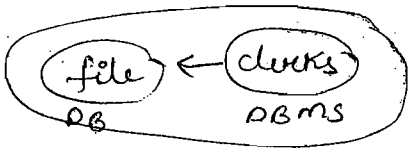
Data base management system :-

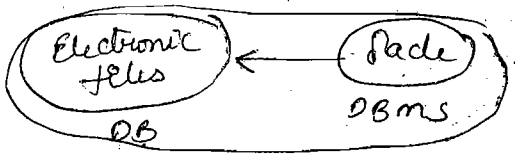
Data : Independent Component.
Record : Inter related Component.
Data Base : Set of record.
DBMS : S/w used to create, manipulate

DB + DBMS → Data base system (DBS)

Paper - Data base
Pen - DBMS
↓
Data Base system.



DBS



DBS

Now a days wide varieties of DBMS are available from many vendors like Oracle.

House Construction :-

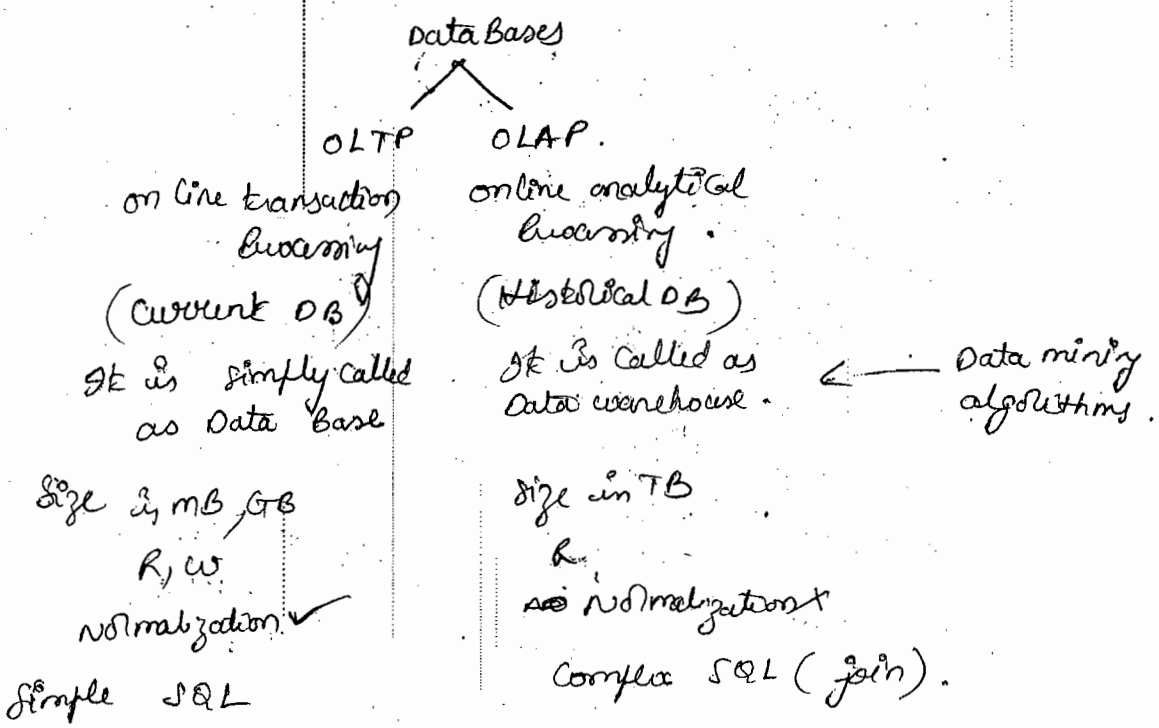
Requirements
↓
2D - Plan
↓
Prototype (3D - Plan)
↓
Physical Construction.

Data Base design :

Requirements.
↓
Plan (Conceptual) (ER diagrams.)
↓
Logical design stage (Function dependencies, normalization, SQL)
↓
Physical Construction (B, B+, files)

once the construction is over, we cannot change it.

→ In single user application we may not think of concurrency and transaction management, but if we are using the multiuser application then they comes into picture.



another classification :-

Commercial DB : inventory control, library, student info system.
multimedia DB : voice, video clips, images.
Temporal DB : attaching time aspect to data · Ex: Railway reservation.
GIS DB : (Geographical information system) Ex: To give ways to reach destination (route map) (google earth)
Distributed DB : web environment. (Consistency is very high) ⇒ (concurrency and transactions)
Deductive DB : Rule based · (A=B, B=C ⇒ A=C)
Document DB
active DB
web DB.

deductive DB:- If we buy one thing, they then is chance of buying the other.

Document DB:- Google, Content information system.

active DB:- Triggers flooded with triggers

web DB:- Any thing can become a web data base if they are available in web environment.

Different types of DBMS's :-

Relational DBMS
OR DBMS (Object relational DBMS) & Oracle 9i
OO DBMS (Object Oriented DBMS)

ORDBMS :- In RDBMS we added Object Oriented principles.

OODBMS :- In Object Oriented we added Relational principles.

Except for multimedia, GIS, Document and web we can use ORDBMS

Only for the four we used OODBMS.

we are going to study only RDBMS.

ER-diagrams :-

model : imitation of reality.

different models : ER, UML, DFD, HIPO etc.

Requirements (Data)

Static	Dynamic
ER.	ST. D.
EER.	DFD.
UML.	

At time of development of DBMS, the data is not going to be changed and so it is static data. and we can go for any one of techniques available in this Category. Let us go for ER.

→ It gives the pictorial representation of all the requirements in the information system.

→ It is a data model, in which information stored in the data base is viewed as set of entities and their relationships.

→ The components of ER diagrams

(i) Entities
(ii) Relationships
(iii) attributes.

A thing or object with an independent existance is called an entity.

entities

Physical
→ Can be touched (sensed)
→ Ex:- FAN

Conceptual
cannot be touched (sensed)
→ Ex:- course offered, Brand name etc. Good will.

→ Entities can have conceptual or physical.

→ storing conceptual entities is a big problem.

Ex:- student

Roll No	name	age
1	a	5
2	b	6
3	c	7

} set

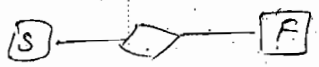
we have 3 entity occurances, all these are called entity set.

Relationships:

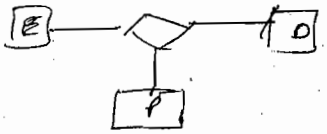
Association among the entities is known as relationship

Types:

Binary



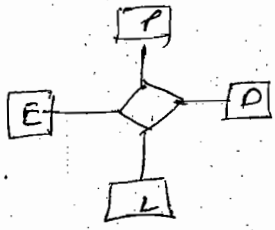
Three



E - employee.
D - Department.
P - Project.

ER diagrams cannot take care of ternary relationship.

four

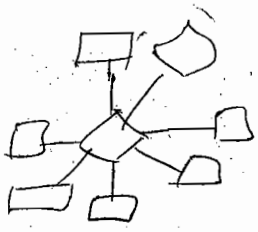


quadernary relationship.

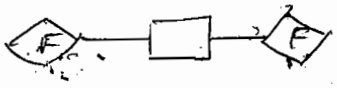
L - location.

ER cannot take care of four also.

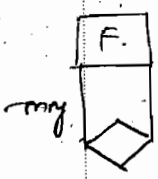
n-ary



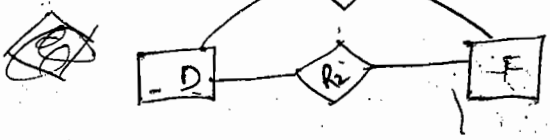
unary

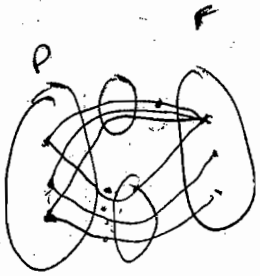


=>



multiple relationships b/w same entities





~~other (or) type of classification~~

degree of relationship.

1 : 1.

1 : many

many : many.

Identify : unary, binary, ternary then identify degree

total and partial participation.

Constraints on Relationships:

(i) Structural Constraints: (applicable for binary relationships)

a) Cardinality Ratio
b) Participation Constraints.

(ii) overlap constraints: } applicable for EER (extended) diagrams

(iii) covering constraints: }

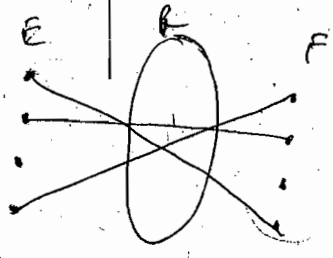
Cardinality Ratio: It describes the maximum number of possible relationship occurances for an entity to an another entity through the relationship.

Participation: It describes whether all or only some entity occurances are participating in a relationship. It is total if all entity occurances are participating. otherwise, it is partial.

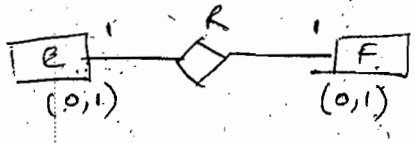
minimum Cardinality = 0 for partial participation and equal to 1 for total participation.

maximum Cardinality = 1 iff one entity occurance relates to another entity occurance.

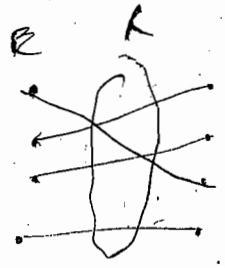
and max Cardinality = n iff one entity occurance relates to multiple entity occurances.



min Card (E,R) = 0 Partial
max Card (E,R) = 1

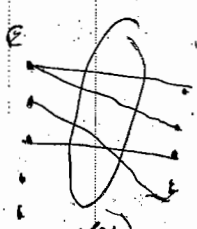


min Card (F,R) = 0 Partial.
max Card (F,R) = 1



min = 0 (Partial)
max = 1

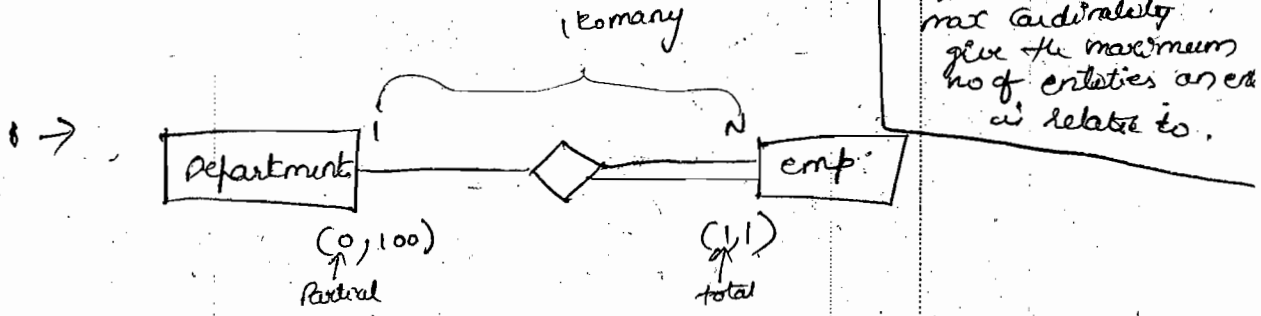
min = 1 (Total)
max = 1



min = 0 (Partial)
max = 2 (N)

min card = 1 (Total)
max card = 1

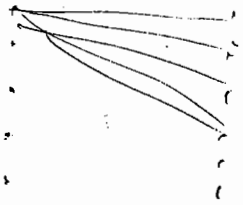
min Cardinality give partial & total
max cardinality give the maximum no of entities an entity is related to.



— partial participation

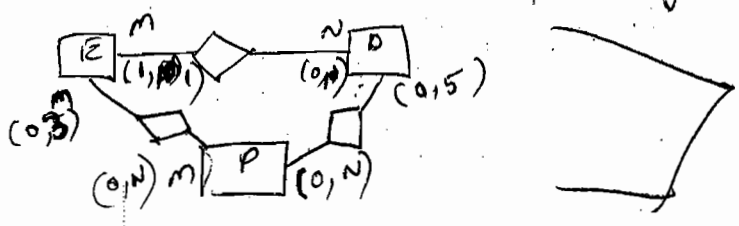
═ total participation

~~A dept~~ 1 to many ⇒ one dept can have many employees.



Description :-

Employees may belong to different dept and participate in different projects. minimum number of employees in a project should be '100' and a single emplee can participate max upto 3 project and max of 5 dept can participate in project. Then every employee should belong to a dept. but they need not participate in the project; (note cardinaly ratio can take only max value).

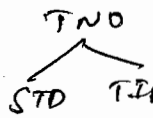


Attributes:- stored, derived → attributes → simple, composite; attributes → single valued, multi valued.

Simple: It will have a single atomic value and cannot be divided further.

Ex:- Roll no, Company name, job code etc.

Composite: It is the combination of many atomic values and will have complete meaning if we combine all the attributes



Single valued: A attribute that have always a single value.

Roll no, Gender, Date of birth

multivalued: An attribute that will have multiple values.

Ex:- qualification of person, Tel id of a person, Email id of a person

Stored: An attribute that supplies the value is called a stored attribute.

1) DOB AGE

2) I/D R/D duration time.

3)

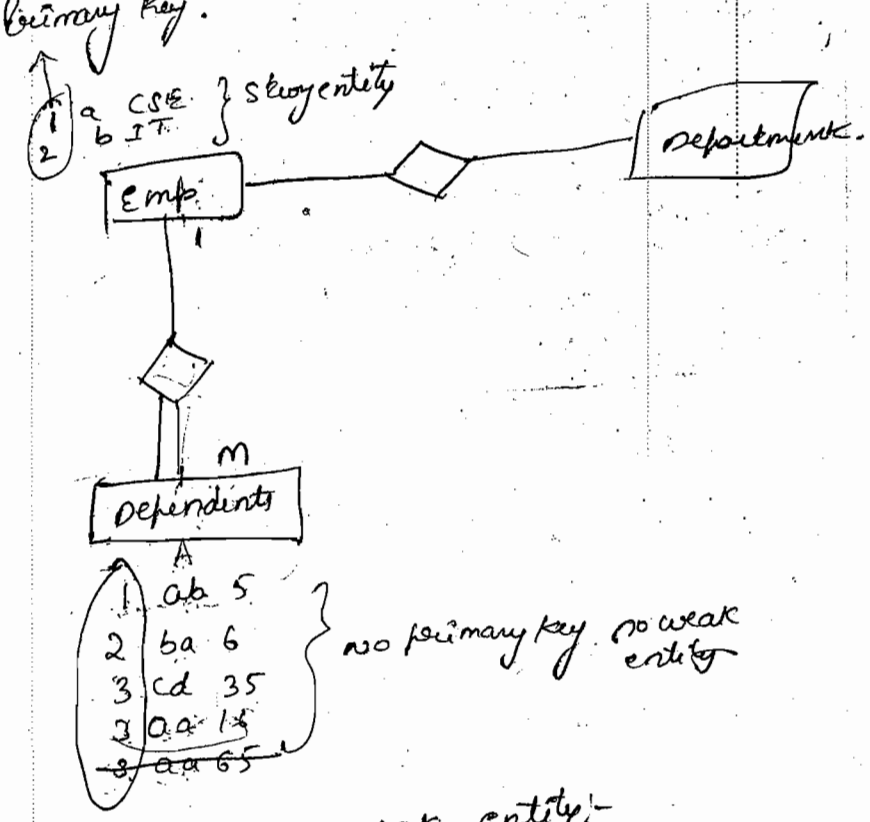
RNO	Name	DeptNO
1	a	1
2	b	1
3	c	2
4	d	3
5	e	1

DNO	DName	DS
1	CSE	3
2	IT	4
3	ECE	1

An attribute can be derived from a single attribute. (1)
An attribute can be derived from multiple attributes. (2)
An attribute can be derived from a seperate table (3).

Special type of entities:-

1) Strong entity 2) weak entity 3) Associative entity.

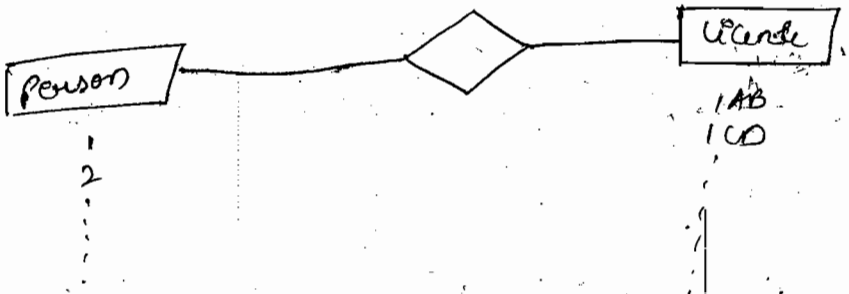


Characteristics of Strong and weak entity:-

(i) They will b identified by primary keys & dependencies.

(ii) The relationship b/w strong and weak entity is one to many.

(iii) the participation of strong entity is partial but weak entity is total.



here even the license is depending on person entity, license entity cannot be called as a weak entity because it will have its own primary key.

∴ classification of entities may be done by using

primary keys.

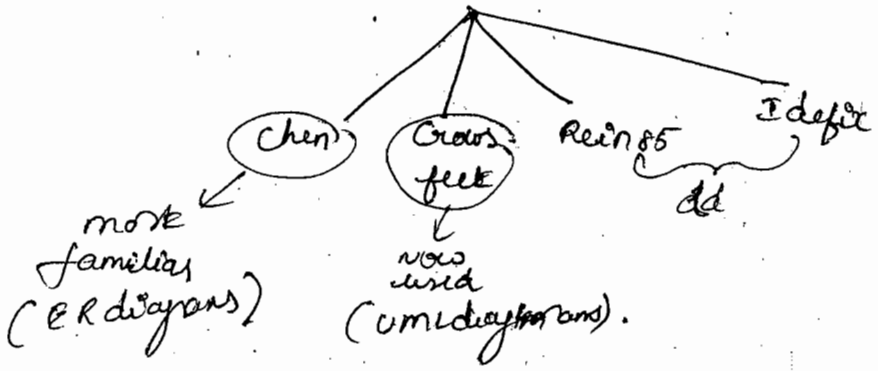
Associative entity:

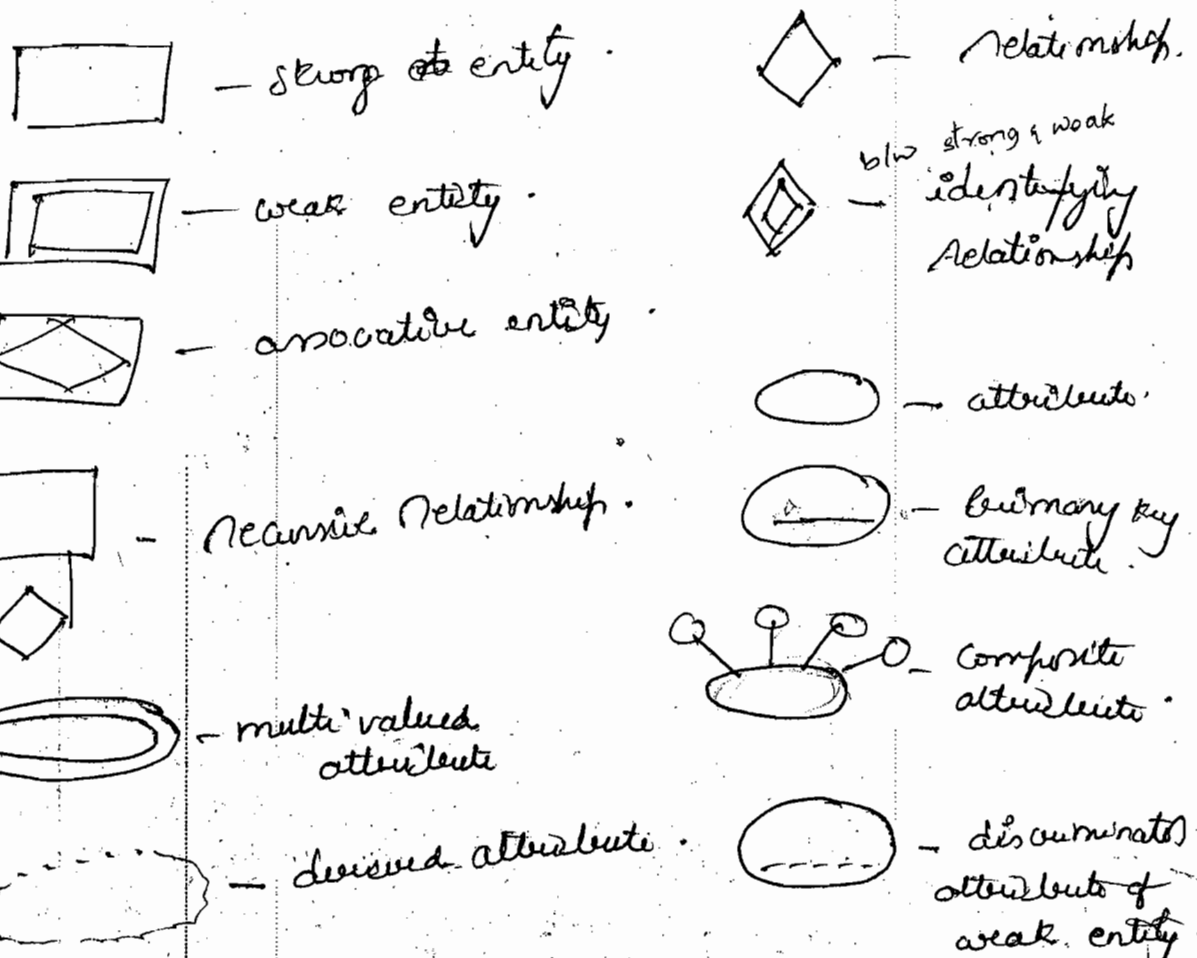
The entity that looks like a relationship but not a relationship is known as associative entity.

Ex: Person, Product, Sale.

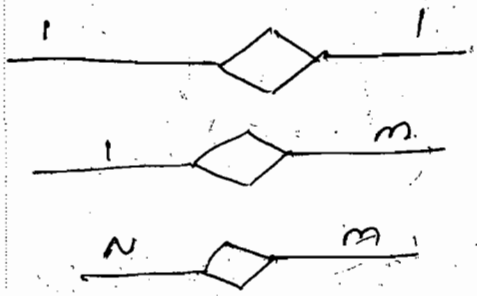
Person and product are entities but Sale is looking like a relationship b/w these two but it is a seperate entity with its own attributes.

notations available to give pictorial representations





The attribute of the weak entity that is ~~added to~~ combined with primary key attribute of the strong entity to declare the combination of a primary key of the weak entity is known as discriminator attribute.



→ now ER diagrams are to be converted to tables of objects dependent on RDBMS or OORDBMS respectively.

→ ~~we have~~ Conversion of ER diagrams into tables (7 step criteria)

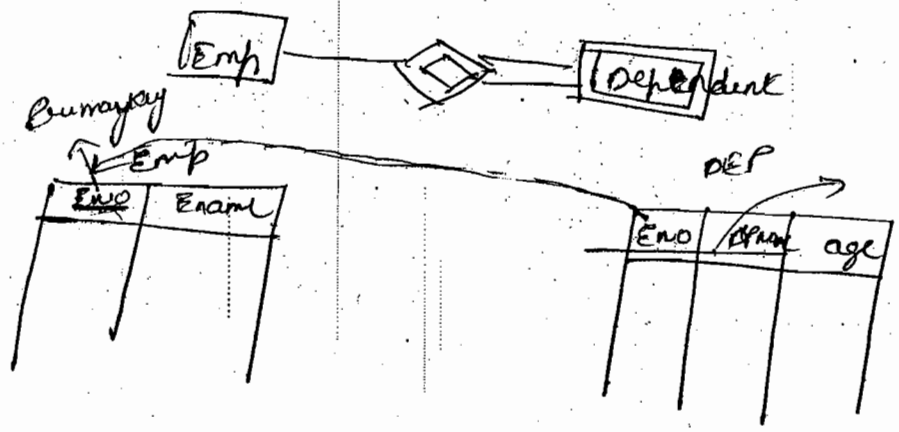
Step 1:- Conversion of strong entities:

a) For each strong entity in ER diagram, create a seperate table with the same name.
b) Create all simple attributes.
c) Break the composite attributes into simple attributes and create them.
d) Choose a primary key for the table.

Step 2:- conversion of weak entities:

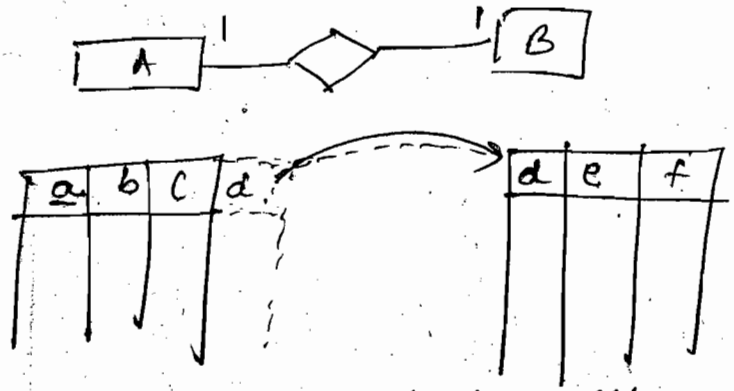
a) For each weak entity create a seperate table with the same name.
b) Include primary key of the strong entity as a ~~for~~ foreign key in this table.
c) Select the primary key attribute of strong entity and discriminator attribute of weak entity

and declare them as primary key.

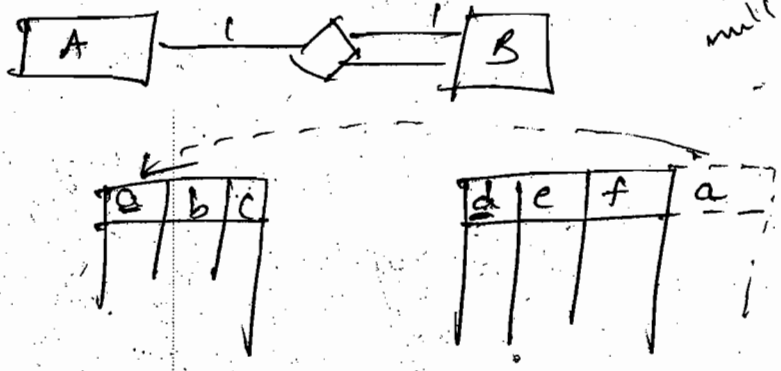


Step 3:- Conversion of 1 to 1 relationships

a) For each 1 to 1 relationship, say b/w A and B tables, modify either A or B to include the primary key of other table as a foreign key.

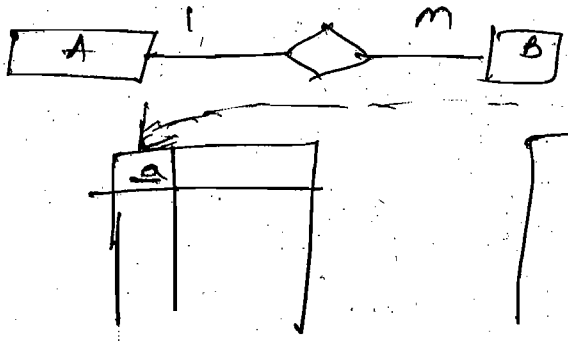


b) If one of these table either A or B has total participation, then modification must be done only on that side.



Step 4:- Conversion 1 to many relationships:

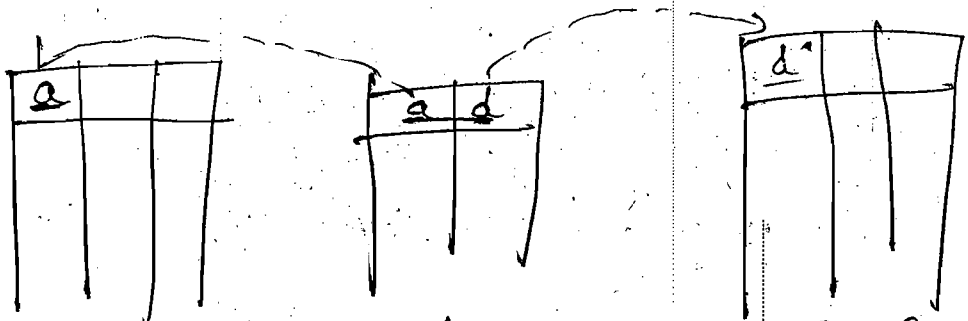
a) For each 1 to many relationship b/w A and B modify many side relation to include primary key from one side as a foreign key.



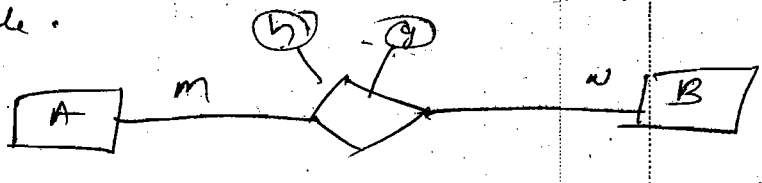
Eg: stud no. 1 many

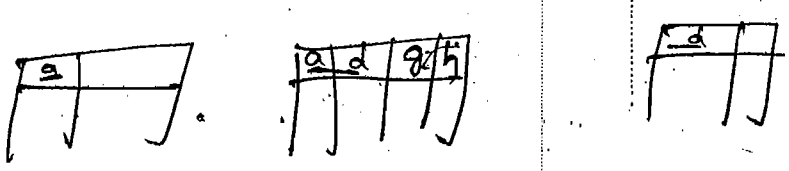
Step 5:- Conversion of many to many relationship

a) For each many to many relationship say b/w A and B, create a seperate table and include primary key of both the tables as foreign key.



b) If relationship is having one or more attributes, these must also be included in the table.





Step 6:-

Conversion of n-ary relationship.

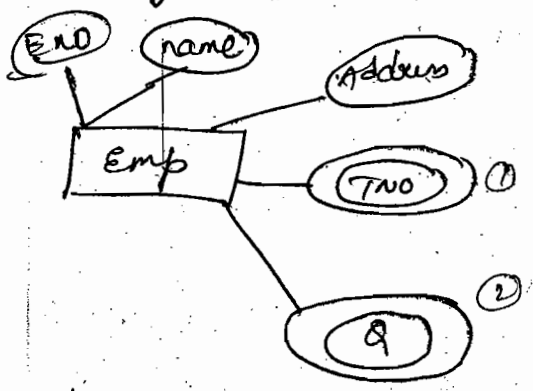
a) For each n-ary relationship, create a seperate table and include primary keys of all other entities as a foreign key.

b) If the relationship has some attributes those must also be included

Step 7:- conversion of multivalued attributes:

a) For each multivalued attributes, create a seperate table, then include all of its simple attributes.

b) ~~Select~~ Include the primary key of the original table as a ~~primary~~ foreign key.



Emp

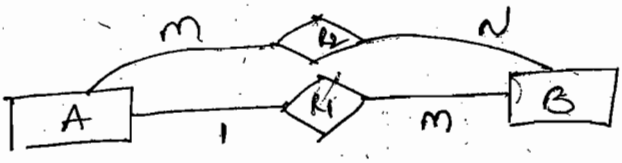
ENO	name	DNO	Sty	City

(1)

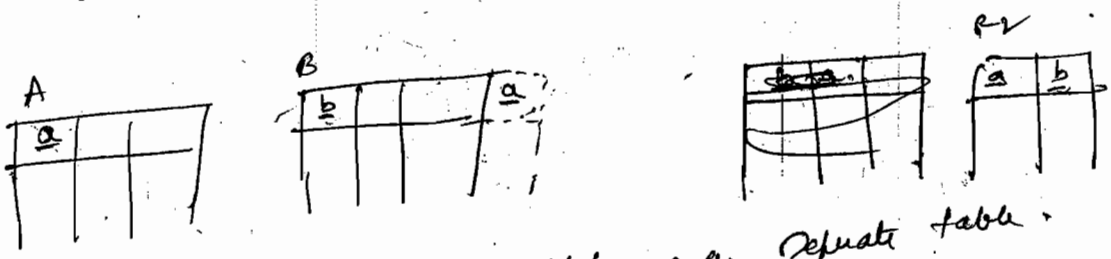
Empno	TNO

(2)

ENO	Q

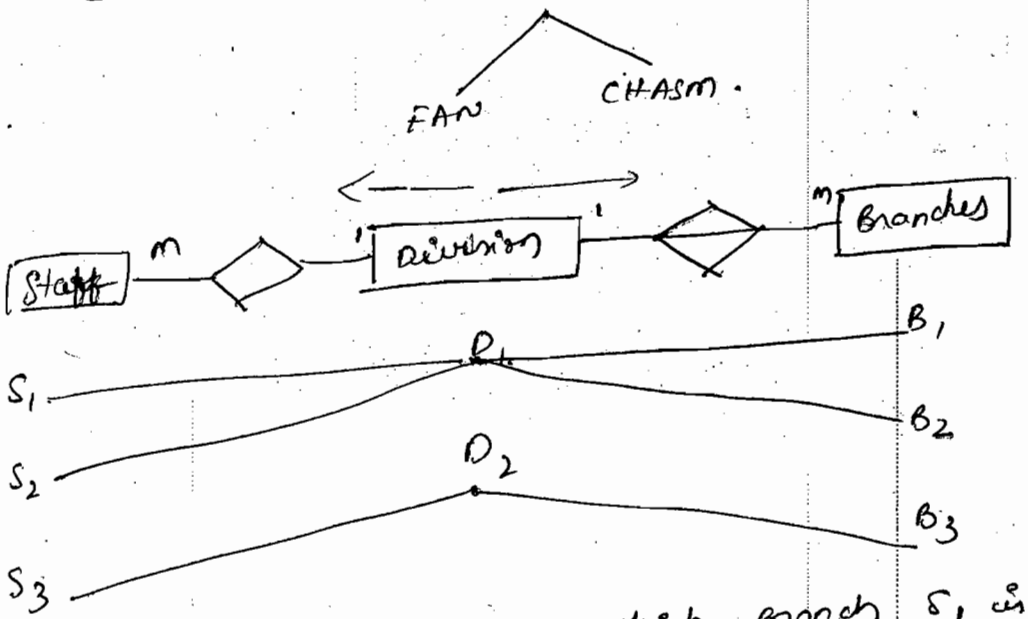


min : ? 3 A B R_2
max : ? A B R_1 R_2



R_1 can also be represented as a relate table.

→ Traps in ER diagram :-



we cannot infer as in which Branch S_1 is working.

It is called FAN trap.

This problem is
Identification of the fan trap :-

A fan trap may exist if two or more one to many relationships fan out from the entity.

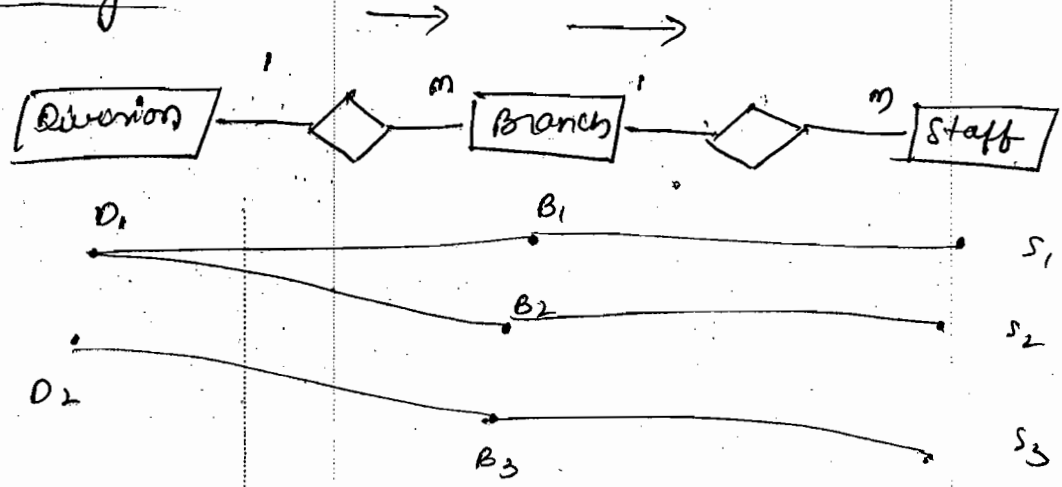
⟷

Information cannot be retrieved to the full extent if there is a fan trap.

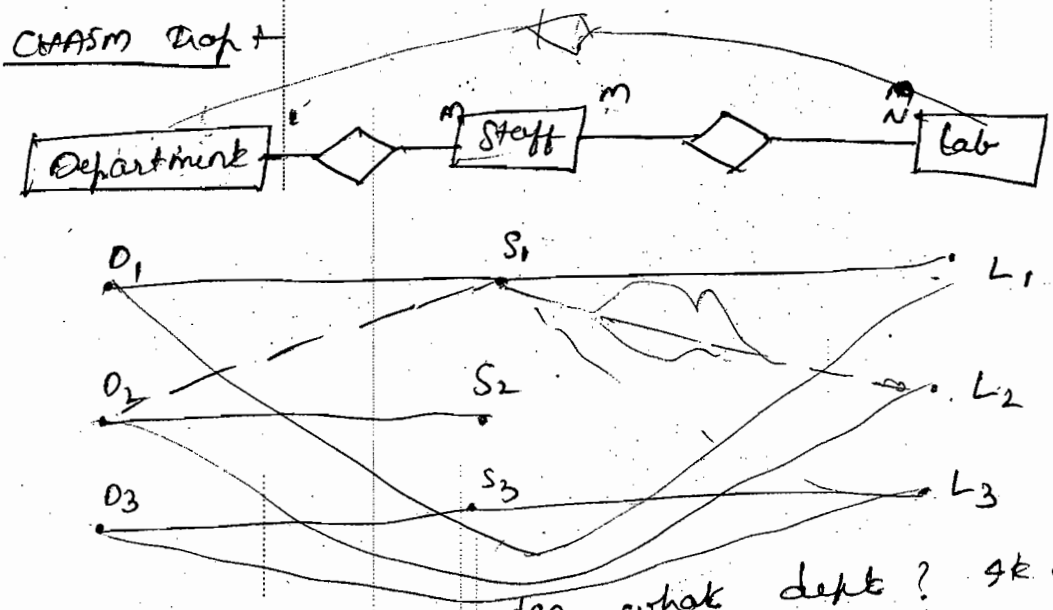
Solution:-

If there is a fan trap, rearrange the entities and redraw the ER diagrams.

Rearrangement:-



Chasm Trap:-



Lab 2 comes under what dept? It is chasm trap.

Identify chasm trap: A chasm trap may occur if one or more relationships with a cardinality of '0' forming a path b/w the related entities.

Solution:- Create an addition relation ship b/w these related entities.

The advantages ~~and~~ of Relational ER diagrams :-

(i) Conceptually it is very simple
(ii) Better visual representation.
(iii) It is an effective communication tool among users, domain experts and data base designers.
(iv) It is tightly integrated with relational model, so converting ER diagrams to tables is very simple.

Disadvantages :-

(i) Limited Constraint & specification
Ex:- min cardinality. max = n, even if more than n

(ii) ~~Loss~~ of loss of information content.
Ex:- FAN, CHASM trap (may be)

(iii) Limited relationship representation.
only binary

EER diagrams :- (Extended Entity relationship)

= ER diagrams + Extentions

These extensions are sub classes, Super classes, Categories, attribute and relationship instances.

TOP down

E
↓
R
↓
A
↓
ER diagram
↓
Tables

Bottom up

A R.NO Name FNO BookNO Pub...
↓
R (Functional dependencies)
↓
E
↓
normalization
↓
Tables

→ In BU, To identify the relationships among the attributes we use functional dependencies.

→ In TD, we use ER diagrams.

→ In BU, we use functional dependencies and normalization.

→ when project size is big, we use TD and for small we use BU.

→ BU is most powerful but in practice very difficult to apply.

→ In actual practice, we go for mixed approach.

E
↓
R
↓
A
↓
ER
↓
Tables (5 or 6)
↓
evaluate using normalization
↓
Tables (8 or 10)
(3 or 4)

Functional dependency:

→ we look at relationships among the attributes.

RNO name age Branch.

RNO → name
name → Rollno ⇒ RNO → name, age, Branch.
RNO → age
RNO → Branch

RNO name age Branch FNO Fname DepNO. Depname

① RNO → name, age, branch, depno.
② depno → d name.
③ F.NO → F.name. (faculty no.)
④ F.NO → depno, depname.

~~From the test~~

a → b
× b → a

a	b
1	2
2	2
3	2
4	5
6	5

FD's are depending

The properties of FD's:-

→ The FD's will deal with one-one relationship among the attributes, some times it may be one-many (or) many to one.
→ FD's must be defined on schema, not on instances of the schema.

ab → cd
a → b
ab →

→ FD must be a non trivial & completely non trivial.
A trivial dependency is the one where RHS is a subset of LHS

Ex for trivial : $abc \rightarrow bc$

→ An FD is non trivial if atleast one of the RHS attributes are not part of LHS attributes.
An FD is completely non trivial if none of the RHS attributes are part of the LHS attributes.

Ex for non trivial : $abc \rightarrow cde$

Ex for complete NT : $abc \rightarrow def$.

→

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

identify the FD's that holds good.

$A \rightarrow B$

$B \rightarrow C$

$C \rightarrow A$

$AC \rightarrow B$ (means A itself can determine B and even we add C also it will no change).

$BC \rightarrow A$.

trivial is a

RHS

RHS

→

	A	B
t_1		
t_2		
①	If t_1 and t_2 agree here	Then they must agree here
②	If t_1 and t_2 disagree here	They may agree & disagree here.

A = RollNo B = Name

1	a
2	b
3	a

1	a
1	a

RollNo → Name

→

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

B and RNo

$A \to B$

$A \to C$

$B \to C$

$AC \to B$, $AB \to C$ } A is already able to determine B and C

$B \not\to A$

$C \not\to A$

→

A	B	C
10	b_1	c_1
10	b_2	c_2
11	b_4	c_1
12	b_3	c_4
13	b_1	c_1
14	b_3	c_4

$A \not\rightarrow B$

$B \rightarrow C$

$C \not\rightarrow B$

$B \not\rightarrow A$

$C \not\rightarrow A$

$AB \rightarrow C$

$BC \not\rightarrow A$

$AC \rightarrow B$

→

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

$A \rightarrow B$

$B \not\rightarrow A$

$C \not\rightarrow A$

$B \not\rightarrow C$

$C \rightarrow B$

$D \rightarrow B$

$A \not\rightarrow D$

$B \not\rightarrow C$

$C \not\rightarrow D$

$D \rightarrow C$

→

$A \rightarrow B$

$B \rightarrow C$

$AD \rightarrow F$

$= F_1$

$A \rightarrow C$

$= F_2$

$F_1 + F_2 = F \rightarrow$ nötig.

Semantics ?

Functional dependencies (F_1) can be identified by using semantics from the problem domain and it can be called as F_1 set.

Additional FD's (F_2) can also be derived from F_1.

Then total FD's = $F_1 + F_2$

Before making a move to the normalization process, we have to have both F_1 and F_2.

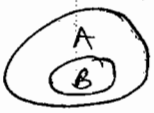
To identify F_2, the following methods can be used.

(i) Inference rules,

(ii) closure set of attributes.

Inference rules:

Reflexive rule: If B is a subset of A then, $A \rightarrow B$.



$A \rightarrow B$.

Transitive rules: If $A \rightarrow B$ and $B \rightarrow C$, then we can say $A \rightarrow C$.

Decomposition Rule: If $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$.

Augmentation Rule: If $A \rightarrow B$

$\downarrow$

$AC \rightarrow BC$

Union rule: If $A \rightarrow B$ and $A \rightarrow C$ then $A \rightarrow B, C$.

(applicable only if LHS's are same).

Composition rule: If $A \rightarrow B$, and $C \rightarrow D$ then $AC \rightarrow BD$

Self determination: $A \rightarrow A$.

By armstrong

Ex: F_1 : $A \to B$
$B \to C$
$C \to D$
$D \to E$
$D \to H$
$E \to F$
$F \to G$
$G \to H$.

By using inference rules

~~F~~
F_2:

By transitivity

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

$B \to C$
$C \to D$ $\} B \to D$

By union rule:

$D \to E$
$D \to H$ $\} D \to EH$

$B \to EH$ (if we mention it it will cause problems).

∴ ~~Inference~~

Even though this method is very powerful, it is time consuming and error prone.

∴ we need an alternative.

closure set of attributes:

Algorithm to identify the closure sets:

Step 1: let 'x' be a set of attributes that will become ~~the closure~~ finally a closure.

Step 2: Repeatedly search for some FD such that LHS is available in x, then add RHS of that FD to 'x'. (if it not available)

Step 3: Repeat step 2 as many times as necessary until no more attributes can be added to 'X'.

Step 4: The set 'X' after no more attributes can be added will become the closure of the attributes.

→ $A \rightarrow B$
$BC \rightarrow DE$
$AEG \rightarrow G$

Compute AC^+

$X = AC$
$= ABC$
$= ABCDE$

$AC^+ = ABCDE$.

→ $A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

Compute B^+.

$X = B$
$= BD$

$B^+ = BD$.

→ $AB \rightarrow C$
$BC \rightarrow AD$
$D \rightarrow E$
$CF \rightarrow B$

Compute AB^+

$X = AB$
$= ABC$
$= ABCD$
$= ABCDE$

$\therefore AB^+ = ABCDE$.

→ The application of ⊕ closure set of attributes:

(i) It is used to identify the additional FD's.

(ii) It is used to indentify keys (candidate keys or primary keys)

(iii) It is used to identify equivalence of FD's.

(iv) It is used to identify irreducible set of FD's & canonical cover of FD's.

(i)

F_1	F_2
$AB \to C$	~~$AB \to BDE$~~
$BC \to AD$	$AB^+ = ABCDE$
$D \to E$	
$CF \to B$	

Identification of additional FD's by any closure sets:

→ To determine whether a particular FD $AB \to DE$ can be derived from the original set of not, compute AB^+ from the original FD's, if AB^+ includes 'DE', then $AB \to DE$ is true, and will become additional FD.

→ Repeatedly check all additional FD's of to find out F_2 set.

F_1

→ $A \to BC$
$CD \to E$
$E \to C$
$D \to AEH$
$ABH \to BD$
$DH \to BC$

Find whether $BCD \to H$ is not.

$BCD^+ = BCD$
$= BCDE$
$= ABCDEH$

∴ DE is true.

Whether ~~AB#~~ ∴ AED → C

~~AB#~~ = $AED^+ = AED$

$= ABCDE$

Identification of keys with closure set of attributes:-

Different keys:

(i) Super key
(ii) Primary key
(iii) Candidate key
(iv) Foreign key
(v) Surrogate key
(vi) Composite primary key.

Primary key (not null)

RNO Name age

Candidate keys

VNO ENO Type Vom cost.

Select any one as primary key.

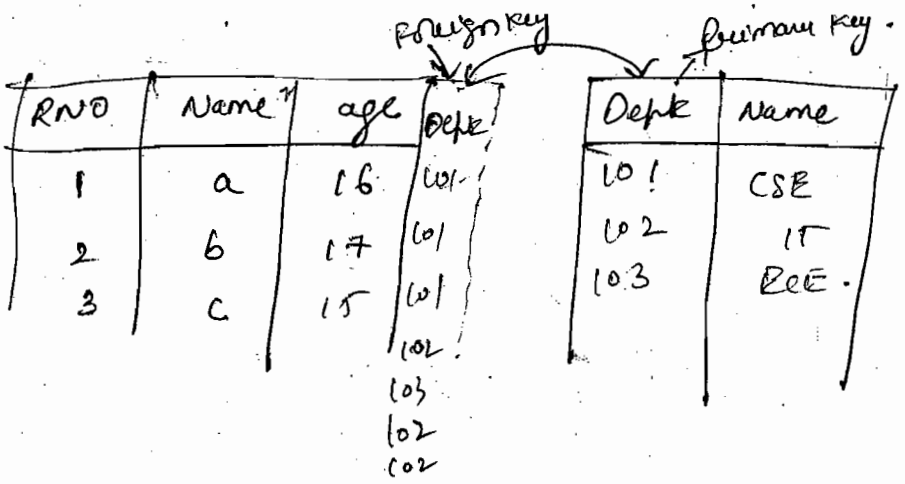
Student table

Book table

RNO (Foreign key)	BNO (PKey)	I/date	R/date	Fine	S.Key
1	101	1/1	1/2		1
1	102	11	11		2
2	101	1/2	1/3		3
1	101	1/4	1/5		4
					5
					1

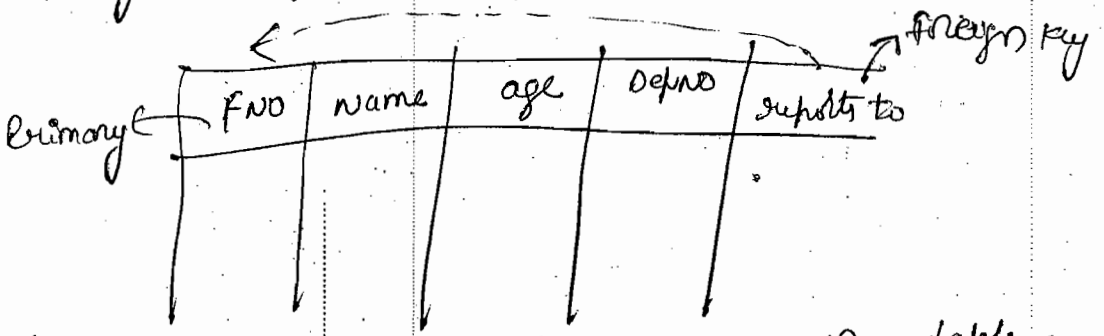
Surrogate key.
It is a system generated key.

If ~~I date~~ one cannot take and return on the same day then (RNO BNO Idate) can only form Composite key.

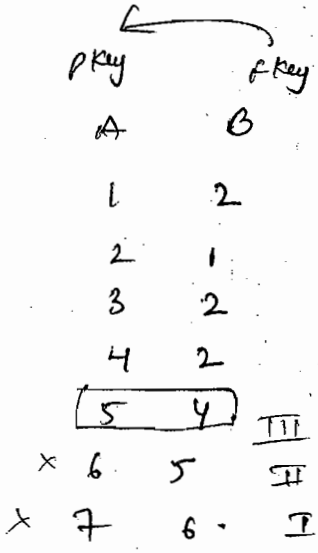


Foreign key is used for referencial integrity.

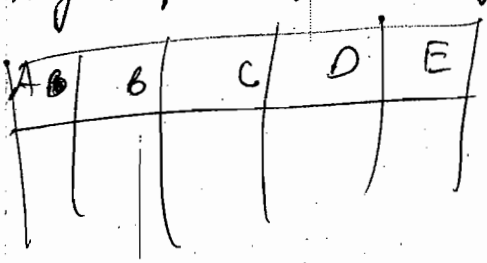
always foreign key refers a primary key of some table.



There can be any no of foreign keys in a table.



Superkey: Any superset of primary key is a super key.



$A \rightarrow B$

$B \rightarrow DE$

$BC \rightarrow CE$

units all LHS attributes

whether ~~AB# → D~~ . AED → C

~~AB#+~~ $AED^+ = AED$

$= ABCDE$

Identification of keys with closure set of attributes:-

Different keys:

(i) Super key.
(ii) Primary key.
(iii) Candidate key.
(iv) Foreign key.
(v) Surrogate key.
(vi) Composite primary key.

Primary key (not null)
↑
RNO Name age

Candidate keys
VNO ENO Type Xom cork.
select any one as primary key.

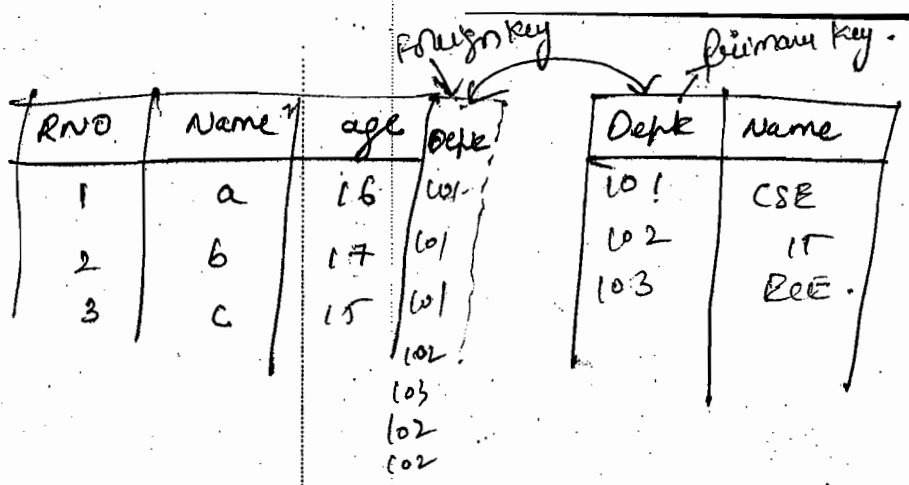
Student table / Book table

↑ Foreign key / F.key

RNO	BNO	I/date	R/date	Fine	S.key
1	101	1/1	1/2		1
1	102	"	"		2
					3
2	101	1/2	1/3		4
1	101	1/4	1/5		5

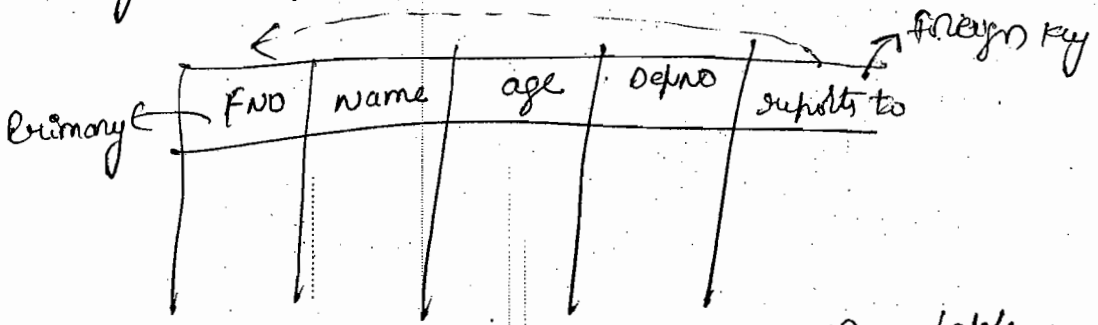
→ Surrogate key. It is a system generated key.

If ~~I date~~ one cannot take and return on the same day then (RNO BNO Idate) ~~can~~ will form composite key.

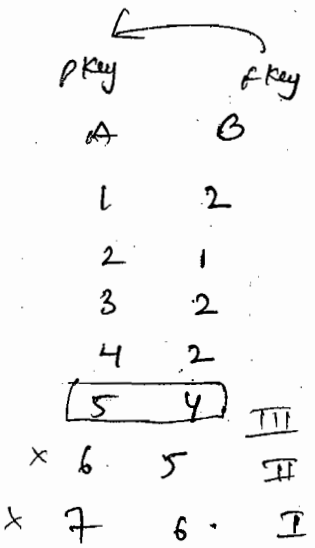


Foreign key is used for referencial integrity.

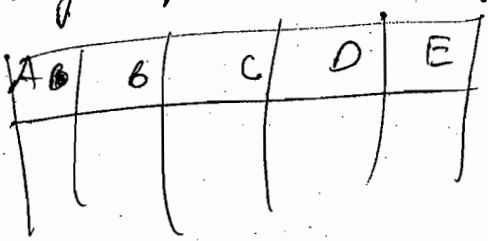
always foreign key refers a primary key of some table.



There can be any no of foreign keys in a table.



Superkey: Any superset of primary key is a super key.



$A \rightarrow B$

$B \rightarrow DE$

$BC \rightarrow CE$

unite all LHS attributes

→ If we combine all the LHS attributes of the functional dependencies, then we will get a Super key.

→ Key of a table must be subset of the Superkey

Identification of keys with the help of closure set

→ Compute closure for the attributes or combination of attributes on the LHS of functional dependencies.

→ If any closure includes all the attributes, then that can be declared as a key of the table.

→ $A \rightarrow BC$
$CD \rightarrow E$
$E \rightarrow C$
$D \rightarrow AEH$
$ABH \rightarrow BD$
$DH \rightarrow BC$

Find the keys from the dependencies!

$A^+ = ABC$

$E^+ = EC$

$D^+ = ADEH$
$= ACDEH$
$= ABCDEH$

∴ D can be a key. then we can have many superkeys

AD, BD, CD, ——

→ A → B.
BC → E
ED → A

$A^+ = AB$

$BC^+ = BCE$

$ED^+ = EDA$
$= EDAB$

$CDE^+ = CDE$
$= ABCDE.$

Candidate

$ACD^+ = ACD$
$= ABCD$
$= ABCDE.$

$BCD^+ = BCD$
$= BCDE$
$= ABCDE.$

→ R = ABCDE

AB → C
CD → E
DE → B

$AB^+ = ABC$

$CD^+ = CDE$
$= CBDE.$

$ACD^+ = ABCDE.$

$DE^+ = DE$

ABD
ADE
ACD.

→ R = ABCDEFGHIJ

AB → E
A → DE
B → F
F → GH
AD → IJ

$A^+ = ADE$
$= ADEIJ$

$B^+ = BF.$
$= BFGH.$

$D^+ = DIJ$

$AB^+ = ABC$
$= ABCDEFGHIJ.$

→ 16) R = ABCDEFGHIJ

$AB \rightarrow C$

$BD \rightarrow EF$

$A \rightarrow GH$

$A \rightarrow I$

$H \rightarrow J$

ABD

Ans:-

17) AB, BC, BD

18) A, CD

19) ABCD

→ R = ABCDEFG → moved in RD.

$A \rightarrow BC$

$B \rightarrow DE$

$C \rightarrow BC$

A is our key according to attributes of F

Now AFG will be the actual key of the table.

→ Equivalences of functional dependencies:-

Ex1

10)

F: $A \rightarrow C$

$AC \rightarrow D$

$E \rightarrow AD$

$E \rightarrow H$

Reqs

G = $A \rightarrow CD$

$E \rightarrow AH$

Take set F and check 'G' is covered or not

$A^+ = AC$

$= ACD$

$E^+ = E$

$= EAD$

$= EADH$

both $A \rightarrow CD$ and $E \rightarrow AH$ are covered.

2. G can be derived from 'F'; Hence G is Covered by F → (2).

Step 4: Take set 'G' and check whether F is covered from 'G':

$A^+ = ACD$ $(A \to C)$

$AC^+ = ACD$ $(AC \to D)$

$E^+ = E$
$= AH$
$= ACDH$ $(E \to AD, E \to H)$

Hence F is ~~equivalent~~ covered by 'G' $\to$ ②.

From ① and ②, F and G are equivalent.

∴ G is preferred one.

$F_1 : G \Rightarrow A \to CD$
$E \to AH$.

Extend F_2 $F_1 + F_2 \to$ normalize.

→ F: $B \to CD$
$AD \to E$
$B \to A$

G: $B \to CDE$
$B \to ABC$
$AD \to E$

Take F: $B^+ = ABCDE$ $(B \to CDE, B \to ABC)$

$AD^+ = ADE$ $(AD \to E)$.

Take G: $B^+ = ABCDE$ $(B \to CD)$

$AD = ADE$. (

Prefer : F.

covered

→ Irreducible set of FD's & Minimal set of FD's & Standard form of FD's & Canonical form with no redundancies:

From semantics F_1 then F_2 then normalization.

Once F_1 and F_2 are identified, some of the functional dependencies among them may have redundant attributes either on LHS & on RHS. These attributes must be eliminated before making a move to the normalisation process.

And it is a four step procedure:

Step I: have single attributes on RHS

Step II: evaluate attributes on LHS by using some of the irreducible principles then evaluate attributes on RHS by any closure set of attributes

Step IV: Apply the union rule to get the original FD's.

→ Find the irreducible set from the following FD's:

(i) ① $AB \to C$
② $C \to A$
③ $BC \to D$
④ $ACD \to B$
⑤ $BE \to C$
⑥ $CE \to AF$
⑦ $CF \to BD$
⑧ $D \to EF$

Step I:-

$AB \to C$
$C \to A$
$BC \to D$
$ACD \to B$
$BE \to C$
× $CE \to A$
$CE \to F$
$CF \to B$
$CF \to D$
$D \to E$
$D \to F$

$C \to A$
By augmentation rule
$CE \to AE$
By decomposition
$CE \to A$
$CE \to E$.
∴ $CE \to A$ is redundant in given set, because
$C \to A \Rightarrow CE \to A$.

augmentation

$CF \rightarrow B \Rightarrow CF \rightarrow BC$

~~$CF \rightarrow B$~~

~~$CF \rightarrow$~~

$BC \rightarrow D \Rightarrow \therefore CF \rightarrow D$.

$\therefore CF \rightarrow D$ is redundant

But during this is the way we go, it is a time consuming process and error prone.

$\therefore$ Instead of using inference rules, we can make use of closure set of attributes to identify the duplicates.

Ex: 20:

$A \rightarrow B$
$C \rightarrow B$
$D \rightarrow ABC$
$AC \rightarrow D$

(There can be several minimal covers for a set of dependencies).
(Fourth - multivalued dependency
fifth - join dependency).

Step 1:

① $A \rightarrow B$
② $C \rightarrow B$
③ $D \rightarrow A$
④ $D \rightarrow B$
⑤ $D \rightarrow C$
⑥ $AC \rightarrow D$

Step 2:

Remove $A \rightarrow B$ and Compute A^+ from the ~~rem~~ ②, ③, ④, ⑤, ⑥ if we get B we can say that A determines B without having FD $A \rightarrow B$ and no $A \rightarrow B$ otherwise it is not redundant.

$A^+ = A$
$\therefore A \rightarrow B$ is not redundant.

Remove $C \to B$ and Compute C^+ from ① ③ ④ ⑤ ⑥

$C^+ = C$. ∴ it is not redundant

Remove $D \to A$ and Compute D^+ from ① ② ④ ⑤ ⑥

$D^+ = DBC$

∴ it is not redundant.

Remove $D \to B$ and compute D^+ from ① ② ③ ⑤ ⑥.

$D^+ = DACB$

∴ $D \to B$ is redundant.

Remove $D \to C$ and compute D^+ from ① ② ③ ⑤ ⑥.

$D^+ = DAB$

∴ it is not redundant.

Remove $AC \to D$ and compute AC^+ from ① ② ③ ⑤ ⑥

$AC^+ = AC$
$= ABC$

∴ it is not redundant.

~~Now~~

Step ③ :-

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

remove AC in Left FD

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

$C^+ = CB$

$A^+ = AB$

$C^+ = CD$
$= ACD$
$= ABCD$ ✗

Remove C in FD

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

$A^+ = ABD = ABCD$. ✓

∴ we need both A and C on LHS.

step 4 :-

$A \rightarrow B$

$C \rightarrow B$

~~A~~

$D \rightarrow A$

$D \rightarrow C$

$AC \rightarrow D$

apply union for the common attributes

$A \rightarrow B$

$C \rightarrow B$

$D \rightarrow AC$

$AC \rightarrow D$

Ex 21

$AB \rightarrow C$

$C \rightarrow B$

$A \rightarrow B$

step ① :-

① $AB \rightarrow C$

② $C \rightarrow B$

③ $A \rightarrow B$

step ② :-

① $AB \rightarrow C$

② $C \rightarrow B$

③ $A \rightarrow B$

Remove ① and compute AB^+ from 2 and 3.

$AB^+ = AB$

It is not redundant.

Remove ② and compute C^+ from ① and ③.

$C^+ = C.$

It is not redundant.

Remove ③ and compute A^+ from ① and ②.

$A^+ = A.$

It is not redundant.

Step 3:

$AB \to C$
$C \to B$
$A \to B$

$A^+ = ABC$
$B^+ = B$

Remove A

$B \to C$
$C \to B$
$A \to B$

$B^+ = BC$
not equal

Remove B

$A \to C$
$C \to B$
$A \to B$

Compute $A^+ = ABC$
equal
∴ we can remove 'B'.

Step ii:

$A \to C$
$C \to B$
$A \to B$

Transitive
⇒

$A \to BC$
$C \to B$
is irreducible set.

22) FD's are $ABD \to AC$
$C \to BE$
$AD \to BF$
$B \to E$.
Find minimal set.

Ans

$AD \to CF$
$C \to B$
$B \to E$

Step 1:

1) $ABD \to A$
2) $ABD \to C$
3) $C \to B$
4) $C \to E$
5) $AD \to B$
6) $AD \to F$
7) $B \to E$

Step 2:

Remove A①.

$ABD^+ = ABD$

Remove ②

$BD^+ = BDE$.
$AD^+ = ADBFEC$
$AB^+ = ABE$

Step 3 :

$ABD \to C$
$C \to B$
$AD \to B$
$AD \to F$
$B \to E$

Remove A X

$BD^+ = BDCE$.
B ✓
$AD^+ = ABDFEC$
D X
$AB^+ = ABCE$.

Remove B from I FD.

$\Rightarrow$ 1) $AD \rightarrow C$
2) $C \rightarrow B$
3) $AD \rightarrow B$
4) $AD \rightarrow F$
5) $B \rightarrow E$.

Remove (1) $AD^+ = ADBFE$.
Remove (2) $C^+ = C$
Remove (3) $AD^+ = ADCFB$.
Remove (4) $AD^+ =$

$\Downarrow$

$AD \rightarrow C$
$C \rightarrow B$
$AD \rightarrow F$
$B \rightarrow E$

Remove A X
$D^+ = DCB$
not equal
Remove D X
$A^+ = ACB$
not equal

Remove A X
$D^+ = DF$.
Remove D X
$A^+ = AF$.

$D^+ = D$
$A^+ = A$

$\Downarrow$

Step IV:

$AD \rightarrow C$
$C \rightarrow B$
$AD \rightarrow F$
$B \rightarrow E$

$\Rightarrow$

$AD \rightarrow CF$
$C \rightarrow B$
$B \rightarrow E$.

$\Rightarrow$ [redacted]

23) (Any)
$A \rightarrow BC$
$AE \rightarrow H$
$C \rightarrow D$
$D \rightarrow G$
$E \rightarrow F$

24) (Any)
$BC \rightarrow AE$
$C \rightarrow D$
$A \rightarrow F$
$F \rightarrow G$.

classification of FD's :

(i) Partial FD :
(ii) Transitive FD
(iii) Full FD.

classification of Dependences :

1) Functional dependency. (2NF, 3NF, BCNF)
2) multivalue dependency (4NF)
3) Join dependency (5NF, PJNF)
4) Template dependency.
5) Inclusion dependency (dependences among the data bases).

ONE ER digrams are converted to tables 4NF and 5NF are not needed.

Partial FD :-

A FD in which one of more non key attributes are functionally depending on a part of the primary key.

Ex:- $R = ABCD$

$AB \rightarrow C$

$B \rightarrow D$

Key = AB , $\therefore B \rightarrow D$ is a partial dependency.

Transitive dependency :-

A dependency in which there is a relationship among non key attributes i.e one non key attribute is determined by another non key attribute.

Ex:- $R = ABCD$

$AB \rightarrow C$

$B \rightarrow D$ (PD)

$C \rightarrow D$ (TD)

Full functional dependency:-

A functional dependency $X \rightarrow Y$ is a full functional dependency, if removal of any attribute from X the dependency does not hold any more.

Ex:- If $AB \rightarrow C$ is full functional dependency, then we cannot remove A or B. i.e C is fully dependent on AB.

If we are able to remove A or B, then it is not full functional dependency.

Normalization:-

→ It is the process of evaluating logical design by applying the series of rules & normalization is a formal process based on primary key and FD's to decide what attributes should be grouped together in different tables.

1st NF: All columns in table with single value, no multiple vals, all entries in col of same kind.

Second normal form:-

A Table is said to be in the second normal form, if it is already in the I normal form and it is free from partial functional dependencies.

The table is automatically in second normal form.

If primary key consists only one attribute & all attributes of primary key of table consist only 2 attributes.

25) $R = ABCDEFGHIJ$.

$AB \rightarrow C$

$A \rightarrow DE$

$B \rightarrow F$

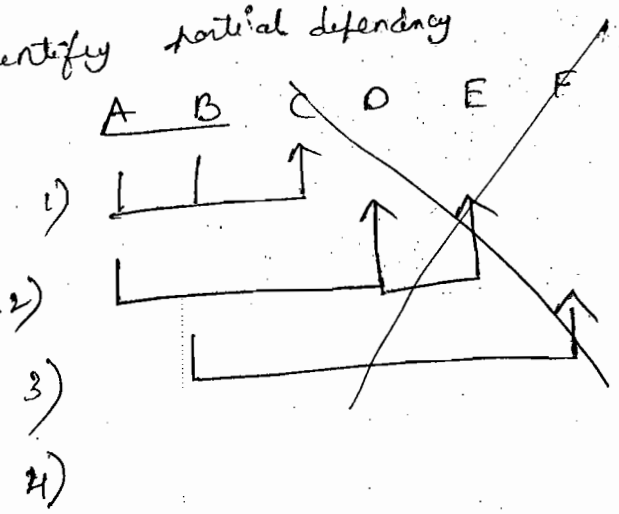
$F \rightarrow GH$

$D \rightarrow IJ$.

Decompose the above upto II normal form.

Identify key: $AB^+ = ABCDEFGHIJ$.

Identify partial dependency



Confirm and circle them.

Find $A^+ = ADEIJ$.

$B^+ = BFGH$

~~A BC D E F G H I J~~ A B C D E F G H I J.

$R_1 = \underline{A}DEIJ$

$R_2 = \underline{B}FGH$

$R_3 = \underline{AB}C$

R_1: $\underline{A}DEIJ$

R_2: $\underline{B}FGH$.

$R_3 = \underline{AB}C$.

If there is any partial dependency, remove partially dependent attributes from original table place them in a separate table along with a copy of its determinant.

26) A B C D E

Book (Book_title, Auth0_name, book_type, Price, author_affiliation, F Publisher)

$A \rightarrow FC$
$C \rightarrow O$
$B \rightarrow E$

Decompose upto II normal form:

Identify the key:

$AB^+ = ABCDEF$
$\therefore AB$ is key.

$A^+ = ACDF$
$B^+ = BE$.

A B ~~C D E F~~.

R_1 : $\underline{A}CDF$. (in 2nd normal form because only one key attribute)
R_2 : $\underline{B}E$. (only two attributes).
R_3 : $\underline{AB}$. (all are key attributes)

27) $R = ABCDE$. Primary key = AC.

$B \rightarrow E$
$C \rightarrow D$
$A \rightarrow B$

decompose upto II normalization.

$A^+ = ABE$
$C^+ = CD$

R_1 : $\underline{AB}E$
R_2 : $\underline{C}D$
R_3 : $\underline{AC}$

R = ABCDEFGHIJ.

AB → C
BD → EF
AD → GH
A → I
H → J

Primary Key : ABD

~~A+ = AI~~
~~B+ = B~~
~~D+ = D~~

AB+ = ABCI
ABD+ = BDEF
AD+ = ADGHJI

A B ~~C~~ D ~~E~~ ~~F~~ ~~G~~ ~~H~~ ~~I~~ ~~J~~

R1 : AB CI .
R2 : BD EF
R3 : AD GHJI .
ABD (II normal form because all attributes in primary key).

AB C~~I~~
A+ = AI. B+ = B.
AI   AB C

BDE F
B+ = B   D+ = D
no partial dependency
BD EF

AD GH~~I~~J
A+ = AI   D+ = D
AI   ADGHJ

ABD

∴
AI
AB C
BD EF
AD GHJ
ABD

III normal form:-

26)

→ A table is said to be in III normal form, if it is already in the second normal form and free from transitive dependencies.

→ A table is automatically in III normal form if it one of following hold:

(i) If table consists of two attributes.

(ii) If II normal form table consists only one non key attributes.

25) A B C D E F G H I J

$AB \rightarrow C$

$A \rightarrow DE$

$B \rightarrow F$

$F \rightarrow GH$

$D \rightarrow IJ$

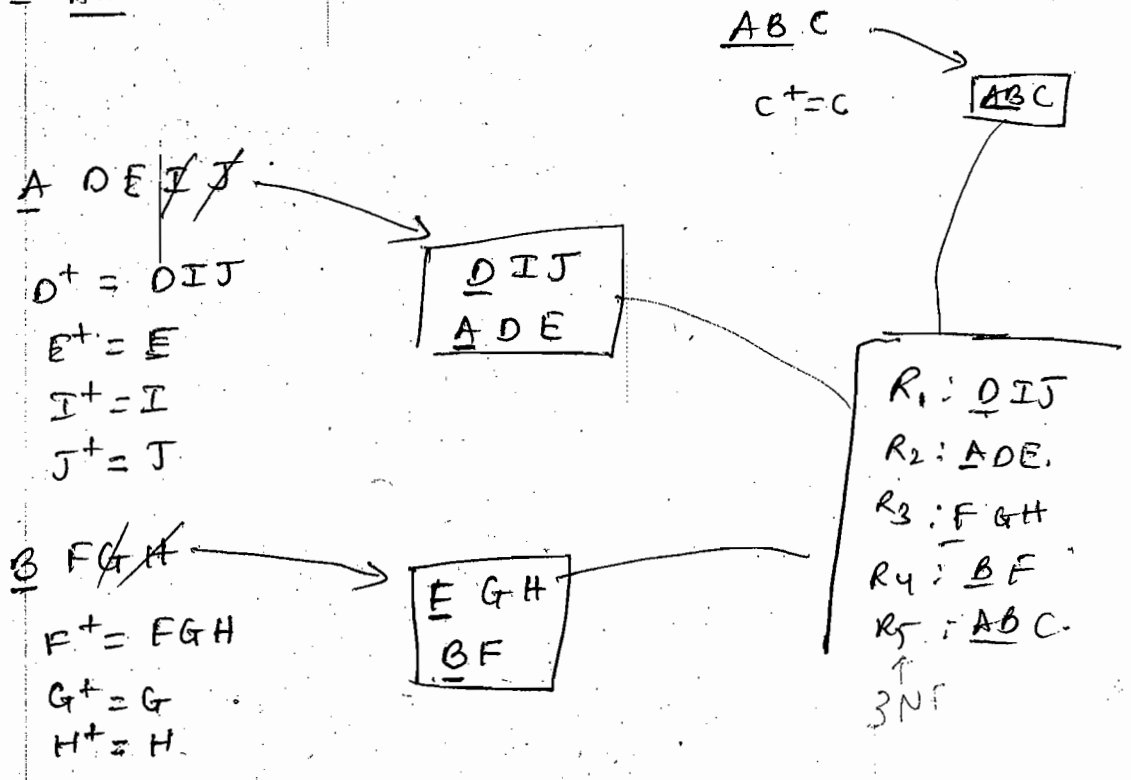
II normal

R_1 : A D E I J

R_2 : B F G H

R_3 : A B C

27)



26) A B C D E F

$A \rightarrow FC$
$C \rightarrow D$
$B \rightarrow E$.

II normal table:

R_1 : A C D F
R_2 : B E
R_2 : A B.

} III normal form because only two attributes.

A C ∅ F

$C^+ = CD$
$D^+ = D$
$F^+ = F$

C D
A C F

∴

C D
A C F
B E
A B

III normalized tables.

27) R = A B C D E

$B \rightarrow E$
$C \rightarrow D$
$A \rightarrow B$

R_1 : A B E < B E, A B

R_2 : C D
R_2 : A C

} II normal

28) R: A B C D E F G H I J

FDs: $AB \rightarrow C$
$BD \rightarrow EF$
$AD \rightarrow GH$
$A \rightarrow I$
$H \rightarrow J$

2 NF

AI ——— AI
ABC ——— ABC
BDEF ———
AD GHJ
ABD

29) a) $B \rightarrow C$
$B \rightarrow D$
$A \rightarrow C$
$A \rightarrow D$

b) $C \rightarrow D$ (or)
$D \rightarrow C$

→ If $X \rightarrow A$ is a dependency, then the table is in the III normal form if one of the following conditions exists:

(i) If X is a super key (Because if x is should not be non key attribute and not also not be part of a key, if it is a part, then it is partial dependency).

(ii) If A is a part of key.

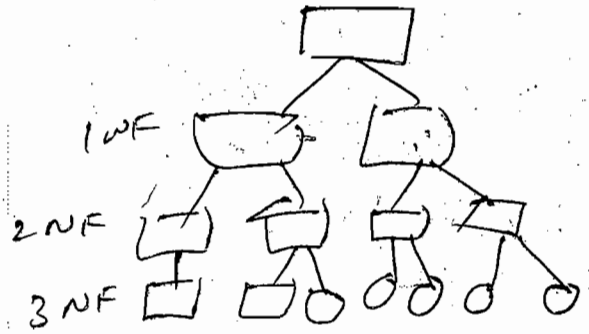
→ If ~~x~~ $X \rightarrow A$ is a dependency, then the table is said to be not in II normal form if the following

(i) if X is a proper subset of some key (Partial dependency)

(ii) If X is not a proper subset of key. (non key)

→ Dependency preserving and lossless join properties :-

If we apply the normal forms, to the universal table, it may be splitted up into different fragments.



at any stage, if we combine the fragments (denormalization) it should give the original table in terms of columns and rows, and it will be described as the following properties:

(i) lossless join property :- (mandatory)

(ii) dependency preserving property (optional)

(i) lossless :-

→

R

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

R_1

A	B
a_1	b_1
a_2	b_2
a_3	b_1

R_2

B	C
b_1	c_1
b_2	c_2
b_1	c_3

A decomposition of a relational schema R into different relations like R_i where $i = 1, 2, \ldots n$ and it is said to be lossless join decomposition if natural joins of all decompositions gives the original relation.

$$R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R) \ldots \ldots \ldots \pi_{R_n}(R) \rightarrow (1).$$

In this case $R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R)$

Suppose the natural join of all decompositions look like $R \subseteq \pi_{R_1}(R) \bowtie \pi_{R_2}(R)$, we have spurious tuples.

$\pi_{R_1}^{(R)} \bowtie \pi_{R_2}^{(R)}$ =

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

Formula ① is time consuming and every person do use another simple checking. It is as follows

$R_1 \cap R_2 \rightarrow R_1$ we can now determine lossless property easily

If R is decomposed into R_1 and R_2, then we dont have any losses if

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

or

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

i.e common attribute is a key for atleast one table.

Ex: $R = ABC$

$A \rightarrow B$

$R_1(A, B)$

$R_2(A, C)$

Check whether decomposition is lossless & lossy.

'A' is primary key for R which is common attribute and so it is lossless.

Defend

Dependency preserving property:

If original table is decomposed into multiple fragments then some how, we suppose to get all original FD's from these fragments. A decomposition is said to be decomposition preserving if

$$(F_1 \cup F_2 \cdots \cup F_n)^+ = F^+.$$

$F_1 \cup F_2 \cdots \cup F_n$ → Functional dependencies in R_1

F^+ → original (R).

Ex: Consider a relation 'R' with A, B, C, D attributes decomposed into $R_1(ABC)$ $R_2(C, D)$

Original dependencies: 1) $A \to B$
2) $A \to C$
3) $C \to D$

check whether it is decomposition preserving or not

1 and 2 from R_1, 3 from R_2 and so it is preserving.

FD $R_1(ABD)$, $R_2(B, C)$, 1 from R_1 but 2 and 3 cannot be preserved.

It is lossy also because 'B' is not a key so we should not go for this decomposition because it is lossy.

we can go for any decomposition even if dependency preserveney does not hold but lossless holds.

Note: Even if decomposition is not preserving functional dependencies we can use them as in information systems because we very rarely we are expecting all those attributes together.

31)

BCNF

A table is said to be in BCNF, if it is already in 3NF and every FD will have a Candidate key on the LHS & all determinants are the candidate keys.

The differences b/w 3NF and BCNF:

3NF	BCNF
1) It concentrates on primary key	1) It concentrates on candidate keys.
2) Redundancy is high compared to BCNF	2) Redundancy is low compared to 3NF.
3) It may preserve all the dependencies	3) It may not preserve the dependencies.
4) A dependency $x \to y$ is allowed in 3NF if 'x' is a super key of y is a part of some key.	4) A dependency $x \to y$ is allowed if x is a superkey.

→ The Third normal form may not deal the following cases in a proper way.

(i) If table consists two & more candidate keys

(ii) If candidate keys overlap, i.e atleast one common attribute among the keys.

doubt

3

31) 1) $C \to D$, $C \to A$, $B \to C$

a) Key : B.

B A C D

b) 2NF

c) $C \to D$, $C \to A$ } T(D).

3NF Tables : C D
C A
B C.

all Determinants are keys ∴ It is in BCNF.

2) $B \to C$ $D \to A$.

a) Key : B D

b) B D A C 1NF

Partial dependencies are there :
∴ It is only in 1NF.

2NF : B D B C D A.

It is also in 3NF.

all determinats are super key ∴ It is in BCNF.

3) $ABC \to D$, $D \to A$.

$ABC^+ = ABCD$.

B C D D A

a) Key : (A B C) and B C D

b) A B C D , ~~3NF~~ It is in 3NF.

c) It is not in BCNF because D is not a super key.

A B C D, A D

even though the decomposition satisfies BCNF it causes lot of redundancies.

Normalization will effectively identify redundancies within the table. but it may not identify among the tables.

→ Identification of Redundancy among the tables:

Non key attributes should not be repeated in the decomposed tables.

4) $A \to B$,
$BC \to D$ 2NF.
$A \to C$

a) Key: A

b) A | BCD
2NF

c) A BC , BC D

A and BC are determinants and are superkey
∴ it is in BCNF.

5) $AB \to C$
$AB \to D$
$C \to A$
$D \to B$

a) Key: (AB) CD, BC, AD

b) AB C D

3NF

Determinats AB, C is superkey but C, D are not.

∴ it is not in BCNF.

~~ABCD, C A, D B~~

C → A and D → B are not allowed on AB CD

BCNF as per the rules.

∴ TO satisfy BCNF, we can have

AB CD, C A, D B as tables

but it will have lot of redundancy among the tables, ∴ it causes problems.

32) R = ABCDEFGH.

AB → CEFGH
A → D (PD)
F → G (TD)
FB → H
HBC → ADEFG
FBC → ADE.

AB is key:

2nd normal form

R_1: AD R_2: ABCEFGH.

3rd normal form

R_1: AD | R_2: ABCEFGH | R_3: FG

A → D | AB → CEFGH | F → G.
FB → H (BCNF)
~~HBC → ADEFG~~.

BCNFF

R_1: AD | R_2: AB CEF

35)

1) a) key BD

b) BC and AD

Since there is no common attribute, the natural join of the two table leads to multiple spurious tuples.

∴ It is not a good decomposition

2) AB→C, C→A, C→D ; decompose into ACD, BC.

a) Keys : AB, BC, CB

b) (C is a key in ACD and so it is lossless decomposition)

There is a common attribute i.e C and it can be a key in I table hence it is lossless; but it is not preserving FDs.

3) A→BC, C→AD, ABC, AD.

a) Key: A, C

b) The common attribute A can be key for both tables and so it is lossless.

4) lossless Key: A.

5) lossy. Key: A.

36) ABCDE ; FD: AB→CDE, A→C, D→E.

Key : AB

II NF

AB→CDE , A→C

AB→CDE

D→E

A→C

II NF

AB , DE , AC

D→E A→C.

It is upto BCNF.

37) $R = ABCDE$.

key : AB.

AB → DE A → C

A → C

II Normalform AB E AC D. $A^+ = ACD$

III

AB E AC C D.

Since in BCNF

38)

The limitations of Normalization

→ Normalization cannot identify the ~~limitations~~ redundancies among the tables, it can do it with in the tables.

→ Normalization leads to the so many table fragments ∴ during the query Retrieval process, the response of system is slow.

→ The table fragments may not have the real world meaning ∴ difficult to understand and maintain the system.

Advantages:

→ It reduces anomalies like insertion, deletion and modification.

→ maintaining data integrity is very easy

⇒ Normalization increases the speed of query retrieval.

Denormalization:

The process of combining multiple table fragments selectively is known as Denormalization.

we will go for denormalization under following circumstances

(i) If majority of the queries are data Retrieval ~~queries~~

(ii) If a particular query requires join of multiple tables Ex: while preparing balance sheet.

(iii) If update problems are not important but performance is important.

SQL

1) Select * from department where DNO = 101 Group by AGE.

2) Select RNO, max(marks) from student.

3) select min(distinct AGE) from student.

4) Select * from Odors where max(price) > 1000

5) Select Ordno, Ordname, Ordquantity from Odors having count(Ordno) > 10.

6) Select Rollno, name, Age from student order by 1, 2.

7) Select Rollno, name, Age from student order by name ascending, age descending.

8) Select name A, Age B, rollno C from student having C > 15 order by A, B.

9) Select order no, Agent no, Agent name from Odors where customer name between C and B.

10) Select * from student S1 where 2 = (Select Count(Rollno) from student S2 where S1.marks ≤ S2.marks.

Ans:

1) Group by and '*' should not be used.

2) Add Group by Rollno)

3) distinct is not necessary.

4) Aggate cannot appear

5) Group by needed.

8) always cannot be used in having.

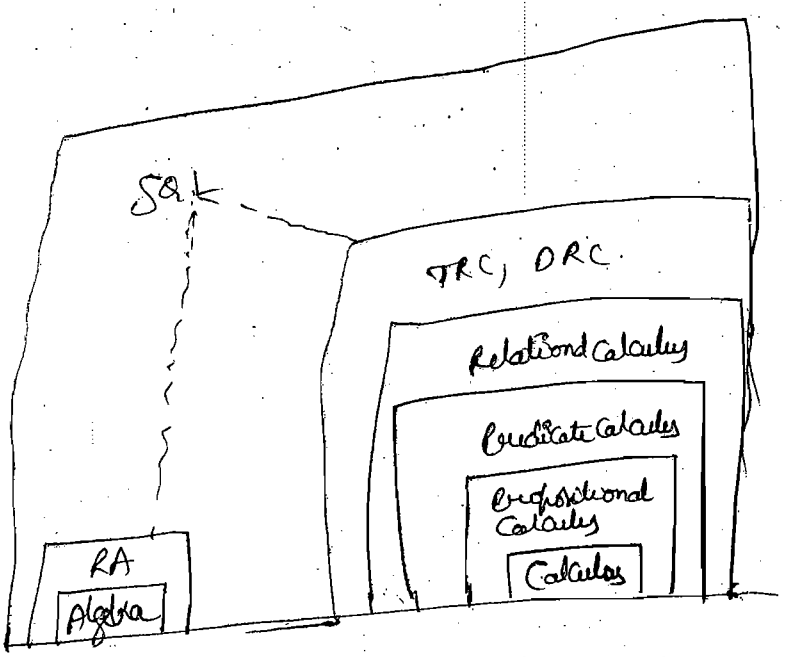
9) lower value should be first.

10) It gives the second max marks student.

Classification of database languages:

1) Procedural languages
   Ex:- SQL.
   Input are tables and o/p are tables.
2) Graphical oriented languages
   Ex:- QBE.
3) menu driven ~~4th generation languages~~ languages.
   Ex:- connected websites.
4) Fourth generation languages. (English like languages).

The roots of SQL:



SQL Standards:

SQL1 also SQL86
SQL2 also SQL92
SQL3 also SQL99

Components of SQL:

(i) DDL (ii) DML (iii) DCL.

where clause:

Based on the filtering action, the SQL Queries are divided into 5 groups

(i) Queries that uses basic search operations.
(ii) Queries that uses range operations
(iii) Queries that uses comparision operations.
(iv) Queries that uses pattern ~~matching~~ matching.
(v) Queries that uses nulls and not nulls.

Ex:

(i) select * from student where Roll no = 101.

(ii) select * from student where % > 50 and branch = IT

(iii) select * from student where % > 50 and branch = IT and year = 2.

Find the details of students who belongs to CSE, IT, ECE, EEE and MECH.

select * from student where Branch = CSE or branche IT or ECE or EEE or MECH.

more efficient:
select * from students where Branch in (IT, CSE, EEE, ECE, MECH)

select * from student where branch not in (civil).

Comparision:

select * from student where age > 15;

select * from student where city != hyderabad and branch = IT and year = 2 and age > 15.

Pattern m

% : 9t d

_ : 9t

A% : 9

%A : C

%A% :

%ACE%

A---

Searching

search

→ Find th
with

· Select

→ Find all

select

→ Find the
IT le

Selec

→ Find
Comm

Sel

Pattern matching:

%: It indicates sequence of zero or more characters.

_ : It indicates a single character.

A%: First letter is A followed by any no of characters.

%A: Last character is A preceded by 0 or more characters.

%A%: A character A may appear anywhere in the word.

%ACE%: A word 'ACE' may appear anywhere in the pattern

A---: A four letter word starting with 'A'.

Searching special characters using escape symbol (/)

searching for % in a word.

% / % %

↓

Pattern we are searching for.

→ Find the details of student whose name starts with 'S' and from branch 'CSE'.

Select * from Student where name like 'S%' and branch = 'CSE'

→ Find all the student details who opted network protocol subject

select * from student where subjects like '% network protocol %'

→ Find the details of student whose I letter is 'A' and IV letter is 'C' and last letter is D.

Select * from student where name like 'A--C%D'.

→ Find the details of students who are in CSE branch coming from hyd and whose 3rd letter is 'A'.

Sol: select * from student where name like '--A%' and city = hyd and branch = 'CSE'.

→ Range queries → (b/w and not b/w operator).

→ Find details of Student whose marks b/w 500 and 600.

Select * from Student where marks between 500 and 600.

→ Select * from Student where marks between 350 and 400.
104 and 107. (here b/w is inclusive)

→ Select * from Student where marks not between 350 and 400.
101, 102, 103, 105, 106, 108, 109, 110

→ Select * from Student where name between 'A and K'
'a' and 'b'..

103, 104, 108, 110, 111

111 K 19 M IT 3 500.

(Here b/w accept from A, AB, ABinas..... K)
Ka, KB, K — X are not included.

→ while executing the b/w query with characters, it will take lower boundary and stops exactly at the upper boundary.
∴ 102 and 106 will not appear as an o/p.
but they will appear in not b/w case.

→ If any name consists only a single letter 'k', that will be included in the b/w case..

→ Select * from Student where marks between 400 and 350
Results in Syntax error because lower bound must be mentioned first in the b/w query.

Null C
→ Find
Select
→ Find
addre
Sel
Aggreg
main
Star
→ Selec
→ Selec
max
non
→ Sele
→ Sele
Stu
qu
→ Sele
→ Sel
dis
will
→ Sele
→ Sel
→ Selec
→ Selec

Null queries :-

→ Find the details of students who are free from fine.

Select * from Library where fine is null.

→ Find the details of all Students who are having email address

Select * from Students where email is not null.

Aggregate functions :-

maximum, minimum, sum, average, count, count(*), Standard deviation (stdev), variance (var)

→ Select max(name) from student. ✓

→ Select max(marks) from student. ✓

max and min can be used on both numeric and non numeric columns.

→ Select sum(name) from student ✗

→ Select sum(marks) from student ✓

Sum and ave must be applied only on numeric queries.

→ Select distinct sum(distinct marks) from students.

→ Select min(distinct name) from students.

distinct has no effect with min and max, but will have a considerable impact on sum and average.

→ Select count(Roll no) from students. ✓

→ Select sum(mark*10) from student ✓

→ Select ave(mark1+mark2) from student ✓

→ Select ave(mark1, mark2) from student ✗

Aggregate functions operate on a single column of a table and on expressions and returns a single value; but cannot work on multiple columns simultaneously.

→ ~~Select~~ select * from Student where max(marks) = 500 ✗

→ select * from Student order by max(marks) ✗

→ select * from Student Group by count(roll no) ✗

aggregate functions can be used only in the select and having clauses, but not in where, order by and Group by clauses.

→ select roll no, max(marks) from Student ✗

If Select clause consists ~~con~~ aggregate functions ~~of~~ along with ordinary columns, then it must be associated with group by clause.

Group By:

→ Find the maximum marks among students?

Sol: Select max(marks) from student.

→ Find the details of student who got maximum marks.

Select * from student where marks = (select max(marks) from student);

→ Find the student details who got second maximum & 3rd maximum & in general nth maximum?

(used to generate the group aggregate and works on non aggregate columns in the select columns).

Table

Roll no	marks
101	450
102	300
103	500
104	480
105	490
106	300

Sol: Select

Q: Fi

less

Exa

Two

Q: Fi

Q:

Restri

1) If

Select * from student S_1 where

(2nd second maximum) 2 = (Select Count(S_2.marks) from student S_2 where S_1.marks <= S_2.marks);

S_1 (alias of students)

101	450
102	500
103	480

S_2 (another alias of student)

101	450
102	500
103	480

Q: Find the details of the student who got less than the average marks of the class.

Ans: select * from student where marks < (select Av (marks) from student);

Ex: Rollno:
100
100
—
—
200

Count(*) returns '6' (it counts null values)

Count(Rollno) — 4 (it counts the duplicate values).

Two ways for average calculations: Avg(marks), sum(marks)/count(Roll

This is more accurate

Q: Find the the total no of students in each and every branch?

Select branch_id, Count(Roll no) from student group by branch_id;

Ex:

101	a	IT
102	b	CSE
103	c	I

output:

IT	60
CSE	60
ECE	120
EEE	40

Restriction for Group By:

1) If the SELECT list Consist a non-aggregate column with an aggregate function, then the query must be associated with a group by clause and it should Consist all non aggregate columns in the group by list.

Ex: select age, bid, count (roll no) from student group by bid, age

) we cannot use a column in group by clause that is not available in select list.

select roll no, count (bid) from student group by age ✗

3) we cannot use aggregate function in group by clause.

Q: Find out the branch detail whose total strength is more than 50 students.

select bid, Count (roll no) from student group by bid having count (roll no) >50

Note: Group By is applied on independent rows.
Having is applied among groups.

Q: Find the details of student who paid more than 20 Rupees as a total fine?

Roll	Date	Fine
1	1-1-08	100
1	2-1-08	200

select roll no, sum (fine) from library group by roll no having sum (fine) > 20.

How to know the execution time of a query:

1. before execution of a query at command prompt write "set timer on".
2. execute query
3. Then at the command prompt write "set timer off"

where clause	Having clause
1. It is used to filter rows	1. It is used to filter groups.
2. Aggregate functions cannot be used	2. Aggregate functions can be used.
3. It is mandatory (i.e no alternative for where)	3. It is optional (i.e the results given by having can also be applied without having clause)

4. The col... clause ... select ...

ORDER ...

→ It is ... way ...
→ It a... group ...

Ex:- Se...
Ex:- S...

Ex:- Se... S...

Ex:- selec... ord...

correct ...

Ro...

Ex: Fir... the d... as... selec... ha...

Note:-

Ex:- Se... S...

4. The columns appear in where clause need not appear in select list.

4. The columns in where clause must either be aggregated in select list & appear in group by clause.

ORDER BY:

→ It is used to sort out the output in a proper way like descending order & in ascending order.
→ It is applied on dependent columns sort not on group of columns.

Ex:- select * from student order by name;

Ex:- select roll no, name, age from student where age > 15 order by (1, 2) → place order

Ex:- select roll no, name, age from student where age > 1
order by roll no descending, name ascending.

Ex:- select branch id, count (roll no) group by branch id
order by count (roll no) ×.

Correct form:
select branch id, count (roll no) a group by branch id
order by a;

Ex: Find total strength of each branch and display the details of branches where the total no of students is more than 45 and display them in ascending order

select bid, count (roll no) a from student group by bi
having x >= 50 order by a;

Note:- aliases are not allowed in having clause.

Ex:- select roll no, max (marks) from student group by
roll no having roll no in (101, 110, 115);

Sub Query :- A query embedded in another query is called sub query.

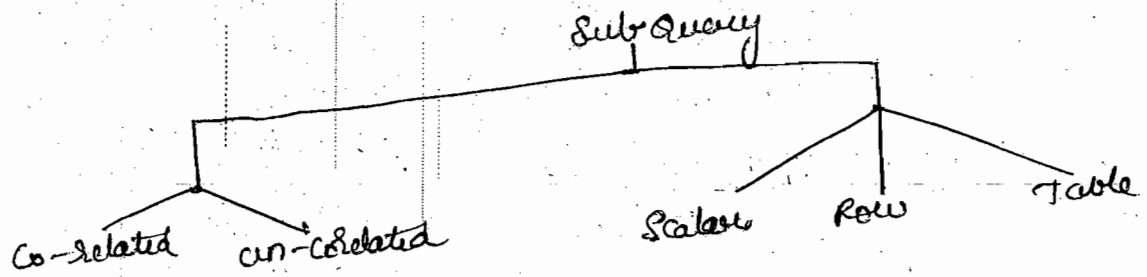
Characteristics :-

- The inner query will run first and substituted results in outer query.
- The variable from outer query can be used in the inner query but reverse is not true.
- The subquery can be used in the where and having clauses and sometimes in from clause also.
- The inner query will run always only once. If it is independent of our query, then it is called uncorrelated sub query.
- Inner query will run sometime as many times as the no of rows available in the outer query. If it refers a variable in outer query. Then it is called correlated sub query.

Ex:- Select * from student where marks = (select max(marks) from student) (un-correlated sub query).

Select * from student S_1 where 2 = (select count(S2.marks) from student S_2 where S_1.marks <= S_2.marks) (Correlated sub query).

Classification of Sub query



Scalar Sub-Query :-
It returns only one row and one column.

Row sub Query :-
It returns multiple columns but only one row.

Table sub Query :-
It returns multiple rows and multiple columns.

The possible ways of writing Sub-queries:

→ By using In-predicate (Between, like, Is null cannot be used with the sub queries).
→ By using quantified Comparision predicates (Any, All)
→ By using exist and not exist.

Ex: Determine the information about agents who lives in Bombay & Delhi.

Sol: Select * from agents where city in ('Bombay', 'Delhi');

Ex: Determine the details of customer who placed orders with agents in Bombay & Delhi?

Sol: Select Cid from Orders where aid in (Select aid from agents where city in ('Bombay', 'Delhi'));

Ex: Find the complete details of customers who place orders with the agents who are living in Bombay & Delhi.

Sol: Select * from Customers where cid in (Select cid from Orders where aid in (Select aid from agent where city in ('Bombay', 'Delhi')));

Ex: Find details of students from the IT and CSE Branch where student table having (RollNo, name, BID) and Dept table having (Dept, BID, BName).

Sol: Select * from student where BID in (Select BID from dept where Bname in ('IT', 'CSE'));

Ex: Find out the total strength.

Select BID, count (RollNo) where BID in (Select BID from dept where bname in ('IT', 'CSE') group by BID having count (RollNo) > 60 order by 1.

Ex: Select customer id, count(ord id) ~~where~~ ord
from orders where aid in (select aid from agents where
percent = 6) group by customer id having count(ord no)
< (select avg(ord no)) ord orders by customer id.

→ Find

Restrictions for sub queries:

The following operators cannot be used b/w main query and sub query.

a) b/w and not b/w op.
b) like and not like op.
c) is null, is not null.

The following operators are normally used b/w main query and sub query

(a) in predicates: in & not in.
(b) quantified comparison predicates: Any, all, some.
(c) Exist predicate: Exist and not exist.

→ Select the cust

Any, all, some:-

> any( )

5 ✓
6 ✓
7 ✓
8
0 ✗

1
2
3
10

( ) > all ( )

5 ✗
6 ✗
7 ✗
20 ✓

1
2
3
10

= any ⟺ in.

If subquery is prefixed with 'any' key word, then the condition will be true if it is satisfied by any of the values produced by inner queries.

If subquery is prefixed with 'all' key words, then the condition is true if it satisfied by all the values produced by inner query.

→ idea

→ Find cid values of customers with discount smaller than those of any customers who live in bombay.

Select cid from customers where disct < all (select disct from customer where city = "Bombay")

× 10	10
× 12	8
× 8	
× 8	
0 ✓	

only one row.

→ Select aid value of agents who receive more than the minimum % commision.

Without subquery

Select aid from agents where percent > 5 ; (Hard Coded query).

With subquery :-

Select aid from agents where percent > any (select percent from agents).

6 ✓	6
6 ✓	6
7 ✓	7
6 ✓	6
5	5
5	5

→ identify the customer number who purchases both products P01 and P05.

Select cid from orders where Pid in (P01, P02);

it give all cids who purchase P01 & P02 & both.

Select cid from orders where Pno = P01 and

cid in (select customer number from orders where Pno = P05);

→ Exists and not exists :-

If the sub query is prefixed with exist, then the condition is treated as true, if inner query returns non empty rows.

If the inner query is prefixed with not exists then the condition is treated as true, if inner query returns empty rows.

→ always exists and not exists should be have select

Retrieve all customer names, whose customer places an order through agent a05.

1) select cname from customers where cno in (select cno from orders ano='a05').

2) or =any in place of 'in'.

3) select c.cname from customer c where exists (select * from orders where c.cid=o.cid and o.cid='a05');

(time complex).

4) select c.cname from customer c, orders o where c.cid=o.cid and o.cid='a05';

(space complex.

select cno from cho order both p01 and p02

→ select cno from orders o1 where o1.pno='p01' and

exists (select * from orders o2 where o1.cno=o2.cno and o2.pno='p05').

exists	All and any
→ Any query formulated in all and any can also be formulated in exists.	→ Any query formulated in exists may not be formulated in all and any.
→ Exists queries always leads to co related sub queries.	→ They need not be co related queries.
→ They are less efficient.	→ more efficient.

Joins

→ It is multi

(i)

(ii)

(iii)

(iv)

Exp

Variou

1

2

3

Joins :-

→ It is an alternative to subqueries to retrieve the results from multiple tables.

(i) self join: The table is joined with itself to get appropriate results.

Ex:- To get the manager id's of all the employees.

(ii) natural join: where two tables are combined based on the available common columns.

(iii) inner join: Join operation is performed on matching columns and result is retrieved if there is a match.

(iv) outer join:- Join is performed on matching column and results are retrieved based on matching rows if there is no matching row, appropriate results are displayed with null values.

Ex:- (a) left join: (b)

Student left join library.

It displays all the rows from student and rows from the library if they have a match with the student, otherwise, library values will be null for a student who has not performed any transaction with the library.

(b) right join:

Student right join library.

Here all the rows from library are listed out and rest of the results are same as left join.

(c) Full join:

student full join library.

Here all the rows from library and student are listed out and rest of the results are same as left and right outer joins.

Various ways of specifying join condition:-

(1) ..... from customer c' join orders o on c.cid = o.c

(2) ..... from customer join orders using cid.

(3) ..... from customer natural join orders.

4) ..... from Customer, Orders where Cid = O.Cid;

-> Consider the following two tables to execute the queries.

Branch:

S.NO	Cname	Flight FNO	Station from
1	chennai	A4	Delhi
2	Bng	B4	Hyd
3	hyd	C4	chennai.

① Select B.*, F.* from Branch B, Flight F where B.Cname = F.Station from;

1	chennai	C4	chennai.
3	hyder	B4	hyder.

(Inner join).

2) Select B.*, F.* from Branch B left join Flight F where B.Cname = F.Station from;

1	chennai	C4	chennai.
2	Bang	x	x
3	hyd	B4	Hyd.

(left outer join).

3) Select B.*, F.* from Branch B right join Flight F where B.Cname = F.Station from;

x	x	A4	Delhi
3	hyd	B4	Hyd
1	chennai.	C4	chennai.

4) Branch B full Flight F.

1	chennai	C4	chennai.
3	hyd	B4	Hyd.
2	Bang	x	x
x	x	4	Delhi.

(left outer join
O right outer join).

Relational algebra :-

Different operators in RA,

(i) Native relational algebra operations:

σ, π, $\div$, rename (ρ), $\leftarrow$ (assignment),

$\bowtie$ (join).

(ii) Set theoretic operators:

$\cup$, $\cap$, $-$, $\times$ (Cross product).

(iii) Extended relational algebra operators

max, min, Avg, sum, count, count(*)

select (σ) :-

Get details of students whose age is greater than 15.

Syntax - $\sigma_{\text{<condition>}}(\text{relation})$

$\sigma_{age>15}(\text{student})$

Characteristics :-

(i) It is unary operator.

(ii) operation is applied to each tuple individually.

(iii) degree of the relation from select operation is same as input relation.

(iv) It is Commutative in nature.

$$\sigma_{<c_1>}\left(\sigma_{<c_2>}(R)\right) = \sigma_{<c_2>}\left(\sigma_{<c_1>}(R)\right).$$

Ex:-

$$\sigma_{age>15}\left(\sigma_{branch='CSE'}(\text{student})\right) = \sigma_{branch='CSE'}\left(\sigma_{age>15}(\text{student})\right)$$

$$\sigma_{<c_1>}\left(\sigma_{<c_2>}\left(\sigma_{<c_3>}(R)\right)\right) = \sigma_{<c_1> \text{ and } <c_2> \text{ and } <c_3>}(R).$$

The no of rows returned by selection operation is less than or equal to no of rows in original table.

$$|\sigma_{<c>}(R)| \leq |R|.$$

Projection :- It is used to filter the columns

Syntax: $\Pi_{<\text{attribute list}>}(R)$

Ex: $\Pi_{name, age}(R)$

The degree of o/p relation = no of attributes mentions in the attribute list.

* Projection eliminates duplicates automatically
* Commutative property cannot be applied for projection.

$$\Pi_{name}(\Pi_{name, age}(student)) \neq \Pi_{name, age}(\Pi_{name}(student))$$

→ Find the student name and age who belong to CSE and marks > 500.

SQL: select name, age from student where branch='CSE' and marks > 500;

RA: $\Pi_{name, age}(\sigma_{branch=CSE \text{ and } marks > 500}(student))$

→ SQL: select * from student.

RA: (student)

Rename operation :-

It is used to store the intermediate result of any query.

$$\rho(branch\_cse) \leftarrow \sigma_{branch='CSE' \text{ and } marks > 500}(student)$$

intermediate results are stored in a temporary table using assignment operation and it is renamed to branch_CSE

The rename operation can also be used to rename attributes

P (Studentname, studentage) ← Π name, age (Branch_CSE).

Division (÷) :

It is well suited for the queries that consists the word "for all".

This operation defines a relation and the attributes "C" that consists set of tuples from "R", that match the combination of every tuple in the attribute list "C".

Condition for ÷ operation :

The denominator must be subset of numerator.

Ex : $\frac{abc}{c} = ab$.

and result is numerator − denominator.

R

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2
a_1	b_2	c_3
a_1	b_2	c_4
a_1	b_1	c_5

$S = \frac{C}{c_1}$

$R \div S =$

A	B
a_1	b_1
a_2	b_1
a_1	b_2

$S =$

C
c_1
c_2

$R \div S =$

A	B
a_1	b_2
a_2	b_1

all combinations of A and B having both c_1 and c_2.

$R \Longleftarrow S =$

C
c_1
c_2
c_3
c_4

$R \div S$

A	B
a_1	b_2

$S =$

B	C
b_1	c_1

$R \div S =$

A
a_1
a_2

→ Set theoretic operations :-

R		
A	B	C
a_1	b_1	C_1
a_1	b_2	C_3
a_2	b_1	C_2

S		
A	B	C
a_1	b_1	C_1
a_1	b_1	C_2
a_1	b_2	C_3
a_3	b_2	C_3

R ∪ S =

a_1	b_1	C_1
a_1	b_2	C_3
a_2	b_1	C_2
a_1	b_1	C_2
a_3	b_2	C_3

R ∩ S =

a_1	b_1	C_1
a_1	b_2	C_3

R − S = $a_2\ b_1\ C_2$

S − R =

a_1	b_1	C_2
a_3	b_2	C_3

Ex: The name of two relations are R_1 and R_2, where R_1 contains n_1 tuples and R_2 contains n_2 tuples and $n_2 >> n_1 >> 0$. Give the minimum and max size in tuples of the result relation produced by each of the following relational algebra expression, and mention assumption if any.

o/p	assumption	min	max
$R_1 \cup R_2$	union compatible	N_2	$N_1 + N_2$
$R_1 \cap R_2$	union compatible	0	N_1
$R_1 - R_2$	"	0	N_1
$R_1 \times R_2$		$N_1 \times N_2$	$N_1 \times N_2$

$\sigma_{<c_1>}(R_1)$ | 0 | N_1

$\pi_{<attribute\ list>} R_2$ | attribute list contains primary key. | N_2 | N_2

$\frac{R_2}{R_1}$ | $R_2 \supseteq R_1$ | 0 | N_2

A complete set of Relational algebra operators:

$\sigma, \pi, \cup, -, \times, \cap, \div, \bowtie$

1) Intersection can be expressed in terms of union and difference.

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

2) Join can be expressed with help of σ and cartesian product.

$$R \bowtie_{<c_1>} S = \sigma_{<c_1>}(R \times S)$$

3) division can be expressed with 'π', $\times$, $-$

T_1

A	B
a_1	b_1
a_2	b_1
a_1	b_2

$S \times T_1$

a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_1

$$T_1 \leftarrow \pi_{attr\_list}(R)$$

$$T_2 \leftarrow \pi_{attr\_list}((S \times T_1) - R)$$

$$T_{result} \leftarrow T_1 - T_2$$

$$R \div S = T_{result}$$

→ The precedence of RA operators.

H: π, σ, $\times$, $\bowtie, \div$, $\cap$

L: $\cup, -$

, where

s

l max

ion)

nal

tion)

max $N_1 + N_2$

N_1

N_1

$N_1 \times N_2$

~~Transactions~~ ;

DBMS
STLD

TRANSACTIONS

A Transaction is a logical unit of work and consists ~~mult~~ multiple read and write operations on the data base.

It is associated with the following 3 problems ;

(i) It may create an inconsistent result.

(ii) It may create problem in concurrent execution.

(iii) It may create an uncertainity to decide when to make changes permanent.

→ To solve the above 3 problems, we have defined some properties for the transactions and they are called acid properties. atomicity, consistency, isolation and durability.

→ If any transaction satisfies these properties, then we can say it will be free from above problems.

Atomicity :- Either all updates of transactions occur in the data base & none of them occurs ie there is no partial transactions.

→ It is the responsibility of transaction recovery manager of dbms to ensure this property.

Consistency :- operations of the transactions must bring data base from one consistent state to another consistent state.

To ensure the consistency we will use integrity constraints and they are depending on policies of organisation.

Isolation :-

interfe

To ense

must

~~Dur~~
Durabil

by a

permon

To en

along

The pro

A ~~trig~~

A transc

start

Isolatio

Na

mar

Scott

Joh

(i) Dir

Con

At

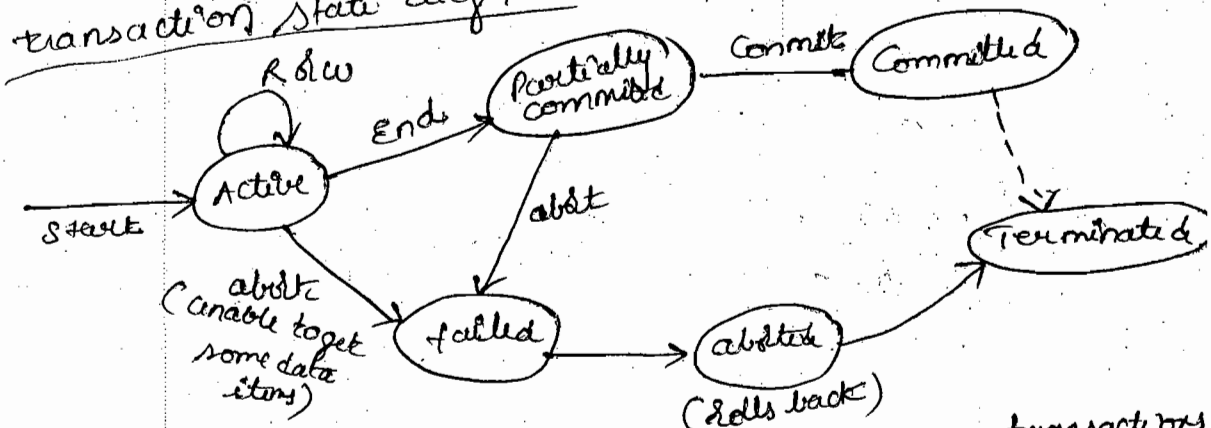
Isolation :- The execution of one transaction should not interfere with the execution of another transaction. To ensure the isolation, concurrency control protocols must be implemented.

Durability :- The changes applied to a database by a committed transaction must be made permanent. To ensure this logs in the dbms will be used along with the recovery system.

The primitives in the transaction :

A ~~tstg~~ ~~begin~~ Begin, Read, Write, End, Commit, Rollback (abort)

A transaction state diagram



Isolation problems & problems in interleaving the transactions

name	sal	dept
mary	300	sales
scott	500	mktg
John	400	production

(i) **Dirty read & reading uncommitted data & WR conflict :**

At 10 am — St tx1
10:10 am — set mary's Sal to 500
10:20 am — ~~Commit~~ Rollback tx1.

At 10:15 Am st TX_2
10:15 Am Read all sal (there uncommit 500 will be Read).

→ Not Repeatable Reads (values):

At 10 am st TX_1
At 10:05 am Read all salaries (300, 500, 400)
At 10:10 am Read all salaries (500, 500, 400)

At 10:05 am st TX_2
At 10:06 am set mays salary to 500
10:07 Commit.

→ Not Repeatable Reads (~~Read~~ Phantom Row) & RW conflict

At 10:00 Am st TX_1
10:05 Am Read all salaries (300, 500, 400).
10:10 Am Read all Salaries (300, 500, 400, 1000).

At 10:05 Am st TX_2
10:06 Am Insert ('Sam', 1000, 'personal')
10:07 Am C2.

→ Incorrect summary problem:

At 10:00 Am st TX_1
10:05 Am Sum (Sal) 1200
10:10 Am Sum (Sal) 1400

At 10:05 Am st tx_1
10:06 Am Set may salary to 500
10:07 Commit.

→ Lost update problem (WW conflicts)

10:00 AM St Tx1
10:10 AM Set mary sal to 500
10:20 AM commit

10:10 AM St Tx2
10:15 AM Set mary sal to 600
10:21 AM C2.

→ Unstable cursors:

10:00 AM St Tx1
10:05 AM Sets mary sal to ~~6000~~ 500
10:15 AM Read all sal

At 10:10 AM St Tx2
10:11 AM drop Sal Colum
10:12 AM C2.

There are two ways to drop a column:

(i) Physical dropping:

Alter table tablename drop column Column_name

Ex: Alter table student drop column sid;

(ii) Logical drop:

Alter table tablename set unused (column_name);

afterwards, if you want to have its again set unused.

Schedule :-

The order of the execution of the transactions is known as schedule.

Two types of schedules :-

(i) serial (ii) non serial.

The no serial schedules = $n!$ where n = no of transactions.

Assume there are m transactions and no of operations in these transactions are $n_1, n_2, n_3 \cdots n_m$.

the no non serial schedules = $\frac{(n_1 + n_2 + \cdots n_m)!}{n_1! * n_2! \cdots n_m!}$

Ex :- Assume there are two transactions and ~~Transcate~~ T_1 contains 2 and T_2 contains 5 operations Find the no. of serial and non serial schedules.

Sol) serial schedules = 2.

non serial schedules = $\frac{7!}{2! \times 5!} = \frac{6 \times 7}{2} = 21$.

Serializability :-

A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same 'n' transactions.

There are ~~two~~ three types of serializabilities.

(i) Result serializable.

(ii) ~~view~~ conflict serializable.

(iii) view serializable.

$X=100$	$X=100$
$R(X)$	$R(X)$
$X = X+10$	$X = X * 1.1$
$W(X)$	$W(X)$
$=110$	$=110$

If $X=200$ then they are not equal.

The end

input

Conflict

Two

all

(i)

(ii)

(iii)

Ex :

→ consider

The end results of schedules are heavily depend on input of schedules

∴ result equivalences are not generally used.

Conflict ~~Equivalence~~ serializability:

It is based on the conflict operations.

Two operations are conflict, if they satisfy all three of the following conditions.

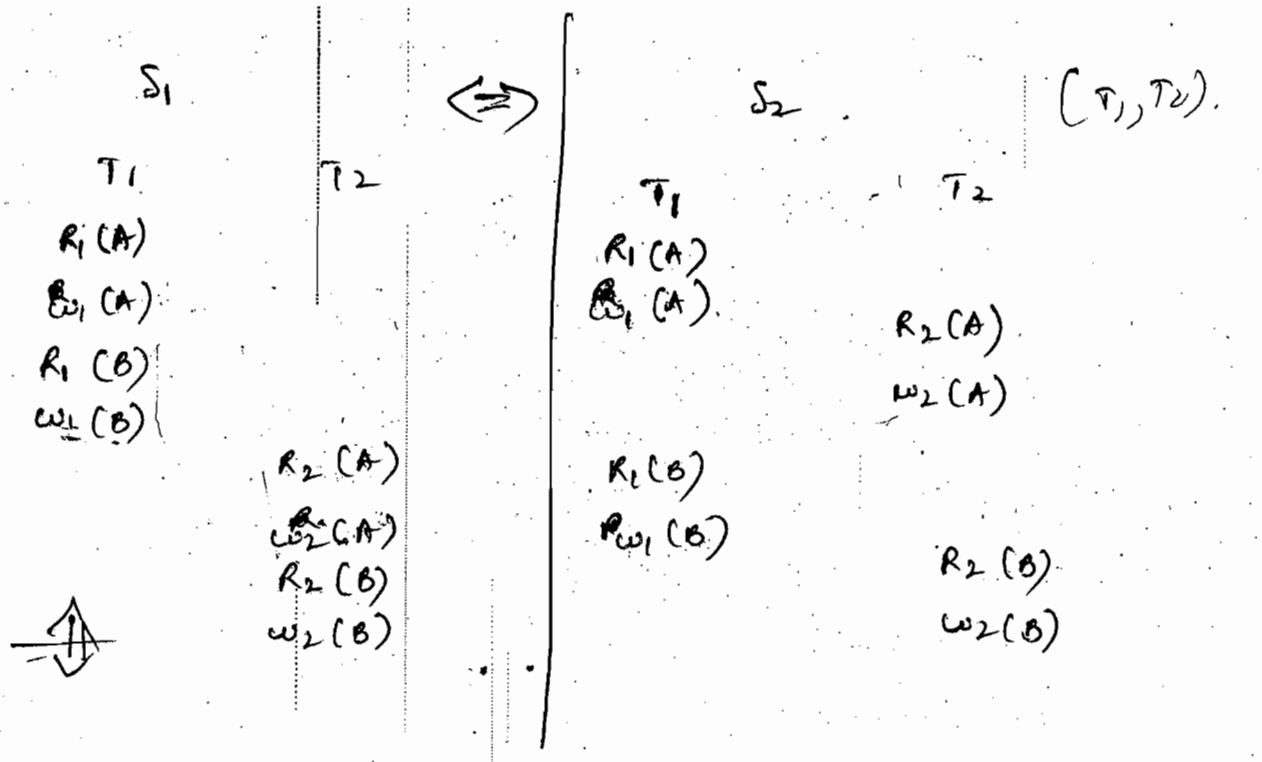
(i) they belong to different transactions
(ii) they access the same data item
(iii) At least one of the operations is a write operation.

Ex: If we have $X_i(A)$ and $Y_j(B)$ mention under what conditions these two operations are confliction.

(1) $A = B$,
(2) $i \neq j$,
(3) x & y must be write operation.

→ Consider T_i and T_j are two transactions. I_i and I_j are two consecutive instructions of T_i and T_j. Then operations are said to be conflicts if they refer same data item and atleast one of them must be write operation.

→ Identify non conflicting operations and we change the order of non conflicting operations to get a non serial schedule, that is equal to serial schedule.



S_1 and S_2 are conflict

S_3

$R_2(A)$
$W_2(A)$

$R_1(A)$
$W_1(A)$

$R_2(B)$
$W_2(B)$

(T_2, T_1)

$R_1(B)$
$R_2(B)$

view equivalent :-

Consider the two schedules S_1 and S_2, they are said to be view equivalent if the following conditions are satisfied.

(i) For each data item 'Q', if T_1 ~~reads~~ in S_1 reads the initial value of 'Q', then T_1 in S_2 must also read the initial value.

(ii) For each data item 'Q', if T_1 in S_1 reads a data item produced by a transaction T_0, then T_1 in S_2 must also read the data item

produced by T_j.

(iii) For each data item 'Q', if T_i in S_1 performs a final write operation, then T_i in S_2 must also perform a final write operation on data item

S_1

T_1	T_2
R(A)	
A=A-50	
W(A)	
R(B)	
B=B+50	
W(B)	
	R(A)
	t=A*0.1
	A=A-t
	W(A)
	R(B)
	B=B+t
	W(B)

S_2

T_1	T_2
R(A)	
A=A-50	
W(A)	
	R(A)
	t=A*0.1
	A=A-t
	W(A)
R(B)	
B=B+50	
W(B)	

S_3

T_1	T_2
R(A)	
A=A-50	
W(A)	
	R(A)
	t=A*0.1
	A=A-t
	W(A)
R(B)	
B=B+50	
W(B)	
	R(B)
	B=B+t
	W(B)

S_1 and S_3 are view equivalent.

→ Differences b/w conflict and view:

(i) conflict serializability is easy to achieve but view serializability is difficult to achieve.

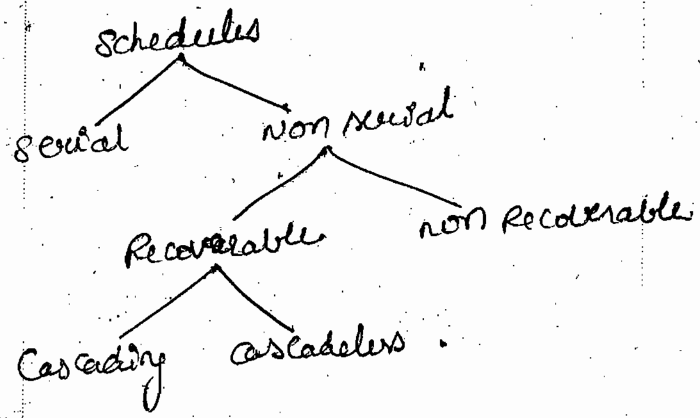
(ii) Every conflict serializable is a view serializable but reverse is not true

(iii) It is easy to test conflict serializability but expensive to test view serializability.

(ii) most of the concurrency control schemes used in practice are based on conflict serializability.

~~schedules can also~~

→ The ~~recoverable~~ schedules can characterize



S1: $R_1(X), R_2(X), W_1(X), R_1(Y), W_2(X), C_2, W_1(Y), C_1$.

T_1	T_2
$R_1(X)$	
	$R_2(X)$
$W_1(X)$	
$R_1(Y)$	
	$W_2(X)$
	C_2
$W_1(Y)$	
C_1	

S2: $R_1(X), W_1(X), R_2(X), R_1(Y), W_2(X), C_2, a_1$;

T_1	T_2
$R_1(X)$	
$W_1(X)$	
	$R_2(X)$
$R_1(Y)$	
	$W_2(X)$
	C_2
a_1	

used

equivalizability

→ If there is no necessity to roll back a committed transaction, then it is called a Recoverable schedule otherwise it is a non Recoverable schedule.

S_1 is recoverable and S_2 is non recoverable.

→ Cascading schedules

To make a non recoverable schedule to Recoverable like S_2, the following modification is needed i.e. Commit T_2 only after T_1 is Committed or abbted.

Here if T_2 is reading uncommitted data from T_1 but T_2 should commit only after T_1 is committed if you roll back T_1, T_2 must also be rolled back and it is called cascading roll back

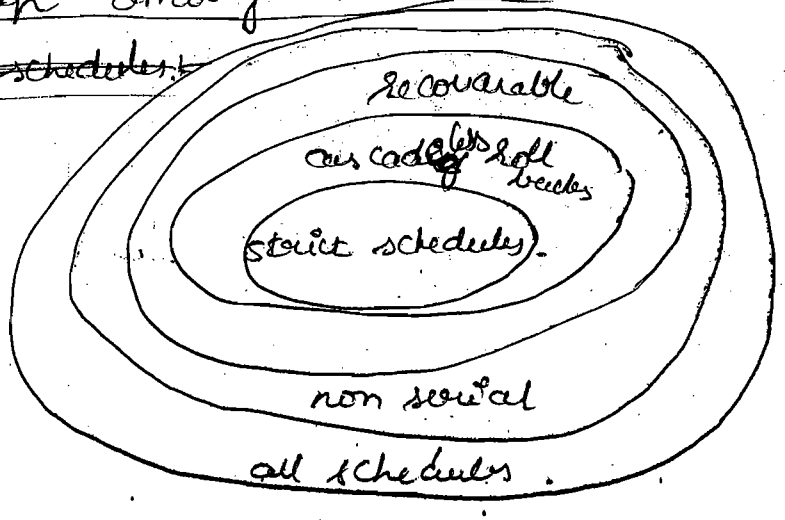
To avoid the cascading roll backs, modification is provided to S_2 like.

T_1	T_2
$R_1(X)$	
$W_1(X)$	
$R_1(X)$	
a_1	
	$R_2(X)$
	$W_2(X)$
	C_2

but it leads to a serial schedule.

→ The Relationship among the schedules

(i) ~~Strict schedules~~



→ Recovarable schedule :- (Pg 418)

T_2 commits after T_1, if T_2 has read any data item written by T_1. Ex: $r_1(x); r_2(x); w_1(x); r_1(y); w_2(x); w_1(y)$

Cascadeless schedule :- (Text 419)

T_2 reads a data item only after T_1 has written to it and terminated.

Strict schedule :-

T_2 reads a data item after T_1 has written to it and terminated, in addition T_2 writes a data item after T_1 has written to it and terminated.

Page 98:

Problems

1) which of following is conflict serializable schedule, determine the equivalent serial schedules

(i) $r_1(x), r_3(x), w_1(x), r_2(x), w_3(x)$.

$T_2 < T_3$

$T_3 < T_2$ $\quad T_1 < T_2$.

$3 \to 1 \to 2$.

C not ✓

(ii) $r_1(x), r_3(x), w_3(x), w_1(x), r_2(x)$. ✓

$3 < 1$

$3 < 1$ $\quad 1 < 2$

T_3 T_1 T_2

$3 \to 1 \to 2$.

(iii) $r_1(x), r_3(x), w_2(x), w_1(x), r_2(x)$.

$T_1 < T_2$ ✗

$T_3 < T_2$ $\quad T_2 < T_1$ ✗

$3 \to 2 \to 1$

(iv) $r_3(x), r_2(x), w_3(x), r_1(x), w_1(x)$

$2 < 3$ $\quad 3 < 1$

T_2, T_3, T_1.

→ $r_3(x)$

2) a) de

b) R

c)

3) (a)

(b)

→ $r_3(x)$, $r_2(x)$, $r_1(x)$, $w_3(x)$, $w_1(x)$

$T_1 < T_3$

2) a) dirty read

$R_1(x)$, $R_2(y)$, $w_2(x)$, $R_2(x)$ - - - -

b) Repeatable reads.

$R_1(x)$, $R_2(y)$, $w_2(x)$, $R_2(x)$ $R_1(x)$

c) $R_1(x)$ $R_2(x)$ $R_2(y)$ $w_2(x)$ $w_2(x)$ - —

3) (a). $T_1: R(x)$, $T_2: R(x)$, $T_1: w(x)$, $T_2: w(x)$.

$T_2 < T_1$ $T_1 < T_2$

not conflict serializable.

~~(b) $T_1: w(x)$, $T_2: R(y)$, $T_1: R(y)$, $T_2(R(x)$.~~

T_1	T_2
$R_1(x)$	
	$R_2(x)$
$w_1(x)$	
	$w_2(x)$

Cascadeless schedule, recovarable, view serializable (T_1, T_2)

(b) $T_1: w(x)$; $T_2: R(y)$, T_1: $R(y)$, $T_2: R(x)$

$T_1 < T_2$

ok is conflict serializable so ad view serializable.

To decide above have recoverability, commit statement should be given.

T_1	T_2
$w_1(X)$	
	$R_2(X)$
$R_1(Y)$	
	$R_2(X)$

It is cascading schedule, if T_1 is rolled back, then T_2 must also be rolled back. If T_1 commits first than T_2, it is recoverable if T_2 commits first then T_1 it is not recoverable.

c) $T_1 : R(X)$, $T_2 : R(y)$ $T_3 : w(x)$, $T_2 : R(X)$

It is conflicts and view serializable.

It is cascading

T_1	T_2	T_3
$R_1(X)$		
	$R_2(Y)$	
		$w_1(x)$.
	$R_2(X)$.	

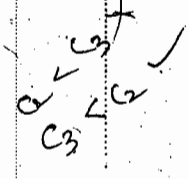
If T_3 commits before T_2, it is recoverable else it is not.

d) $T_1 : R(X)$ not conflicts, [illegible], not cascadeless.

$T_1 : R(X)$, $T_1 : R(Y)$; T_1 $w(x)$ T_2 $R(Y)$, T_3 : $w(x)$

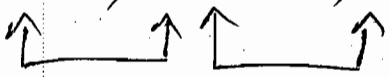
$T_1(w(x)$, $T_2 R(x)$.

T_1	T_2	T_3
$R_1(X)$		
$R_1(Y)$		
$w_1(X)$	$R_2(Y)$	$w_3(Y)$
$w_1(X)$	$R_2(Y)$	



e) It is conflict ~~here~~ view serializable, Cascadeless and recoverable.

$T_1: R(x)$, $T_2: w(x)$, $T_1: w(x)$, T_2 : abort T_1 : Commit



f) not conflict serializable, not view, recoverable and Cascadeless

g) Conflict, view, recoverable, Cascadeless.

h) not Conflict, not view, not recoverable, not Cascadeless

i) $T_1: w(x)$, $T_2: R(x)$, T_1 : $w(x)$, T_2 : C, T_1 : Abort, T_1 Abort.

conflict, view, not cascadeless, not, recoverable.

j) conflict and view, ~~not~~ recoverable, Cascadeless.

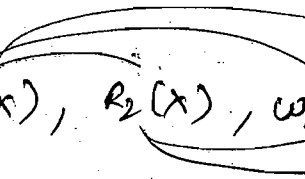
k) ~~Same as j~~ Not conflict, not v, Cascading, recoverable.

l) not conflict, not view, recoverable, not Cascadeless.

Testing for Conflict serializability :-

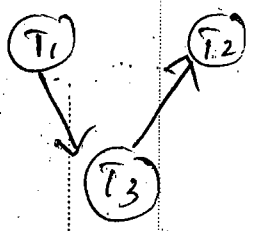
To test the conflict serializability we can draw a graph and it consists set of nodes and set of edges. no of nodes = no of transactions. Each edge is of the form $T_a \rightarrow T_b$ where T_a is starting node and T_b is ending node such an edge must be created if if one of the operations in T_a appears in schedule before some conflicting operation in T_b.
once the graph is drawn, check for the loops. of there is a cycle, the schedule is not in conflict serializability.

4) a) $R_1(x), R_2(x), w_1(x), w_2(x), c_1, c_2$.



not ~~serial~~ Conflict serializable

b) $R_1(x), R_2(y), w_3(x), R_2(x), r_1(y), c_1, c_2$.



Conflict serializable.

c) $R_1(x), w_2(x), w_1(x), a_2, c_1$

node T_2 is removed because of 'a_2'.

(T1)

d) $w_1(x)$

e) R_1

f)

g)

h)

(i)

Implementa

1) Access mo
read and

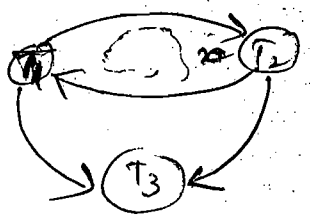
2) Diagona
that can
executed

3) Isolation
transacti
executing
Concurren
A

e) $w_1(x), R_2(x), w_1(x), C_2, a_1$

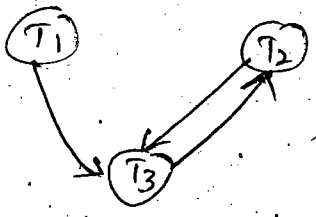
T_1

f) $R_1(x), w_2(x), w_1(x), R_3(x), C_1, C_2, C_3$.



not conflict serializable.

f) $R_1(x), R_1(y), W_1(x), R_2(y), W_3(y), W_1(x), R_2(y)$



g) Draw procedure graphs for $W_3(A), R_1(A), W_1(Z), R_2(B), W_2(y), W_3(B), C_1, C_2, C_3$

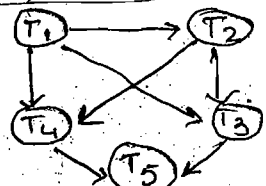
g) $T_2 \rightarrow T_3 \rightarrow T_1$ ✓

h) Draw procedure graph for $R_1(A), W_1(A), R_2(A), R_2(B), C_2, R_1(B), W_1(B), C_1$

h)

X

i) Consider the procedure graph is the corresponding schedule Conflict Serializable



(i) no cycles,

Implementation of transaction SQL:-

1) Access mode:- There are 2 types of access mode read only and read and write.

2) Diagonastic Size:- It determines the no of error conditions that can be recorded on the user on the most recently executed SQL statement.

3) Isolation level:- It controls the extent to which a given transaction is exposed to the actions of other transactions executing concurrently.

Concurrency is	Dirty read	unrepeatable	phantom
Level1	✓	✓	✓
Level2	✓	✓	✗
Level3	✗	✗	✗

test with ?

Part ★ classification of concurrency protocol

(i) lock based protocol
  (a) 2 phase locking protocol → Basic 2PL
      → conservative 2PL.
      → Strict 2PL
      → rigerous 2PL.
  (b) Graph based protocol.

(ii) Time stamp based protocol
  (a) Timestamp ordering protocol
  (b) Thomas write rule.

(iii) multiple granularity protocol.

(iv) multi version protocols.

T_i
R(A)

① $TS(T_i) < WTS(A)$ ✗
900 < 910.

② $TS(T_i) \geq WTS(A)$ ✓

T_i
W(A)

① $TS(T_i) < RTS(A)$ ✗
900 910

② $TS(T_i) \leq WTS(A)$ ✗

③ $TS(T_i) > WTS(A)$ ✓

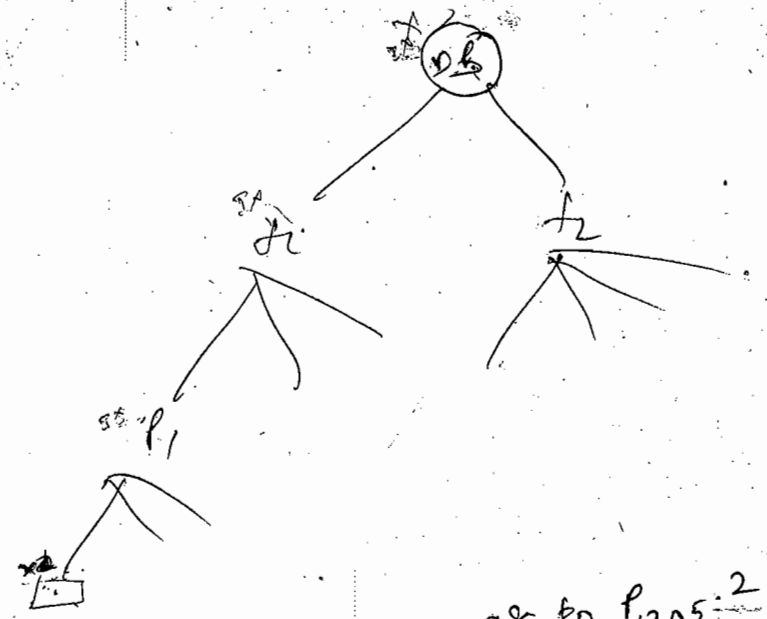
→ Consider a data base, that is organised in terms of the following hierarchy.
• The database itself is an object and it contains 2 files, each file contains 1000 pages each page contains 100 records and these records are identified as p.i where p is a page no and i is a record number.
For each of the following operations indicate the sequence of lock request that must be generated by a transaction manager.

① Read

② Read

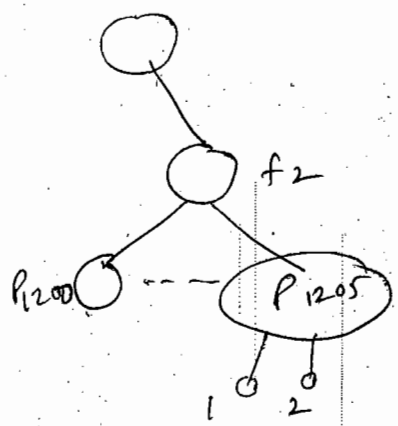
③ Read

(1) Read record $P_{1200}.5$



IS on DB
IS on f2
W on P_{1200}
S on $P_{1200}.5$

(2) Read records $P_{1200}.98$ to $P_{1205}:2$



IS on P_{1205}
S on P_{1205} : 1/2
IS on P_{1200}
S on P_{1200} : 99/98/100
S on P_{1201}, P_{1202}, P_{1203}, P_{1204}
IS on f2
IS on DB.

(3) Read P_{500} to P_{600}.

IS on DB
S on F1

IS on DB
IS on F1
S on $P_{500}, 501 \ldots 600$

→ Read all pages in F_1 and modify about 10 pages which can be identified only after reading F_1.

S:IX on DB
S on f_1

→ Delete the I record from each ~~file~~ Page.

IX on DB
X on f_1 and f_2

→ Delete all records

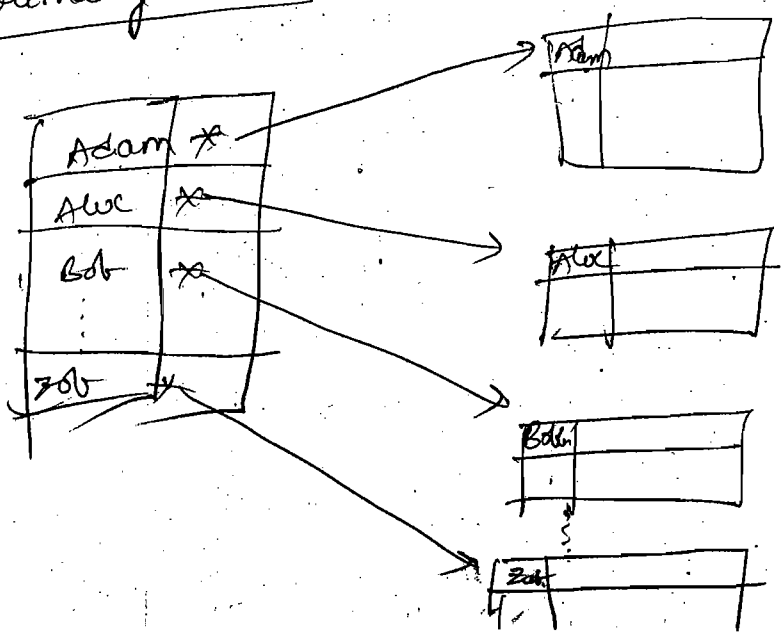
IX on D
X on F_1 and F_2.

Indexes

→ Types of indexes:
(i) Primary indexes.
(ii) Secondary indexes.
(iii) clustering indexes.

other classification:
(i) Sparse index.
(ii) Dense index.

Primary index:



out
ter

Characteristics

(i) It is an ordered file

(ii) It consists 2 entries.

(a) key (b) a ~~block~~ block pointer

(iii) The no of index records = no of files & pages & blocks.

(iv) It is an example for sparse index.

→ Consider a block size = 1024 B, then record length = 100

∴ no of records/block = $\frac{1024}{100}$ = 10 records

Assume there are 30,000 records in data base

∴ no of blocks = $\frac{30000}{10}$ = 3000 blocks.

If the search key length = 9 bytes and block pointer contains 6 bytes then each index record length = 15 bytes

size of total no of index file records = 3000 × 15 Bytes.

∴ no of index blocks = $\frac{3000 \times 15}{1024}$

= 45 Blocks.

Search operation

(a) without Index

$\log_2 b = \log_2 3000 = 12$

(b) with index

$\log_2 45 = 6$,

$6 + 1 = 7$.

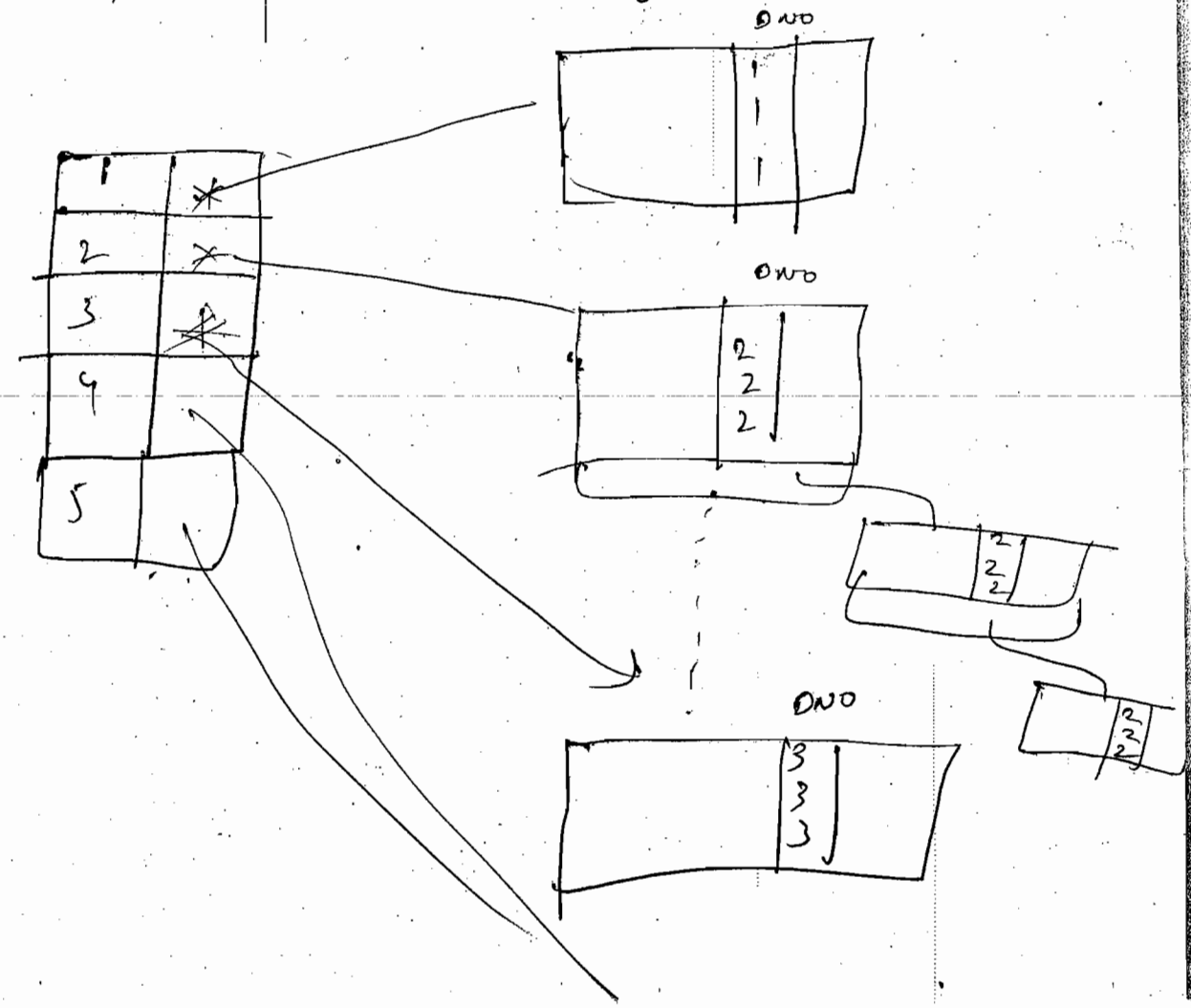
→ Problems with primary index :-

The problem of with primary key index is insertions and deletions. If we insert a record in its correct position, then there is a possibility of moving anchor records. Then it leads to changes in index file structure.

Solution :- To avoid this, we will use a fill factor normally from 0.5 to 1

Clustering index :-

Primary index must be also created on the field that must have a unique value for each record, then if you want to create an index on a field that does not have a unique value, then we use clustering index.

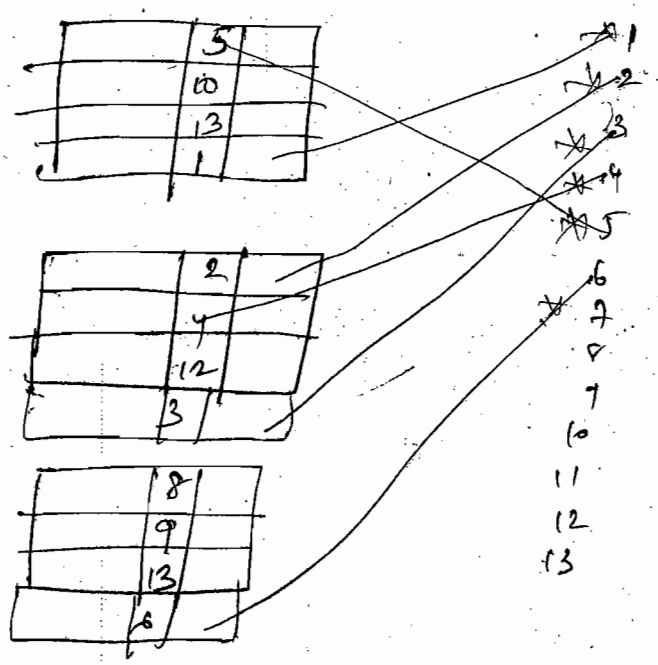


Characteristics :-

It is sparse index, we can have only one clustered index per table and it will have two entries; one is search key and other is block pointer.

Secondary indexes :-

A secondary index is on ordered file and it is created on the fields and they need not have unique values and multiple secondary indexes are possible as they need not require physical reordering of the records.



It is a dense index.

→ Block size = 1024, record length = 100 bytes.

no records / block = $\frac{1024}{100}$ = 10 records

Index record length = 9+6 = 15

no of ~~block~~ index records / block = $\frac{1024}{15}$ = 68.

no of secondary index records = 30,000

no of blocks required = $\frac{30000}{68}$ = 442 blocks

search.

(a) without secondary index :

$$\frac{30,000}{2} = 15000$$

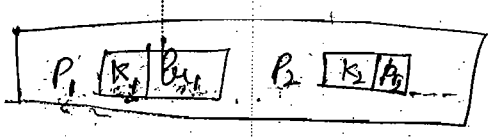
(b) with secondary index.

$$\log_2 4,42 = 9 + \log_2 68$$

→ multilevel index :

These are implemented with B and B^+ trees and these trees grow horizontally instead of vertically.

The structure of B tree node will look like



P_i is a tree pointer, pointing to another node.

Pr_i is a data pointer whose key value is k_i.

⇒ Consider a p is order of the tree, then no of node pointers = p, then the no of data pointers and search key values = p-1.

→ Consider a node size & block size = 512 B, a search key length (K_i) = 9 bytes, data pointer = 7 bytes, block pointer (P_i) = 6 bytes, then find the order of node

$$p6 + (p-1)(16) = 512.$$

$$22p = 512+16$$

$$p = \frac{528}{22} = 24.$$

Let p be the order of the tree, then

$$p * P_i + (p-1) K_i + (p-1) Br_i \leq B.$$

→ Consider the full factor for the node = 69%.

∴ Order of the node = $24 * 0.69 = 16$.

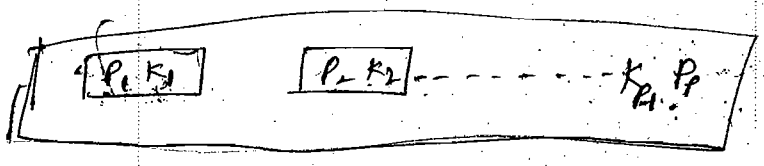
root	1 node	15 data entries	16 ptrs.
Level1	16 nodes	$(16 \times 15) = 240$	$16 \times 16 = 256$.
Level2	256	3840 (256×15)	$256 \times 16 = 4096$
Level3	4096	61440 (4096×15)	4096×16

→ B+ tree :-

In B tree every node consists search key value, data pointer along with the block pointer, so redundancy is high. But in B+ tree, data pointers are stored only at the leaf nodes of the tree. ∴ B+ tree will have two types of the nodes.

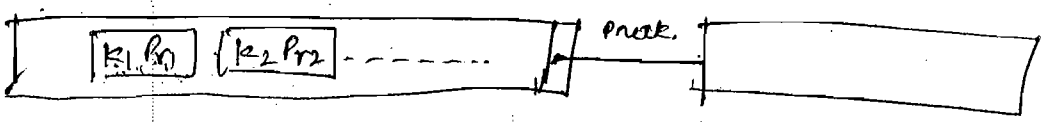
(i) internal node (ii) leaf node.

Structure of internal node :-



P_i - block ptr K_i : search key value, p - Order of the tree

Struct of leaf node :-



internal :-

$$(p_{ptr} * P) + (p_{internal} - 1) K_i \leq B.$$

leaf :-

$$(p_{leaf} * P_r) + (p_{leaf} * K_r) + P_{next} \leq B.$$

Prob 1

$B = 512$, $P_{ptr} = 7$, $P_d = 6$, $R_d = 29$.

$P_{intn} = 34$.

$P_{leaf} = 31$.

→ Consider fill factor as 69 % ∴ $p_{int} = 34 \times 0.69 = 23$.

Root	1 node	22 data entries	23 node ptr.
Level 1	23 node	506 (23 × 22)	23 × 23 = 529.
Level 2	529	11,638 (529 × 22).	529 × 23 = 12,167
Level 3	12,167	2,55,507 (12,167 × 22)	- - -

So no of level in B^+ are less than B for the same data size.

⇒

Strict 2PL: A transact^n T doesn't release any of its exclusive (write) locks until after it commits or aborts. Hence no other transact^n can read or write an that is written by T until T commit
Dead lock.

Basic 2PL: All locks once then all unlocks i.e expanding a growing phase & shrinking phase.

Conservative 2PL: Lock all items it accesses befor the transact^n begins execut^n. If any item already locked then don't lock rest of items also & wait. Deadlock free.

Rigorous 2PL: In this, a transact^n T doesn't release any of its locks (exclusive or shared) until after it commits or aborts.

Diff b/w conservative & Rigorous:

In conservative must lock all its items before its starts so once the transact^n starts it is in its shrinking phase where as the latter does not unlock any of its items until after it terminates (c or a) so this transact^n is still in expanding phase.

Finite Automata and Regular expressions

Conversion of FA to RE there are 3 methods

(i) mcnaughton yamada theorem (6 hours)
(Rij^k method)

(ii) Arden methods (2 & 3 mins)

(iii) state elimination method (1 mins)

Arden method:

(i) this method is not applicable for ε-NFA.

(ii) If P, Q are two RE's and P does does not contain ε, then the unique solution

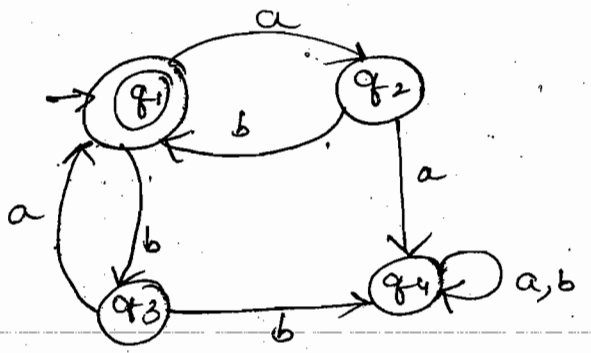
$R = Q + RP$ can be written as

$R = QP^*$.

(iii) If P contains ε, then we will get infinite solutions

Questions:

Give the RE for following FA, by using ardens method

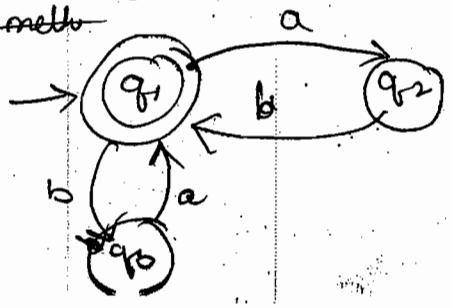


⇓
simplification

1) Remove non reachable states.

2) q_4 does not reach final state so remove it
(Remove those states which does not leads to final state.

Arden method



1) find

→ find

q

q

q

1) Find characteristic equation of every state

$q_1 = \epsilon + q_2 b + q_3 a \rightarrow (1)$

$q_2 = q_1 a \rightarrow (2)$

$q_3 = q_1 b \rightarrow (3)$

$q_1 = \epsilon + q_1 ab + q_1 ba$

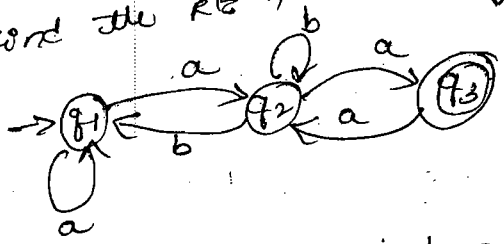
$= \epsilon + q_1 (ab + ba)$

$= \epsilon + q_1 ab$.

$q_1 = \epsilon (ab+ba)^*$.

$\Rightarrow q_1 = (ab+ba)^*$.

→ Find the RE of following FA by using arden method.



$q_1 = \epsilon + q_1 a + q_2 b$.

$q_2 = q_2 b + q_1 a + q_3 a$.

$q_3 = q_2 a$.

$q_2 = q_2 b + q_1 a + q_2 aa$.

$q_2 = q_1 a + q_2 (b+aa) \Rightarrow q_2 = (q_1 a)(b+aa)^*$

$q_2 = a + q_1 aa + q_2$

$q_1 = \epsilon + q_1 a + q_1 a (b+aa)^* b$

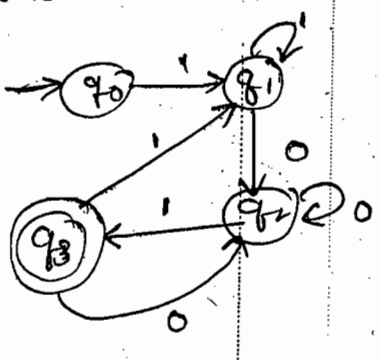
$q_1 = \epsilon + q_1 (a + a(b+aa)^* b)$.

$= \epsilon + q_1 \epsilon (a(b+aa)^*)$

$q_1 = \epsilon . (a + a(b+aa)^* b)^*$.

$q_2 = \epsilon (a+a(b+aa)^* b)^* a + \epsilon$

→ Find RE for following FA using arden method.



$q_0 = \epsilon$

$q_1 = q_0 1 + q_1 1 + q_3 1$

$q_2 = q_1 0 + q_2 0 + q_3 0$

$q_3 = q_2 01$.

$q_1 = 1 + q_1 1 + q_3 1$

$q_2 = q_1 0 + q_2 0 + q_3 0$

$q_3 = q_2 1$

$q_2 = q_1 0 + q_2 0 + q_2 10$.

$= q_1 0 (0 + 10)^*$

$q_2 = q_1 0 (0+10)^* 1$

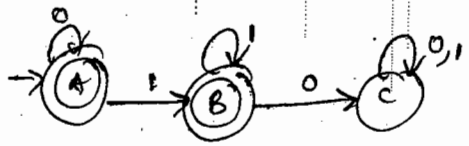
$q_1 = 1 + q_1 1 + q_1 0 (0+10)^* 11$.

$q_1 = 1 (1 + 0 (0+10)^* 11)^*$.

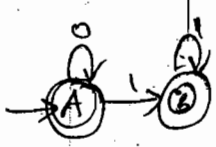
$q_2 = 1 (1 + 0 (0+10)^* 11)^* + q_2 0 + q_2 10$

$q_3 = 1 (1 + 0 (0+10)^* 11)^* (0+10)^* 1$.

→



And 'C' can be Removed.



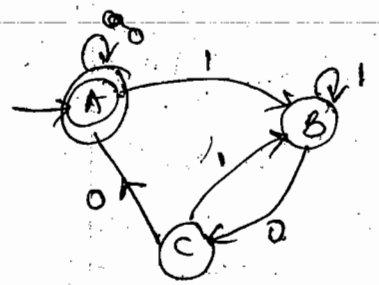
$A = \epsilon + A0$. $\Rightarrow A = 0^*$.

$B = A1 + B1$ $\Rightarrow B = 0^* 1 1^*$

$A + B = 0^* + 0^* 1 1^*$

$= 0^* (\epsilon + 11^*) = 0^* 1^*$.

→



$A = \epsilon + A0 + C0$

$B = B1 + A1 + C1$

$C = B0$

$B = B1 + A1 + B01$

$B = A1 (1+01)^*$.

$A = \epsilon + A0 + A1 (1+01)^* 00$.

$A = \epsilon (0 + 1 (1+01)^* 00)^*$.