Books

- Principles of Database systems - J. D. Ulman
- Database System concepts - H. F Karth & A. Silbar
- DBMS - Rama Krishnan

Raw Data - set of characters / alpha-numeric numbers
↓ Processing and labelling
Fields (Name, dept, Roll no, etc)
↓ collection of fields
Record
↓ collection of similar type of Records
Data File
↓ collection of inter-related data Files
Database

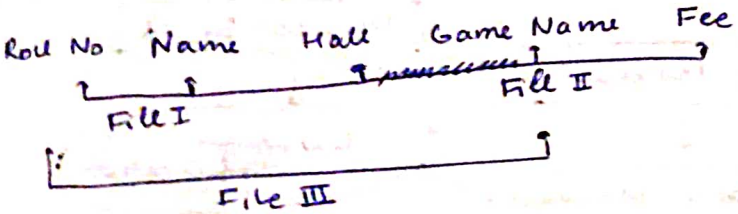
DBMS - Software that helps to create database and ~~manage datafiles and~~ manipulate it

- Speed
- accuracy

fast and accurate way to modify data

Advantages / Need of database when high-level languages C, C++, Python already exists

1. Data Redundancy and Inconsistency
   ↓
   Duplicate data



Game_Name, Fee, Hall, Name all will be repeated
Is this duplicacy required ?
Can we manipulate this table itself or to break this into different tables

When breaking the File, the original properties should remain intact

Even now, game name and Roll number repeated
This will be removed if there exists any connection

Designing database - How will I group data with n fields into different data files to remove duplicacy, while being consistent (same changes being applied to all similar files)

2. Easy Access of Data

Once database is created, the user need not know the underlying structure (data types, etc) of the database.

3. Multiples users can access the data base simultaneously.

4. It takes care of security problems

Admin can always decide which block of data is accessible to which users

5. Integrity problems

We can incorporate constraints in the DBMS at the time of creation. (e.g. lower bills to senior citizens, ladies)

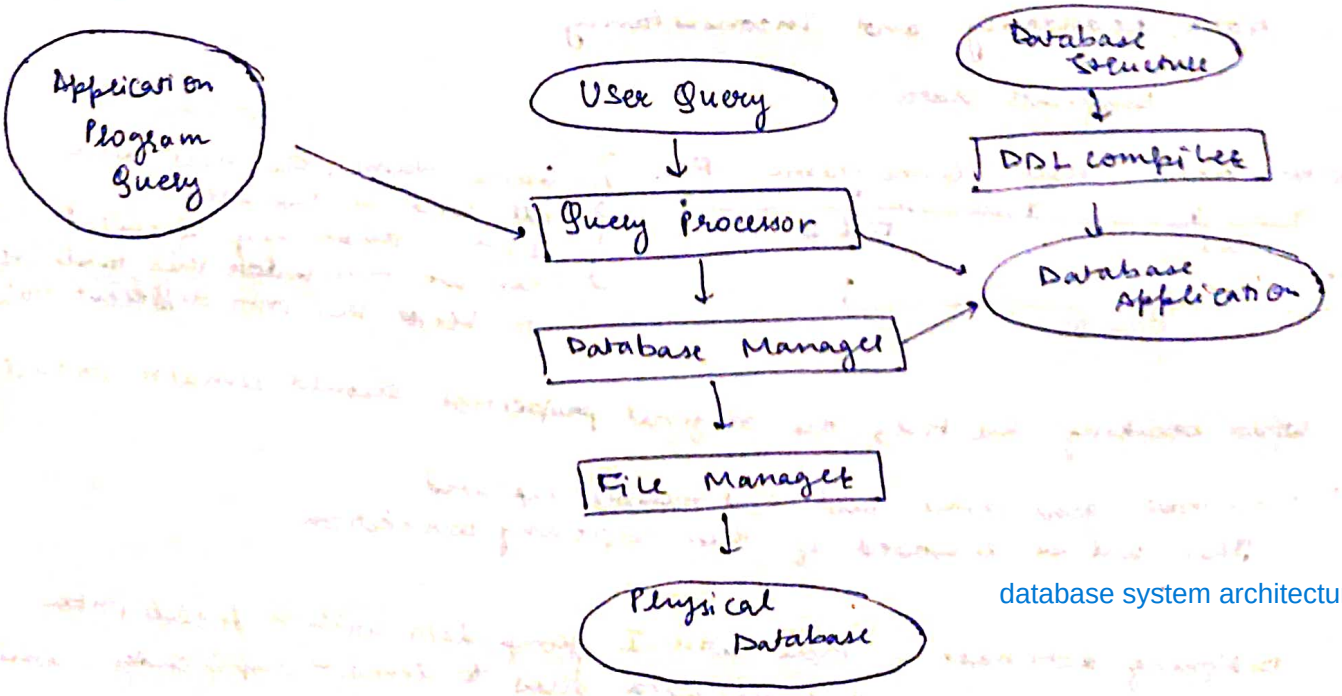
6. Synchronisation

Access to the database when you want to modify the database will be given to only one person.

Access to the database when you want to read only, multiple users are allowed.

Recovery Process

Periodically taking back up and storing it in a isolated system



database system architecture

DML - process queries
File manager - details about individual files
Physical database

Field ≡ Attribute

Levels of Abstraction

User Group — The way user is viewing it

Conceptual Database (while designing) — The way designer is viewing it.

Physical Database (stored in binary format)

Object Based Logical Methods

1) Entity - relationship model
2) Binary model
3) Semantic data model
4) Infological model

Record Based Logical Methods

1) Relational Model (RDBMS)
2) Network model
3) Hierarchial model

Entity Relationship Model

Entity : Object that exists & is distinguishable from other objects

Collection of entity → Entity Set

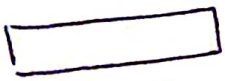
Customer (name, ph. no., city) } Entity sets
employee (name, ph. no.)

one - one
one - many
many - one
many - many
} Relationships

Primary key :
Collection of one or more than one attributes whose value will uniquely identify a record.

Modified
Minimum number of attributes which will uniquely identify records is key / primary key.

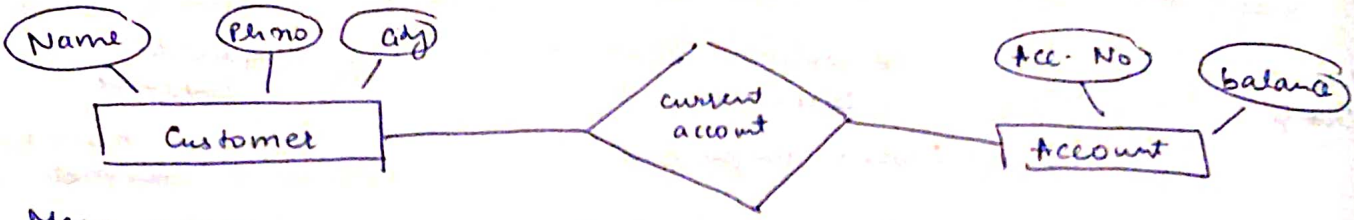
Diagram


Entity set

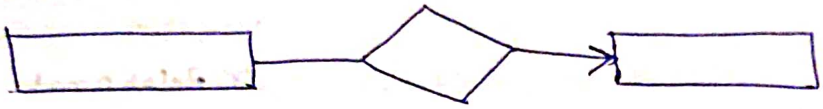

Attributes


Relationship

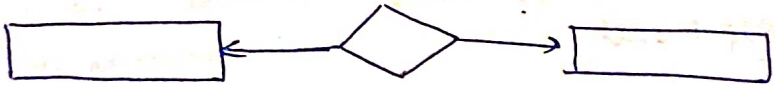
No arrow on both sides : many - many



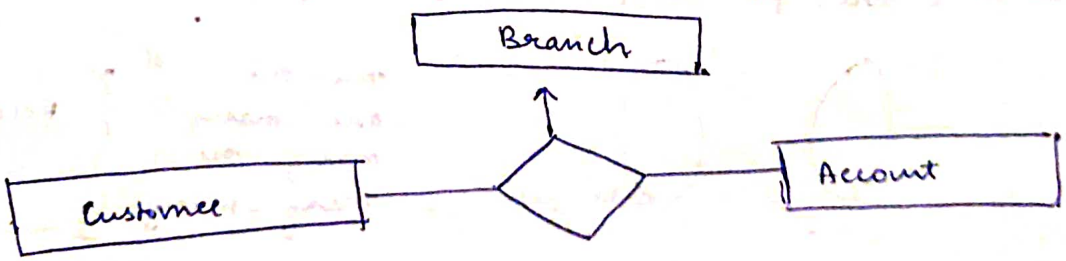
Many - many
A customer may have many accounts.
An account may belong to many customer



Many - one
A customer will have only 1 account
An account may belong to many customers



One - One
A customer will have only 1 account
An account will belong to only one customer.



A customer may have multiple account but they belong to a single branch

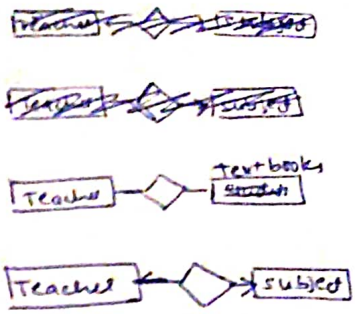
Draw 1 diagram for a database representing textbook, Subject, student, teacher

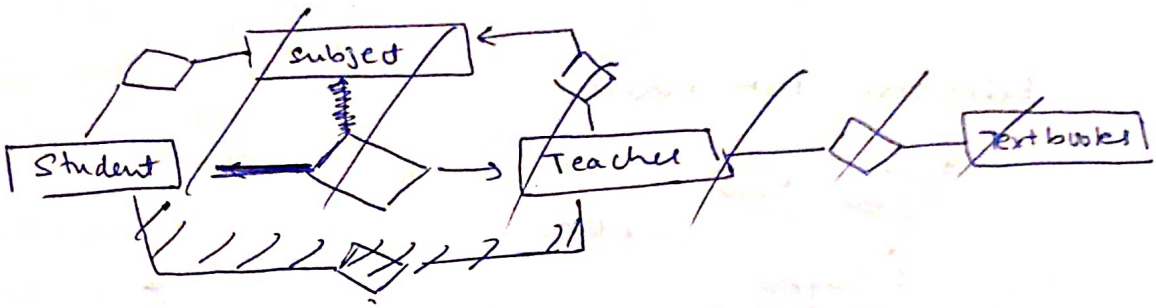
For each subject, each student is taught by a single teacher

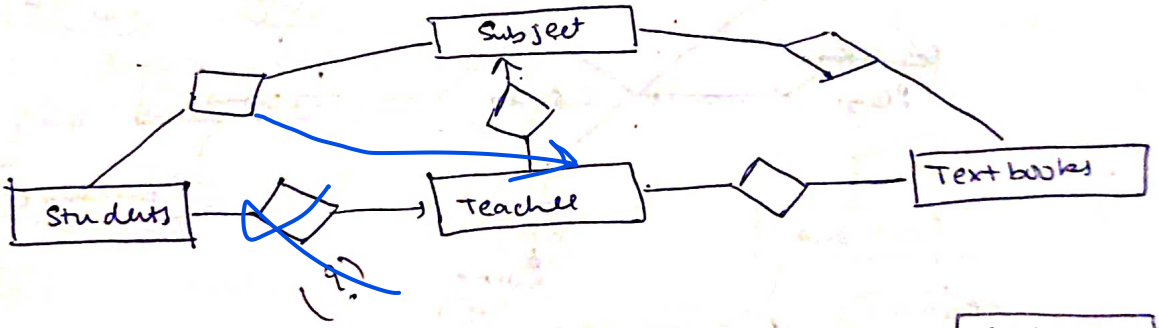
Each teacher teaches only 1 subject

Each subject is taught by several teachers

Teachers choose their respective text books

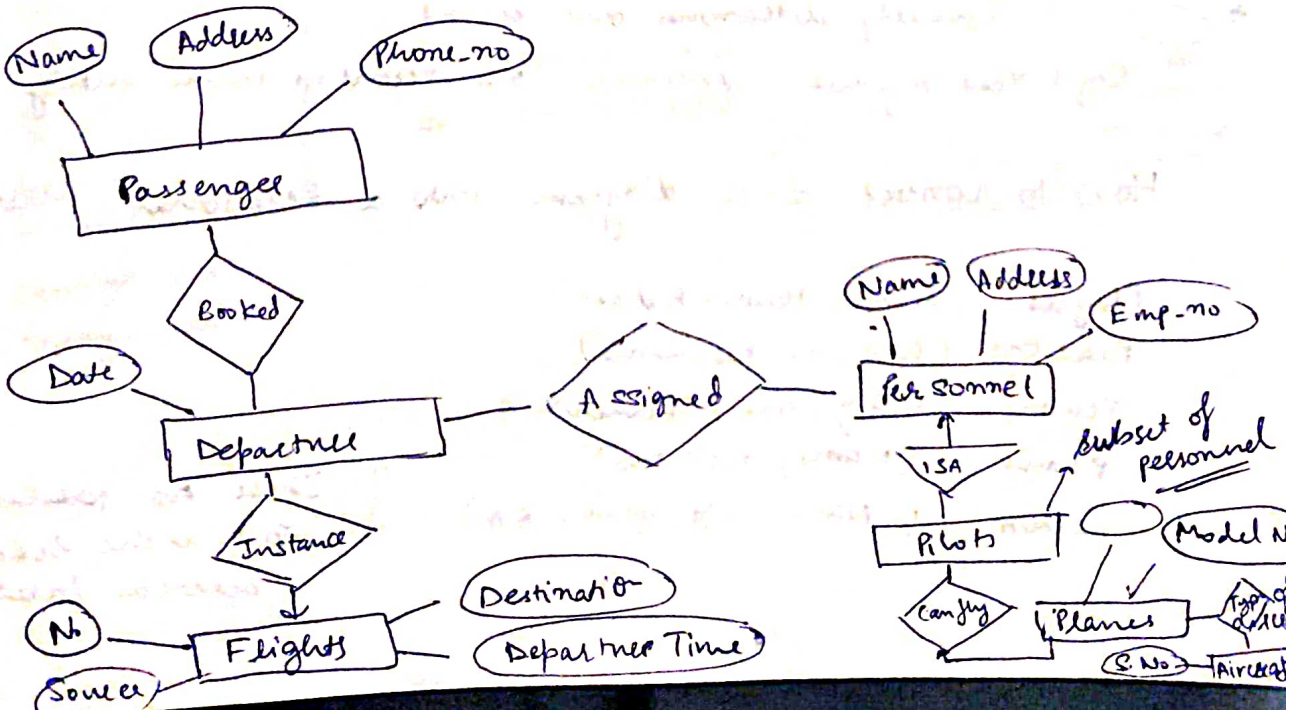


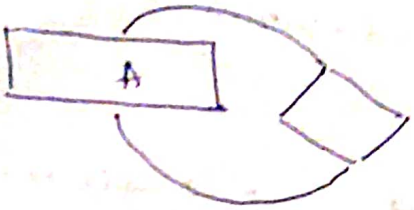




9.01.2023

Example



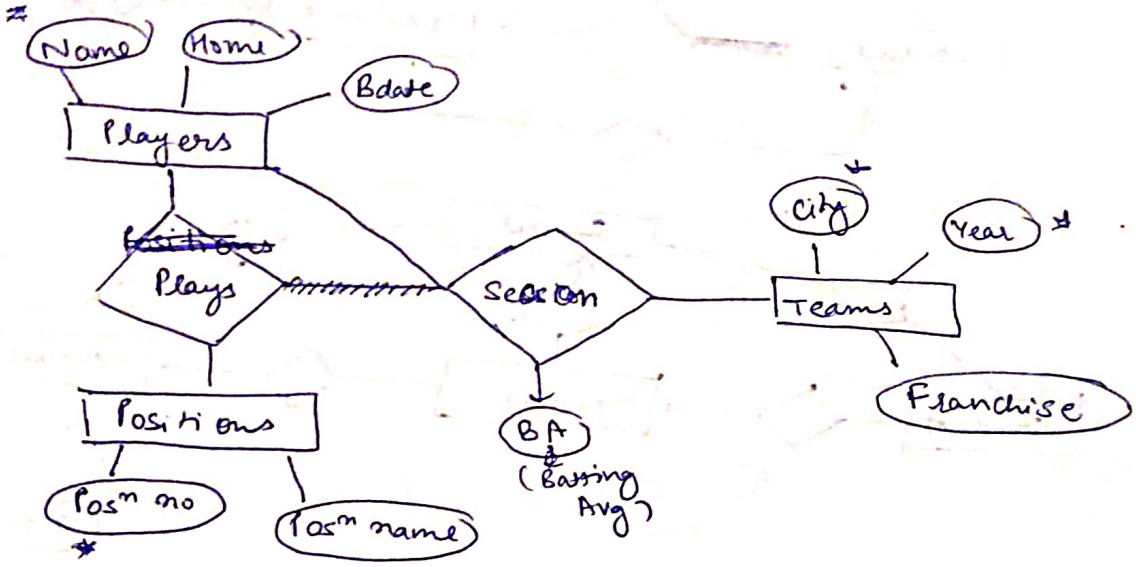


e.g. select employees whose salary is greater than that of his/her manager

E1	1000	M1
M1	900	----

E1 1000 ✓ M1 M1 900 ✓

Relational Data Model



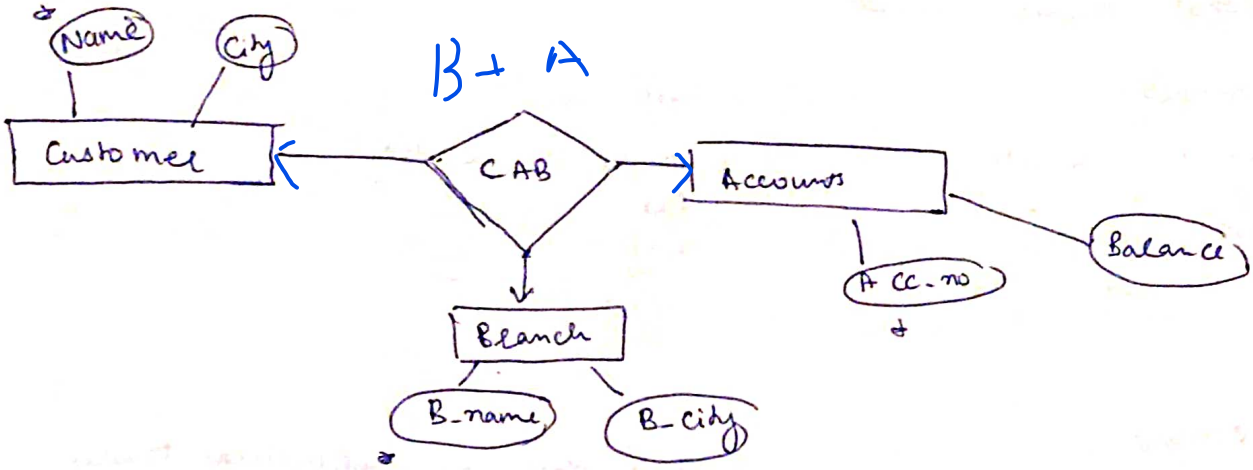
* ◯ : Uniquely determines one record

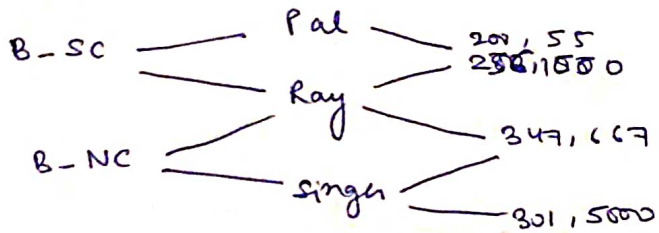
City & Year together determines one record in teams entity

How to Convert E-R diagram into a Relational Model?

Players (Name, Home, B date)
Positions (Pos_no, Pos_name)
Teams (City, Year, Franchise)
Plays (Name, Posn no)
Season (Name, City, year, BA)

} These two tables will give us the relation between tables





Relational Model

Customer

Accounts

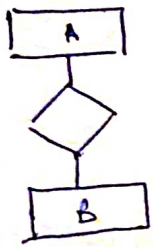
Branch

CAB (name , B-name , acc_no)

Pal B-SC 200

Network Model

- Binary and Many-one Relationships
  - 2-entities



2 entities but many-many

} convert to many-one for network model

Temporary file with many one relations with A & B.

Relational model

Branch

B-name	B-city
B-SC	-
B-NC	-

Customer

C-name	C-city
Pal	-
Ray	-
Singh	-

Account

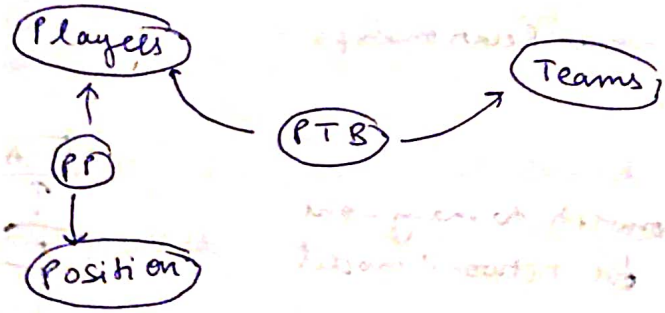
AC-no	balance
200	-
256	-
347	-
301	-

These three are individual tables corresponding to the entities

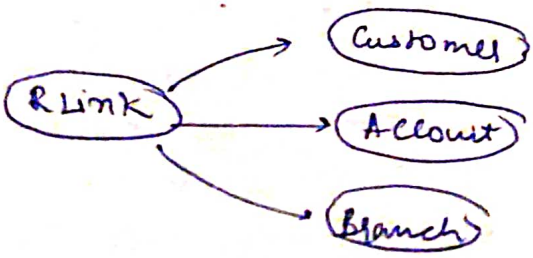
Now, CAB

C-name	B.name	AC-no
Pal	B-SC	200
Ray	B-SC	256
Ray	B-NC	347
Singh	B-NC	347
Singh	B-NC	301

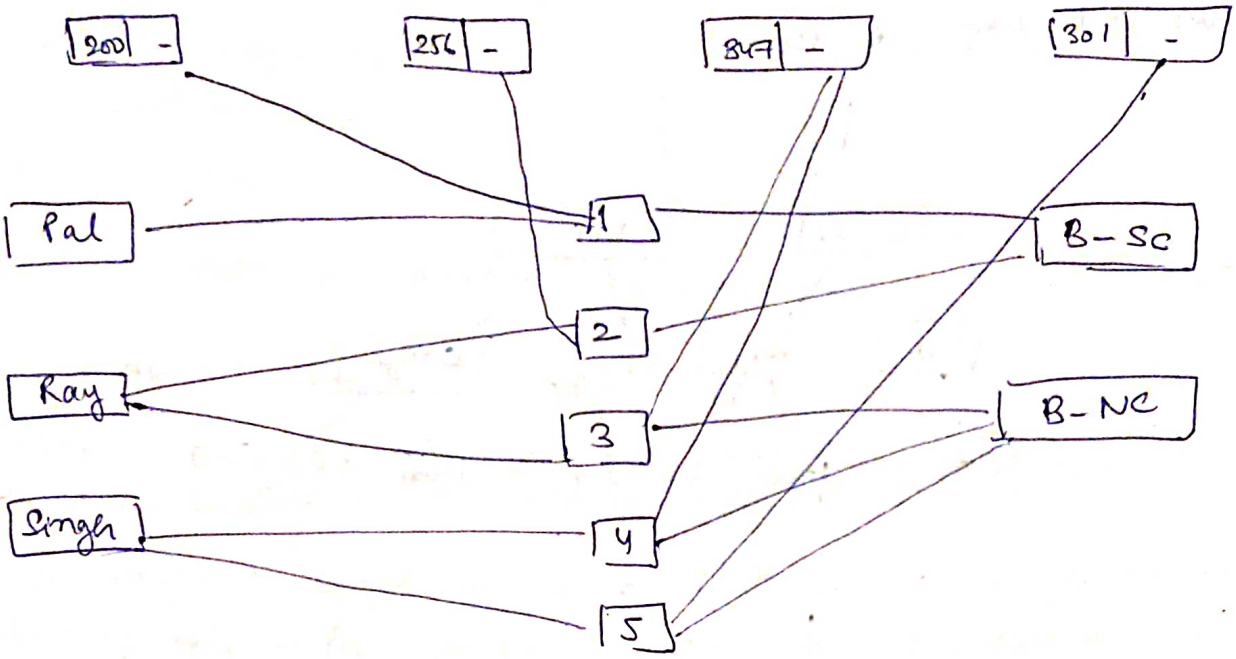
Consider player example



Consider customer-bank example

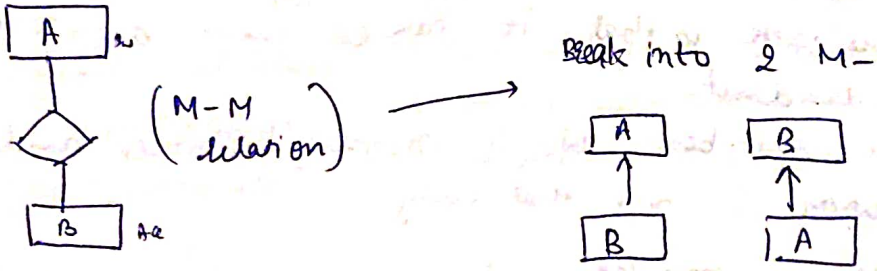


Converting relational model into network model

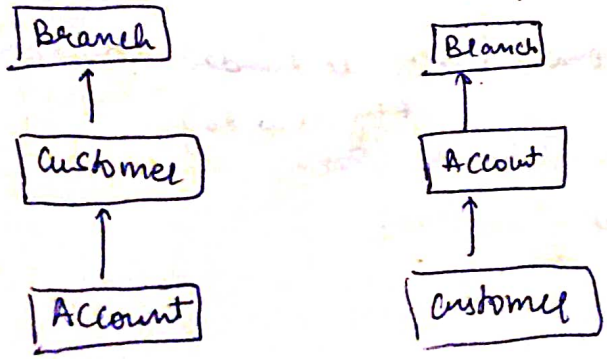


Here, there is no redundancy in data (all keys are mentioned only once) whereas, in relational models, the key may repeat again. Thus, this works faster but complex & implementation might be difficult. Also, adding new data is easy in relational model (just add a new record, while it's difficult in NW model. Thus, hierarchial model was invented

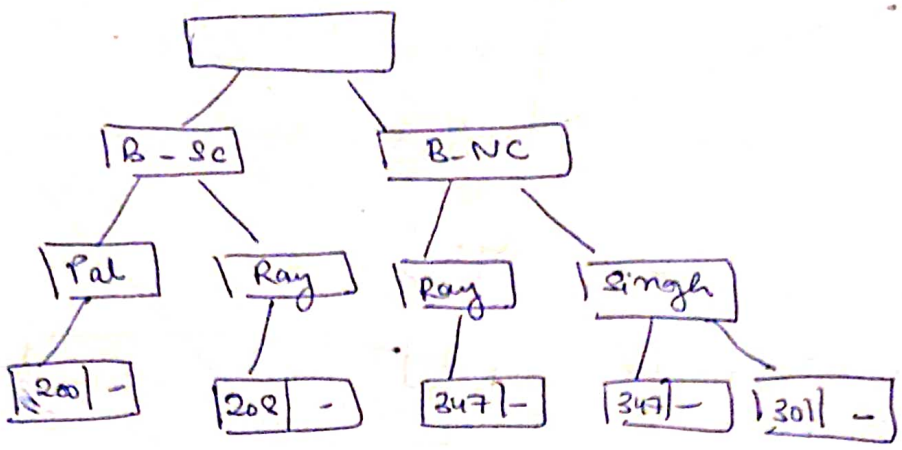
Hierarchial model

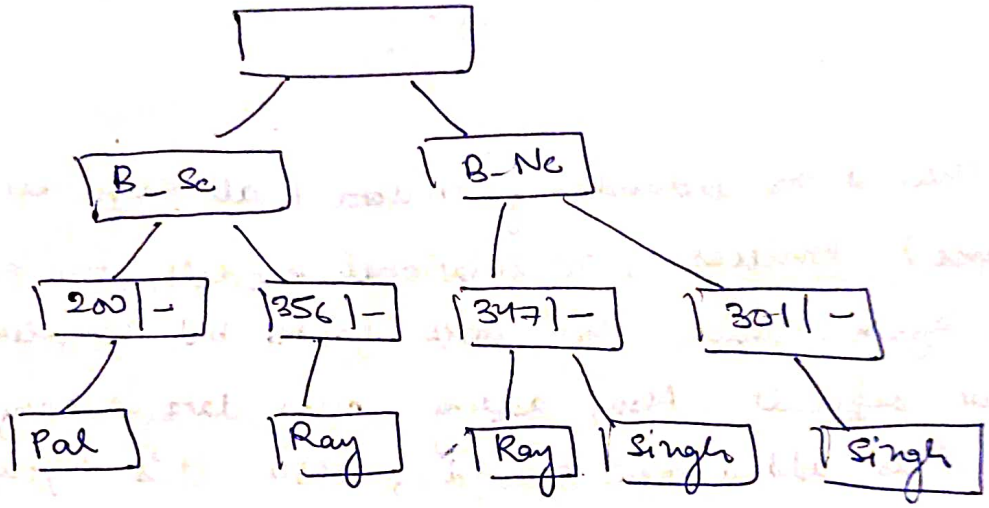


In branch-customer example,



If all the relations in the example are M-M, then we will have 6-tiers





In this model, we can add/delete on elements easily

+ Searching for a query is easy
+ Complexity is less.
* But, the drawback is that it takes more data & the method is redundant.

Still, it is not the best b/c it involves pointers and pointer application is not that easy.

The flow is network
↓
hierarchical
↓
relational (less redundant in the key records)

Relational Database design :

1) Redundancy
2) Updation anomolies
3) Insertion anomolies
4) Deletion anomolies

Relational database design has the above 4 drawbacks :

e.g. S_name Roll_no Hall Game Fee

A student can have multiple game participation & hence there is data redundancy

- If we have to update the fee of cricket, if we forget for one record by mistake for the game cricket, the the data becomes inconsistent.

  This is updation anamoly and to get rid of that, every record has to be updated.

- whenever new info has to be added, we need the values of all the attributes. This in sertion anamoly.

- For suppose, there is a record with game hockey, and if we delete it, we also delete the fee of the game (which may be come imp. later) while deleting the records of the student. This is deletion anamoly.

To avoid this, we can form additional tables as follows.

(S name, RN, Hall)
(RN, Game)
(Game, Fee)

] This will have redundancy in the key attributes but will solve the anomolies discussed above

Functional Dependency (FD)

$U \rightarrow U_i \subset U$

U → Universal set of attributes

$X \rightarrow Y$ (X and Y all subsets of U_i)

$X \rightarrow Y$: X functionally determines Y

or Y is functionally dependent on X

Relation, $R : X \to Y$ $\quad \mu_1$ & μ_2 (~~tuples~~ / records)

If $\mu_1[x] = \mu_2[x] \xRightarrow{\text{then}} \mu_1[y] = \mu_2[y]$

Then we say that X functionally determines Y

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

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

$A \to C \quad C \to A$

$D \to B$

$R : X \to Y$ has to be a valid f^n (1-1 mapping)

Also, FD's can be

$AB \to D$ (?)

$B \to DX$ not

$A \to A$ (Trivial FD)

(*) $A \to B$, $B \to C \Rightarrow A \to C$

If $\mu_1[a] = \mu_2[b]$, ~~then~~ & $\mu_1[b] = \mu_2[b] \Rightarrow \mu_1[c] = \mu_2[c]$

* Closure of a set of FD's

Let F be a set of FD's

Then, $F^* = \{ X \to Y \mid F \Rightarrow X \to Y \}$

F^* : closure of F

$X \to Y$ can be logically from FD's of set F

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

$F^+ = \{ A \to A, B \to B, A \to C, A \to ABC, AB \to ABC \cdots \}$

$C \to C$ Trivial

$A \to B$

$A \to C$

$A \to A$

$A \to ABC$

Primary Key

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

$X \subseteq A_1, A_2, \ldots, A_n$ (X has to be minimal

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

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

Armstrong's Axioms

1) Reflexivity : If $Y \subseteq X \subseteq U$ then $X \to Y$
2) Augmentation : If $X \to Y$ and $Z \subseteq U$, then $XZ \to YZ$.
3) Transitivity : If $A \to B$, $B \to C$ then $A \to C$

e.g. R(city, street, pin)

$F = \{$ city street $\to$ pin, pin $\to$ city $\}$

* city street $\to$ pin city street (from (1))

  * key

* pin $\to$ city

  pin street $\to$ city street

  $\Rightarrow$ $\to$ pin street

  $\to$ city street pin

∴ pin street is also a key

Inference Rules

1) Union Rules : — $\{X \to Y, X \to Z\}$ F $X \to YZ$

$XZ \to YZ$ & $X \to XZ$ (from ①)

∴ $X \to YZ$ (Transitivity)

2) Pseudo transitivity Rules : —

~~If $X \to Y$ & $Z \subseteq Y$, then $X \to Z$~~ $\{X \to Y, WY \to Z\}$ F $XW \to Z$

3) Decomposition Rule :

If $X \to Y$ & $Z \subseteq Y$, then $X \to Z$

4) If $X \to YZ$ then $X \to Y$ & $X \to Z$

$X \to YZ$ & $YZ \to Y \Rightarrow X \to Y$

$YZ \to Z \Rightarrow X \to Z$

* Closure of a set of attributes

X is a set of attributes, then X^+ (closure of X) is a closure of X if it is derived from X using A's axioms

* lemma: $X \to Y$ iff $Y \subseteq X^+$
* Theorem : Armstrong's axioms are sound & complete
* Complete : No other FD can be obtained beyond one's obtained from armstrong's axioms
* Sound : we can derive any invalid FD (invalid wrt given F) from armstrong's axioms

X^+ algorithm of Beth stein :-

1) let $N=0$ and $X(N) = X$
2) If there is $A \to B$ in F whose LHS A is contained in $X(N)$ But RHS B is not contained in $X(N)$, then make
$$X(N+1) = X(N) \cup B$$
otherwise, terminate
3) $N = N+1$, and go to step (2)

Ex: R(A, B, C, D, E, G}

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

$(BD)^+ = ?$

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

D → EG; X(1) = BDEC
BE → C; X(2) = BDEGCA
CE → AG; X(3) = BDEGCA

* R(FI, MOC, SAL, SS, JOB, COR, LUCK, ED, EFF, SEM)

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

Note: Attribute not on the RHS must be there in the set of attributes (min in size) whose closure is a set of Attributes

We can observe that

(LUCK, SS, JOB, SEN, EFF)$^+$ = X &

(LUCK, SS, ED, SEN, EFF)$^+$ = Y

↳ These 2 are keys now (since they are minimal)

Partial dependency (Redundant attributes)

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

↓

elementary FD, if there is no $x' \subset X$ s.t. $X' \to Y$

Ex: $AB \to C$ & $A \to C$ then B is redundant, since A uniquely determines C.

* F, set of FD's and

f: $X \to A$

if $B \in X$ & $(X \to \{B\}) \to A$ is in F^+, then

B is extraneous attribute

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

$(A)^+ = ACG$

$(AC)^+ = ACG$

Now observe C is redundant

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

⇓

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

* Redundant FD:

f in (F) f: $X \to A$ is redundant if

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

A set of FD's F is minimum if there is no set G with less number of FD's than F s.t. $G^+ = F^+$

e.g. $\{A \to B, A \to C\}$
F

F is not minimum since $G = \{A \to BC\}$ is minimum.

* L-minimum :- A set of FD's F is L-minimum if

1) F is minimum
2) ∃ no partial dependency in every FD

e.g $\{ABC \to D, A \to B\}$

It is minimum but not L-minimum

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

◯ LR-minimum: A set of FD's E is L-minimum if it is L-minimum & minimum on RHS

Ex: $A \to AB$

LR minimum is $A \to B$

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

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

Step 1: check for partial dependencies

These are for no partial dependencies

Step 2: check for redundant FD's

$AB \to E$:- $(AB)^+ = ABCFD$

↓

No E, so it is not redundant.

$AC \to F$:- $(AC)^+ = ACDBE$

⇒ NR

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

⇒ NR

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

⇒ NR

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

⇒ NR

NO redundant FD's

Step 3: minimize the no. of FD's

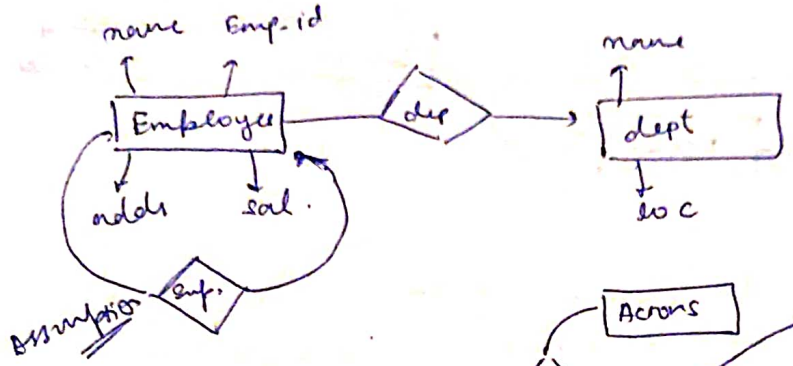
Note: You can minimize only if there all 2 FD's with same LHS, minimization can be done by merging the RHS, & else few info will be lost because of which some queries cannot be answered.

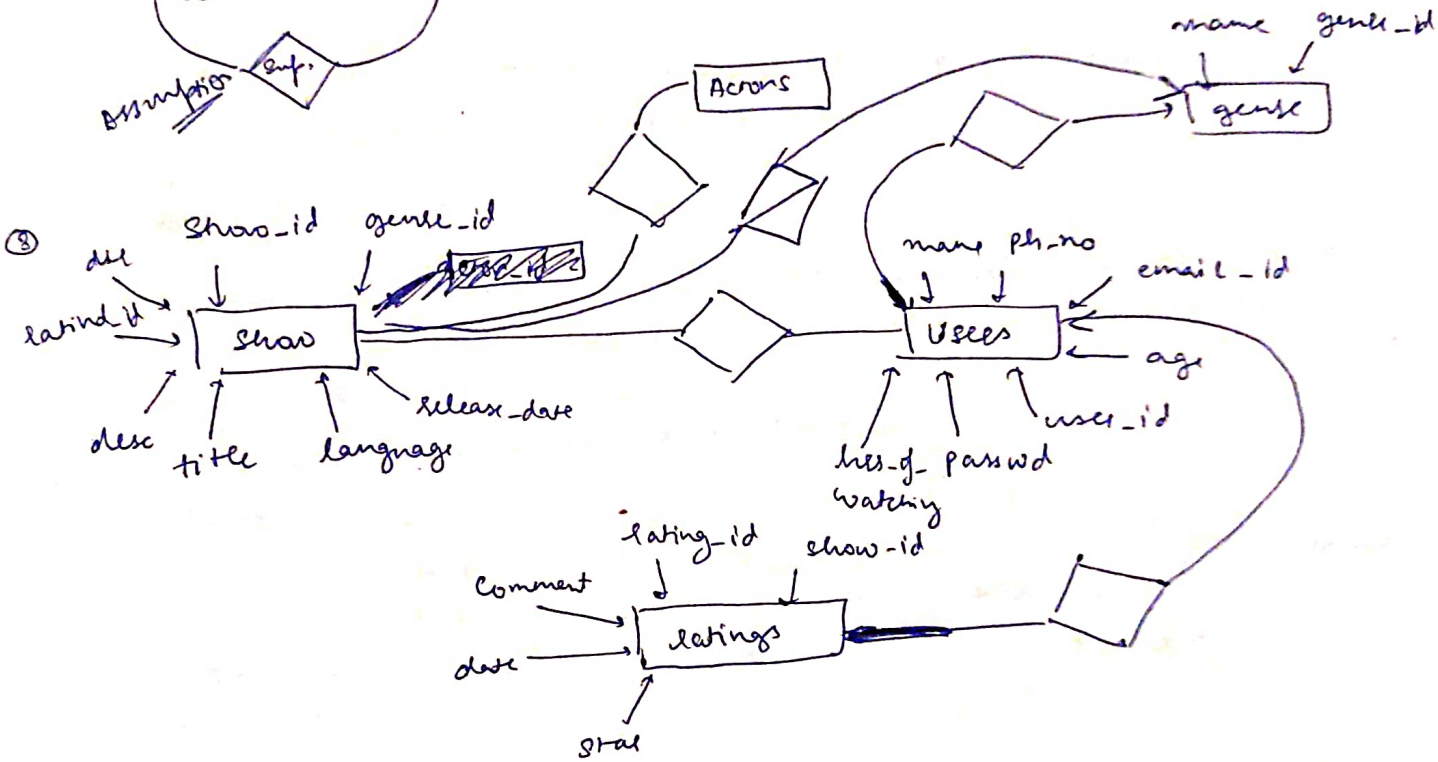
— x —

Lab Work

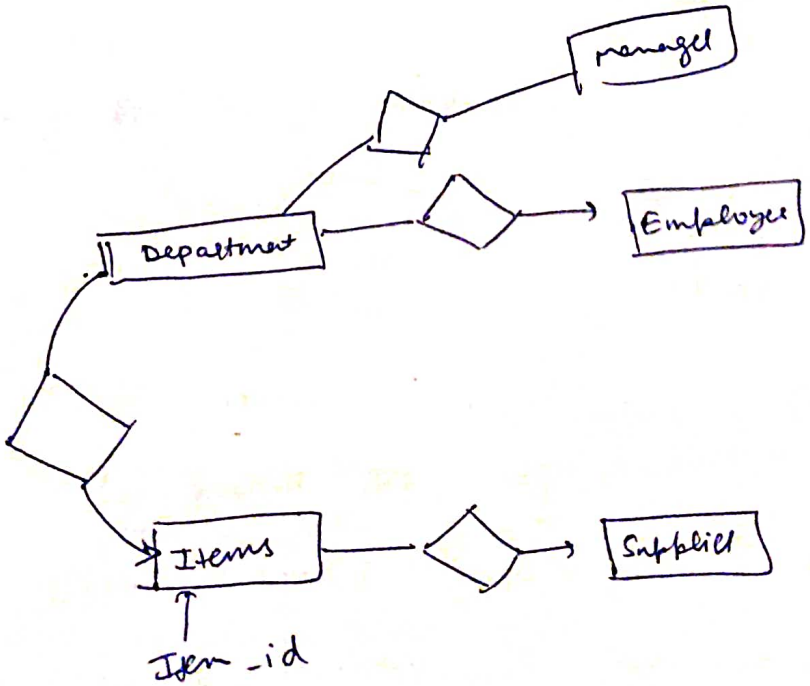
passwd

(2)





(4)



no dept or 1 dept

1 item
1 dept ??

Partial Dependency

Partial Dependancy: if by considering one/ more fields unknown on the LHS we can still reach the RHS, then the relation in partial and the unknown fields are redundant

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

$B \subseteq X$ $\quad\quad$ $AB \to C$

$A \to C$

FD : $f = X \to Y$

F (system of FD's)

When can we say that F is redundant?

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

Redunduancy: if we can reach from lhs to rhs without using the current relation, then the current relation is redudant

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

Attributes : A, B, C, D, E, G

Rewrite the FD 's s.t. RHS of each FD is a single attribute so that you have the minimum in the very beginning

Step I : Check Redundant dependencies

$AB \to C$	N.R
$C \to A$	N.R
$BC \to D$	N.R
$ACD \to B$	R ⊗
$D \to E$	N.R
$D \to G$	N.R

$BE \to C$	N.R
$CG \to B$	N.R
$CG \to D$	R ⊗
$CE \to A$	R ⊗
$CE \to G$	N.R.

Step II : Check for Partial dependency

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

No partial dependency

So, Now, LHS is minimum & RHS was already min.

Step III : Is it minimum or can we reduce it!

If for any two FD's, if RHS is same, merge the RHS

What is the advantage of merging?

Each FD will determine one table

↳ I have to handle less number of tables and less no. of data so that I can run my query in less time.

Decomposition of Relational Scheme

$R(A_1, A_2, \ldots, A_n)$ → n attributes

$P = \{R_1, R_2, \ldots, R_K\}$ decompose into $R_1, R_2, \ldots, R_K$

If I join then, I must get R

$R = R_1 \cup R_2 \cup \ldots \cup R_K$

① Lossless Join Property

R → $R_1, R_2, \ldots, R_K$

D is a set of dependencies

$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \ldots \bowtie \Pi_{R_K}(r)$

- r: Set of tuples of relations R
- $\Pi_{R_1}(r)$: From R_1, you only project attributes from R_1
- $\bowtie$: Natural join

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

$R(A, B, C)$

$\Pi_R(A, C)$

A	C
a_1	c_1
a_2	c_2
a_2	c_2

Theorem: $P = (R_1, R_2)$ is a decomposition of R

F → set of dependencies. Then P is a lossless join decomposition wrt F

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

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

Decomposition I: $R_1(A, B)$, $R_2\{A, C\}$

$R_1 \cap R_2 = A$

$R_1 - R_2 = B \qquad R_2 - R_1 = C$

$(R_1 \cap R_2')$

$A \to B$ $\quad F \alpha F^+$ $\quad$ lossless Join Prop. ✓

Decomposition II : $R_1(A, B)$, $R_2(B, C)$

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

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

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

$\Pi_{AB}(r) \bowtie \Pi_{BC}(r)$

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

$\neq r$

$\Rightarrow$ Not a loss less decomposition

16/01/23
S-C

Preservation of dependencies

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

$$\bigcup_{i=1}^{k} \Pi_{R_i}(F) = F$$

* If dependencies all not preserved, we may not be able to answer all the queries.

Let $X \to Y$ be there in F, but not in union, then we cannot determine Y if we have X.

* Both lossless join prop. & preservation prop all equally imp

e.g. $R(C, S, P)$

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

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

$R_1 \cap R_2 = P$

$\because P \to C$, $\quad P \to R_2 - R_1$ $\quad \therefore$ loss less prop satisfies

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

$\pi_{R_2}(F) = \{P \to C\}$

$\pi_{R_1}(F) \cup \pi_{R_2}(F) = \{S \to S, P \to P, P \to C\}$

$\neq F$

It does not preserve dependencies.

$\pi_{R_1}(F) \to$ FDs contain attributes of R_1

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

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

$R_1 \cap R_2 = \phi \to AB$ ✗

$R_2 \cap R_1 = \phi \to CD$ ✗

} No loss less

$\pi_{R_1}(F) = \{A \to B\}$

$\pi_{R_2}(F) = \{C \to D\}$

$\pi_{R_1}(F) \cup \pi_{R_2}(F) = F$

} Preserves dependencies

Candidate keys

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

Primary keys

It is the columns we chose to maintain uniqueness in a table.

Foreign keys

Collection of fields (one or more) in 1 table that uniquely identifies a row of another table

Super key

In primary key, if we add another column, it becomes super key

(S_name, RN, Hall)

RN → Primary

RN Hall → Super key

Normal Forms

Normalisation is a process by which relations can be decomposed into sub-relations with no anamolies.

* Boyce Codd Normal form (BCNF)

R → Relation
F → Set of dependencies

Every FD $X \to Y$ is in BCNF if X is the super key of the table

$\forall$ $F: X \to Y$ if $Y \notin X$ (Y not a sub set of F) and then X is a key attribute in X, then we say F is in BCNF
(X is a Key of relation R)

If all the FD's are in BCNF, then the relation is in BCNF.

* R (PAN, PI, DI, DRUG, QTY, COST)

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

Key :- { PAN, DRUG }

Because { PAN, DRUG }$^+$ = all attributes

PAN → PI, PI $\notin$ PAN
PAN is not key
} Not in BCNF ⇒ R is not in BCNF

Now, break it into two sub relations

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

R_1 ⇓ contains attributes of FD not in BCNF

R_2 ↳ contains attributes other than RHS in R_1 ↓ FD's become DRUG QTY → CO... ↓ Not in BCNF. Again break it

Find out the FD's of R_2 means FD's in F+ involving attributes of R_2 only.

② R_3 (DRUG, QTY, COST)
DRUG QTY → COST

R_4 (DRUG, QTY, DI, PAN)
PAN → DI (present in F+)
↳ Not in BCNF

contains attributes of R_2 excluding RHS of R_3

Normal Forms

Normalisation is a process by which relations can be decomposed into sub-relations with no anamolies.

* Boyce Codd Normal form (BCNF)

R → Relation
F → Set of dependencies

Every FD $X \to Y$ is in BCNF if X is the super key of the table

∀ $F: X \to Y$ if $Y \not\subseteq X$ (Y not a subset of F) and then X is a key attribute in X, then we say F is in BCNF (X is a Key of relation R)

If all the FD's are in BCNF, then the relation is in BCNF.

* R (PAN, PI, DI, DRUG, QTY, COST)

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

Key :- { PAN, DRUG }

Because { PAN, DRUG }$^+$ = all attributes

PAN → PI, PI $\not\subseteq$ PAN
PAN is not key
} Not in BCNF ⇒ R is not in BCNF

Now, break it into two sub relations

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

R_1 ⇓ contains attributes of FD not in BCNF

R_2 ↳ contains attributes other than RHS in R_1 ↓ FD's become DRUG QTY → COST ↓ Not in BCNF. Again break it

Find out the FD's of R_2 means FD's in F+ involving attributes of R_2 only.

② R_3 (DRUG, QTY, COST)
DRUG QTY → COST

R_4 (DRUG, QTY, DI, PAN)
PAN → DI (Present in F+)
↳ Not in BCNF

R_4 contains attributes of R_2 excluding RHS of R_3

(3) R_5 (PAN, DI)

R_6 (PAN, DRUG, QTY) → All the relation in R_6 contain key

Decomposition of R is R_1, R_3, R_5 & R_6

Theorem: BCNF decomposition produces lossless join decomposition always

(The key has to be there in one of the tables ultimately)

For the above example, PI → DI is lost on combining the FD's of sub relation ⇒ BCNF does not preserve FD!

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

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

key :- CD

Prime Attribute

An attribute is called a prime attribute if A is a member of key of R. otherwise, it is called non-prime attribute.

e.g. keys : AB, BC

A, B, C ⇒ Prime attributes

Third Normal Form (3NF) → used to reduce data duplication

R → set of relations is in 3NF if there is a FD,

X → A if (A ∉ X)

X is a key of R (or) A is a prime attribute (Subset Key)

* A relation in 3NF may / may not be in BCNF
* A relation in BCNF is also in 3NF.

Bernstein's 3NF Algorithm :-

Input : A set of attributes and a relation

Output : Sub relations free from diff anamolies

1) Rewrite the FD's ∋ ∃ only one attribute on RHS of each FD.

2) Rewrite the list of FD's ∋ ∃ no redudant FD

3) Rewrite FDs ∋ no proper sub set of LHS functionally determines RHS (Removing partial dependencies)

4) Combine the dependencies with same LHS using union rule.

   Then, if $X \to Y$ is in the list, make a decomposition with attributes $\{X, Y\}$. (Ensures preservation of dependency)

5) Calc. keys of original relation. If no key is included in any of the decomposed sub-relations, then create a new sub relation with the attributes of key only (Ensures loss less join prop.)

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

Ex. R (PAN, PI, DI, DRUG, QTY, COST)

$F = \{PAN \to PI, PI \to DI, PI\,DRUG \to QTY, DRUG\,QTY \to COST\}$

1) RHS is only single attributes

2) Same as F

3) No partial dependencies

4) No chance of union. Now, break into sub-relations

   R_1 (PAN, PI), R_2 (PI, DI), R_3 (PI, DRUG, QTY),

   R_4 (DRUG, QTY, COST)

5) Key - PAN DRUG

   Relation containing PAN DRUG is not there. So, make a new one

   R_5 (PAN, DRUG)

e.g. $R(A, B, C, D, E, F, G, H, I)$

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

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

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

3) Same as above

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

$R_1(A, B, C, D)$ $R_2(A, E, F)$ $R_3(F, A, E, G)$
$R_4(C, H, I)$

5) Key :- F

Then all relations with AE & AF

$R(A, E, F, G)$

suppose AE and Af are keys and if only AE is included in a relation and not AF then should we construct a new R= {AF}

So, 3NF form is

$R_1(A, B, C, D)$; $R_2(A, E, F, G)$; $R_3(C, H, I)$

Multi-Valued Dependency

Math	T1	B1
Math	T1	B2
Math	T2	B1
Math	T2	B2
	⋮	

Combinations can be ←

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

→ $X \to\to Y$ (MVD)

An MVD would mean that for some single value of 'X', multiple values of 'Y' can exist.

(C, t_1, x_1), (C, t_2, x_2) appears then (C, t_1, x_2), (C, t_2, x_1) will also appear. This is called multivalued dependency.

** $X \to\to Y$ holds in R if

$\forall$ t, s (pair of tuples in R) s.t $t[X] = s[X]$, then $\exists$ two more tuples U, V such that

1) $U[X] = V[X] = t[X] = s[X]$
2) $u[Y] = t[Y]$ & $u[R-X-Y] = s[R-X-Y]$
3) $U[Y] = s[Y]$ & $u[R-X-Y] = T[R-X-Y]$

X	Y	R − X − Y
t[X]	t[Y]	t[R−X−Y]
s[X]	s[Y]	s[R−X−Y]
U[X]	U[Y]	U[R−X−Y]
V[X]	V[Y]	V[R−X−Y]

R(X, Y, Z)

* $X \to\to Y$ then $X \to\to Z$
  (X multi determines Y)
* $\{X \to\to Y, Y \to\to Z\} = X \to\to (Z-Y)$
* $\{X \to Y\} \neq X \to\to Y$

4th Normal Form (4NF) (BCNF + no MVD)

R(X, Y, Z) → Relation is 4NF if $\exists X \to\to Y$ where $Y \notin X$ and X Y does not include all the attributes of R, then X is a key of R

[No non trivial MVD other than candidate key]

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

  D = {A →→ B, B →→ G, B →→ EF, CD → E}

  Key : ACD

  A →→ B not in 4NF (So, Breakdown)

(?) R_1(A, B) ✓

A →→ B
B →→ EF
A →→ EF − B
A →→ EF, A →→ F

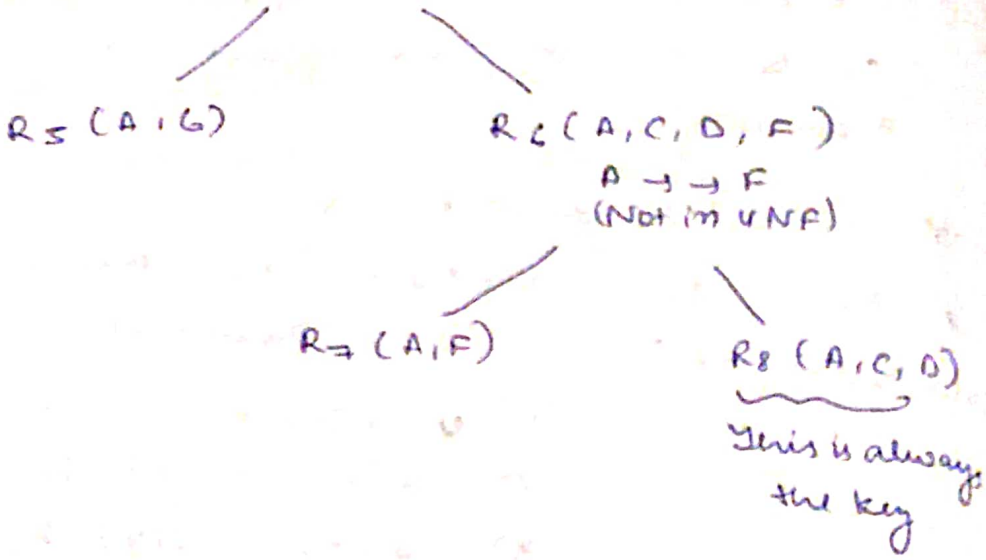
, R_2(A, C, D, E, F, G) ← (similar to BCNF)

CD → E (not in 4NF)

R_3(C, D, E)

R_4(A, C, D, F, G)

A →→ G − B ⇒ A →→ G


R5 (A, G)
R6 (A, C, D, F)
A →→ F
(Not in UNF)
R7 (A, F)
R8 (A, C, D)
This is always the key

DBMS

SQL Documentation

database:

a	g	d
4	6	3
7	4	2
4	9	1
3	8	4
7	6	3

- select g, a, d, a from z
  ↑ project

g	a	d	a
6	4	3	4
4	7	6	7
9	4	1	4
8	3	4	3
6	7	3	7

- Select * from z where a+d > g [this select is used to drop the row]

a	g	d
4	6	3
7	4	6
7	6	3

- Select max(g) from z
- Select sum(d) from z
- Select a, max(g), d from z XX INVALID XX
- Select * from z where g = a
- Select * from z where g = (select max(g) from z)

a	g	d
4	9	1

- Select * from z where g in (9, 8, 6)

a	g	d
4	6	3
4	9	1
3	8	4
7	6	3

- Select max(g) from z group by a
  ⇒ Rows having same value of a are used, and we find max(g) is returned.

Output : 9, 6, 8

- select a, max(g) from z group by a

a	max(g)
4	9
7	6
3	8

- Select d, max(g) from z group by a
  XX MEANINGLESS OPERATION X X

However, intuitively, we want to create a table as follows -

d	max(g)
1	9
3	6
4	8

This can be done by

- Select d, g from z where (a, g) in (select (a, max(g)) from z group by a).

Why would
" select d, g from z where g in (select max g from z group by a)
not work ?

- Select a, sum(g), d. group by a

a	sum(g)	d
4	15	
7	10	
3	8	

- Select d, g from z where (a, g) in (select a, sum(g) from z group by a)

{(4, 15), (7, 10), (3, 8)}

d	g
4	8

There exists some integrity constraints

- Create table U (a int, k int, check a < k)
  ↳ INTEGRITY CONSTRAINT

- insert into U values (5, 9)

a	k
5	9

- insert into U values (3, 6)

a	k
5	9
3	6

- insert into U values (8, 4) XX INVALID XX

- create table U (a int, k int, primary key a)
- insert into U values (8, 4)

a	k
5	9
3	6
8	4

- insert into U values (3, 1)
  XX INVALD XX (a is primary key)

- Create table V (p int, n int, m int, primary key p, m, foreign key m references U(k))

Table V

p	n	m

- insert into V values (4, 3, 6)
  insert into V values (4, 8, 1)
  insert into V values (6, 7, 1)

p	n	m
4	3	6
4	8	1
6	7	1

- insert into V values (9,1,7)

  XX Invalid XX [m is a foreign key referencing U(k) and 7 is not present in U(k)]

- insert into U values (8,7)
- insert into V values (9,1,7)

Example : Pawan Premier League

Players :

Name	Age
Arul	43
Dipu	17
Gyan	29
Hari	23
Jalaj	18
Kapil	37

Matches :

No.	Place
#1	Delhi
#2	Bombay
#3	Kanpur
#4	Patna
#5	Delhi

Performance :

Name	No.	Runs
Gyan	#3	65
Dipu	#4	71
Hari	#3	82
Gyan	#1	73
Jalaj	#4	71
Dipu	#2	63
Hari	#5	89

- create table player (Name, char (10) , age int, primary key Name)
- create table matches (no. int, place char(6), primary key no.)

(?) · update matches insert key place.

- create table performance (Name, char (10), no int, runs int, primary key (Name, no) , foreign key Name references player (Name) , foreign key no references matches (no))

K

p	g
3	6
2	9
8	4

- select * from z, k
  - ↳ Join(x) ⇐ cartesian product

a	g	d	f	g
4	6	3	3	6
4	6	3	2	9
4	6	3	8	4
⋮	⋮	⋮	⋮	⋮

6 * 3 = 18 rows

Suppose still all columns with same name (say g) in both the tables, then z, k =>

a	z.g	d	p	k.g

- select * from z, k (Natural join) where z.g = k.g.

a	z.g	d	f	k.g
4	6	3	3	6
7	4	6	8	4
4	9	1	2	9
7	6	3	3	6
4	9	5	2	9

- select a, z, g, d, p from (select * from z, k where z.g = k.g)

k.g column not displayed

We see that ad → g

P	Q	R
4	2	8
4	7	6
5	1	9
4	7	8
5	1	3
4	2	6

$\therefore P \rightarrow Q$ absent or, $P \not\rightarrow Q$

$Q = \{2, 7\}$ are independent

Hence $P \not\rightarrow Q$

$\{2, 7\}$

↙ ↘

$\{8, 6\}$

Relational Algebra :

Operations

1. Union (∪) R ∪ S

R :

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

S :

D	E	F
b	g	a
d	a	f

R ∪ S :

a	b	c
d	a	f
c	b	d
b	g	a

2. Set difference (−) : R − S

a	b	c
c	b	d

3. Cartesian Product (×) : R × S

$R^{K_1} \times S^{K_2}$

$(K_1 + K_2)$

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

Projection (Π)

$\Pi_{i1}, \Pi_{i2}, \cdots, \Pi_{im}(R)$

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

C	A
c	a
f	d
d	c

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

E	F
g	a
a	f

Remark: We select all values based on operations, from all tuples without checking if the row values are unique or not. It is possible that the values can duplicate.

5. Selection (σ)

F : a formula)

$\sigma_F(R)$ $\quad$ $\sigma_{2>3}(R)$

$\sigma_{1='s' \vee 2='p'}(R)$ $\quad$ operation is or

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

A	B	C
a	b	c
c	b	d

6. Additional Operations

1. Intersection ($\cap$)

$R \cap S$ =

d	a	f

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

2. Quotient ($\div$)

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

then $R \div S$ is the set of tuples with attribute no. $(r-s)$ such that $\forall$ tuples u in S, t is in R, t is a tuple of $R \div S$

R

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

S

c	d
e	f

$R \div S$

a	b
e	d

3. θ-join : $R \bowtie_{i \theta j} S$

Take the cartesian product R×S

$R \bowtie_{i \theta j} S$

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

R:

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

S:

D	E
3	1
6	2

$R \bowtie_{B<D} S$

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

4. Natural Join ($\bowtie$)

i) R×S

ii) R.A = S.A

iii) Remove S.A

R:

A	B	C
a	b	c
a	d	c
b	b	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

* Employee (emp-id, name, salary, m-id)
↳ 1 for managers
0 otherwise

Q Find out the name of employees whose salary is more than atleast one manager

Soln :- $\Pi_2(\sigma_{4='10' \wedge 8='1' \wedge 3>7}(\text{Employee} \times \text{Employee}))$

* Customer (c-name, street, c-city)
* Deposit (b-name, ac-no, c-name, balance)
* Borrow (b-name, loan-no, c-name, amt)
* Branch (b-name, b-city, asset)
* client (c-name, emp-name)

Q Find all customer names such that customer name & employee name are the same

$\Pi_1(\sigma_{1=2}(\text{client}))$ it should be 1=2 right?

Q Find all clients of company "XYZ" and corresponding cities of those clients

$\Pi_{1,5}(\sigma_{2='XYZ' \wedge 1=3}(\text{client} \times \text{customer}))$

Q Find all customer who have both acct & loan at kgp branch.

$\Pi_3(\sigma_{1='KGP'}(\text{Borrow} \times \text{Deposit}))$

Q Find all customers who have account in all branches in the city of Kolkata

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

Lives (p-name, street, city)
Works (p-name, c-name, salary)
Located in (c-name, c-city)
Manages (p-name, m-name)

Q Find name, street & city of all employees who work for HCL company and whose salary is more than 50,000.

$$\Pi_{1,3}\left(\text{LIVES} \bowtie \Pi_{\text{p-name}} \sigma_{\substack{\text{c-name}='HCL' \\ \text{salary} > 50,000}} (\text{works})\right)$$

Q Find all employees who live in the same city & company they work for!

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

Q Find all employees who live in the same city & street as their manager!

$$\Pi_1\left[\sigma_{\substack{\text{p-name}=\text{m-name} \wedge 2=4 \wedge 3=5 \\ \wedge \text{p-name} \neq \text{p-name}}} (\text{Lives} \times (\text{Lives} \bowtie \text{Manages}))\right]$$

Q Find all employees whose salary is more than every employee of "XYZ" company.?

~~works / works~~

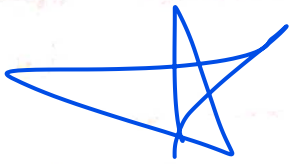
~~Π_3 / p / p / c-name~~

$P = \Pi_3\left(\sigma_{\text{c-name}='XYZ'} (\text{works})\right)$ — 10, 5, 7, 20

$Q = \Pi_1\left(\sigma_{1<2} (P \times P)\right)$ — 10, 5, 7

$R = P - Q$ — 20

$$\Pi_1\left(\sigma_{\text{salary} > R} (\text{works})\right)$$



Propositional logic

$\neg$ (not), $\wedge$ (and), $\vee$ (or)
$\Rightarrow$ (implication or if-then)
$\equiv$ (equivalence or iff)

Proposition: Any statement which is either true or false.

Symbols to represent propositions : atom(s)

Tuple Relational Calculus

Expression form : $\{ t \mid \psi(t) \}$

- t : tuple variables of some fixed length
- $\psi(t)$ is a formula built using atom(s)

$R(s)$, $s \in R$: s is a tuple of relation R.

$s[i]\ \theta\ u[j]$: i^{th} component of tuple s is in relation θ with j^{th} component / attribute of tuple u

- s : tuple
- i, j : attribute no / name

* Relation is in terms of tuple (whereas in relational algebra relation was in terms of attributes)

1) Exential Quantifier ($\exists$)

$(\exists x)\ (\psi(x))$: There exists atleast one tuple x for which $\psi(x)$ is true

2) Universal Quantifier ($\forall$)

$(\forall x)\ \psi(x)$

e.g. $(\exists s)(R(s))$: $\exists$ at least one tuple s which is a tuple of relation R

e.g. $(\forall s)(\psi(s))$: Any tuple s you take, it is a member of ψ

These form of formula .

Lemma : If you can express 1 query in relational algebra, then you can express that query in Tuple Relational Calculus

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

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

$R^{(r)} \times S^{(s)}$: $\{t^{(r+s)} \mid (\exists u)(\exists v)(\underbrace{R(u)}_{\substack{u \text{ is a} \\ \text{tuple variable} \\ \text{of } R}} \wedge S(v) \wedge \underbrace{\quad}_{\substack{v \text{ is a tuple} \\ \text{variable of } S}}$

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

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

$\Pi_{A,C}(R) = \{t \mid (\exists u)(R(u) \wedge (t[1] = u[1]) \wedge (t[2] = u[3]))\}$

U
A B C D

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

expression that we'll write in relational algebra is complement of the exp. that we'll write in tuple calculus

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

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

Note $(2=3) \neq (u[2] = v[1])$

$R \bowtie S = \{t \mid (\exists u)(\exists v)(R(u) \wedge S(v) \wedge (t[1] = u[1]) \wedge (t[2] = u[2]) \wedge (t[3] = v[2]) \wedge (u[2] = v[1])\}$

Q Find the B_name, loan_no, customer_name and amount for loan > 10000.

~~$\{t \mid (\exists u)(Borrow(u) \wedge t[1] = u[1]$~~

$\{t \mid t \in Borrow \wedge t[4] > 10,000\}$

(If exactly type of some table $\Rightarrow$

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

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

Q Find all customers who have loan in 'IIT' branch and their cities where they live.

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

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

Q Find all customers having a loan and account of both at 'IIT' branch.

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

wrong

or

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

right

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

~~$\{t \mid (\exists u)(\exists v)\}$~~

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

Note: $P \Rightarrow Q \quad \neg P \vee Q$

$\{ t \mid (\forall u)(u \notin Branch \vee u[2] \neq 'KGP' \vee (\exists v)(v \in Deposit \wedge u[1] = v[1] \wedge t[1] = v[3])) \}$

SQL

Select $A_1, A_2, \ldots, A_n$ (Ai are attributes from relations R_i)
from $R_1, R_2, \ldots, R_m$
where P; (Predicate / condition)

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

Note: union intersection and or not

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

distinct (for unique branch names)

```
Select   b_name
from     Deposit ;
```

Q Find all customers who have account at 'IIIT' branch

```
Select   c_name
from     Deposit
where    b_name = 'IIIT' ;
```

Q Find all customers having a loan and account of both at 'IIIT' branch

```
( Select   c_name
  from     Deposit
  where    b_name = 'IIIT' )
union
( select   c_name
  from     Borrow
  where    b_name = 'IIIT' );
```

Q Find all customers having a loan at any branch and their city.

```
Select   c.c_name, c_city
from     customer c , borrow b
where    c.c_name = b.c_name ;
```

Note: If we want loan only from 'IIT' branch

Add condn in where clause

```
and   b_name = 'IIIT'
```

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

```
select    c_name
from      deposit , borrow
where     ( deposite. c_name = borrow. c_name ) and
          ( deposit. b_name = 'IIIT' ) and ( borrow.c_name = 'IIT')
```

Or

```
Select   b_name
from     Deposit
where    b_name = 'IIIT'
and      c_name in
(select  c_name
 from    Borrow
 where   b_name = 'IIIT')
```