DBMS (Naseer Sir)

→ entity : something existing & distinguishable

→ mini-world / scenario : The whole components together in a system.

eg: company, institution

→ degree/arity : no. of entities types involved in a relationship type

ER MODEL.

E.F. Codd

Turing award for relational model

phases of DB design :

mini-world.

↓

req. collectn & analysis

↓

conceptual design (ER)

↓

logical design (relational)

↓

physical design.

mini-world → functnal analysis

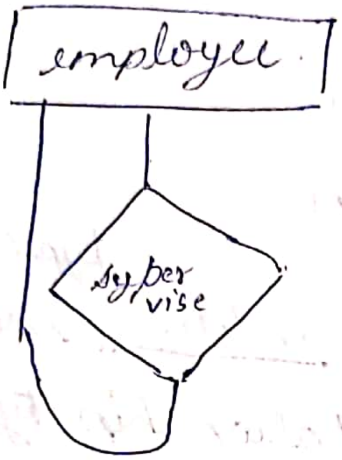
↓

app. prgrm design.

Every entity has their own attributes

→ There can be unary relationship

eg: an employee



Supervisor & worker are both employee

The whole of employee forms entity Type and one employee is called an entity

⇒ Dependent is also a type of entity in case of a company scenario. ie, Dependent dependent on employee

$\underbrace{\text{dependent}}_{\text{relation}}$ on employee — entity

⇒ Strong entity : entity type whose individuals can be ~~uniqly~~ uniquely identified

eg: employee (using empID), projects (proj.no)...

⇒ entity Type can be correlated to a table with column heading as the entity's attribute

⊙ An entity which can exist only in relation with another entity is called a <u>weak entity</u>.

The one which weak entity is dependent on is called <u>owner entity</u> / <u>identifying entity</u>

eg: i.e., In a company mini-world, the dependent exist in association with an employee

i.e., we need empID as well as dependant-name to identify a row in dependant entity type

<u>Attributes</u>

↳ simple : It has a single value (sex, name

↳ composite : attribute composed of many components.

eg: Name (1st, middle, last name)

↳ Multivalued: multiple values for that attribute eg: color of car({ }

→ complex - multi valued. eg :

{ prev. degree (college , year) }

composite

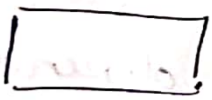
→ attribute that uniquely identify a row - key

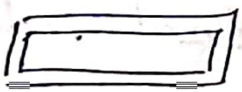
→ key can be composite

→ key can be more than 1 attribute

(1 is primary , others are candidate)

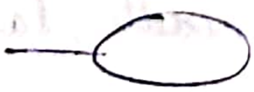
Symbols in ER

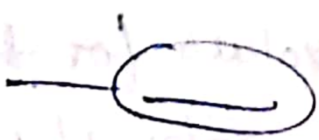
 — entity type

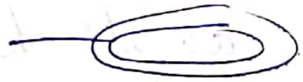
 — weak entity

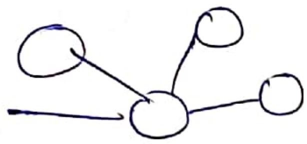
 — relation

 — identifying relationship type (relation b/w weak & owner)

 — attribute

 — key attribute

 — multi values.

 — composite attribute

 — derived attribute (month of birth from DOB)

alias name : alternate name given to an entity depending on its relation to another entity

eg. employee, supervisor

⇒ Relationship of same type are grouped into a relationship type.

⇒ Entities involved in a relationship type are called participating entities.

⇒ <u>Weak entity</u>

→ No key attribute

→ partial key of weak entity and entity related to that weak entity is required to act as its key attribute

Constraints on Relationships

① Cardinality ratio

Max. no. of ~~entity that~~/relationship instances that an entity can participate in

eg: a) one → one. eg: employee manages dept.

b) 1 → many or (1:N) many → 1 eg: emp. works on dept

c) many → many (m:n) eg: emp. works on proj

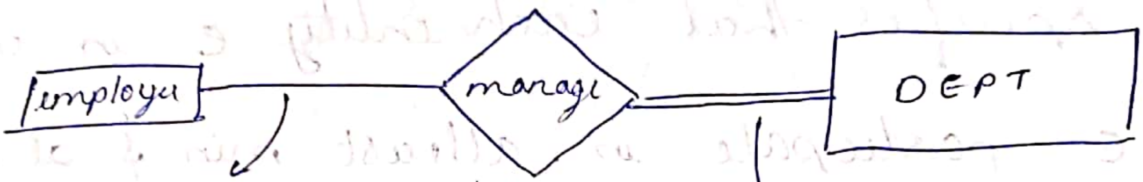
② Participation

Specifies whether the existence of an entity depends on its being related to another entity via the relation type

→ Also called minimum cardinality constraint

→ Minimum no. of relationship instances an entity can participate in a relationship type

eg.



partial participation
i.e. all employees need not manage a dept

Total participation
all entities of dept must have a manager.

Recursive relationship type

same entity assuming diff roles in the same relationship type

eg. supervises

⇒ Relationship type can also have attribute

eg: hours can be attr. for "works for" relation

⇒ Derived attributes

eg: no. of employee's can be one for dept. entity

(Min, max) notation for relationship constraint (alternate for =, & ratio's)

(captures both cardinality & participation)

Specifies that each entity e in entity-type E participate in atleast min & at most max relationship instances in R

→ Default: no constraint min=0, max=1

→ min ≤ max, min ≥ 0, max ≥ 1

eg: each department must be managed by ~~at~~ exactly 1 employee.

∴ participation of department in relation is (1,1)
each dept must have atleast 1 & atmost 1 manager

& participation of employee in relation is (0,1)
each employee participate in atleast 0 dept & max 1 dept

Draw ER diagram for a virtual store that allow consumer to order products. The store obtains products from producer

→ Customers ~~by~~ are identified by Aadhar.

→ They have email & phys. add.

→ several customers may live in same phy. add but no two 2 have same emai

→ product have type, model ~~name~~ number

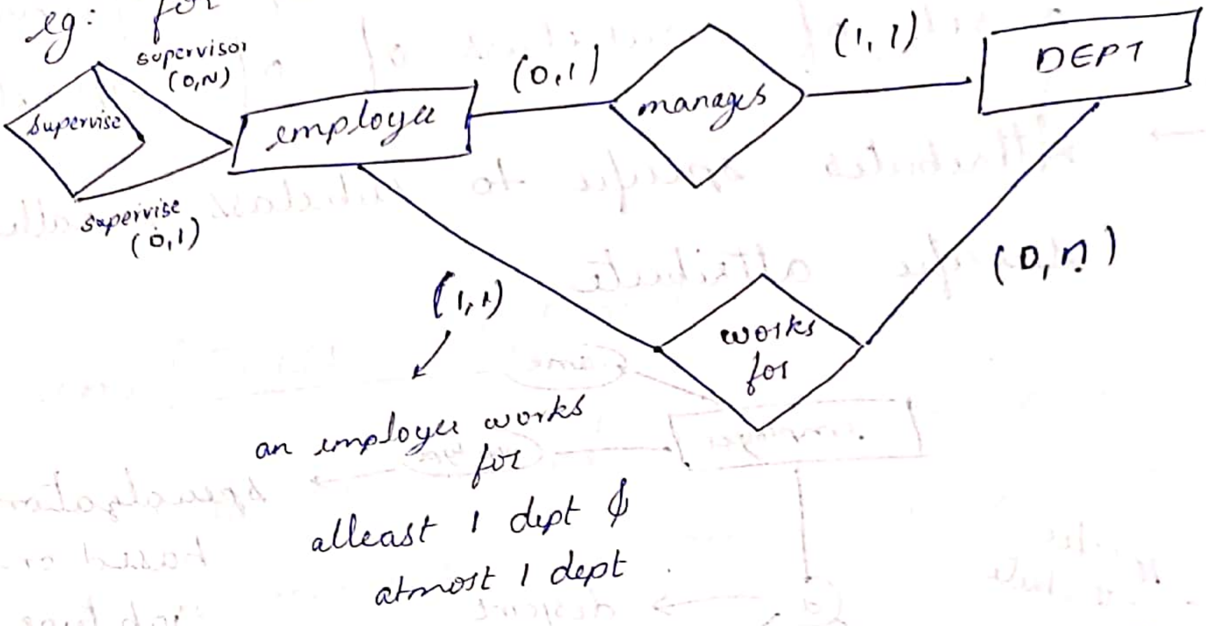
→ each product by a producer. But ~~each~~ diff producer may have diff. product with ~~diff~~ same model no. But 2 prod of same producer cannot have same model no.

→ manufacturer have name, add, & no.

→ Order has order no. and a date.

→ Order is placed by 1 customer.

→ Each order has 1 or more products ordered & each have a quantity eg: for min-max notation



EER (enhanced ER model)

It captures : sub class/super class
: specialization / generalisation

eg: secretary, manager etc are subclass of employee entity type.
or subset

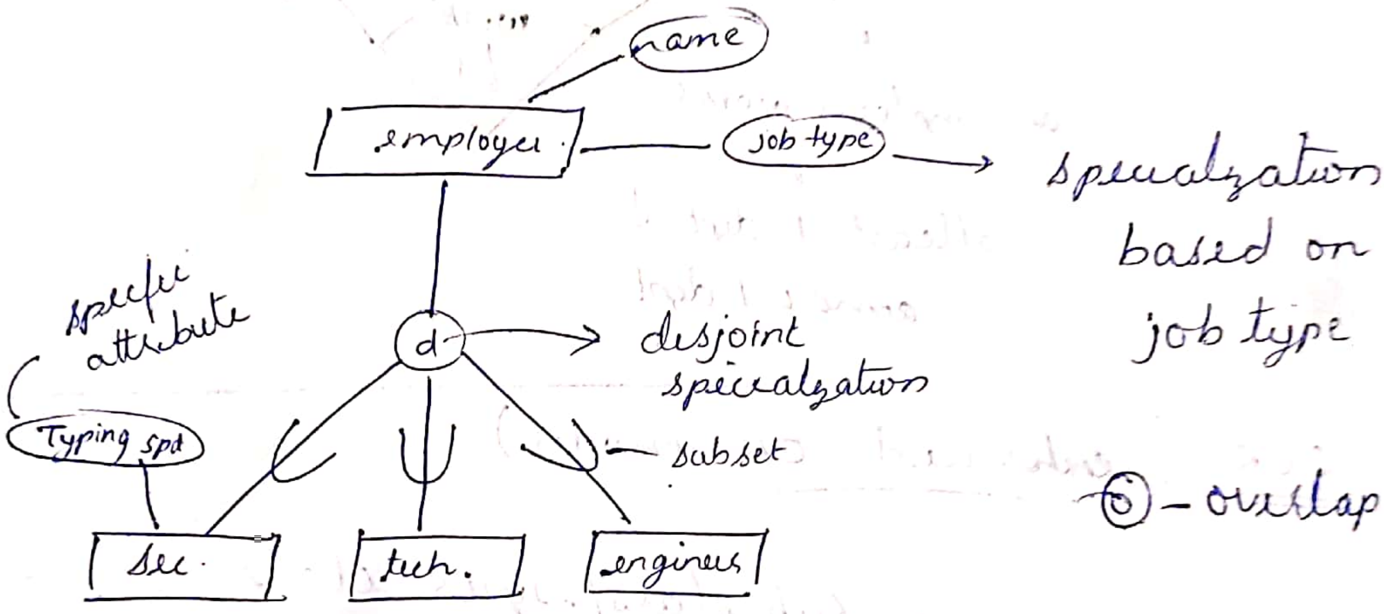
→ The subclass is said to be a 'is-a' relation with its super class.

→ entity of subclass inherits all attributes and relationships of super class

Scanned by CamScanner

→ specialization : process of defining a set of subclass of of superclass

→ Attributes specific to subclass is called specific attribute



Generalisation

→ reverse of specialisation

→ several class with common features are grouped to a super class.

→ If members of a subclass are identified from superclass based on

(i) a condn : subclass is predicate defined

(ii) an attribute : subclass is attribute defined

completeness constraint

total ⇒ every member of superclass must be a member of some subclass in specialisation / generalisation

4 specialisation/generalisation
- disjoint, tot
- disj., partial
- overlap, tot
- overlap, part.

specialisation/generalisation hierarchy, lattice

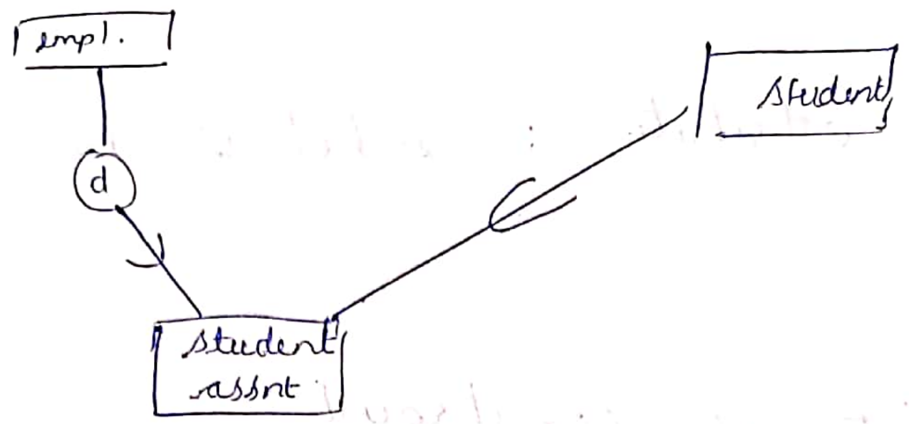
→ subclass can have multiple subclasses

→ If every subclass has one superclass
– forms hierarchy

→ subclass has more than 1 superclass
– form lattice

In university

eg:



lattice

Conceptual Model → logical Model

rows —, tuple (ordered set of values) → relationship

EF Codd

RDBMS

ACM

Schema of relation $\underset{\text{relation name}}{R}(\underbrace{A_1, A_2 \ldots A_n}_{\text{attributes}})$

⇒ domain may have data-type // format defined for it
set of valid values

⇒ Given $R(A_1, A_2 \ldots A_n)$ → schema

$$r(R) \subset dom(A_1) \times dom(A_2) \ldots \times dom(A_n)$$

⇒ $r(R)$ — a specific value or population of R

⇒ R ~~is~~ called intension of a relation/schema

r – extension of relation / population of relation

eg: $S_1 = \{0,1\}$ $S_2 = \{a,b,c\}$ $R(S_1, S_2)$

$r(R) = \{<0,a>, <0,b>, <1,c>\}$ is a 'state' or 'population' or 'extension' of relation R.

Relational Integrity constraints

Condns that must hold on all valid relation instances

Types of constraints
- key constraints
- entity integrity
- referential integrity

key constraints

A set of attr. that uniquely identify a row in a table

① superkey : set of attribute that is diff for each valid instances of relation

② key : minimal superkey (removal of any attribute makes it not a superkey)

③ Relation has several candidate key of which one is chosen as 1° key (underline

Entity integrity

Relational database schema : A set S of relation schemas that belong to same data base.

S - name of data base (.
↳ set of relations in a mini-world

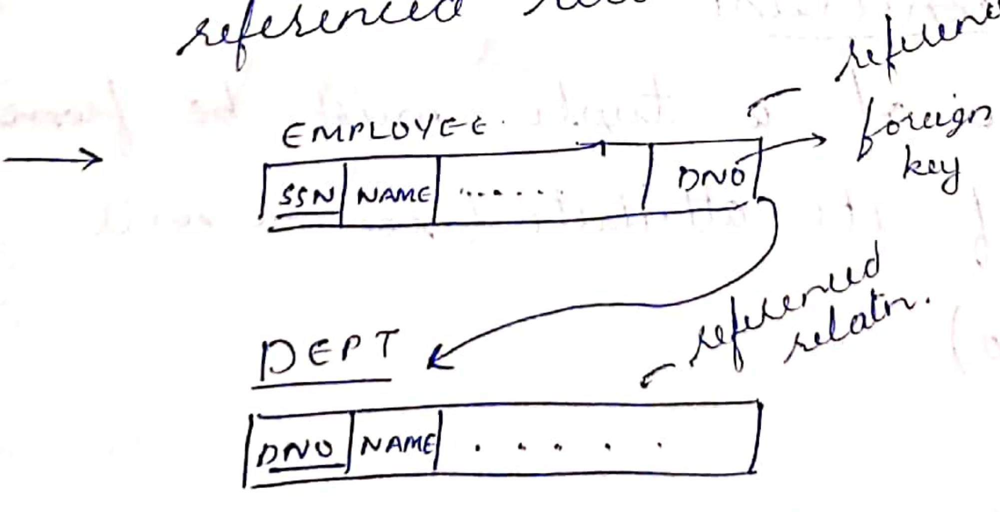
$S = \{ \underbrace{R_1, R_2 \ldots R_n}_{\text{tables in mini world}} \}$

entity integrity: 1° key of each relation schema R in S cannot have null values in any tuple of r(R)

(other attributes of a relation can be constrained to disallow null values though its not 1° key)

Referential Integrity

→ Involves 2 relations
→ used 2 specify relationship among tuples in 2 relations: referencing relation & referenced relation



~~value of~~ a foreign key attribute must be ~~present in~~ values in referencing relation's primary key

i.e. DNO given in instance of employee must be present in dept relation

→ Tuples in referencing relation R_1 have attributes (foreign key attributes) that reference the 1° key of referenced relation

[Attribute in referencing relation that refer to 1° key of referenced relation - foreign key]

→ This integrity can be represented as directed arc from $R_1.\underbrace{FK}_{\text{foreign key}}$ to $R_2.\underbrace{PK}_{\text{1° key}}$

ie $\boxed{t_1[FK] = t_2[PK]}$

t_1 - tuple in R_1
t_2 - tuple in R_2

→ domain constraint

Value of a tuple must be from domain of its attribute (can be null if allowed)

⇒ In referential integrity, foreign key can be null as long as its not 1° key of its own relation ssn of supervisor employee

	SSN			super_SSN

an attribute of a relation can refer to another attribute of same relation

Operation for changing a database:

ⓐ INSERT tuple (may violate all constraint)

ⓑ DELETE (can violate referential integrity)

ⓒ MODIFY (can violate all)

Actions that can be taken on violation:

① cancel the operation (RESTRICT/REJECT)

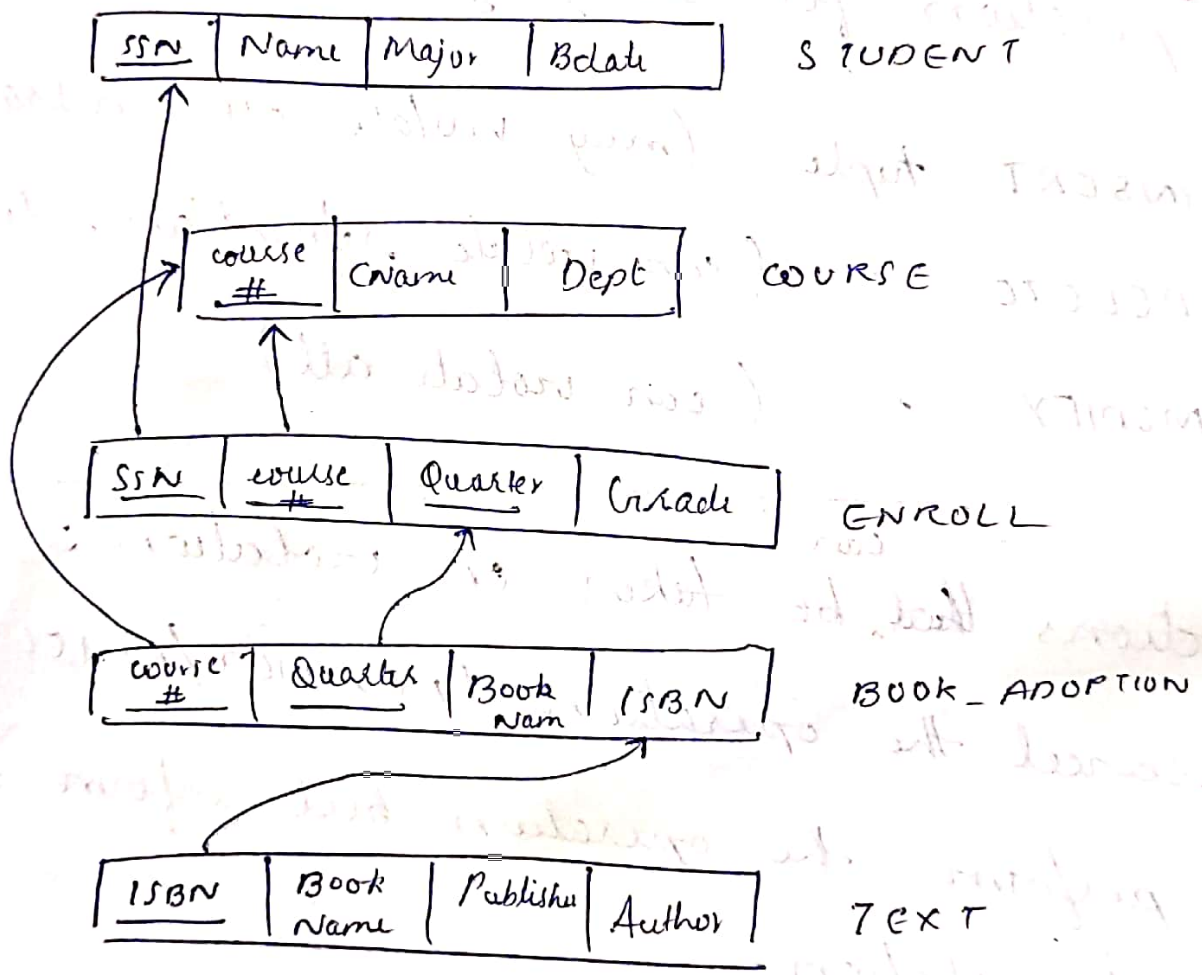
② perform the operation but inform user of violation

③ Trigger additional updates so that violation is corrected (CASCADE/NULL option)

④ execute a user-specified error-correction routine

Employee is having a <u>relation schema</u> while the whole relations in the company together forms the relational data base schema

HW



Mapping ER model to logical (relational) Model

① Mapping of regular entity type

→ for each reg. entity type create a relation with simple attributes of it.

→ Choose one of attributes as key.

→ If composite key is 1° key, then the simple attributes that forms the composite key are 1° keys

② Mapping of weak entity type

→ create relation including all simple attributes (& components of composite key) as attributes of owner entity type (E)

→ Include foreign key attributes of E as 1° key of dependent relation

(3) Mapping of 1:1 relation

identify entity type S & T that participate in that relation

(a) Foreign key approach.

Choose an entity type say S, having total participation and include a foreign key attribute in it that refers to 1° key of the other entity type T, participating in that relation

(4) Mapping 1:N

→ Identity relation S that represent participating entity type @ N side

→ Include as foreign key in S the 1° key of T that represent other entity participating in that relationship type

(5) Mapping M:N

→ create a new relation S

→ Include as foreign keys in S, the 1° key of relations that represent the participating entity types

→ their combination is 1° key of S

→ Also include simple attributes of M:N relationship type in S (components if composite)

(6) Mapping of multi valued attribute (A)

→ For each of them, create a new relation R

→ Include attributes of A & 1° key of relation K (which A is a part of) as foreign key

→ 1° key of R is combination of A & foreign key.

(7) Mapping N-ary relationships

→ create relation S

→ Include foreign key in S, the 1° keys of participating entity types

→ Also include the attributes (simple ones) of n-ary relationship type

Relational calculus.

→ no order of operation to specify how to retrieve data - specifies only what result must contain (diff b/w rel. algebra & rel. calculus)

→ Non-procedural or declarative language (whereas relational algebra is a procedural lang)

Tuple relationalal calculus (TRC)

Use variable to represent tuples of a relation.

eg: $\{t \mid condition(t)\}$ ⟹ set of all t that satisfies condn t.

eg: $\{t.FNAME, t.LNAME \mid \underbrace{EMP(t)}_{\text{range relatn.}} \text{ AND } \underbrace{t.salary > 50000}_{\text{selection condn}}\}$

Quantifiers

Existential : $\exists$ } Variable having quantifiers are said to be bounded
Universal : $\forall$ }

eg. $\{t.NAME \mid EMP(t) \text{ AND } (\exists d)(DEPT(d) \text{ and } d.DNAME = \text{'Research' and } d.DNUM = t.DNO)\}$

⟹ Retrieve name of employee who works in research dept, (if exists)

→ Emp. who work on all proj of dept 5

$$\{ e.NAME \mid EMPL(e) \text{ and } ((\forall x)(\text{not}(PROJ(x)) \text{ or } \text{not}(x.DNUM = 5)\}$$

Domain Relational calculus.