BD
Bases de Datos

UD 3
El modelo lógico relacional

UD3 – El modelo lógico relacional

ÍNDICE

1. El modelo lógico relacional

El modelo entidad-relación es un modelo conceptual que sirve para cualquier tipo de SGBD, en cambio, el modelo relacional es un modelo lógico que sólo sirve para SGBD relacionales.

Todos los diseñadores y administradores de bases de datos relacionales usan esquemas conceptuales entidad-relación porque se adaptan muy bien a este modelo.

Hay que tener en cuenta la diferencia de la palabra relación en ambos modelos. En el modelo relacional una relación es una tabla mientras que en el Entidad/Relación es la asociación que se produce entre dos entidades.

1.1 RELACIÓN (TABLA)

Según el modelo relacional el elemento fundamental es lo que se conoce como relación, aunque más habitualmente se le llama tabla. Se trata de una estructura formada por filas y columnas que almacena los datos referentes a una determinada entidad o relación del mundo real.

NOMBRE

atributo 1	atributo 2	atributo 3	....	atributo n	
valor 1,1	valor 1,2	valor 1,3	....	valor 1,n	← tupla 1
valor 2,1	valor 2,2	valor 2,3	....	valor 2,n	← tupla 2
.....	.....	.....	....	.....	....
valor m,1	valor m,2	valor m,3	....	valor m,n	← tupla m

Acerca de una tabla, además de su nombre, podemos distinguir lo siguiente:

Atributo

Representa una propiedad que posee esa tabla. Equivale al atributo del modelo E-R. Se corresponde con la idea de campo o columna.

Tupla o Registro

Cada una de las filas de la tabla. Se corresponde con la idea de registro. Representa por tanto cada elemento individual (ejemplar, ocurrencia) de esa tabla.

Dominio

Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio. Un domino en realidad es un conjunto finito de valores del mismo tipo. Los dominios poseen un nombre para poder referirnos a él y así poder ser reutilizable en más de un atributo.

1.2 CLAVE

Clave candidata

Conjunto de atributos que identifican unívocamente cada registro de la relación. Es decir, columnas cuyos valores no se repiten para esa tabla. Las claves candidatas pueden ser:

- ***Clave primaria (PK o Primary Key)***

  Clave candidata que se escoge como identificador de los registros. Se elige como primaria la candidata que identifique mejor a cada registro en el contexto de la base de datos. *Por ejemplo, un campo con el DNI sería clave candidata de una tabla de clientes, aunque si en esa relación existe un campo de código de cliente, este sería mejor candidato para clave principal, porque es mejor identificador para ese contexto.*

- ***Clave alternativa (AK o Alternative Key)***

  Cualquier clave candidata que no sea primaria.

Clave ajena (FK o Foreign Key)

Atributo cuyos valores coinciden con una clave primaria de otra tabla.

1.3 RESTRICCIÓN

Una restricción **es una condición de obligado cumplimiento por los datos** de la base de datos. Las hay de varios tipos.

1. **Aquellas que son definidas por el hecho de que la base de datos sea relacional:**
   - No puede haber dos registros iguales
   - El orden de los registros no es significativo
   - El orden de los atributos no es significativo
   - Cada atributo sólo puede tomar un valor en el dominio en el que está inscrito

2. **Aquellas que son incorporadas por los usuarios:**

   - **Clave primaria (PRIMARY KEY)**

     Hace que los atributos marcados como clave primaria no puedan repetir valores. Además, obliga a que esos atributos no puedan estar vacíos. Si la clave primaria la forman varios atributos, ninguno de ellos podrá estar vacío.

- **Unicidad (UNIQUE)**
  Los valores de los atributos marcados de esa forma no pueden repetirse. Esta restricción debe indicarse en todas las claves alternativas.

- **Obligatoriedad (NOT NULL)**
  Prohíbe que el atributo marcado de esta forma no tenga ningún valor (es decir impide que pueda contener el valor nulo, NULL).

- **Restricción arbitraria (CHECK)**
  Permite introducir restricciones relacionadas con el dominio de los atributos. Ejemplos. Podemos introducir restricciones CHECK sobre los campos que afecten a:
  - Precio mínimo y máximo de un libro 0€ y 50€
  - Fecha préstamo libro no puede ser anterior a la fecha del día
  - Precio de venta de una casa no puede ser menor al precio de compra

- **Integridad referencial (FOREIGN KEY)**
  Sirve para indicar una clave externa. Cuando una clave se marca con integridad referencial, no se podrán introducir valores que no estén incluidos en los campos relacionados con esa clave.

Esto último, **la integridad referencial, causa problemas en las operaciones de borrado y modificación de registros**, ya que si se ejecutan esas operaciones sobre la tabla principal quedarán filas en la tabla secundaria con la clave externa sin integridad.

Esto se puede manipular agregando las siguientes cláusulas:

- **CASCADE**: propagar el borrado de los registros afectados o modificación de una clave en una fila en la tabla referenciada.

- **SET NULL**: asignar el valor NULL a las claves foráneas con el mismo valor.

- **NO ACTION**: las claves ajenas no se modifican, ni se eliminan filas en la tabla que las contiene (aplazado), el SGBD devuelve error de integridad.

- **SET DEFAULT**: implica asignar el valor por defecto a las claves foráneas con el mismo valor.

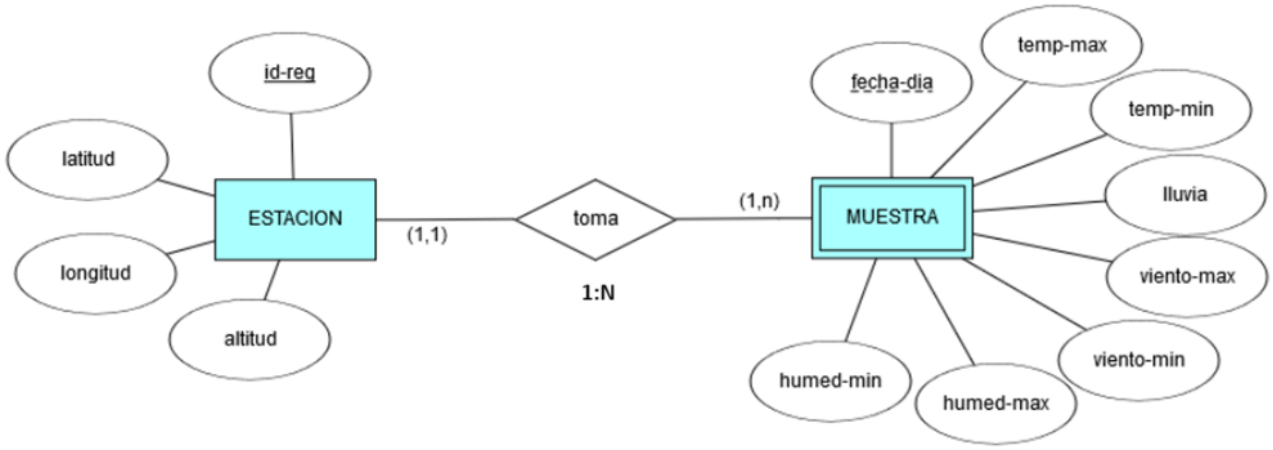
1.4 ESQUEMA LÓGICO RELACIONAL

Una entidad, en el esquema relacional, se define de la siguiente forma:

<nombre_entidad> (<atributo 1>, <atributo 2>, ...)

donde el atributo clave principal aparece subrayado y donde también se señala, de alguna forma, cuáles son claves foráneas (por ejemplo, indicándolo con CA o FK).

Por ejemplo, el esquema conceptual del siguiente diagrama ER se expresa con los siguientes esquemas relacionales:



ESTACIONES (id_reg, longitud, latitud, altitud)

PK: id_reg

MUESTRAS (id_reg, fecha_dia, temp_min, temp_max, lluvia, viento_min, viento_max, humed_min, humed_max)

PK: id_reg, fecha_dia
FK: id_reg → ESTACION

2. Paso de Entidad-Relación al modelo lógico relacional

Previo a la aplicación de las reglas de transformación de esquemas entidad- relación a esquemas relacionales es conveniente la preparación de los esquemas entidad-relación mediante la aplicación de unas reglas que faciliten y garanticen la fiabilidad del proceso de transformación.

Estas reglas preparatorias se basan en la aplicación de la 1FN (*1ª Forma Normal, la cual veremos más adelante en el tema*) y su objetivo es eliminar las siguientes anomalías:

- Atributos con valores múltiples (multivaluados)
- Atributos compuestos

2.1 ELIMINACIÓN DE ATRIBUTOS MULTIVALUADOS

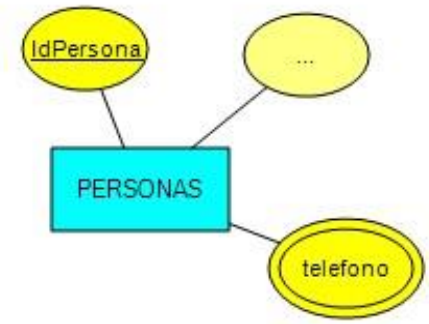
Todos los atributos múltiples se deben transformar en un tipo de entidad débil por existencia con una relación de muchos a muchos o de uno a muchos, según sea el caso, con el tipo de entidad sobre el cual estaba definido. Si se considera que la nueva entidad creada resulta ambigua, se le pueden añadir atributos o heredarlos de la otra entidad.

Suponemos para el siguiente ejemplo que una persona puede tener varios números de teléfono.

En este caso sería necesario expresar el esquema relacional de la forma:

```
PERSONAS (idPersona, ... atributos ...)
PK: idPersona

TELEFONOS (idPersona, telefono)
PK:  idpersona, telefono
FK: idpersona → PERSONAS
```

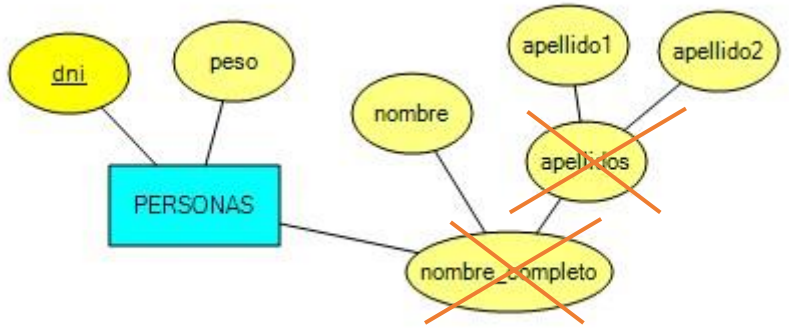


Es decir, habría que crear una tabla TELEFONOS en la que guardar los números de teléfono asociados a las PERSONAS (por eso tiene una clave ajena idpersona a la tabla PERSONAS).

2.2 ELIMINACIÓN DE ATRIBUTOS COMPUESTOS

Todos los atributos compuestos deben ser descompuestos en atributos simples que quedan asociados a la misma entidad.

El esquema entidad-relación:



Se expresaría mediante el siguiente esquema relacional:

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

PK: DNI
```

Como puede observarse, nos hemos quedado con los atributos terminales (aquellos que no tienen hijos) y que por tanto son simples y podemos representarlos en una tabla.

2.3 TRANSFORMACIÓN DE LAS ENTIDADES FUERTES

En principio las entidades fuertes del modelo E-R son transformadas al modelo relacional siguiendo estas instrucciones:

- **Entidades**. Las entidades pasan a ser tablas.
- **Atributos**. Los atributos pasan a ser columnas.
- **Identificadores principales**. Pasan a ser claves primarias.
- **Identificadores candidatos**. Pasan a ser claves candidatas.

Esto hace que la transformación se produzca según este ejemplo:

ENTIDAD (identificador, atributo1, atributo2, atributo3)

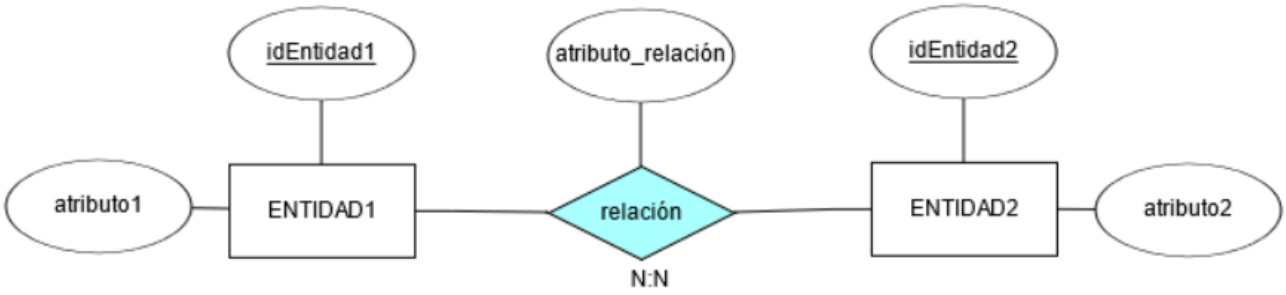
PK: identificador

2.4 TRANSFORMACIÓN DE RELACIONES

La idea inicial es transformar cada relación en una tabla, pero hay que distinguir según el tipo de relación.

Relaciones muchos a muchos

En las relaciones varios a varios la relación se transforma en una tabla cuyos atributos son: los atributos de la relación y las claves de las entidades relacionadas (que pasarán a ser claves externas). La clave de la tabla la forman todas las claves externas.

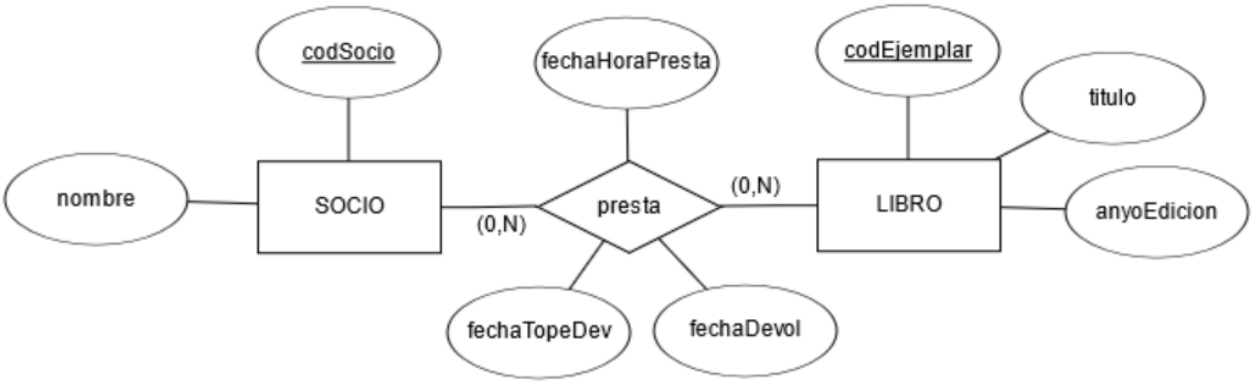


ENTIDAD1 (identificador1, atributo1)
PK: identificador1

ENTIDAD2 (identificador2, atributo2)
PK: identificador2

RELACION (identificador1, identificador2, atributo_relacion)
PK: identificador1, identificador2
FK: identificador1 → ENTIDAD1
FK: identificador2 → ENTIDAD2

Ejemplo. Transformar el siguiente diagrama E-R:



Aplicando las reglas, tendríamos el siguiente modelo relacional:

SOCIOS (codSocio, nombre)
PK: codSocio

LIBROS (codEjemplar, titulo, anyoEdicion)
PK: codEjemplar

PRESTAMOS (codSocio, codEjemplar, fechaHoraPrestamo, fechaTopDev, fechaDevol)
PK: codSocio, codEjemplar
FK: codSocio → SOCIOS
FK: codEjemplar → LIBROS

Pero, entonces, imagina que un Socio (Carlos), que ya ha reservado un Libro anteriormente (BD para Pros) necesita volver a reservarlo … ¿podría hacerlo?

La respuesta es sencillamente NO. La PK es codSocio y codEjemplar, por lo que si volvemos a tomar prestado el libro se repetiría. Ante esta situación tenemos dos posibilidades:

1. Utilizar una **clave compuesta con otro atributo** que consiga que no se repita la tupla

2. Utilizar una **clave "artificial"** (*normalmente id o cod + nombre relación N:N*) que haga que nunca se repita y que las PKs de ambos lados de la relación sean tan solo FKs que siempre tengan valor (NOT NULL).

OPCION DE DISEÑO 1 (ampliar la PK con otro atributo)

SOCIOS (codSocio, nombre)
PK: codSocio

LIBROS (codEjemplar, titulo, anyoEdicion)
PK: codEjemplar

PRESTAMOS (codSocio, codEjemplar, fechaHoraPrestamo, fechaTopDev, fechaDevol)
PK: codSocio, codEjemplar, fechaHoraPrestamo
FK: codSocio → SOCIOS
FK: codEjemplar → LIBROS

OPCION DE DISEÑO 2 (utilizar una PK "artificial")

SOCIOS (codSocio, nombre)
PK: codSocio

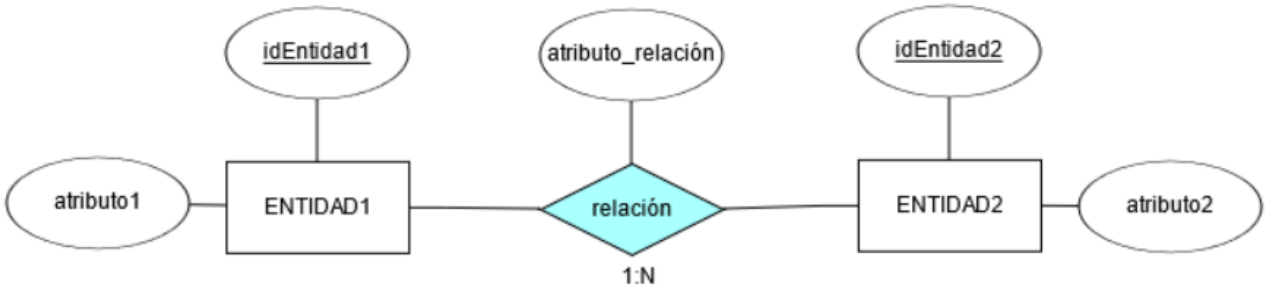
LIBROS (codEjemplar, titulo, anyoEdicion)
PK: codEjemplar

PRESTAMOS (codPrestamo, codSocio, codEjemplar, fechaHoraPrestamo, fechaTopDev, fechaDevol)
PK: codPrestamo
FK: codSocio → SOCIOS (NOT NULL)
FK: codEjemplar → LIBROS (NOT NULL)

Nota importante: Hay relaciones que se prestan a esta técnica y otras que no. Para diferenciarlas debe considerarse si la acción de la relación puede ejecutarse más de una vez y si hacer esto es.

Relaciones uno a muchos

Las relaciones de tipo "uno a varios" no requieren ser transformadas en una tabla en el modelo relacional. **La tabla del lado varios incluye como clave ajena el identificador de la entidad del lado uno**. En el caso de que el número mínimo de la relación sea de cero (puede haber ejemplares de la entidad uno sin relacionar), se deberá permitir valores nulos en la clave externa identificador1. En otro caso no se podrán permitir (ya que siempre habrá un valor relacionado).



ENTIDAD1 (identificador1, atributo1)
PK: identificador1

ENTIDAD2 (identificador2, atributo2, identificador1, atributo_relacion)
PK: identificador2
FK: identificador1 → ENTIDAD1

Podemos mejorar el diseño teniendo en cuenta que:

Si la cardinalidad mínima del lado 1 fuera 1:

- Indicaremos que la clave ajena que pasa al lado N debe cumplir con la restricción NOT NULL.

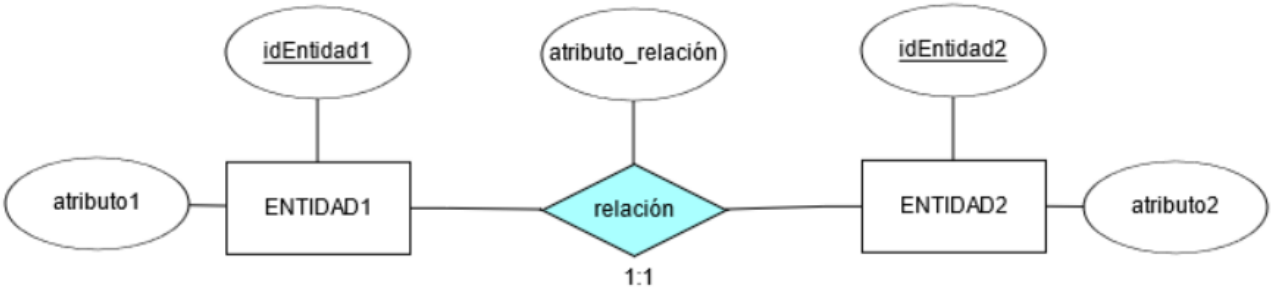
Ejemplos.

- Cliente <compra> Coche (matriculado y único)

- Un libro se incluye en una sola categoría y una categoría engloba uno o muchos libros

Relaciones uno a uno

En el caso de las relaciones uno a uno, ocurre lo mismo: la relación no se convierte en tabla, sino que se coloca en una de las tablas (**en principio daría igual en cuál**) el identificador de la entidad relacionada como clave ajena. En el caso de que una entidad participe opcionalmente en la relación, entonces es el identificador de ésta el que se colocará como clave externa en la tabla que representa a la otra entidad.



Opción 1

ENTIDAD1 (identificador1, atributo1)
PK: identificador1

ENTIDAD2 (identificador2, atributo2, identificador1, atributo_relacion)
PK: identificador2
FK: identificador1 → ENTIDAD1

Opción 2

ENTIDAD1 (identificador1, atributo1, identificador2, atributo_relacion)
PK: identificador1
FK: identificador2 → ENTIDAD2

ENTIDAD2 (identificador2, atributo2)
PK: identificador2

NOTA DE DISEÑO
Puesto que la cardinalidad máxima de ambos lados es la misma, deberemos fijarnos en la cardinalidad mínima. Ante cardinalidades mínimas iguales (0,0) o (1,1) podremos propagar la clave hacia cualquiera de los dos lados de la relación (pero solo a un lado).
En caso de que uno de los lados tenga cardinalidad mínima 1 y el otro 0, propagaremos la clave del lado 1 al lado 0.

Relaciones reflexivas

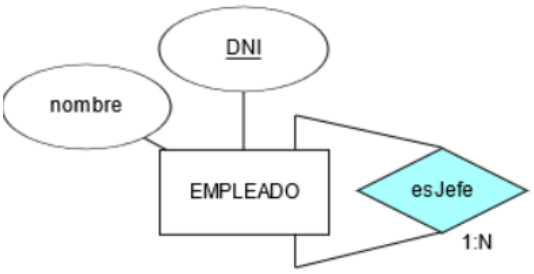
Las relaciones reflexivas se tratan de la misma forma que las otras, sólo que un mismo atributo puede figurar dos veces en una tabla como resultado de la transformación.

Proceso de transformación:

- **Uno a Muchos (1:N)**: se incluye la clave principal como clave ajena en la misma tabla.
- **Muchos a Muchos (N:N)**: se genera nueva tabla, igual que en las relaciones binarias.

UNO A MUCHOS

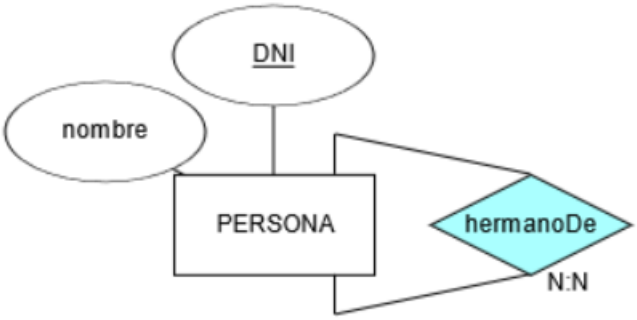
EMPLEADOS (DNI, nombre, **DNI_jefe**)
PK: DNI
FK: DNI_jefe → EMPLEADO



MUCHOS A MUCHOS

PERSONAS (DNI, nombre)
PK: DNI

HERMANOS (DNI1, DNI2)
PK: DNI1, DNI2
FK: DNI1 → PERSONA
FK: DNI2 → PERSONA



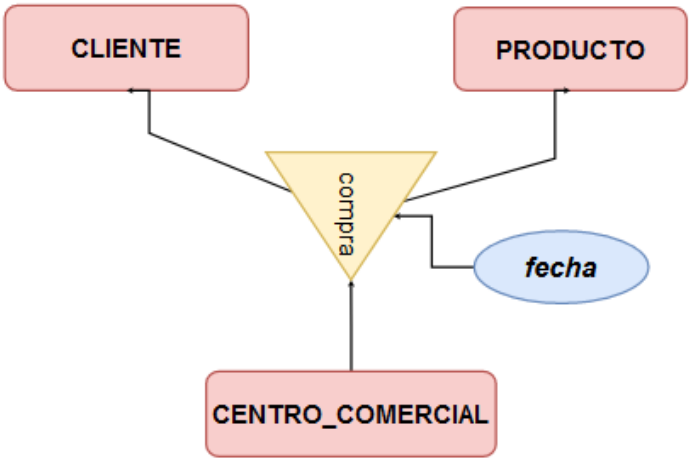
Relaciones ternarias

Las relaciones ternarias o n-arias (siendo n el número de entidades que participan en la relación) se tratan de modo similar a las binarias muchos a muchos. Veamos cada caso uno a uno:

Cardinalidad N:N:N

Creamos una nueva tabla. La clave primaria de la nueva tabla será la combinación de las tres claves principales de las entidades relacionadas.

***Ejemplo**: "Los clientes pueden comprar productos en centros comerciales"*



CLIENTES (idCliente, …)
PK: idCliente

PRODUCTOS (idProducto, …)
PK: idProducto

CENTROS_COMERCIALES (idCentro, …)
PK: idCentro

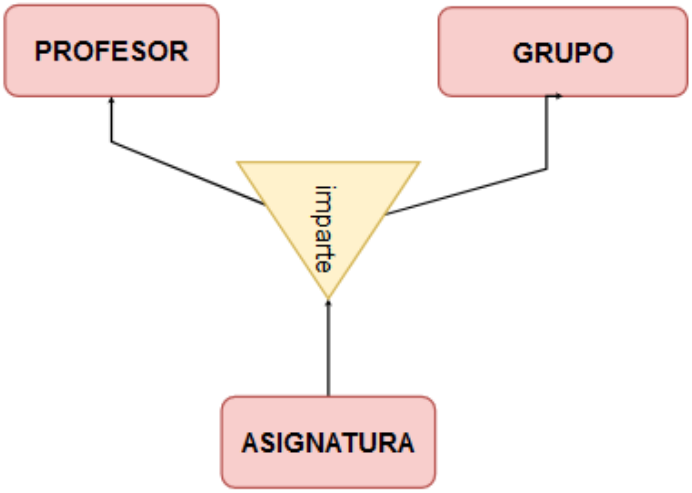
OPCION 1
COMPRAS (idCliente, idProducto, idCentro, fecha)
PK: idCliente, idProducto, idCentro, fecha
FK: idCliente → CLIENTES
FK: idProducto → PRODUCTOS
FK: idCentro → CENTROS_COMERCIALES

OPCION 2
COMPRAS (idCompra, idCliente, idProducto, idCentro, fecha)
PK: idCompra
FK: idCliente → CLIENTES
FK: idProducto → PRODUCTOS
FK: idCentro → CENTROS_COMERCIALES

Cardinalidad 1:N:N

Creamos una nueva tabla. La clave primaria de la nueva tabla será la combinación de las dos PK de las entidades relacionadas con cardinalidad N. La entidad con cardinalidad 1 tan solo será FK.

***Ejemplo**: "Un profesor imparte una o varias asignaturas a uno o varios grupos".*



PROFESORES (idProfesor, …)
PK: idProfesor

GRUPOS (idGrupo, …)
PK: idGrupo

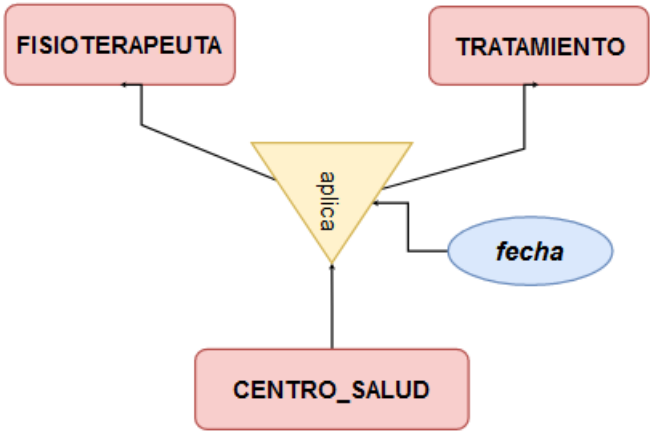
ASIGNATURAS (idAsignatura, …)
PK: idAsignatura

IMPARTICIONES (idAsignatura, idGrupo, idProfesor, …)
PK: idAsignatura, idGrupo
FK: idGrupo → GRUPOS
FK: idAsignatura → ASIGNATURAS
FK: idProfesor → PROFESORES (NOT NULL)

Cardinalidad 1:1:N

Creamos una nueva tabla. Tendremos dos claves candidatas, la PK del lado N y la PK de uno de los dos lados 1 y otra de la PK del lado N con el otro lado 1. Una será clave primaria y la otra alternativa.

Ejemplo. *"Un fisioterapeuta aplica uno o varios tratamientos en un centro de salud" (vamos a imponer esta restricción para obligar a tener 1:1:N).*



FISIOTERAPEUTAS (idFisio, …)
PK: idFisio

CENTROS_SALUD (idCentro, …)
PK: idCentro

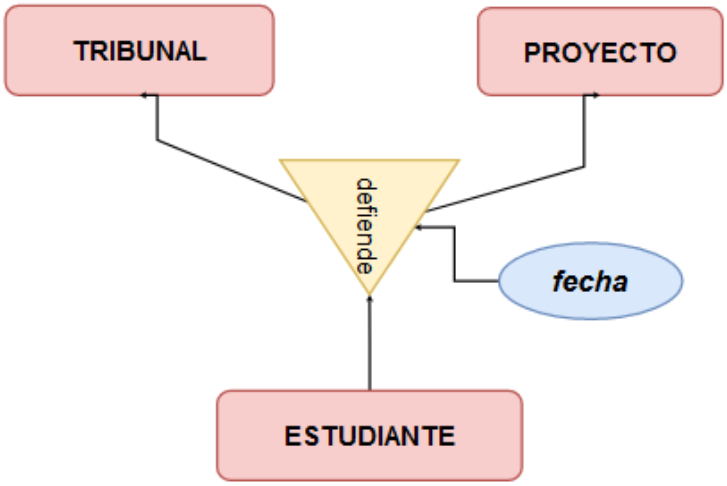
TRATAMIENTOS (idTratamiento)
PK: idTratamiento

APLICACIONES (idTratamiento, idCentro, idFisio, fecha)
PK: idTratamiento, idCentro
AK: idTratamiento, idFisio
FK: idTratamiento → TRATAMIENTOS
FK: idCentro → CENTROS_SALUD
FK: idFisio → FISIOTERAPEUTAS

Cardinalidad 1:1:1

Creamos una nueva tabla. La clave primaria de la nueva tabla estará formada por dos entidades cualesquiera de las tres relacionadas, por lo que tendremos tres opciones igualmente válidas.

***Ejemplo**: "Un/a estudiante defiende ante un tribunal un proyecto".*



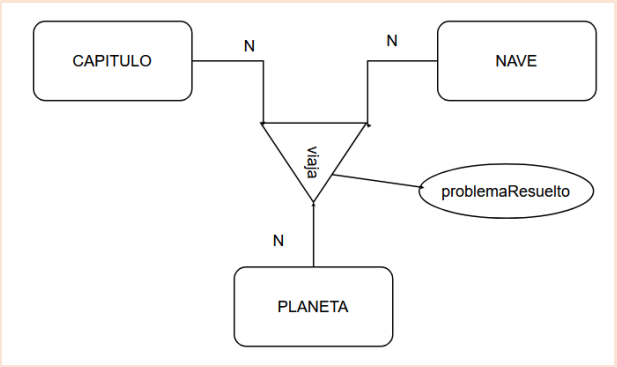
TRIBUNALES (idTribunal, …)
PK: idTribunal

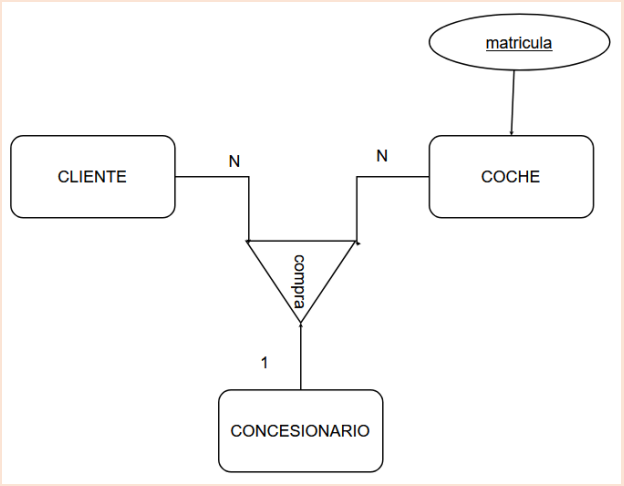
PROYECTOS (idProyecto, …)
PK: idProyecto

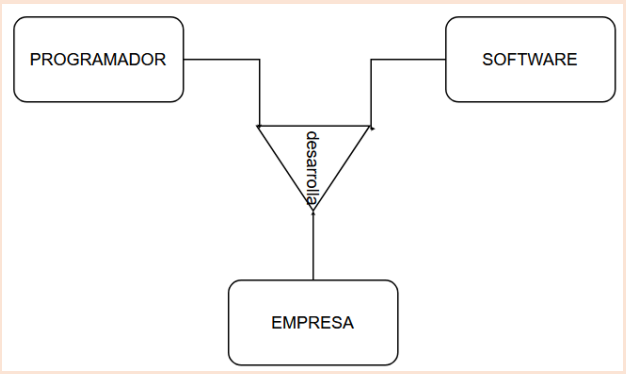
ESTUDIANTES (idEstudiante, …)
PK: idEstudiante

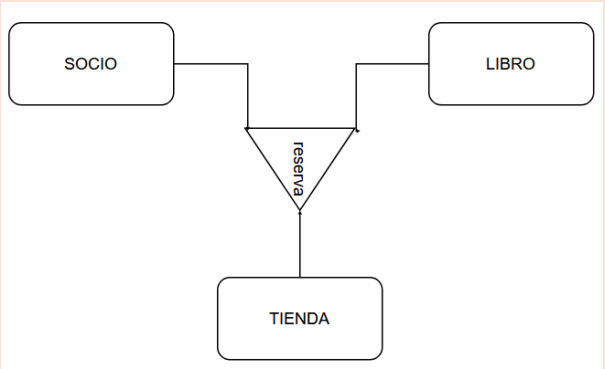
DEFENSA (idTribunal, idProyecto, idEstudiante, fecha)
PK: idTribunal, idProyecto
AK: idTribunal, idEstudiante
AK: idEstudiante, idProyecto
FK: idTribunal → TRIBUNALES
FK: idProyecto → PROYECTO S
FK: idEstudiante → ESTUDIANTES

Ejercicios para practicar la cardinalidad de las relaciones ternarias.



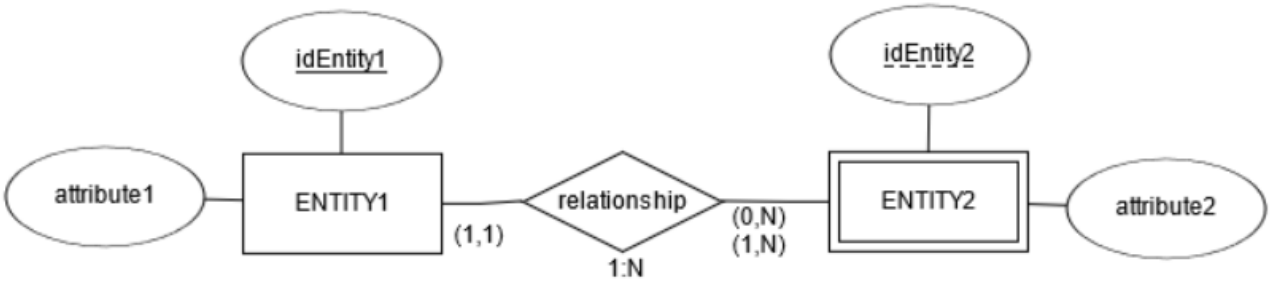






2.5 TRANSFORMACIÓN DE LAS ENTIDADES DÉBILES

Toda entidad débil incorpora una relación implícita con una entidad fuerte y esta relación no necesita incorporarse como tabla en el modelo relacional. Es más, normalmente esa clave ajena forma parte de la clave principal de la tabla que representa a la entidad débil. La cardinalidad de la entidad débil siempre será (1,1) en el lado de la entidad fuerte y (0,N) o (1,N) en el lado de la entidad débil.

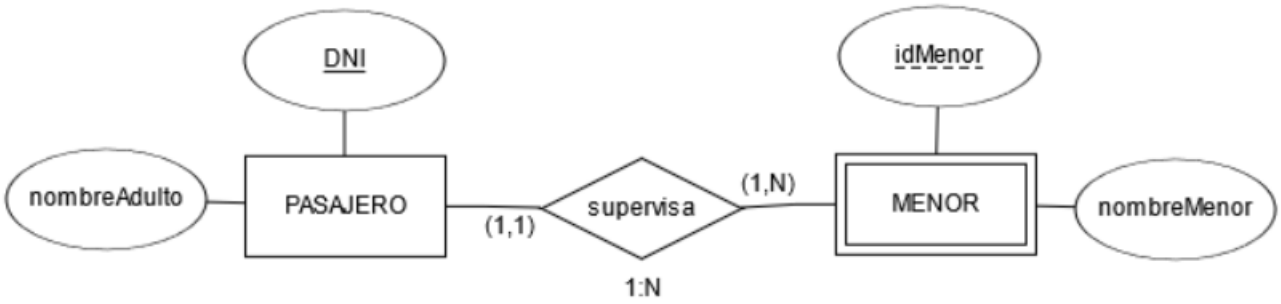


La transformación quedaría que se propagaría la clave principal de la entidad fuerte a la débil, pero, además, la clave principal de la entidad débil sería la combinación de la PK de la fuerte + la PK de la débil.

ENTITY1 (idEntity1, attribute1)
PK: idEntity1

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

Ejemplo.



PASAJERO (DNI, nombreAdulto)
PK: DNI

MENOR (idMenor, DNI, nombreMenor)
PK: idMenor, DNI
FK: DNI → PASAJERO

Tabla PASAJERO.

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

Tabla MENOR.

idMenor (PK)	DNI (PK)	nombre
1	111A	Menor 1
2	111A	Menor 2
3	222B	Menor 3
4	111A	Menor 4
…	…	…

2.6 TRANSFORMACIÓN DE LAS GENERALIZACIONES

Los tipos y subtipos no son objetos que se puedan representar en el modelo relacional estándar. Existen varias posibilidades para su transformación, como, por ejemplo:

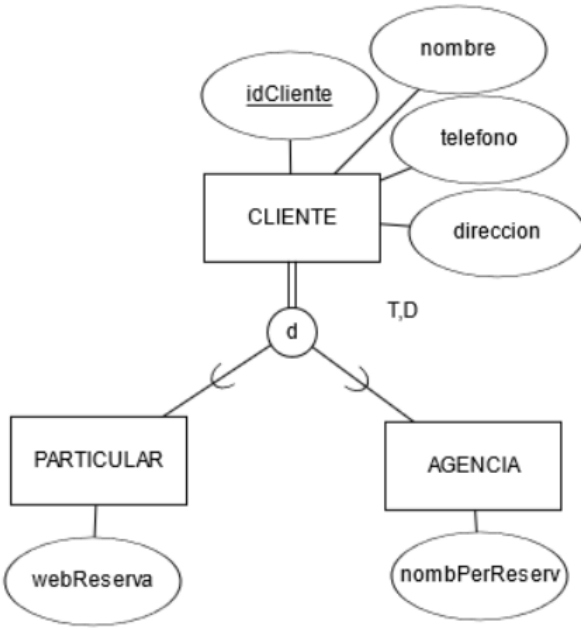
OPCIÓN 1: Crear una tabla para la superentidad, y tantas tablas como subentidades.

Ejemplo.

CLIENTES (idCliente, nombre, dirección, teléfono)
PK: idCliente

PARTICULARES (idCliente, webReserva)
PK: idCliente
FK: idCliente → CLIENTES

AGENCIAS (idCliente, nombPerReserva)
PK: idCliente
FK: idCliente → CLIENTES



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 (codCliente, 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, **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**.

3. Normalización

La normalización es una técnica que busca dar eficiencia y fiabilidad a una base de datos relacional. Su objetivo es, por un lado, llevar la información a una estructura donde prime el aprovechamiento del espacio; y por otro lado, que el manejo de información pueda llevarse a cabo de forma rápida.

Cuando realizamos un diseño en el modelo relacional existen diferentes alternativas, pudiéndose obtener diferentes esquemas relacionales. No todos ellos serán equivalentes y unos representarán mejor la información que otros.

Las tablas obtenidas pueden presentar problemas:

- **Redundancia**. Datos que se repiten continua e innecesariamente por las tablas de las bases de datos. Cuando es excesiva es evidente que el diseño hay que revisarlo, es el primer síntoma de problemas y se detecta fácilmente.

- **Ambigüedades**. Datos que no clarifican suficientemente el registro al que representan. Los datos de cada registro podrían referirse a más de un registro o incluso puede ser imposible saber a qué ejemplar se están refiriendo.

- **Pérdida de restricciones de integridad**. Normalmente debido a dependencias funcionales. Más adelante se explica este problema.

- **Anomalías en operaciones de modificación de datos**. El hecho de que al insertar un solo elemento haya que repetir registros en una tabla para variar unos pocos datos. O que eliminar un elemento suponga eliminar varios registros necesariamente (*por ejemplo, que eliminar un cliente suponga borrar seis o siete filas de la tabla de clientes, sería un error muy grave y por lo tanto un diseño terrible*).

3.1 FORMAS NORMALES

Las formas normales se corresponden a una teoría de normalización iniciada por Edgar F. Codd y continuada por otros autores (entre los que destacan Boyce y Fagin). Codd definió en 1970 la primera forma normal. Desde ese momento aparecieron la segunda, tercera, la Boyce-Codd, la cuarta y la quinta forma normal. No obstante, para considerar un modelo normalizado bastará con llegar a la 3ª forma normal, puesto que el esfuerzo que se invierte en pasarlo a sucesivas formas normales no revierte en un mejor diseño en muchos de los casos.

Primera forma normal (1FN) → atributos multivaluados

Es una forma normal inherente al esquema relacional, por lo que su cumplimiento es obligatorio; es decir toda tabla realmente relacional la cumple. Se dice que una tabla se encuentra en primera forma normal si impide que un atributo de un registro pueda tomar más de un valor. La siguiente relación **no cumple la primera forma normal**:

TRABAJADOR		
idTrabajador	**nombre**	**teléfono**
1	Elías	965111111
2	David	965222222 631123123

Para resolver este problema deberemos detectar el campo que se está repitiendo (teléfono) y tratarlo como un atributo multivaluado. Recordemos que la transformación de atributos multivaluados a modelo relacional se realizaba utilizando una tabla a parte:

TRABAJADOR	
idTrabajador	**nombre**
1	Elías
2	David

TLF_TRABAJADOR	
idTrabajador	**teléfono**
1	965111111
2	965222222
2	631123123

TRABAJADOR (idTrabajador, nombre)

PK: idTrabajador

TLF_TRABAJADOR (idTrabajador, teléfono)

PK: idTrabajador, teléfono

FK: idTrabajador → TRABAJADOR

Segunda forma normal (2FN) → atributos no clave que no correspondan

La tabla deberá cumplir con la primera forma normal (1FN) y además si cualquier atributo que NO sea clave depende funcionalmente y de forma completa de dicha clave. Es decir, deberemos analizar que las dependencias de los atributos sean lógicas con lo que la tabla almacena (los campos de datos relacionados entre sí se almacenan juntos).

MÓDULO		
codMódulo	**nombre**	**horas**
0483	Sistemas informáticos	160
0484	Bases de datos	160
0485	Programación	256

ESTUDIANTE			
NIA	**codMódulo**	**nombreEstudiante**	**nota**
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

Suponiendo que el NIA y el código de curso forman la clave principal para esta tabla, sólo la nota tiene dependencia funcional completa. El nombre depende de forma completa tan solo del NIA, por lo que el nombre deberemos llevarlo a otra tabla que tenga como PK sólo el NIA. La tabla no satisface la segunda forma normal (no es 2FN). Para arreglarlo separamos la tabla original en dos tablas:

ESTUDIANTE	
NIA	**nombre**
11111111	Chewbakka
22222222	Han Solo
33333333	Luke Skywalker

NOTA_MODULO		
NIA	**codMódulo**	**nota**
11111111	0484	10
11111111	0485	10
22222222	0484	5
22222222	0485	
33333333	0483	7
33333333	0484	
33333333	0485	6

Modelo relacional normalizado:

MODULO (idModulo, nombre, horas)

PK: idModulo

ESTUDIANTE (NIA, nombre)

PK: NIA

NOTA_MODULO (NIA, codModulo, nota)

PK: NIA, codModulo

FK: NIA → ESTUDIANTE

FK: codModulo → MODULO

Tercera forma normal (3FN)

Ocurre cuando una tabla está en segunda forma normal (2FN) y además ningún atributo que no sea clave depende funcionalmente de forma transitiva (que un cambio en un registro concreto pueda afectar a la coherencia de los datos del resto de los registros) de la clave primaria. Veámoslo con un ejemplo:

ESTUDIANTE			
NIA	**nombreEstudiante**	**codProvincia**	**nombreProvincia**
11111111	Chewbakka	11	Cádiz
22222222	Han Solo	41	Sevilla
33333333	Luke Skywalker	29	Málaga
44444444	Darth Vader	11	Cádiz
55555555	Yoda	08	Madrid

¿Qué ocurre si actualizamos el nombre de la provincia con código 11 para Darth Vader a 'Estrella de la Muerte'? Valora si los datos resultantes serían coherentes.

ESTUDIANTE			
NIA	**nombreEstudiante**	**codProvincia**	**nombreProvincia**
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

Para evitar que se produzcan estas incoherencias, separaremos en dos tablas, por un lado, los datos de los alumnos y por otro lado los datos de las provincias:

ESTUDIANTE		
NIA	**nombreEstudiante**	**codProvincia**
11111111	Chewbakka	11
22222222	Han Solo	41
33333333	Luke Skywalker	29
44444444	Darth Vader	11
55555555	Yoda	08

PROVINCIA	
codProvincia	**nombreProvincia**
11	Cádiz
41	Sevilla
29	Málaga
08	Madrid

Modelo relacional normalizado:

PROVINCIA (codProvincia, nombreProvincia)

PK**:** codProvincia

ESTUDIANTE (NIA, nombreEstudiante, codProvincia)

PK: NIA

FK: codProvincia → PROVINCIA

Conclusión: con este cambio, si actualizamos la provincia de un estudiante, modificaremos su código, pero no modificaremos en ningún caso el nombre de la provincia. Si quisiéramos hacerlo, la modificaríamos para TODOS los estudiantes.

4. Álgebra relacional.

Es el sistema matemático que utilizan los **SGBD** para obtener registros de las tablas. El lenguaje que utiliza el álgebra relacional en los **SGBD** es el **SQL** (*Structured Query Language*), no obstante, es un lenguaje sencillo, muy parecido al inglés que facilita el diseño de las consultas.

4.1 Operaciones unarias (*sobre una misma tabla*)

Son aquellas operaciones del álgebra relacional que sólo afectan a una relación. Se destaca dos operaciones, la selección y la proyección.

Operación "Selección"

Crea una nueva relación a partir de otra, pero incluyendo sólo algunas de las tuplas a partir de un criterio dado. El criterio se basa en restricciones sobre los atributos de la relación R y no pueden incluirse otras relaciones en dicho criterio que no estén en R. Para representarla podemos utilizar tanto el símbolo σ como la letra S.

Sintaxis: S (Tabla, condición)

Las expresiones siguientes son equivalentes:

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

Ejemplo. Tabla Película.

Título	Año	Duración	Tipo	Estudio
Star Wars	1977	124	color	Fox
Mighty Ducks	1991	104	color	Disney
Wayne's World	1992	95	color	Paramount

S (Película, duración>=100)

Título	Año	Duración	Tipo	Estudio
Star Wars	1977	124	color	Fox
Mighty Ducks	1991	104	color	Disney

S (Película, duración >=100 AND Estudio='Fox')

Título	Año	Duración	Tipo	Estudio
Star Wars	1977	124	color	Fox

Operación "Proyección"

En lugar de seleccionar registros como en el caso anterior, se seleccionan atributos/columnas. Para representarla podemos utilizar el símbolo π o la letra P.

Sintaxis: P ((col1, col2, …colN), S (Tabla, condición))
Recibe dos parámetros (columnas y una tabla, que a su vez puede ser una selección)

Siguiendo con el ejemplo anterior, tenemos:
P ((título, estudio), S (Película, duración >= 100))

Título	Estudio
Star Wars	Fox
Mighty Ducks	Disney

También podríamos haberla escrito del siguiente modo:

πtítulo,estudio(σduración>=100 (Película))

4.2 Operaciones binarias

Son aquellas **operaciones** del álgebra relacional que **afectarán a dos relaciones**. Es importante recalcar que en la unión y en la diferencia, el número de columnas de las dos relaciones deberá ser el mismo. Por otra parte, en el producto cartesiano no sería necesario.

Los ejemplos que se describirán a continuación para cada operación vendrán dados por las siguientes relaciones:

Tabla R.

Nombre	Dirección	Género	FecNac
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

Tabla S.

Nombre	Dirección	Género	FecNac
Harrison Ford	789 Palm Dr.	M	7/7/77
Carrie Fisher	123 Maple St.	F	9/9/99

Operación binaria "Unión"

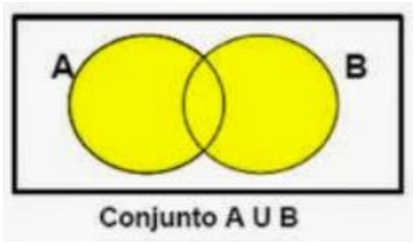
R ∪S, la unión de R y S define el conjunto de **elementos que están en R, en S o en ambos**. **Un elemento solo aparece una vez, por lo que no habrá tuplas repetidas**.

Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

Resultado de la operación R ∪S.

Nombre	Dirección	Género	FecNac
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



Operación binaria "Intersección"

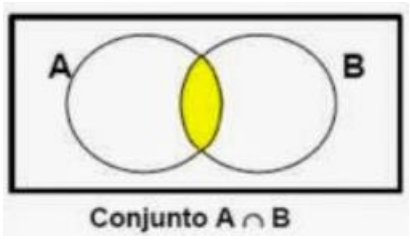
R ∩S, define el conjunto de elementos que aparecen en simultáneamente tanto en la relación R como en la relación S.

Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

Resultado de la operación.

Nombre	Dirección	Género	FecNac
Carrie Fisher	123 Maple St.	F	9/9/99



Operación binaria "Diferencia"

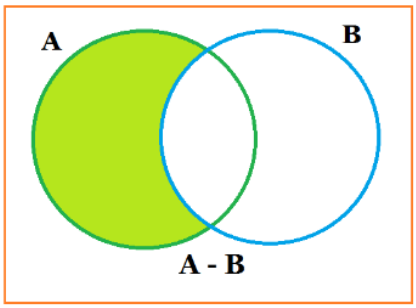
R – S o también denominada diferencia de R y S, define el conjunto de elementos que están en R pero no en S. Es importante resaltar que R - S es diferente a S - R.

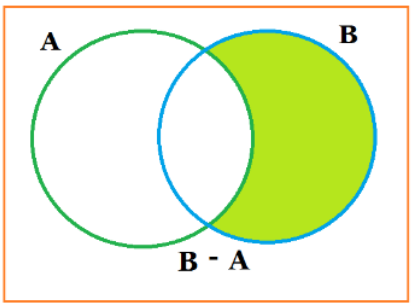
Requisitos para poder utilizarla:

- R y S deben tener esquemas idénticos.
- El orden de las columnas debe ser el mismo

Resultado de la operación R - S.

Nombre	Dirección	Género	FecNac
Mark Hamill	456 Oak Rd.	M	8/8/88





Operación binaria "Producto cartesiano"

R x S, los esquemas de ambas relaciones se mezclan y unen. No es necesario que R y S tengan esquemas idénticos ni tampoco es importante el orden de las columnas de ambas relaciones.

Resultado de la operación.

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

Operación binaria JOIN

Enlazando por las columnas que tengan el mismo nombre en la tabla A y en la tabla B o bien que representen el mismo campo (PK en una tabla y FK en la otra) conseguimos sacar información de otra tabla.

CLIENTES

idCliente	nombre
1	Pepe
2	Ana
3	María
4	Antonio

PAGOS

idPago	importe	fecha	IdCiente
1	100	03/10/2023	2
2	50	07/10/2023	2
3	120	11/10/2023	2
4	67	11/10/2023	3
5	32	11/10/2023	4
6	109	18/11/2023	3
7	89	20/11/2023	3
8	79	21/11/2023	4

Haciendo la JOIN entre estas dos tablas podemos mostrar información como "Pagos realizados por los clientes y el nombre de los clientes".

Como el nombre de los clientes está en la tabla CLIENTES, enlazaremos ambas tablas a través de un campo común, el cual siempre será la PK de una de las dos tablas con la FK de la otra, siempre y cuando sea el mismo campo.

RESULTADO DE LA CONSULTA

p.idPago	p.importe	p.fechaPago	p.idCliente	c.nombre
1	100	03/10/2023	2	Ana
2	50	07/10/2023	2	Ana
3	120	11/10/2023	2	Ana
4	67	11/10/2023	3	María
5	32	11/10/2023	4	Antonio
6	109	18/11/2023	3	María
7	89	20/11/2023	3	María
8	79	21/11/2023	4	Antonio

Las consultas no crean tablas nuevas, sino que son tan solo una visualización o instantánea en un momento dado del estado de las tablas.

En SQL la consulta se representaría del siguiente modo:

```
SELECT p.idPago, p.importe, p.fecha, p.idCliente, c.nombre
  FROM PAGOS p,
       CLIENTES c
 WHERE PAGOS.idCliente = CLIENTES.idCliente;
```