



# *Conceptual Database Design*

*Information Systems – Design & Development*

# Conceptual Data Model and Logical Data Model

- Conceptual Data Modeling Elements: Entity-Relationship Model
- The Enhanced Entity Relationship model
- Design Considerations
- Mapping EER model to relations

# Imagine you are involved in this database design

- Name
- Age



We need to know:

- How Many teachers teach on each course?
- What is the maximum number of students there can be on a course?
- Can a student sign up for two courses?



**Our  
Courses**

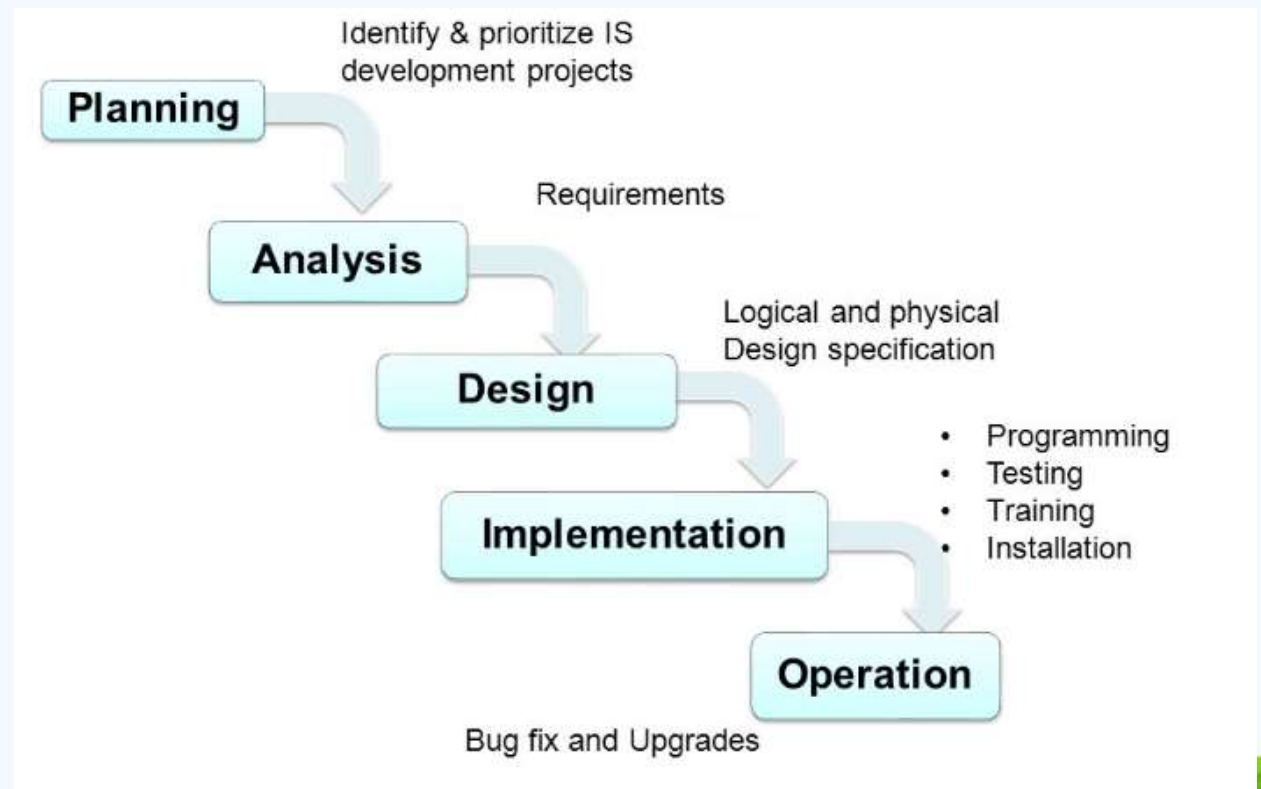
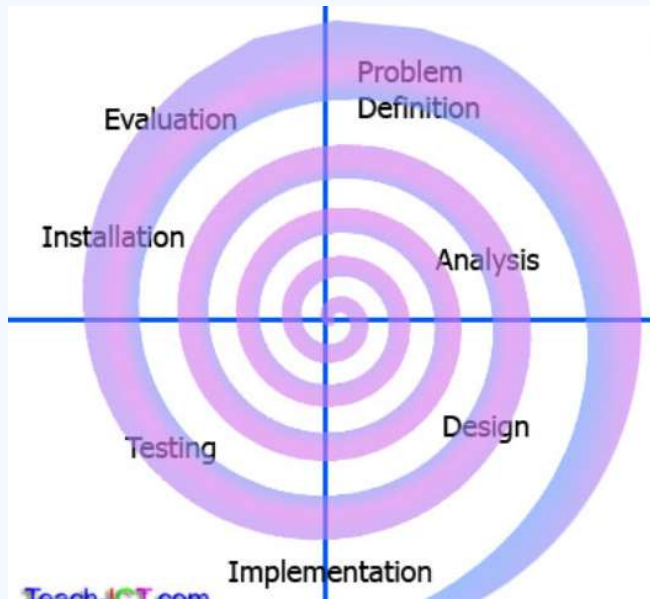
- Name
- When?
- Where?
- How long?

How many students are in each course?  
The list of students of each course  
What teacher teaches each course?



- Name
- Age
- Phone
- Country

# Database life cycle





# Critical Success Factors in Database Design

- *Work interactively with the users as much as possible.*
- *Follow a structured methodology throughout the data modelling process.*
- *Employ a data-driven approach.*
- *Incorporate structural and integrity considerations into the data models.*
- *Combine conceptualization, normalization, and transaction validation techniques into the data modelling methodology.*
- *Use diagrams to represent as much of the data models as possible.*
- *Use a Database Design Language (DBDL) to represent additional data semantics that cannot easily be represented in a diagram.*
- *Build a data dictionary to supplement the data model diagrams and the DBDL.*
- *Be willing to repeat steps.*

*These factors are built into the methodology we present for database design.*

# Why Conceptual Modelling is important?

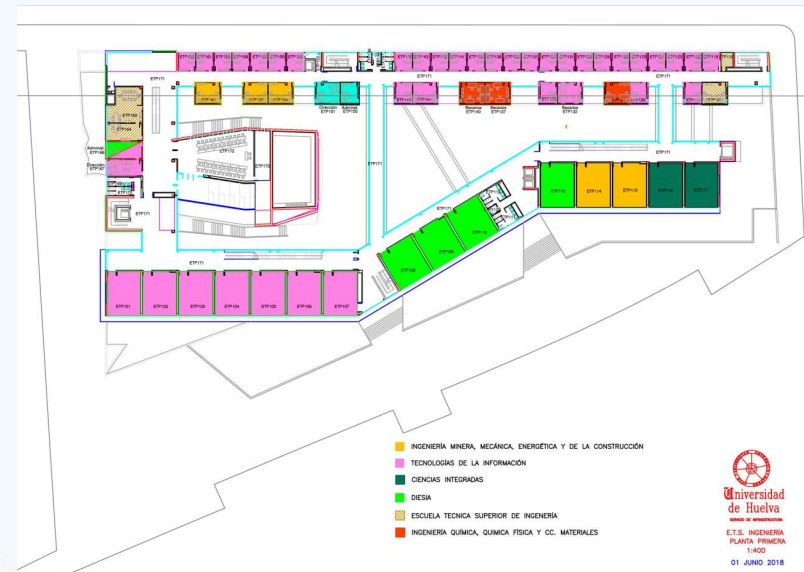
- What is going to be stored?
- How is it going to be used?
- What are we going to do with the data?
- Who should access the data?

A *conceptual model* is a representation of a system that uses concepts and ideas to form said representation.

*Conceptual modelling* is used across many fields.

## What is Conceptual Design?

- A high-level description of the database
- Sufficiently precise that technical people can understand it
- But, not so precise that non-technical people can't participate



# Design Methodology

## Step 1. Build conceptual data model

Step 1.1 Identify entity types

Step 1.2 Identify relationship types

Step 1.3 Identify and associate attributes with entity or relationship types

Step 1.4 Determine attribute domains

Step 1.5 Determine candidate, primary, and alternate key attributes

Step 1.6 Consider use of enhanced modeling concepts (optional step)

Step 1.7 Check model for redundancy

Step 1.8 Validate conceptual model against user transactions

Step 1.9 Review conceptual data model with user

## Step 2 Build and validate logical data model

Step 2.1 Derive relations for logical data model

Step 2.2 Validate relations using normalization

Step 2.3 Validate relations against user transactions

Step 2.4 Check integrity constraints

Step 2.5 Review logical data model with user

Step 2.6 Merge logical data models into global model (optional step)

Step 2.7 Check for future growth



# Design Methodology

## Step 3 Translate logical data model for target DBMS

- Step 3.1 Design base relations
- Step 3.2 Design representation of derived data
- Step 3.3 Design general constraints

## Step 4 Design file organizations and indexes

- Step 4.1 Analyze transactions
- Step 4.2 Choose file organizations
- Step 4.3 Choose indexes
- Step 4.4 Estimate disk space requirements

## Step 5 Design user views

## Step 6 Design security mechanisms

## Step 7 Consider the introduction of controlled redundancy

## Step 8 Monitor and tune the operational system

# Building our Conceptual Data Model

- *Entity-Relationship (ER) model, proposed by Peter Chen in 19976, is a popular high-level conceptual data model.*
- *Object modeling methodologies such as the Unified Modeling Language (UML) are becoming increasingly popular in both database and software design.*



# Building our Conceptual Data Model

- *A conceptual model consists of*

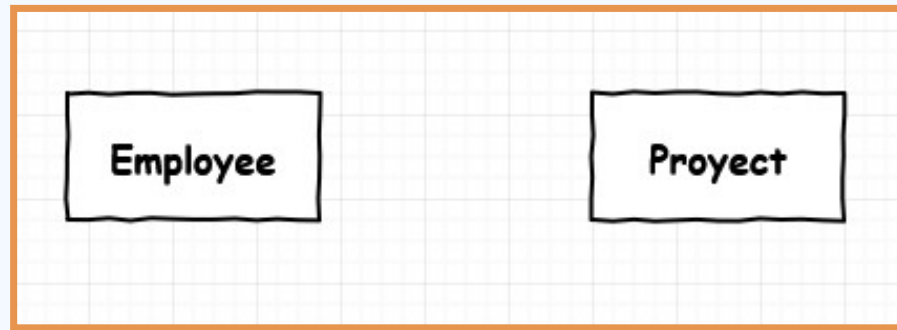
- *entity sets;*
- *relationship sets;*
- *attributes and attribute domains;*
- *primary keys and alternate keys;*
- *integrity constraints.*



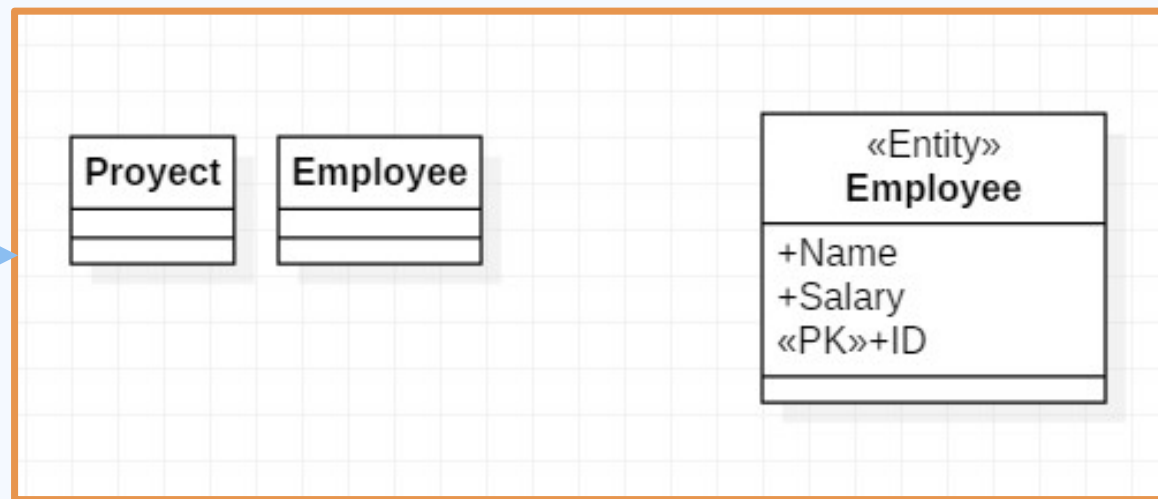
# Entities and Entity Set

- *Entities & entity sets are the primitive unit of the E/R model*
  - *Entities are the individual objects, which are members of entity sets*
    - *Ex: A specific person or product*
  - *Entity sets are the classes or types of objects in our model*
    - *Ex: Person, Product*
    - *These are what is shown in E/R diagrams – as rectangles*
    - *Entity sets represent the sets of all possible entities*

Notation (Chen)



Notation (UML)





# What an entity really means?

DDSI_060.MYSTUDENTS	
NAME	VARCHAR2 (100 BYTE)
P ID	NUMBER
ADDRESS	VARCHAR2 (150 BYTE)
CONTRY	VARCHAR2 (20 BYTE)
MYSTUDENTS_PK (ID)	
MYSTUDENTS_PK (ID)	

Oracle SQL Developer : Tabla DDSI\_060.MYSTUDENTS@rabida

Archivo Editar Ver Navegar Ejecutar Equipo Herramientas Ventana Ayuda						
Conexiones						
lab1 rabida MYSTUDENTS						
Conexiones						
lab1 rabida						
Tablas (Filtrado)						
MYSTUDENTS						
NAME						
ID						
ADDRESS						
CONTRY						
Vistas						
Vistas de Edición						
Índices						
Paquetes						
Columnas Datos   Model   Restricciones   Permisos   Estadísticas   Disparadores   Flashback   Dependencias   Detalles   Particiones   Índices   SQL						
Acciones...						
COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	
1 NAME	VARCHAR2 (100 BYTE)	No	(null)		1 (null)	
2 ID	NUMBER	No	(null)		2 (null)	
3 ADDRESS	VARCHAR2 (150 BYTE)	Yes	(null)		3 (null)	
4 CONTRY	VARCHAR2 (20 BYTE)	Yes	(null)		4 (null)	

# How can we identify entities?

Sometimes it is difficult to decide if a certain object or concept can be modeled as an entity

Example: color is usually a property of an entity (the color of a car), but in a paint factory it would probably be appropriate to model “color” as an entity with its own properties

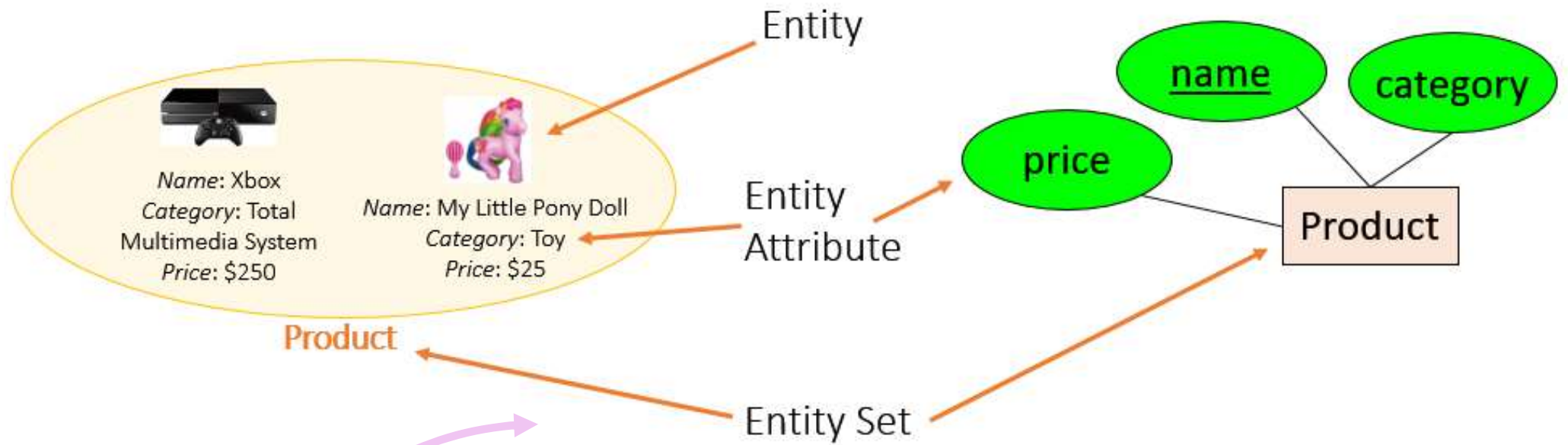


Some authors have tried to specify the concept of entity

TARDIEU et al. (1979) propose three general rules that an entity must comply with:

- it has to have its own existence
- each copy of a type of entity must be distinguishable from the others
- all instances of an entity type must have the same properties

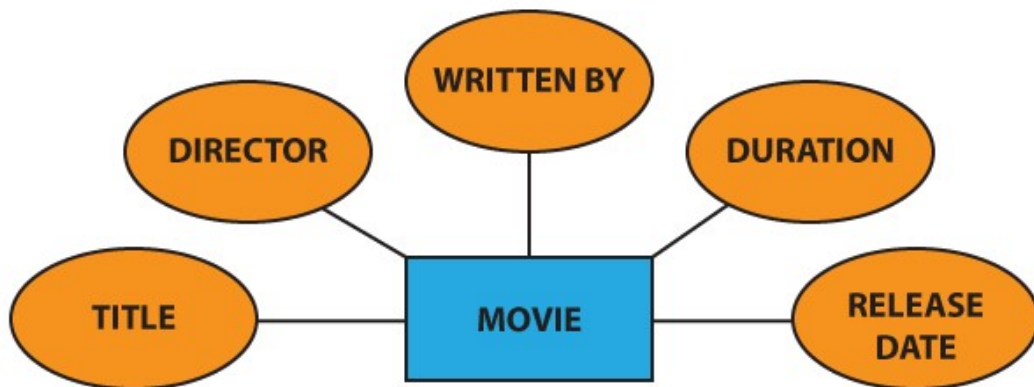
# Attributes



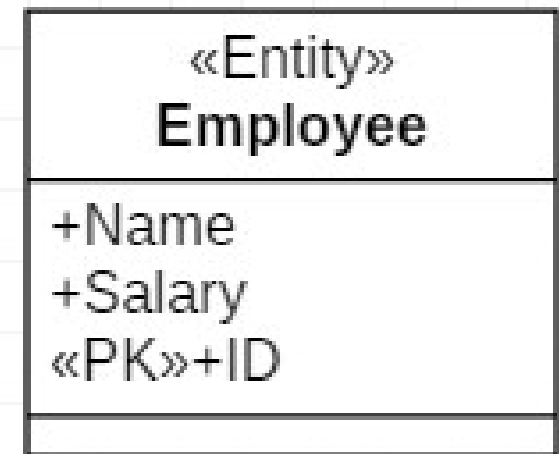
Example

# Attributes

In the Chen notation, each attribute is represented by an oval containing attribute's name



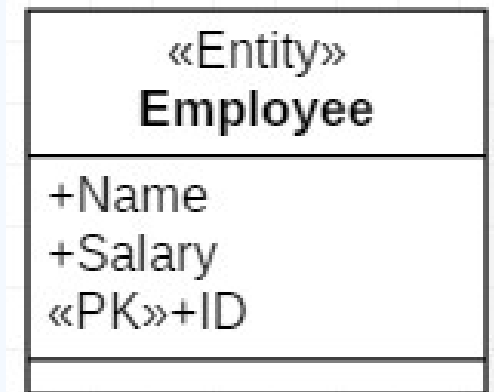
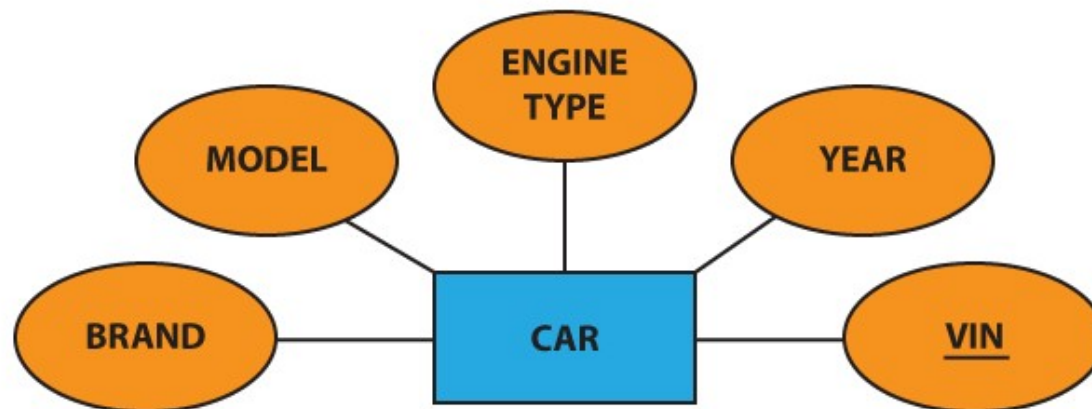
UML notation



Domain: set of possible values for an attribute – Attributes may share a domain

# Key attribute

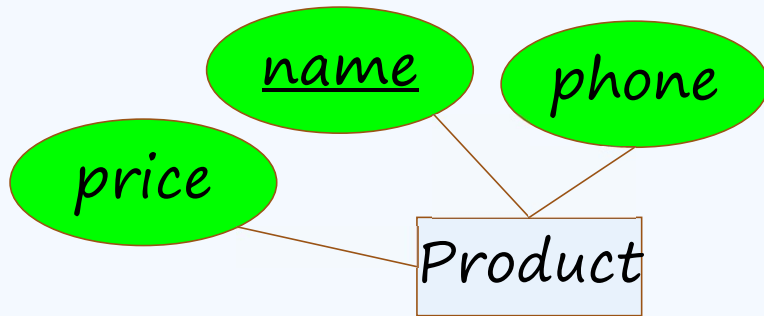
- *key attribute* – an attribute that uniquely identifies a particular entity.
- The name of a key attribute is underscored (Chen Notation).





## Key Attribute(s)

- A key is a minimal set of attributes that uniquely identifies an entity.

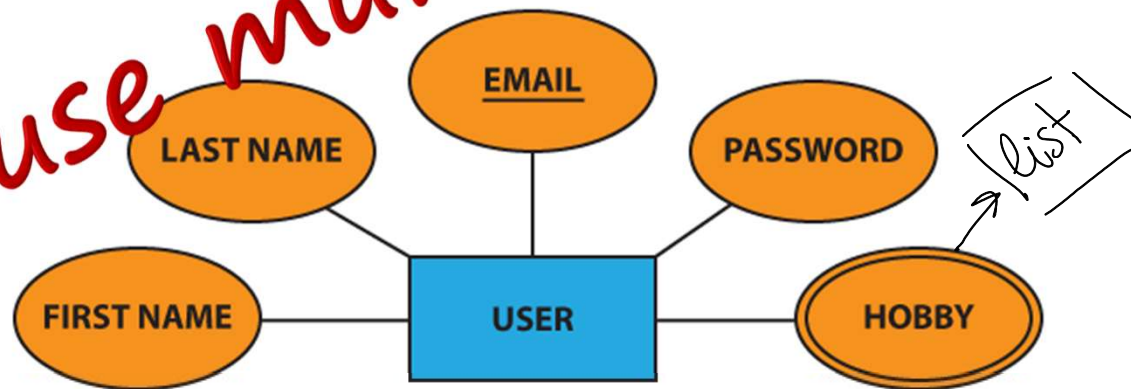


Here, {name, phone} is not a key  
(it is not minimal).  
If it were, what would it mean?

The E/R model forces us to designate a single primary key, though there may be multiple candidate keys

## Multivalued attribute

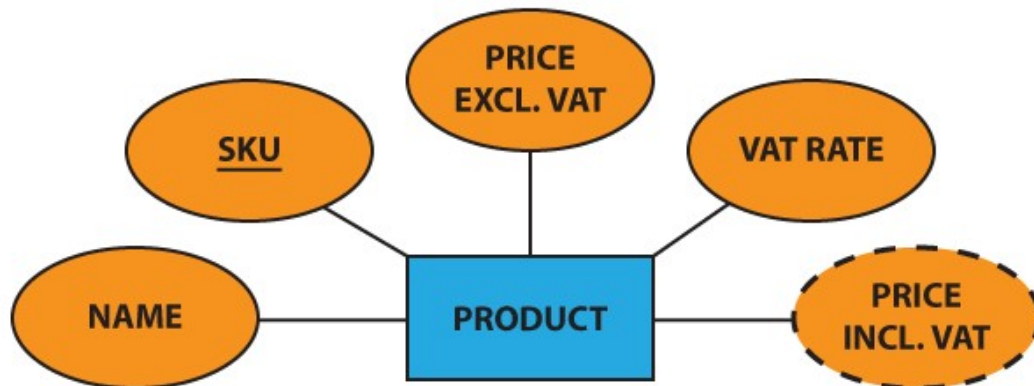
- **multivalued attribute** – an attribute that can have Many values (there are Many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval:



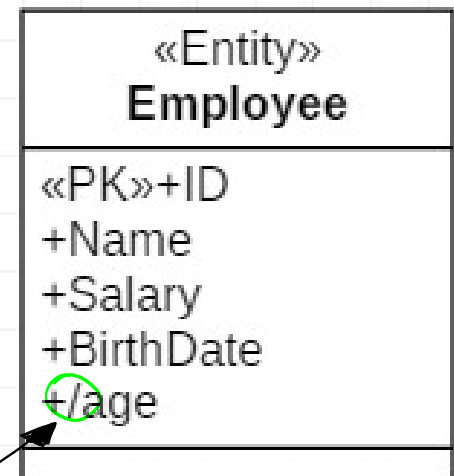
It have to be  
different attributes  
1 hobby  
2 hobbies  
⋮  
n hobbies

# Derived attribute

- *derived attribute* (or *computed attribute*) – an attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database.
- In the *Chen notation*, this attribute is represented by dashed oval



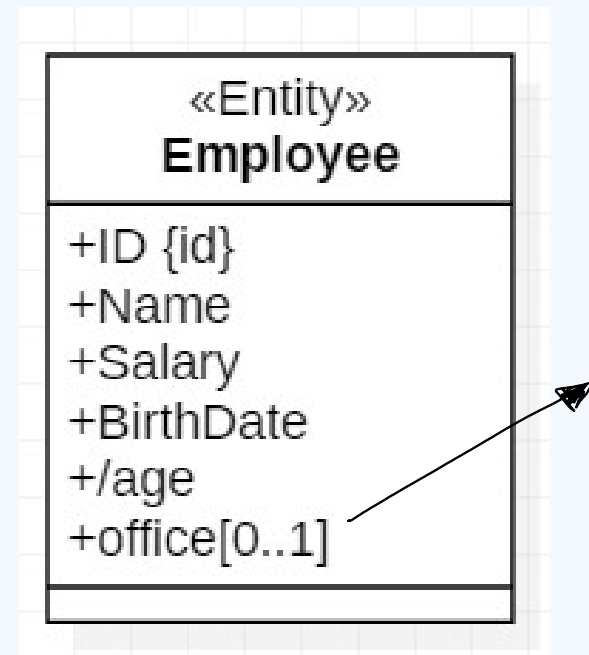
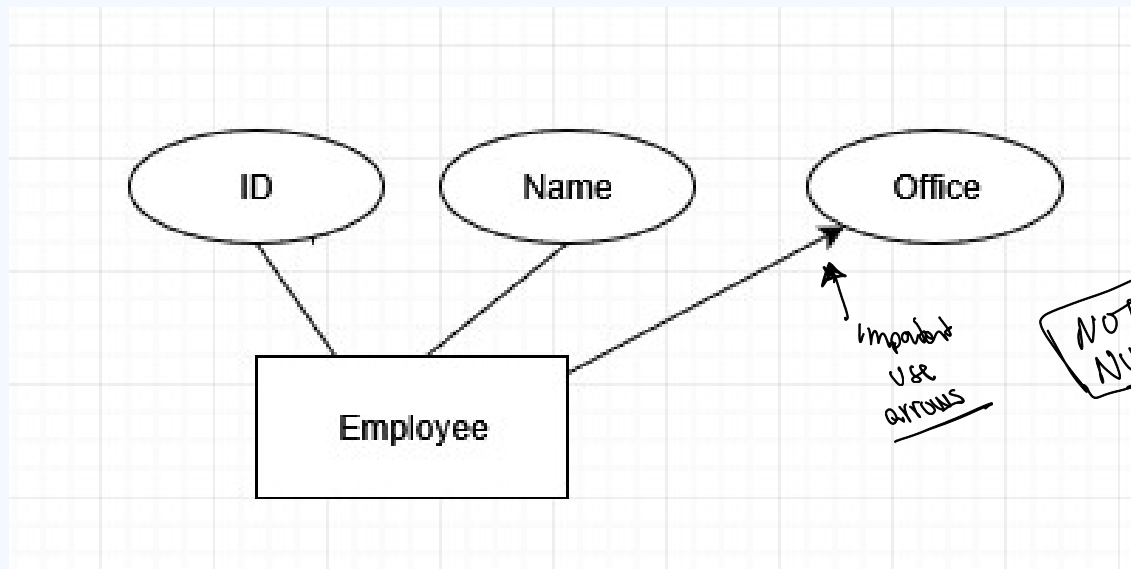
Can be calculated with other attributes,  
Means Redundancy  
Triggers  
to be well calculated!



UML

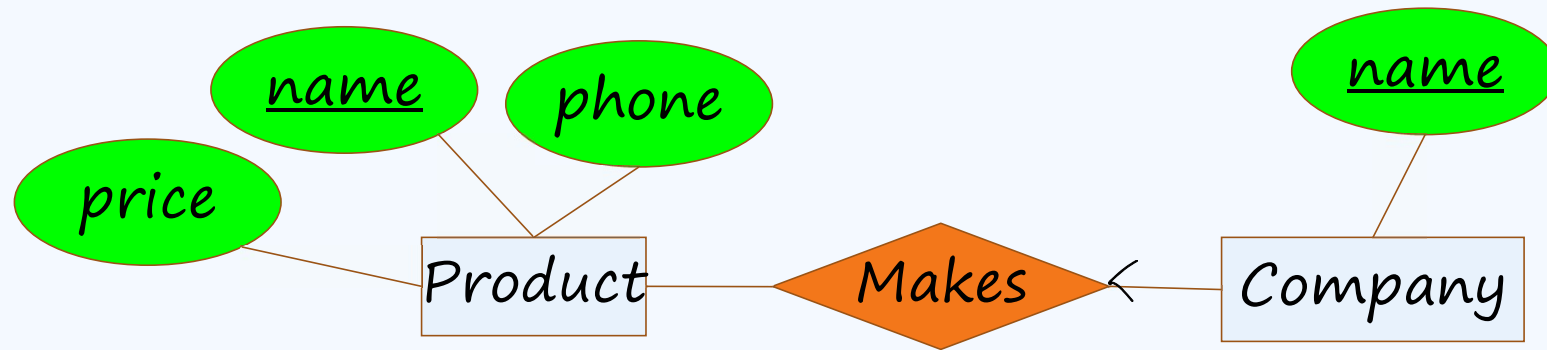
# Optional attribute

- Optional attribute: may be left empty



# Relationships

- A relationship is between two (or more) entities



Association or correspondence between entity instances

Example: Correspondence between the author of a book with the book he has written

A type of relationship is the set of relationships of the same type.

Example: A type of relationship WRITES relates the types of entities BOOK and AUTHOR

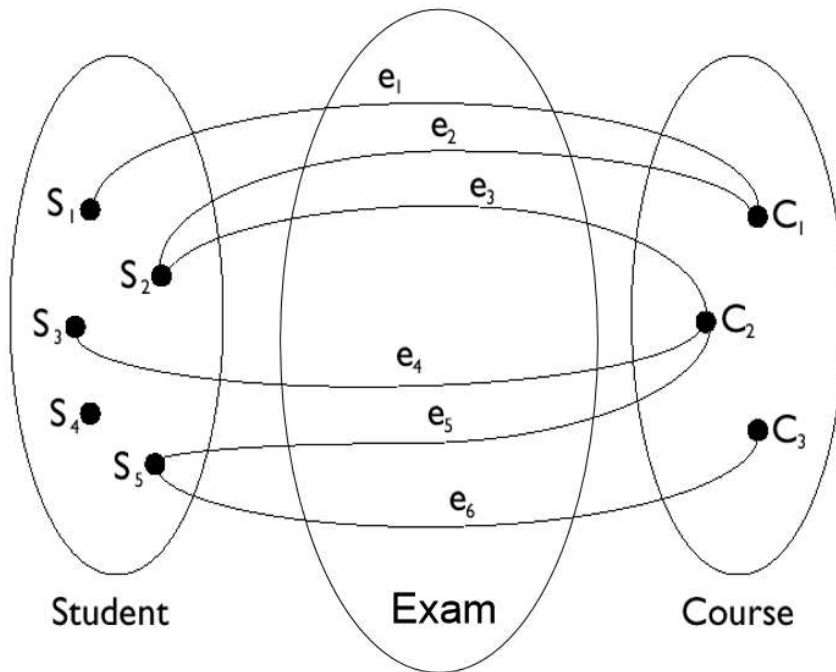
In the same way as with entities, we will simply call the relationship type the relationship



# But what does a relationship really mean?

## Example Instances for Exam

Adapted from chapter 5 of Atzeni et al, "Database Systems" McGraw Hill, 1999

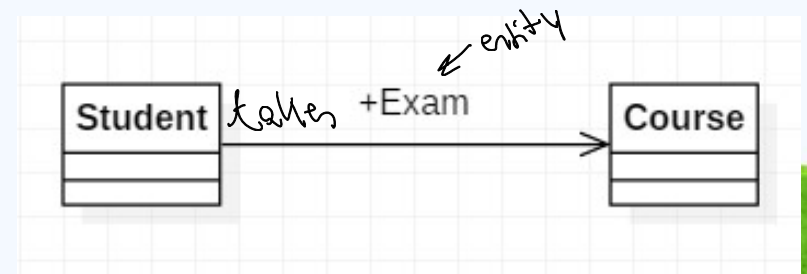


*Student and Course are entities*

- Their instances are particular students (eg S<sub>1</sub>) and particular courses (eg C<sub>1</sub>)

*Exam is a relationship*

- Its instances describes particular exams (eg e<sub>1</sub>)
- Each exam has exactly one associated student and course



But what does a relationship really mean?

**Company**

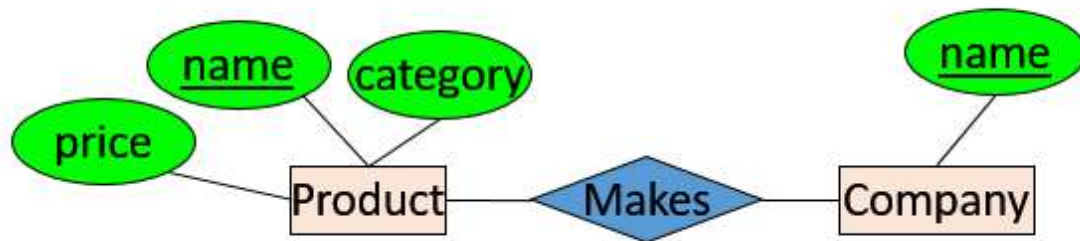
<u>name</u>
GizmoWorks
GadgetCorp

**Product**

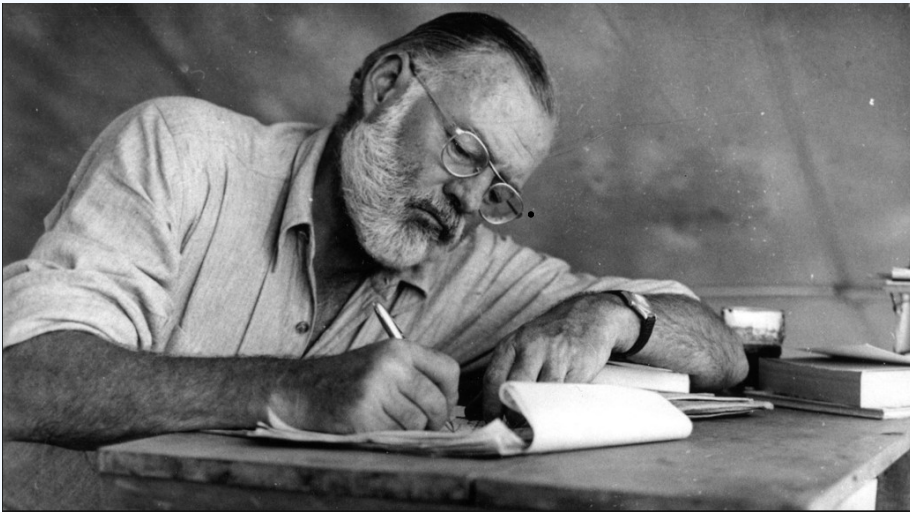
<u>name</u>	category	price
Gizmo	Electronics	\$9.99
GizmoLite	Electronics	\$7.50
Gadget	Toys	\$5.50

**Makes**

<u>C.name</u>	<u>P.name</u>
GizmoWorks	Gizmo
GizmoWorks	GizmoLite
GadgetCorp	Gadget



## Easy example



Writers (id, name, age) create novels (id, title, date)

Two or more novels can't have the same title



- Novels are bought by Clients (id, name, phone number)



## Easy example (II)

*Clients (id, name...) rent cars (id, model, color...). It is very import to know the model*

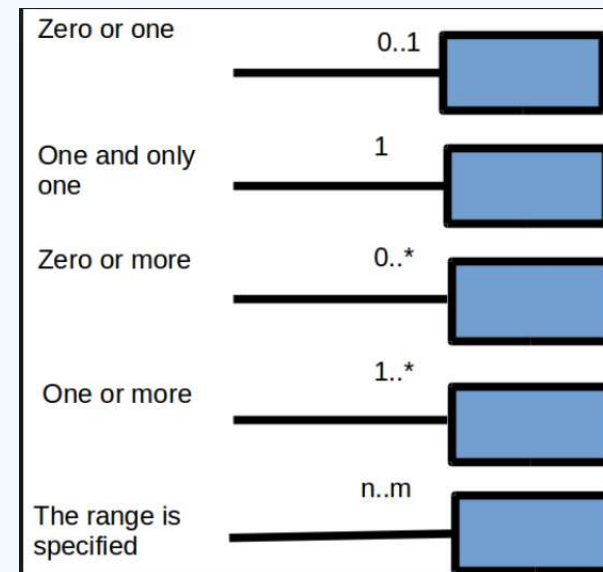
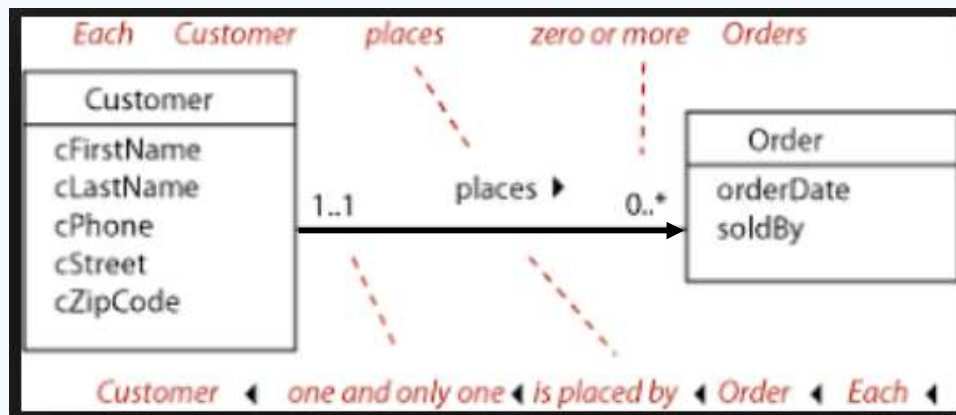


The background of the slide features a stylized landscape. The top half is composed of several layers of wavy, horizontal bands in various shades of blue, creating a sense of depth and movement. The bottom half is dominated by a large, light blue area that serves as a backdrop for the text. At the very bottom, there are rolling green hills in different shades of green, also rendered with a wavy, layered effect. The overall aesthetic is clean, modern, and visually appealing.

*Let's try your example...*

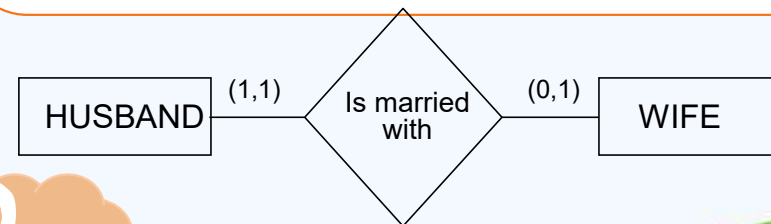
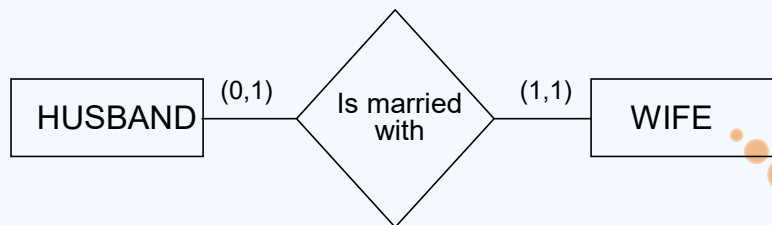
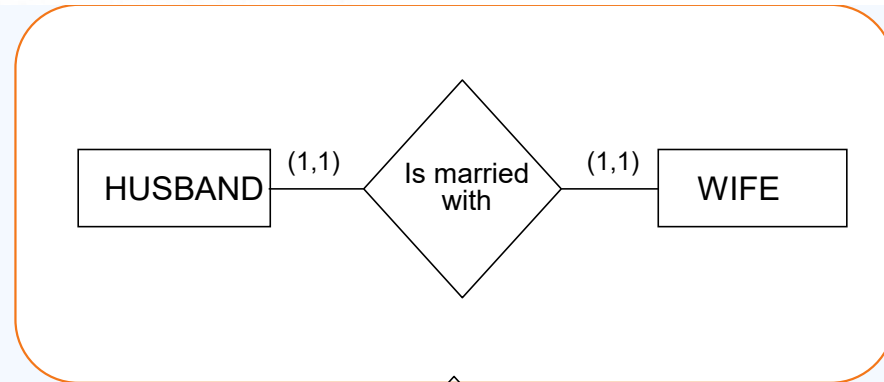
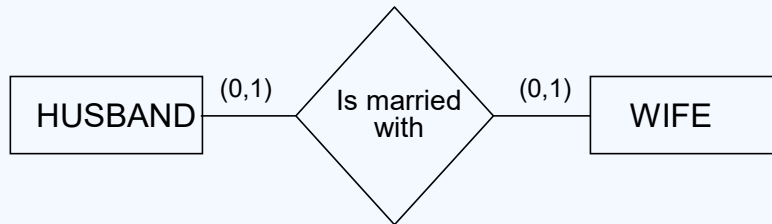


# Cardinality



# Cardinality

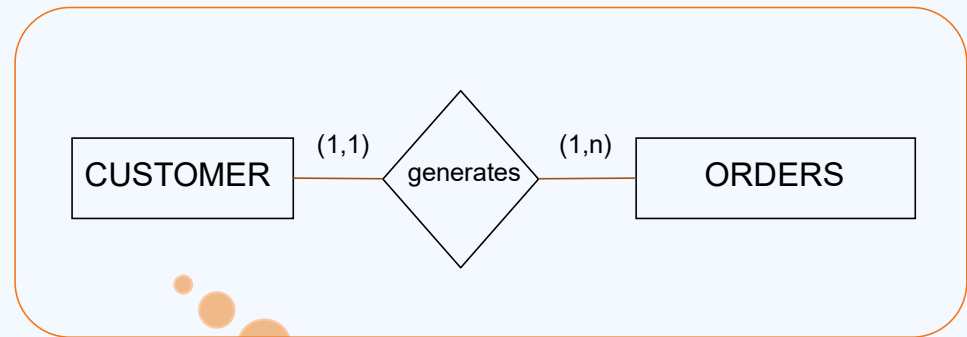
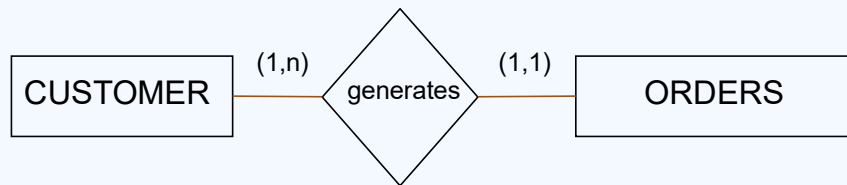
- When both participants can have only one instance of each other. (less used but stable).
- For Example, HUSBAND can have only one WIFE and WIFE can have only one HUSBAND. This is 1:1 relationship between HUSBAND and WIFE participants.



One to One  
One to Zero

## Cardinality – Example 2

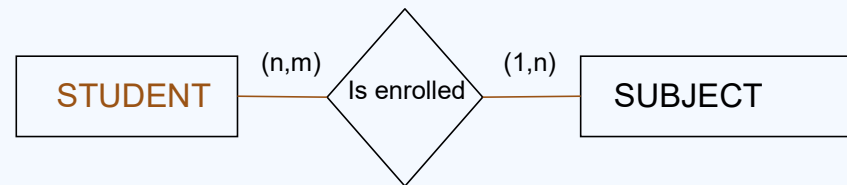
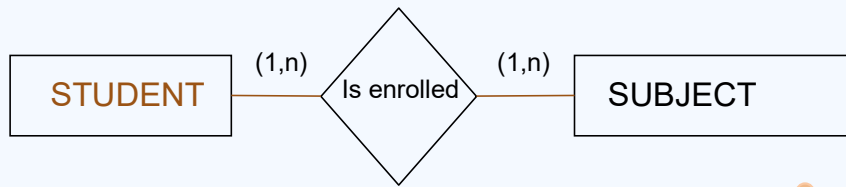
- When one participant can have multiple instances of other participants but other participant can have only one instance of first participants. (mostly used and stable).
- For Example, CUSTOMER may generate many ORDERS but each ORDER is generated by one CUSTOMER. This is 1:M relation between CUSTOMER and ORDER.



*One to Many*

## Cardinality – Example 3

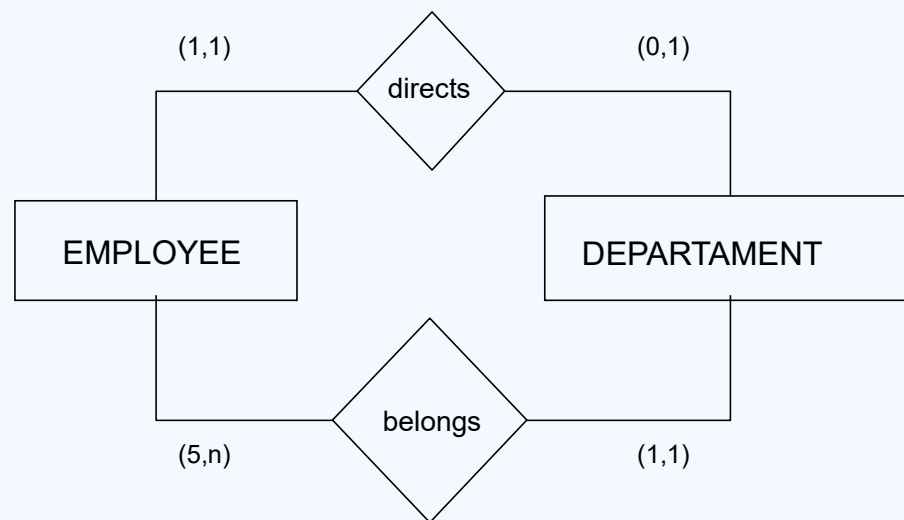
- Both participants can have multiple instances of each other
- A Student can be enrolled in Many Subjects and one Subject can be chosen by Many Students



Many to Many

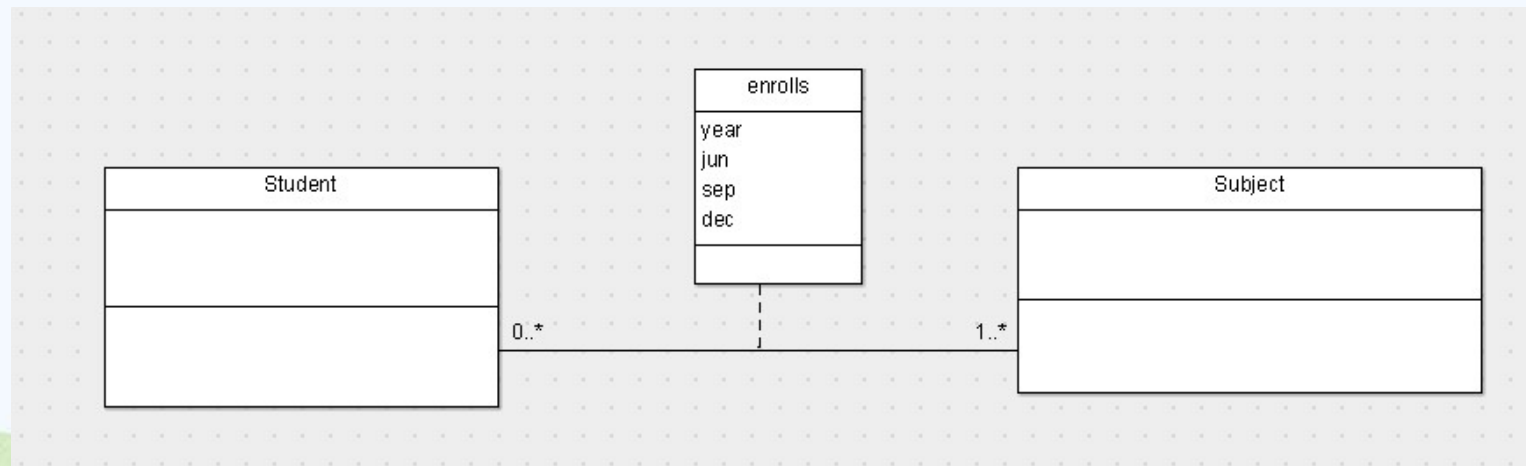
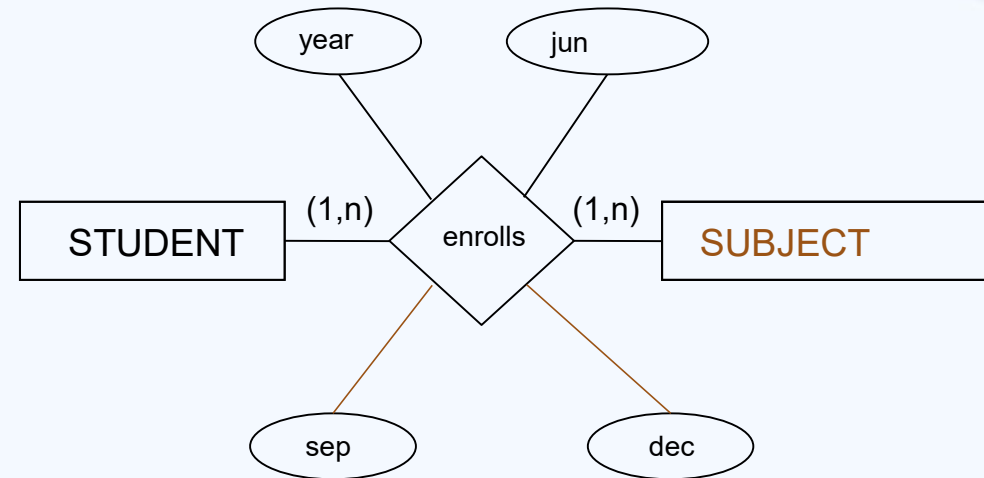
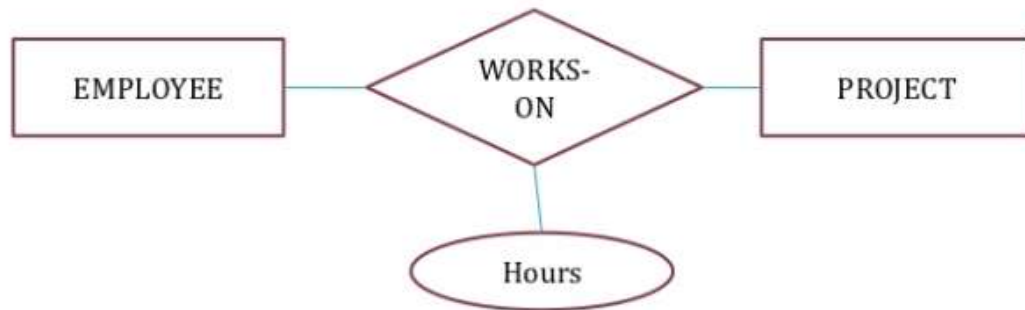


## Cardinality – Example 4



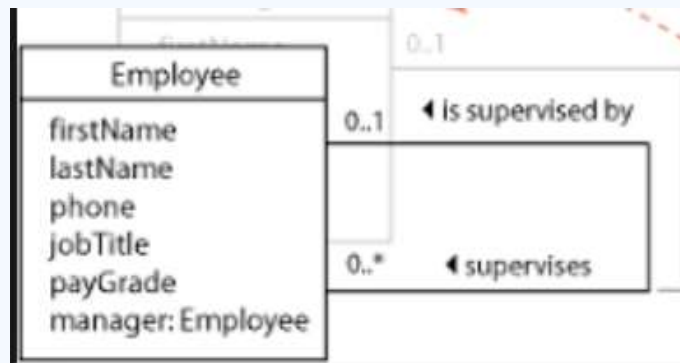


# Relationships with attributes



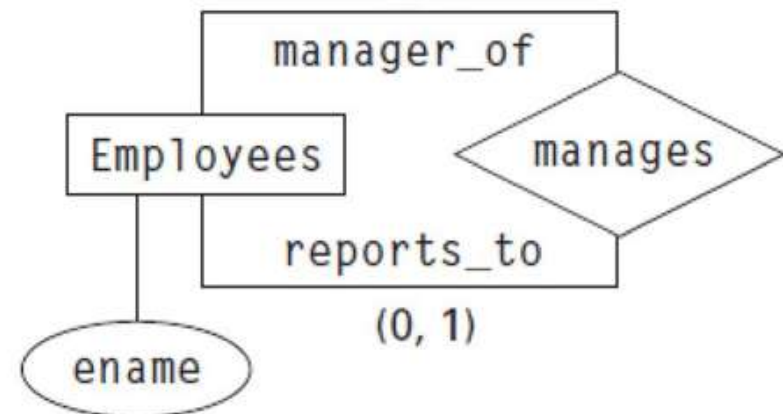
# Recursive relationships

- A recursive relationship is one in which a relationship can exist between occurrences of the same entity

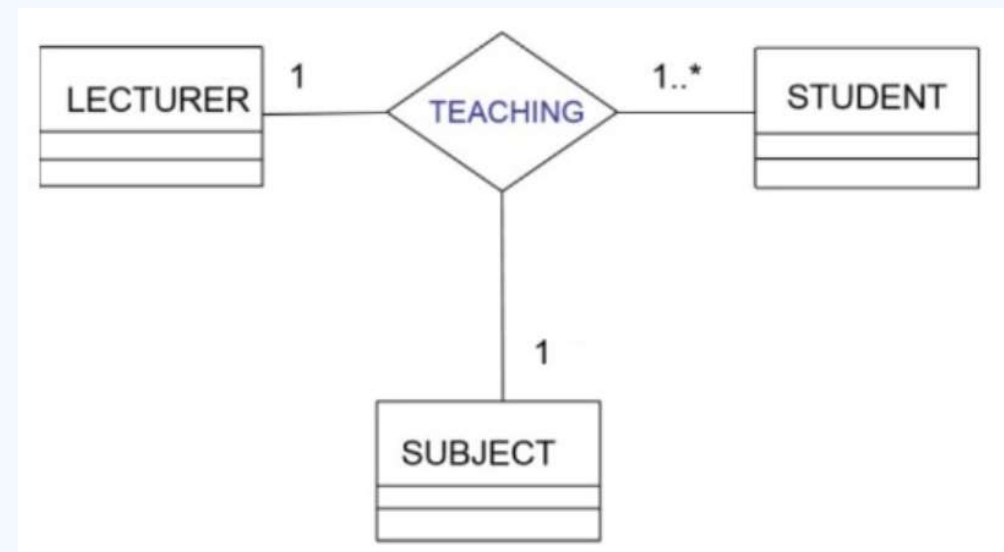
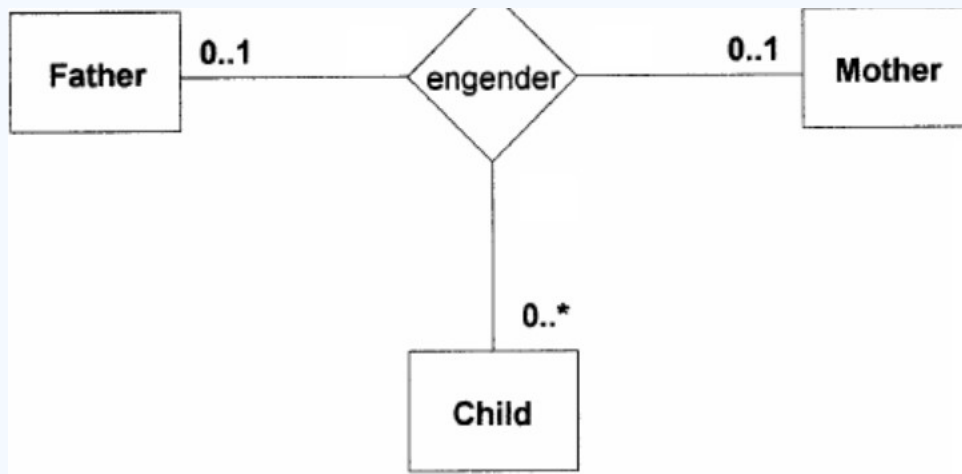


Each employee has at least zero managers (the case of general manager/CEO) and at most one manager (reports to)

A manager could have zero (☺) or more subordinates (0, N)



# Ternary Relationships



# Conceptual Model $\rightarrow$ Logical Model

Three fundamental principles:

- Every entity is transformed into a table
- Every relationship with maximum cardinality Many to Many (M: N) is transformed into a table
- Any relationship with maximum cardinality one to Many (1: N) can be transformed in two ways:
  - by key propagation
  - by transforming into a new table

# Relationships – Implementation – Many to Many

An M: N relation is transformed into a table whose primary key is the concatenation of the primary keys of the tables that are generated when transforming the entities that are part of the relationship. Along with these attributes are included the attributes of the relationship

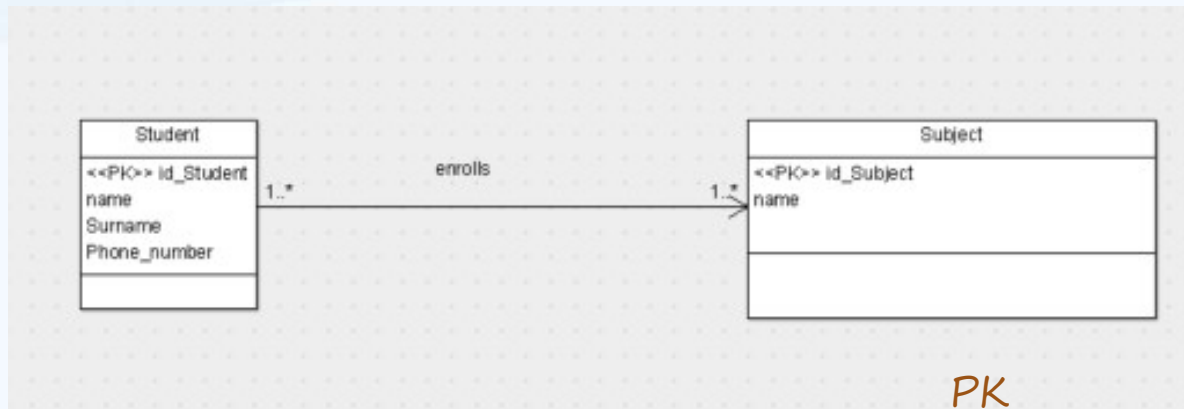
Each of the attributes that form the primary key of this table are foreign keys (FK) that refer to each of the tables where said attribute is a primary key

In a transformation of a Many-to-Many relationship, it is sometimes necessary to include, in the primary key, some attribute proper to the relationship. It usually occurs when the relationship between the same two rows can be repeated and / or we wish to maintain a historical record over time



# Relationships – Implementation (I)

## M:N



**STUDENT** (id\_Student, name, surname, phone\_number)  
PK: id\_Student  
NN: name  
NN: surname  
NN: phone\_number

**SUBJECT** (id\_Subject, name)  
PK: id\_Subject  
Unique: name  
NN: name

**ENROLLS** (id\_Student, id\_Subject)  
PK: (id\_Student, id\_Subject)  
FK: id\_Student → STUDENT  
id\_Subject → SUBJECT

PK

Id_Student	name	Surname	Phone_number
S001	Patricia	Smith	567889988
S002	Silvia	Perez	33344466888
S003	Carlos	Smith	222333444

PK

id_Student	Id_Subject
S001	ID01
S002	ID01
S003	ID01
S002	ID89

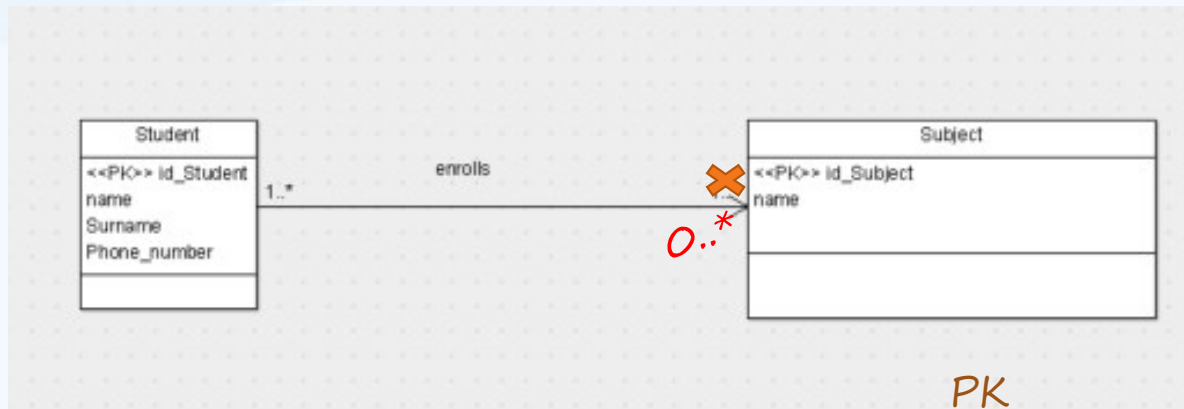
PK

Id_Subject	name
ID01	Information Systems
ID89	Meteorology

FK

FK

# Relationships – Implementation (II)



**STUDENT** (id\_Student, name, surname, phone\_number)

PK: id\_Student

NN: name

NN: surname

NN: phone\_number

**SUBJECT** (id\_Subject, name)

PK: id\_Subject

Unique: name

NN: name

**ENROLLS**(id\_Student, id\_Subject)

PK: (id\_Student, id\_Subject)

FK: id\_Student → STUDENT

id\_Subject → SUBJECT

PK

Id_Student	name	Surname	Phone_number
S001	Patricia	Smith	567889988
S002	Silvia	Perez	33344466888
S003	Carlos	Smith	222333444

FK

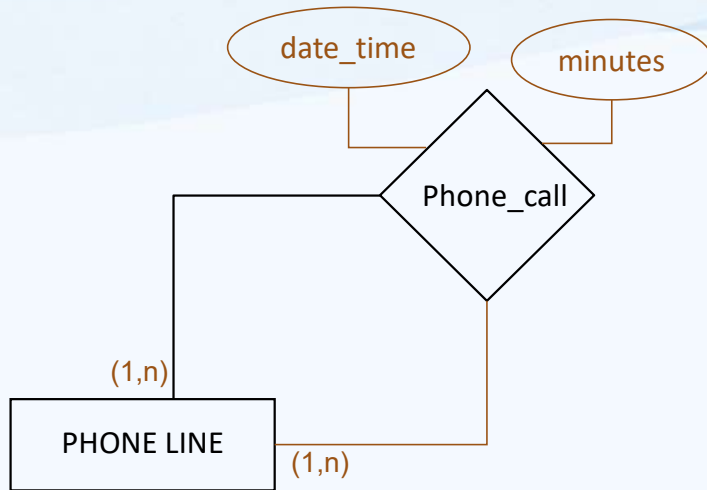
id_Student	Id_Subject
S001	ID01
S003	ID01
S002	ID89

FK

PK

Id_Subject	name
ID01	Information Systems
ID89	Meteorology

# Exceptions



**PHONE\_CALL** (origin, destination, date\_time, minutes)  
PK: (origin, date\_time)  
UNIQUE: (destination, date\_time)  
NN: minutes

origin	destination	Date_time	minutes
4444	5555	2018/01/01 14:10:00	30
4444	5555	2018/01/01 14:50:00	45
5555	4444	2018/01/03 19:00:00	120

- This PK (table PHONE\_CALL) can not be formed only by the origin and destination phone lines since it would prevent that from a phone line, the same phone number could be called more than once.
- Adding the date to the key would solve that problem. However, this solution would allow that from one telephone you could call several phone numbers at the same time (only the combination of "origin, date\_time") would be repeated.
- That situation can not occur in this problem. In this case there is a subset of attributes that would also identify each row of the table. Therefore, that will be the main key.

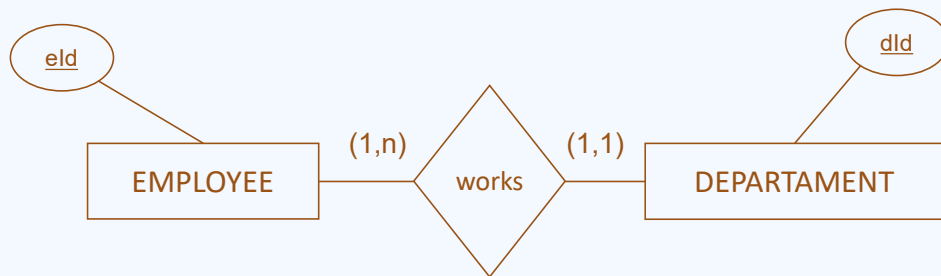
## Example

- Draw an E-R schema to describe the following domain

The University of Huelva has several departments. Each department is managed by a department Manager (professor). Professors must be assigned to one, but possibly more departments. At least one professor teaches each course, but a professor may be on sabbatical and not teach any course. Each course may be taught more than once by different professors. We know of the department name, the professor's name, the professor's employee ID, the course names, the course schedule, the term/year that the course is taught, the departments the professor is assigned to and the department that offers the course.

# Relationships – Implementation – One to Many

- 2 ways:
  - Include in the *M-side* the PK of the entity *1-side* (PK) and all the attributes of the relationship.
  - Create a new relation whose attributes are the primary keys of the other two, which will also be foreign keys to them. PK will be the PK of the M-side



**EMPLOYEE** (eId, ..., departament)  
Pk: eId  
FK: departament → DEPARTAMENT(dId)  
NN: departament

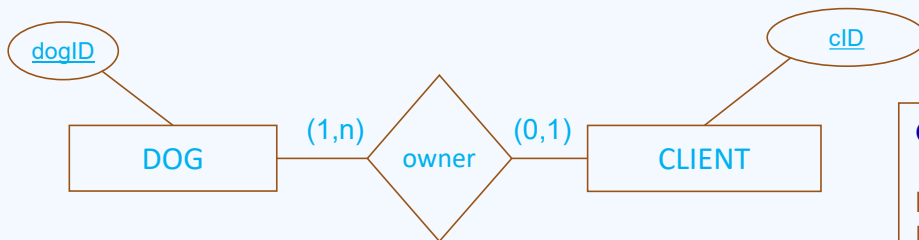
**DEPARTAMENT** (dId, ...)  
Pk: dId



# Relationships – Implementation – One to Many

- 2 ways:
  - Include in the M-side the PK of the entity 1-side (PK) and all the attributes of the relationship.
  - Create a new relation whose attributes are the primary keys of the other two, which will also be foreign keys to them. PK will be the PK of the M-side

This option can be used if few tuples participate in the relationship to avoid excessive NULL values in the foreign key or  
When the relationships has attributes and we want to keep the meaning



## OPTION 1

**DOG** (dogID, name, size, ..., owner)  
PK: dogID  
FK: owner → CLIENT (cID)  
NN: name

**CLIENT** (cID, name, ...)  
PK: cID

## OPTION 2

**DOG** (dogID, name, size, ...)  
PK: dogID  
NN: name

**CLIENT** (cID, name, ...)  
PK: cID

**owner** (dogID, cID)  
PK: dogID  
FK: dogID → DOG  
cID → CLIENT  
NN: cID

# Relationships – Implementation – One to Many

- When the relationships have attributes and we want to keep the meaning

**Book** (bookID, title, ...)

PK: bookID

NN: title

**member** (memberID, passport, name, ...)

PK: memberID

Unique: passport

NN: passport, name

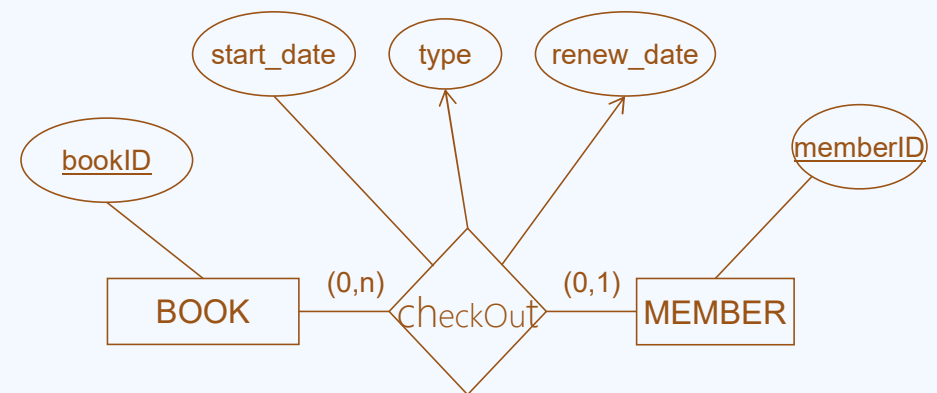
**checkout** (bookID, memberID, start\_date, renew\_date, type)

PK: bookID

FK: bookID → Book

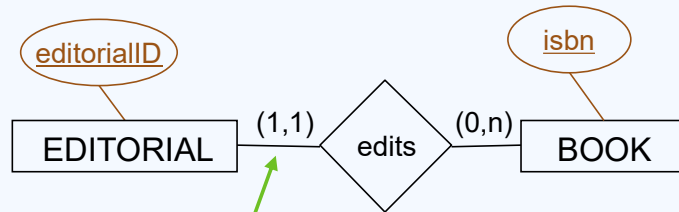
memberID → member

NN: memberID, start\_date



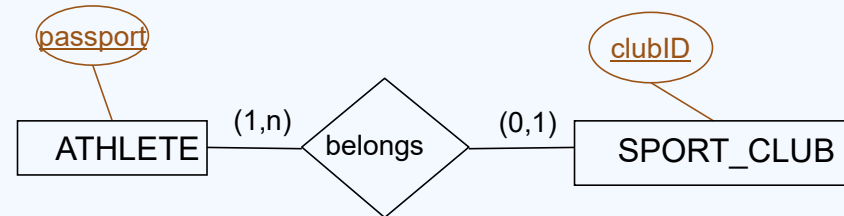
# Relationships – Implementation – One to Many

- Pay special attention to **minimum** cardinality to assign **NN** constraint correctly



**EDITORIAL** (editorialID, city, ...)  
PK: editorialID

**BOOK** (isbn, title, ..., editorial)  
PK: isbn  
FK: editorial → EDITORIAL(editorialID)  
**NN: editorial**

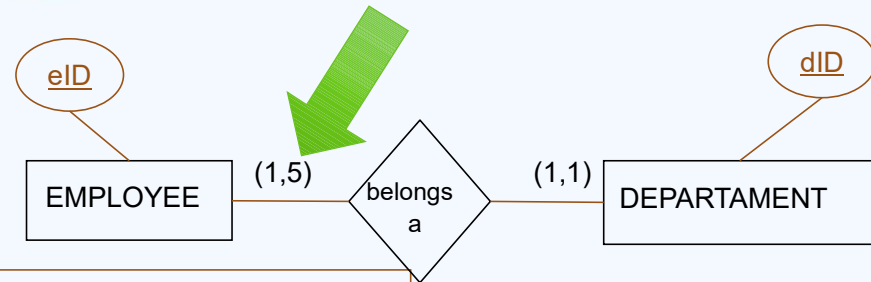


**SPORT\_CLUB** (clubID, name, city, ...)  
PK: clubID

**ATHLETE** (passport, name, ..., club)  
PK: passport  
FK: club → SPORT\_CLUB(clubID)

# Relationships – Implementation – One to Many

- What if we find this?



**EMPLOYEE** (eID, name, ..., departament)

PK: eID

FK: departament → DEPARTAMENT(dID)

NN: departament

**DEPARTAMENT** (dID, name, ...)

PK: dID

```
CREATE OR REPLACE TRIGGER maxEmployee
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
DECLARE
    numEmployee INTEGER;
BEGIN
    SELECT COUNT(*) INTO numEmployee FROM EMPLOYEE
        WHERE departament = :new.departament;
    IF numEMPLOYEEs == 5 THEN RAISE_APPLICATION_ERROR (-20001, 'Department' || 
        :new.departament || ' is full');
    END IF;
END;
```

# To take into account

- When an application that interacts with a DBMS is developed, it is necessary to decide in which layer they will control or test the restrictions or business rules

It is recommended to control restrictions at the database level

That way, the ide will be much simpler and portable

## Software Architecture Pattern

- **Presentation layer**

- Presentation of the web pages, UI forms and end user interacting API's

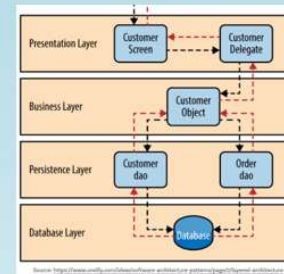
- **Business layer**

- The logic behind the accessibility, security and authentication happens in this layer. This includes ESB (Enterprise Service Buses), middle ware and other various request interceptors to perform validations.

- **Persistence layer**

- This is the presentation layer for the Data. This includes the DAO (Data Access Object) presentation, ORM (Object Relational Mappings) and Other modes of presenting persistent data in the application level
  - The persistence layer will be responsible for mapping between the Oracle DBMS relational model and the OO model of the Java programming language.

All the classes and methods necessary to manage the data stored in the database will be defined and programmed in this layer and also those business rules that could not be defined in the data layer



### Database layer

- Our data layer is the Oracle DBMS 11g
- Each user will have their own space in which they will store the schema of the database (tables) and it will be accessed from the persistence layer

In this layer you can define some of the business rules (LDD restrictions)

- SQL

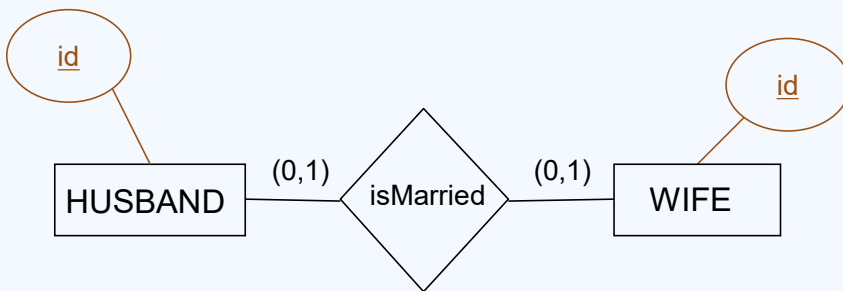


# Relationships – Implementation – One to One

- 2 options
  - Create a new table (like in M:N)
  - Add the PK of one table to the other as a PK

- Keep the meaning
- Avoid null values
- Minimum cardinalities
- Efficiency

If both are 0:1 and we know there isn't a lot of relations, we create a new table to avoid nulls



**HUSBAND** (id, name, ...)

PK: id

**WIFE** (id, name, ...)

PK: id

**isMarried** (idH, idW)

PK: idW

FK: idH → HUSBAND(id)

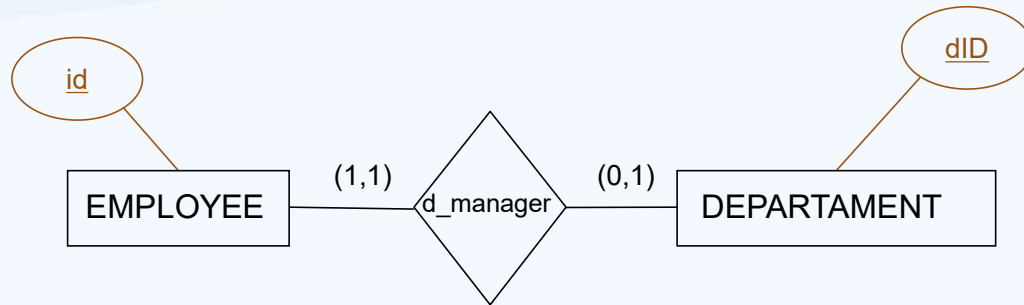
idW → WIFE(id)

NN: idH

UNIQUE: idH

# Relationships – Implementation – One to One

*If one is 0:1 and other is 1:1, to avoid NULLs we add the 1:1 PK to the 0:1 side as a FK*



**EMPLOYEE** (id, name, address, ...)

PK: id

**DEPARTAMENT** (dID, city, ..., manager)

PK: dID

FK: manager → EMPLOYEE(id)

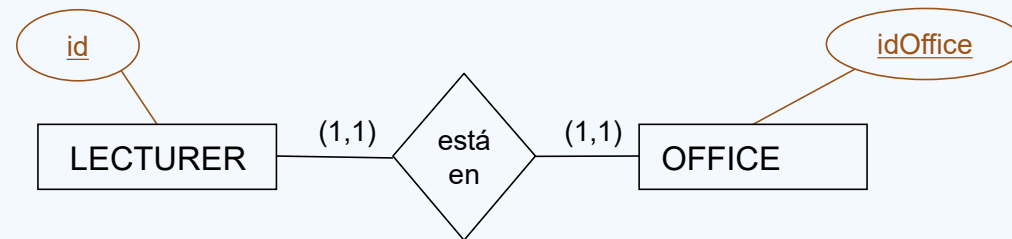
NN: manager

Unique: manager

*If both are 1:1 we can create a table or add PK as FKs*

# Relationships – Implementation – One to One

- If both are 1:1 we can keep all the information in only one table or add PK as FKs



**LECTURER** (id, name, ...)

PK: id

NN: name

**OFFICE** (idOffice, floor, ..., LECTURER)

PK: idOFFICE

FK: LECTURER → LECTURER(id)

NN: floor, LECTURER

Unique: LECTURER

**LECTURER** (id, name, ..., OFFICE)

PK: id

FK: OFFICE → OFFICE(idOFFICE)

NN: name, OFFICE

Unique: OFFICE

**OFFICE** (idOFFICE, floor, ...)

PK: idOFFICE

NN: floor

**LECTURER\_AND\_OFFICE** (id, name, ..., idOFFICE, floor, ...)

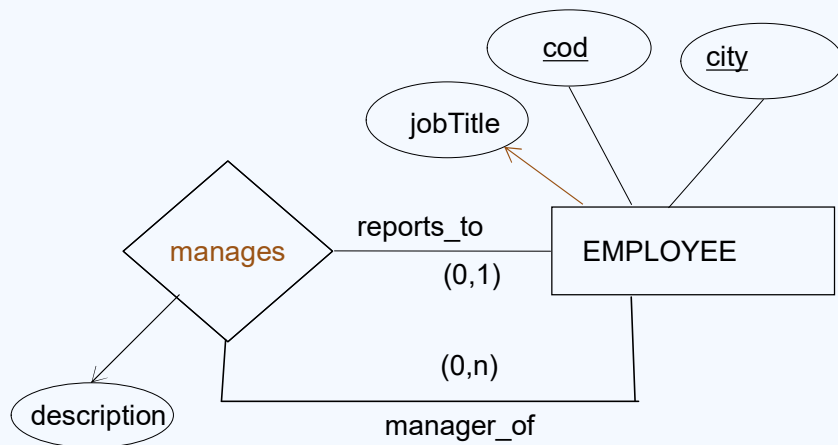
PK: id

NN: name, floor, idOFFICE

Unique: idOFFICE

# Recursive relationships – Implementation

- To derive reflexive relations the same rules are applied (but in this case there is only one entity associated with the relationship)

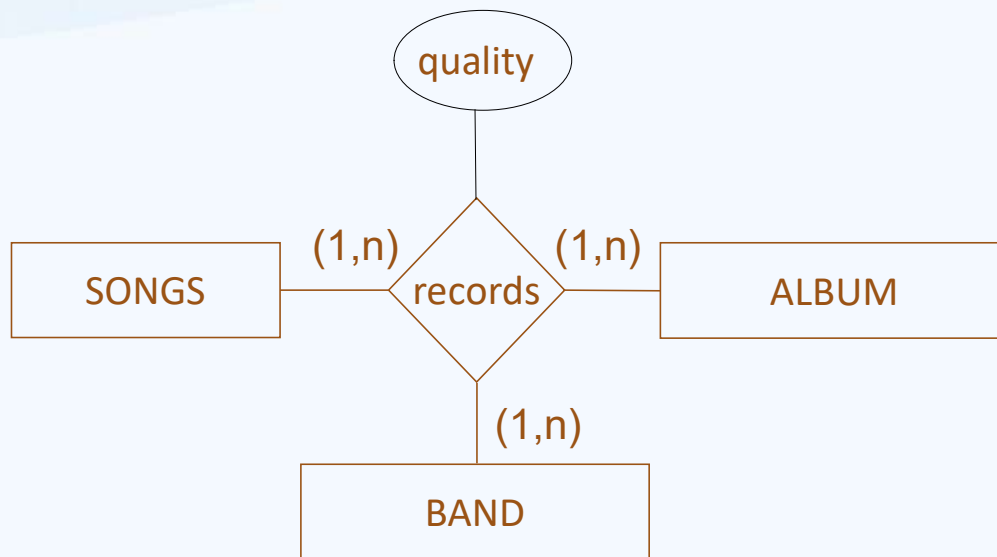


**EMPLOYEE** (**cod**, **city**, jobTitle, ..., **codManager**, **cityManager**, description)  
PK: (cod, city)  
FK: (codManager, cityManager) → EMPLOYEE(cod, city)

**EMPLOYEE** (**cod**, **city**, jobTitle, ...)  
PK: (cod, city)

**manager\_of** (**cod**, **city**, **codManager**, **cityManager**, description)  
PK: (cod, city)  
FK: (cod, city) → EMPLOYEE  
(codManager, cityManager) → EMPLOYEE(cod, city)  
NN: codManager, cityManager

# Ternary relationships M:M:M – Implementation



**SONGS** (songID, title, ...)

PK: songID

**BAND** (bandID, name, nationality, ...)

PK: bandID

**ALBUM** (idA, title,...)

PK: idA

**records** (**songID**, **bandID**, **idA**, quality)

PK: (songID, bandID, idA)

FK: songID → SONGS

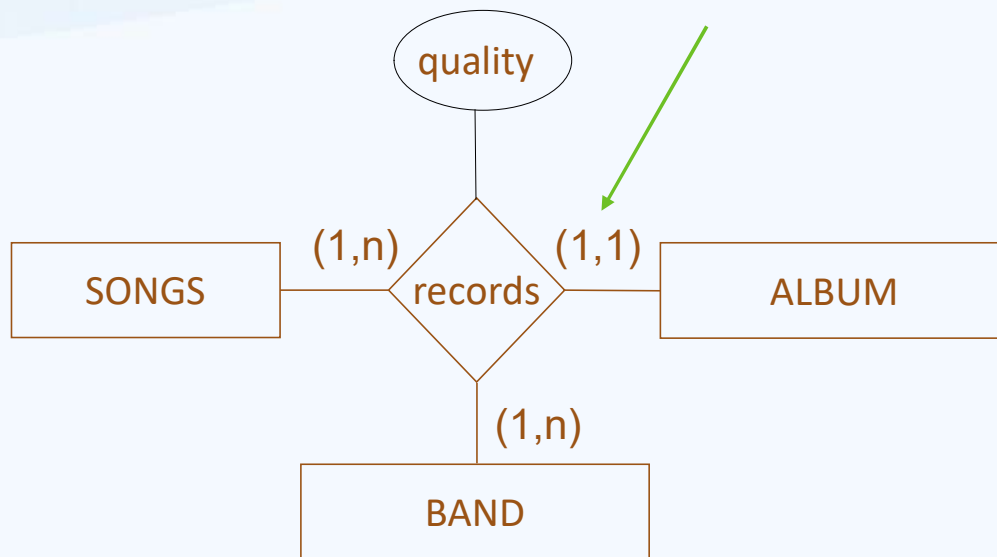
idA → ALBUM

bandID → BAND

NN: quality



# Ternary relationships M:M:1 – Implementation



**SONGS** (songID, title, ...)

PK: songID

**BAND** (bandID, name, nationality, ...)

PK: bandID

**ALBUM** (idA, title,...)

PK: idA

**records** (songID, bandID, idA, quality)

PK: (songID, bandID)

FK: songID → SONGS

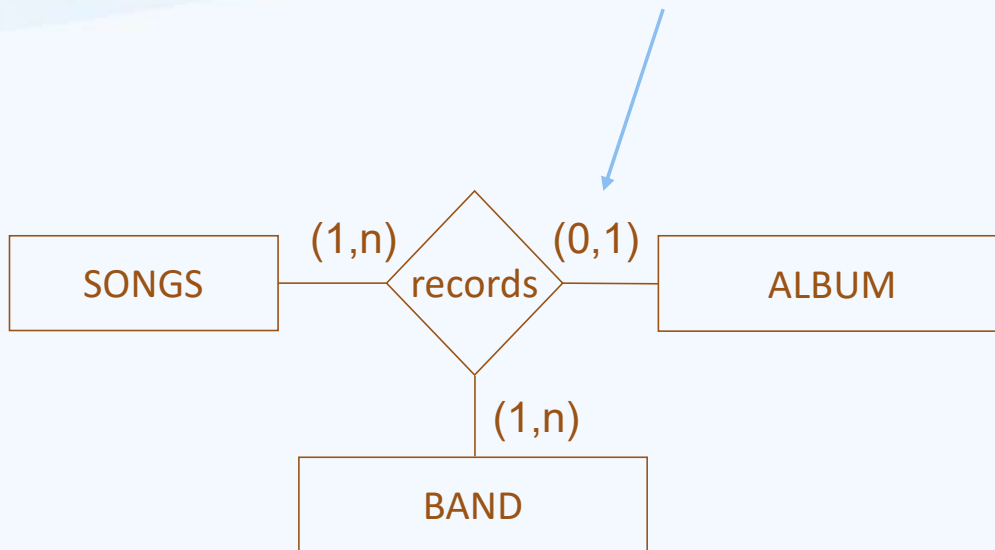
idA → ALBUM

bandID → BAND

NN: quality

NN: idA

Take into account...



Nulls -----> Efficiency problems

**SONGS** (songID, title, ...)

PK: songID

**BAND** (bandID, name, nationality, ...)

PK: bandID

**ALBUM** (idA, title,...)

PK: idA

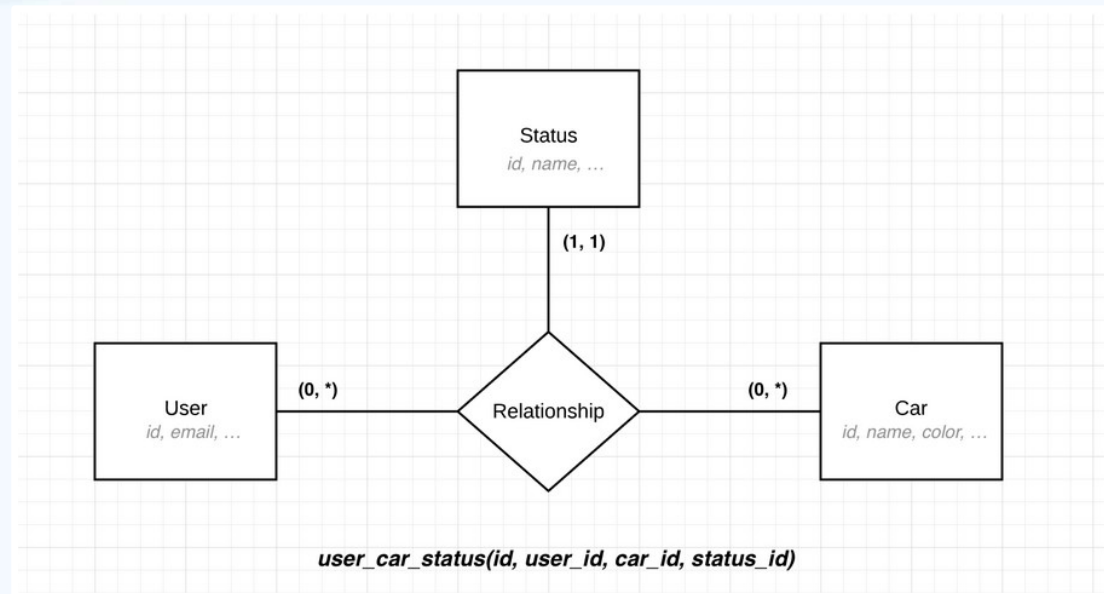
**records** (songID, bandID, idA)

PK: (songID, bandID)

PK: songID → SONGS

idA → ALBUM

bandID → BAND



## Ternary relationships M:1:1 - Implementation

If any entity is connected with "one": primary key = all attributes that make up the keys of all interrelated entities except one (that has connection "one")

## Partial key

- *partial key attribute (discriminator)* – an attribute that, when combined with the key attribute of the owner entity, provides a unique identification for the weak entity. We underline the discriminator with a dashed line:



## Weak entity

- **weak entity** – an entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity's identifier is a combination of the identifier of the owner entity and the partial key of the weak entity.



**WEAK ENTITY**



## Weak Entity – example

- Consider the building and classroom entities and suppose that there may be classrooms with the same number in different buildings. Class A1.1 and A2.1 can be found in building A, and class C1.1 and C1.2 can be found in building C.

In this case, the classroom identifier does not completely identify a classroom. In order to fully identify a classroom, it is necessary to take into account in which building it is located. In fact, we can identify a classroom through the interrelation situation, which associates it with a single building. The identifier of the building where it is located together with the classroom number identifies it completely

We will go back to this example after studying relations