BD
Databases

UD 3
The relational model

Course 2022-2023

UD3 – The relational model

INDEX

1. The relational model

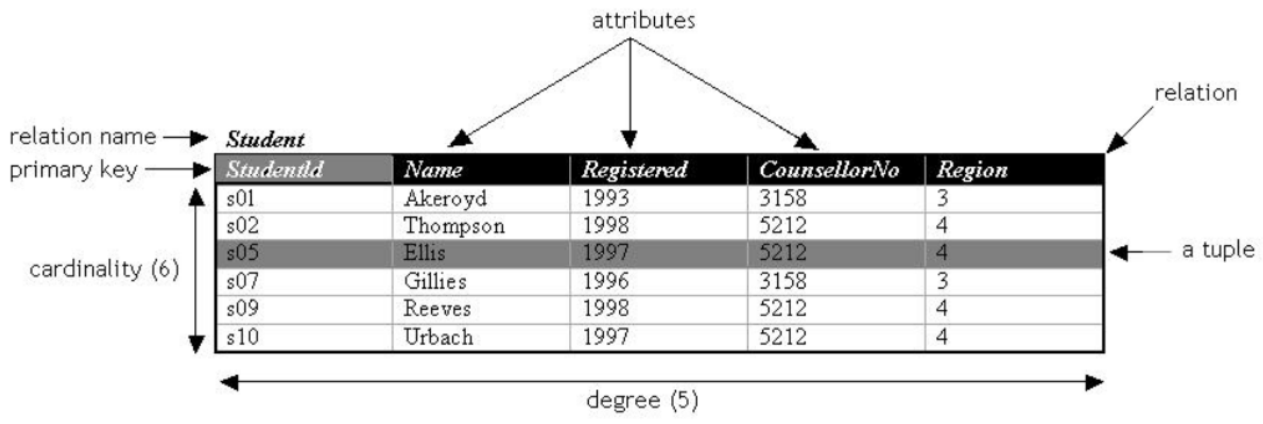
The entity-relationship model is a conceptual model that serves any type of DBMS, whereas the relational model is a logical model that only serves for relational DBMS.

All relational database designers and administrators use entity-relationship conceptual schemas because they fit this model very well.

You have to take into account the difference of the word relationship in both models. In the relational model a relationship is a **table** whereas in the Entity/Relationship it is the **association** that occurs between two entities.

1.1 RELATION (TABLE)

According to the relational model the fundamental element is what is known as a relationship, although it is more commonly called a table. It is a structure made up of rows and columns that stores data relating to a particular entity or relationship in the real world.



StudentId	Name	Registered	CounsellorNo	Region
s01	Akeroyd	1993	3158	3
s02	Thompson	1998	5212	4
s05	Ellis	1997	5212	4
s07	Gillies	1996	3158	3
s09	Reeves	1998	5212	4
s10	Urbach	1997	5212	4

About a table, in addition to its name, we can distinguish the following:

Attribute
Represents a property that owns that table. Equivalent to the attribute of the E-R model. Corresponds to the idea of a field or column.

Tuple or Record
Each of the rows in the table. It corresponds to the idea of registration. It therefore represents each individual element (instance, occurrence) of that table.

Domain
A domain contains all the possible values that a particular attribute can take. Two different attributes can have the same domain. A domain is a finite set of values of the

same type. Domains have a name to be able to refer to it and thus be reusable in more than one attribute.

1.2 KEY

Candidate key

A set of attributes that uniquely identify each record in the relationship. That is, columns whose values are not repeated for that table. Candidate keys can be:

- ***Primary Key (PK)***

  Candidate key that is chosen as the identifier of the records. The primary candidate who best identifies each record in the context of the database. *For example, a field with the DNI would be a candidate key from a customer table, although if a customer code field exists in that relationship, it would be a better candidate for primary key, because it is a better identifier for that context.*

- ***Alternate Key (AK)***

  Any candidate key other than primary.

Foreign, foreign or foreign key (FK)

Attribute whose values match a candidate key (usually parent) in another table.

1.3 RESTRICTION

A constraint **is a condition that is enforced by the data** in the database. There are several types:

1. **Those that are defined by the fact that the database is relational:**
   - No two records can be the same
   - The order of the records is not significant
   - The order of the attributes (columns) is not significant
   - Each attribute can only take one value in the domain in which it is enrolled

2. **Those that are incorporated by users:**

   - **Primary Key (PK)**

     Causes that the attributes marked as primary key not to repeat values. In addition, it requires that these attributes cannot be empty. If the primary key is made up of multiple attributes none of them can be empty.

- **Unique Key (UQ)**
  The values of attributes marked in this way cannot be repeated. this restriction must be indicated on all alternate keys.

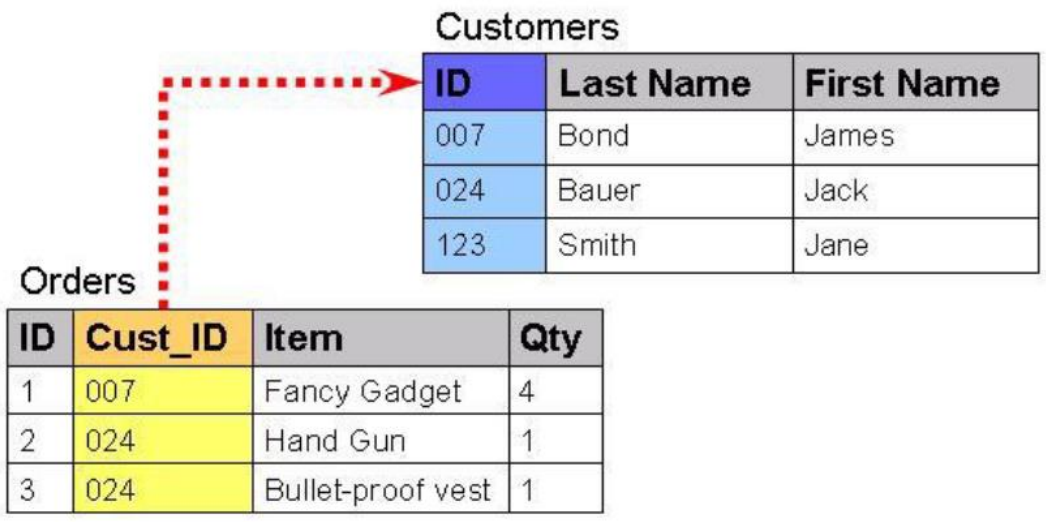
- **Mandatory (NOT NULL)**
  Avoids that the attribute marked in this way from having no value (that is, prevents it from containing the NULL value). NULL means: “no content in particular or void”

- **Check restriction (CHECK)**
  Allows to enter constraints related to the domain of the attributes. Examples. We can introduce CHECK restrictions on fields that affect:
  - Minimum and maximum price of a book 0€ and 50€
  - Book loan date cannot be earlier than the date of the day
  - Sale price of a house cannot be less than the purchase price

- **Referential integrity (FOREIGN KEY)**
  Used to indicate a foreign key. When a key is marked with referential integrity, you cannot enter values that are not included in the fields related to that key.

The latter, **referential integrity, causes problems in the operations of deleting and modifying records,** because if those operations are executed on the primary table there will be rows in the secondary table with the foreign key without integrity.

For example, we cannot delete the tuple 024 from Customers because, there will be an order without a customer.



Customers

ID	Last Name	First Name
007	Bond	James
024	Bauer	Jack
123	Smith	Jane

Orders

ID	Cust_ID	Item	Qty
1	007	Fancy Gadget	4
2	024	Hand Gun	1
3	024	Bullet-proof vest	1

This can be manipulated by adding the following clauses:

- **CASCADE**: Propagate the deletion of the affected records or modification of a key in a row in the referenced table.
- **SET NULL**: Assign the NULL value to foreign keys with the same value.
- **NO ACTION**: Foreign keys are not modified, nor are rows deleted in the table that contains them (deferred), the DBMS returns integrity error.
- **SET DEFAULT**: involves assigning the default value to foreign keys with the same value.

2. Converting E-R to relational model

Prior to the application of the rules of transformation of entity-relationship schemas to relational schemas, it is advisable to prepare the entity-relationship schemas by applying rules that facilitate and guarantee the reliability of the transformation process.

These preparatory rules are based on the application of the 1FN (*1st Normal Form*) and its goal is to eliminate the following anomalies:

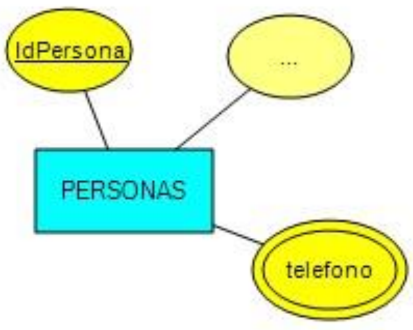
- Multivalued attributes
- Compound attributes

2.1 REMOVING MULTIVALUED ATTRIBUTES

All multiple attributes must be transformed into a weak entity type by existence with a many-to-many or one-to-many relationship, as the case may be, to the entity type on which it was defined. If the newly created entity is considered ambiguous, you can add attributes to it or inherit them from the other entity.

We assume for the following example that a person can have multiple phone numbers.

In this case it would be necessary to express the relational schema of the form:



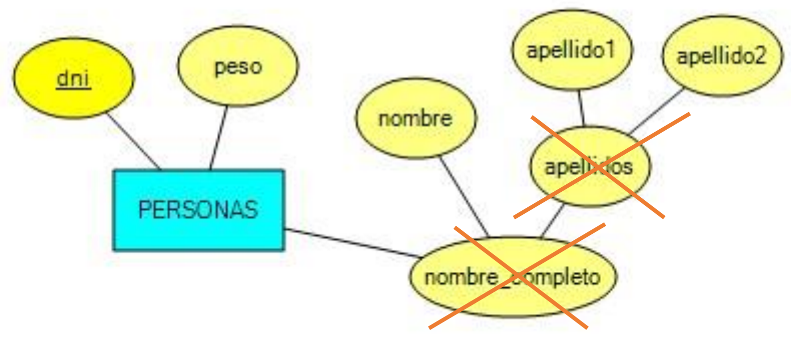
```
PERSON  (idPerson, ... atributtes ...)
PK:   idPerson

PHONE(idPerson, phone)
PK:   idPerson, phone
FK:   idPerson → PERSON
```

That is, you would have to create a PHONE table in which to save the phone numbers associated with the PERSON (that's why you have a key outside idPerson to the PERSON's table).

2.2 REMOVING COMPOSITE ATTRIBUTES

All composite attributes must be broken down into simple attributes that are associated with the same entity.

Analising this E-R schema:



It would be expressed using the following relational schema:

```
PERSON (DNI, peso, nombre, apellido1, apellido2)

PK: DNI
```

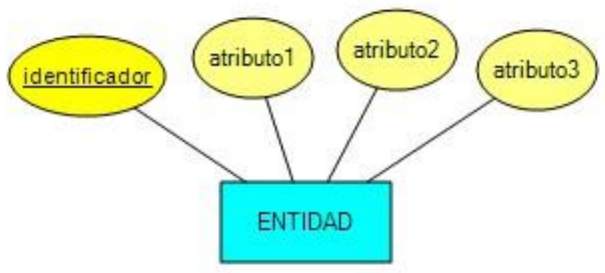
As can be seen, we have stayed with the terminal attributes (those that do not have children) and therefore they are simple and we can represent them in a table.

2.3 TRANSFORMING STRONG IDENTITIES

The strong entities of the E-R model are transformed to the relational model by following these instructions:

- **Entities**. Entities become tables.
- **Attributes**. Attributes become columns.
- **Main identifiers**. Become Primary Keys (PK).
- **Candidate identifiers**. Become Candidate Keys.

This causes the transformation to occur based on this example:



ENTITY (identificador, atributo1, atributo2, atributo3)

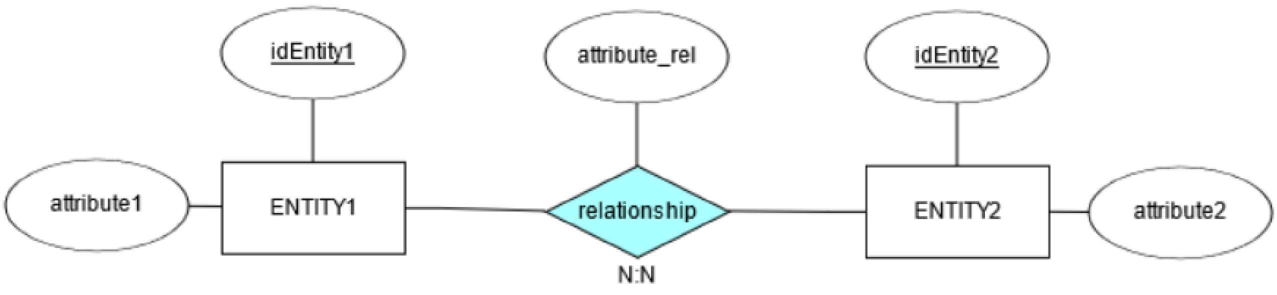
PK: identificador

2.4 TRANSFORMING RELATIONSHIPS

The initial idea is to transform each relationship into a table, but you have to distinguish according to the type of relationship.

Many-to-many relationships

In many-to-many relationships the relationship is transformed into a table whose attributes are: the attributes of the relationship and the keys of the related entities (which will become foreign keys). The key in the table is made up of all foreign keys.



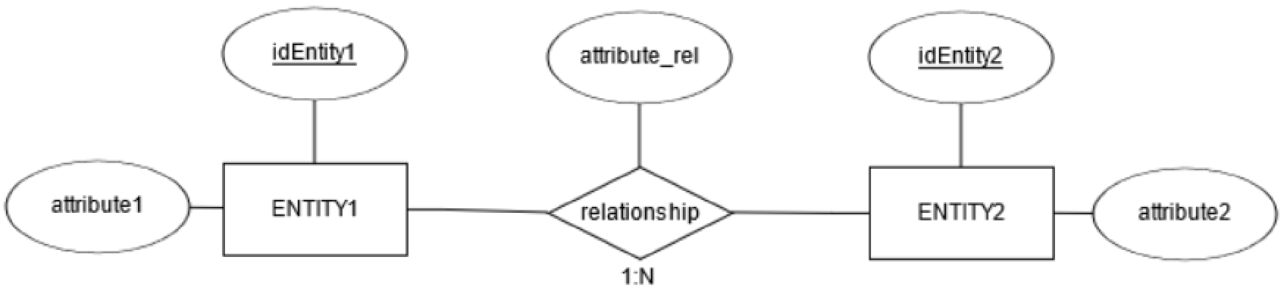
ENTITY1 (idEntity1, attribute1)
PK: idEntity1

ENTITY2 (idEntity2, attribute2)
PK: idEntity2

RELATIONSHIP (idEntity1, idEntity2, attribute_rel)
PK: idEntity1, idEntity2
FK: identificador1 → ENTIDAD1
FK: identificador2 → ENTIDAD2

One-to-many relationships

One-to-many relationships do not need to be transformed into a table in the relational model. **The table on the many side includes as an alien key the identifier of the entity on the side one.** In the event that the minimum number of the relationship is zero (there can be copies of entity one unrelated), null values must be allowed in the foreign key identifier1. Otherwise they cannot be allowed (as there will always be a related value).

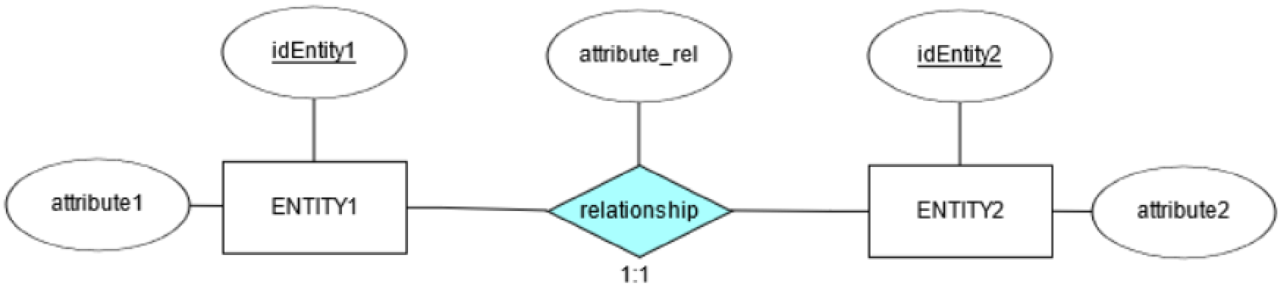


ENTITY1 (idEntity1, attribute1)
PK: idEntity1

ENTITY2 (idEntity2, attribute2, idEntity1, attribute_rel)
PK: idEntity2
FK: idEntity1 → ENTITY1

One-to-one relationships

In the case of one-to-one relationships, the same thing happens: the relationship is not converted to a table but is placed in one of the tables (it would not matter in which).



We could have two valid options:

Option 1

ENTITY1 (<u>idEntity1</u>, attribute1)
PK: idEntity1

ENTIDAD2 (<u>idEntity2</u>, attribute2, idEntity1, attribute_rel)
PK: idEntity2
FK: identificador1 → ENTIDAD1

Option 2

ENTITY1 (<u>idEntity1</u>, attribute1, idEntity2, attribute_rel)
PK: idEntity1
FK: idEntity2 → ENTITY2

ENTITY2 (<u>idEntity2</u>, attribute2)
PK: idEntity2

KEY POINT

Example: Let's see this relationship between TUTOR and STUDENT:

[TEACHER] <is tutor of> [GROUP OF STUDENTS]

A student would always have a tutor, whereas a teacher would not always be tutor of a group of students. In this case, it would be better to propagate the PK of the table Teacher to the table GROUP OF STUDENTS.

<u>DESIGN TIP</u>

Since the maximum cardinality of both sides is the same, we must look at the minimum cardinality. In case of equal minimum cardinalities (0,0) or (1,1) we can propagate the key towards any of the two sides of the relation (but only to one side).

In case one of the sides has minimum cardinality 1 and the other 0, we will propagate the key from side 1 to side 0.

Reflexive relationships

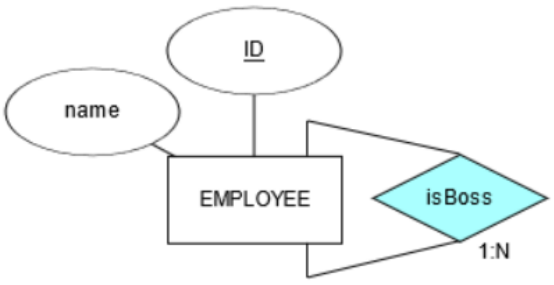
Reflexive relationships are treated in the same way as the others. The difference is that the same attribute can appear twice in a table as a result of the transformation.

Transformation process:

- **One-to many**: the primary key is included as a foreign key in the same table.
- **Many-to-many**: new table is generated, just as in binary relationships.

ONE-TO-MANY

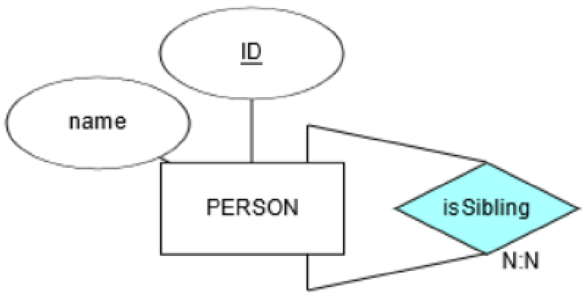
EMPLOYEE (ID, name, **ID_boss**)
PK: ID
FK: ID_boss → EMPLOYEE



MANY-TO-MANY

PERSON (ID, name)
PK: ID

SIBLING (ID1, ID2)
PK: ID1, ID2
FK: ID1 → PERSON
FK: ID2 → PERSON



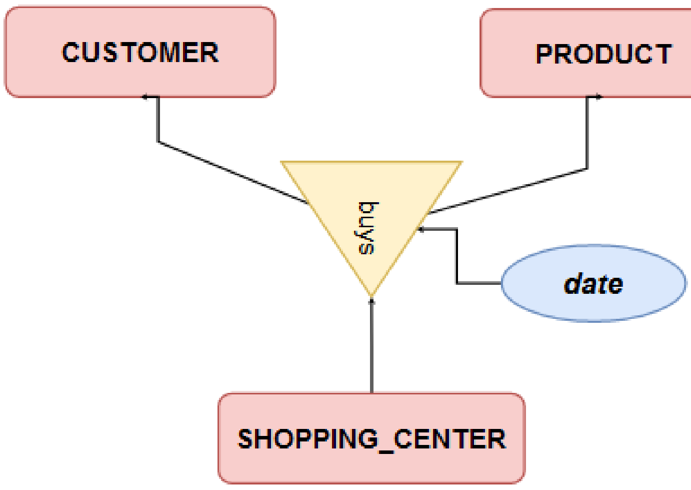
Ternary relationship

Ternary or n-ary relationships (where n is the number of entities participating in the relationship) are treated similarly to binary many-to-many relationships. Let's see each case one by one.

N:N:N cardinality

A new table needs to be created. The primary key of the new table will be the combination of the three PK of the related entities.

Example: *"Customers can buy products in shopping center"*



CUSTOMER (<u>idCustomer</u>, …)
PK: idCustomer

PRODUCT (<u>idProduct</u>, …)
PK: idProduct

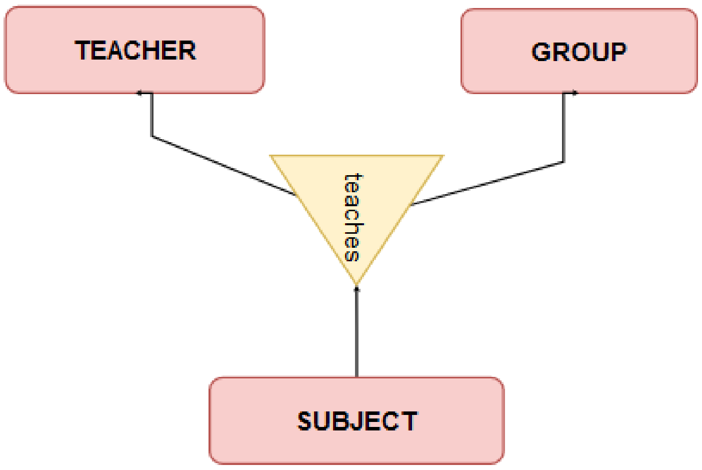
SHOPPING_CENTER (<u>idCenter</u>, …)
PK: idCenter

PURCHASE (idCustomer, idProduct, idCenter, date)
PK: idCustomer, idProduct, idCenter
FK: idCustomer → CUSTOMER
FK: idProduct → PRODUCT
FK: idCenter → SHOPPING_CENTER

1:N:N cardinality

A new table needs to be created. The primary key of the new table will be the combination of the two PKs of the related entities with cardinality N. The entity with cardinality 1 will only be FK in the new table.

Example: *"A teacher teaches one or more subjects to one or more groups"*.



TEACHER (<u>idTeacher</u>, …)
PK: idTeacher

GROUP (<u>idGroup</u>, …)
PK: idGroup

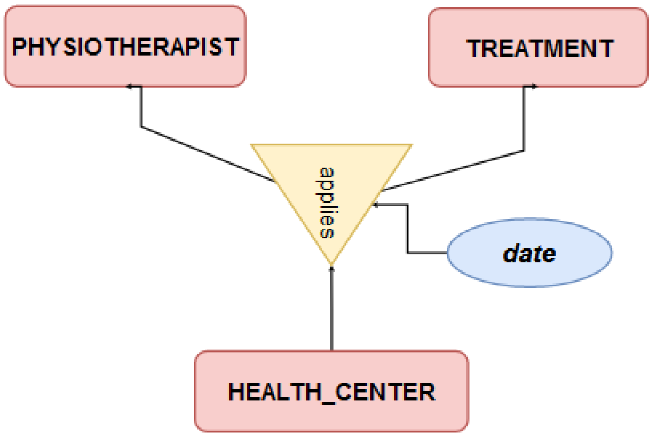
SUBJECT (<u>idSubject</u>, …)
PK: idSubject

TEACHES (<u>idSubject, idGroup</u>, idTeacher …)
PK: idSubject, idGroup
FK: idGROUP → GROUP
FK: idSubject → SUBJECT
FK: idTeacher → TEACHER (NOT NULL)

1:1:N cardinality

In this case, we DO NOT create a new table. The transformation will consist of treating it as in binary with cardinality 1:N, that is, the entity with cardinality N will receive the PK attributes of the entities with cardinality 1 as foreign keys. If there were an attribute in the relationship, it would be absorbed by the entity with cardinality N.

Example*. "A physiotherapist applies one or several treatments in a health center" (we are going to impose this restriction to force having 1:1:N).*



PHYSIOTHERAPIST (idPhysio, …)
PK: idPhisio

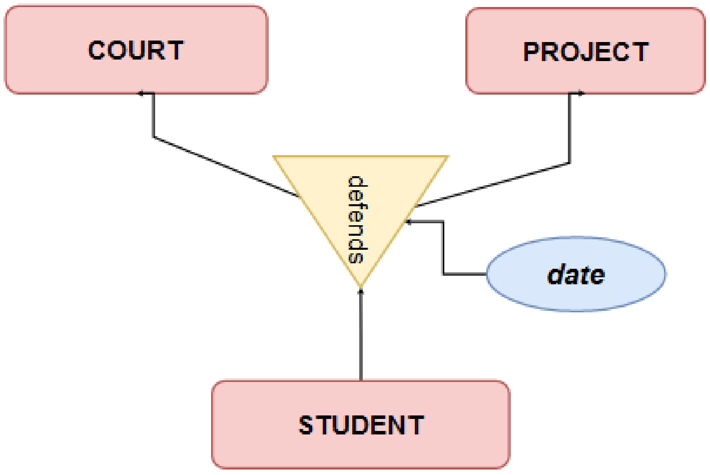
HEALTH_CENTER (idCenter, …)
PK: idCenter

TREATMENT (idTreatment, idCenter, idPhysio, date)
PK: idTreatment
FK: idCenter → HEALTH_CENTER (NOT NULL)
FK: idPhysio → PHYSIOTHERAPIST (NOT NULL)

1:1:1 cardinality

The primary key of the new table will be formed by any two of the three related entities, so we will have three equally valid options.

***Example**: "A student defends a project in front of a court (tribunal)".*



COURT (<u>idCourt</u>, …)
PK: idCourt

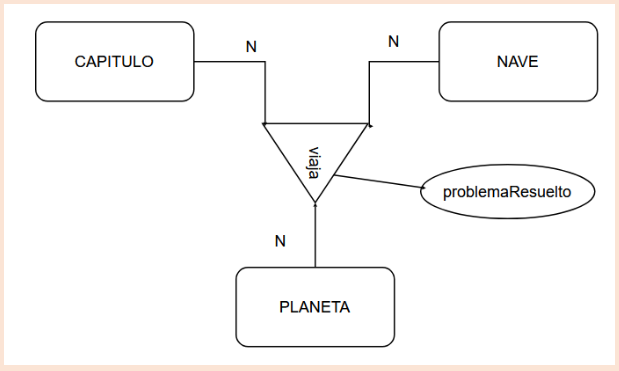
PROJECT (<u>idProject</u>, …)
PK: idProject

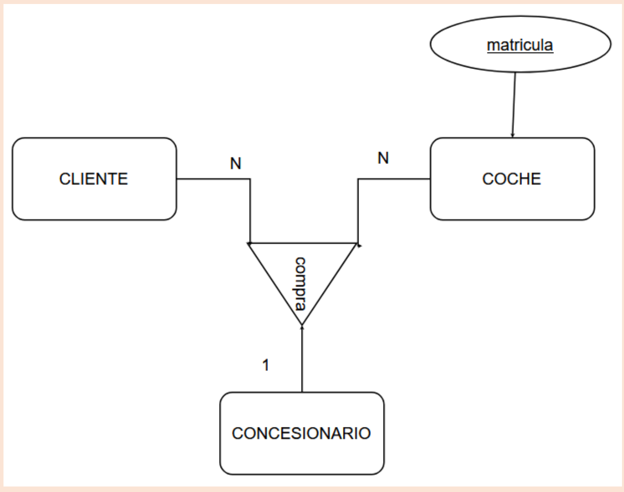
STUDENT (<u>idStudent</u>, …)
PK: idStudent

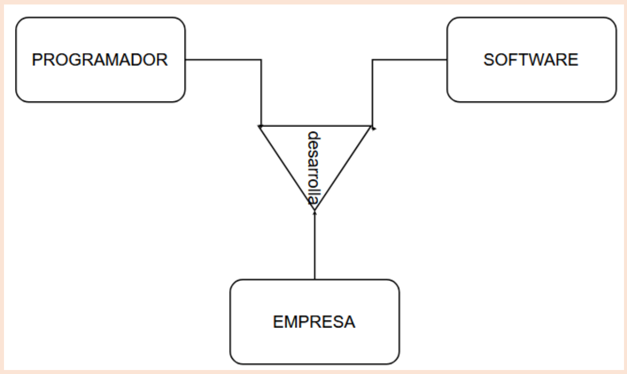
DEFENCE (idCourt, idProject, idStudent, date)
PK: idCourt
AK: idCourt, idProject
AK: idCourt, idStudent
FK: idCourt → COURT
FK: idProject → PROJECT
FK: idStudent → STUDENT

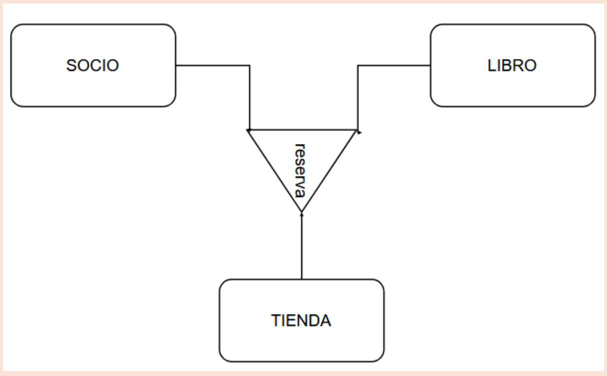
***NOTE**: One of them will be the PK of the table. The other two, will be compound alternative keys two by two*

Exercises to practice the cardinality of ternary relationships.



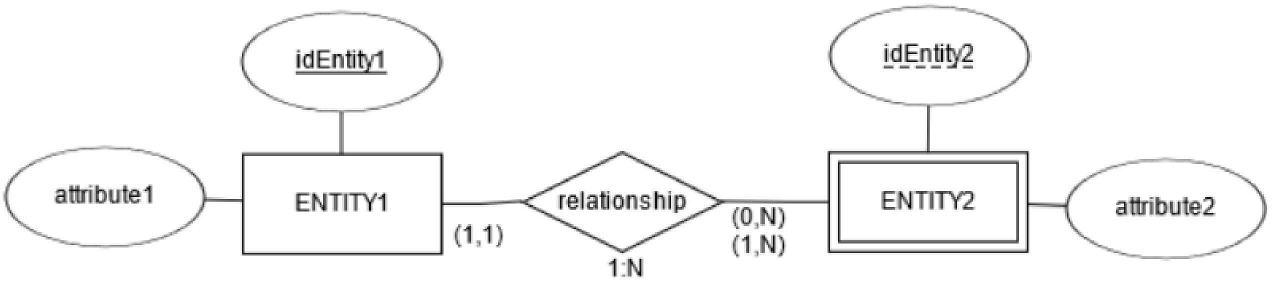






2.5 TRANSFORMATION OF WEAK ENTITIES

Every weak entity incorporates an implicit relationship with a strong entity, and this relationship does not need to be incorporated as a table in the relational model. Furthermore, normally that foreign key is part of the primary key of the table that represents the weak entity. The cardinality of the weak entity will always be (1,1) on the strong entity side and (0,N) or (1,N) on the weak entity side.

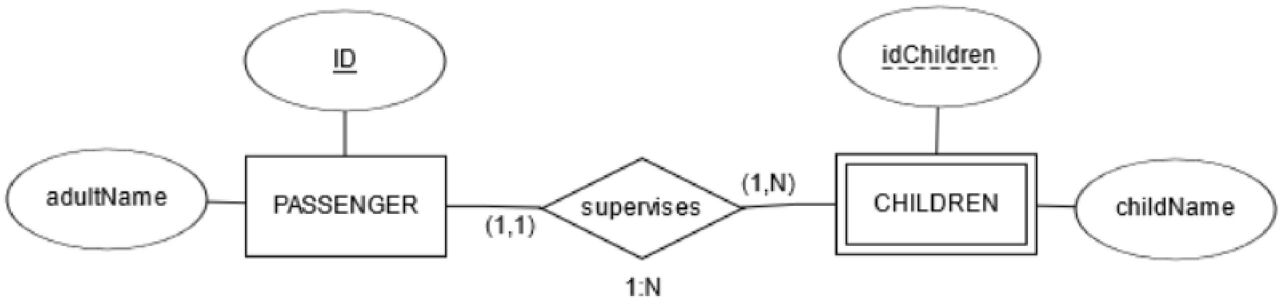


The transformation would be that the primary key of the strong entity would be propagated to the weak entity, but, in addition, the primary key of the weak entity would be the combination of the PK of the strong entity + the PK of the weak entity.

ENTITY1 (<u>idEntity1</u>, attribute1)
PK: idEntity1

ENTITY2 (<u>idEntity1, idEntity2</u>, attribute2)
PK: idEntity1, idEntity2
FK: idEntity1 → ENTITY1

Example.



PASSENGER (<u>DNI</u>, adultName)
PK: DNI

CHILDREN (<u>idChildren, DNI</u>, childName)
PK: idChildren, DNI
FK: DNI → PASSENGER

Table PASSENGER.

DNI (PK)	adultName
111A	Adulto 1
222B	Adulto 2
…	…

Table CHILDREN.

idChildren (PK)	DNI (PK)	childName
1	111A	Menor 1
2	111A	Menor 2
3	222B	Menor 3
4	111A	Menor 4
…	…	…

2.6 TRANSFORMATION OF GENERALISATIONS

Types and subtypes are not objects that can be represented in the standard relational model. There are several possibilities for its transformation, such as, for example:

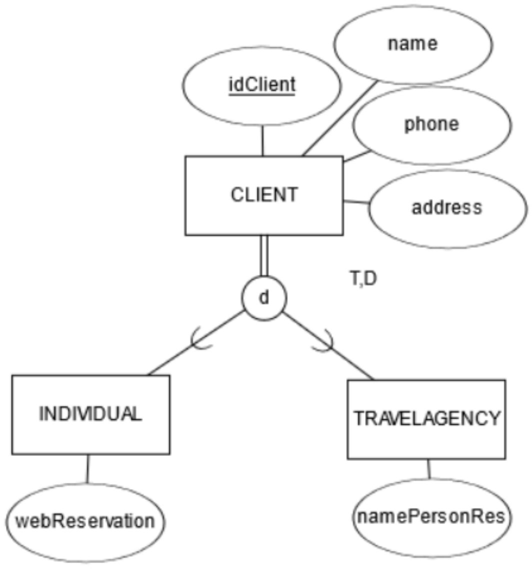
OPTION 1: Create a table for the SuperEntity, and as many tables as Subtypes exist.

Example.

CLIENTS (<u>idClient</u>, name, address, phone)
PK: idClient

INDIVIDUALS (<u>idClient</u>, webReservation)
PK: idClient
FK: idClient → CLIENTS

TRAVELAGENCIES (<u>idClient</u>, namePersonRes)
PK: <u>idClient</u>
FK: <u>idClient</u> → CLIENTS



OPCIÓN 2: Utilizar una única tabla CLIENTE y no crear tablas para las subentidades. Esto solo es recomendable hacerlo a nivel laboral cuando las subentidades tengan muy pocos atributos y sepamos que no van a haber más atributos. Esta decisión debe ser consensuada previamente con el analista/responsable del proyecto y a **nivel académico siempre utilizaremos la opción 1 (crear una tabla para cada subentidad)**.

CLIENTES (<u>codCliente</u>, nombre, dirección, teléfono, partic_webreserva, agencia_nombPerReserva)
PK: codCliente

De esta manera, si almacenamos un cliente que sea un particular el atributo agencia_nombPerReserva estará vacío (NULL) y por otra parte si el cliente es agencia el atributo partic_webreserva estará vacío. Puede parecer que el diseño en una única tabla no es óptimo, pero, todo lo contrario, **<u>las consultas SELECT serán mucho más sencillas y también más eficientes cuando la generalización no sea de muchas subentidades y estas no tengan muchos atributos</u>**.

3. Normalization.

Standardization is a technique that seeks to give efficiency and reliability to a relational DB. Its objective is, on the one hand, to bring information to a structure where the use of space prevails; and on the other hand, that the handling of information can be carried out quickly.

When we make a design in the relational model there are different alternatives, being able to obtain different relational schemas. Not all of them will be equivalent and some will represent the information better than others.

The tables obtained can present problems:

- **Redundancy**. Data that is repeated continuously and unnecessarily by the tables of the databases. When it is excessive, it is evident that the design must be reviewed, it is the first symptom of problems and it is easily detected.

- **Ambiguities**. Data that do not sufficiently clarify the registry they represent. The data in each record may refer to more than one record or it may even be impossible to know which item they are referring to.

- **Loss of integrity constraints**. Usually due to functional dependencies. This problem is explained later.

- **Anomalies in data modification operations**. The fact that when inserting a single element it is necessary to repeat records in a table to vary a few data. Or that deleting an element necessarily means deleting several records (*for example, deleting a customer means deleting six or seven rows from the customer table, that would be a very serious error and therefore a terrible design*).

3.1 NORMAL FORMS

The normal forms correspond to a normalization theory initiated by Edgar F. Codd and continued by other authors (among which Boyce and Fagin stand out). Codd defined the first normal form in 1970. From that moment the second, third, Boyce-Codd, fourth and fifth normal forms appeared. However, to consider a normalized model, it will be enough to reach the 3rd normal form, since the effort invested in passing it to successive normal forms does not result in a better design in many cases.

First normal form (1FN) → multivalued attributes

It is a normal form inherent to the relational scheme, so its compliance is mandatory; that is, every truly relational table complies with it. A table is said to be in first normal form if it prevents an attribute of a record from taking more than one value. **The following relationship is not in first normal form**:

WORKER		
idWorker	**name**	**phone**
1	Elías	965111111
2	David	965222222 631123123

To solve this problem we will have to detect the field that is being repeated (phone) and treat it as a multivalued attribute. Let us remember that the transformation of multivalued attributes to a relational model was carried out using a separate table:

WORKER	
idWorker	**name**
1	Elías
2	David

PHONE_WORKER	
idWorker	**phone**
1	965111111
2	965222222
2	631123123

WORKER (idWorker, name)

PK: idWorker

PHONE_WORKER (idWorker, phone)

PK: idWorker, phone

FK: idWorker → WORKER

Second normal form (2FN) → non-key attributes that do not match

The table must comply with the first normal form (1FN) and also if any attribute that is NOT a key depends functionally and completely on said key. That is, we must analyze that the dependencies of the attributes are logical with what the table stores (data fields related to each other are stored together).

SUBJECT		
codSubject	**name**	**hours**
0483	Sistemas informáticos	160
0484	Bases de datos	160
0485	Programación	256

STUDENT			
NIA	**codSubject**	**studentName**	**mark**
11111111	0484	Chewbakka	10
11111111	0485	Chewbakka	10
22222222	0484	Han Solo	5
22222222	0485	Han Solo	
33333333	0483	Luke Skywalker	7
33333333	0484	Luke Skywalker	
33333333	0485	Luke Skywalker	6

Provided the NIA and course code form the primary key for this table, only the grade has full functional dependency. The "name" depends completely only on the NIA, so the name must be taken to another table that has only the NIA as PK. The table does not satisfy second normal form (it is not 2FN). To fix it we split the original table into two tables:

STUDENT	
NIA	**name**
11111111	Chewbakka
22222222	Han Solo
33333333	Luke Skywalker

SUBJECT-MARK		
NIA	**codSubject**	**mark**
11111111	0484	10
11111111	0485	10
22222222	0484	5
22222222	0485	
33333333	0483	7
33333333	0484	
33333333	0485	6

Normalized relational model

SUBJECT (idSubject, name, hours)

PK: idSubject

STUDENT (NIA, name)

PK: NIA

MARK_SUBJECT (NIA, codSubject, mark)

PK: NIA, codSubject

FK: NIA → STUDENT

FK: codSubject → SUBJECT

Third normal form (3FN)

Occurs when a table is in second normal form (2FN) and also no non-key attribute functionally depends transitively (that a change in a specific record can affect the data consistency of the rest of the records) of the key primary. Let's see it with an example:

STUDENT			
NIA	**nameStudent**	**codProvince**	**nameProvince**
11111111	Chewbakka	11	Cádiz
22222222	Han Solo	41	Sevilla
33333333	Luke Skywalker	29	Málaga
44444444	Darth Vader	11	Cádiz
55555555	Yoda	08	Madrid

What happens if we update the name of the province with code 11 for Darth Vader to 'Estrella de la Muerte'? Evaluate whether the resulting data would be consistent or not.

STUDENT			
NIA	**nameStudent**	**codProvince**	**nameProvince**
11111111	Chewbakka	11	Cádiz
22222222	Han Solo	41	Sevilla
33333333	Luke Skywalker	29	Málaga
44444444	Darth Vader	11	Estrella de la Muerte
55555555	Yoda	08	Madrid

To prevent these inconsistencies from occurring, we will separate the student data into two tables, on the one hand, and the province data on the other:

STUDENT		
NIA	**name**	**codProvince**
11111111	Chewbakka	11
22222222	Han Solo	41
33333333	Luke Skywalker	29
44444444	Darth Vader	11
55555555	Yoda	08

PROVINCE	
codProvince	**name**
11	Cádiz
41	Sevilla
29	Málaga
11	Cádiz
08	Madrid

Normalized relational model

PROVINCIA (codProvince, name)

PK**:** codProvince

STUDENT (NIA, name, codProvince)

PK: NIA

FK: codProvince → PROVINCE

Conclusion: With this change, if we update a student's province, we will modify their code, but we will not change the name of the province in any case. If we wanted to do it, we would modify it for ALL students.

4. Relational algebra.

It is the mathematical system that **DBMS** use to obtain records from tables. The language that relational algebra uses in **DBMs** is **SQL** (Structured*Query Language)*,however, it is a simple language, very similar to English that facilitates the design of queries.

4.1 Unary operations *(on the same table*)

They are those operations of relational algebra that only affect one relationship. Two operations stand out, the selection and the projection.

Operation "Select"

Creates a new relationship from another, but including only some of the tuples from a given criterion. The criteria is based on constraints on the attributes of the relation R, and no other relationships can be included in the criteria that are not in R. To represent it we can use both the symbol σ or just the letter S.

Syntax: S (Table, condition)

the following expressions are equivalent:

S (R, A3>16) $\sigma_{A3>16}$ (R)	S (R, A3>16 AND A3<45) σ A3>16 AND A3 < 45 (R)	S (R, nombre='Carlos' OR edad=45) σ nombre='Carlos' and edad=45 (R)

Example. "Movie" table.

Title	Year	Duration	Type	Studio
Star Wars	1977	124	color	Fox
Mighty Ducks	1991	104	color	Disney
Wayne's World	1992	95	color	Paramount

S (MOVIE, duration>=100)

Title	Year	Duration	Type	Studio
Star Wars	1977	124	color	Fox
Mighty Ducks	1991	104	color	Disney

S (MOVIE, duration >=100 AND Studio='Fox')

Title	Year	Duration	Type	Studio
Star Wars	1977	124	color	Fox

Operation "Projection"

Instead of selecting records as in the previous case, attributes/columns are selected. To represent it we can use the symbol π or the letter P.

Syntax: P ((col1, col2, …colN), S (Table, condition))

Receives two parameters (columns and a table, which in turn can be a selection)

Continuing with the previous example, we have:
P ((title, studio), S (MOVIE, duration >= 100))

Title	Studio
Star Wars	Fox
Mighty Ducks	Disney

We could also have written it as follows:

πtitle,studio(σduration>=100 (MOVIE))

4.2 Binary operations

They are those **operations** of the relational algebra that will affect **two relations**. It is important to emphasize that in the union and in the difference, the number of columns in the two relationships must be the same. On the other hand, in the cartesian product it would not be necessary.

The examples described below for each operation will be given by the following relationships:

Table R.

Name	Address	Gender	BirthDate
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

Table S.

Name	Address	Gender	BirthDate
Harrison Ford	789 Palm Dr.	M	7/7/77
Carrie Fisher	123 Maple St.	F	9/9/99

Binary operation "Union"

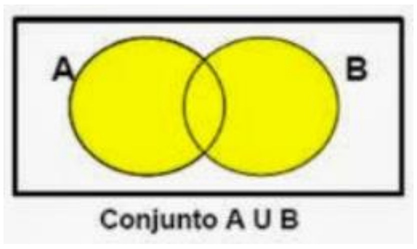
R ∪S, the union of R and S defines the set of elements that are in **R, in S, or in both**. **An element only appears once, so <u>there will be no repeated tuples.</u>**

Requirements to be able to use it:

- R and S must have identical schemas
- The column order must be the same

Result of the operation: R ∪S.

Name	Address	Gender	BirthDate
Carrie Fisher	123 Maple St.	F	9/9/99
Harrison Ford	789 Palm Dr.	M	7/7/77
Mark Hamill	456 Oak Rd.	M	8/8/88



Binary operation "Intersection"

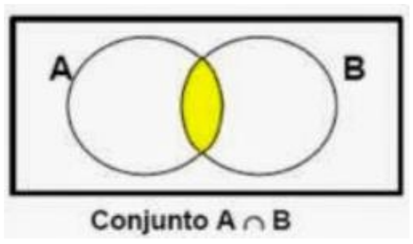
R ∩S, defines the set of elements that appear in both the relation R and the relation S simultaneously.

Requirements to be able to use it:

- R and S must have identical schemas
- The column order must be the same

Result of the operation: R ∩S.

Name	Address	Gender	BirthDate
Carrie Fisher	123 Maple St.	F	9/9/99



Binary operation *"Difference"*

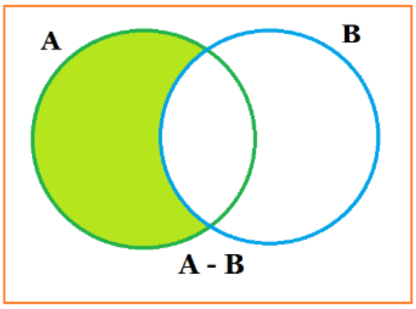
R – S or also called difference of R and S, defines the set of elements that are in R but not in S. Note that R - S is different from S - R.

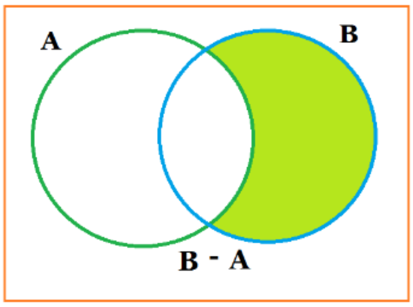
Requirements to be able to use it:

- R and S must have identical schemas
- The column order must be the same

Result of the operation R - S.

Name	Address	Gender	BirthDate
Mark Hamill	456 Oak Rd.	M	8/8/88





Binary operation *"Cartesian product"*

R x S, the schemas of both relationships are mixed and joined. <u>R and S do not need to have identical schemas</u>, nor is the order of the columns in both relationships important.

The result of the operation.

A	B
1	2
3	4

R

B	C	D
2	5	6
4	7	8
9	10	11

S

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

R X S

Binary operation NATURAL JOIN

It is a cartesian product where we will be interested only in the records that match one table and the other, linking by the columns that have the same name in table A and table B.

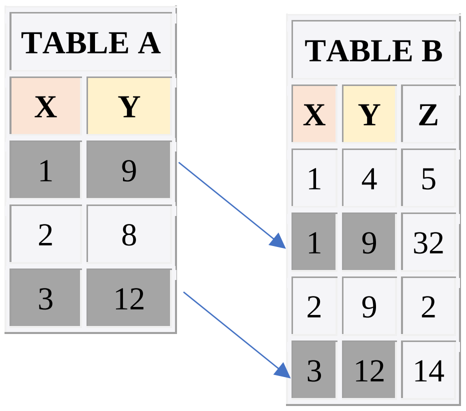


TABLE A	
X	**Y**
1	9
2	8
3	12

TABLE B		
X	**Y**	**Z**
1	4	5
1	9	32
2	9	2
3	12	14

NATURAL JOIN OF A and B		
X	**Y**	**Z**
1	9	32
3	12	14

Records selected match that:
- Column X in table A is equal to column X in table B
- And, in addition, column Y in table A is equal to column Y in table B

In SQL it would be represented as follows:

```
SELECT X, Y, Z
  FROM A, B
 WHERE A.X = B.X
   AND A.Y = B.Y;
```