Unidad Didáctica 3:

ELABORACIÓN DEL DISEÑO LÓGICO:

MODELO RELACIONAL

1 El modelo relacional

En 1970, el modo en que se veían las bases de datos cambió por completo cuando E.F. **Codd** introdujo el **modelo relacional** . En aquellos momentos, el enfoque existente para la estructura de las bases de datos utilizaba punteros físicos (direcciones de disco) para relacionar registros de distintos ficheros. Si, por ejemplo, se quería relacionar un registro A con un registro B, se debía añadir al registro A un campo conteniendo la dirección en disco del registro B. Este campo añadido, un puntero físico, siempre señalaría desde el registro A al registro B. Codd demostró que estas bases de datos limitaban en gran medida los tipos de operaciones que los usuarios podían realizar sobre los datos. Estos sistemas se basaban en el **modelo de red** y el **modelo jerárquico** , los dos modelos lógicos que constituyeron la primera generación de los SGBD.

El modelo relacional representa la **segunda generación de los SGBD** . En él, todos los datos están estructurados a nivel lógico como **tablas** formadas por **filas** y **columnas** , aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica.

En los últimos años, se han propuesto algunas **extensiones** al modelo relacional para capturar mejor el significado de los datos, para disponer de los conceptos de la orientación a objetos y para disponer de capacidad deductiva.

El modelo relacional, como todo modelo de datos, tiene que ver con tres aspectos de los datos: **estructura de los datos** , **integridad de los datos** y **manejo de los datos** .

2 Objetivos del modelo relacional

En las bases de Codd se definían los objetivos de este modelo:

- **Independencia física**. La forma de almacenar los datos, no debe influir en su manipulación lógica. Por lo tanto, los usuarios que acceden a esos datos no tienen que modificar sus programas por cambios en el almacenamiento físico.

- **Independencia lógica**. El añadir, eliminar o modificar objetos de la base de datos no repercute en los programas y/o usuarios que están accediendo a subconjuntos parciales de los mismos (vistas).

- **Flexibilidad**. En el sentido de poder presentar a cada usuario los datos de la forma que éste prefiera. La base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones.

- **Uniformidad**. Las estructuras lógicas siempre tienen una única forma conceptual (las tablas), lo que facilita la concepción y manipulación de la base de datos por parte de los usuarios.

- **Sencillez**. Las características anteriores, así como unos lenguajes de usuario muy sencillos, producen como resultado que el modelo de datos relacional sea fácil de comprender y de utilizar por parte del usuario final.

3 Elementos básicos del modelo relacional

El modelo relacional se basa en el concepto matemático de **relación** , que gráficamente se representa mediante una tabla. Codd, que era un experto matemático, utilizó una terminología perteneciente a las matemáticas, en concreto de la teoría de conjuntos y de la lógica de predicados.

Una **relación** se define como un conjunto de atributos, cada uno de los cuales pertenece a un dominio, y que posee un nombre que identifica la relación. El conjunto de atributos de la relación y el nombre de la relación representan el *esquema*. Se representa gráficamente por una tabla con columnas y filas. Un SGBD sólo necesita que el usuario pueda percibir la base de datos como un **conjunto de tablas** . Esta percepción sólo se aplica a la estructura lógica de la base de datos (en la arquitectura de tres niveles ANSI-SPARC). No se aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento.

Esquema:

Películas(título,año,duración)

Figura 1. Ejemplo de la relación **PELÍCULAS**

Un **atributo** es el nombre de una columna de una relación. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos. Una relación se representa gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros. Los atributos pueden aparecer en la relación en cualquier orden.

> Por ejemplo, la información de las oficinas de la empresa inmobiliaria se podría representar mediante la relación OFICINA, que tiene columnas para los atributos idOficina, calle, area, población, telefono y fax .
>
> La información sobre la plantilla se representa mediante la relación PLANTILLA, que tiene columnas para los atributos idEmpleado, nombre, apellido, dirección, teléfono, puesto, fechaNacimiento, salario, dni y idOficina (número de la oficina a la que pertenece el empleado).

Un **dominio** es un conjunto finito de valores homogéneos y atómicos para uno o varios atributos. *Homogéneo* significa que los valores son todos del mismo tipo y *atómicos* significa que son indivisibles. Los dominios constituyen una poderosa característica del modelo relacional. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio.

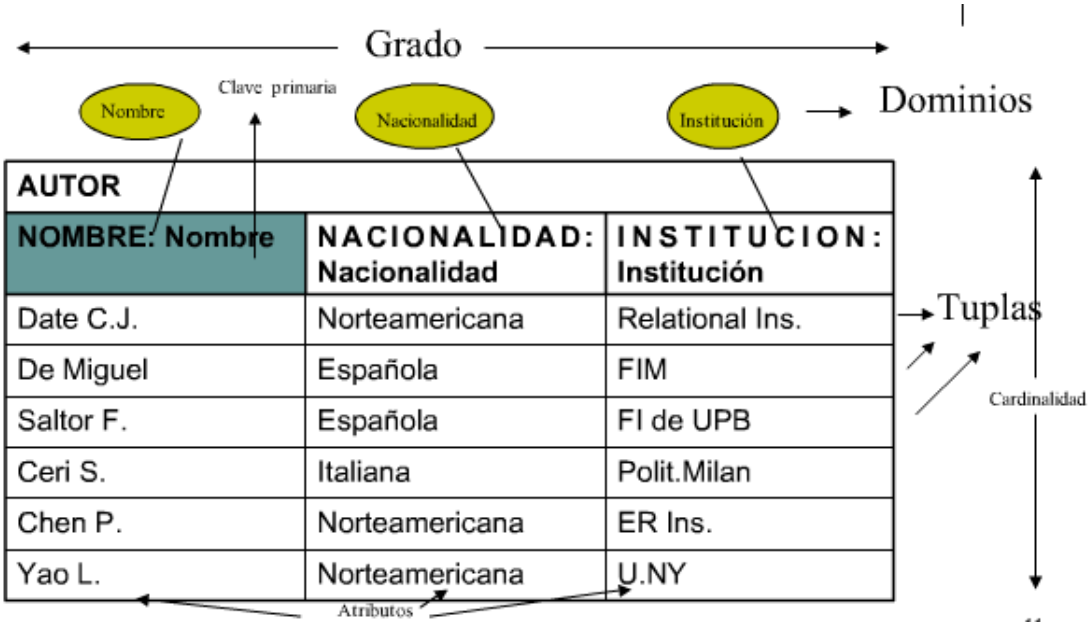


Figura 2. Ejemplo de la relación **AUTOR**

Cada dominio puede definirse de dos maneras:

- Por **extensión** (enumeración de sus elementos):

```
días de la semana = {lunes, martes, miércoles, jueves,
viernes, sábado, domingo}
```

- Por **intensión** (mediante una propiedad que recoja el recorrido de sus valores admisibles):

```
edad: entero / 0 < edad<150
```

Un **dominio compuesto** se puede definir como una combinación de dominios simples a la que se puede aplicar ciertas restricciones.

```
Ej.: el dominio compuesto denominado Fecha se construye por
agregación de los dominios simples Día, Mes y Año,
incorporando las restricciones a fin de que no aparezcan
valores inválidos como: 29/2/2003, 31/4/2004.
```

Cada dominio incorpora su nombre y un tipo de datos.

```
Ejemplos de dominio:

    Dominio de Dirección: 50 caracteres
    Dominio de Nacionalidad: Español, Francés, Italiano,...
```

Una **tupla** es una fila de una relación. Los elementos de una relación son las tuplas o filas de la tabla. En la relación AUTOR , cada tupla tiene tres valores, uno para cada atributo. Las tuplas de una relación no siguen ningún orden.

Un **esquema de relación** R se denotará por R(A1, A2, ..., An), y se compone de un nombre de relación R y una lista de atributos A1, A2..., An. Un esquema de relación sirve para describir una relación.

AUTORES (nombre, nacionalidad, institución)

El **grado** de una relación es el número de atributos que contiene. La relación AUTOR es de grado tres porque tiene tres atributos. Esto quiere decir que cada fila de la tabla es una tupla con tres valores. El grado de una relación no cambia con frecuencia.

La **cardinalidad** de una relación es el número de tuplas que contiene. Ya que en las relaciones se van insertando y borrando tuplas a menudo, la cardinalidad de las mismas varía constantemente.

4 Propiedades de las relaciones

Una **base de datos relacional** es un conjunto de relaciones normalizadas que tienen que cumplir las siguientes características:

- Cada relación tiene un nombre y éste es distinto del nombre de todas las demás.
- Los valores de los atributos son atómicos : en cada tupla, cada atributo toma un solo valor. Se dice que las relaciones están normalizadas o en primera forma normal.
- No hay dos atributos que se llamen igual.
- El orden de los atributos no importa: los atributos no están ordenados.
- Cada tupla es distinta de las demás: no hay tuplas duplicadas. De ahí, la diferencia entre tabla y relación: las tablas pueden tener filas duplicadas. Lamentablemente, SQL permite tener filas duplicadas.
- El orden de las tuplas no importa: las tuplas no están ordenadas.

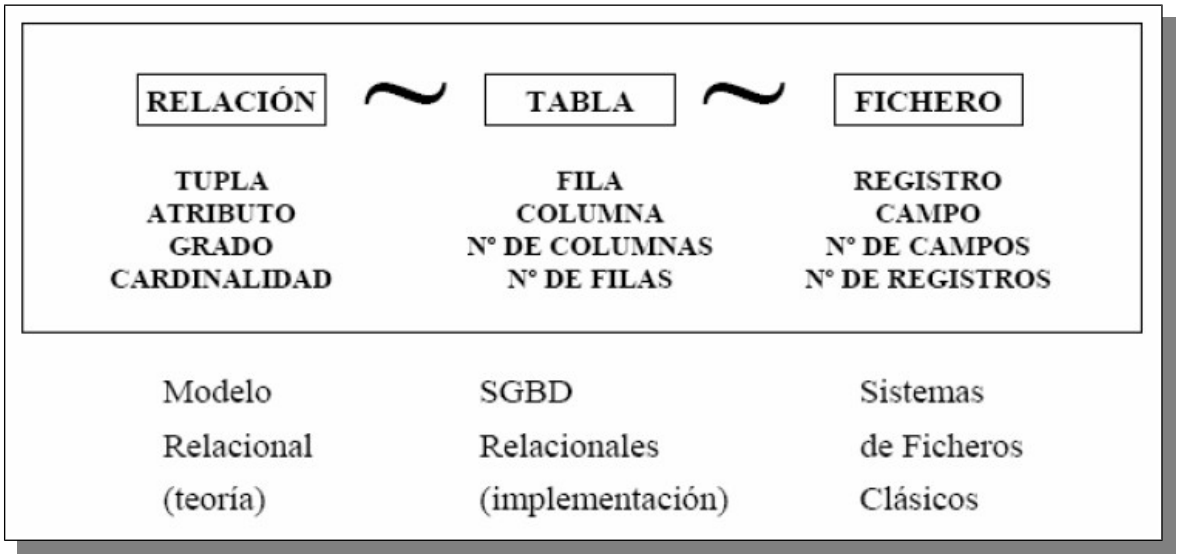


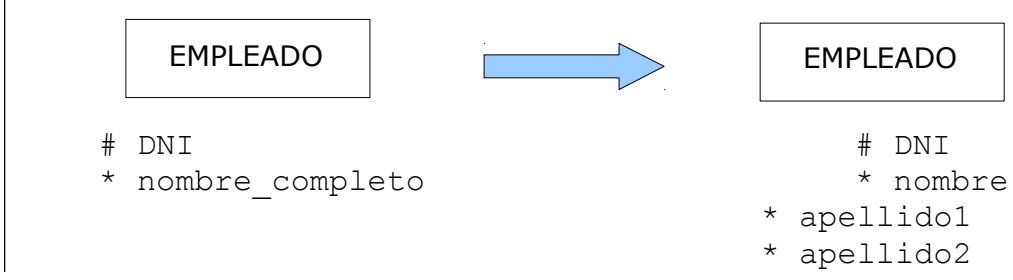
Figura 3. Diferentes terminologías

5 Transformación del modelo E/R al modelo relacional

Vamos a describir brevemente unas **reglas preliminares** que nos permitirán preparar los modelos conceptuales basados en el modelo Entidad-Relación para poder traducirlos posteriormente a modelos lógicos basados en el modelo relacional. El seguimiento de estas reglas facilitará y garantizará la fiabilidad del proceso de transformación.

■ Eliminación de atributos compuestos

Los atributos compuestos de una entidad han de ser descompuestos en los atributos simples por los que están formados. De esta forma, un atributo denominado *nombreCompleto* habría que descomponerlo en sus atributos simples, es decir, *nombre* , *apellido1* y *apellido2.*



■ Eliminación de atributos multivaluados

Si nuestro diagrama incluye la existencia de un atributo multivaluado, este se ha de convertir en una entidad relacionada con la entidad de la que procede. Para esta nueva entidad se elegirá un nombre adecuado y tendrá un único atributo (el correspondiente al antiguo atributo múltiple).

Este atributo es posible que funcione correctamente como clave primaria de la entidad pero a veces es posible que no. En este caso, la entidad que hemos creado es posible que sea débil. Deberemos ajustar en cualquier caso correctamente las claves primarias o los discriminadores tal y como se ha visto anteriormente.

EMPLEADO

DNI
* nombre
* apellido1
* apellido2

• direcciones (supongamos que tiene 1,N direcciones)

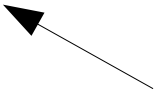
Solución:

Crearíamos una nueva tabla (relación) DIRECCION_PERSONA, en la que el atributo multivaluado se representa como un atributo simple *dirección*.

La nueva tabla contendrá, un atributo, clave ajena a la clave primaria de la tabla EMPLEADO.

La clave primaria de la nueva tabla será la concatenación de *DNI+Dirección*

EMPLEADOS(dni, nombre, apellido1, apellido2)



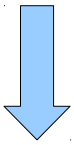
DIRECCIONES_PERSONA(dni, dirección)

5.1 Conversión de entidades y sus atributos

- Toda **entidad** se transforma en una relación.
- Todo **atributo** de la entidad pasa a ser un atributo de la relación.
- El **atributo identificador principal** de la entidad se transforma en la clave primaria de la relación (PRIMARY KEY).
- Los **atributos identificadores alternativos** serán marcados como claves alternativas (UNIQUE + NOT NULL).

MER


```
# id_articulo
* articulo
* color
* precio
```



MR

ARTICULOS(id_articulo, articulo, color, precio)

ARTICULO			
idArticulo	articulo	color	precio
a1	Camisa hombre manga larga	Rojo	35,00
a2	Jersey hombre de cuello alto	Blanco	56,95
a3	Cazadora hombre piel vuelta	Marrón	96,95
a4	Camiseta mujer	Amarilla	16,90
a5	Pantalones vaqueros mujer	Azul	66,50

Figura 1. Tabla ARTICULOS

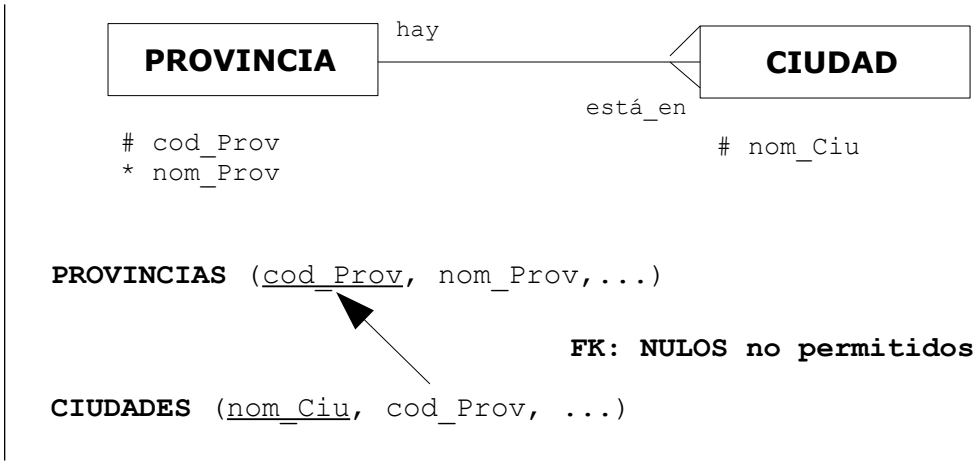
5.2 Conversión de relaciones binarias con cardinalidad 1:N

Para las **relaciones 1:N** existen 2 soluciones:

1. **Propagar la clave**, es decir, se propaga la clave primaria de la entidad con cardinalidad 1 a la entidad cuya cardinalidad es N (en la relación correspondiente a la otra entidad aparecerá esta clave como clave ajena).

   Se suele realizar cuando:

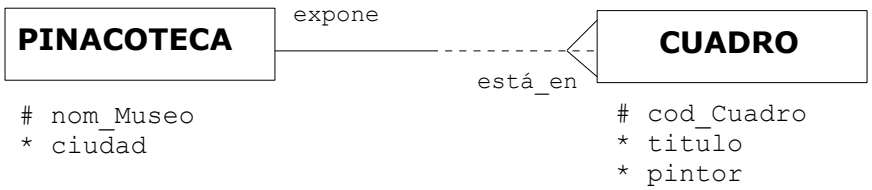
   - La participación de la entidad cuya cardinalidad máxima es N es obligatoria (participación TOTAL u obligatoria).



2. **Transformar la relación en una tabla.** Dicha relación tendría como atributos las claves de ambas entidades . Su clave sería la clave de la entidad que interviene en la interrelación con N ocurrencias.

   Se puede realizar cuando ocurren algunos de estos casos:

   - La participación de la entidad cuya cardinalidad máxima es N es opcional y, por lo tanto, aparecen muchos nulos en la clave propagada.
   - Se prevé que se convertirá en una relación N:M.



Solución si NO hay muchos nulos

PINACOTECAS (nom_Museo, ciudad,...)

FK: NULOS permitidos

CUADROS (cod_Cuadro, nom_Museo, titulo, autor,...)

Solución si hay muchos nulos

PINACOTECAS (nom_Museo, ciudad,...)

EXPOSICIONES (cod_Cuadro, nom_Museo)

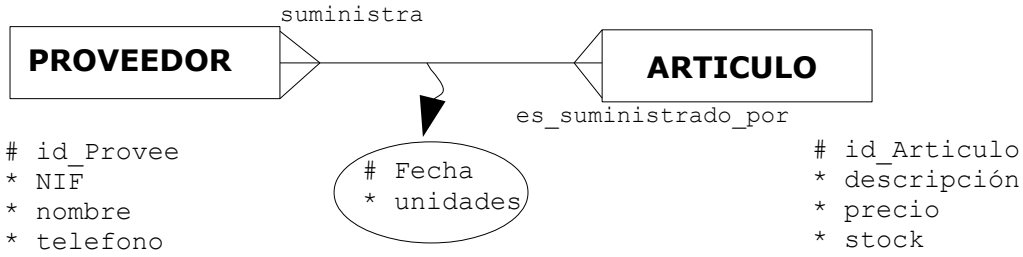
CUADROS (cod_Cuadro, titulo, pintor,...)

Clave primaria: atributo procedente de la entidad con cardinalidad N

5.3 Conversión de relaciones binarias con cardinalidad N:M

Toda relación N:M se transforma en una tabla, que tendrá como claves ajenas las claves primarias de las entidades que asocia. Podrá tener como clave primaria la concatenación de los atributos clave de las entidades que asocia ,si es posible. Sino, se utilizan junto con uno o varios atributos de la relación o se le agrega un campo identificador nuevo como clave primaria.

Ejemplo: *Cada proveedor suministra varios artículos, y cada artículo puede ser suministrado por distintos proveedores.*



El **esquema relacional** sería:

PROVEEDORES (id_Provee, NIF,nombre, teléfono).

SUMINISTROS (id_Suministro, id_Provee, id_Artículo, fecha, unidades).

ARTÍCULOS (id_Artículo, descripción, precio, stock).

En este caso, **se crea la tabla nueva con los atributos de la relación y las claves primarias de las entidades como claves ajenas**. Como no puede usarse como clave primaria de SUMINISTROS la concatenación de *id_Provee* e *id_Artículo*, pues en dicha tabla pueden aparecer suministros del mismo artículo por el mismo proveedor, añadimos *id_Suministro*. (también podría ser PK la concatenación de *id_Provee*, *id_Articulo* y *fecha*).

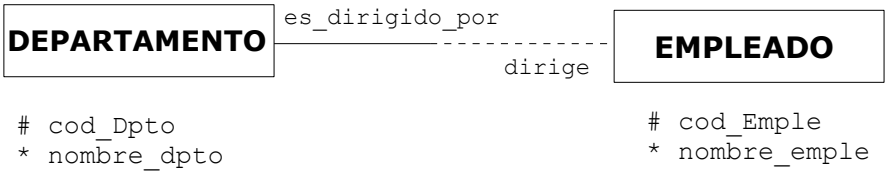
5.4 Conversión de relaciones binarias con cardinalidad 1:1

Se puede considerar un caso particular de las anteriores, y por tanto las soluciones anteriores son válidas también en este caso.

Para las relaciones 1:1 se tienen en cuenta la participación de las entidades que participan en la relación. Por lo tanto, podemos tener tres soluciones:

1. **Propagar la clave**, cuando una entidad tiene participación total y la otra participación parcial. Se propaga la clave primaria de la entidad con participación parcial a la tabla resultante de la entidad con participación total convirtiéndose en clave ajena.

Ejemplo: *Un empleado de la empresa puede dirigir un (único) departamento o bien, no dirigir ninguno. Todo departamento tiene un director.*



El esquema relacional sería:

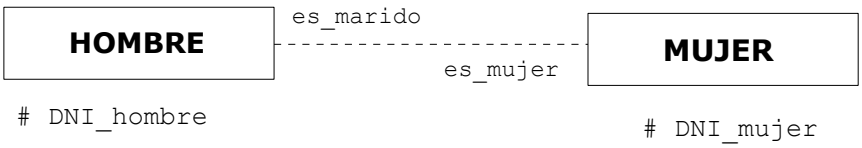
DEPARTAMENTOS(cod_Dpto, nombre_dpto, cod_Director)

EMPLEADOS(cod_Emple, nombre_emple)

En este caso, al tener la entidad **EMPLEADO** participación parcial en la relación dirigir, se propaga su clave *cod_Emple* a la entidad con participación total **DEPARTAMENTO**.

2. **Transformar la relación en una tabla**, cuando ambas entidades tienen participación parcial.

Ejemplo: *Un hombre está casado con una mujer o con ninguna y una mujer está casada con un hombre o con ninguno.*



El esquema relacional sería:

HOMBRES(DNI_hombre, ...)

MATRIMONIOS(DNI_hombre, DNI_mujer, ...)

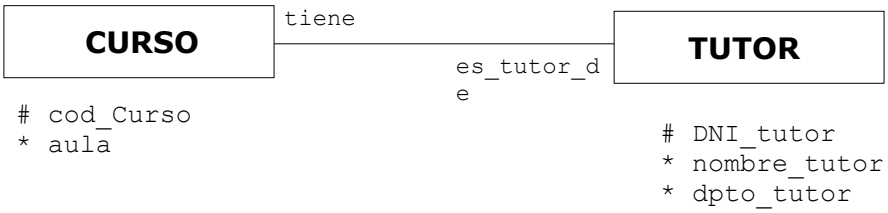
MUJERES(DNI_mujer, ...)

En este caso, al tener las dos entidades participación parcial en la relación, se crea una nueva tabla **MATRIMONIO** y su PK será o la clave de HOMBRE o la clave de MUJER.

En este caso, evitaríamos todos los valores nulos que aparecerían en el caso de propagar la clave de MUJER a la tabla HOMBRE o viceversa, ya que no todos los hombres ni todas las mujeres están casados.

3. Cuando ambas entidades tienen participación total en la relación, se escoge **cualquiera de las dos soluciones anteriores**.

Ejemplo: *Cada curso tiene un sólo tutor y cada tutor lo es de un solo curso.*



Un esquema relacional sería:

CURSO_Y_TUTORES(cod_Curso,DNI,nombre_tutor,dpto_tutor, aula)

En este caso, al tener ambas entidades participación total, se funden ambas entidades en **una sola tabla**, y se toma como PK cualquiera de las claves de ambas entidades. En este caso se elige *cod_Curso*.

El otro esquema relacional sería:

CURSOS(cod_Curso,aula, DNI_tutor)

TUTORES(DNI_tutor,nombre_tutor,dpto_tutor)

o

CURSOS(cod_Curso,aula)

TUTORES(DNI_tutor,nombre_tutor,dpto_tutor, cod_Curso)

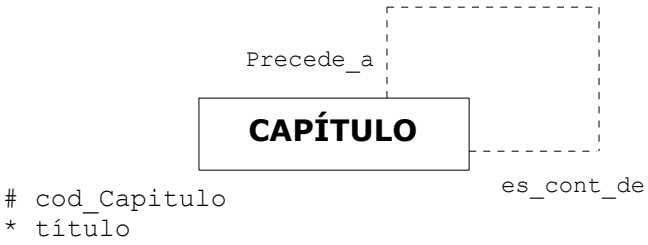
También es necesario tener en cuenta que, la clave foránea que se les añade se convierte en una clave alternativa de la relación porque no admite valores repetidos.

5.5 Conversión de relaciones recursivas

Son las relaciones binarias en las que únicamente participa un tipo de entidad. Pueden encontrarse los siguientes casos:

1. Si la relación es **1:1**, no se crea una segunda tabla, si no que en la tabla resultante se agregará 2 veces el mismo atributo, como clave primaria y como clave ajena a ella misma.

Ejemplo: *Una capítulo de una serie de televisión es el siguiente de otro capítulo o de ninguno (si es el primero). Un capitulo de la serie es el anterior de otro o de ninguno (si es el último).*

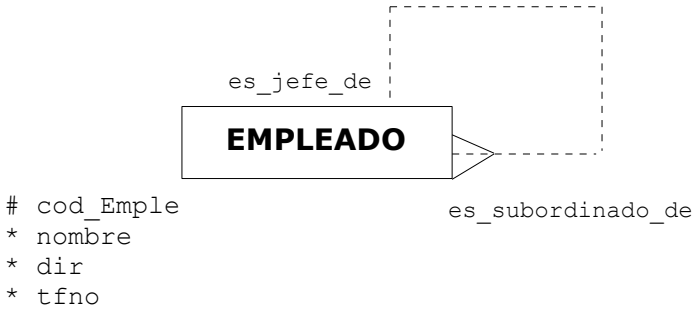


El esquema relacional sería:

CAPITULOS(cod_Capitulo, titulo, cod_Capitulo_anterior)

2. Si la relación es **1:N**, podemos adoptar dos soluciones:

Ejemplo: *Un empleado puede dirigir a muchos empleados si es el jefe, o a ninguno si no es el jefe. Un empleado es dirigido por un jefe, o por ninguno si él mismo es el jefe.*



Una solución sería:

EMPLEADOS(cod_Emple, nombre, dir, tfno, cod_Jefe)

Solución problemática si puede haber muchos empleados sin jefe → demasiados nulos.

La otra posible solución es:

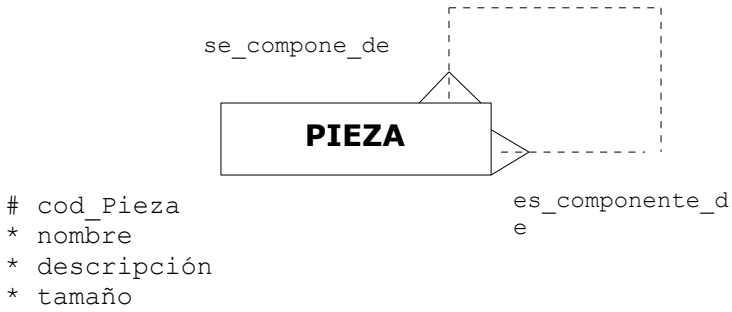
EMPLEADOS(cod_Emple, nombre, dir, tfno)

JEFES(cod_Emple, cod_Jefe)

En este caso, se crea la tabla **JEFE** que tiene a la vez como clave primaria y ajena la misma que la de la tabla **EMPLEADO**, identificando al empleado al que se hace referencia; y además, tiene el mismo campo como clave ajena que indica qué otro empleado es su director.

3. Si la relación es **N:M**, se trata igual que en las relaciones binarias. La tabla resultante de la relación contendrá 2 veces la clave primaria de la entidad del lado muchos, más los atributos de la relación si los hubiera. La clave de esta nueva tabla será la combinación de las 2.

Ejemplo: *Una pieza se compone de muchas piezas, que a su vez están compuestas de otras piezas. Es decir, una pieza se compone de una o varias piezas más pequeñas. Una pieza forma parte de una o varias piezas más grandes.*



El esquema relacional sería:

PIEZAS (Cód_Pieza, Descripción, Tamaño, Nombre)

COMPONENTES (CódPieza, Cód_Pieza_Componente)

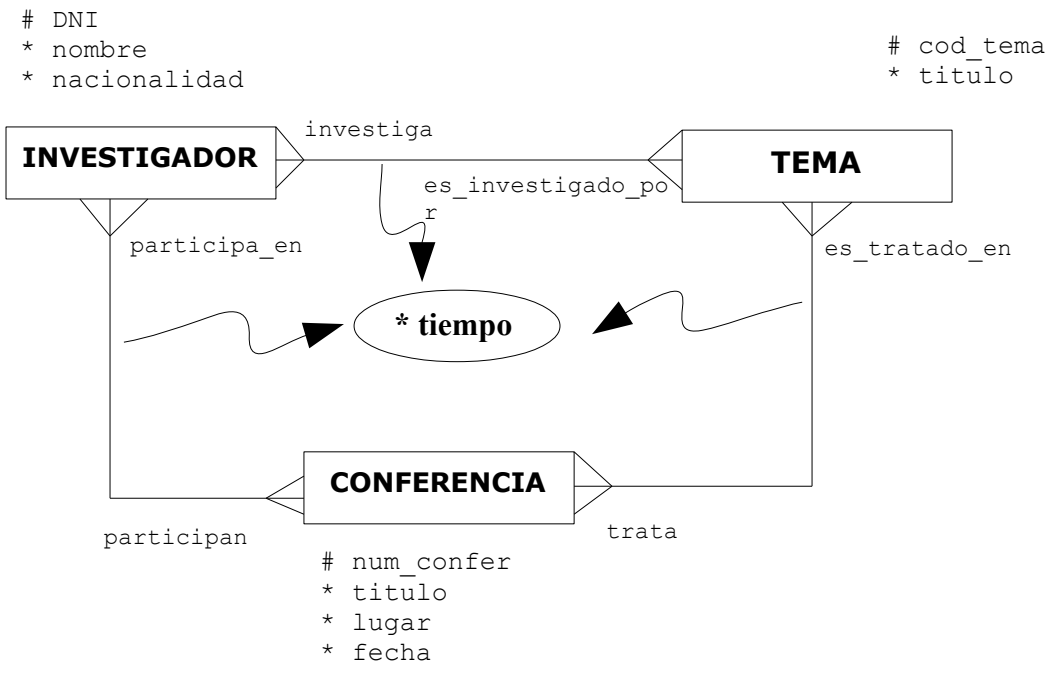
En este caso, se crea la tabla **COMPONENTES** que tiene como clave primaria la concatenación de la clave primaria de la tabla PIEZA (identificando la pieza compuesta), más el campo *Cód_Pieza_Componente* también (identificando la pieza de la que se compone la pieza compuesta). Además, ambas son claves ajenas.

5.6 Conversión de relaciones N-arias (ternarias, cuaternarias,...)

En este tipo de relaciones se asocian 3 ó más entidades. Se pasan todas las entidades a tablas tal cual. La relación también se convierte a una tabla, que va a contener sus atributos más las claves primarias de todas las entidades que asocia como claves ajenas.

Hay 2 casos:

1. Si la relación es **N:N:N**, es decir, todas las entidades participan con cardinalidad máxima N, la clave de la tabla resultante de la relación es la unión de las claves primarias de todas las entidades que participan en la relación.



El esquema relacional sería:

INVESTIGADORES (DNI, nombre, nacionalidad)

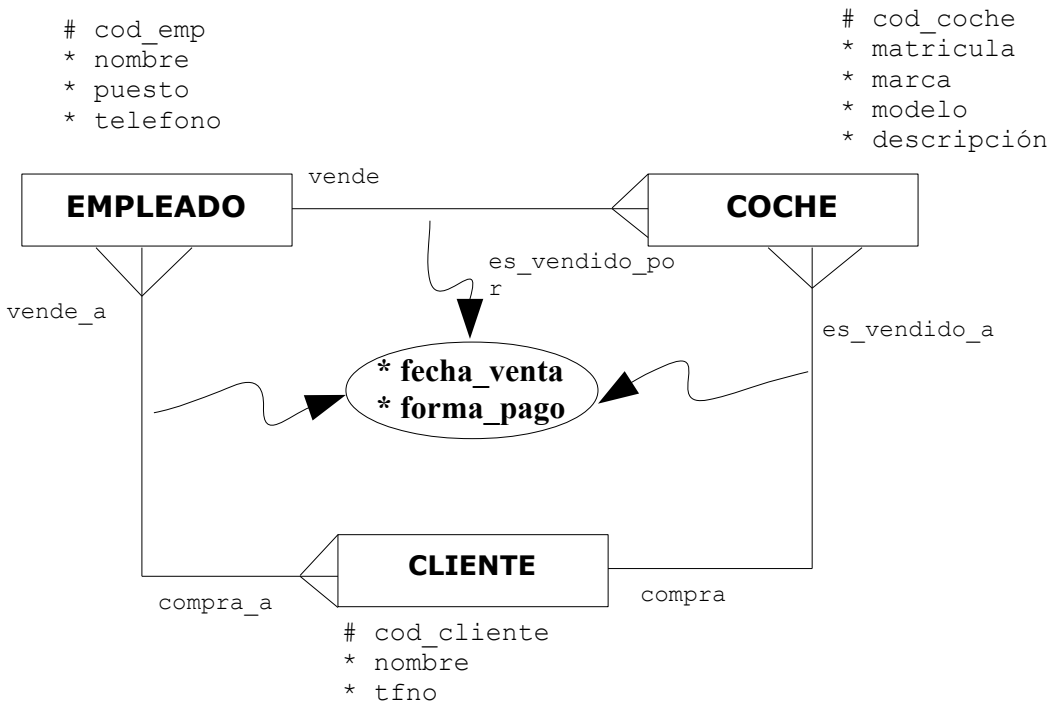
CONFERENCIAS (num_confer, titulo, lugar, fecha)

TEMAS (cod_tema, titulo)

PARTICIPACIONES (DNI, num_confer, cod_tema, tiempo)

siendo cada componente de la PK una FK a cada tabla.

2. Si alguna entidad participa en la relación con cardinalidad máxima 1, su clave primaria no forma parte de la clave primaria de la tabla creada.



El esquema relacional sería:

EMPLEADOS (<u>cod_emple</u>, nombre,puesto, teléfono)

COCHES (<u>cod_coche</u>, matrícula, marca, modelo, descripción)

CLIENTES (<u>cod_cliente</u>, nombre, tfno)

VENTAS (cod_emple, cod_cliente, <u>cod_coche</u>, fecha_venta, forma_pago)

5.7 Transformación de tipos y subtipos

En las jerarquías de tipos y subtipos, se **denomina entidad padre** a la entidad genérica y **entidades hijo** a las entidades subtipo. Hay tres opciones distintas para representar estas jerarquías. La elección de la más adecuada se hará en función de su tipo (total/parcial, con solapamiento/exclusiva).

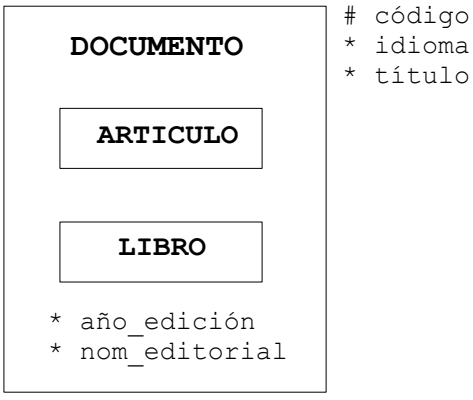
- **Integrar todas las entidades en una única tabla absorbiendo los subtipos**: Se crea una tabla que contiene todos los atributos del supertipo, todos los de los subtipos, y un atributo discriminatorio para distinguir a qué subtipo pertenece cada fila de la tabla. Esta regla puede aplicarse a cualquier tipo de jerarquía (total o parcial y exclusiva o solapada), es muy simple de realizar, pero genera demasiados valores nulos en los atributos opcionales propios de cada subtipo.

Puede hacerse cuando:

- los **atributos** de los subtipos son **similares.**
- las **interrelaciones** que involucran a los subtipos son las mismas (o no existen).

Será necesario implementar las restricciones semánticas necesarias a través de CHECKS o DISPARADORES.

MER



MR

DOCUMENTOS(<u>código</u>, título, idioma, tipo, año_edición, nom_editorial)

discriminante

Restricciones semánticas

```
CREATE TABLE DOCUMENTO(
codigo ... PRIMARY KEY,
titulo... ,
idioma ... ,
tipo ... ,
nom_editorial ... ,
año_edicion ... ,
...
CHECK (( tipo = "ARTICULO" AND
            año_edicion IS NULL AND
            nom_editorial IS NULL)
      OR ( tipo = "LIBRO" AND
            año_edicion IS NOT NULL AND
            nom_editorial IS NOT NULL))
```

Ventajas e Inconvenientes:

- Acceso eficiente a TODA la información sobre una entidad concreta (acceso a una sola tabla).
- Aparición de nulos (atributos que proceden de subtipos para filas que no pertenecen a tales subtipos).
- Toda operación sobre subtipos debe "buscar" las instancias de los subtipos en el conjunto completo (supertipo) de instancias.

- **Eliminación del supertipo en jerarquías totales y exclusivas**: Transfiriendo los atributos del supertipo a cada uno de los subtipos, creándose una tabla por cada subtipo. El supertipo no tendrá tabla, y se elimina el atributo que distingue entre subtipos. Se crea redundancia en la información pues los atributos del supertipo se repiten en cada uno de los subtipos. El número de relaciones aumenta, pues las relaciones del supertipo pasan a cada uno de los subtipos. Esta opción sólo sirve para jerarquías totales y exclusivas.

Puede hacerse cuando:

- los subtipos tienen **atributos dispares y/o interrelaciones diferentes.**
- Existen **pocos atributos comunes en el supertipo.**

MR

ARTICULOS(<u>código</u>, título, idioma)

LIBROS(<u>código</u>, título, idioma,año_edición, nom_editorial)

Ventajas e Inconvenientes:

- Funciona bien para jerarquías totales y exclusivas.
- Con jerarquías solapadas aparecen "repeticiones".
- Con jerarquías parciales surgen problemas de "falta de representación" de filas no pertenecientes a ningún subtipo.

- **Crear una tabla por cada entidad** . Las tablas de las entidades hijo añaden como clave primaria la de la entidad padre. Por lo tanto, las claves primarias de las tablas que representan a las entidades hijo son también clave ajena haciendo referencia al padre. Esta opción sirve para cualquier tipo de jerarquía, total o parcial y exclusiva o superpuesta.

Puede hacerse cuando:

- Los **subtipos se diferencian en muchos atributos.**
- Se desea **mantener los atributos comunes** en una relación separada.

MR

DOCUMENTOS(<u>código</u>, título, idioma)

ARTICULOS(<u>código</u>)

LIBROS(<u>código</u>,año_edición, nom_editorial)

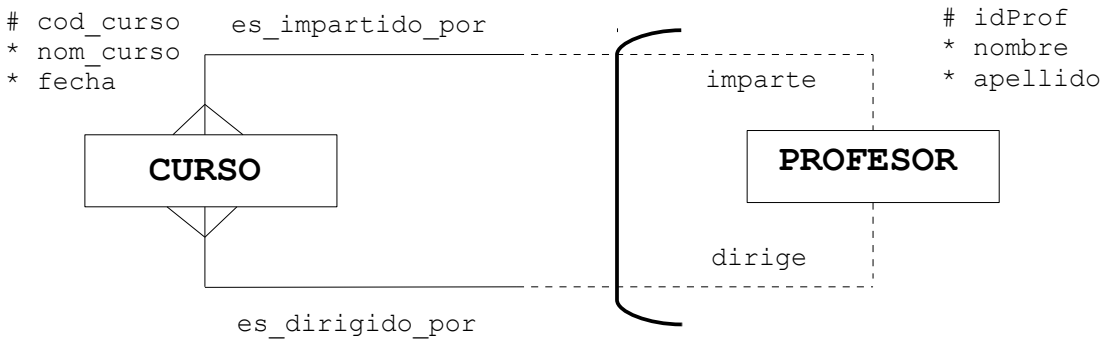
Ventajas e Inconvenientes:

- Funciona para todo tipo de jerarquías.
- Es la mejor desde el punto de vista semántico.
- Menos eficiente en el acceso.

5.8 Restricciones que no pueden plasmarse en el diseño lógico: arcos exclusivos

Los arcos exclusivos son elementos del MER que no son directamente convertibles al MR y que deben ser sometidos a un proceso previo de eliminación.

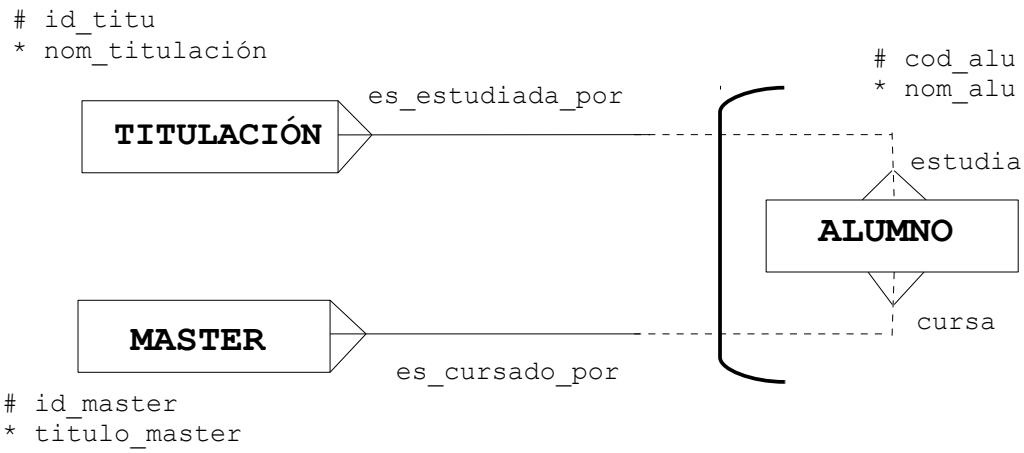
Caso 1:N:
Curso es organizado o impartido por profesores.




```
CREATE TABLE Curso (
cod_curso PRIMARY KEY,
Nom_curso...
……
director   ..REFERENCES   Profesor(idProf)   ON   UPDATE
CASCADE
profesor..REFERENCES    Profesor(idProf)    ON    UPDATE
CASCADE
……
CONSTRAINT organiza_xor_imparte
CHECK (( director NOT IN (SELECT profesor FROM CURSO)
    AND (profesor NOT IN (SELECT director FROM CURSO))
);
```

Caso N:M:

Alumno estudia titulaciones o cursa masters.

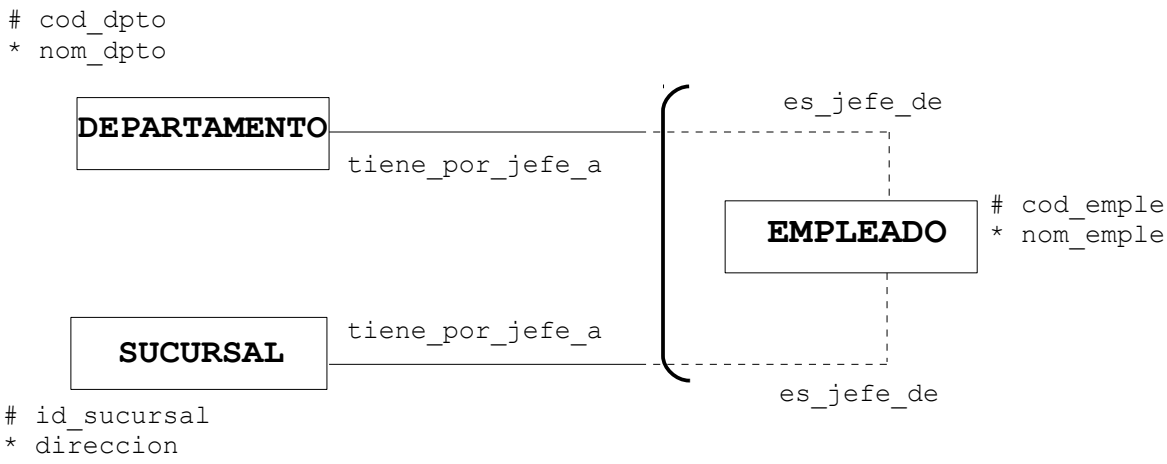



```
CREATE TABLE Alumno_estudia_titulacion (
……
alu ..REFERENCES Alumno(cod_alu) ...
titu..REFERENCES Titulacion(id_titu) ...
….
PRIMARY_KEY(alu, titu),
...
CONSTRAINT titulacion_xor_master
CHECK (( alu NOT IN (SELECT alu FROM
alumno_cursa_master))
);
```

Similar para la tabla Alumno_cursa_master.

Caso 1:1:

Empleado jefe de departamento o director de sucursal.




```
CREATE TABLE Departamento (
codDep ….PRIMARY KEY
...
jefe   ..REFERENCES   Empleado(cod_emple)   ON   UPDATE
CASCADE
...
CONSTRAINT jefe_ok
CHECK (jefe NOT IN (SELECT director FROM Sucursal))
);
```

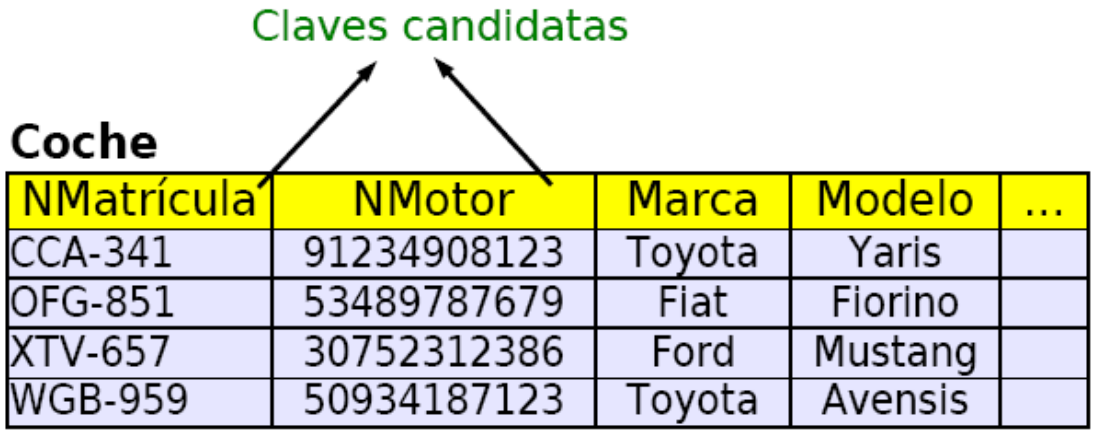
Similar para la tabla Sucursal.

6 Atributo clave

Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos.

Una **clave** es un atributo o un conjunto de atributos que identifican de modo único las tuplas de una relación.

Una **clave candidata** es una clave en la que ninguno de sus subconjuntos es una clave de la relación. El atributo o conjunto de atributos K de la relación R es una clave candidata para R si y sólo si satisface las siguientes propiedades: *unicidad* (nunca hay dos tuplas en la relación R con el mismo valor de K) e irreducibilidad o *minimalidad*: ningún subconjunto de K tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de K sin destruir la unicidad.



Cuando una clave candidata está formada por más de un atributo, se dice que es una *clave compuesta*. Una relación puede tener varias claves candidatas.

> Por ejemplo, en la relación **OFICINA** , el atributo *población* no es una clave candidata ya que puede haber varias oficinas en una misma población. Sin embargo, ya que la empresa asigna un código único a cada oficina, el atributo *idOficina* sí es una clave candidata de la relación OFICINA . También son claves candidatas de esta relación los atributos *teléfono* y *fax* .
>
> En la base de datos de la **inmobiliaria** hay una relación denominada **VISITA** que contiene información sobre las visitas que los clientes han realizado a los inmuebles.
>
> Esta relación contiene el número del cliente *idCliente* , el número del inmueble *idInmueble* , la fecha de la visita *fecha* y un *comentario* opcional. Para un determinado número de

> cliente *idCliente* , se pueden encontrar varias visitas a varios inmuebles. Del mismo modo, dado un número de inmueble *idInmueble* , puede que haya varios clientes que lo hayan visitado. Por lo tanto, el atributo *idInmueble* no es una clave candidata para la relación VISITA, como tampoco lo es el atributo *idCliente* . Sin embargo, la combinación de los dos atributos sí identifica a una sola tupla, por lo que los dos juntos son una clave candidata de VISITA .
>
> Si se desea considerar la posibilidad de que un mismo cliente pueda visitar un mismo inmueble en varias ocasiones, habría que incluir el atributo *fecha* para identificar las tuplas de modo único.

Para identificar las claves candidatas de una relación no hay que fijarse en un estado o instancia de la base de datos. El hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles.

Sin embargo, la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. Sólo usando esta información semántica se puede saber con certeza si un conjunto de atributos forman una clave candidata.

> Por ejemplo, viendo la instancia anterior de la relación PLANTILLA se podría pensar que el atributo *apellido* es una clave candidata. Pero ya que este atributo es el apellido de un empleado y es posible que haya dos empleados con el mismo apellido, el atributo no es una clave candidata.

La **clave primaria (PRIMARY KEY)** de una relación es aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función.

Una **clave alternativa(ALTERNATIVE KEY)** de una relación es aquella o aquellas claves candidatas que no se han escogido como claves primarias.

Una **clave ajena (FOREIGN KEY)** es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (que incluso puede ser la misma). Las claves ajenas representan relaciones entre datos. Notar que la clave ajena y la correspondiente clave primaria han de estar definidas sobre los mismos dominios.

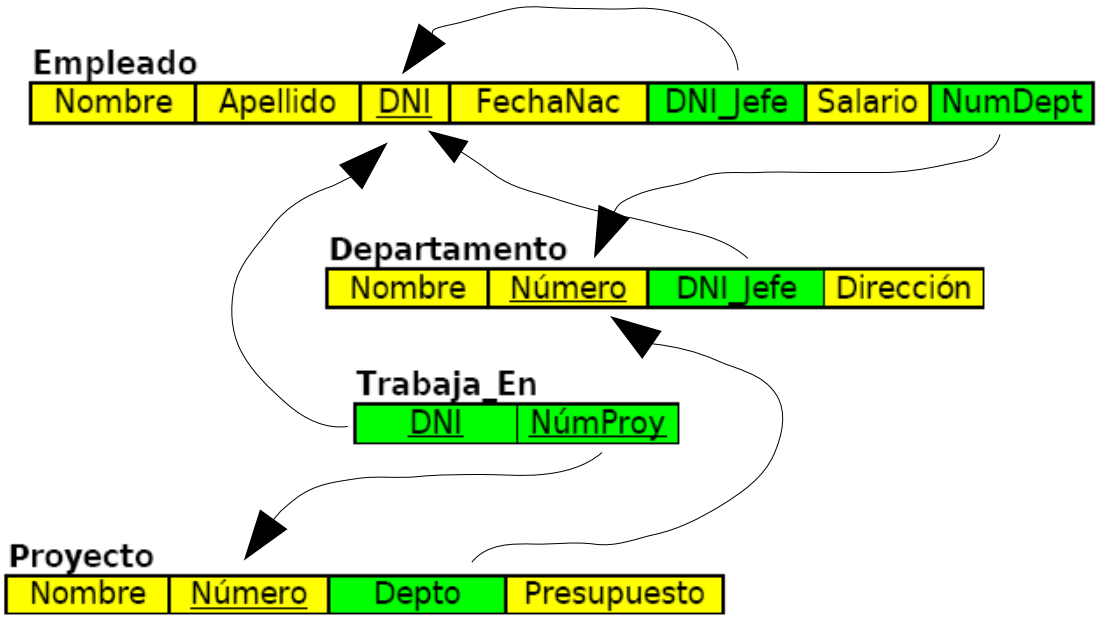


Figura 4. Esquema Relacional de la BD **EMPRESA**.

7 Restricciones del modelo relacional

Se trata de unas condiciones de obligado cumplimiento por los datos de la base de datos. Las hay de varios tipos.

7.1 Inherentes al modelo relacional

Son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional. Por ejemplo:

- No puede haber dos tuplas iguales.
- El orden de las tuplas no importa.
- El orden de los atributos no importa.
- Cada atributo sólo puede tomar un valor en el dominio en el que está inscrito. Es lo que se denomina ***restricciones de dominios*** : al definir cada atributo sobre un dominio, se impone una restricción sobre el conjunto de valores permitidos para cada atributo.
- Ningún atributo que forme parte de la clave primaria puede tomar valor nulo, es decir, un valor desconocido o inexistente (***regla de integridad de entidad).***

7.2 Semánticas de usuario

El modelo relacional permite a los usuarios incorporar restricciones personales a los datos.

Las principales son:

- **Restricción de clave primaria (PRIMARY KEY)**. Hace que los atributos marcados como clave primaria no puedan repetir valores.

- **Restricción de unicidad (UNIQUE)**. Impide que los valores de los atributos marcados de esa forma, puedan repetirse. Permite al usuario definir claves alternativas.

- **Restricción de obligatoriedad (NOT NULL)**. Permite al usuario declarar si uno o varios atributos no pueden tomar valores nulos, por tanto, deben tener siempre un valor.

  Cuando en una tupla un atributo es desconocido, se dice que es **nulo** . Un nulo no representa el valor cero ni la cadena vacía, éstos son valores que tienen significado. El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

  Ya que los nulos no son valores, deben tratarse de modo diferente, lo que causa problemas de implementación. De hecho, no todos los SGBD relacionales soportan los nulos.

- **Restricción de integridad referencial (FOREIGN KEY)**. Si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser todos nulos.

  Tras definir las claves ajenas hay que tener en cuenta qué ocurrirá cuando se borren **(ON DELETE en SQL)** o se modifiquen **(ON UPDATE en SQL)** tuplas de la relación cuya clave primaria se corresponde con esta clave ajena.

  - **Borrado y/o modificación en cascada (CASCADE en SQL):** Si el usuario especifica esta restricción, el borrado o modificación de una tupla en una relación ocasiona un borrado o modificación de las tuplas de otras relaciones asociadas a ésta, cuyas claves ajenas se correspondan con su clave primaria.

    > *Por ejemplo, si se borra un departamento de la relación DEPARTAMENTOS, se borrarán todos los empleados de la tabla EMPLEADOS que pertenezcan a ese departamento borrado; y si se modifica el código de un departamento de la relación DEPARTAMENTOS, automáticamente cambiarán todos los valores del campo Departamento en la relación EMPLEADOS que tenía el mismo código al nuevo valor ya modificado.*

  - **Borrado y/o modificación restringidos (RESTRICT en SQL):** Si el usuario especifica esta restricción, el borrado o modificación de una tupla en

una relación será imposible si existen tuplas de otras relaciones asociadas a ésta, cuyas claves ajenas se correspondan con su clave primaria.

> *Así, no se podría, por ejemplo, eliminar un departamento si tiene empleados asociados a él, ni tampoco se podría cambiar su código si ya tenía empleados; por tanto, únicamente podrán realizarse esas operaciones en departamentos sin empleados asociados.*

- **Borrado y/o modificación con puesta a nulos (SET NULL en SQL):** Si el usuario especifica esta restricción, el borrado o modificación de una tupla en una relación ocasiona la puesta a NULL de la clave ajena de las tuplas de otras relaciones asociadas a ésta, cuyas claves ajenas se correspondían con su clave primaria.

- **Borrado y/o modificación con puesta a valor por defecto (SET DEFAULT en SQL):** Si el usuario especifica esta restricción, el borrado o modificación de una tupla en una relación ocasiona la puesta al valor por defecto especificado de la clave ajena de las tuplas de otras relaciones asociadas a ésta, cuyas claves ajenas se correspondían con su clave primaria.

- **Restricción de validación (CHECK).** Esta restricción permite al usuario especificar condiciones que deban cumplir los valores de los atributos. Cada vez que se realiza una inserción o una actualización de datos se comprueba si los valores cumplen la condición, rechazando la operación si no la cumplen.

```
CREATE TABLE emp(
empno NUMBER(4) PRIMARY KEY,
enombre VARCHAR(20),
puesto VARCHAR(15),
fecha DATE,
sal NUMBER(7),
comm NUMBER(7),
deptno NUMBER(2),
FOREIGN KEY (deptno) REFERENCES dept,
CONSTRAINT ck_puesto CHECK (puesto IN ('ADMINISTRATIVO',
'VENDEDOR', 'GERENTE', 'ANALISTA', 'PRESIDENTE')),
CONSTRAINT ck_comm1 CHECK ((comm IS NULL AND puesto!='VENDEDOR')
OR (comm IS NOT NULL AND puesto = 'VENDEDOR' AND comm>=0)),
CONSTRAINT ck_comm2 CHECK (comm<=2*sal));
```

- **Restricción de aserción (ASSERTION):** Igual que CHECK pero puede afectar a 2 o más relaciones, por tanto, la condición a cumplir se establece sobre campos de distintas relaciones. Pueden implicar a subconsultas en la condición.

```
Solo puede existir un presidente en la empresa.

CREATE ASSERTION as_presidente CHECK
( (SELECT COUNT(*) FROM emp WHERE puesto='PRESIDENTE') < 2);
```

- **Disparadores (TRIGGER):** Las restricciones anteriores son declarativas, sin embargo este tipo es procedimental. El usuario podrá programar una serie de acciones distintas ante una determinada condición.

Ejemplo: Un disparador que puede realizar el usuario para auditar las operaciones de modificación y borrado de datos de la tabla EMPLE, de forma que cada vez que se realiza una operación de actualización o borrado se inserta en la tabla AUDITAEMPLE una fila que contendrá varios campos: fecha y hora de la operación, número y apellido del empleado afectado, y la operación que se realiza.

```
CREATE OR REPLACE TRIGGER auditar_act_emp
BEFORE INSERT OR DELETE ON EMPLE FOR EACH ROW
BEGIN
IF DELETING THEN
INSERT INTO AUDITAREMPLE VALUES
(TO_CHAR(sysdate,'DD/MM/YY*HH24:MI*')   ||   :OLD.EMP_NO||   '*'
|| :OLD.APELLIDO || '*BORRADO');
ELSIF INSERTING THEN
INSERT INTO AUDITAREMPLE VALUES
(TO_CHAR(sysdate,'DD/MM/YY*HH24:MI*')   ||   :NEW.EMP_NO||   '*'
|| :NEW.APELLIDO || '*INSERCION');
END IF;
END;
```