Unidad 3 : Diseño Lógico: Modelo relacional

- 1. Introducción
  - 1.1. Historia
- 2. Estructura del modelo relacional
  - 2.1. Relaciones
  - 2.2. Dominios
  - 2.3. Claves
    - Clave candidata
    - Clave primaria
    - Clave alternativa
    - Clave externa, ajena o foránea
- 3. Restricciones del Modelo Relacional
  - 3.1. Inherentes al modelo
  - 3.2. Semánticas o de usuario
    - 3.2.1. Restricción de clave primaria o principal (**primary key**)
    - 3.2.2. Restricción de unicidad (**unique)**
    - 3.2.3. Obligatoriedad (**not null**)
    - 3.2.4. Clave alternativa (**alternate key**)
    - 3.2.5. Integridad referencial (**foreign key**)
      - Políticas de actualización y eliminación
    - 3.2.6. Regla de validación (check)
- 4. Transformación del MER al Modelo Relacional
  - 4.1. Entidades y atributos.
  - 4.2. Relaciones binarias.
    - 4.2.1. Relaciones binarias N:M
    - 4.2.2. Relaciones binarias 1:N
    - 4.2.3. Relaciones binarias 1:1
  - 4.3. Relaciones Reflexivas.
  - 4.4. Relaciones ternarias.
  - 4.5. Entidades débiles.
  - 4.6. Relaciones de jerarquía.
  - 4.7. Restricciones externas
- 5. Normalización
  - 5.1. Teoría de la normalización
    - 5.1.1. Primera Forma Normal (1FN)
    - 5.1.2. Segunda Forma Normal: 2FN
    - 5.1.3. Tercera Forma Normal: 3FN
    - 5.1.4. Forma Normal de Boyce-Codd: FNBC
    - 5.1.5. Ejemplo del proceso de normalización

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Fecha	Versión	Descripción
30/09/2022	1.0.2	Versión completa y corregida

Unidad 3 : Diseño Lógico: Modelo relacional

1. Introducción

Ya vimos en la primera unidad que los SGBD se pueden clasificar de acuerdo a diversas características, siendo la clasificación más importante dela basada en el modelo lógico, (Jerárquico, en Red, Relacional, Orientado a Objetos, ...).

El Modelo Relacional es el modelo de representación que siguen la gran mayoría de los SGBD relacionales (MySQL, SQL Server, Oracle, PostgreSQL, Ms Access, entre otros) en la actualidad, puesto que es el modelo de datos más extendido.

En el modelo relacional los datos y sus relaciones se representan a través de tablas, y los atributos se traducen en campos de esas tablas.

Nuestro objetivo consistirá en transformar (o traducir) nuestro Modelo Entidad-Relación a un Modelo Relacional si queremos crear nuestra Base de Datos en algún SGBD relacional.

1.1. Historia

Durante los primeros años de las bases de datos, se utilizó el **modelo jerárquico** como la primera forma de describir de forma más humana una base de datos. Después reinó el **modelo en red** especialmente en su norma **Codasyl**. Así a principios de los 70 parecía que el modelo a aplicar al implementar bases de datos sería Codasyl y lo sería por muchos años.

Sin embargo, **Edgar Frank Codd** definió las bases del modelo relacional a finales de los 60. En 1970 publica el documento *"A Relational Model of data for Large Shared Data Banks"* ("*Un modelo relacional de datos para grandes bancos de datos compartidos*"). Actualmente se considera que ese es uno de los documentos más influyentes de toda la historia de la informática. Lo es porque en él se definieron las bases del llamado **Modelo Relacional de Bases de Datos**. Anteriormente el único modelo teórico estandarizado era el modelo **Codasyl** que se utilizó masivamente en los años 70 como paradigma del modelo en red de bases de datos.

Codd se apoya en los trabajos de los matemáticos **Cantor** y **Childs (**cuya teoría de conjuntos es la verdadera base del modelo relacional). Según Codd, los datos se agrupan en **relaciones** (actualmente llamadas **tablas**), las cuales son una estructura que aglutina datos referidos a una misma entidad de forma organizada. Las relaciones, además, estructuran los datos de forma independiente respecto a su almacenamiento real en la computadora. Es decir, es un elemento conceptual, no físico.

Lo que Codd intentaba fundamentalmente es evitar que las usuarias y usuarios de la base de datos tuvieran que verse obligadas a aprender los entresijos internos del sistema. Esto es lo que ocurría con el modelo en red, dominante cuando Codd diseñó el modelo relacional, que era bastante *físico*. Su enfoque fue revolucionario al evitar conceptos del mundo de la computación en su modelo.

Aunque trabajaba para **IBM**, esta empresa no recibió de buen grado sus teorías. De hecho, IBM continuó trabajando en su sistema gestor de bases de datos en red **IMS**. Fueron otras empresas (en especial **Oracle**) las que implementaron sus teorías.

Pocos años después, el modelo se empezó a utilizar cada vez más hasta, finalmente, ser el modelo de bases de datos más popular. Hoy en día casi todas las bases de datos siguen este modelo, aunque en estos años han aparecido rivales cada vez más fuertes en las llamadas bases de datos **NoSQL**, que han demostrado un gran eficacia en bases de datos que necesitan una enorme cantidad de instrucciones de modificación por

minuto.

Los objetivos del Modelo Relacional son:

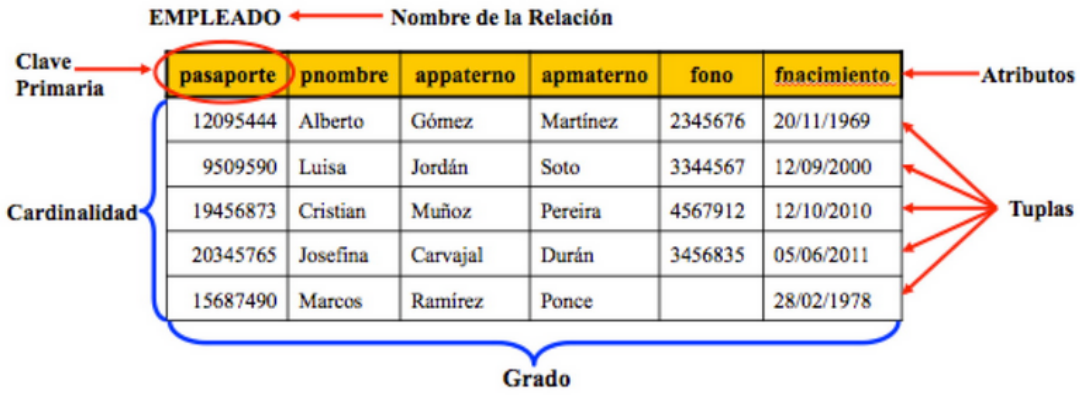
- **Independencia física**, para que la manera de guardar los datos no influya en su manipulación lógica, y así los usuarios que acceden a estos datos no tengan que modificar sus programas por cambios en el almacenamiento físico.
- **Independencia lógica**, para que las vistas externas no se vean afectadas por cambios en el esquema conceptual de la B.D.
- **Flexibilidad**, para poder ofrecer los datos a cada usuario de la forma más adecuada a su aplicación.
- **Uniformidad**, las estructuras lógicas de los datos presentan un aspecto simple y uniforme (las mesas), cosa que facilita la concepción y manipulación por parte de los usuarios.
- **Sencillez**, las características anteriores, unidas a unos lenguajes de usuario sencillos, hacen que el M. Relacional sea fácil de entender y de utilizar por el usuario final.

2. Estructura del modelo relacional

La estructura fundamental del modelo relacional es precisamente esa, la «**relación**», es decir una tabla bidimensional constituida por **filas** (registros o tuplas) y **columnas** (atributos o campos).

Las relaciones (o tablas) representan las entidades del MER mientras que las columnas de la relación representarán las propiedades o atributos de dichas entidades.

Por ejemplo, si en la base de datos se tienen que representar la entidad EMPLEADO, está pasará a ser una relación o tabla llamada «EMPLEADO», cuyos atributos describen las características de los empleados (tabla siguiente). Cada tupla o registro de la relación «EMPLEADO» representará un empleado concreto.



pasaporte	pnombre	appaterno	apmaterno	fono	fnacimiento
12095444	Alberto	Gómez	Martínez	2345676	20/11/1969
9509590	Luisa	Jordán	Soto	3344567	12/09/2000
19456873	Cristian	Muñoz	Pereira	4567912	12/10/2010
20345765	Josefina	Carvajal	Durán	3456835	05/06/2011
15687490	Marcos	Ramírez	Ponce		28/02/1978

2.1. Relaciones

El elemento fundamental del modelo es lo que se conoce como **relación**. La forma de representar relaciones es mediante **tablas** regulares en las que las columnas se corresponden con los atributos y las filas con las tuplas (tal y como puede verse en la imagen anterior). Esta forma es más visual y entendible.

📌 `<!--No hay que confundir la idea de relación según el modelo de **Codd**, con lo que significa una relación en el modelo Entidad/Relación de **Chen**. No tienen nada que ver-->`

Las relaciones constan de:

- **Atributos**. Es cada una de las propiedades de los datos de la relación (*pasaporte, pnombre,...*). Las relaciones representan conjuntos de objetos o elementos reales, cada atributo es propiedad o

característica de dicho elemento.
- **Tuplas.** Referido a cada ejemplar, objeto o elemento de la relación. Por ejemplo si una relación almacena empleados, una tupla representaría a un empleado en concreto.

Además hay que tener en cuenta conceptos como :

- **Grado**: Indica el tamaño de una relación en base al número de columnas (atributos) de la misma. Lógicamente cuanto mayor es el grado de una relación o tabla, mayor es su complejidad de manejo.
- **Cardinalidad**: Número de tuplas de una relación, o número de filas de una tabla. Hay tablas que pueden tener una enorme cardinalidad: cientos, miles e incluso millones de filas.
- **Valor**: Está representado por la intersección entre una fila y una columna. Por ejemplo, en la tabla anterior, serían valores *"Alberto", "Durán", "9509590", "28/02/1978"*, ...
- **Valor nulo**: Representa la ausencia de información.

Las relaciones tienen las siguientes **propiedades**:

- Cada relación tiene un **nombre** y éste es distinto del nombre de todas las demás relaciones de la misma BD.
- Los valores de los atributos son **atómicos**: en cada tupla, cada atributo toma un solo valor.
- No hay dos **atributos que se llamen igual** en la misma relación.
- El orden de los **atributos** es irrelevante; **no están ordenados**.
- Cada tupla es distinta de las demás: **no hay tuplas duplicadas**. (Como mínimo se diferenciarán en la clave principal)
- Al igual que los atributos, el orden de las tuplas no importa: **las tuplas no están ordenadas**.

Los términos vistos tienen distintos sinónimos según la nomenclatura utilizada. A ese respecto se utilizan tres nomenclaturas:

Terminología Relacional		Terminología de Tablas		Terminología de Archivo
Relación	=	Tabla	=	Archivo
Tupla	=	Fila	=	Registro
Atributo	=	Columna	=	Campo
Grado	=	Número de columnas	=	Número de campos
Cardinalidad	=	Número de filas	=	Número de registros

2.2. Dominios

Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio.

Un **dominio** está formado por un conjunto finito de valores **homogéneos** (del mismo tipo) y **atómicos** (indivisibles) que puede tomar cada atributo. Los valores contenidos en una columna pertenecen a un dominio que previamente se ha definido.

Todos los dominios tienen un nombre (así podemos referirnos a ese nombre en más de un atributo) y un tipo de datos asociado.

Por ejemplo si tenemos un atributo llamado Fecha_nac que sirve para almacenar fechas de nacimiento, podremos entender que el valor "Hola" no tiene sentido en ese atributo. Técnicamente se dice que Hola no pertenece al dominio de las fechas. Sí pertenece el valor: 3/9/1982. Una expresión como 34/3/1982 tampoco pertenece al dominio, no es válida (no hay meses de 34 días). Es decir, un dominio expresa de forma inequívoca los valores posibles dentro de un atributo.

📌 `<!--Se confunde la idea entre **tipo de datos** y dominio. Lo cierto es que son conceptos semejantes, pero no iguales. La diferencia es que el dominio impone más restricciones que los tipos de datos.-->`

Por ejemplo de dominio: el que hace que un valor para el atributo dni sólo se considere perteneciente a su dominio si tiene ocho números, una letra y además la letra cumple una regla matemática concreta sobre los números.

La forma de indicar un dominio se puede hacer utilizando dos posibles técnicas:

- **Intensión.** Se define el domino indicando la definición exacta de sus posibles valores. Por intensión se puede definir el dominio de edades de los trabajadores como: *números enteros entre el 16 y el 65* (un trabajador sólo podría tener una edad entre 16 y 65 años).
- **Extensión**. Se indican algunos valores y se sobreentiende el resto gracias a que se autodefinen con los anteriores. Por ejemplo el dominio localidad se podría definir por extensión así: *Palencia, Valladolid, Villamuriel de Cerrato,...*

Principalmente existen dos tipos de dominios:

- ***Dominios Generales***: son aquellos cuyos valores están comprendidos entre una máximo y un mínimo. Por ejemplo, *codigo_postal*, formado por todos los números enteros positivos de cinco cifras.
- ***Dominios restringidos***: son aquellos que solo pueden tomar un conjunto de valores. Por ejemplo, *sexo*, solamente puede tomar los valores H y M.

Atributo	Nombre del Dominio	Descripción	Definición
noficina	NUM_OFICINA	Posibles valores de número de oficina	3 caracteres, rango 100 - 990
calle	NOM_CALLE	Nombres de calles y numero de Santiago donde se ubica la oficina	25 caracteres
area	NOM_AREA	Área de Santiago en la que se encuentra ubicada la oficina	20 caracteres
telefono	NUM_TEL_FAX	Números de teléfono de Santiago	9 caracteres
fax	NUM_TEL_FAX	Números de teléfono de Santiago	9 caracteres

2.3. Claves

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. Así, toda fila debe estar asociada con una **clave** que permite identificarla.

A veces las filas se pueden idenrficar con un mismo atributo, pero otras veces es necesario recurrir a más de un atributo.

Clave candidata

Conjunto de atributos que permiten identificar en forma única cada tupla de la relación. Es decir columnas cuyos valores no se repitenen ninguna otra fila de esa tabla.

Toda tabla, en el modelo relacional, debe terner al menos una clave candidata, pero puede haber muchas más.

Por ejemplo, los atributos candidatos para una tabla de CIENTE serían el *SIP*, el *Id_cliente* o el *NIF*.

Clave primaria

Clave candidata que se escoge como **identificador** de las filas de la tabla. Se elige como primaria la candidata que identifique mejor a cada tupla en el contexto de la base de datos. Además se debe dar prioridad a atributos cuyos dominios sean más eficientes de cara al tamaño que ocupan en disco y a la facilidad de proceso por parte de la máquina.

Por ejemplo un atributo con el *NIF* sería clave candidata de una tabla de CIENTE, aunque si en esa relación existe un atributo *Id_cliente*, este sería mejor candidato para clave principal, porque es mejor identificador para ese contexto.

Las claves primarias son los únicos datos en el modelo relacional que provocan redundancia ya que se duplican en las claves secundarias para establecer las relaciones entre las tablas. Por ello hay que intentar que las claves primarias contengan datos pequeños, nunca textos largos y de tamaño variable (como nombres, títulos,...).

Buenos tipos de datos para claves primarias son las fechas, números enteros y textos, si son cortos y de tamaño fijo.

Clave alternativa

Cualquier clave candidata que no sea primaria y que también puede identificar de manera única una tupla. Al momento de crear la relación como tabla en la Base de Datos se debe definir una restricción de tipo UNIQUE.

Clave externa, ajena o foránea

Atributo cuyos valores coinciden con una clave primaria de otra tabla. Lo entenderemos mejor en el siguiente apartado cuando expliquemos la restricción de integridad.

3. Restricciones del Modelo Relacional

En todos los modelos de datos existen restriciones que a la hora de diseñar una base de datos se han de tener en cuenta. Los datos almacenados en la base de datos han de adaptarse a las estructuras impuestas por el modelo y debe cumplir una serie de reglas para garantizar que son correctas.

3.1. Inherentes al modelo

Son aquellas que no requieren que se establezcan de forma explícita, sino que son definidas por el propio hecho de que la base de datos sea relacional. Las más importantes son:

- **IDENTIDAD**: No puede haber dos filas iguales
- El **orden** de las filas y columnas no es significativo
- **UNICIDAD**: Cada atributo sólo puede tomar un valor del dominio al que pertenece, en la intersección entre fila y columna.
- **VALOR NO NULO**: Ningún atributo que forma parte de la clave primaria puede tomar un valor nulo.

3.2. Semánticas o de usuario

El modelo relacional permite incorporar restricciones personales a las tablas. Son las más importantes y son fundamentales para que la información de la base de datos sea coherente y eficiente.

Se comentan a continuación las principales restricciones semánticas.

3.2.1. Restricción de clave primaria o principal (primary key)

Marca uno o más atributos (una o más columnas) como identificadores de la tabla. En el modelo relacional, toda tabla requiere de un identificador o clave principal para asegurar que todas las filas son diferentes en cada tabla.

Esta restricción (además de marcar los datos identificativos de una tabla) prohíbe que las columnas que forman la clave primaria puedan contener valores repetidos en distintas filas o que queden vacías (nulos) en alguna fila. Es decir el contenido de las claves primarias es único y distinto de nulo.

3.2.2. Restricción de unicidad (unique)

Impide que los valores de los atributos marcados de esa forma, puedan repetirse en distintas filas. Es decir, en esa columna los valores deben ser distintos para cada fila, o bien quedar vacíos.

3.2.3. Obligatoriedad (not null)

Prohíbe que el atributo marcado de esta forma quede vacío (es decir impide que pueda contener el valor nulo, **null**) en alguna fila. También se debe de indicar en las columnas que son clave alternativa.

3.2.4. Clave alternativa (alternate key)

Como ya se ha explicado antes, las claves candidatas que no se eligen para ser claves principales se marcan como clave alternativa. En casi todos los sistemas gestores de bases de datos no es posible marcar claves alternativas de forma explícita.

Sin embargo, si una columna es una clave alternativa, no podrá repetir valores ni quedar vacía. Es decir, las claves alternativas se marcan con restricciones de unicidad (*unique*) y de obligatoriedad (*not null*). Esto significa que si detectamos claves alternativas habrá que marcar estas dos restricciones sobre ellas.

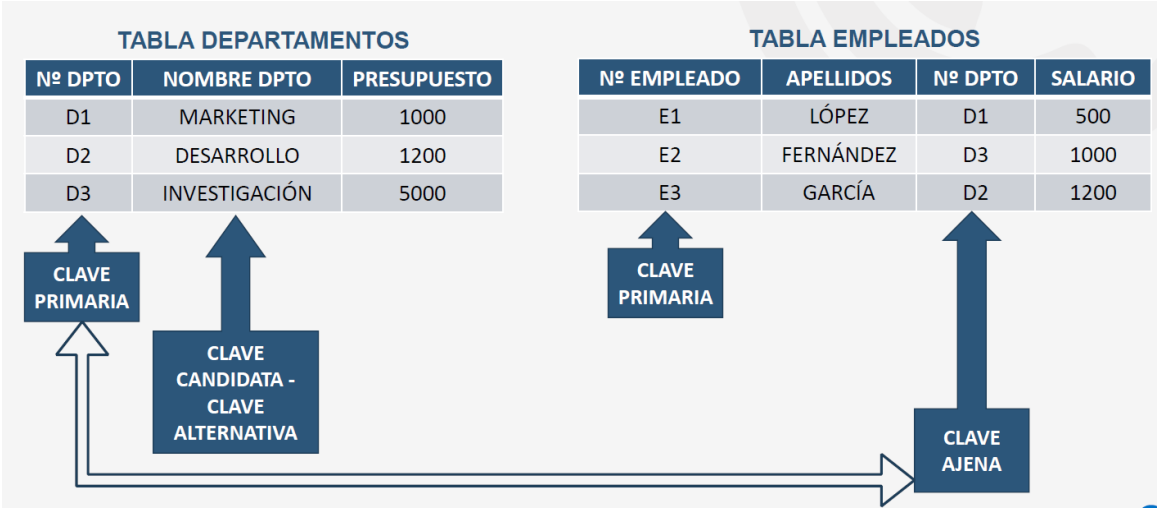
3.2.5. Integridad referencial (foreign key)

Las claves ajenas (foraneas o externas) es el mecanismo del modelo relacional para poder asociar datos de diferentes tablas.

La restricción de integridad referencial esta ligada a las claves ajenas y lo que implica es que las columnas marcadas como claves ajenas (foreign keys) no puedan contener valores que no se puedan relacionar con las clave principal de la tabla que relacionan (llamada tabla principal). Sin embargo, sí se permite dejar nulas estas columnas.

Por ejemplo:

Tenemos la tabla DEPARTAMENTOS y la tabla EMPLEADOS con la información de los empleados incluido el deparatamento en el que trabajan.



Si intentamos añadir un nuevo empleado a la tabla indicando en la columna Nº DPTO (marcada con una restricción de integridad o *foreign key*) el valor **D4**, no nos lo permitirá, solo se podrán utilizar números de departamentos que existan en la tabla de DEPARTAMENTO.

Es muy importante esta restricción porque genera bases de datos muy coherentes. Bien es cierto que, a veces, incomoda porque obliga a una forma muy rígida de trabajar y, además, con relaciones muy complejas el relleno es francamente difícil. Pero, es muy conveniente y, de hecho, obligatoria en el modelo relacional.

Se podría decir que es la **restricción más importante** del modelo relacional, y la base del mismo.

Políticas de actualización y eliminación

La restricción de 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 haciendo referencia a un valor que ya no existe, y eso la propia restricción no lo permite.

Es decir no podemos cambiar la clave o eliminar clientes que tengan alquileres relacionados. Es decir si en el ejemplo anterior, borramos la fila en clientes que hace referencia al cliente que se llama *Josu*, la base de datos reaccionará prohibiendo esta operación ya que hay dos alquileres relacionados con ese cliente. Lo mismo ocurre si intentamos cambiar el código de cliente de *Josu*.

Para solventar esta situación se utilizar **políticas** especiales cuando creemos la restricción en la clave secundaria.

- **Prohibir la operación** (*no action)*. Esto no permitiría hacer en las claves principales ninguna operación si hay claves secundarias relacionadas con ellas. Suele ser la opción por defecto. Es muy rígida.
- **Transmitir la operación en cascada** *(cascade*). Es decir si se modifica o borra un departamento, también se modificarán o borrarán los empleados que trabajan en él.
- **Colocar nulos** *(set null*). Si modificamos o borramos un departamento, sus empleados marcarán el antiguo código de ese departamento como nulo. Por ejemplo si borramos al departamento **D3** el trabajador **E2** indicaran como número de departamento el valor *null*.
- **Usar el valor por defecto** *(default).* Se colocan un valor por defecto en las claves ajenas relacionadas cuando se borre o modifique la clave primaria relacionada. Este valor por defecto se indica al crear la tabla (opción **default**).

No todas las bases de datos admiten todas estas posibles políticas a aplicar en operaciones de actualizar o eliminar. Además, podemos indicar una política al actualizar y otra al eliminar. Es decir podremos, por ejemplo, indicar nulos ante cambios en las claves primarias y actuar en cascada ante eliminaciones en las claves.

3.2.6. Regla de validación (check)

Es una restricción que impone una condición lógica que deberá de cumplir una o más columnas cuando se la añadan o modifiquen los datos. Por ejemplo, podríamos restringir la columna llamada *sueldo* para que siempre acepte valores mayores de *1000*; no se permitiría entonces indicar sueldos menores de 1000 en dicha columna.

A veces las reglas implican a varias columnas, como por ejemplo que la *fecha de inicio* sea mayor que la *fecha final*.

4. Transformación del MER al Modelo Relacional

La traducción de un MER a un modelo relacional se puede llevar a cabo, en gran parte, siguiendo una serie de reglas o pautas, que veremos a continuación.

De este modo continuamos el proceso de diseño de una Base de datos. En la UD2 hemos aprendido a hacer el esquema en el Modelo Entidad-Relación. Ahora lo traduciremos al Modelo Relacional, y ya se podrá implementar en cualquier SGBD Relacional.

Faltaría solo el proceso de Normalización (que veremos en el siguiente apartado) para acabar de dejar las tablas perfectamente diseñadas. De todas maneras las Bases de datos que diseñamos nosotros, con el proceso descrito anteriormente, tendrán unas tablas muy "normalizadas", siempre que diseñamos bien.

4.1. Entidades y atributos.

Toda entidad se transformará en una tabla, y todos sus atributos (que se considerarán como simples) se transformaran en las columnas dentro de la tabla. El identificador único (ya sea un solo atributo o un conjunto de atributos) se convierte en la clave principal, y lo denotaremos subrayándolo o con la indicación: **PK (Primary Key)**. Si hay clave alternativa esta se pone en «negrita».

> **NOTA**: **Las entidades débiles las estudiaremos mejor más adelante.**

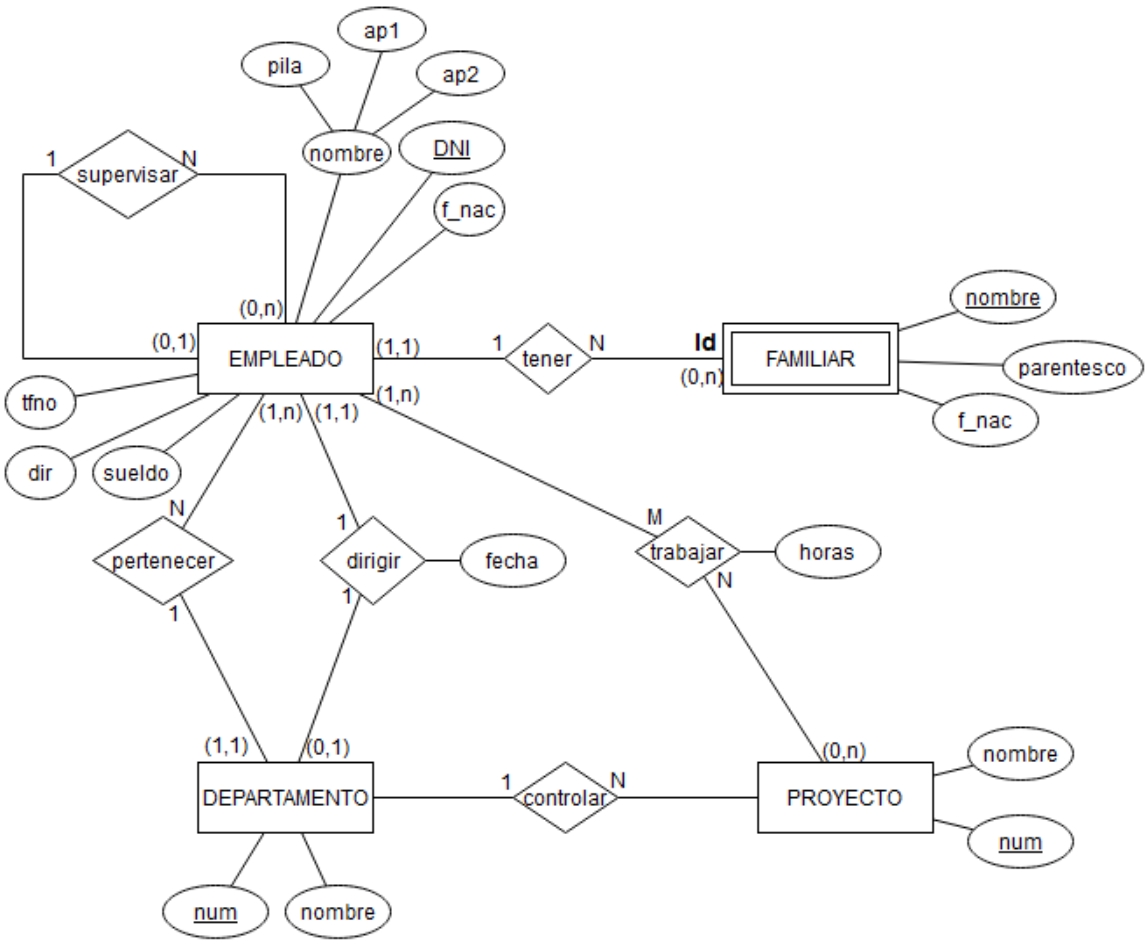
Representaremos la entidad con el nombre de la tabla en mayúsculas seguido, entre paréntesis, en minúsculas y separados por comas, por los nombres de los campos, con la clave principal subrayada. También es conveniente huir de los caracteres especiales (vocales acentuadas, ç, *ñ, guión, ...) por no tener problemas cuando vayamos a implementarla en un SGBD determinado (Access, ORACLE, PostgreSQL, MySQL ...). Para una mejor lectura intentaremos poner siempre la clave principal al principio, del o los primeros campos. A continuación iremos definiendo el resto de atributos que necesitamos.

```
TABLA(Campo_1, Campo_2, Campo_3, ….)
    PK = Campo_1
```

También podemos utilizar una forma alternativa de representarla, con un recuadro que coge toda la tabla, arriba el nombre de la tabla, y abajo cada uno de los campos, poniendo la clave principal en negrita o subrayada.

NOMBRE_TABLA	
PK	Campo_1
	Campo_2
	Campo_3

En nuestro ejemplo, como teníamos 4 entidades, nos saldrán de momento 4 tablas.




```
EMPLEADO (dni, nombre, dir, tfno, sueldo, f_nac)
    PK = dni
DEPARTAMENTO num, nombre)
    PK = num
PROYECTO (num, nombre)
    PK = num
FAMILIAR (nombre, parentesco, f_nac)
    PK = nombre
```

En el caso de que hubiera algún atributo que fuera *no nulo* (no es el caso del ejemplo), por ejemplo el parentesco en la entidad FAMILIAR, la forma de representarlo sería la siguiente:

```
FAMILIAR (nombre, parentesco, f_nac)
    PK = nombre
    VNN = parentesco
```

4.2. Relaciones binarias.

4.2.1. Relaciones binarias N:M

Es el caso más sencillo. **Siempre generan tabla**.

Se crea una tabla que incorpora como claves ajenas o foráneas **FK (Foreign Key)** cada una de las claves primarias de las entidades que participan en la relación. La clave primaria de esta nueva tabla está compuesta por dichos campos. Es importante resaltar que no se trata de 2 claves primarias, sino de una clave primaria compuesta por 2 campos.

Si hay atributos propios, pasan a la tabla de la relación. Se haría exactamente igual si hubiera participaciones mínimas 0.

Orden de los atributos en las claves compuestas: Se deben poner a la izquierda todos los atributos que forman la clave. El orden de los atributos que forman la clave vendrá determinado por las consultas que se vayan a realizar. Las tuplas de la tabla suelen estar ordenadas siguiendo como índice la clave. Por tanto, conviene poner primero aquel/los atributos por los que se va a realizar la consulta.

En nuestro caso de ejemplo la relación TRABAJAR entre EMPLEADO y PROYECTO, es una relación binaria del tipo M:N. La traducción quedaría del siguiente modo:

```
TRABAJAR (dni, num, horas)
    PK = dni, num
    FK = dni --> EMPLEADO (dni)
    FK = num --> PROYECTO (num)
```

Como observamos, las claves ajenas se definen tras la definición de clave primaria (PK), con la indicación: **FK (Foreign Key)**. También será necesario indicar la tabla con la que se relaciona y entre paréntesis el atributo clave primaria de dicha tabla.

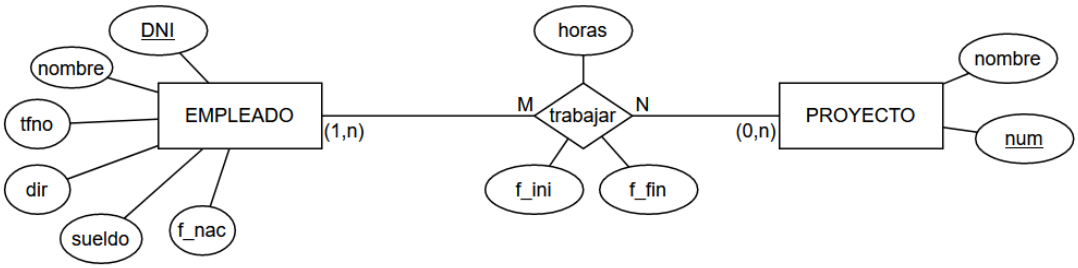
En el caso anterior, sería conveniente modificar el nombre del atributo *num* por *n_proyecto* para diferenciarlo de otros atributos *num* quedando la definición de la tabla de la siguiente forma:

```
TRABAJAR (dni, n_proyecto, horas)
    PK = dni, n_proyecto
    FK = dni --> EMPLEADO (dni)
    Fk = n_proyecto --> PROYECTO (num)
```

Cada vez que nos encontramos con una relación M:N y la traducimos por una nueva tabla tendremos que preguntarnos si es suficiente con la clave principal formada por las dos claves ajenas, o si hace falta añadir otro campo. Tenemos que ser conscientes que la clave principal no pueda repetirse, que haya 2 filas con la misma clave principal.

Así, en el ejemplo, nos tendríamos que hacer la siguiente pregunta: puede un empleado trabajar en el mismo proyecto más de una vez? En este caso la contestación es negativa, y por tanto es suficiente con esta clave principal.

Pero suponemos que la respuesta es que **sí** que puede trabajar más de una vez a lo largo del tiempo. En este caso sería una especie de histórico, donde haría falta, además, saber cuando empieza y cuando acaba de trabajar en un proyecto un determinado trabajador (serían atributos de la relación):



Entonces, como no es suficiente con la clave primaria formada por las dos claves ajenas, incluiremos otro campo en la clave primaria. Parece que el más adecuado sería *f_ini* (ya no se puede dar el caso que el mismo trabajador trabaje más de una vez en el mismo proyecto, empezando el mismo día). La traducción al modelo relacional seria la siguiente:

```
TRABAJAR (dni, n_proyecto, f_ini, f_fin, horas)
    PK = dni, n_proyecto, f_ini
    FK = dni --> EMPLEADO (dni)
    FK = n_proyecto --> PROYECTO (num)
```

Pero por otro lado, las claves primarias muy largas (formadas por muchos atributos) no son operativas, y aunque la traducción literal sea cómo hemos dicho, por motivos prácticos podemos cambiar la clave primaria. Consideraremos que el número máximo de campos en la clave principal es de 3. 4 ya son demasiado, y entonces buscaremos otra clave principal (un código de trabajo, por ejemplo). Las claves externas continuarían siendo las mismas.

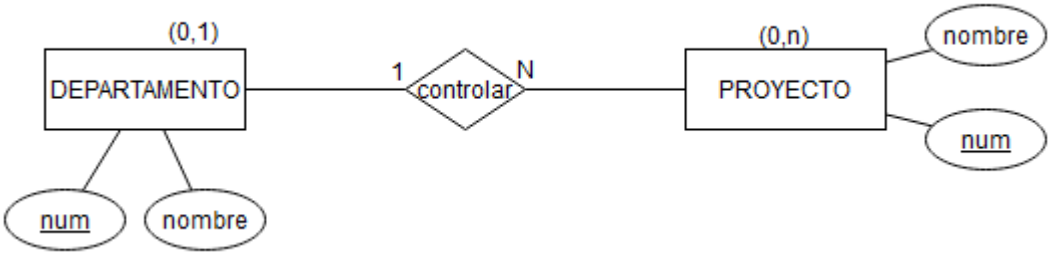
4.2.2. Relaciones binarias 1:N

Este tipo de relaciones binarias no requieren ser transformadas en una tabla en el modelo relacional.

En su lugar la tabla del lado *muchos* (**tabla relacionada**) incluye como clave ajena el identificador de la entidad del lado *uno* (**tabla principal**). Es lo que se denomina "propagación de clave".

También se incluirán en dicha tabla, todos los atributos de la relación.

En nuestro caso de ejemplo la relación CONTROLAR entre DEPARTAMENTO y PROYECTO, es una relación binaria del tipo 1:N.



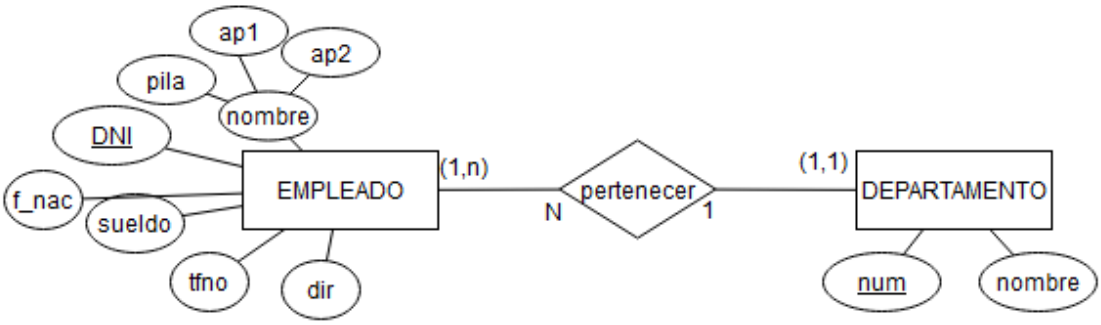
La traducción quedaría del siguiente modo:

```
DEPARTAMENTO (num, nombre)
    PK = num
PROYECTO (num, nombre, n_dpto)
    PK = num
    FK = n_dpto --> DEPARTAMENTO (num)
```

Como vemos, se ha incluido la clave primaria de DEPARTAMENTO (se ha modificado el nombre del atributo por *n_dpto* para diferenciarlo de la clave primaria de PROYECTO), y se ha definido como clave ajena (**Foreign Key - FK**)

La cardinalidad mínima hay que revisarla bien ya que en caso de que la clave externa se relacione obligatoriamente con la principal (cardinalidad mínima de uno), habrá que indicar una restricción de tipo NOT NULL para indicar esa situación.

En nuestro caso de ejemplo la relación PERTENECER entre DEPARTAMENTO y EMPLEADO, es una relación binaria del tipo 1:N, pero con cardinalidad mínima de 1.



La traducción, en este caso, quedaría del siguiente modo:

```
DEPARTAMENTO (num, nombre)
    PK = num
EMPLEADO (dni, nombre, dir, tfno, sueldo, f_nac, n_dpto)
    PK = dni
    FK = n_dpto --> DEPARTAMENTO (num)
    VNN = n_dpto
```

Como vemos, se ha incluido la clave primaria de DEPARTAMENTO (se ha modificado el nombre del atributo por *n_dpto* para identificarlo mejor), y se ha definido como clave ajena (**Foreign Key - FK**). Además, al tener cardinalidad mínima, estamos obligando que todo empleado pertenezca a un departamento. Esta "obligatoriedad" se representa con la restricción **NOT NULL** (**VNN**) sobre el atributo *n_dpto*.

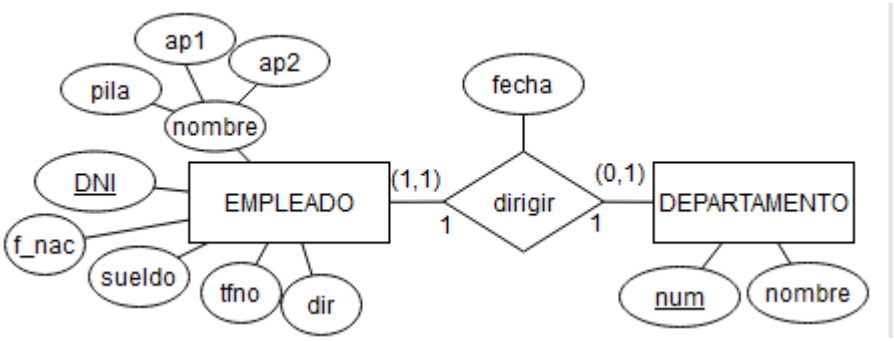
4.2.3. Relaciones binarias 1:1

No hay una forma única de traducir este tipo de relaciones. Existen tres posibles traducciones, según las cardinalidades mínimos de las entidades que participan en la relación, aunque también debe primar el "sentido común".

Por regla general se actúa de igual forma que en las relaciones anteriores, solo que la propagación de clave puede ir en cualquira de las dos entidades participantes.

Si de las dos entidades que participan en la relación, solo una de ellas tiene cardinalidad mínima 1, la propagación de clave se realizará hacía ella.

En nuestro caso de ejemplo la relación DIRIGIR entre DEPARTAMENTO y EMPLEADO, es una relación binaria del tipo 1:1, pero con cardinalidad mínima de 1.



La traducción, en este caso, quedaría del siguiente modo:

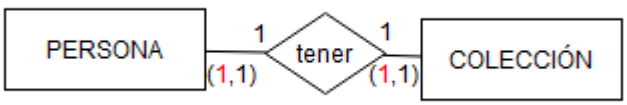
```
DEPARTAMENTO (num, nombre, director, fecha)
    PK = num
    FK = director --> EMPLEADO (dni)
    VNN = director
    UK = director
EMPLEADO (dni, nombre, dir, tfno, sueldo, f_nac)
    PK = dni
```

Como vemos, se ha incluido la clave primaria de EMPLEADO (se ha modificado el nombre del atributo por *director* para identificarlo mejor) en la tabla DEPARTAMENTO, y se ha definido como clave ajena (**Foreign Key - FK**). Además, al tener cardinalidad mínima, estamos obligando que todo empleado pertenezca a un departamento. Esta "obligatoriedad" se representa con la restricción **NOT NULL** (**VNN**) sobre el atributo *director*. También podríamos hacer que este atributo sea único (no se puede repetir, ha que si se repitiera ya no sería una relación 1:1, sino una relación 1:N), añadiendo la restricción de **UK** (**Unique Key**).

También hemos incluido en la relación EMPLEADO el atributo *fecha* que presentaba la relación DIRIGIR.

¿Porqué definimos la relación en la tabla DEPARTAMENTO? Porque es la presenta la obligatoriedad, es decir, es la que presenta cardinalidad mínima 1. Es decir, todos los departamentos tienen un director, pero no todos los empleados son directores. Si lo hiciéramos al revés, el atributo añadido (por ejemplo *dpt_dirigirdo*) aparecería muchas veces vacío, ya que son pocos los empleados que dirigen departamentos.

Si las dos entidades presentan cardinalidades mínimas de 1, se puede considerar todo (las dos entidades, la relación y los atributos) como una sola tabla, aunque en la práctica este caso será muy extraño, porque ya en la parte del diseño la habríamos considerado como una sola entidad.

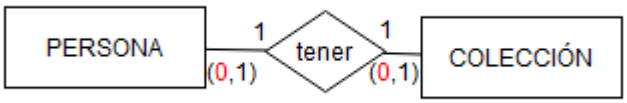


La traducción sería la siguiente:

```
PERSONA (dni, ...datos_persona ..., datos_colección ...)
```

De todas formas, puede que nos interese formar dos tablas para separar claramente los dos tipos de información. Si fuera el caso, la traducción se realizaría como en el caso anterior, con una clave ajena, no nula y única, en la tabla que menos se utilice.

Si ninguna de las dos entidades presenta cardinalidad mínima de 1, para evitar que la clave ajena definida en una de los dos tablas quede muchas veces vacia, podríamos crear una tercera tabla para la relación, igual que hacíamos en la relaciones binarias M:N.



La traducción sería la siguiente:

```
PERSONA (dni, ..)
    PK = dni
COLECCION (cod_col, ...)
    PK = cod_col
PROPIETARIO (dni, cod_col)
    PK = dni, cod_col
    FK = dni --> PERSONA (dni)
    FK = cod_col --> COLECCION (cod_col)
```

Pero como comentábamos al principio, tendremos que aplicar el sentido común, puesto que quizás una de las dos podría participar de forma "casi" total (por ejemplo, casi todas las colecciones son de una persona). Entonces podría ser mejor traducirlo cómo en el primer caso, poniendo la clave externa en la que participa de forma "casi" total, puesto que esta tendrá relativamente pocos valores nulos, y sería más costoso mantener otra mesa. Evidentemente la clave externa sí que podría ser *nula, en este caso.

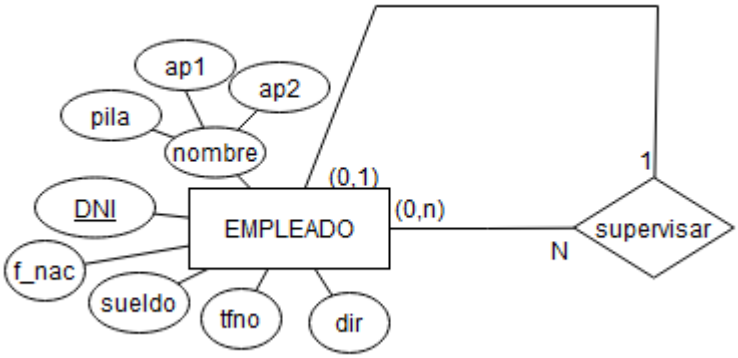
Resumiendo, una relación 1:1 casi siempre la traduciremos como una clave externa en la tabla que participa en la relación de forma total o casi total (o la que previsiblemente tiene más ocurrencias en la relación).

4.3. Relaciones Reflexivas.

Este tipo de relaciones se tratan como si fueran binarias, es decir, generan tabla o no en función de la cardinalidad.

- Si es 1:1 o 1:N, no se genera tabla. En la entidad se introduce dos veces la clave, una como clave primaria (**FK**) y otra como clave ajena (**PK**). Se suele introducir una modificación en el nombre por diferenciarlas.
- Si es N:N, la relación genera una nueva tabla. En la nueva tabla aparecerá la clave primaria de entidad dos veces, introduciendo una modificación para diferenciarlas. Las dos juntas formarán la clave primaria de la nueva tabla, y cada una de manera independiente será clave ajena a la entidad (cada una con un sentido propio según la lectura de la relación.)

En nuestro caso de ejemplo la relación SUPERVISAR entre EMPLEADO y EMPLEADO, es una relación reflexiva del tipo 1:N, con cardinalidades mínimas de 0. La traducción sería:




```
EMPLEADO (dni, nombre, dir, tfno, sueldo, f_nac, supervior)
    PK = dni
    FK = supervisor --> EMPLEADO (dni)
```

Como vemos, hemos añadido el atributo *supervisor*, como clave primaria de la entidad EMPLEADO que hace referencia a la relación SUPERVISAR (en principio sería *dni*, pero no podemos repetir el mismo nombre). Además, supervisor, pasa a ser la clave ajena que hace referencia a la entidad EMPLEADO (sería el *dni* del empleado que es su supervisor).

4.4. Relaciones ternarias.

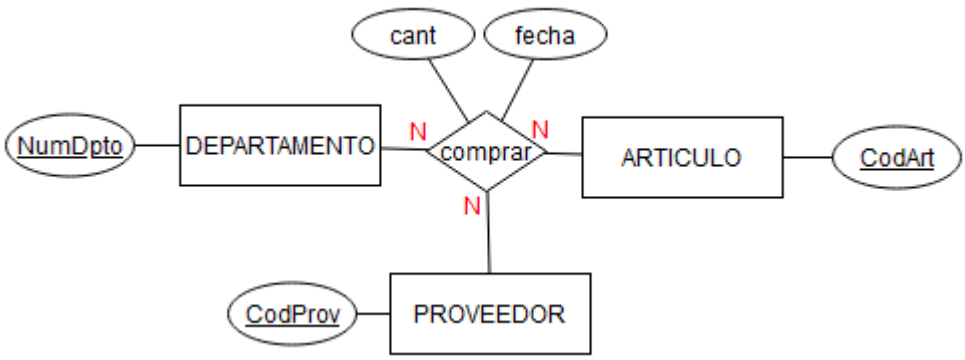
En una relación ternaria o superior construiremos una nueva tabla, donde incluiremos como claves ajenas, las claves primarias de todas las entidades, y además los atributos de la relación.

Habitualmente, la clave primaria de la nueva tabla será la combinación de todas las claves primarias de las entidades. Ocasionalmente, si alguna entidad participa con cardinalidad 1, la clave primaria de esta entidad no entraría a formar parte de la clave primaria de la nueva tabla.

Igual que en el caso de las relaciones M:N, nos tendremos que preguntar si es suficiente con la clave primaria generada o si se tendrá que incluir alguno otro campo.

En el caso de relaciones ternarias con cardinalidad 1:1:N, **no es necesario crear una tabla**. La entidad que participa como N recibe las claves de las dos entidades que participan como 1.

En el siguiente ejemplo de relación ternaria, suponiendo los atributos de la relación la fecha de compra y la cantidad:



La traducción sería la siguiente:

```
DEPARTAMENTO(NumDpto, ...)
    PK = NumDpto
ARTICULO (CodArt, ...)
    PK = CodArt
PROVEEDOR (Cod_prov)
    PK =
COMPRAR (NumDpto, CodArt, Cod_prov, fecha, cant)
    PK = NumDpto, CodArt, Cod_prov, fecha
    FK = NumDpto --> DEPARTAMENTO (NumDpto)
    FK = CodArt --> ARTICULO (CodArt)
    FK = Cod_prov --> PROVEEDOR (Cod_prov)
```

Hemos puesto todas las claves ajenas formando parte de la clave primaria, puesto que todas entran con cardinalidad N. Pero no teníamos bastante con esta clave principal, puesto que el mismo departamento puede comprar el mismo artículo al mismo proveedor más de una vez. Como no teníamos bastante, hemos puesto otro campo, fecha.

Sería momento, seguramente, de sustituir la clave principal que está formada por 4 campos, puesto que son demasiados.

Pondríamos otra clave principal, pero las claves externas continuarían siéndolo, y además serían no nulos, quedando la traducción de la siguiente manera:

```
DEPARTAMENTO(NumDpto, ...)
    PK = NumDpto
ARTICULO (CodArt, ...)
    PK = CodArt
PROVEEDOR (Cod_prov)
    PK =
COMPRAR (id, NumDpto, CodArt, Cod_prov, fecha, cant)
    PK = id
    FK = NumDpto --> DEPARTAMENTO (NumDpto)
    FK = CodArt --> ARTICULO (CodArt)
    FK = Cod_prov --> PROVEEDOR (Cod_prov)
    VNN = NumDpto, CodArt, Cod_prov
```

4.5. Entidades débiles.

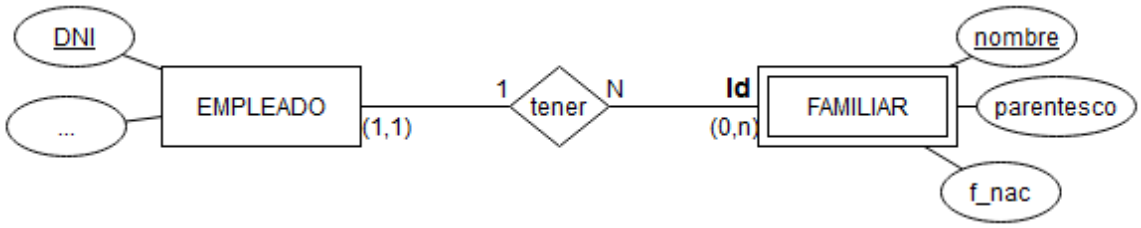
Las entidades débiles, dado que como mínimo dependen de otra, podremos ser más restrictivos. Una entidad siempre es débil a través, como mínimo, de una relación que la comunica con la entidad principal. Además participa de forma total (como que no puede existir sin la otra, toda ocurrencia está en la relación o lo que es lo mismo, posee cardinalidad mínima 1).

Por lo tanto, toda entidad débil tendrá una clave ajena, que apunta a la principal y será no nula. Pero todavía podemos ir más allá:

- **Dependencia en existencia**: haremos que la clave externa borro y actualizo en cascada, puesto que si deja de existir la principal no tiene sentido la débil.
- **Dependencia en identificación**: además de borrar y actualizar en cascada, la clave externa formará parte de la clave principal.

- Si la relación por la cual depende en identificación es 1:N, hará falta otro campo en la clave principal.
- Si es 1:1, con la clave ajena será suficiente como clave principal.

En nuestro ejemplo quedará así:



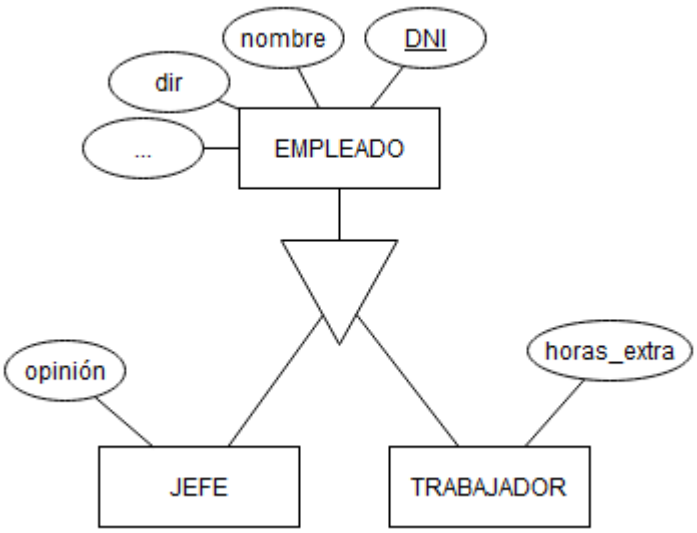

```
EMPLEADO (DNI, nombre, dir, tfno, sueldo, f_nac)
    PK = DNI
FAMILIAR (DNI, nombre, f_nac, parentesco)
    PK = DNI, nombre
    FK = DNI --> EMPLEADO (DNI) /* borrar en cascada */
```

4.6. Relaciones de jerarquía.

El problema de las relaciones de jerarquía (o especializaciones) es que, aunque de forma teórica la solución sea correcta, en la práctica supone muchas tablas y con cierto grado de mantenimiento entre ellas.

Existen varias soluciones para realizar el paso a tablas de una especialización. La solución que se elija en cada caso dependerá del tipo de especialización que estemos resolviendo: total, parcial, solapada o exclusiva.

Partimos del siguiente ejemplo:



Las 3 soluciones posibles que podemos aplicar son las siguientes:

1. Crear una tabla para cada una de las entidades, tanto para la superclase como las subclases. En este caso las subclases tendrían que guardar la clave de la primaria de la superclase. Válido para relaciones de jerarquía **exclusiva/total**.
2. Crear una tabla sólo para las subclases. En este caso los atributos de la superclase habría que guardarlos en cada una de las subclases.

3. Crear una única tabla para la superclase. En este caso todos los atributos de las subclases se guardarían en la superclase. Válido para relaciones de jerarquía **solapada/total**.

En el ejemplo nos quedaría para la solución 1:

```
EMPLEADO (dni, nombre, dir, ..., tipo)
    PK = dni
JEFE (dni, opinion)
    PK = dni
    FK = dni --> EMPLEADO (dni)
TRABAJADOR (dni, horas_extra)
    PK = dni
    FK = dni --> EMPLEADO (dni)
```

En el ejemplo nos quedaría para la solución 2:

```
JEFE (dni, nombre, dir, ..., opinion)
    PK = dni
TRABAJADOR (dni, nombre, dir, ..., horas_extra)
    PK = dni
```

En el ejemplo nos quedaría para la solución 3:

```
EMPLEADO (dni, nombre, dir, ..., tipo, opinion, horas_extra)
    PK = dni
```

4.7. Restricciones externas

Ya hemos comentado que restricciones externas son restricciones que no se pueden expresar por medio del modelo de datos. Entonces las expresaremos de palabra.

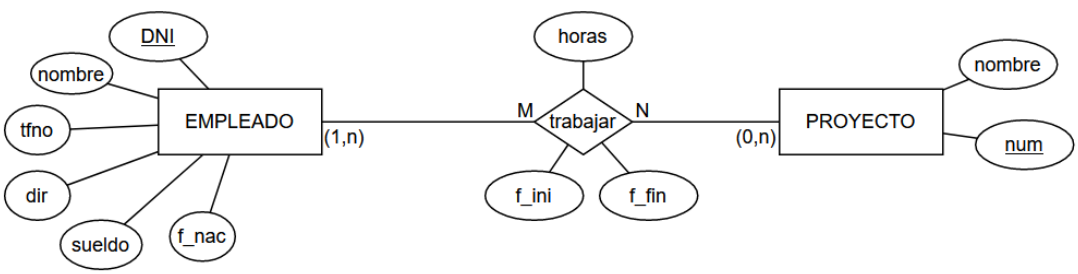
Normalmente, las restricciones externas del Modelo E-R continuarán siéndolo en el Modelo Relacional, porque tampoco se podrán expresar. En nuestro ejemplo teníamos las restricciones externas del Modelo E-R:

- **Rex1**: El jefe de un departamento tiene que ser miembro de este.
- **Rex2**: Un empleado solo puede trabajar en proyectos coordinados por su departamento.

Esto tampoco se puede expresar con el Modelo Relacional, por lo tanto las mantendremos.

Por otro lado, se pueden crear más restricciones externas, porque con el Modelo Relacional no se puede expresar todo lo que se podía expresar con el Modelo E-R.

Por ejemplo, en la relación:



ya hemos comentado que la traducción al Modelo Relacional dará lugar, además de las tablas de las entidades, a otra tabla, quedando la traducción:

```
TRABAJAR (dni, n_proyecto, f_ini, f_fin, horas)
    PK = dni, n_proyecto, f_ini
    FK = dni --> EMPLEADO (dni)
    FK = n_proyecto --> PROYECTO (num)
```

donde *dni* y *n_proyecto* son claves ajenas. Pero la entidad PROYECTO participa de forma total en la relación, es decir, en todo proyecto tiene que haber un empleado trabajando (cardinalidad mínima 1). ¿Como lo controlamos esto? Pues es una nueva restricción externa que la podríamos formular así:

- **RexR3**: Todo proyecto tiene que tener como mínimo un empleado trabajando.

Las cardinalidades mínimas (o participaciones totales) que nos supondrán una restricción externa son:

1. En una relación 1:N, una participación total de la entidad que participa con grado 1.
2. En una relación M:N, cualquier participación total (si las dos participan de forma total, entonces habrá dos restricciones externas).
3. En una relación 1:1, depende de la manera de traducirse.

La manera de implementar las restricciones externas será por medio de un **TRIGGER**, que se active cuando haya una actualización (inserción, modificación o borrado) que afecte a la restricción externa. Por ejemplo, en la restricción **RexR3** los momentos importantes son después de insertar un nuevo proyecto, y antes de eliminar o modificar en la tabla TRABAJAR (por si un proyecto se queda sin gente trabajando en él).

Las acciones a desarrollar podrían ser sacar un aviso, u obligar a insertar como mínimo una tupla en la tabla TRABAJAR, en el caso de la inserción de un nuevo proyecto; en el caso de modificación o eliminación en TRABAJAR podría impedirse esta actualización.

En nuestro ejemplo las restricciones externas al Modelo Relacional que deberías indicar serían:

- **RexR1**: El jefe de un departamento tiene que ser miembro de este.
- **RexR2**: Un empleado solo puede trabajar en proyectos coordinados por su departamento.
- **RexR3**: Todo proyecto tiene que tener como mínimo un empleado trabajando.

5. Normalización

El diseño de una BD relacional se puede realizar aplicando al mundo real, en una primera fase, un modelo como el modelo E/R, a fin de obtener un esquema conceptual; en una segunda fase, se transforma dicho esquema al modelo relacional mediante las correspondientes reglas de transformación. También es posible, aunque quizás menos recomendable, obtener el esquema relacional sin realizar ese paso intermedio que es el esquema conceptual. En ambos casos, es conveniente (obligatorio en el modelo relacional directo) aplicar un conjunto de reglas, conocidas como **Teoría de normalización**, que nos permiten asegurar que un esquema relacional cumple unas ciertas propiedades, evitando:

- La redundancia de los datos: repetición de datos en un sistema.
- Anomalías de actualización: inconsistencias de los datos como resultado de datos redundantes y actualizaciones parciales.
- Anomalías de borrado: pérdidas no intencionadas de datos debido a que se han borrado otros datos.
- Anomalías de inserción: imposibilidad de adicionar datos en la base de datos debido a la ausencia de otros datos.

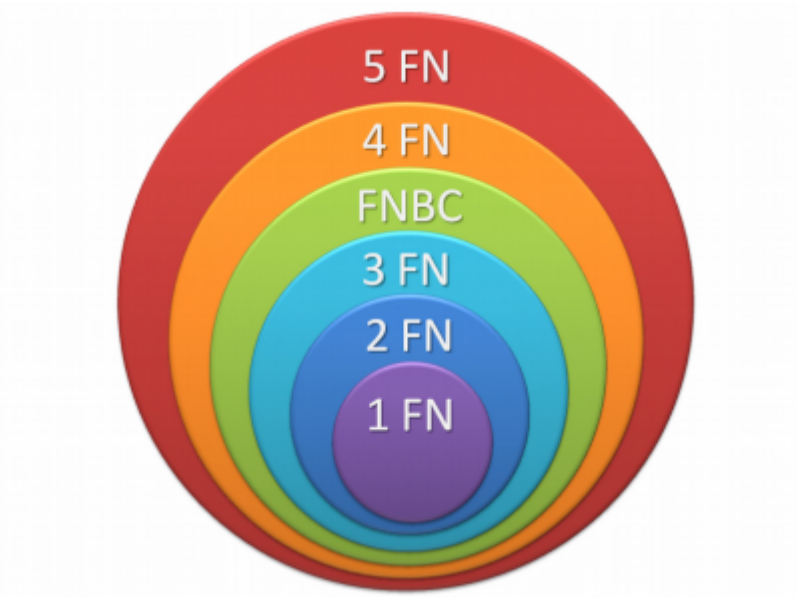
Así pues, el proceso de normalización consiste en optimizar las tablas para eliminar redundancia y un posible mal diseño.

Hemos de ser conscientes de que si la base de datos está bien diseñada desde el principio, estará prácticamente normalizada, no siendo necesaria la normalización. Por otro lado **si nos proporcionan una base de datos creada sin realizar un diseño previo (por ejemplo bases de datos antiguas basadas en sistemas de ficheros), es muy probable que necesitemos normalizar**.

5.1. Teoría de la normalización

En la teoría de bases de datos relacionales, las **formas normales (FN)** proporcionan los criterios para determinar el grado de vulnerabilidad de una tabla a inconsistencias y anomalías lógicas. Cuanto más alta sea la forma normal aplicable a una tabla, menos vulnerable será a inconsistencias y anomalías.

Edgar F. Codd originalmente definió las tres primeras formas normales (**1FN, 2FN, y 3FN**) en 1970. Estas formas normales se han resumido como requiriendo que **todos los atributos sean atómicos, dependan de la clave completa y en forma directa (no transitiva)**. La forma normal de Boyce-Codd (**FNBC**) fue introducida en 1974 por los dos autores que aparecen en su denominación. Las cuarta y quinta formas normales (**4FN y 5FN**) se ocupan específicamente de la representación de las relaciones muchos a muchos y uno a muchos entre los atributos y fueron introducidas por Fagin en 1977 y 1979 respectivamente.Cada forma normal incluye a las anteriores.



5.1.1. Primera Forma Normal (1FN)

Una tabla está en 1FN si y solo si los valores que componen cada atributo de una tupla son atómicos. Es decir, en un atributos no han de aparecer valores repetitivos.

Por ejemplo, supongamos que tenemos la siguiente tabla con datos de alumnado de un centro de enseñanza secundaria.

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno	Teléfonos
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla	660111222
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla	660222333 660333444 660444555

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno	Teléfonos
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla	
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba	
55555555E	José	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba	661000111 661000222

Como se puede observar, esta tabla no está en 1FN puesto que el campo **Teléfonos** contiene varios datos dentro de una misma celda y por tanto no es un campo cuyos valores sean atómicos (es lo que nos ocurriría con los atributos multivaluados de nuestro MER).

Para pasar a 1FN una tabla que no lo estaba se descompone en dos distintas:

- La primera tabla será la proyección de la tabla original sobre los siguientes atributos:

  La clave de la tabla original.

  Los atributos atómicos (los que contienen valores únicos).

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba
55555555E	José	2ESO-A	02-Julio-2016	Federico		

- La segunda tabla será la proyección de la tabla original sobre los siguiente atributos:
  - La clave de la tabla original.
  - Los atributos que tiene valores múltiples, distribuidos en varias filas distintas, por tanto existirá una fila con un único valor elemental.

DNI	Teléfono
11111111A	660111222
22222222B	660222333
22222222B	660333444
22222222B	660444555
55555555E	661000111
55555555E	661000222

Nota

También tendremos que ser capaces de detectar que no está en 1FN cuando tenemos unos atributos *multivaluados "encubiertos". Por ejemplo, una variante del ejemplo anterior podría ser:

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno	Tfno1	Tfno2	Tfno3
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla	660111222		
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla	660111222	660111222	660444555
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla			

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno	Tfno1	Tfno2	Tfno3
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba			
55555555E	José	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba	660111222	660111222	

Se ve que se trata de una manera de "disimular" los atributos multivaluados. Estamos ante el mismo caso que en el ejemplo de arriba y tendremos los mismos problemas, y por tanto la solución es la misma.

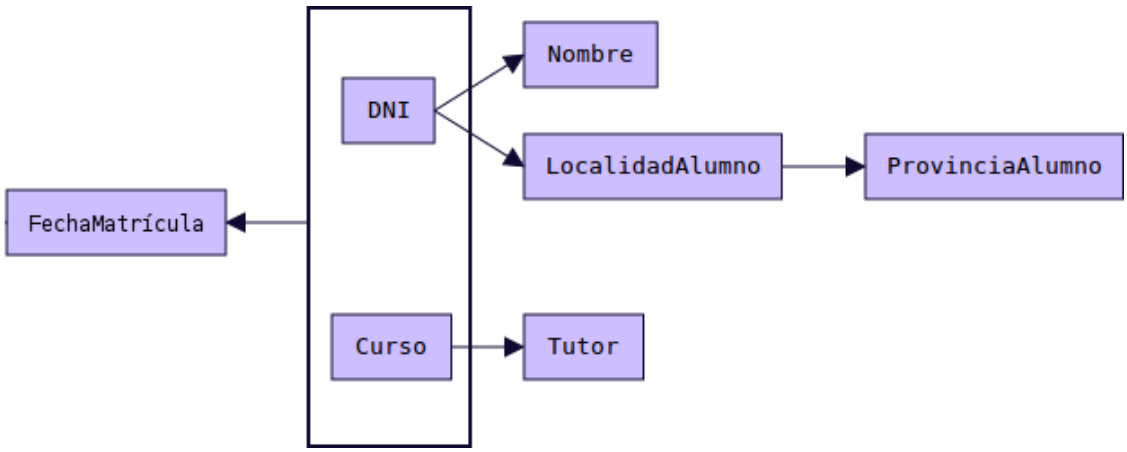
5.1.2. Segunda Forma Normal: 2FN

Una Relación esta en 2FN si y sólo si está en 1FN y todos los atributos que no forman parte de la Clave Principal tienen dependencia funcional completa de ella.

Ejemplo: Seguimos con el ejemplo anterior. Trabajaremos con la siguiente tabla:

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba
55555555E	José	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba

Vamos a examinar las dependencias funcionales. El gráfico que las representa es el siguiente:



- Siempre que aparece un DNI aparecerá el Nombre correspondiente y la LocalidadAlumno correspondiente. Por tanto DNI → Nombre y DNI → LocalidadAlumno. Por otro lado siempre que aparece un Curso aparecerá el Tutor correspondiente. Por tanto Curso → Tutor. Los atributos Nombre y LocalidadAlumno no dependen funcionalmente de Curso, y el atributo Tutor no depende funcionalmente de DNI.
- El único atributo que sí depende de forma completa de la clave compuesta DNI y Curso es FechaMatrícula: (DNI,Curso) → FechaMatrícula.

A la hora de establecer la Clave Primaria de una tabla debemos escoger un atributo o conjunto de ellos de los que dependan funcionalmente el resto de atributos. Además debe ser una dependencia funcional completa. Si escogemos DNI como clave primaria, tenemos un atributo (Tutor) que no depende funcionalmente de él. Si escogemos Curso como clave primaria, tenemos otros atributos que no dependen de él.

Si escogemos la combinación (DNI, Curso) como clave primaria, entonces sí tenemos todo el resto de atributos con dependencia funcional respecto a esta clave. Pero es una dependencia parcial, no total (salvo FechaMatrícula, donde sí existe dependencia completa). Por tanto esta tabla no está en 2FN. La solución sería la siguiente:

DNI	Nombre	Localidad	Provincia
11111111A	Eva	Écija	Sevilla
22222222B	Ana	Écija	Sevilla
33333333C	Susana	El Villar	Córdoba
44444444D	Juan	El Villar	Córdoba
55555555E	José	Écija	Sevilla

DNI	Curso	FechaMatrícula
11111111A	1ESO-A	01-Julio-2016
22222222B	1ESO-A	09-Julio-2016
33333333C	1ESO-B	11-Julio-2016
44444444D	2ESO-A	05-Julio-2016
55555555E	2ESO-A	02-Julio-2016

Curso	Tutor
1ESO-A	Isabel
1ESO-B	Roberto
2ESO-A	Federico

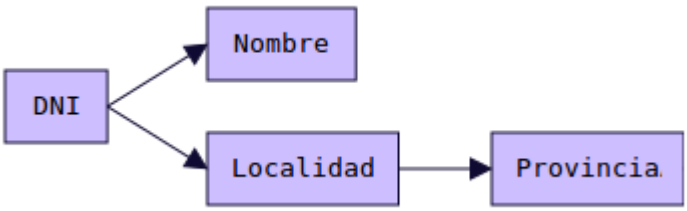
5.1.3. Tercera Forma Normal: 3FN

Una Relación esta en 3FN si y sólo si está en 2FN y no existen dependencias transitivas. Todas las dependencias funcionales deben ser respecto a la clave principal.

Ejemplo: Seguimos con el ejemplo anterior. Trabajaremos con la siguiente tabla:

DNI	Nombre	Localidad	Provincia
11111111A	Eva	Écija	Sevilla
22222222B	Ana	Écija	Sevilla
33333333C	Susana	El Villar	Córdoba
44444444D	Juan	El Villar	Córdoba
55555555E	José	Écija	Sevilla

Las dependencias funcionales existentes son las siguientes. Como podemos observar existe una dependencia funcional transitiva: DNI → Localidad → Provincia



Para que la tabla esté en 3FN, no pueden existir dependencias funcionales transitivas. Para solucionar el problema deberemos crear una nueva tabla. El resultado es:

DNI	Nombre	Localidad
11111111A	Eva	Écija
22222222B	Ana	Écija
33333333C	Susana	El Villar
44444444D	Juan	El Villar
55555555E	José	Écija

Localidad	Provincia
Écija	Sevilla
El Villar	Córdoba

RESULTADO FINAL

DNI	Nombre	Localidad
11111111A	Eva	Écija
22222222B	Ana	Écija
33333333C	Susana	El Villar
44444444D	Juan	El Villar
55555555E	José	Écija

Localidad	Provincia
Écija	Sevilla
El Villar	Córdoba

DNI	Teléfono
11111111A	660111222

DNI	Teléfono
22222222B	660222333
22222222B	660333444
22222222B	660444555
55555555E	661000111
55555555E	661000222

DNI	Curso	FechaMatrícula
11111111A	1ESO-A	01-Julio-2016
22222222B	1ESO-A	09-Julio-2016
33333333C	1ESO-B	11-Julio-2016
44444444D	2ESO-A	05-Julio-2016
55555555E	2ESO-A	02-Julio-2016

Curso	Tutor
1ESO-A	Isabel
1ESO-B	Roberto
2ESO-A	Federico

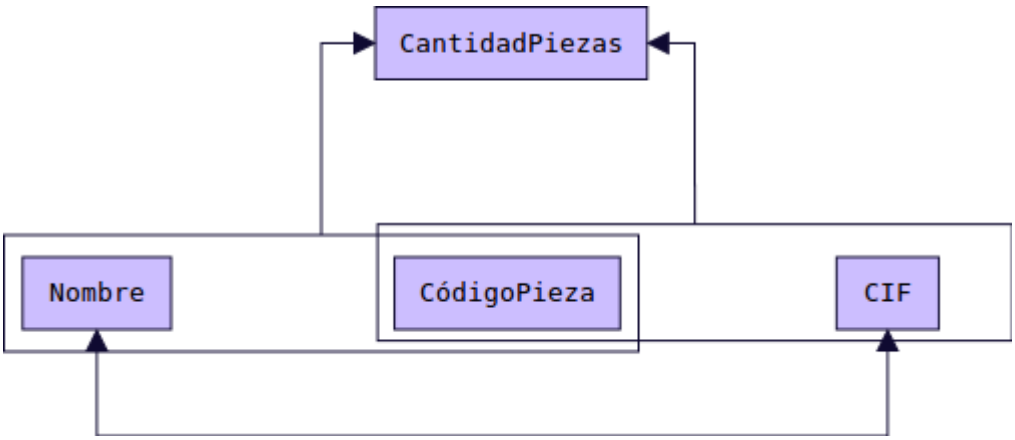
5.1.4. Forma Normal de Boyce-Codd: FNBC

Una Relación esta en FNBC si está en 3FN y no existe solapamiento de claves candidatas. Solamente hemos de tener en cuenta esta forma normal cuando tenemos varias claves candidatas compuestas y existe solapamiento entre ellas. Pocas veces se da este caso.

Ejemplo: Tenemos una tabla con información de proveedores, códigos de piezas y cantidades de esa pieza que proporcionan los proveedores. Cada proveedor tiene un nombre único. Los datos son:

CIF	Nombre	CódigoPieza	CantidadPiezas
S-11111111A	Ferroman	1	10
B-22222222B	Ferrotex	1	7
M-33333333C	Ferropet	3	4
S-11111111A	Ferroman	2	20
S-11111111A	Ferroman	3	15
B-22222222B	Ferrotex	2	8
B-22222222B	Ferrotex	3	4

El gráfico de dependencias funcionales es el siguiente:



El atributo CantidadPiezas tiene dependencia funcional de dos claves candidatas compuestas, que son:

- (NombreProveedor, CodigoPieza)
- (CIFProveedor, CódigoPieza)

Existe también una dependencia funcional en doble sentido (que no nos afecta): NombreProveedor <-> CIFProveedor.

Para esta tabla existe un solapamiento de 2 claves candidatas compuestas. Para evitar el solapamiento de claves candidatas dividimos la tabla. La solución es:

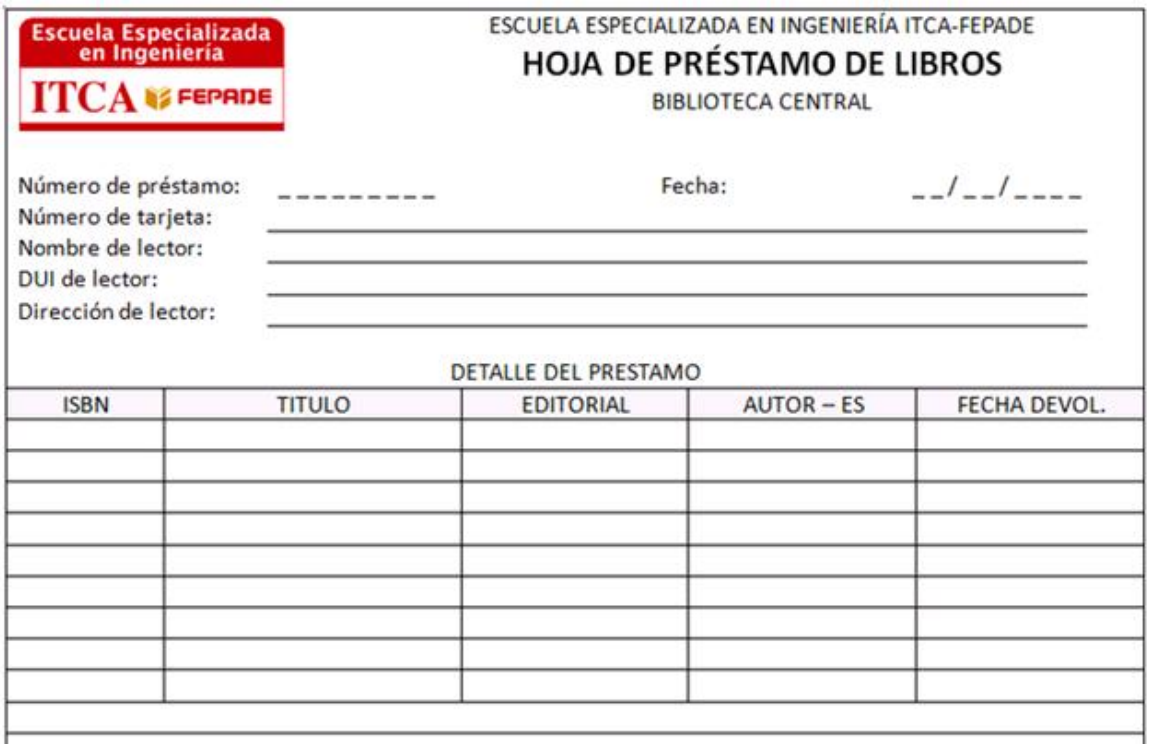
CIF	Nombre
S-11111111A	Ferroman
B-22222222B	Ferrotex

CIF	Nombre
M-33333333C	Ferropet

CIF	CódigoPieza	CantidadPiezas
S-11111111A	1	10
B-22222222B	1	7
M-33333333C	3	4
S-11111111A	2	20
S-11111111A	3	15
B-22222222B	2	8
B-22222222B	3	4

5.1.5. Ejemplo del proceso de normalización

Vamos a realizar el proceso de Normalización para el documento "Hoja de control de Libros" para la Biblioteca de la Escuela Especializada en Ingeniería ITCA-FEPADE.



Escuela Especializada en Ingeniería
ITCA FEPADE

ESCUELA ESPECIALIZADA EN INGENIERÍA ITCA-FEPADE

HOJA DE PRÉSTAMO DE LIBROS

BIBLIOTECA CENTRAL

Número de préstamo: _____ Fecha: __/__/____
Número de tarjeta: _____
Nombre de lector: _____
DUI de lector: _____
Dirección de lector: _____

DETALLE DEL PRESTAMO

ISBN	TITULO	EDITORIAL	AUTOR – ES	FECHA DEVOL.

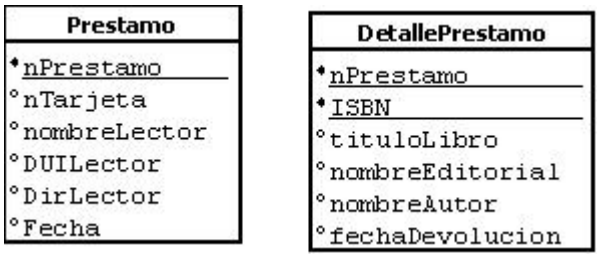
1FN: Dependencia funcional.

No deben existir grupos repetidos para un valor clave.

El primer paso es colocar todos los datos del documento en una tabla para analizar la información:

	nPrestamo	fechaPrestamo	nTarjeta	nombreLector	DUILector	dirLector	ISBN	titulo
Prestamo 1	5	05/05/2013	8989	Jhony Mikel Escobar	01014769-9	Santa Tecla	895623	Administración de Bases de Datos
	5	05/05/2013	8989	Jhony Mikel Escobar	01014769-9	Santa Tecla	784512	Introducción a las bases de datos
	5	05/05/2013	8989	Jhony Mikel Escobar	01014769-9	Santa Tecla	159846	Procesamiento de Bases de Datos
Prestamo 2	6	08/05/2013	5654	Kenny Mendoza	09618949-2	Antiguo Cus	852564	Introducción a Visual Basic
	6	08/05/2013	5654	Kenny Mendoza	09618949-2	Antiguo Cus	845923	21 Lecciones de Visual Basic

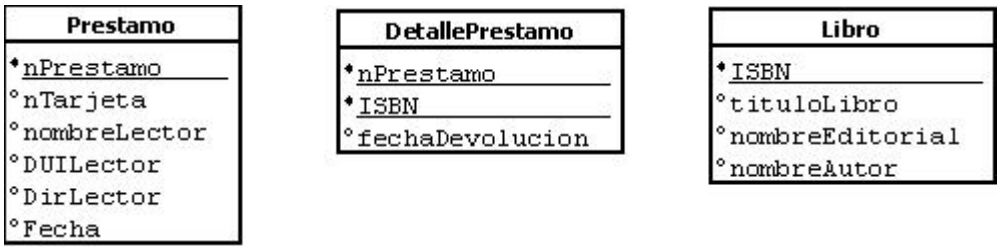
Podemos observar que un usuario puede solicitar en un mismo prestamo uno o más libros, por lo que los datos del prestamo y del lector aunque pareciera que no se repiten en el documento si se repiten en la tabla. Separamos los datos que se repiten de los que no y crearemos dos tablas en la primera forma normal.



2FN: Dependencia Funcional Completa.

Se trabaja sobre la tabla Detalle y se examinan todos los campos para verificar la dependencia funcional competa de todos los campos con respecto a las claves nPrestamo e ISBN.

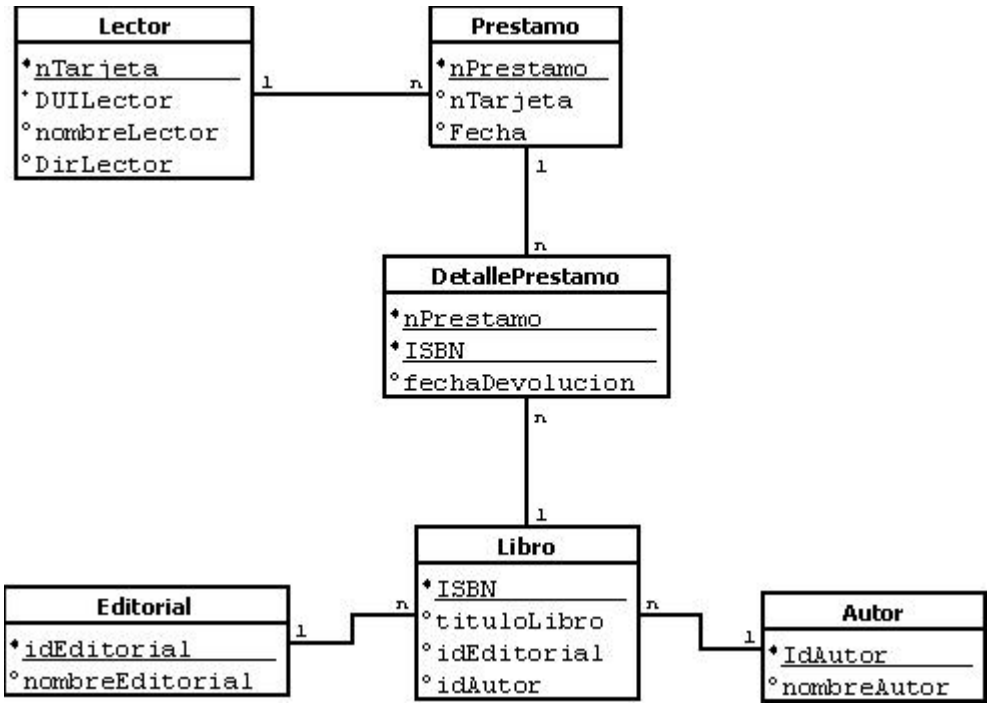
Se crea una nueva tabla con los atributos restantes, siendo su clave el subconjunto de atributos de la clave inicial de los que dependen de forma completa.



3FN: Dependencia Transitiva.

Se evalúan todas las tablas examinando si existe dependencia transitiva entre los campos y la clave principal. Si se encuentra una de estas dependencia se crean nuevas tablas.

Se crea una nueva tabla con los dos atributos no clave, que intervienen en la dependencia transitiva, seleccionando entre ambos a aquel que cumpla los requerimientos de clave. En la tabla origen el atributo no clave que origina la dependencia transitiva se convierte en una clave ajena.


Lector
nTarjeta
DUILector
nombreLector
DirLector
Prestamo
nPrestamo
nTarjeta
Fecha
DetallePrestamo
nPrestamo
ISBN
fechaDevolucion
Libro
ISBN
tituloLibro
idEditorial
idAutor
Editorial
idEditorial
nombreEditorial
Autor
IdAutor
nombreAutor
1
n