, \ldots, A_n)$ is the following,
$P = \{R_1, R_2, \ldots, R_n\}$. such that,
$R = R_1 \cup R_2 \cup \cdots \cup R_n$ (may not be disjoint,

Attributes may ~~not~~ be present in more than 1 sub-relation.

Eg: $R = \{A_1, A_2, A_3\}$
$R_1 = \{A_1, A_2\}$
$R_2 = \{A_3, A_2\}$
$R_1 \cup R_2 = \{A_1, A_2, A_3\}$
$R_1 \cap R_2 = \{A_2\}$
$R_1 - R_2 = \{A_1\}$
$R_2 - R_1 = \{A_3\}$

→ Lossless join Property :-

Suppose 'R' is broken down into $R_1, \ldots R_K$ with set of dependencies 'D'. The decomposition is a lossless join decomposition if

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots\ldots \bowtie \pi_{R_K}(r)$$, where 'r' is set of all the tuples of 'R'.

R_i is a subset of R is terms of rows & columns.

& $\bowtie$ denotes natural join.

Ex:

R_1	R_2
$a_1\ b_1\ c_1$	$b_1\ c_1\ d_1$
$a_1\ b_2\ c_1$	$b_2\ c_1\ d_2$

$R_1 \bowtie R_2$: $a_1\ b_1\ c_1\ d_1$
$a_1\ b_2\ c_1\ d_2$

Theorem: If $P = (R_1, R_2)$ is a decomposition of R, then P is a lossless join decomposition w.r.t FDs if and only if any one of the 2, holds.

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

Ex: $R(A,B,C)$ $F\{A \to B\}$

(i) $R_1 = \{A,B\}$ $R_2 = \{A,C\}$

$R_1 \cap R_2 = \{A\}$

$R_1 - R_2 = \{B\}$

$R_2 - R_1 = \{C\}$

$R_1 \cap R_2 \to R_1 - R_2$

$\therefore$ Lossless join

(ii) $R_1 = \{A,B\}$ $R_2 = \{B,C\}$

$R_1 \cap R_2 = \{B\}$ ✗

Not lossless (No 'B' is the LHS of FDs 'F')

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

$R_1 = \pi_{AB}(R) = \{a_1 b_1, a_2 b_1\}$

$R_2 = \pi_{BC}(R) = \{b_1 c_1, b_1 c_2\}$

$R_1 \bowtie R_2 = \{a_1 b_1 c_1, a_1 b_1 c_2, a_2 b_1 c_1, a_2 b_1 c_2\}$

$\neq R, r$

\# R represents a relation consisting of a set of attributes & tuples.

\# Result is wrong if lossless join property is not satisfied

→ Preservation of Dependencies :-

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

$F \to F_1, F_2, \ldots, F_k$

set of dependencies

If $F_0 = F_1 \cup \ldots \cup F_k$ → Preservation of dependencies property.

\# If this property does not hold, loss of property occurring which may have questions unanswered.

Lossless join Property must always be satisfied even if preservation of dependencies cannot be held.

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

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

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

$R_1 \cap R_2 = \{P\}$ $\quad R_1 \cap R_2 \rightarrow R_2 - R_1$,

$R_2 - R_1 = \{C\}$ $\quad \therefore$ Lossless

$F_1 = \{\} = \phi$ $\quad F_2 = \{P \rightarrow C\}$

$F_1 \cup F_2 = \{P \rightarrow C\} \neq F$

* **Super Key** : The combination of fields by which a row is uniquely identified. If you add any column or attribute to a primary key, then it becomes a super key.

Candidate Key : The minimal super key. These are individual columns in a table that qualifies for uniqueness of all rows.

Foreign Key : It is a field or collection of fields in (one) a table that uniquely identifies a row of another (2nd) table. Thus foreign key is defined in another (2nd) table but refers to the primary key in the 1st table.

Eg: Employee {EmID, FullName, S.S.No. (social security No.), DeptID}

Primary Key : Em.ID or S.S.No → private

Super Key : {EmID, full name} or {Em ID, DeptID} or {EmID, S.S.No}

Candidate Key : Em.ID AND SS.No

→ all columns that can individually identify rows.

T2 (T_name , T_id, E_id, ...)

→ Foreign Key as primary key of Employee

* NORMAL FORMS :-

→ Boyce Codd Normal Form (BCNF) :-

relation ~~FD X → Y~~

A ~~relation~~, is in BCNF if whenever $X \to Y$ & $Y \not\subseteq X$, & X is a key.

Eg: R { PAN, PI, DI, DRUG, QTY, COST }

F = { PAN → PI , PI → DI , PI DRUG → QTY,
DRUG QTY → COST } whether in BCNF?

Key : (PAN, DRUG) PAN → PI not in BCNF

Hence whole relation is not in BCNF

Decomposing R to satisfy BCNF :-

PAN → PI : R_1 (PAN, PI) (attributes of FD)

All except RHS attrib → R_2 (PAN, DI, DRUG, QTY, COST)

R_1 in BCNF? Yes

R_2 = { PI DRUG → QTY, DRUG QTY → COST }

DRUG QTY → COST not in BCNF

R_3 : {DRUG, QTY, COST} BCNF

~~DRUG QT~~

DRUG QTY is key.

R_4 : { PAN, DI, DRUG, QTY)

PAN → DI

R_5 : { PAN, DI } BCNF

R_6 { PAN, DRUG, ~~QTY~~ }

PAN DRUG → QTY

Date 23/1/20
Page

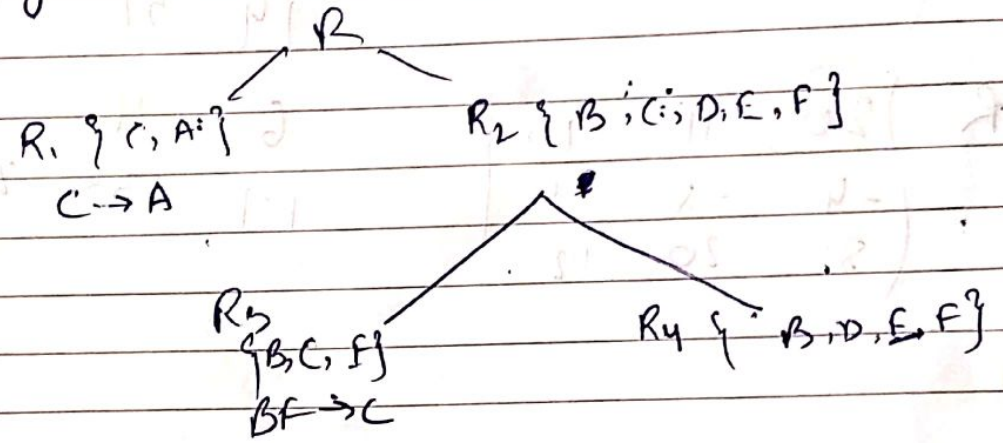
Theorem:- BCNF Decomposition always produces lossless join decomposition.

BCNF
Easy to decompose but loss of FD's may occur. BCNF forms are not unique.

Ex: R(A, B, C, D, E, F) (F.D.)

F = {C → A, AE → B, BF → C, CD → EF, EF → AD}

Key: CD (or) EF



* Prime attribute :-

An attribute which is a member or subset of a key - otherwise it is known as non-prime attribute.
(minimal key)

A R(A, B, C, D)

F: {AB → C, B → D, BC → A}

Key = AB, BC

prime attributes : ABC

Non-prime " : D

Date __/__/____
Page _____

* Third Normal Form :- (3NF)

$X \to A$ is in 3NF if $A \notin X$ and either X is a key of relation or A is a prime attribute.

BCNF → 3NF

Removes Partial dependency & Transitive dependency.
$(A \to B \to C)$
$A \to C$

→ Bernstein's 3NF Algorithm :-

1. Rewrite the FDs such that each FD contains only 1 attribute on RHS
2. Rewrite the list of FD's s.t ∄ redundant FD's.
3. Rewrite FDs such that no proper subset of LHS functionally determines R.H.S. (ensures no partial dependency)
4. Combine dependencies with same LHS using union rule. Then if $X \to Y$ is an F.D, make a decomposition X, Y.
5. If Key is not present in any of the sub relation, then create a new subrelation with Key only.
6. If any subrelation is subset of other subrelation, remove it.

Ex: $R_1(PAN, PI)$ $R_2(PI, DI)$ $R_3(PI, DRUG, QTY)$, $R_4(DRUG, QTY, COST)$, $R_5(PAN, DRUG)$

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

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

Key : ~~AE~~ F, AE

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

~~$AE \to F$~~, ~~$AE \to G$~~

$F \to A$, $F \to E$, $F \to G$

$C \to H$, $C \to I$

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

$\{A, B, C, D\}$ $\{A, E, F, G\}$ $\{C, H, I\}$

classmate
Date 27/1/20
Page

* Multi-Valued Dependency :-

$X \rightarrow\rightarrow Y$

"X multi-determines Y"

'X' multi-determines 'Y' holds in a relation if we have tuples 't's' whose X values are same, and there exist 2 tuples 'u' & 'v' such that

Subject	Teacher	
MATH I	T1	B1
	T2	B2
	T3	B3
	⋮	⋮
PBS	T11	B11
	T12	B12

1. $u[X] = v[X] = t[X] = s[X]$
2. $u[Y] = t[Y]$, $u[R-X-Y] = s[R-X-Y]$
3. $v[Y] = s[Y]$, $v[R-X-Y] = t[R-X-Y]$

i.e

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

M_1	T_1	B_1
M_1	T_1	B_2
M_1	T_2	B_1
M_1	T_2	B_2
M_1	T_3	B_1
M_1	T_3	B_2
M_1		

→ X, Y, Z = R-X-Y, then

$X \rightarrow\rightarrow Y \Rightarrow X \rightarrow\rightarrow Z$

→ X,Y,Z then $\{X \rightarrow\rightarrow Y, Y \rightarrow\rightarrow Z\} \vdash X \rightarrow\rightarrow (Z-Y)$

→ X, Y : $X \rightarrow Y \Rightarrow X \rightarrow\rightarrow Y$

* 4NF

"$X \to\to Y$" is in 4NF if X is the key of the relation

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

F: { $A \to\to B$, $B \to\to G$, $B \to\to EF$, $CD \to E$ }

~~$A \to\to (G - B = G)$~~ $A \to\to B, B \to\to G \Rightarrow A \to\to G$

Key = ~~EEF~~ ACD

$A \to\to EF$

$A \to\to F$

Decomposition :-

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

$CD \to E \not\models CD \to\to EF$

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

$A \to\to G$

$R_5(A, G)$ $\quad$ $R_6(A, C, D, F)$

$A \to\to F$

$R_7(A, F)$ $\quad$ $R_8(A, C, D)$ → Only key ⇒ No FD, MD in 4NF

R_1, R_3, R_5, R_7, R_8

QUERY LANGUAGES

RELATIONAL ALGERA :-

* BASIC OPERATORS :-

1. Union (U) :-

R U S : Set of tuples either in R or S or both.
R, S must have same no. of attributes

Ex:

R1

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

S1

D	E	F
b	g	a
d	a	f

R1 U S1

col1	col2	col3
a	b	c
d	a	f
c	b	d
b	g	a
~~d~~	a	f

if not allowed more

T

B	A	C
b	a	c

RUT

B		
a	b	c
d	a	f
c	b	d
b	a	c

2. Set Difference (−) :-

R − S : Set of tuples present in R but not in S.
'R', 'S' must have same no. of attributes.

Ex: R1 − S1

a	b	c
c	b	d

3. Cartesian Product (X) :-

Let R, S be relations with arity K_1, K_2. (No. of attributes / cardinality)

R×S : Set of tuples with arity $K_1 + K_2$
where 1st K_1 components form a tuple of R
& last K_2 components form a tuple of S.

Eg :

R1 x S1

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

4. Projection (π) :-

$\pi_{i_1, i_2, i_3, \ldots, i_K}(R)$: Set of tuples with attributes $i_1, \ldots i_K$ with content same as that of the attributes of tuples in R.

i_K : attribute name/no.

$\pi_{C,A}(R1)$

C	A
c	a
f	d
d	c

$\pi_{2,3}(S1)$

E	F
g	a
a	f

5. Selection (σ) :-

F → $>, <, <>, \ldots, \vee, \wedge, \neg$
↳ Formula

$\sigma_F(R)$: set of tuples satisfying formula 'F'.

Ex: $\sigma_{B='b'}(R1)$

A	B	C
a	b	c
c	b	d

$\sigma_{2>3}(R1)$: Tuples with attribute 2 value > attribute 3 value

A	B	C

$\sigma_{1='a' \vee 1='b'}(R1)$

A	B	C
a	b	c

* ADDITIONAL OPERATORS :-

1. Intersection ($\cap$) :-

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

Ex: $R \cap S$

A	B	C
d	a	f

2. Quotient ($\div$) :-

R, S s.t $n > s$
(arity)

$R \div S$: set of tuples 't' with arity $(n-s)$ s.t for all tuples 'u' in S, the tuple tu is in R

Ex:

R2			
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
e	d	e	f
a	b	d	e

S2	
c	d
e	f

R2 ÷ S2	
a	b
e	d

Not "b c" as "b c c d" ∉ R2

$$R \div S = \pi_{1,2,\ldots,r-s}(R) - \pi_{1,2,\ldots,r-s}((\pi_{1,2,\ldots,r-s}(R) \times S) - R)$$

3. θ- join :

$R \bowtie_{i\,\theta\,j} S$: Set of tuples that is the subset of $R \times S$ satisfying "i θ j".

i θ j
- An arithmetic comparision operator
- Attributes of R, S respectively

Ex:

R3

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

S3

D	E
3	1
6	2

$R3 \bowtie_{B<D} S3$

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

4. Natural Join ($\bowtie$) :-

$R \bowtie S$: Subset of $R \times S$ in which only those tuples with same values for same attributes in R, S are present with same attributes represented in a single column.

must have attribute names

R

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

S

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

$R \bowtie S$

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

→ DATABASE & QUERY EXAMPLE :-

Ex1.

Customer (c_name, street, c_city)
Deposit (b_name, ac_no, c_name, balance)
Branch (b_name, b_city, asset)
Borrow (b_name, loan_no, c_name, amount)
Client (c_name, emp_name)

1. Find the customer names, where the customer name & the name of the employee handling the customer are same.

(From Client table)

$$\Pi_{c\_name}\left[\sigma_{\substack{c\_name \\ = emp\_name}}(Client)\right]$$

2. Find all clients of employee XYZ and corresponding cities of those clients.

$$\Pi_{\text{client.cname, c\_city}} \left[\sigma_{\substack{\text{emp\_name} = \text{'XYZ'} \\ \wedge\ \text{client.cname} = \text{customer.cname}}} (\text{client} \times \text{customer}) \right]$$

(or customer)

3. Find all customers who have both account & loan at KGP branch.

$$\Pi_{\text{c\_name}} \left[\sigma_{\text{b\_name} = \text{'KGP'}} (\text{deposit} \bowtie \text{borrow}) \right]$$

↓ only 1 cname in natural join

Using θ join ?

4. Find all customers who have account at all branches in the city Kolkata.

$$\left(\Pi_{\text{c\_name, b\_name}} (\text{Deposit}) \div \left(\Pi_{\text{b\_name}} \left(\sigma_{\text{b\_city} = \text{Kolkata}} (\text{Branch}) \right) \right) \right)$$

classmate
Date 30/1/20
Page

Ex2.

Employee (Emp_ID, Name, Salary, Manager_ID)

↘ = 1 => manager

Display the name of employees (not managers) whose salary is more than any manager.

$$\Pi_2 \left[\sigma_{3>7} \left[\sigma_{(4\neq 1 \wedge 8=1)} (\text{Employee} \times \text{Employee}) \right] \right]$$

$4\neq 1 \wedge 8=1 \wedge 3>7$

Emp Emp

1	2	3	4	5	6	7	8
			≠1				=1

Ex3. Lives (P_name, street, city) P_name Key

Works (P_name, C_name, salary)

Located_in (C_name, C_city)

Manager (P_name, m_name)

1. Find the name & city of all persons who work for 'XYZ' company.

$$\Pi_{P\_name, city} \left[\Pi_{1,2} \left(\sigma_{C\_name = 'XYZ'} (\text{Works}) \right) \bowtie \text{Lives} \right]$$

Lives ~~located_in~~

↘ Left column remains.

2. Find all persons who lives in the same city ~~and~~ as the company they work for.

$$\Pi_1 \left[\sigma_{3=6} (\text{Lives} \bowtie \text{works} \bowtie \text{Located\_in}) \right]$$

1 2 3 4 5 6

P_name street city C_name Salary C_city

3. Find all persons who live in the same city & same street as their manager

$$\pi_1 \left[\sigma_{4=5 \wedge 3=7 \wedge 2=6} \left[(\text{lives} \bowtie \text{Manager}) \times \text{lives} \right] \right]$$

4. Find all persons whose salary is more than every employee of 'XYZ' company.

3/2/20

~~(stolen)~~

$$\pi_1 \left(\sigma_{5='XYZ',\ 3>6} (\text{Works} \times \text{Works}) \right) = T$$

$$Q = \pi_1 \left(\sigma_{1<2} (T \times T) \right)$$

$$\text{Ans} = T - Q$$

Assgn: 3NF decomposition

classmate
Date 3/2/20
Page

* PROPOSITIONAL LOGIC

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

$\Rightarrow$ (implication) , $\equiv$ (equivalence iff)

→ Proposition is a declarative statement that will be either true or false.

→ A proposition is made up of certain symbols and atoms

→ Any expression which is represented by that contains propositions is called formula.

Tuple Relational Calculus ~~Algebra~~ :-

An expression in Tuple relational Calculus is written as

$$\{ t \mid \varphi(t) \}$$

t : tuple variable ← resultant
$\varphi(t)$: tuple formula satisfying

→ R(s) : 's' is a tuple of the relation 'R'.
s[i] : value of i^{th} attribute of tuple s.

s[i] Θ u[j] : i^{th} component of tuple 's' is in relation to j^{th} component of tuple 'u'

→ Universal Quantifier :-

$(\forall x)$

$(\forall x)\ G(x)$: 'G' is true (formula) for all 'x'.

→ Existential Quantifier :-

$(\exists x)$

$(\exists x)\ G(x)$: There exists atleast one value of 'x' for which G is true

$(\exists s)\ (R(s))$: There is a tuple in Relation R
Relation R is non-empty

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

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

$R^{(r)} \times S^{(s)}$: $\{ t^{(r+s)} \mid (\exists u)(\exists v)\ (R(u) \wedge S(v) \wedge t[1] = u[1] \wedge \dots \wedge t[r] = u[r] \wedge t[r+1] = v[1] \wedge \dots \wedge t[r+s] = v[s] \}$

$\Pi_{i_1, i_2, \dots, i_k}(R)$: $\{ t^{(k)} \mid (\exists u)\ (R(u) \wedge t[1] = u[i_1] \wedge \dots \wedge t[k] = u[i_k]) \}$

$\sigma_F(R)$: $\{ t \mid R(t) \wedge F' \}$

Relation Algebra form $\Longleftrightarrow$ Tuple Calculus Form

classmate
Date 4/2/20
Page

$R^{(2)}$, $S^{(2)}$

$\pi_{1,4}(\sigma_{2=3}(R \times S)) : \{ t^{(2)} \mid (\exists u)(\exists v) (R(u) \wedge S(v) \wedge u[2] = v[1] \wedge t[1] = u[1] \wedge t[2] = v[2]) \}$

Ex: Write Banking database's example queries in Tuple Calculus.

4/2/20

Ex: BANKING DB

1) Find the branch name, loan no., cust. name and amount for loans more than 10,000.

Borrow: R

$\{ t \mid R(t) \wedge t[4] > 10000 \}$

2) Find the cust. names whose amount is greater than 10000.

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

3) Find all customers having a loan at IIT branch and the cities where they live.

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

$\downarrow \forall \Rightarrow$

$\exists u (u \in Borrow \wedge u[1] = 'IIT' \wedge \exists v (v \in Customers \wedge u[3] = v[1] \wedge t[1] = v[1] \wedge t[2] = v[3]))$

Time & Space complexity lesser.

7th | 8th March

4) Find all customers having a loan or account ~~or both~~ at IIT branch.

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

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

$\wedge$: For both loan & account

5) Find all customers who have account at all branches in the city KGP.

$\forall \rightarrow \exists$ → any

$$\{ t \mid \exists u (u \in Branch \wedge u[2] \neq 'KGP') \vee (\exists v)(v \in Deposit \wedge u[1] = v[1] \wedge t[1] = v[1]) \}$$

STRUCTURED QUERY LANGUAGE

→ Creating database, tables, constraints in attributes, Key types. Commands not to be discussed.

→ SQL based on Relational Algebra.

→ STANDARD FORM :-

SELECT $A_1, A_2 \ldots, A_n$ FROM $R_1, R_2 \ldots, R_m$ WHERE P ; → necessary for system omitted henceforth

optional

A_i : Attribute R_i : Relation P: Predicate / Formula

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

Eg: Find the names of all branches in Deposit Relation.

select b_name from deposit ;

- select distinct b_name from deposit ;
  → to remove duplicates

→

Keyword	relational algebra
union	$\cup$
intersection	$\cap$
minus	-
and	$\wedge$
or	$\cup$
not	$\neg$

Ex: Find all customers having an account at IIT branch

```
select c_name
from Deposit
where b_name = 'IIT'
```

Ex: Find all customers having a loan & account or both at IIT branch.

from Deposit, Borrow → Deposit D, Borrow B

where ~~deposit.c_name = Borrow.c_name~~ ~~deposit.b_name~~ Borrow.b_name = 'IIT'

∧ deposit.b_name = 'IIT'

```
select deposit.c_name
(select c_name
from Deposit
where b_name = 'IIT')
union
(select c_name
from Borrow
where b_name = 'IIT');
```

Alias : from Deposit D, Borrow B
where D.c_name = 'SATVIK'

Ex. Find all customers having a loan & the city where they stay.

```
select  B.C_name , C-city
from    Borrow B, customer C
where   B.C_name = C.C_name
```

Ex: Find all customers who have both loan & account at IIT branch.

```
select D.C_name
From   Borrow B , Deposit D
where  B.b_name = D.b_name ∧ B.C name = D.C_name
       ∧ B.b_name = IIT
```

```
select  C_name                          } select 'a' C_name
from    Borrow                          } and
where   b_name = 'IIT'                  }
and  C_name in                          }
( select  C name                        } check if C_name is
  from    Deposit                       } in this "set".
  where   b_name = 'IIT' )              }
```

"and C_name in .. (- - -) " is another condition.

---

customer (C_name, street, C_city)
deposit (b name, ac_no, C_name, balance)
branch (b_name, b_city, asset)
borrow (b_name , loan_no, C_name, amount)
client (C_name , emp_name)

classmate
Date 6/2/20
Page

Ex: Find all customers who have account at IIT branch but no loan in IIT branch.

```
select c_name
from deposit
where b_name = "IIT"
and ~~minus~~ c_name ~~in~~ not in
(select c_name
 from borrow
 where b_name = "IIT")
```

Ex: Find all customers who have account at some branch where "XYZ" have an account.

```
select c_name
from deposit
where b_name in
(select b_name
 from deposit
 where c_name = "XYZ")
```

```
select S.c_name
from deposit S, deposit T
where S.b_name = T.b_name
and T.c_name = 'XYZ'.
```

Ex: Find all branches that have more assets than any branch located in city KGP

```
select b_name
from Branch
where asset > any  → single word of X
(select asset
 from branch
 where b_city = 'KGP')
```

> any
> all

Ex: Find all customers who have account at all branches at city KGP.

```
select C_name
From Deposit S
where ~~branch~~
(select b_name
 from Deposit T
 where S.C_name = T.C_name)
contains ——————→ for set in set
                  "in" for element in set
( select b_name
  from Branch
  where b_city = 'KGP')
```

Ex: Find all customers whose balance is in between 50000 & 60000.

```
select C_name
from deposit
where balance → between 50000 and 60000
                (or)
              >= 50000 and <= 60000
```

Ex: Find all customers whose city includes the substring 'KAL'

```
select C_name
from customer
where C_city like "%KAL%."
```

"KAL%." : start with KAL

"___" : exactly 3 characters

"___%." : atleast 3 characters

Ex: List in alphabetical order all the customers having a loan in IIT branch

```
select c_name
from borrow
where b_name = "IIT"
order by c_name ;
```

/d : opposite order

operators : max
min
avg
sum
count

Ex: Find the avg account balance for all branches.

```
select b_name, avg(balance)
from deposit
group by b_name
```

} only these 2 gives all balance avg.

(and) having avg(balance) > 50000

→ (grouped by b_name)

classmate
Date 10/2/20
Page

Ex: Find the no. of records in customer relation

```
select count(*)
from customer
```

Ex: Find all customers who have a deposit at IIT branch but for whom there is no entry in customer relation.

```
select c_name
from deposit
where b_name = 'IIT' and c_name not in
    ( select c_name
      from customer)
```

```
select c_name → (D.c_name)
from Deposit D
where b_name = 'IIT'
and not exists
(select *
 from customer C
 where D.c_name = C.c_name)
```

set A

not exists → true
if A = ϕ
→ false
if A has some tuple.
It will have tuples only if selected c_name is there in customer.

Commands

* Table ~~Creation~~ :-

Syntax : → CREATE TABLE name
→ DROP TABLE name
→ INSERT INTO name
VALUES (- - - -)
→ DELETE FROM

Ex:-

Teacher (t_name, dept, t_no, s_title)
Student (s_name, course, hall)
Study (s_title , s_name, level, status, marks)

1) Find the students of CSDP course who study DBMS as a core subject and find their marks.

```
select S1.s_name , marks
from Student S1 , Study S2
where S1.s_name = S2.s_name and course = "CSDP"
      and s_title = 'DBMS' and status = 'core'
```

2) Find students of CSDP course living in VS Hall or SN Hall who study no subject taught by prof XYZ

```
select S1.s_name
from Teacher T , Studat S1, Study S2
where (hall = VS or hall = SN) and S1.s_name = S1.s_name
and course = CSDP and S2.s_title not in
    ( select s_title
      from teacher
      where t.name = XYZ)
```

```
select S_name
from student
where course = "CSDP" & (hall = VS or hall = SN)
& S_name not in
( select S_name
  from Study S, Teach T
  where S.S_title = T.S_title & t_name = XYZ')
```

3) Find the avg marks in the subjects taught by XYZ.

```
S.S_title
select, avg(marks)
from study S, Teacher T
where t-name = 'XYZ'
    and S.s-title = T.s_title
group by S.s_title
```

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

```
select distinct s_title
from study
group by s_title
having min(marks) > 80
```

5) Find the name of students who secured equal marks in 2 different subjects.

```
select
From study S1, study S2
```

QUEL

QUEL

→ Used in Ingrace Database system

→ Uses tuple relational calculus

→ Syntax :-

range of t_1 is R_1
range of t_2 is R_2
⋮
range of t_n is R_n
retrieve ($t_1.A_{j_1}, t_{i_2}.A_{j_2}, \ldots, t_{i_m}.A_{j_m}$)
where P;

t_i is the tuple variable of relation R_i
$t.A$ represents attribute A of tuple variable t
P is the predicate (or) condition

$$\{ t \mid \exists t_1, t_2, \ldots, t_n (t_1 \in R_1 \wedge t_2 \in R_2 \wedge \ldots t_n \in R_n \wedge P \wedge t[1] = t_{i_1}[A_{j_1}] \wedge \ldots \wedge t[m] = t_{i_m}[A_{j_m}]) \}$$

Ex1: Find all customers having an account at IIT branch.

```
range of t is Deposit
retrieve (t.c_name)
where t.b_name = 'IIT' ;
```

Ex2: Find all customers having a loan at IIT branch and their cities

```
range of b is Borrow
range of c is customer
retrieve (b.c_name, c.c_city)
where b.c_name = c.c_name and b.b_name = 'IIT'
```

Ex3: Find all customers who have loan and account at IIT branch.

```
range of b is Borrow
range of d is deposit
retrieve (b.c_name)
where b.c_name = d.c_name and b.b_name = 'IIT'
      and d.b_name = 'IIT'
```

Ex 4: Find all customers who have account or loan or both at IIT branch.

~~range of b is Borrow~~
range of d is deposit
retrieve into temp (d.c_name)
where d.b_name = 'IIT'
range of d is Borrow
append to temp (d.c_name)
where d.b_name = 'IIT'
range of d is temp
retrieve unique (d.c_name) ;

temp : temporary table name

QUEL : Nesting or Nested Queries not possible (SQL possible)

Ex: 5 Find all cust. having account but no loan at IIT branch.

range of t is deposit
retrive into temp (t.c_name)
where t.b_name = 'IIT'
range of t1 is borrow
range of t2 is temp
delete (t2)
where t1.c_name = t2.c_name and
t1.b_name = 'IIT'
range of t is temp
retrieve (t.c_name) ;

Ex6: Display customer names in alphabetical order who have account at IIT branch.

```
range of t is deposit
retrieve (t.cname)
where t.b_name = 'IIT'
sort by t.cname
```

Ex7: Find the avg account balance at IIT branch.

```
range of t is deposit
retrieve avg(t.balance where t.b_name = 'IIT')
```

operator (t.A where P)

Ex8: Find all accounts whose balance is higher than the avg balance at that branch.

```
range of t is deposit
retrieve (t.ac_no)
where t.balance > avg(t.balance by t.b_name)
```

Ex9: Find all customers who have account at all branches in the city 'KGP'.

```
range of t in deposit
range of u in branch
range of s in deposit
retrive (t.cname)
where
```

* = countu(u.b_name where u.b_city = 'KGP')

count unique

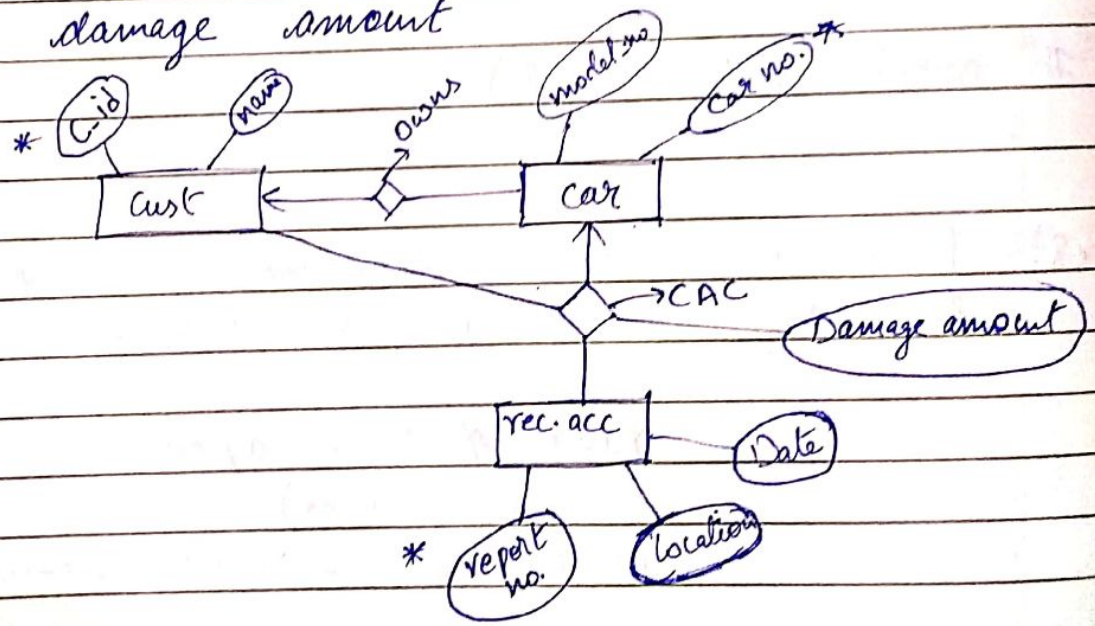
countu(s.b_name where s.b_name = u.b_name and u.b_city = 'KGP' and s.c_name = t.c_name) *

Ex10. Find all subjects such that all students studying the subject secure more than 80 marks.

range of t is study
retrieve (t.s-title)
where min (t.marks by t.s-title) > 80

13/2/20

Ex: Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each car is associated with '0' to any number of recorded accidents. Each accident is associated with some damage amount



CAC: an accident may be associated with 1 or more customers / cars.

classmate
Date
Page

Relational Model:

Cust	car	accident	owns	CAC
			C_id	C_id
			car no.	car no.
				report no.
				damage amt

Ex: $R(A, B, C, D, E)$

F_1 { $A \to B$, $AB \to C$, $D \to AC$, $D \to E$ }

F_2 { $A \to BC$, $D \to AE$ }

Is $F_1 = F_2$?

$\because F_1^+ \cdot F_2^+$ → closure $\Rightarrow F_1 = F_2$

$D \to C \in F_1$ $\quad$ $D \to C \in F_2^+$

Ex: $R(A, B, C, D, E)$

F: { $A \to BC$, $CD \to E$, $B \to D$, $E \to A$ }

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

$R_1 \cap R_2 = A$ $\quad$ $F \Rightarrow A \to BC$

$R_1 - R_2 = BC$

$R_2 - R_1 = DE$ $\quad$ $\therefore$ Loss less join property

$CD \to E$ & $B \to D$ are lost.

$\therefore$ Preservation of FD's not held.

Ex: $X \rightarrow Y$, $W \rightarrow Z$, $Y \subseteq W$

is $X \rightarrow Z$?

$Y \subseteq W \rightarrow \quad W \rightarrow Y$

↓

(W is a set of Attributes like XY , YZ, XYZW etc)

False