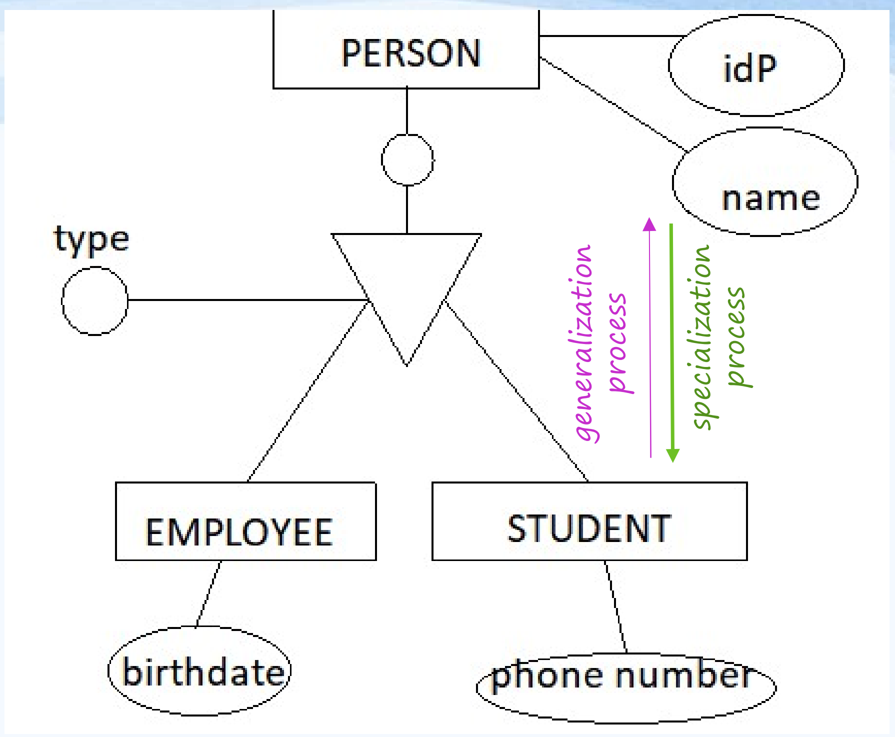
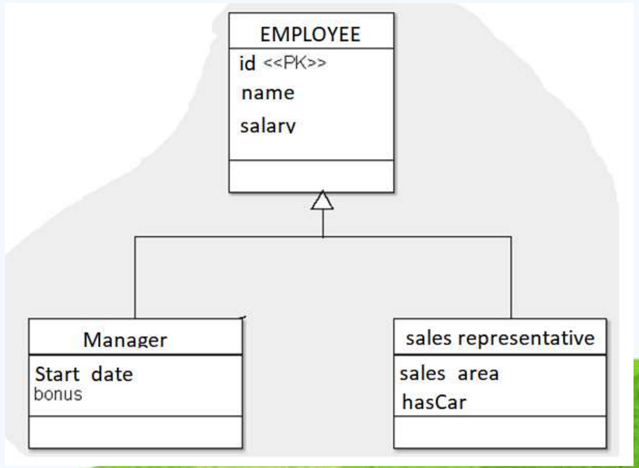
# The Enhanced Entity Relationship model

## Specialization and generalization

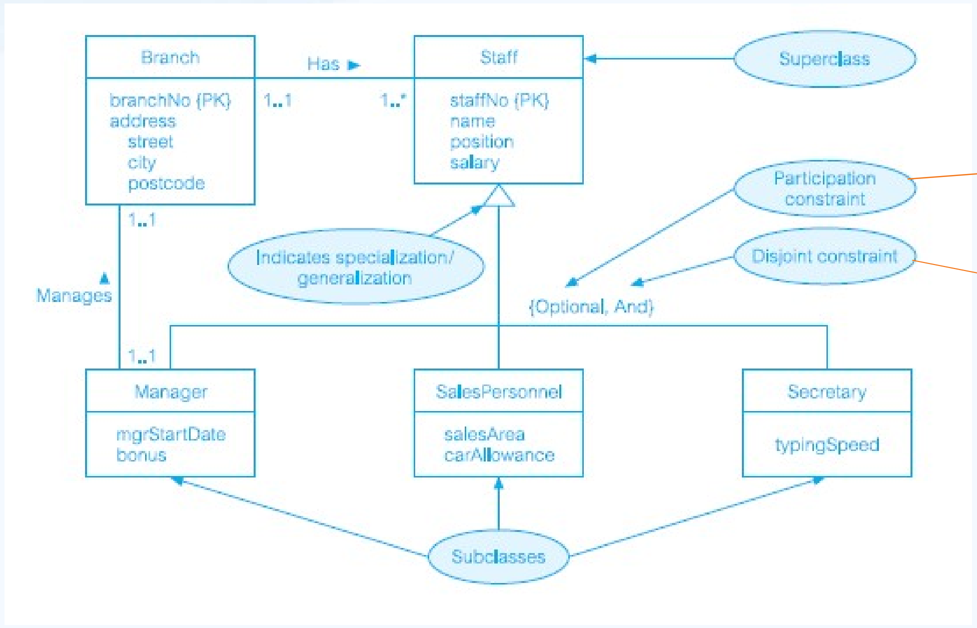
- Subclass and superclass / Subtypes and supertypes
  - Entity Y is a subtype (subclass) of an entity X if and only if every Y is necessarily an X.
  - A subclass entity inherits all attributes and relationships of its superclass entity.
  - A subclass entity may have its own specific attributes and relationships (together with all the attributes and relationships it inherits from the superclass)
  - The relations in which a supertype intervenes also affect the different subtypes -> the subtypes also inherit the attributes associated with these relationships

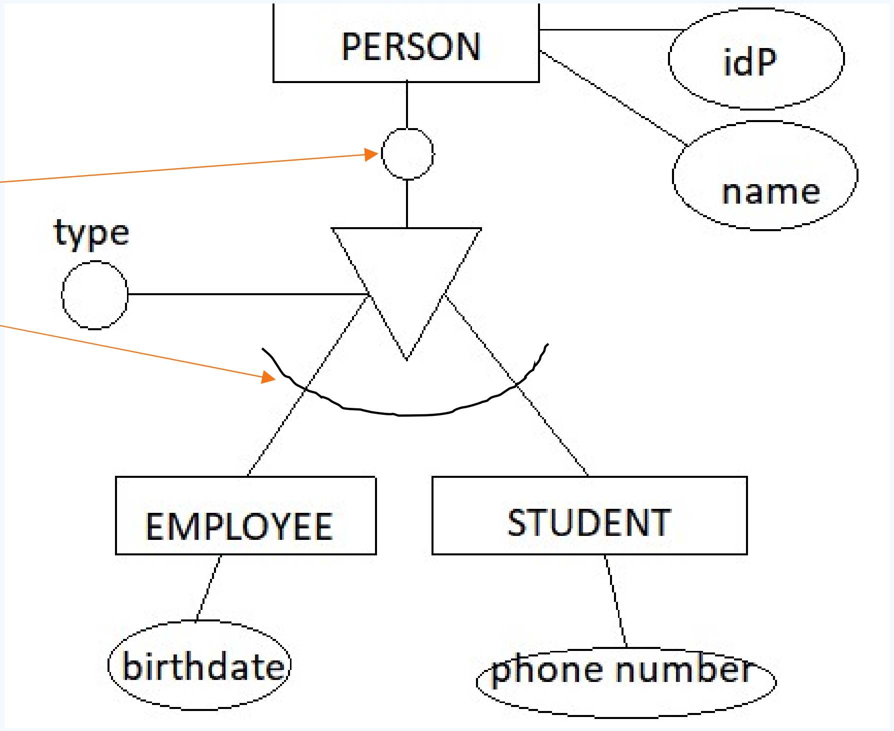
Attribute Inheritance





# Diagrammatic representation of specialization/generalization

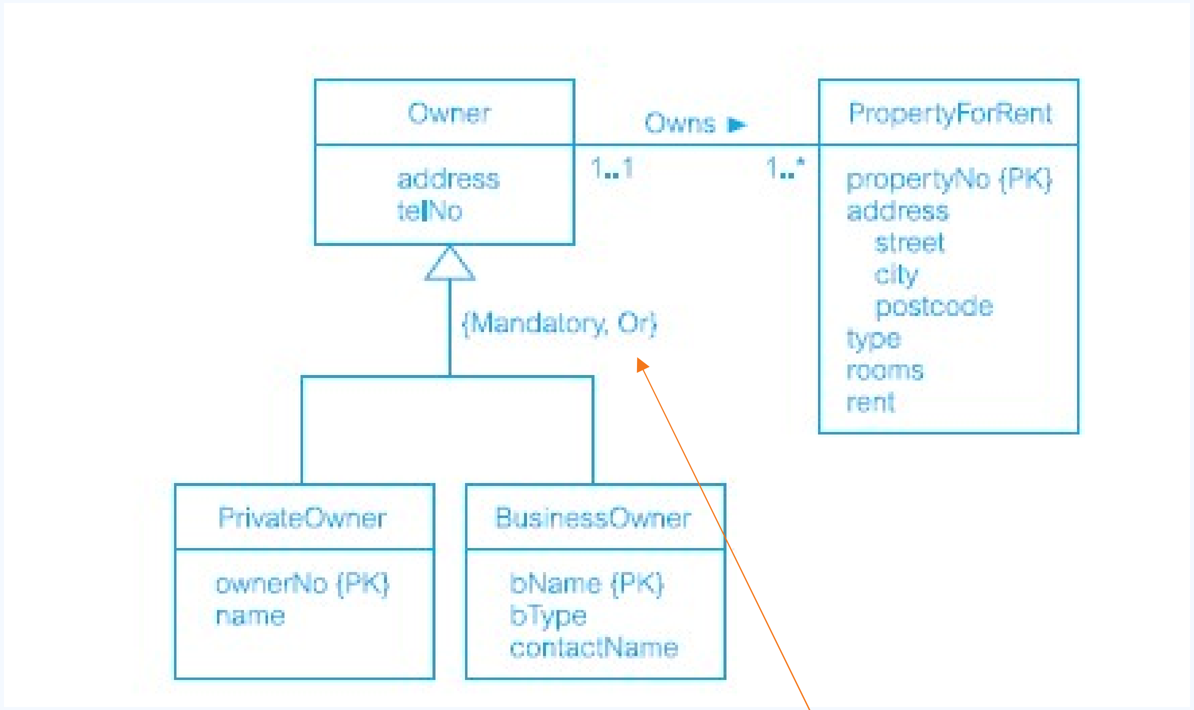




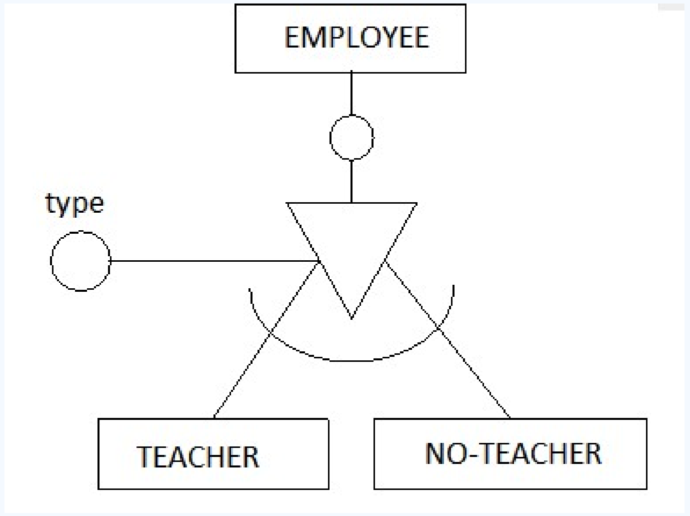
| | |
|---|---|
| **Participation constraint** | Determines whether every member in the superclass must participate as a member of a subclass. |

| | |
|---|---|
| **Disjoint constraint** | Describes the relationship between members of the subclasses and indicates whether it is possible for a member of a superclass to be a member of one, or more than one, subclass. |

## Specialization/generalization mandatory (total) and disjoint

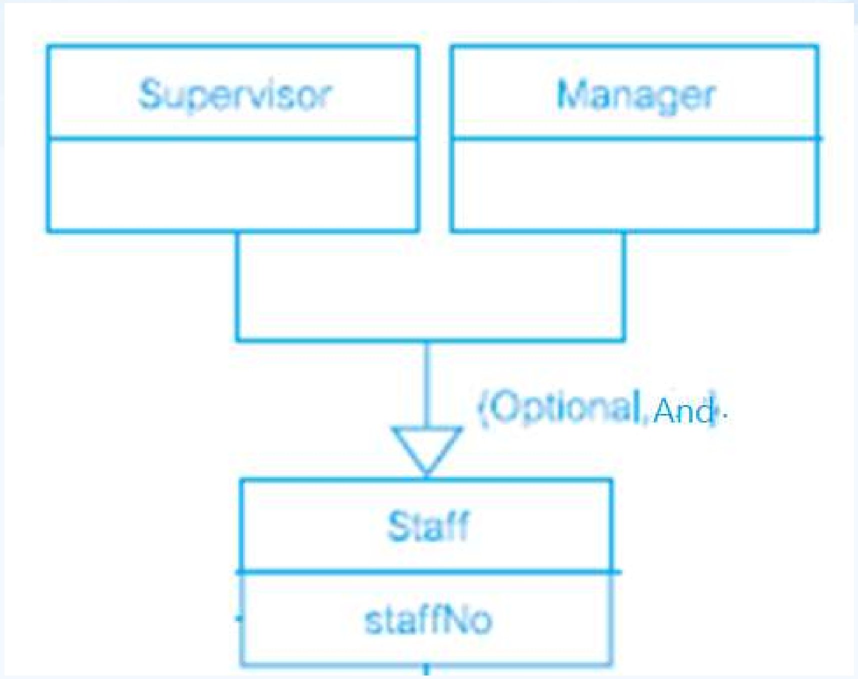


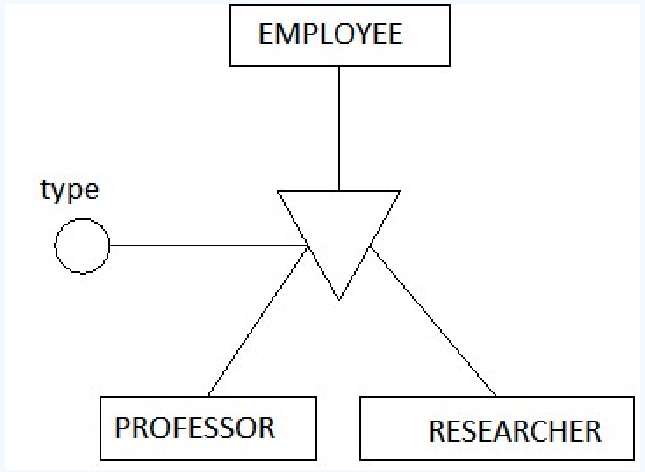
Disjoint



- Both the teacher and the non-teacher are employees
- The same employee can not be both teacher and non-teacher (without overlapping or exclusivity) – disjoint
- Every employee must be a teacher or a non-teacher (totality – mandatory)

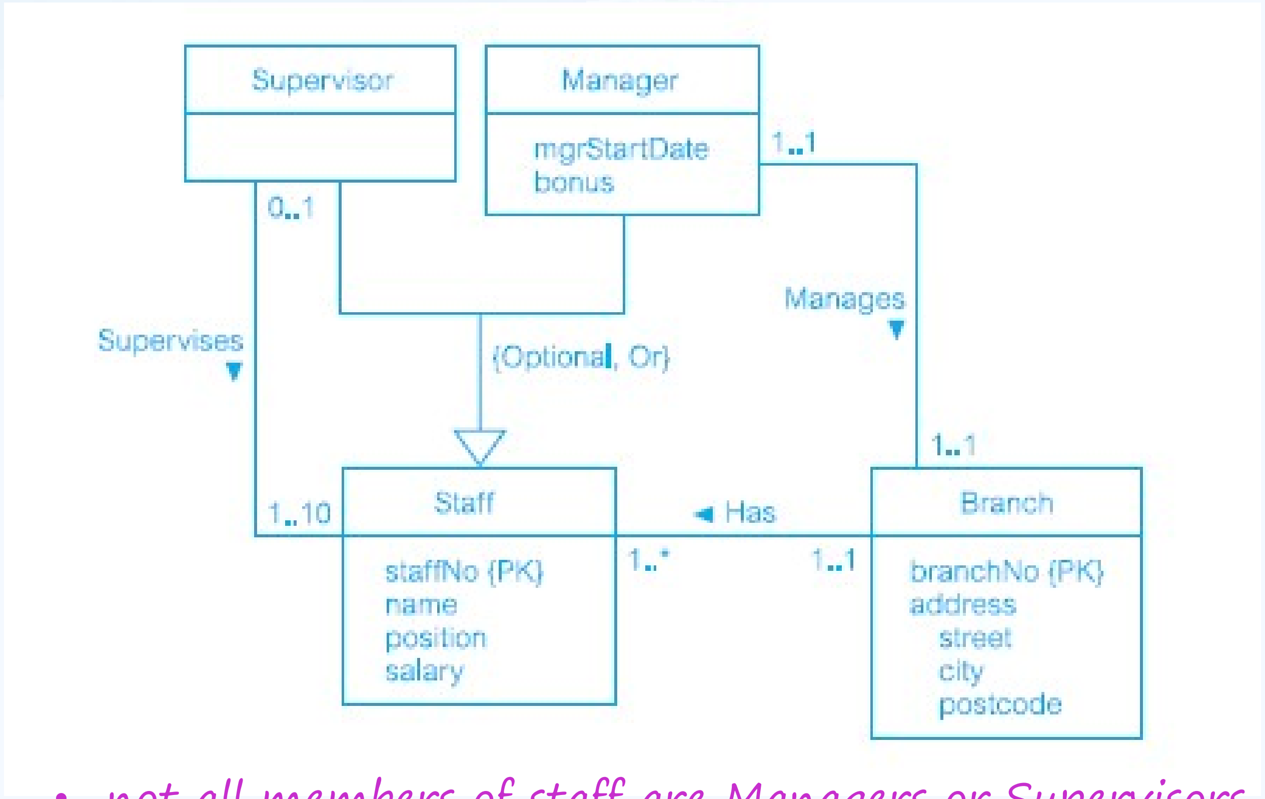
## Specialization/generalization optional (partial) and overlapped

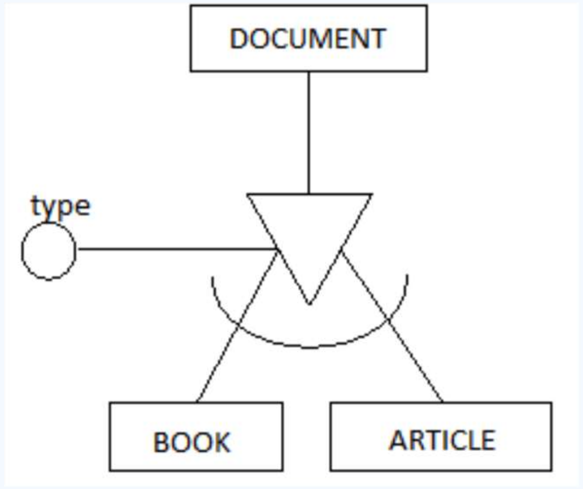




- Both the teacher and the researcher are employees
- The same employee can be both a professor and researcher (overlap)
- An employee may not be a teacher or researcher (partial)

## specialization/generalization optional (partial) and disjoint

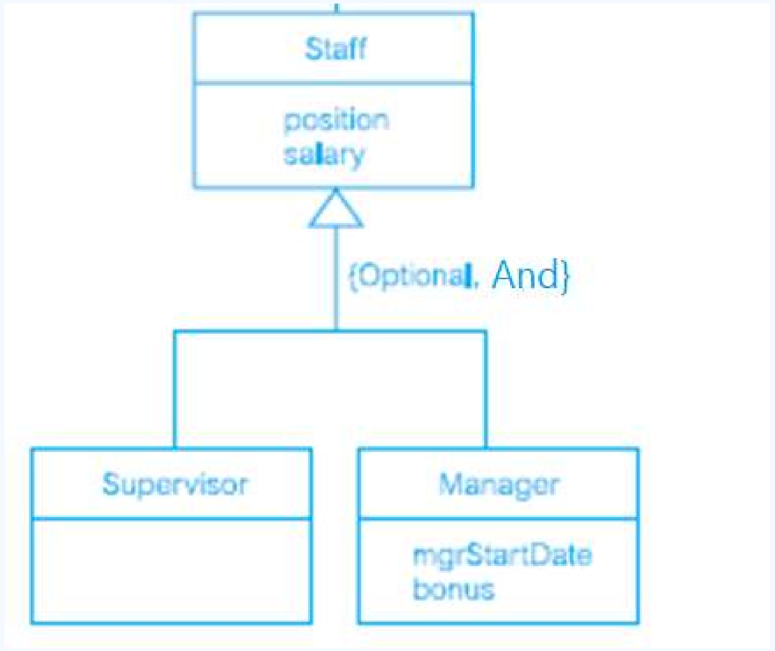


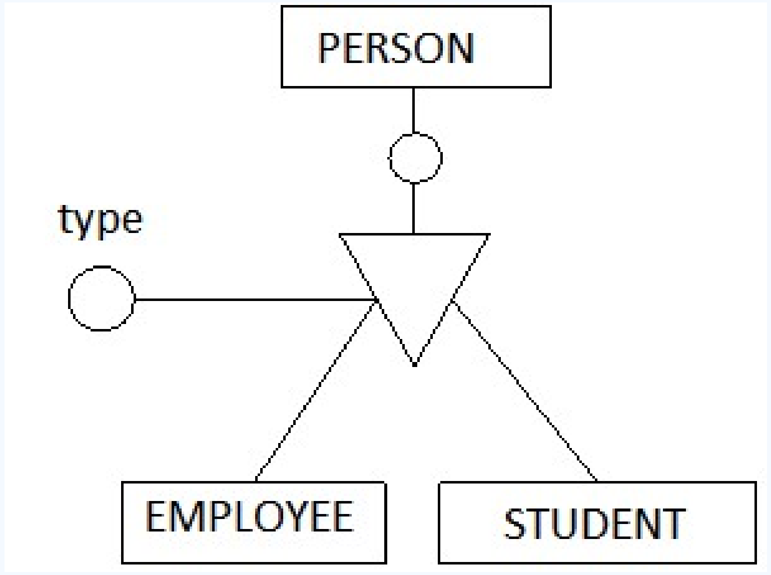


- not all members of staff are Managers or Supervisors, a single member of staff cannot be both a Manager and a Supervisor

- Both the article and the book are documents
- The same document can not be both an article and a book (disjoint)
- There are more documents, not only articles and books (partial)

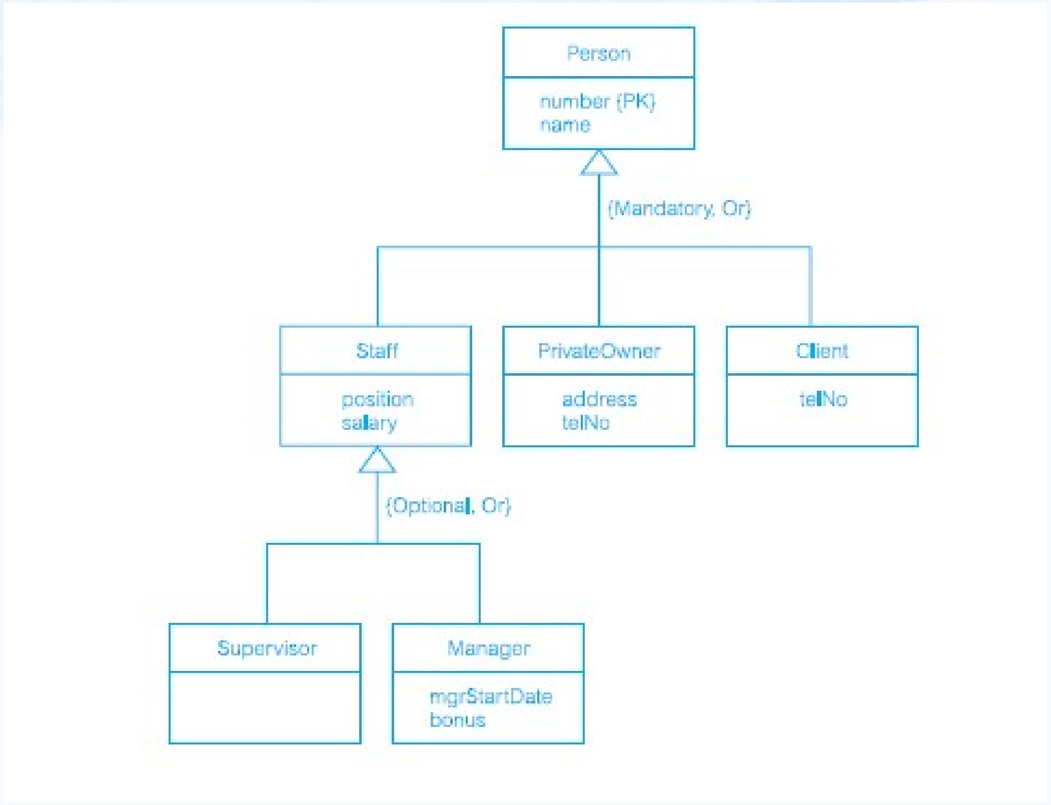
## Specialization/generalization optional (partial) and overlapped

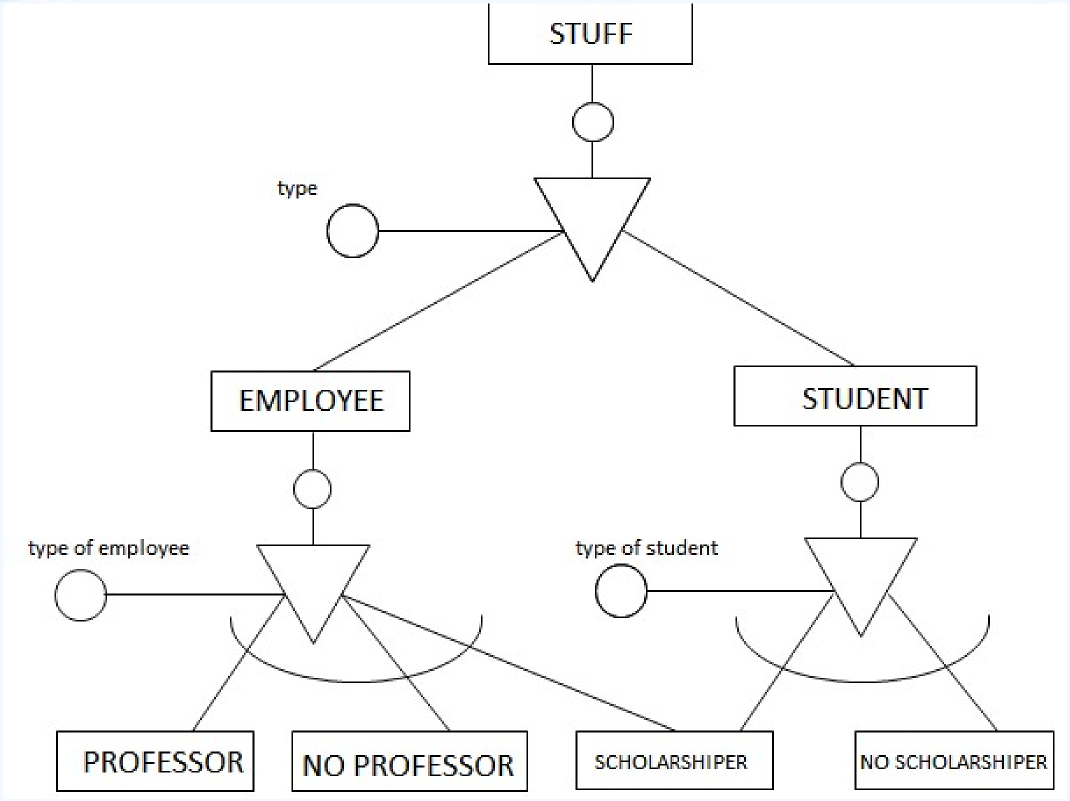




- Both the employee and the student are people
- The same person can be student and employee at the same time (overlap)
- Every person in our database must be a student and / or an employee (complete-TOTAL)

## Specialization Hierarchy



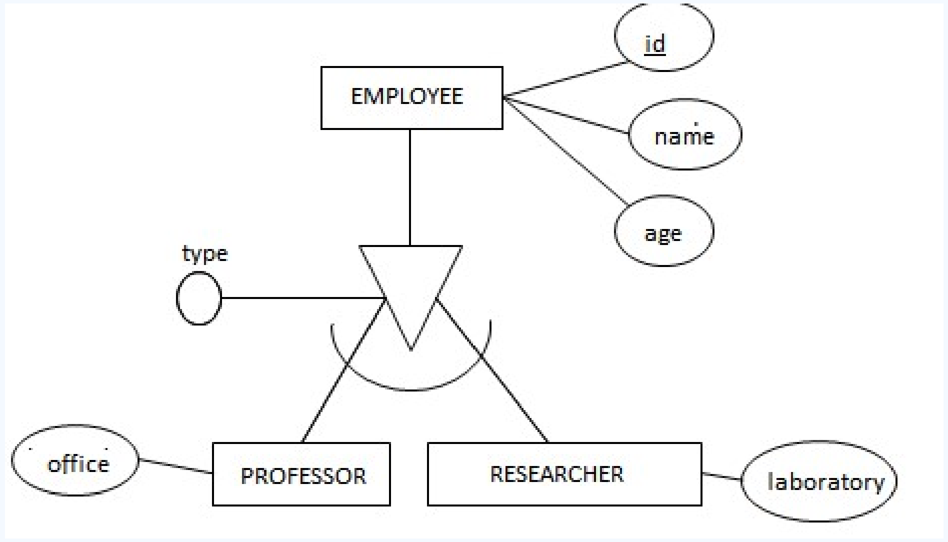


## Derive relations for logical data model

- For each superclass/subclass relationship in the conceptual data model, we identify the superclass entity as the parent entity and the subclass entity as the child entity.
- There are various options on how to represent such a relationship as one or more relations.
  - A. Create only one table (for the superclass and all the subclasses)
  - B. Create many tables (for the superclass and the subclasses)
  - C. Create many tables (but only for the subclasses)

## Create only one table

- Good solution when the subtypes differ in very few attributes and the relationships are the same for all the subtypes.
- To know which subtype a row belongs to, we can add an attribute (called discriminant attribute). Sometimes we can also find out from the information in the table itself



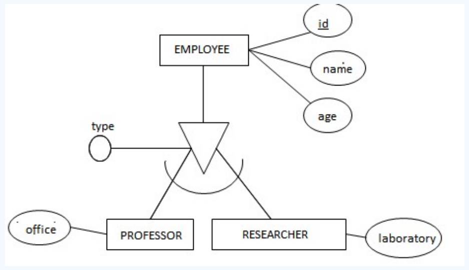
**EMPLOYEE** (id, name, age, …, laboratory, office, **type**)
PK: id
NN: name, age

Defining attribute (discrimintor)

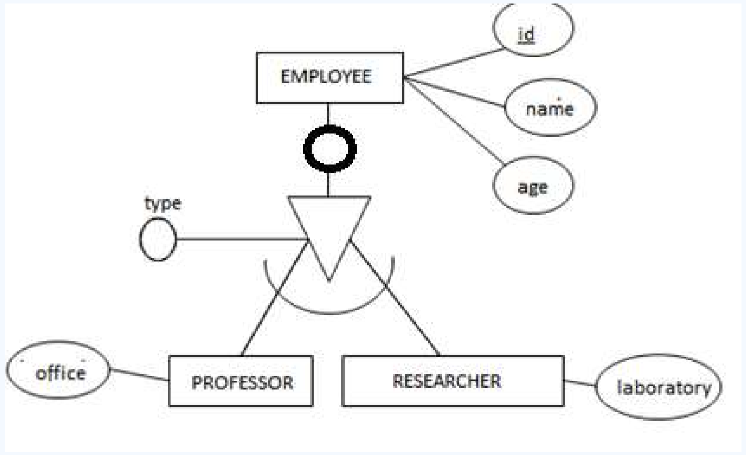
if the attributes "office" and "laboratory" are mandatory in the subtypes, to know if an employee is "professor" we could ask if the attribute "laboratory" has a null value or the attribute "office" has not null value

## Create only one table

- *total specialization -> defining attribute must not admit null values*
- *partial specialization -> defining attribute must admit null values. A null value in the defining attribute would indicate that row does not belong to any subtype*

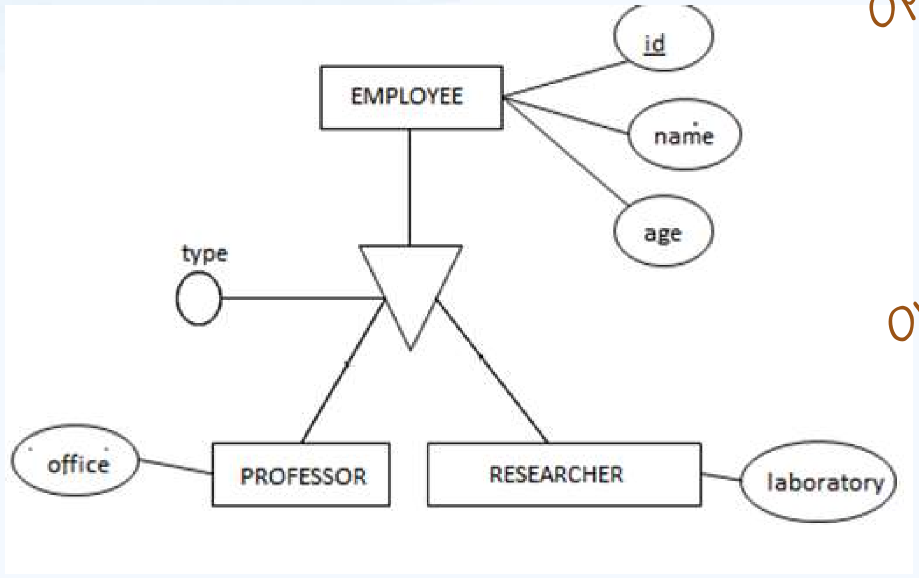


```
CHECK (    (type = 'professor'
            AND laboratory IS NULL
            AND office IS NOT NULL)
       OR  (type = 'researcher'
            AND office IS NULL
            AND laboratory IS NOT NULL)
       OR  (type IS NULL
            AND office IS NULL
            AND laboratory IS NULL) )
```



```
CHECK (    (type = 'professor'
            AND laboratory IS NULL
            AND office IS NOT NULL)
       OR  (type = 'researcher'
            AND office IS NULL
            AND laboratory IS NOT NULL))
```

## An alternative…the case of non disjoint specialization



Option 1

**EMPLOYEE** (id, name, age, …, laboratory, office, **type**)
PK: id
NN: name, age

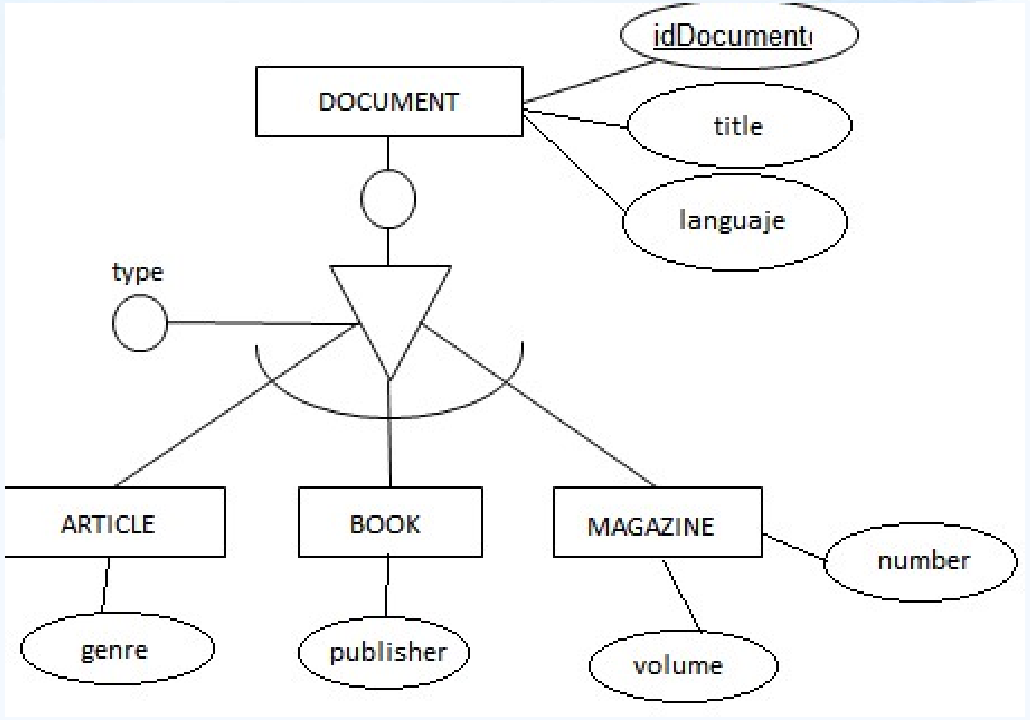
Option 2

**EMPLOYEE** (id, name, age, …, laboratory, office)
PK: id
NN: name, age

**type** (id, type)
PK: (id, type)
FK: id → EMPLOYEE(id)

triggers must be designed to maintain the consistency between the information of both tables

## Create many tables (for the superclass and the subclasses)



**DOCUMENT** (idDocument, títle, languaje, …, type)
PK: idDocument
NN: type

**ARTICLE**(idArticle, genre, …)
PK: idArticle
FKj: idArticle → DOCUMENT(idDocument)

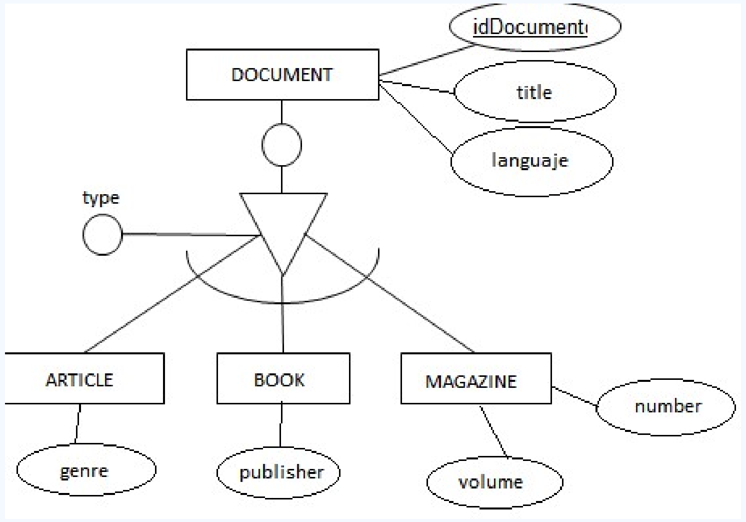
**BOOk** (idBook, publisher, …)
PK: idBook
FK: idBook → DOCUMENT(idDocument)

**MAGAZINE** (idMagazine, volumen, number, …)
PK: idMagazine
FK: idMagazine → DOCUMENT (idDocument)

We will adopt this solution when there are **many different attributes** between the subtypes and, even so, we want to keep the attributes common to all of them in other table.
You can **omit the discriminant** attribute in the supertype table (since it can **produce inconsistencies**).
You have to use triggers to control that there are no overlaps
This option, with the appropriate variants, can be applied in any case: total or partial specializations and with or without overlapping. This solution keeps the meaning of the diagram.

## Create many tables (but only for the subclasses)

- We will create different tables for each subtype that also contain the common attributes
- We will adopt this solution when there are **many different attributes** between the subtypes and accesses that will be made to the different subtypes always affect common attributes.



**ARTICULO** (idArticle, genre, títle, languaje …)
PK: idArticle

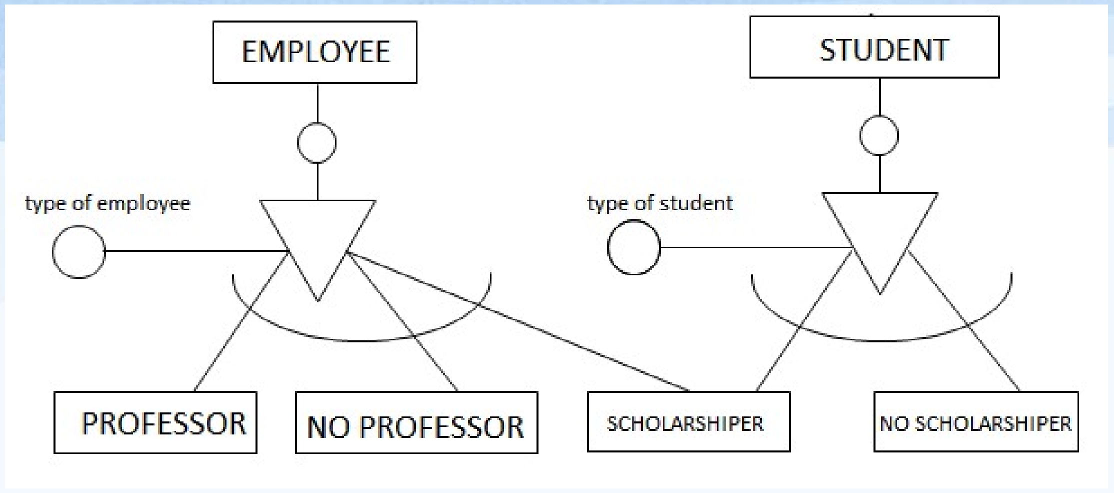
**BOOD** (idBook, publisher, títle, languaje …)
PK: idBook

**REVISTA** (idMagazine, volumen, number, títle, languaje …)
PK: idMagazine

This option is valid for total specializations with or without overlapping. In the case of overlap, a lot of redundancy is introduced and must be controlled if we want to avoid inconsistencies
In case of partial specialization, this solution is incorrect because it would be impossible to store unspecialized supertype occurrences

## Specialization Hierarchy and Lattice

- **Specialization Hierarchy** – has the constraint that every subclass participates as a subclass in only one class/subclass relationship, i.e. that each subclass has only one parent. This results in a tree structure.
- **Specialization Lattice** – has the constraint that a subclass can be a subclass of more than one class/subclass relationship.
- In a lattice or hierarchy, the subclass inherits the attributes not only of the direct superclass, but also all the predecessor super classes all the way to the root.



**EMPLOYEE** (idEmp, department, name, …)
PK: (idEmp, department)

**STUDENT** (idStudent, name, …)
PK: idStudent

Other option

PK: (idEmp, department)
Unique: idStudent
NN: idStudent

**SCHOLARSHIPER** (idEmp, department, idStudent, startDate, …)
PK: idStudent
Únique: (idEmp, department)
NN: (idEmp, department)
FK: idStudent → STUDENT
(idStudent, department) → EMPLOYEE

## To take into account

- When we have a multilevel specialization (or generalization) hierarchy or lattice, we do not have to follow the same mapping option for all the specializations.
- we can use one mapping option for part of the hierarchy or lattice and other options for other parts

## Insertion and Deletion Rules

- The use of specialization involves a set of rules for insertion and deletion

  The insertion of an occurrence in a supertype implies that it must be inserted, automatically, in all the subtypes where the condition is satisfied.

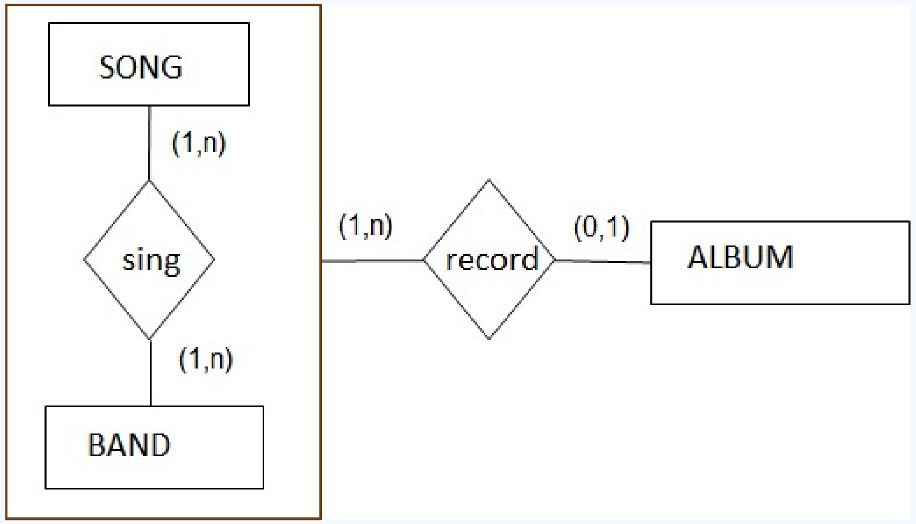
- The insertion of an occurrence in a supertype of a total specialization implies that the occurrence will necessarily be inserted in one of the subtypes of the specialization. If the specialization is without overlapping, it must be inserted only in one of the subtypes

  Deleting an occurrence of a supertype implies that it is automatically removed from the subtypes to which it belongs

  Deleting an occurrence of a subtype involves deleting the corresponding occurrence of the supertype in some cases.

## Aggregation

- Aggregration is a process when the relationship between the two entities is treated as a ***single entity***



**SONG** (idS, titleS)
CP: idS

**BAND** (idB, name, nacionality, ...)
CP: idB

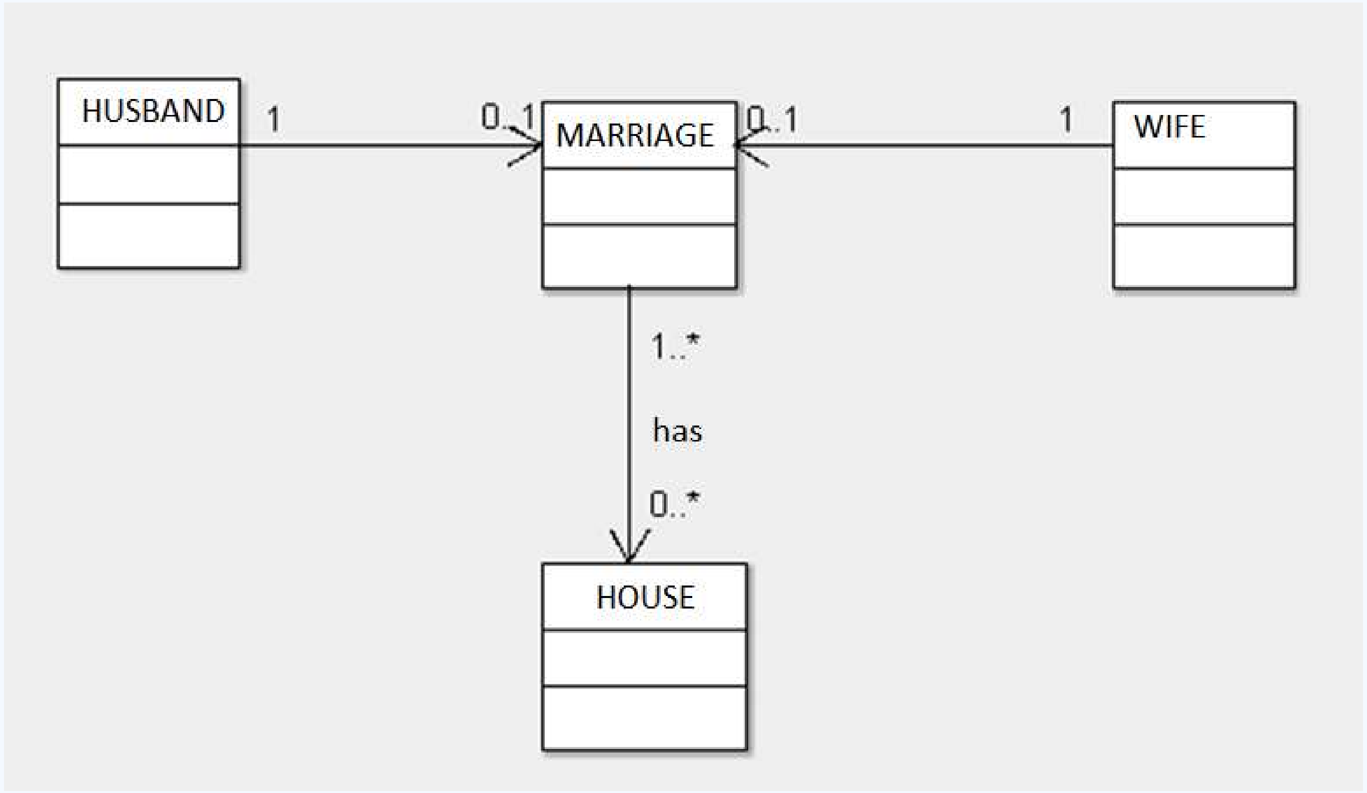
**ALBUM** (idA,titleA,...)
CP: idA

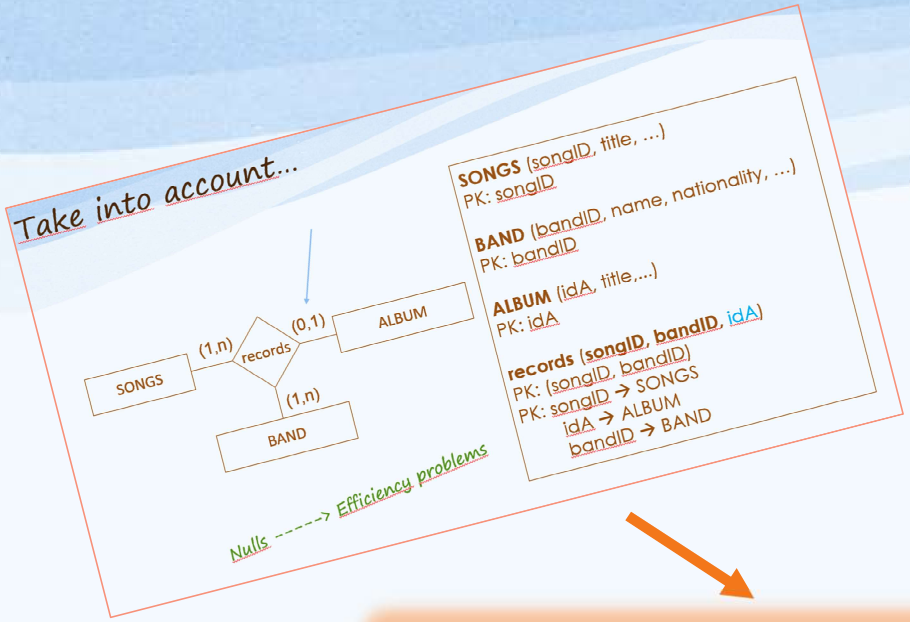
**SING** (idS, idB)
CP: (idS,idB)
CAj: idS → SONG
idB → BAND

**RECORD** (idS, idB, idA)
CP: (idS, idB)
CAj: (idS, idB) → SING
idA → ALBUM
VNN: idA

*In the diagram above, the relationship between **Song** and **Band** together, is acting as an Entity, which is in relationship with another entity **Album***

## Try yourself





## Take into account…

**SONGS** (songID, title, …)
PK: songID

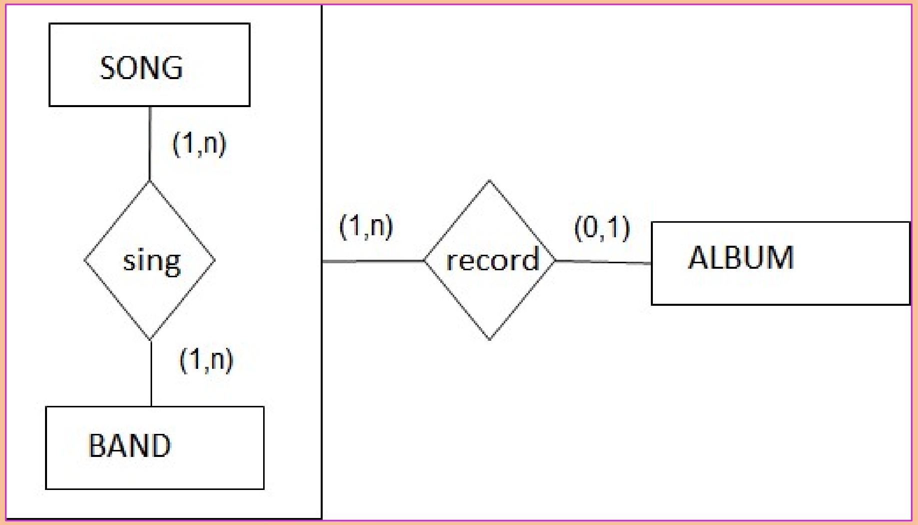
**BAND** (bandID, name, nationality, …)
PK: bandID

**ALBUM** (idA, title,…)
PK: idA

**records** (**songID**, **bandID**, idA)
PK: (songID, bandID)
PK: songID → SONGS
idA → ALBUM
bandID → BAND

Nulls -----> Efficiency problems

*Note that, in this case, there wouldn't be any null values idA column of* ***RECORD****, since the song-band tuples that are not on an album are saved only in* ***sing***



**SONG** (idS, titleS)
CP: idS

**BAND** (idB, name, nacionality, …)
CP: idB

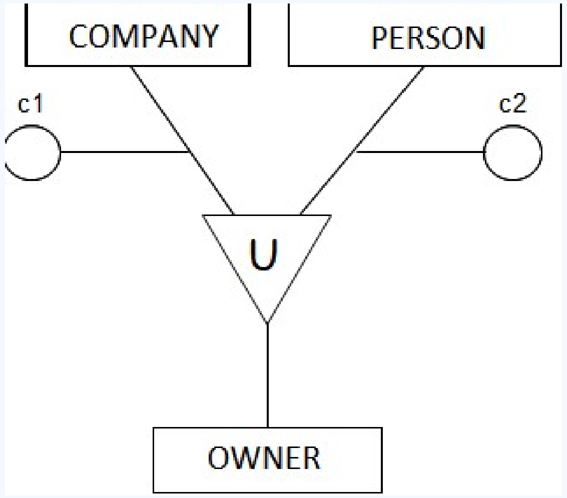
**ALBUM** (idA,titleA,…)
CP: idA

**SING** (idS, idB)
CP: (idS,idB)
CAj: idS → SONG
idB → BAND

**RECORD** (idS, idB, idA)
CP: (idS, idB)
CAj: (idS, idB) → SING
idA → ALBUM
VNN: idA

## Union type (category)

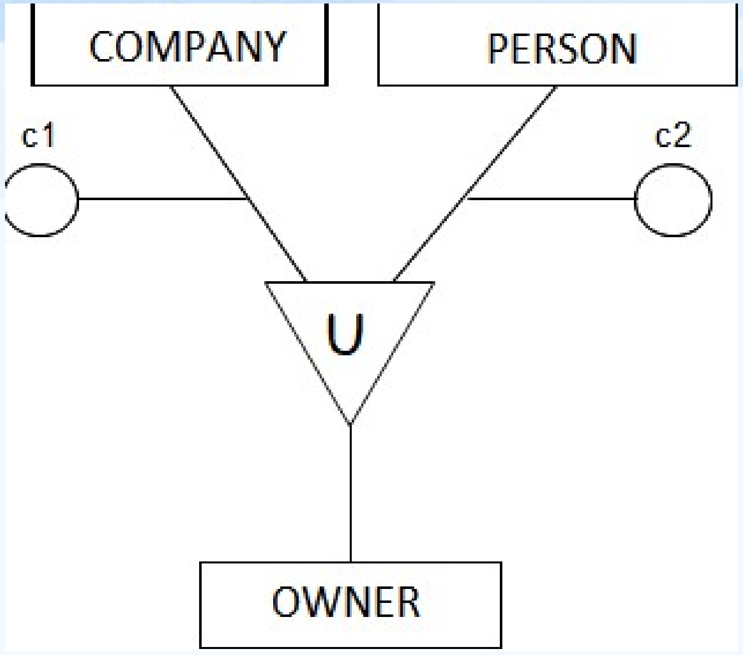
- Represent a single superclass/subclass relationship with *more than one superclass*, where the superclasses represent different entity types.
- In this case, the subclass will represent a collection of objects that is a subset of the UNION of distinct entity types



Example: The entities COMPANY and PERSON, that are conceptually different, can play the role of owners of the current account of a bank

Every OWNER is a company or a person. Companies and individuals may not be account OWNERS (partial)

C1 and C2 are conditions to be an owner



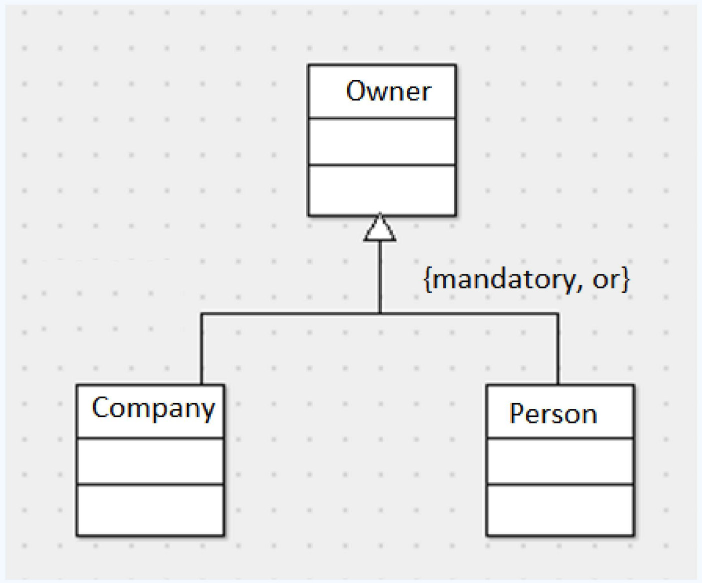
**COMPANY** (idC, nameC, addressC, …, idOwner)
PK: idC
FK: idOwner → OWNER
ÚNIQUE: idOwner
RESTRICCIÓN company_or_persona
  CHECK (( idOwner NOT IN (SELECT idOwner FROM PERSON))
RESTRICCIÓN company
  CHECK ((C1 AND idOwner IS NOT NULL) OR (NOT C1 AND idOwner IS NULL))

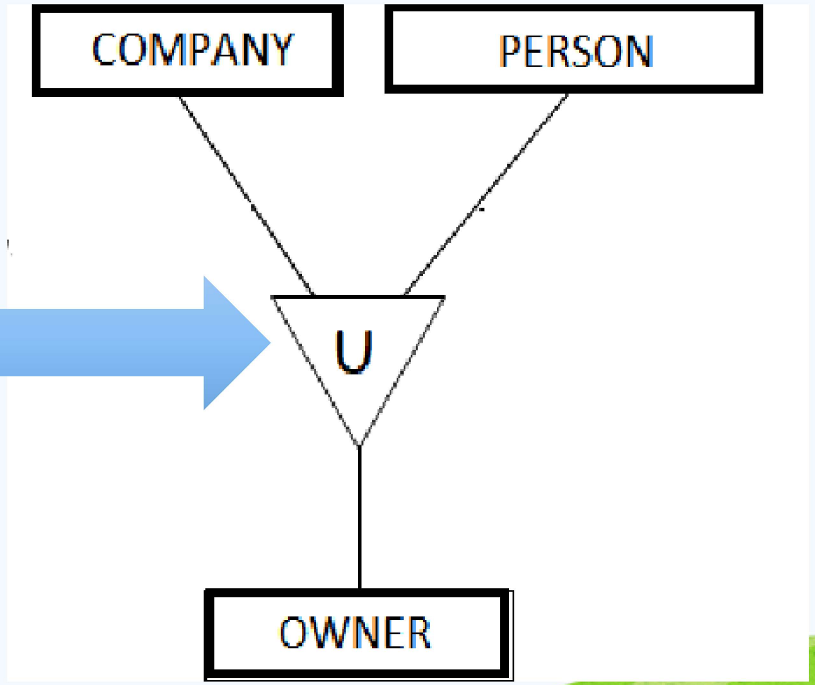
**PERSON** (idP, nameP, …, idOwner)
PK: idP
FK: idOwner → OWNER
ÚNIQUE: idOwner
RESTRICCIÓN company_or_persona
  CHECK (( idOwner NOT IN (SELECT idOwner FROM COMPANY))
RESTRICCIÓN person
  CHECK ((C2 AND idOwner IS NOT NULL) OR (NOT C2 AND idOwner IS NULL))

**OWNER** (idOwner, ...)
CP: idOwner

## Union type in UML

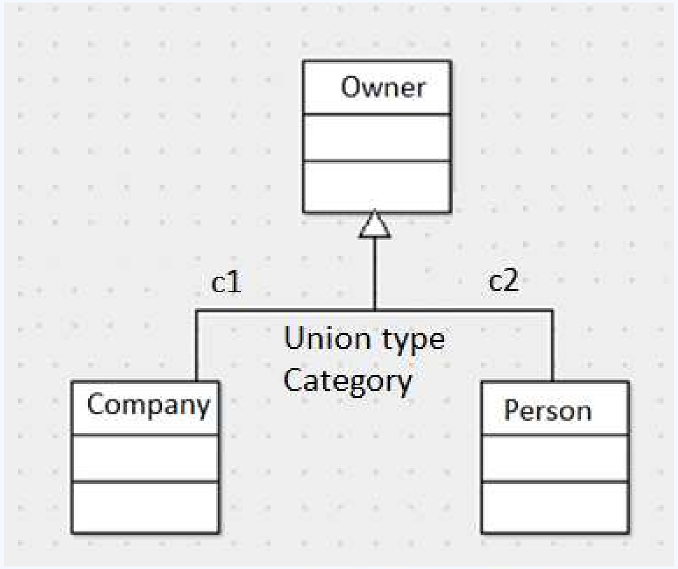
- If the union type is total, it can also be represented with a total specialization without overlapping

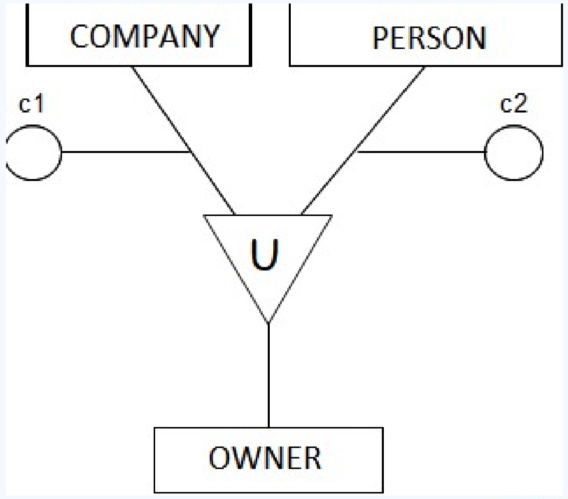




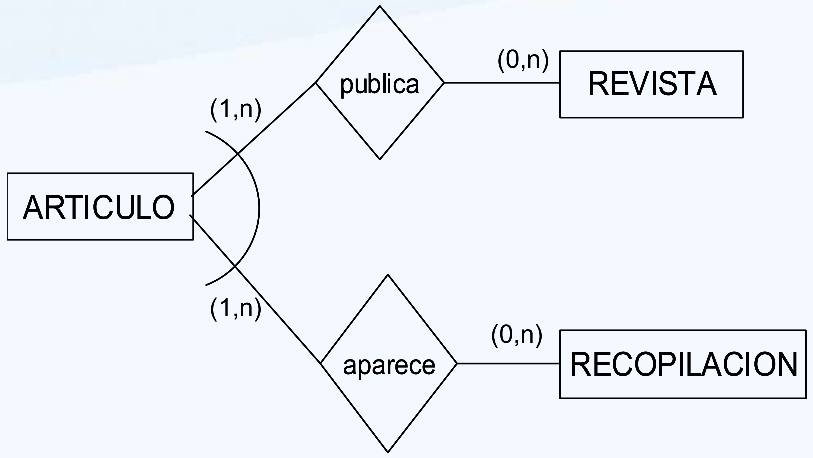
## Union type in UML

- Not having a symbology in UML or an equivalence of equal meaning, we have to use comments and stereotypes to represent categories.
- If we choose this option in UML we do not need to indicate that the specialization is without overlapping since all the categories are disjoint by definition.





# Implementing a Mutually Exclusive Relationship in the Database



Are controlled by a mechanism of the SGBD
**(One table with a check constraint for example)**

**PUBLICA** (idArtículo, idRevista, …)
CP: (idArtículo, idRevista)
CAj: idArtículo → ARTÍCULO
idRevista → REVISTA
RESTRICCIÓN revista_o_recopilación CHECK (( idArtículo NOT IN (SELECT idArtículo FROM APARECE))

**APARECE** (idArtículo, idRecopilación, …)
CP: (idArtículo, idRecopilación)
CAj: idArtículo → ARTÍCULO
idRecopilación → RECOPLIACIÓN
RESTRICCIÓN recopilación_o_revista CHECK (( idArtículo NOT IN (SELECT idArtículo FROM PUBLICA))