Module - 2

Data Modeling: Enhanced entity relationship model and relational data model.

2.1

Introduction:

1) Definitions:

a) Data model: A collection of concepts that can be used to describe the structure of a database.

b) Structure of a database: The datatypes, relationships and constraints that apply to the data.

2) How does data modeling help?

Most data models include a set of basic operations for specifying retrievals and updates on the database.

Data models also provide users with some database specific operations that can be performed on the database. These operations can be called user defined operations.

Eg: a user defined operation could be COMPUTE_GPA, which can be applied to the STUDENT object.

Benefits of data models:

1) Higher quality applications: Data modeling produces applications that are less likely to crash and easier to maintain.

2) Easier to change: Without data modeling, if we need to change the database or add new features, the code will become messy.

3) Reduced cost and time of application development: If you don't have a data model, then your team will have need to update both the database and the code. This can be very time consuming and expensive. With data modeling, you can simply modify your data model & update the existing application.

4) Early detection of data issues and errors: Data modeling approach creates an accurate view of how your users interact with your business. This level of insight provides critical information about where problems exist and how to best employ corrections.

5) Faster Application Performance: Data modeling makes an application run faster & more efficiently because it provides a high level plan for how the application should handle data.

6) Better documentation for long term maintenance.

Types of data models:

Data models categorise according to the types of concepts they use to describe the database structure:

1) High-level / Conceptual data models:

a) Provides concepts that are close to the way many users perceive data (similar to high-level programming languages).

b) Conceptual data models use concepts such as entities, attributes and relationships. An entity represents a real world object or concept, such as an employee or a project from the database.

An Attribute describes represents some property of interest that further describes an entity such as employee's name or salary.

A relationship among two or more entities represents an association among the entities, eg: A 'works-on' relationship between an employee & a project.

Conceptual data models are used in Entity-Relationship models.

2) low-level or physical data models: E

a) Concepts provided by low-level data models are generally meant for computer specialists, not for end users.

b) Physical data models provide concepts that describe the details of how data is stored on the computer media.

3) Representational or implementation data models:
(comes between high & low level data models)

a) Representational data models provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.

b) These are the most frequently used in traditional commercial DBMSs.
Relational data models use Representational data model.

Phases of data modeling:

1) Phase #1:
Requirements collection and analysis:
The database designer needs to interact extensively with domain experts and users to carry out the ~~this~~ task to fully characterize the data needs.

2) Phase #2:
Conceptual database design:

The dest designer chooses a data model and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database.
Typically, the conceptual design phase results in the creation of an entity-relationship diagram that provides a detailed overview of the enterprize.
Our focus at this point is to describe the data & their relationships, rather than on specifying physical storage details.

3) Phase #3:
Specification of functional requirements:
Users describe the kinds of operations (or transactions) that will be performed on the data.
The designer reviews the schema to ensure that it meets all functional requirements.
The conceptual schema is now fully developed.

4) Phase #4:
Logical design phase:
(The first phase of moving from an abstract data to actual implementation).
The designer maps the high level conceptual schema onto the implementation data model of the database system
The implementation data model is typically the relational data model and this step usually consists of mapping the conceptual schema defined using the ER model into a

relational model schem relation schema.

5) Phase #5:
Physical design phase:
Physical features of the database are defined here.

6) Phase #6:
Database system implementation & tuning:
The database & application programs are implemented, tested and eventually deployed for service. The physical schema of a database can be easily changed after an application has built. This is called tuning.

The entity relationship model:

1) The entity-relationship model represents the overall logical structure of a database. The employee E-R data model employs three basic concepts:
   a) Entity Sets
   b) ~~Relationship Sets~~ b) Attributes
   c) ~~Attributes~~ c) Relationship Sets

E-R model is represented using the E-R diagram. An E-R diagram can express the overall logical structure of a database graphically.
We will now look into the three basic concepts:

a) Entity Sets:

i) Definitions: • An entity is a "thing" or "object" in the real world that is distinguishable from all other objects.
eg: each person in a university.
{since each person is unique 👑}

each person has a unique ID number.

- Concrete entity: eg: A person, A book.
- Abstract entity: eg: A course, A flight reservation.

- Entity Set: A set of entities of the same type that share the same properties, or attributes.
eg: Entity Set 'instructor':
The set of all people who are instructors at a given university.
eg: Collection of all students from the student table.

ii) Entity Sets do not need to be disjoint (no element in common).
Eg: A person entity in a university could be a student, a teacher or both.

b) Attributes :

i) Definition : An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. Entities in an entity set may or may not have similar attributes

ii) Types of Attributes :

- Simple Attributes: These attributes cannot be further subdivided into components.
  eg: The ID of a student.

- Composite Attributes: An attribute that can split into components is a composite attribute.
  eg: Address can be further split into house address, street address, etc.

- Single-Valued Attributes: The attribute that takes up only a single value for each entity instance.
  eg: The age of a student.

VERY IMPORTANT NOTE:

Even though the formal definition of an Entity is "an object" and for an Entity Set is "A set of entities", in our course the definition of Entity is "CLASSES of objects", basically the formal definition of Entity Sets.

Thus, in our course, the "Instance of an Entity" means an object in the class. Basically the formal definition of an entity. eg of instance: The age of Student.

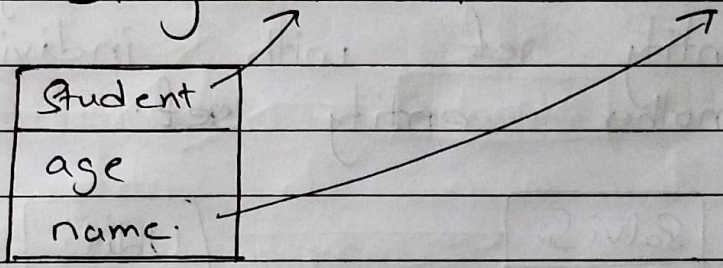
- Multi-valued Attribute:
  The attribute that takes up more than a single value for each entity instance
  Eg: Phone number of a student: landline or mobile.

- Derived Attribute:
  The attribute that can be derived from other attributes.
  Eg: The date of birth → age of a student.

iii) An attribute takes a 'NULL' value when an ~~instance~~ entity instance does not have a value for it.
'NULL' may indicate that the attribute is not applicable to that entity instance.

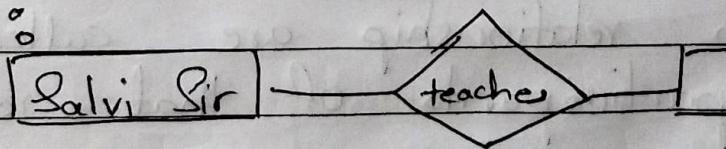
Eg: The middd middle_name attribute will be 'NULL' for a person with no middle name.

a+b) Thus, an entity set is represented as entity instance + attribute like:

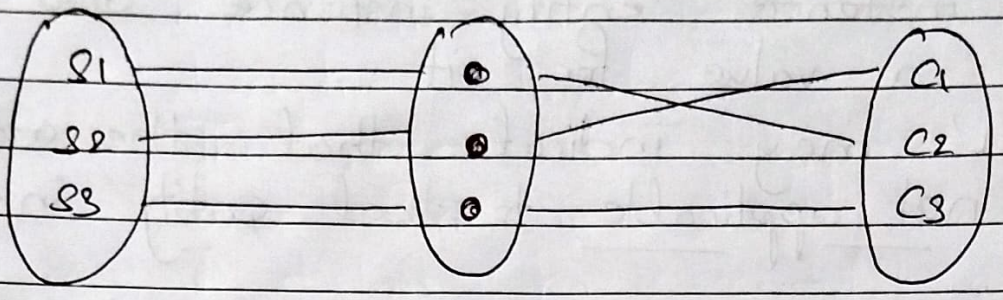


c) Relationship Sets:
i) Definitions:
• Relationship: An association among several entities.

eg:

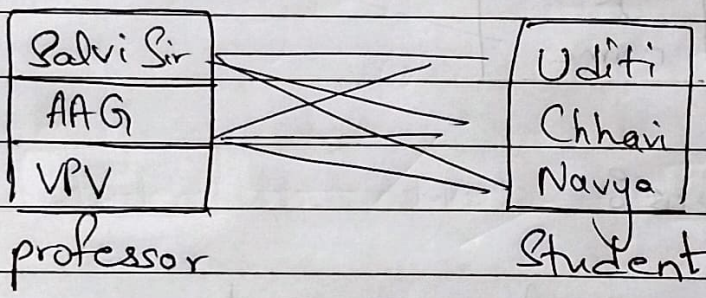


• Relationship Set: A set of relationships of the same type



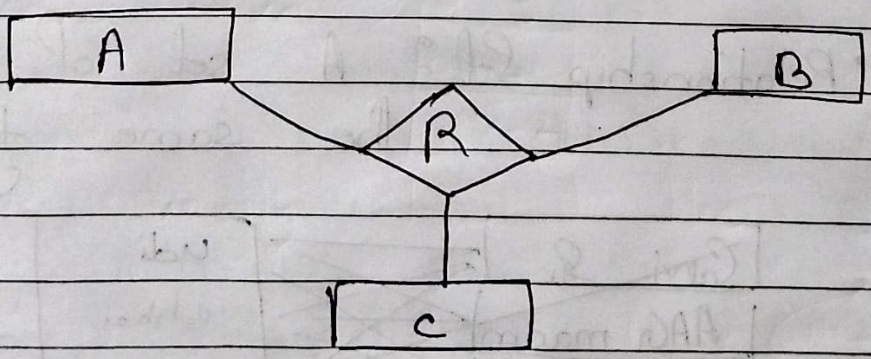
Student Entity | Relationship Entity set | Course Entity Set

- Relationship Instance: Represents an association between individual objects in the entity set with individual objects of another entity set.



- Participation: The entities that participate in a relationship are called the participants of that relationship set.

Eg:



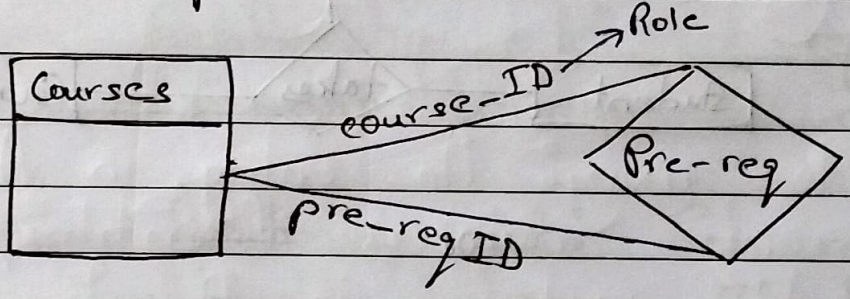
There can be more than or equal to 2 participants in a relationship set. Entities A, B & C are the participants of Relationship set R.

- Recursive Relationship Set: When the same entity participates in a relationship set more than once.

  Eg: We need to study 'RDBMS' as a prerequisite before 'Big Data' thus the entity 'Courses' will have a relationship with itself.

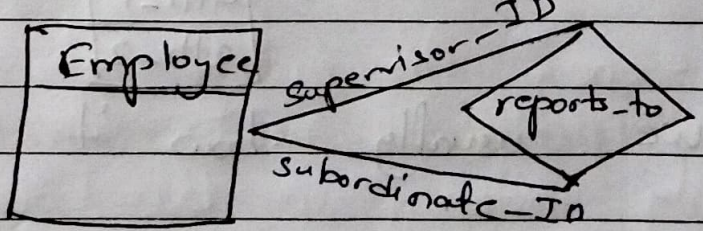
- Role: In a recursive relationship set, the role names are necessary to specify how an entity participates in a relationship set.

Eg:

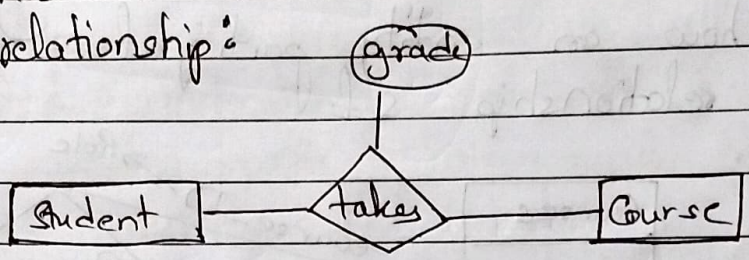


prereq-ID will have the course ID of RDBMS and Course-ID will have the ID of 'Big Data'.

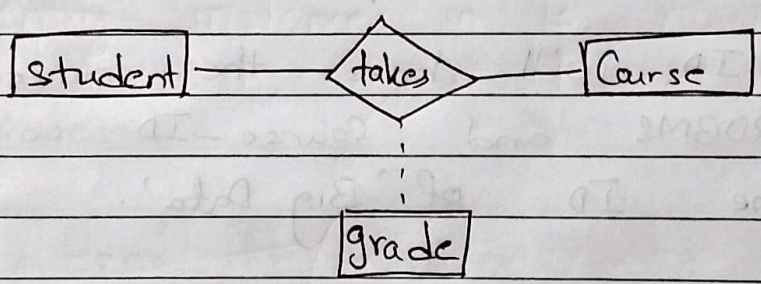
Another Eg: An employee in a corporation could be the supervisor of another employee.



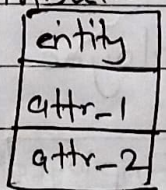
• Descriptive Attributes: A relationship too can have attributes. These attributes are called descriptive attributes.
The need for descriptive attributes may arise ~~from~~ in such cases:
A student has scored a certain grade in one particular subject. Grade, thus, cannot be an attribute of student / course alone, so it has to be an attribute of the relationship:



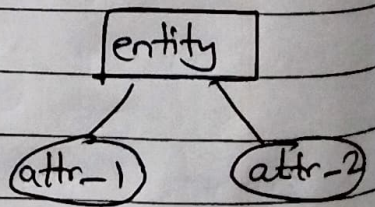
or



✱ Note: In books, attributes of an entity set are shown like



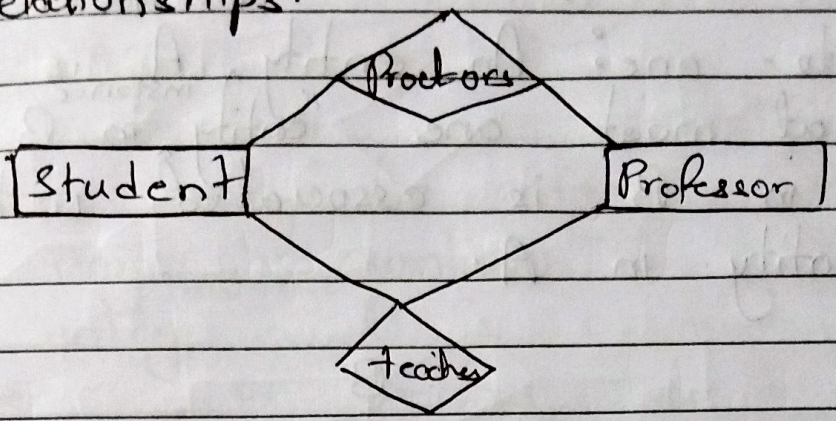
but we usually show it like



Both representations are correct, but in exam we the 2nd representation, simply because that's what we've been 'taught'.

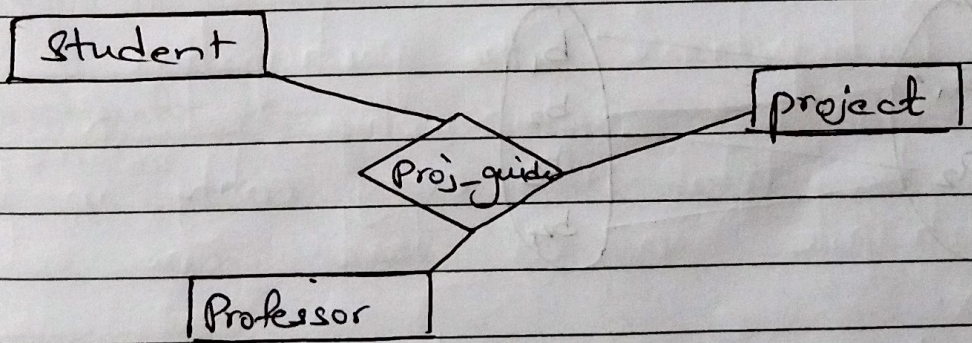
Note:

ii) Two entity sets can have multiple relationships.



iii) n- ary relationship sets:

- Binary Relationship set: Set represents relationship between 2 entity sets.
- Ternary Relationship set: Set represents relationship between 3 entity sets.
- n-ary Relationship set: Set represents relationship between n entity sets.

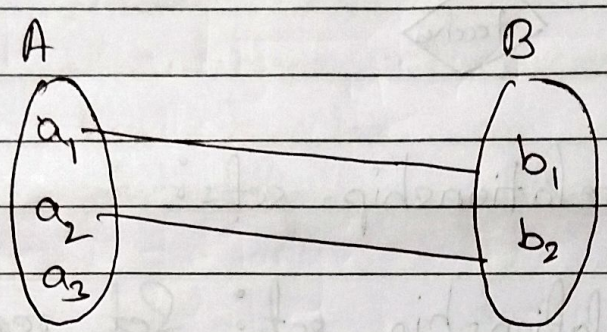


2) Mapping Cardinalities: (or cardinality ratios)

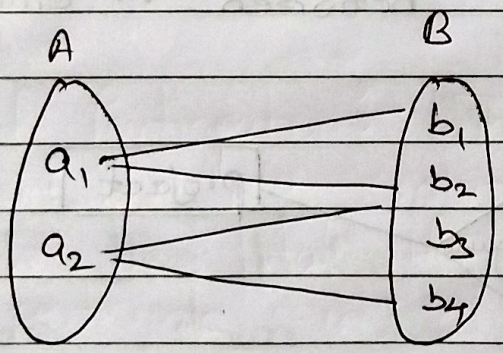
a) Expresses the number of entity instances to which another entity instance can be associated via a relationship.

b) Types of Mapping Cardinalities:
(for Binary relationship)

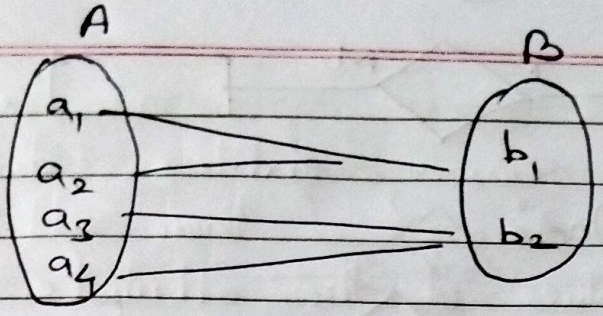
i) One-to-one: An entity instance A is associated with at most one entity instance in B, & an entity instance in B is associated with at most one entity instance in A.



ii) One-to-many: An entity instance in A is associated with any numbers of entity instance in B.
An entity instance in B, however, can be associated with at most one entity instance in A.

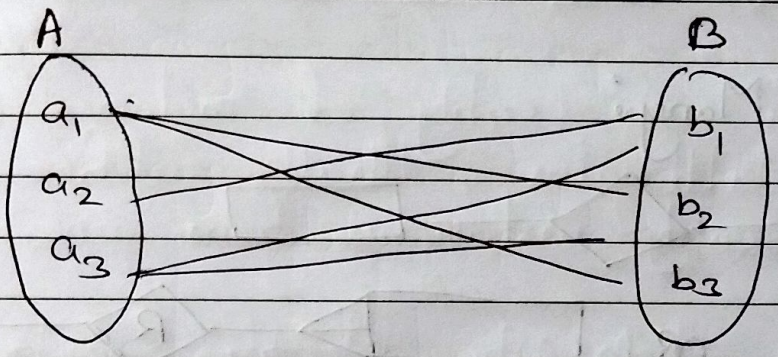


iii) Many-to-one: An entity instance in A is associated with at most one entity instance in B.
An entity instance in B, however can be associated with any number of entity instances in A.



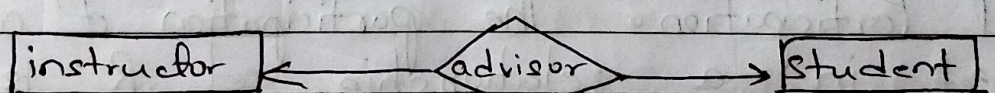
instance

iv) Many - to - many: An entity in A is associated with any number of entities in B, & an entity in B is associated with an number of entities in A.

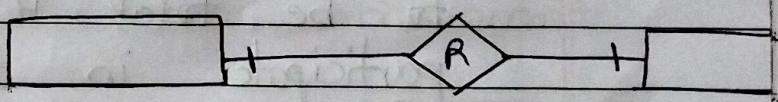


c) Formal representation of relationship cardinalities:

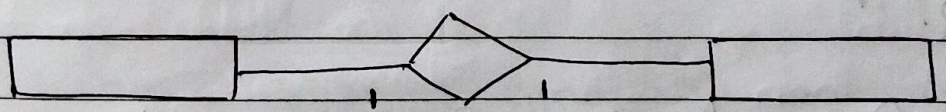
i) One - to - one:



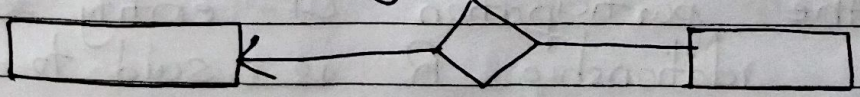
OR



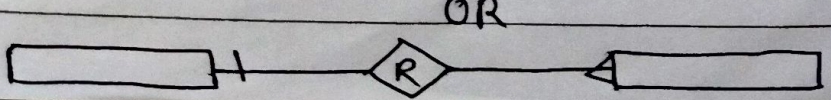
OR



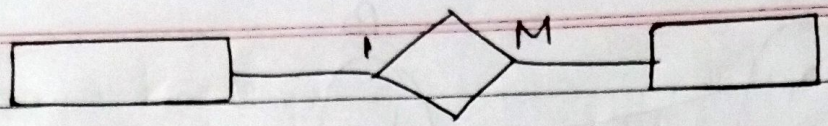
ii) One - to - many:



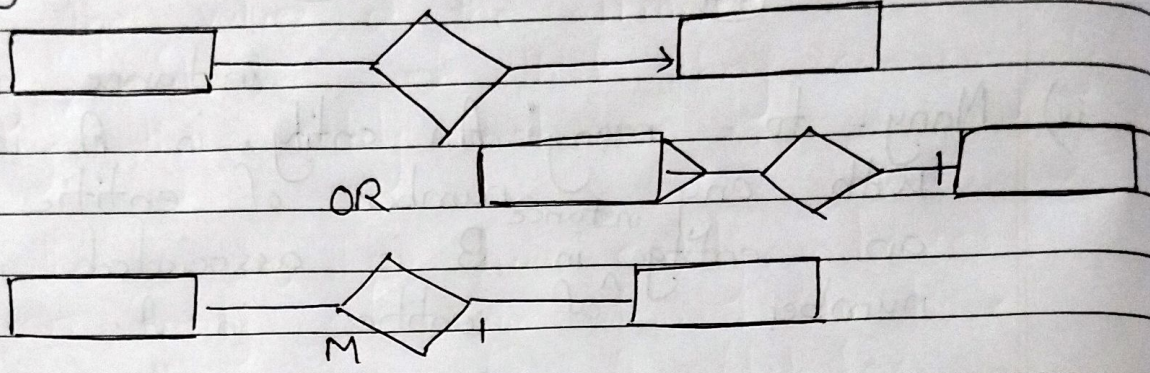
OR



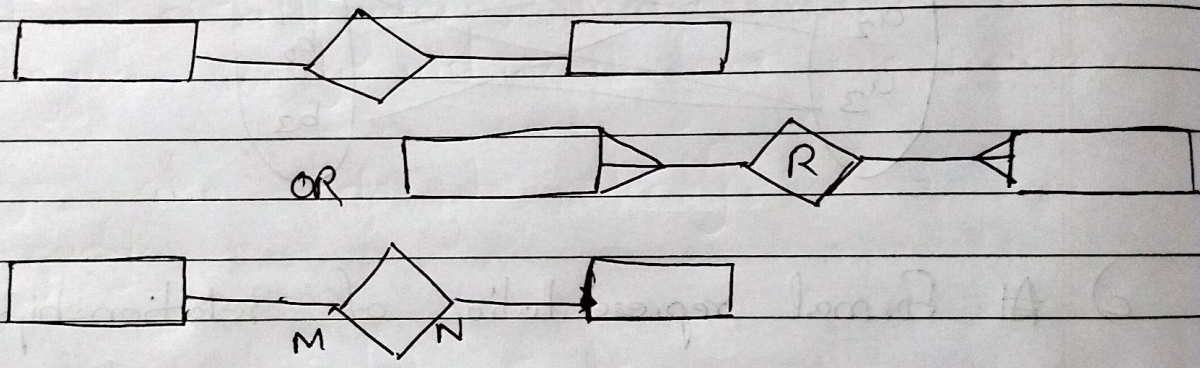
OR



iii) Many - to - One:



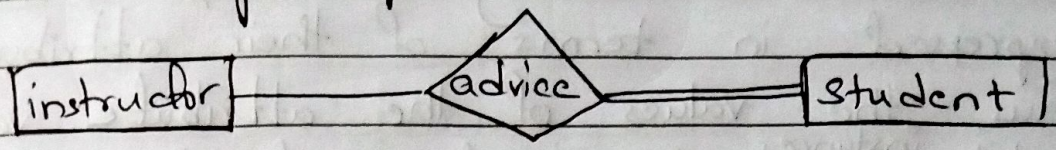
iv) Many - to - Many:



d) Total Participation: The participation of an Entity set in a relationship set R is said to be total if every entity instance in E must participate in at least one relationship in R.

Partial Participation: If some entity instances in E do not participate in relationships in R, the participation of entity set in R E in relationship R is said to be partial.

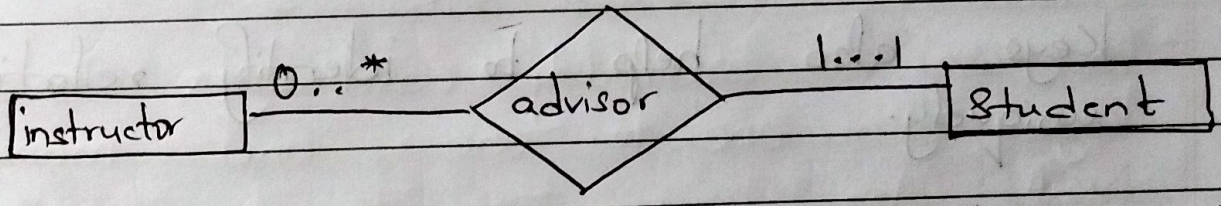
Eg: Every student needs an instructor, thus total participation in student's side



But every instructor does not need to have a student, hence partial participation of instructor side.

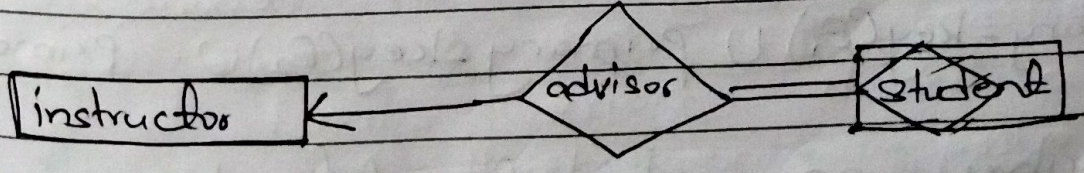
A line may have an associated minimum and maximum cardinality, shown in the form l...h, where l is the minimum and h is the maximum cardinality. A '*' is used in h to represent no upper limit.

Eg:



Note that, the correct way to read the above diagram is that the instructor can advice none or many students, but a student can be advised by only 1 instructor. The constraints are written on opposite ends.

* The above diagram can also be represented as:



3) Primary key :

a) The differences among entities, instances should be expressed in terms of their attributes. Thus, the values of the attribute values of an entity instance must be such that they can uniquely identify the entity instance.

In other words, no two entity instances in an entity set are allowed to have exactly the value for all attributes.

A key for an entity set instance is a set of attributes that suffice to distinguish entity instances from one other.

Keys also help to identify relationships uniquely.

b) Relationship Sets : Let R be a relationship set involving entity sets $E_1, E_2, \ldots E_n$. Let primary-key(E_i) denote the set of attributes that forms the primary-key for entity set E_i.

If relationship set has no attribute of its own, then

primary-key(E_1) $\cup$ primary-key(E_2) $\cup \ldots$ primary-key(E_n)

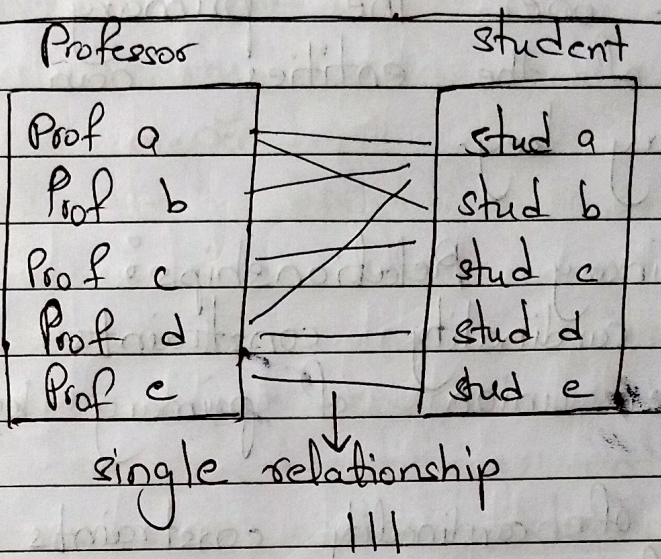
describes an individual relationship in relationship set R.

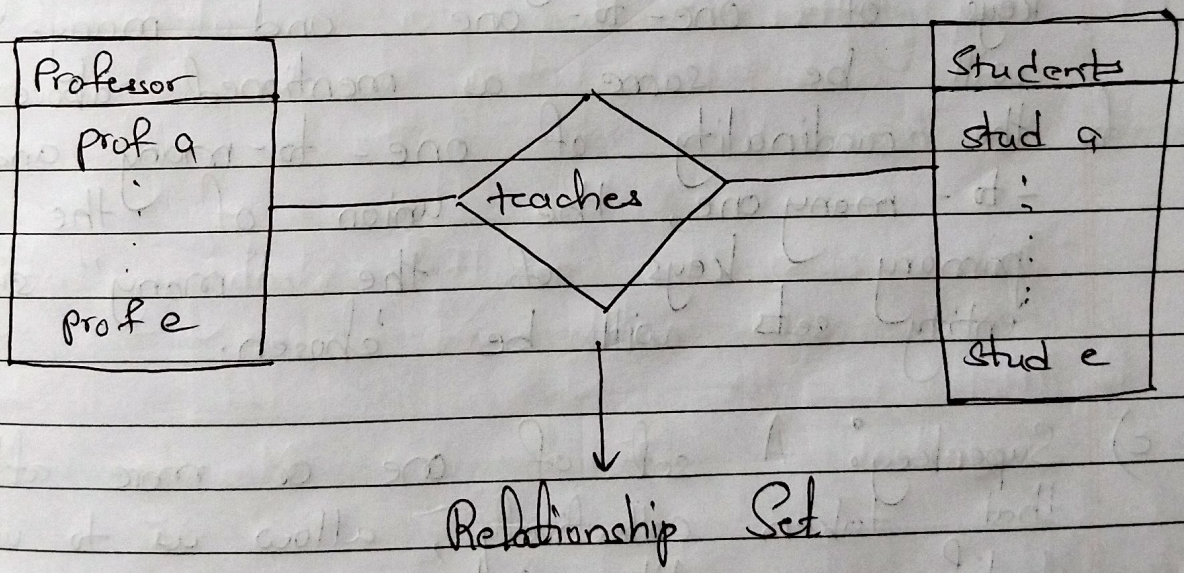
If the relationship set R has attributes $a_1, a_2, \ldots$ an associated with it, then

primary_key(E_1) $\cup$ primary_key(E_2) $\cup \ldots \cup$ primary_key(E_n) $\cup \{a_1, a_2, \ldots a_n\}$

describes an individual relationship in relationship set R.

* Note:





- Mapping Cardinality & primary key:

i) One-to-Many-to-Many: The primary key will be the union of both entities.

ii) One-to-many: The primary key of the "many" side is used as primary key.

iii) Many-to-one: The primary key of the "many" side is used as primary key.

iv) One-to-one: The primary key of either one of the entities can be chosen as primary key.

- Non-binary Relationships:
  If no cardinality constraints are present, then the union of primary keys will be chosen.
  In case of cardinality constraints, the primary keys of one-to-one and many-to-many will be same as mentioned above, but for the cardinality of one-to-many and many-to-one, the union of the primary keys of the "many" side entity sets will be chosen.

c) Superkey: A set of one or more attributes that, taken collectively, allow us to uniquely identify an entity instance.
A superkey may contain redundant attributes. eg: 'ID & name' is a superkey even though 'name' cannot be a superkey, since multiple people can have same name. ID is repeating twice.

Candidate key: A candidate key is a superkey that does not have any redundant values / attributes.

There can be multiple candidate keys, one of the candidate keys is then chosen to be a primary key.

4) Weak Entity Sets:

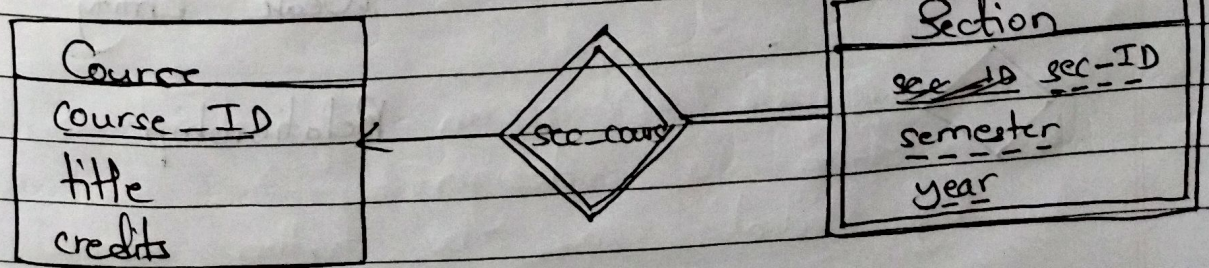
A weak entity set is one whose existence is dependent on another entity set, called its identifying entity set.

Instead of associating a primary key with a weak entity, we use the primary key of the identifying entity, along with extra attributes, called discriminator attributes to uniquely identify a weak entity.

An entity set that is not weak is termed as a strong entity set.

A weak entity must have an identifying strong entity set, thus, it is existence dependent. The relationship associating the weak entity set with the identifying entity set is said to be called identifying relationship.

Eg:



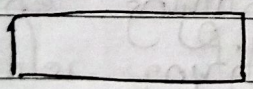
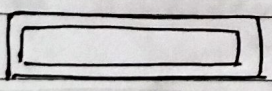
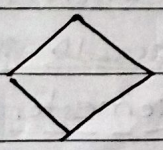
- Primary key of the weak entity set is the primary key of the identifying entity set + the descriptive attributes of the weak entity.

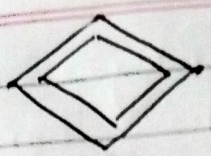
- The weak entity set always has total participation with the strong entity set, hence double line is used to join the weak entity set with the relationship set.

- The rhombus for relationship set joining a strong entity set with a weak entity set has double line.

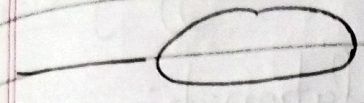
- A weak entity set can have relations with any other strong or weak entity sets & can also have a weak entity set of its own.

- There can be multiple identifying entity set for a weak entity set.

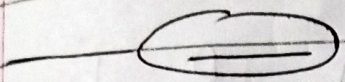
Symbol	Meaning
	Entity
	Weak Entity
	Relationship



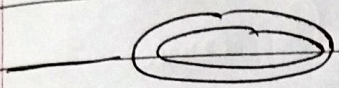
Identifying Relationship



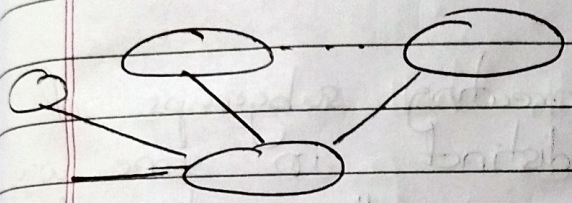
Attribute



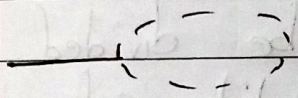
Key Attribute



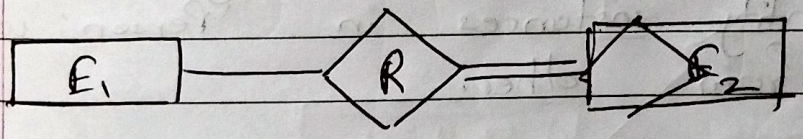
Multivalued Attribute



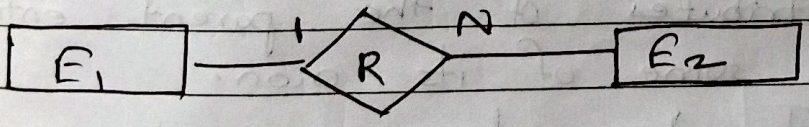
Composite Attribute



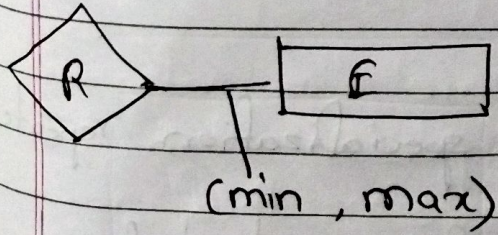
Derived Attribute



Total participation of E_2 in R.



Cardinality ratio 1:N for $E_1 : E_2$ in R.



Structural constraint (min, max) on participation of E in R.

2.2

Enhanced - Entity - Relationship (EER) - Model
Generalization, specialization and Aggregation.

Enhanced / Extended entity relationship diagrams are basically more detailed version of ER diagrams.
EER models are helpful tools for designing databases with high level & models.

1) Specialization

a) Definition: The process of creating subgroups of entity instances that are distinct in some way from other entity instances in the entity set.

Eg: 'Person' Entity set can be divided into 'Employee' and 'student' entity sets because some entity instances in 'Person' were distinct from other.

b) Attributes: The subgroups created will have all the attributes of the parent entity set, including some of its own.
Eg: In the above example, employee will have the additional attribute 'salary'

c) More subgroups: We can specialization repeatedly to refine a design.
Eg: 'Student' can be further divided into 'PG' and 'UG'.

d) Specialization Constraints:

i) Incomplete / Partial specialization:
Not all entity instances are included in the subgroups formed after specialization.
These ~~attri~~ entity instances do not have any attributes of their own apart from parent entity set's attributes.

Eg: A 'visitor' has no attribute apart from the ones already present in 'person'.

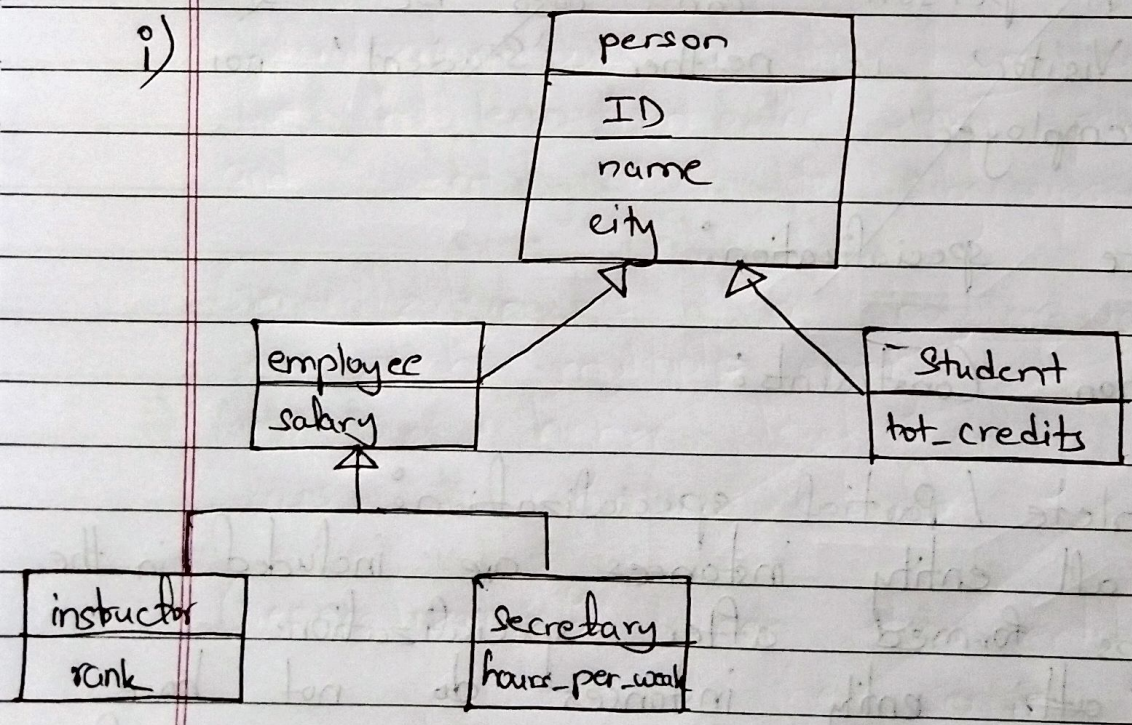
ii) Complete Specialization: All entity instances have unique attributes apart from the attributes already present in parent entity set.

iii) Disjoint specialization: An entity instance (or exclusive) of parent entity set will only be in one specialized subgroup.

iv) Overlapping specialization: An entity instance of parent entity set can be a member of more than one specialized subgroup.

e) Representation: Arrow with hollow head, pointing towards parent entity set:

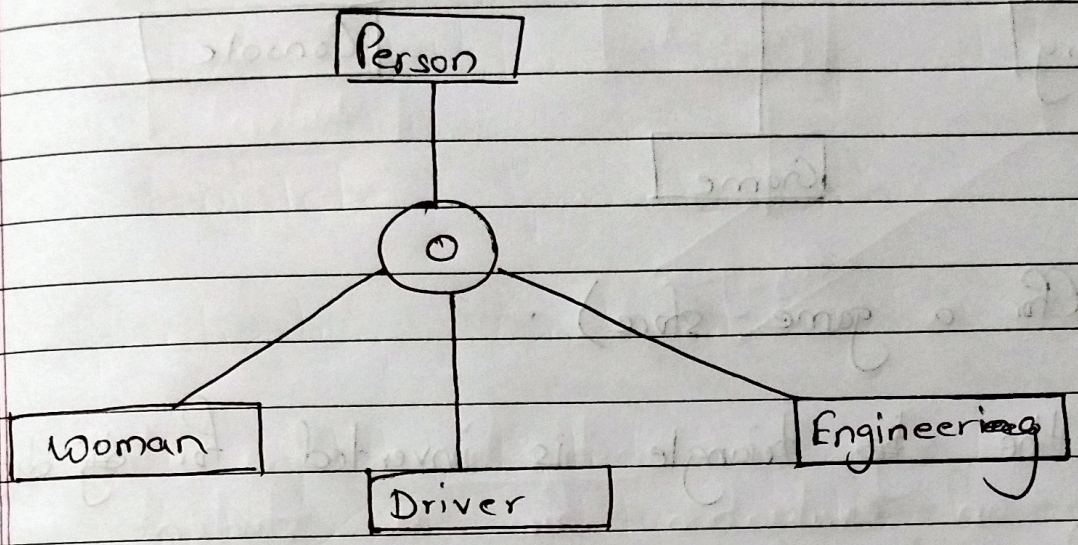
i)



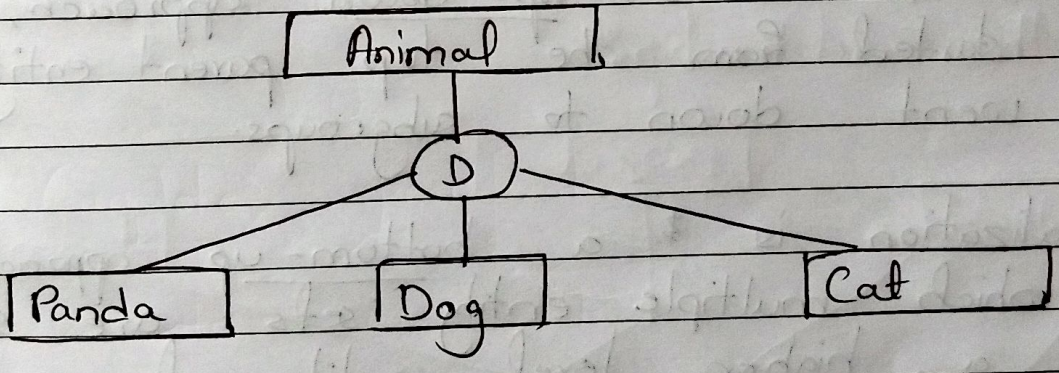
- student & employee are overlapping specializations (because a person can be both student and employee), thus represented by separate arrows.

- Instructor and Secretary in disjoint specialization thus, a single arrow is used.

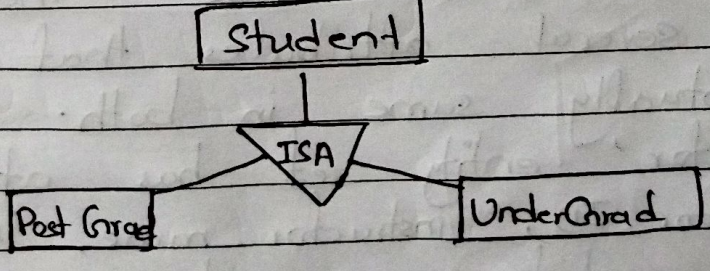
ii) Another way of representation of overlapping specialization:



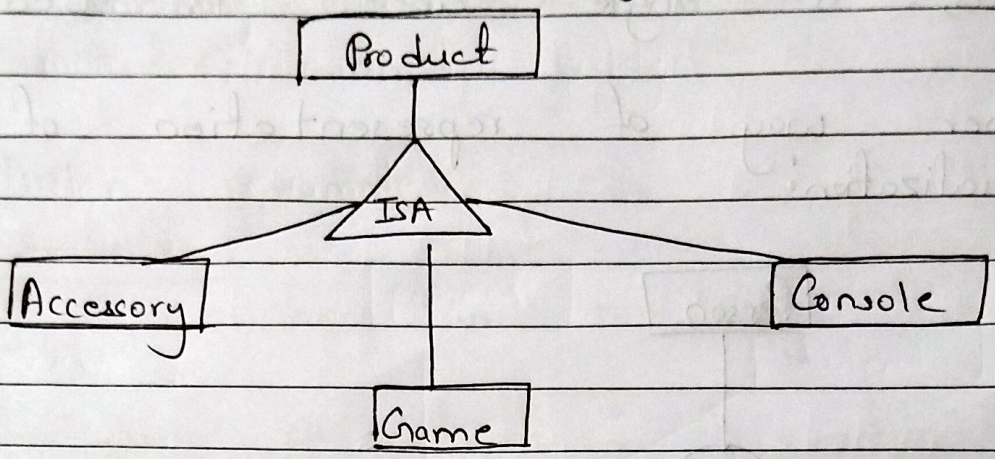
Disjoint specialization:



iii) Another way of representing ~~with no constraint~~ Disjoint specialization:



Another way of representing overlapping specialization:



(For a game store).

Notice the ~~the~~ triangle is inverted for ~~dij~~ disjoint).

2) Generalization

Specialization was a top-down approach, as we started from the top parent entity set and went down to subgroups.

Generalization is a bottom-up approach, in which multiple entity sets are synthesized into a higher level entity set on the basis of common features.

a) Generalization: When two entity sets are similar in the sense that they have several attributes that are conceptually same in both.

Eg: 'instructor' entity set has attributes instructor_ID, instructor_name, instructor_salary, rank which are quite similar to the

attributes of 'secretary' which are secretary_ID, secretary_name, secretary_salary & hours_per_week. Thus, these two entities could be generalized.
In the generalized entity set, the common attributes should have a common names which can be ID, name, salary.
The higher entity set can be named employee.

b) Superclass: The higher-level entity set after generalization / specialization.

Subclass: The lower-level entity set after generalization / specialization.

c) Generalization is simple inversion of specialization.

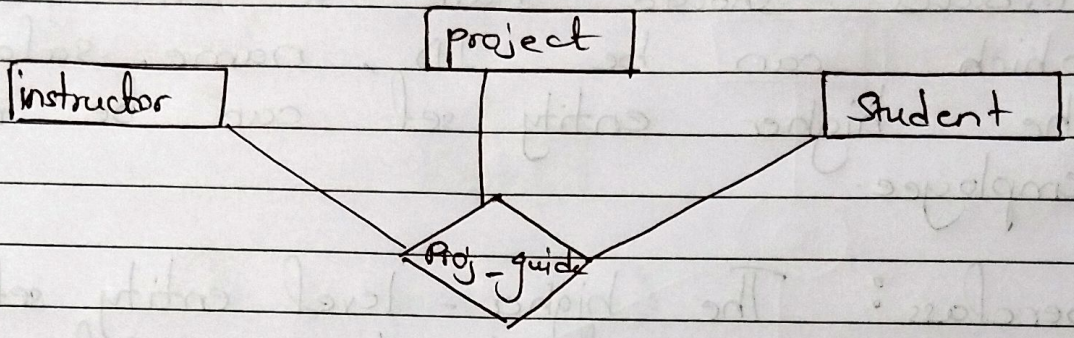
* Distinguishing an entity set: Specialization.
Is useful when difference between entity instances need to be emphasized.

* Synthesizing an entity set: Generalization.
Is useful to emphasize the similarities between lower-level entity sets.

The representation of generalization is same as specialization.

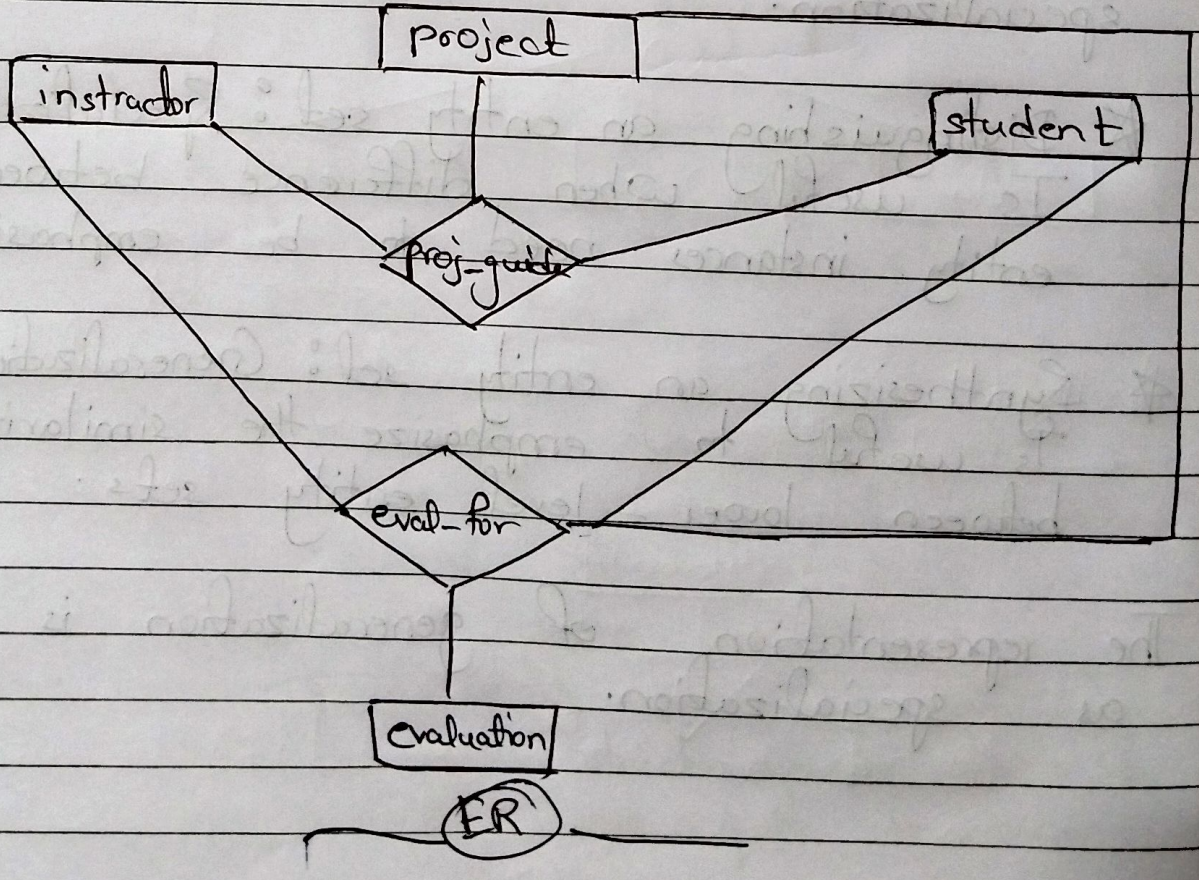
3) Aggregation: Aggregation in DBMS is a process of combining two or more entities to form a more meaningful entity.

Consider example:



Now, lets say the instructor has to prepare an evaluation for a student on his/her project.

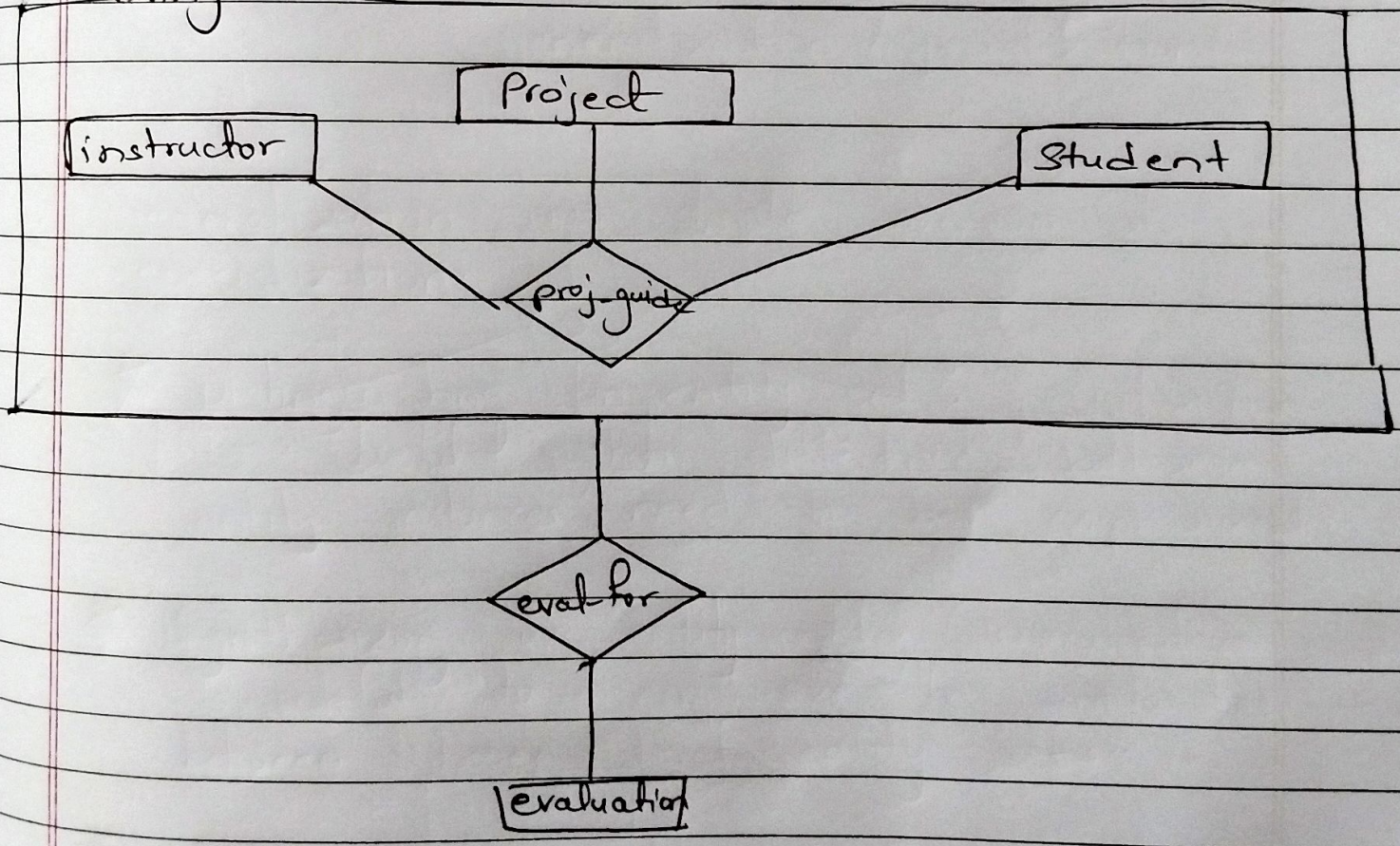
All 3 entities will have to participate in the relationship:



(Here, we cannot combine relationship set 'proj_guide' and 'eval_for' because, even though both are connected to all 3 entities, some instructor, student, project combinations may not have an associated evaluation).

Aggregation will combine the 3 entities & proj_guide into one single entity. Since every evaluations will happen only if proje

This single entity is a higher level entity.



This higher level entity set can be used as any other entity set. Aggregation feature is provided by EER model.

2.3

Relational Model

Introduction:

1) The relational model represents the database as a collection of relations. Each table resembles a table of values or a flat file of records. Flat file is called a "flat file" because each record has a simple linear or flat structure.

However, a flat file database stores data in a single table structure. A relational database uses multiple table structures, where tables are related to each other.

In simple words, the relational model represents how data is stored in Relational databases.

A relation is thought of as a table of values, each row in the table represents a collection of data values.

For example, an Enty Entity set STUDENT will have rows, where each row describes a particular STUDENT entity instance. The column names 'Name', 'Student_roll#', 'class' specify how to interpret the data values in each row. Each value in 1 column has the same data type.

STUDENT

Roll-No	NAME	ADDRESS	Phone	Age
156	Goos	Norway Mumbai	1234	20
188	Navya	Dogeland		19
194	Udit	Pengland	6969	19
199	Anirudha	Mumbai	22011	20
208	Aditya	Mumbai	1711	20
218	Chhavi	Kermitland	0206	19

2) Terminologies :

a) Domain : Domain defines the permitted range of values for an attribute of an entity.

b) Attribute : Defines the name of a column in a particular table.
Eg: In the above table, 'NAME', 'Roll-No' etc are attributes of a student. Every attribute has an attribute domain.

c) ~~Relational Instance :~~

c) Tuple : Each row in a relation (table) is known as tuple.
Eg:
| 188 | Navya | Dogeland | --- | 19 |

d) Relational Instance : Relational instance is represented by a finite set of tuples.
The relation / table above is a relation instance.

Tuples cannot be duplicated in a relational instance. It can also change after data manipulation (insertion, deletion, updation).

e) Degree: The number of relation attributes in the relation is known as the degree of the relation.
The relation above had degree 5.

f) Cardinality: The number of tuples in a relation is known as cardinality.
In the above example, the cardinality was 6.

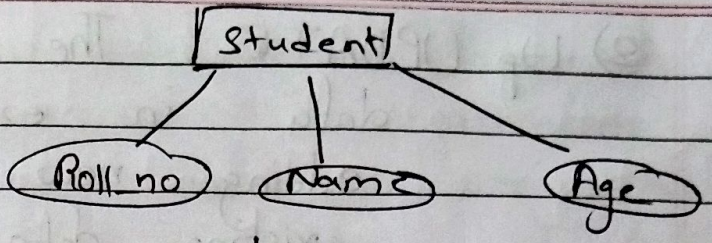
g) Column: Column represents the set of values for a particular attribute.

h) NULL values: The value which is not known or unaivailable is called NULL value.
Eg: Narya's phone_num was NULL value.

i) Relational Schema: A relational schema uses tables to show a relationship between two entities. (using foreign key, which will be defined later).

j) Relation key: Basically, the primary key of a relational model. Relation key can be used to identify a particular tuple uniquely.
Eg: In the above example, relation key was 'Roll_no'.

ER ~~AE~~ ER →



STUDENT

Relational Model.

Roll_no	Name	Age
—	—	—
—	—	—
—	—	—

Data Manipulation :

Four basic data manipulation operations are performed ~~in~~ on relational database models:

INSERT, DELETE, UPDATE, SELECT

1) INSERT : INSERT operation adds rows to a table.

ID	Name	Status
1	a	active
2	b	active
3	c	inactive

→

ID	Name	Status
1	a	active
2	b	"
3	e	inactive
4	d	"

2) DELETE : The DELETE operation removes rows from a record set.

ID	Name	Status
1	a	active
2	b	"
3	c	Inactive
4	d	"

→

ID	Name	Status
1	a	active
2	b	active
4	d	inactive

3) UPDATE: The UPDATE operation changes data in existing rows either by adding new data or modifying existing data.

ID	Name	Status
1	a	active
2	b	active
4	d	inactive

→

ID	Name	Status
1	a	active
2	b	active
4	d	active

4) SELECT: The SELECT statement operation allows the user to extract data from tables.

ID	Name	Status
1	a	active
2	b	active
4	d	active

→

ID	Name	Status
1	a	active
2	b	active
4	d	active

The data manipulations are possible because of SQL data manipulation language (DML), which is covered in module 3.

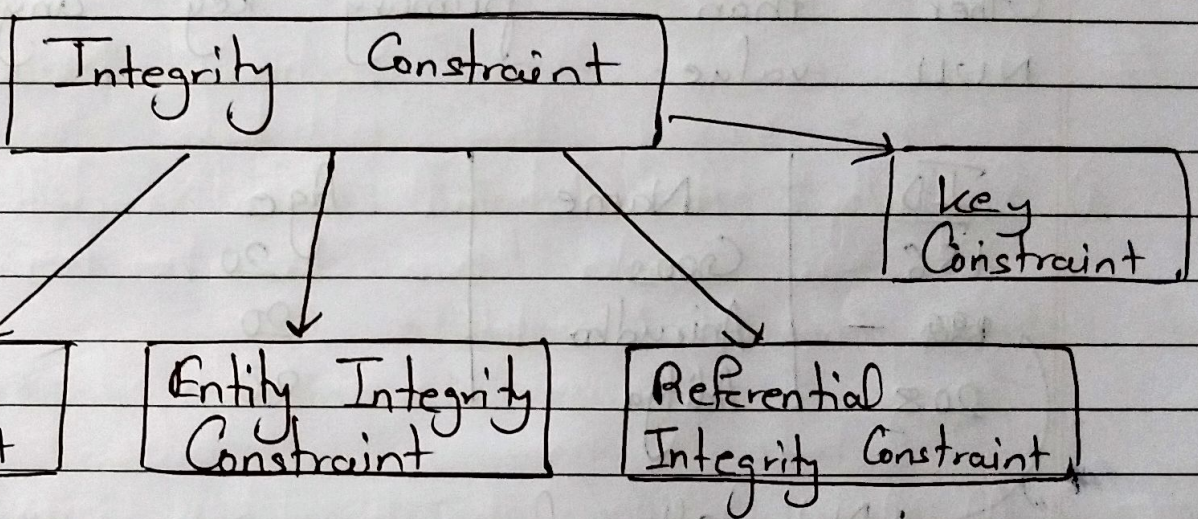
Integrity Constraints: 1) We learned in module 1 that data manipulation like insertion or deletion can cause integrity issues.

2. When consistency constraints are changed, database has to change accordingly. This change can cause integrity issues like false information.

Integrity Constraints ensure that the data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.

Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraints:



a) Domain Constraints: Domain constraints define the valid set of values for an attribute.

Data types can be string, character, integer, time, date, currency, etc.

ID	Name	Age
194	Uditi	19
188	Navya	19
218	Chhavi	Dead → Not valid because Age can only be integer values.

b) Entity Integrity Constraints: The entity integrity constraint states that primary key value cannot be NULL.

Because the primary key value is used to identify individual rows or an entity instance or a tuple.
If primary key is NULL, then we can't identify those rows.
Other than primary key anything can have NULL value.

ID	Name	Age
156	Gaos	20
[redacted]	Anirudha	20
208	Aditya	20

→ Not allowed

c) Referential Integrity Constraints:
~~(This will be understood after 2.4 is covered later)~~

This constraint is specified between two tables.
If table 2 refers to table 1 using a foreign key, then every value of the foreign key in table 2 must be null or be available in table 1.

Table 1

D_No	D_Location
11	Mumbai
24	Delhi
13	Noida

↕ Relationship

Foreign key ←

D_No	Emp_Name	Age
11	Sania	19
24	Tejas	20
18	Surabhi	20
13	Bhavya	20

→ Not allowed because table 1 does not contain D_No 18.

Foreign key: As we can tell by above example a foreign key is a set of attributes in a table that refers to the primary key of another table. The foreign key links these two tables.
In ER diagram we can directly show a relation between two entity sets, but in relational model, to show that two tables are related we have to use foreign keys.

d) Key Constraints: An attribute that can uniquely identify a tuple in a relation is called the key / Primary key of the table. The value of the key attribute for different tuples in the relation has to be unique.

ID	Name	Age
156	Goos	20
188	Navya	19
218	Chhavi	19
218	Surabhi	20

→ Not allowed, because key attribute's value is duplicated.

~~Advantages of the relational Model:~~

1) ~~Simplicity:~~ A relational data model in DBMS is simpler than the ~~her~~ hierarchical & network model.

Advantages of the relational Model:

1) Ease-of-use: It's easy to run complex queries using SQL.

2) ACID Compliance: Relational databases support ACID (Atomicity, Consistency, Isolation, Durability) performance to ensure data validity regardless of mishaps.

3) Structural Independence: The relational database is only considered with data and not with a structure. This improves the performance of the model.

4) Simplicity: Relational model in DBMS is easy as tables consisting of rows and columns are quite natural and simple to understand.

5) Data Independence: The structure of Relational database can be changed without having to change any application.

6) Scalable: The size of a database using relational model can be easily increased.

2.4

Mapping EER Model to Relational Model

We will first look into how the ER model is mapped to a relational model.
Then we will look into how EER is mapped. The only difference will be the addition of Union, Generalization and Specialization.

1) Introduction: Both ER model and Relational database model are abstract, logical representations of real-world enterprises.

The ER model is easy to understand but it is not suitable for implementation directly. Thus relational models are made. Relational models can be easily implemented by RDBMS like mySQL.

2) Lets look into how different types of entities & relationships can be mapped to ~~or~~ relational database model.

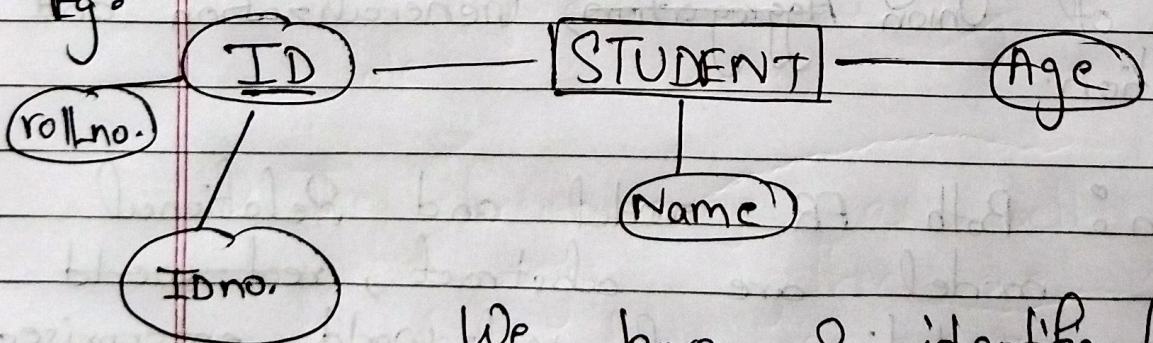
a) Representation of Strong Entity Sets:

i) The attributes of the strong entity set will all be single columns for the relational model.

ii) The primary key attribute for that entity set will be the key for the relational model as well.

iii) If the primary key attribute is a composite attribute (meaning attribute having its own attributes), then all the simple attributes of that composite attribute will be the keys for our relational schema.

Eg:



We have 2 identification numbers (unique) in somaiya, our roll number (160101...) and our ID no. (that we use to issue books from library) which is written on our ID card.

The relational model for the above ER diagram is:

STUDENT

| ID_no. | Roll_no. | Name | Age |

NOTE:

There can only be one primary key in an ER or relational model.
When multiple attributes or Columns are underlined, it means that those attributes/ columns create composite keys.
Composite keys simply mean that those underlined Attributes / Columns compose to create an identifying attribute. (More info in 2.2)

The relations / tables that are created are also called 'entity relations'.

Same rule applies for normal attributes as well. All simple attributes of that composite attribute will be columns for the table.

b) Representation of Weak Entity Sets:

i) Like strong entity sets, each attribute will be written as column for the table of the weak entity set and also all simple attributes of the composite attributes will be columns.

ii) The primary key of the owner / strong entity set for the weak entity set will be used as the foreign key of that weak entity set.

iii) The partial keys of the weak entity set is represented as key for the table as well.

iv) The foreign key as well as the partial keys together are composite keys for that table representing weak entity set.

v) There can be multiple foreign keys as well, if a weak entity set has multiple corresponding strong entity set.
Also, it is not necessary that a weak entity set will always have a partial key.

vi) Partial key: This Is an attribute that can uniquely identify only partial entity instances. Weak entity sets have only partial key and they need the primary key from the corresponding strong entity set to fully identify each entity instance uniquely.

vii) Always create the ~~stro~~ relational table for the strong entity set before its corresponding weak entity set for easily identifying the foreign key.

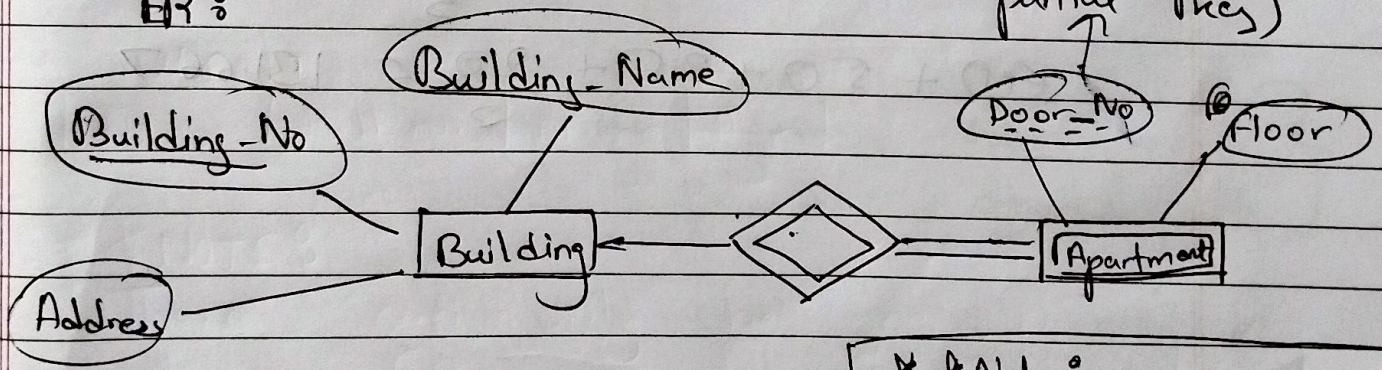
viii) Parent strong Entity Set's table:

Building

Building_No.	Building_Name	Address

(Dotted line to represent partial key)

ER:



Weak Entity Set's table:

Apartment

Building_No.	Door_No	Floor

↓

Foreign key

**Note:

It is good practice to change the name of primary key before using it as a foreign key, but this is not necessary. If we did that here, we could use the term 'B_no.'

c) Representation of Binary 1:1 Relationship types:

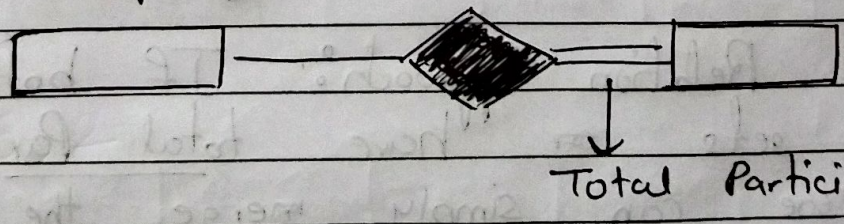
i) Identify the relationships S and T participating in the relationship R.

Now there are 3 possible approaches: the foreign key approach, the merged relation-ship approach, the cross reference or relationship relation approach.

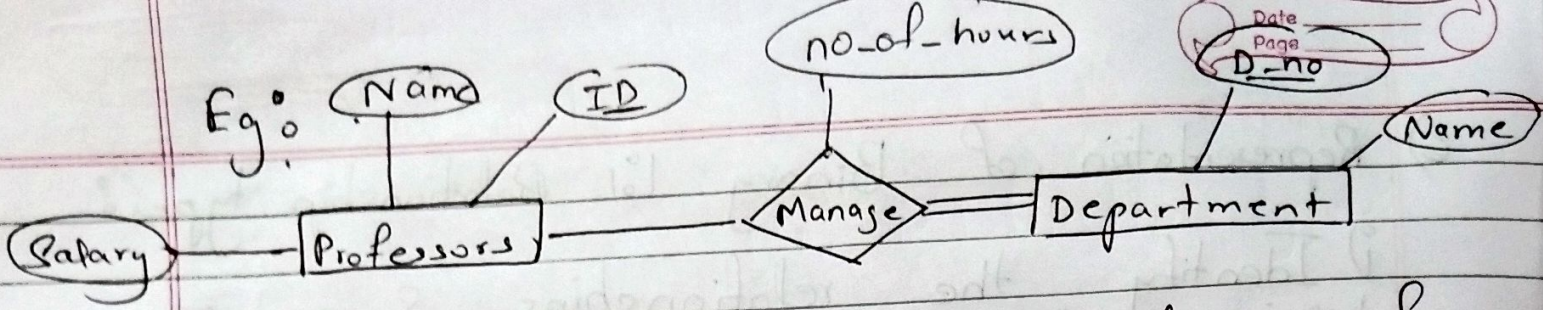
The first approch is the most useful & should be followed unless special conditions exist.

ii) Foreign key approach:

- Choose one of the ~~relationships~~ entity sets and include the primary key of the other ~~relationship~~ entity set as foreign key in the chosen entity set.
- Include all attributes or simple components of composite attributes of relationship R as attributes of chosen entity set.
- It is better to choose the entity set with total participation in the relationship.



Eg:



(No department can exist without a professor but the opposite can't be said).

Professor

ID	Name	Salary

Department

D_no	P_no	Name	no_of_hours

P_no → Foreign key from professor

no_of_hours → Attribute of Relationship 'Manage'

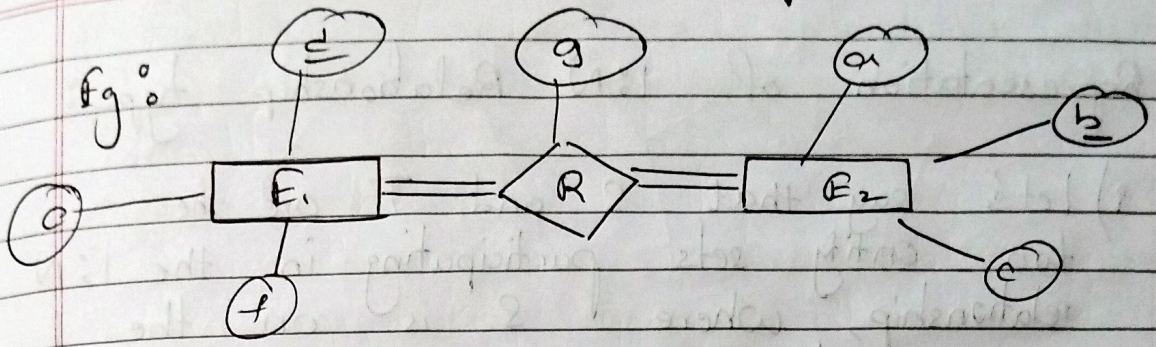
- Note that we could have chosen entity set 'Professor' to have foreign key from department and attributes of ~~Manages~~ 'Manage', but we avoid doing this because then many NULL values can be formed.

If only 5% professors # are there for managing ~~of~~ departments, then the remaining 95% professors will have null values in the column D_no as foreign key.

iii) Merged Relation Approach: If both participating entity sets ~~or~~ have total participation then we can simply merge the two entity sets and relationship to create one relation / table.

When two entity sets have total participation, it's ensured that both entity sets will have same no. of rows / tuples.

Eg:



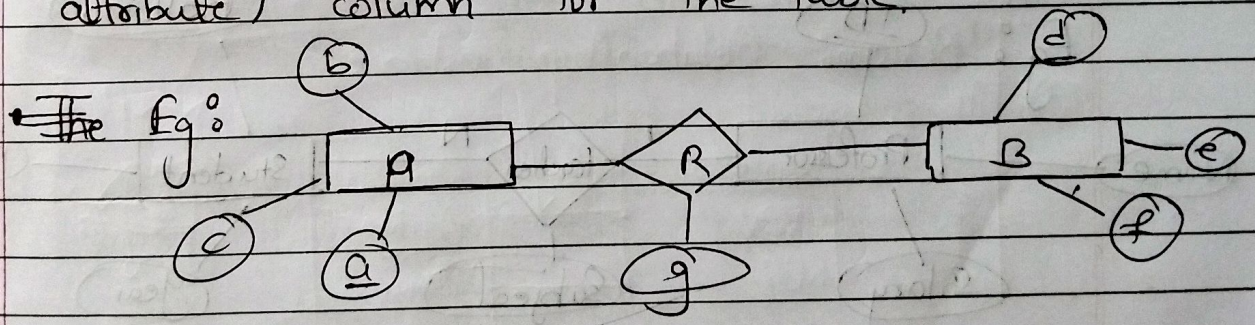
E1RE2

a	b	c	d	e	f	g

iv) Cross-reference or relationship relation approach:

- Third option is to set up a third relation, R, with one of the primary keys of the participating entity sets as foreign keys in this relationship or table R.
- The other primary key will be the normal attribute / column for the table.

Eg:



R2

a	d	g

↓

Foreign key

- The simple attributes or simple components of the composite attributes of the relationship

between the two entities will be the attributes of the new relation

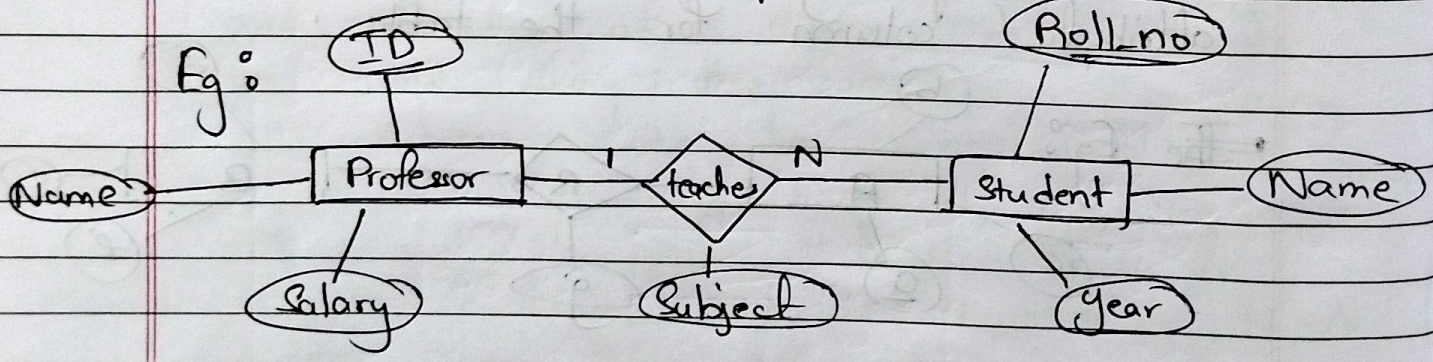
d) Representation of 1:N Relationship type:

i) Let's say that S and T are the two entity sets participating in the 1:N relationship, where S is on the 1 side and T on the N side.

ii) The primary key of S will be included as foreign key in T.
We do this because each entity instance from T will have at most one entity instance from S related. Thus, the primary key can uniquely identify.

iii) Include all simple attributes & simple components of composite attributes from the relationship in T.

Eg:



Professor

ID	Salary	Name

Student

Roll_no	Name	Year	P_ID	Subject

↓
foreign key

iv) We can use cross-referencing method here as well, in the sense that the entity set on the N-side will be giving the new ~~relationship~~ table the primary key & the 1-side will give the foreign key.

We can do this if many tuples/rows from S or T do not have any relationship, with cross referencing method we can avoid many NULL values, as cross-referencing gives us the relations between the two entity sets directly.

tuples of (or rows)

R_{CR}

Roll_no	P_ID	subject

e) Representation of Binary M:N Relationship type:

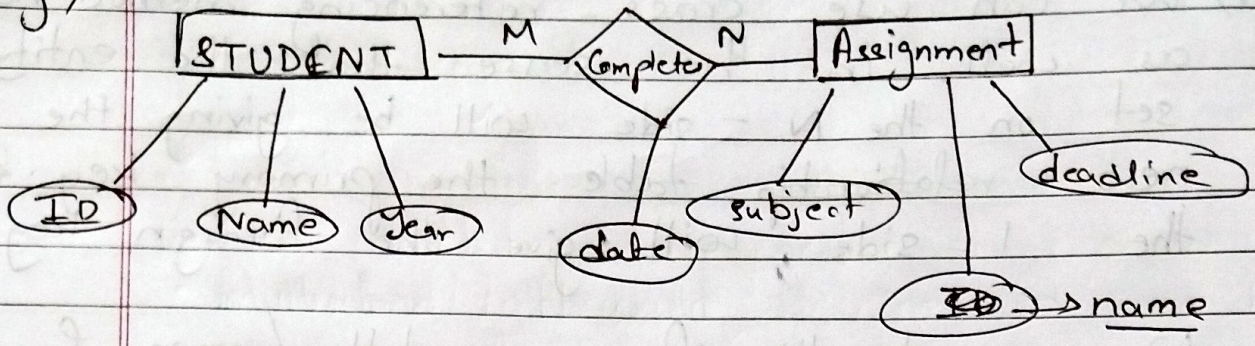
i) We need to use cross-reference method for M:N relationships. relation/table
We create a new ~~relationship~~ R_{CR}.
R_{CR} will have the primary keys of both participating entity sets as the composite keys of ~~RER~~ R_{CR}.

ii) All simple attributes & components of composite attributes ~~will~~ ~~have~~ be added or of the original relationship will be added as simple attributes of R_{CR}

iii) We cannot represent M:N using a single foreign key in one of the participating

entity sets, like in 1:1 and 1:N.

Eg)



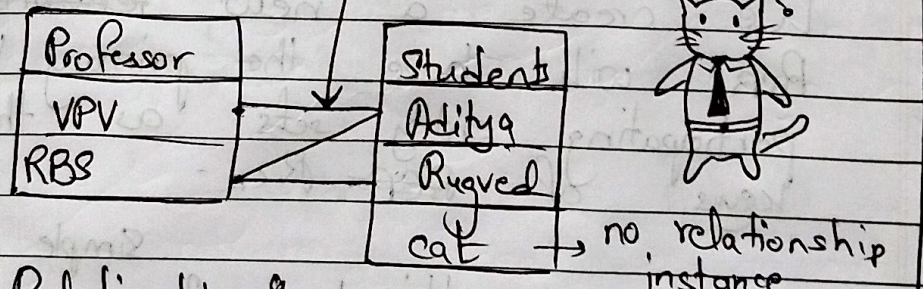
RCR

ID	name	date

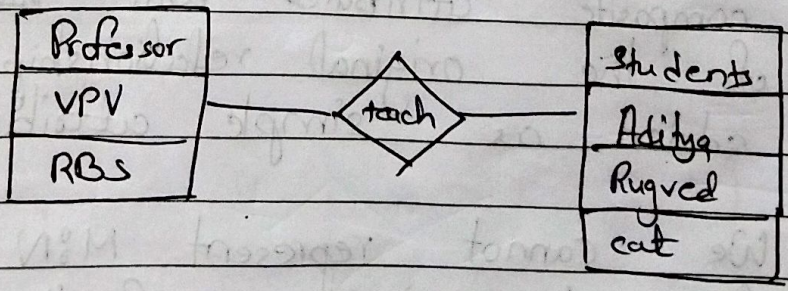
Composite keys.

iii) Note that we can map 1:1 or 1:N like M:N, by using cross-referencing. This alternative is useful when only a few relationship instances exist.

Revision: Relationship instance



Relationship set:



Null values are avoided using cross referencing, as relationship itself has its table, unlike when we use foreign key in one entity set - all tuples/rows will be displayed even if they don't have a relationship instance with any entity instance in the other entity set.

f) Representation of Multivalued attributes:

i) For each multivalued attribute A, a new relation/table ~~can~~ will be created.

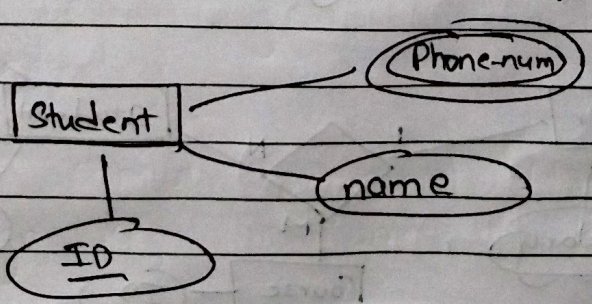
ii) Let's say entity set 'E' has ~~at~~ multiv-alued attribute 'A' & its primary key is 'P':
The new relation/table 'T' will have ~~primary key~~ foreign key 'P'.

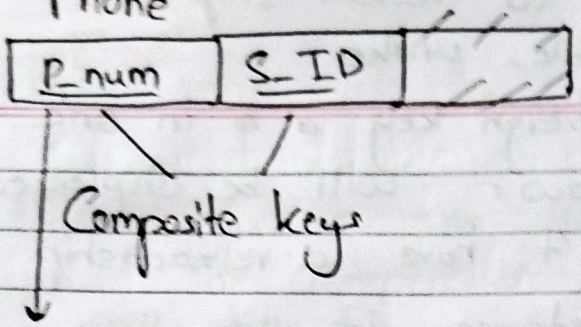
iii) This new relation/table 'T' will also contain an attribute corresponding to 'A'. Meaning, we are basically creating an extra entity set. (IDK how this helps) Let's call this corresponding attribute 'A'.

iv) The primary key of 'T' will be a composition of 'A' and 'P'.

v) If the multivalued attribute is also composite, we include its simple components.

Eg:





New attribute corresponding to previous multivalued attribute 'Phone_num'.

g) Representation of N-ary relationship types: (N>2)

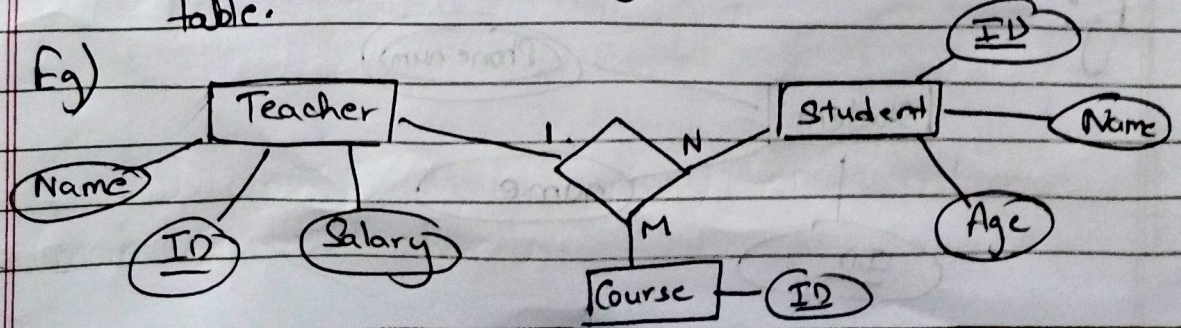
i) We again use cross-referencing method: We create a new relation/table and add the primary keys for all participating entity sets as foreign key for the new relation/table.

ii) We also include the simple attributes or simple components of composite attributes of the original relationship.

iii) The primary key for the new relation/table is a composition of all the foreign keys.

iv) However, if any entity set has 1 as its cardinality constraint, then we do not include its foreign key to be a composite key for the new relation/table.

Eg)



RRc

C_ID	S_ID	T_ID

{All 3 are foreign keys}

↓

Composite Primary keys

**Note: A primary key can be the foreign key as well, & vice versa

3) Mapping EER model to relational model:

a) Mapping Specialization or Generalization:
There are 4 different ways of representing specialization and generalization, depending on the ~~cons~~ the design of EER model:

i) A: Multiple relations / ~~tap~~ tables - Superclass and subclasses.

- Using this mapping technique we will get multiple tables, one for the parent class and ~~one~~ ~~for~~ one each for the subclasses.

(Definitions of terms like class, subclasses, generalization, specialization, etc. can be found in 2.2)

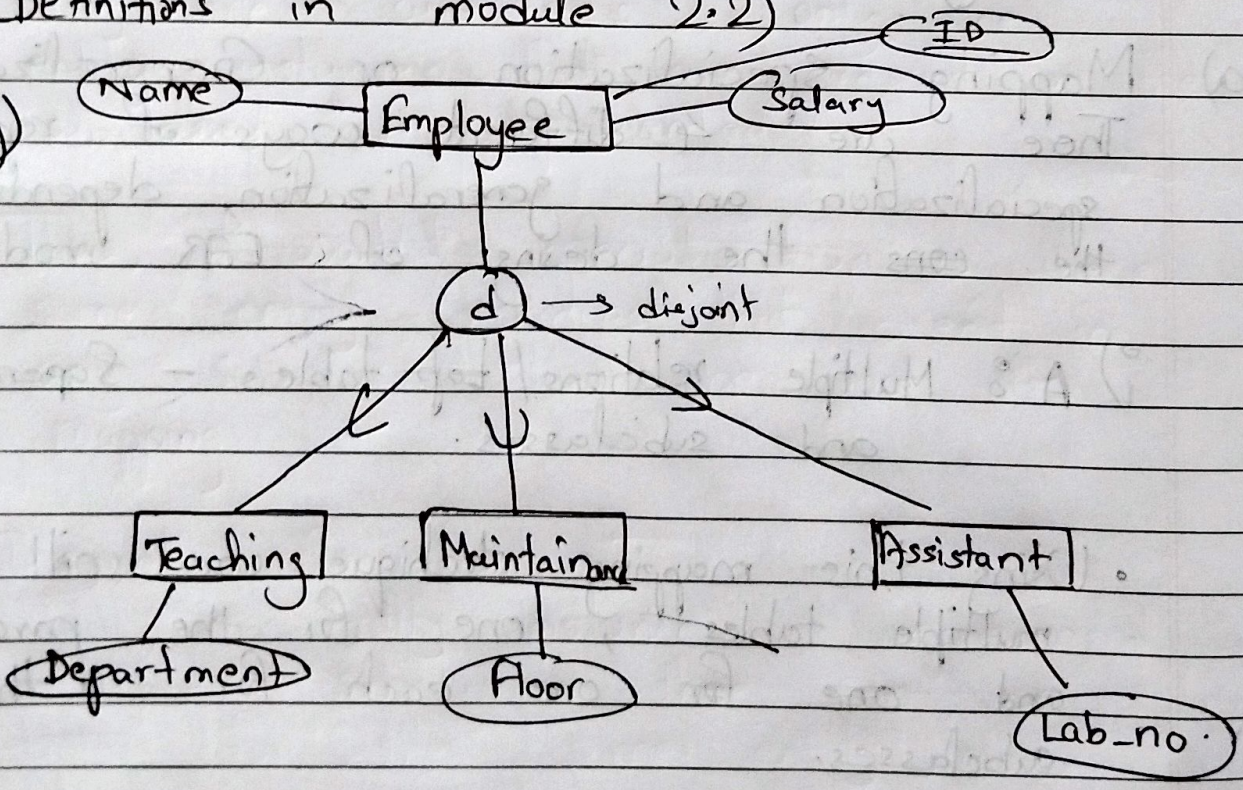
- A relation/table for the parent class/superclass will be created with all the attributes of the superclass and the primary key of the superclass.

- A relation/table will be created for each subclass. ~~This~~ ~~tab~~ Each table for a subclass will have all the attributes of the subclass.
  Each relation/table for each ~~suj~~ subclass will have the primary key same as the primary key of parent class.

- This mapping works for overlapping & disjoint, partial or complete specialization.

(Definitions in module 2.2)

Eg)



(∪ represents union)
(more on that later)

Employee

ID	Salary	Name

Teaching

ID	Department

Maintanance :

ID	Floor

Assistant :

ID	Lab_No

ii) B : Multiple relations/tables - subclass relations only.

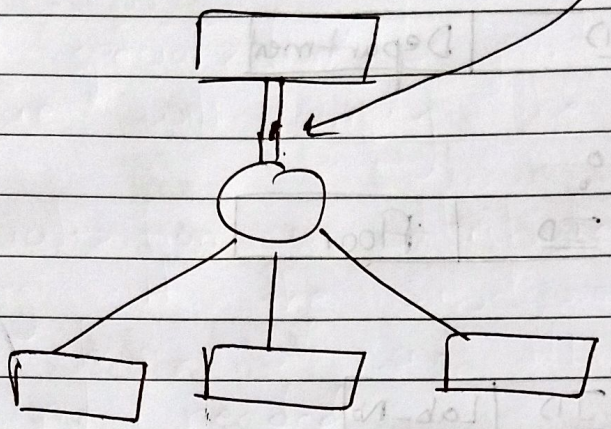
- Multiple Relations / tables are formed but only subclasses' tables are formed.
- Like in 'A', each subclass has its own table and they contain attributes of their corresponding sublasses.

But now, the attributes of the parent class is also included as the attributes for each & every subclass' relation/table.

- The primary key of the parent class is the primary key for each subclass's relation/table.
- This mapping can only work only work for total / complete participation (from the parent

class) and disjoint subclasses.

- Representation of complete / total participation during specialization:



- Reason for the constraints:

Total / Complete Specialization -

Since we are creating the relation for only the subclasses, only the entity instances that are present in the subclasses are represented.

Thus, if partial specialization is there, some entity instances that aren't in any subclass will be lost after mapping.

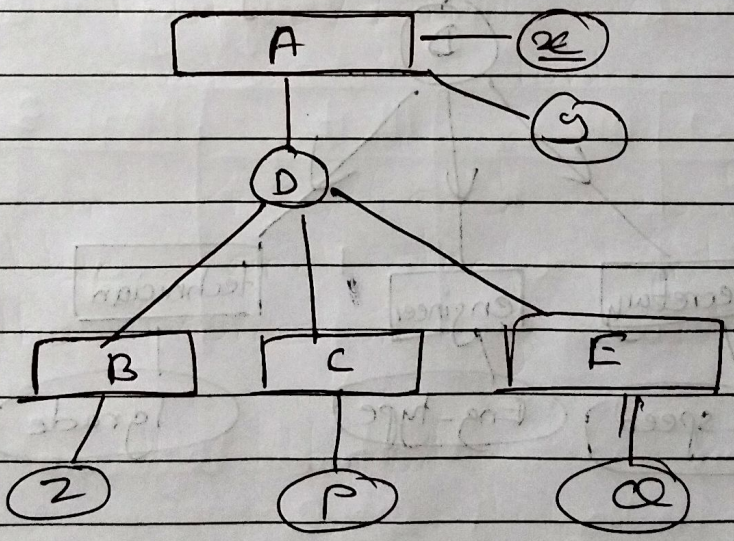
Disjoint Specialization -
If an entity instance is represented in Multiple subclasses, there can be multiple tuples for the same entity instance.

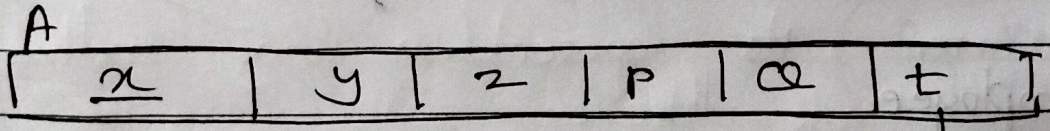
So, we wanna avoid this data redundancy.

iii). C: Single relation with one type attribute.

- Unlike in A and B, there is only 1 relation representing the whole specialization.
- The single relation has the attributes of the parent class & all the subclasses and the primary key is the primary key of parent class, the last attribute in the relation is 't' for 'type / discriminator' attribute.
- Type / Discriminator Attribute is simply an index or ID for each subclass. It can be 1, 2, 3, ... so on.

Eg)



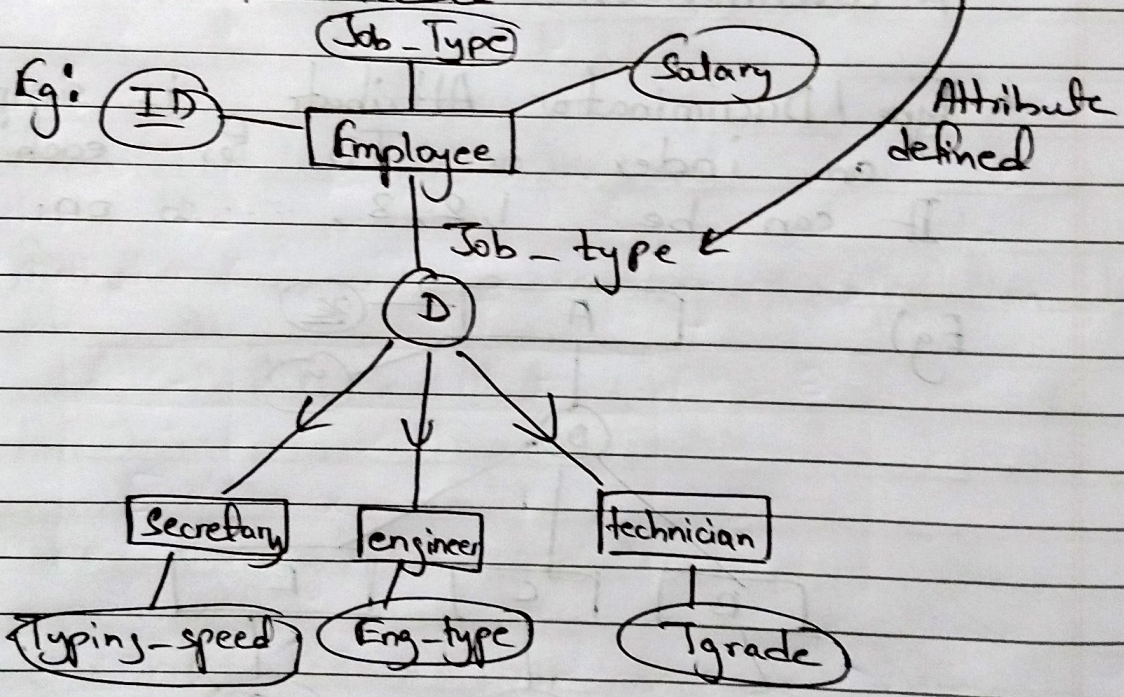


will range from 1 to 3.

- This works only for disjoint subclasses, where the type attribute specifies ~~where~~ which subclass each tuple / entity instance belongs to.

- If the specialization is partial, then the t value will simply be NULL.

- If specialization is attribute-defined, meaning if the subclasses are formed on the basis of an attribute, then t is not needed.



Relation :

Employee

ID	Salary	Jobtype	secretary	Eng-type	Tgrade

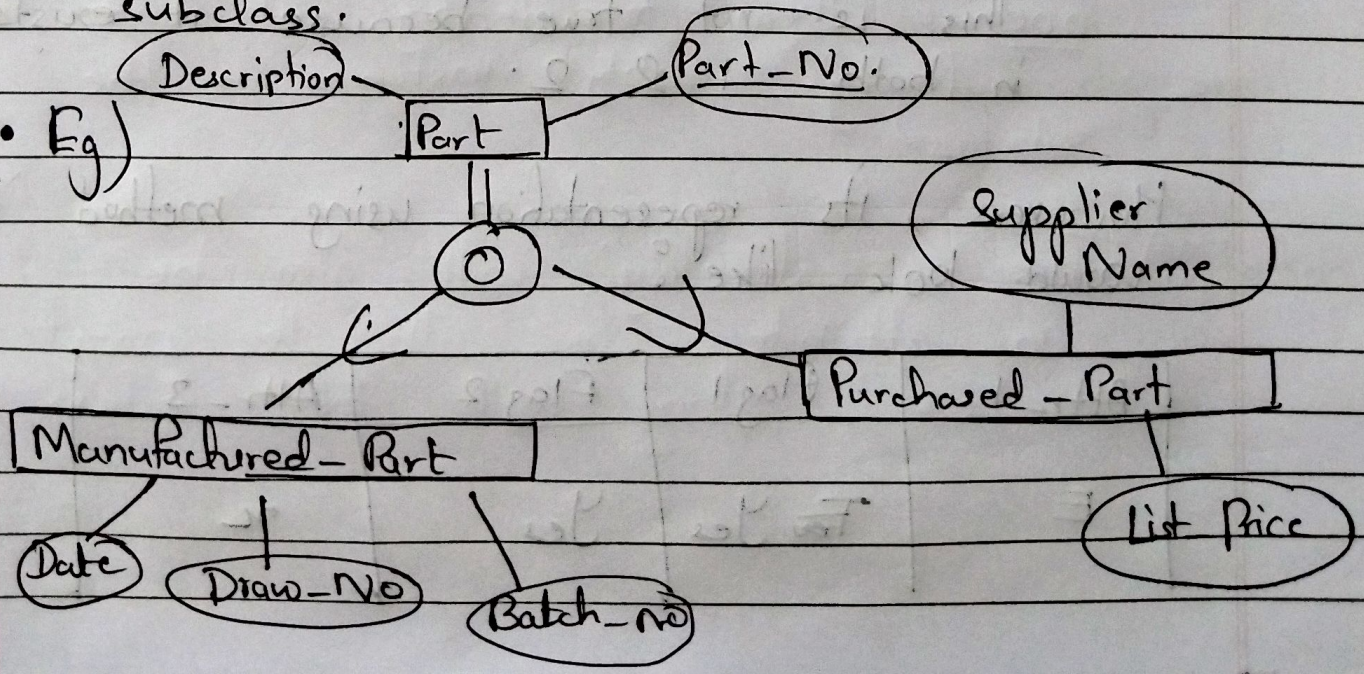
↓

No t attribute needed.

- We avoid using this mapping tchique when there are many subclasses with many attributes.

iv) D: Single relation with multiple type attributes.

- Same as 'C' but there will be multiple type / discriminator attribute.
- We create the relation in the same way as in C but there will be t value attribute for each subclass.
- t value in 'D' can be either 'Yes' or 'No'.
  So each subclass has a t-value attribute which will either be yes or no.
- This mapping works on multi overlapping relations as well because we have multiple attributes for t-value for each subclass.
- Eg)



Relational Model:

PART

Part_No	Description	Mflag	Draw_no	Date	Pflag	name	Price

t value, which can be true or false.

- This t-values is how it is possible to map overlapping subclasses in 'D'

Let's say a tuple / entity instance (E) exists in both superclass '1' & '2', then the mapping for it with method 'C' of mapping will look like:

Attr_1	Attr_2	t	Attr_3
E	x	1	y

This is not true because it exists in both 1 & 2.

However, its representation using method 'D' would look like:

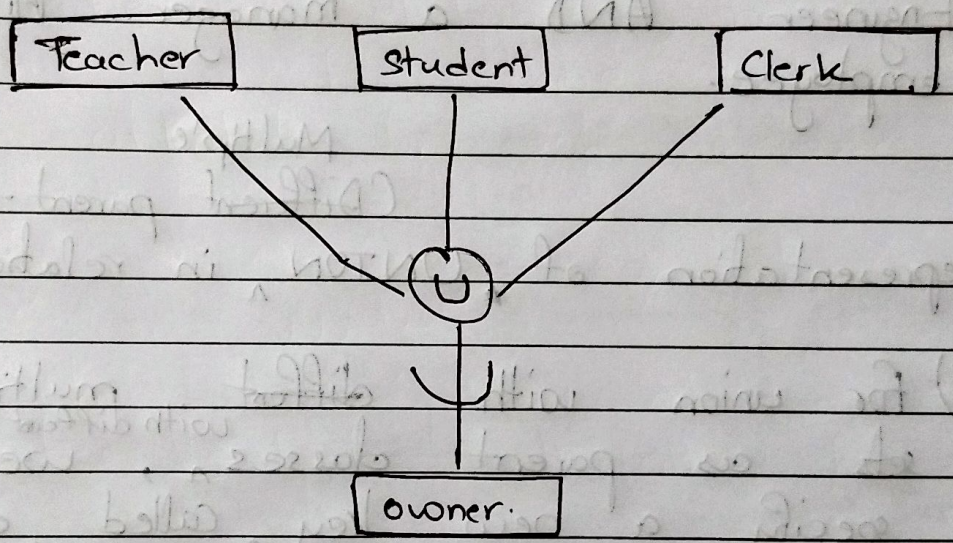
Attr_1	Flag 1	Flag 2	Attr_3
E	Yes	Yes	x

which is correct.

4) Union Types:

a) Union is basically what we mean by specialization.
But there are some special types of unions that aren't covered in specialization. That is, multiple parent classes for one subclass.

b) Eg)

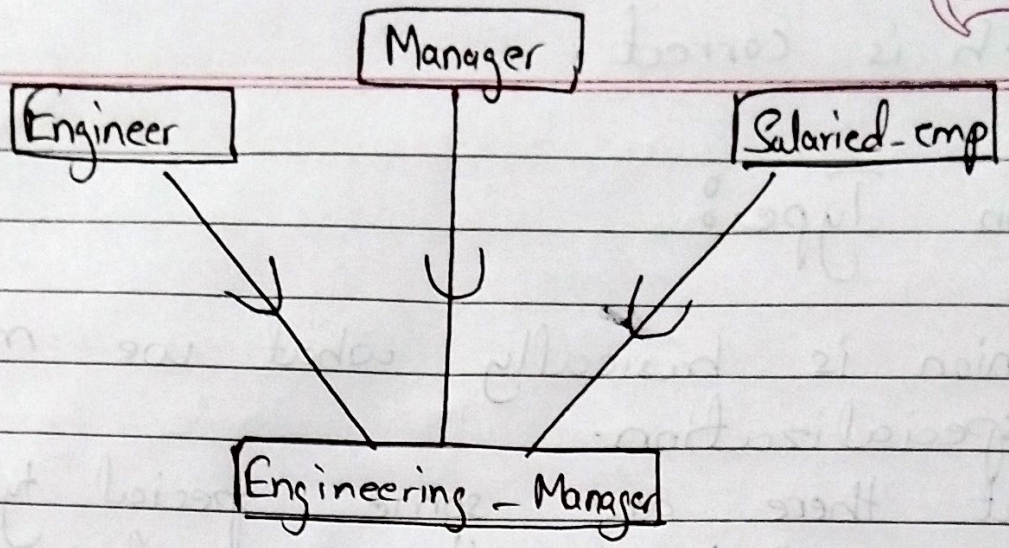


In this example, lets say that we have a pen. This pen can be owned by a teacher OR a student OR a clerk.
This is represented with the EER diagram.

c) There can be another type of multiple parent class one child class representation.

PTO

Eg:



This EER represents that an Engineering_Manager has to be an Engineer AND a manager AND a salaried employee.

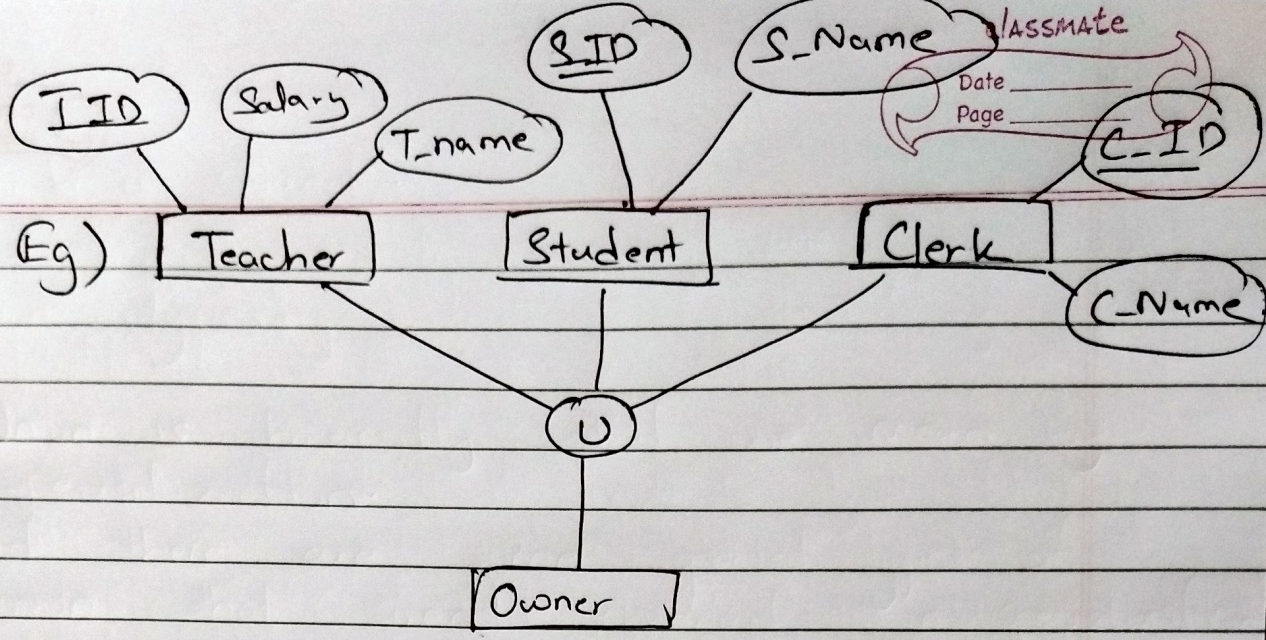
Multiple (Different parent-classes)

d) Representation of 'UNION' in relational Model:

i) For union with ~~diffent~~ multiple entity sets as parent classes with different keys, we need to specify a new key called a 'surrogate key' for relational mapping.

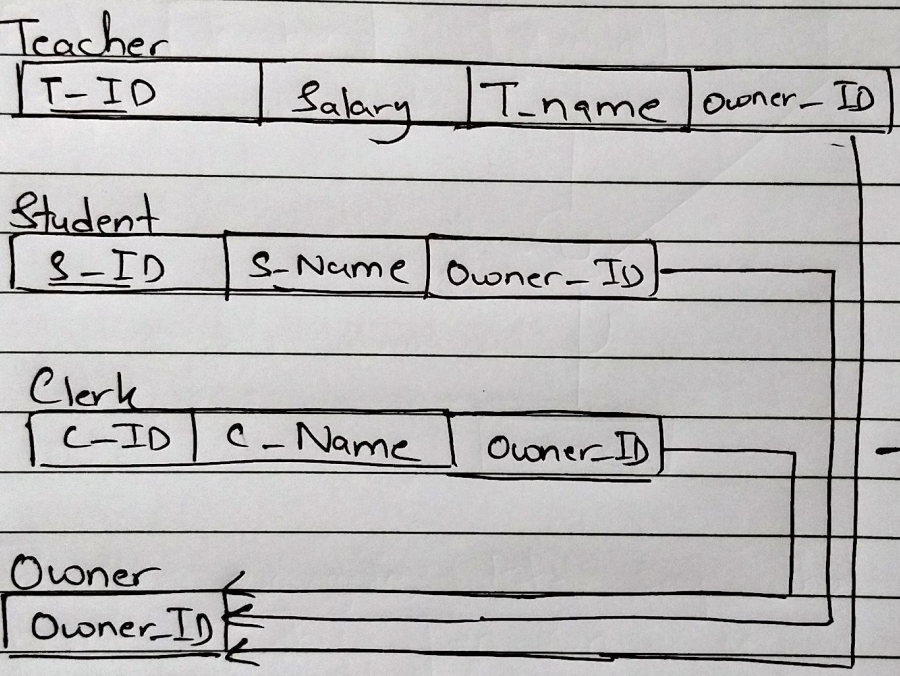
ii) The primary keys of the parent classes are different, so we can't use ~~them~~ any one of them to identify all entities in the ~~category~~. entity instances in the subclass.

iii) This 'surrogate key' will also be an attribute for all the parent classes, along with the subclass



iv) Eg)

Relational Model:



→ Surrogate keys.

We needed to make a surrogate key here because, the primary key of the parent classes were Different.

If the primary key are same for all parent classes, then we would not need to make a surrogate key, & we can simply use that primary key as the key for the subclass as well.