Teoría de
Bases de Datos

Índice de contenido:

1. Introducción 2
1.1 Antecedentes 2
1.2 Bases de Datos y Sistemas de Gestión de Bases de Datos. 2
2. Diseño de bases de datos 3
2.1 Objetivos. 3
2.2 Fases de diseño 3
3. Diseño conceptual: El modelo Entidad-Relación 4
3.1 Elementos del modelo ER 4
3.1.1 Entidades 4
3.1.2 Relaciones 4
3.1.3 Atributos 5
3.1.4 Ejemplo de diagrama ER 6
3.2 Elementos del modelo ER Extendido ó ERE 7
3.2.1 Entidades fuertes y débiles 7
3.2.2 Restricciones de cardinalidad 7
3.2.3 Generalizaciones 8
3.2.4 Ejemplo de diagrama ERE 9
4. Diseño lógico: El modelo Relacional 10
4.1 El modelo relacional 10
4.2 Transformación del modelo entidad-relación al modelo relacional 10
4.3 Restricciones del modelo relacional 14
4.3.1 Restricciones inherentes al modelo 14
4.3.2 Restricciones semánticas o de usuario 14
4.4 Normalización 15
4.4.1 Introducción 15
4.4.2 Definición: Dependencia funcional 15
4.4.3 Formas normales 16

1. Introducción

1.1 Antecedentes

El ser humano, desde sus orígenes, siempre ha tenido necesidad por controlar la información de aquello que lo rodea. Con anterioridad a la era de la informática, la única forma posible de mantener registrada y organizada la información de nuestro mundo era el uso del papel y de la escritura. De esta forma surgieron los documentos y con ellos, conceptos tales como los de: *archivador, ficheros, carpetas, formularios*, etc.; conceptos que permitían tener organizada dicha información y que, como sabemos, heredó la informática.

Con la llegada de los ordenadores, y persiguiendo un acceso más eficiente a los datos, surgió la necesidad de informatizar el manejo de los archivadores manuales, creándose así los denominados *sistemas de ficheros*. Si bien la organización de la información mediante ficheros fue un avance considerable, el hecho de que dichos ficheros trabajaran de forma independiente (con información descentralizada), implicaba una serie de inconvenientes:

1. **Redundancia**: se produce redundancia cuando los mismos datos son repetidos en distintos ficheros. Esta repetición, necesaria cuando se trabaja con ficheros independientes, lógicamente aumenta el espacio necesario para su almacenamiento.
2. **Incoherencias**: debido a la existencia de redundancia, se puede producir incoherencias. Se dice que se ha producido una incoherencia cuando un dato es modificado en un determinado fichero y no en otros. Debido a ello, nos vemos en una situación de inconsistencia, es decir, situación en la que no sabemos qué dato de qué fichero es el correcto.
3. **Dificultad de actualización**: como consecuencia de la redundancia, y con ello, la posible inconsistencia de los datos, los sistemas de ficheros presentan gran dificultad a la hora de actualizar los datos que contienen.
4. Otras dificultades como las de garantizar la **concurrencia, seguridad e integridad** de los datos.

1.2 Bases de Datos y Sistemas de Gestión de Bases de Datos.

Con la intención de dar solución a los mencionados inconvenientes, surgió el concepto de *base de datos*. Una ***Base de Datos*** podría definirse como una colección organizada de datos relacionados y almacenados en algún tipo de soporte físico. La utilización de una base de datos bien diseñada, consigue solucionar los inconvenientes anteriormente mencionados. Especial mención requieren las conocidas como ***bases de datos relacionales***, que son aquellas creadas en base al *modelo relacional* del que hablaremos más adelante.

Por su parte, un ***Sistema de Gestión de Bases de Datos*** (SGBD) es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos, además de proporcionar herramientas para añadir, borrar, modificar y analizar los datos. Un SGBD permite controlar el acceso a los datos, asegurar su integridad, gestionar el acceso concurrente a ellos, recuperar los datos tras un fallo del sistema y hacer copias de seguridad. Existen multitud de sistemas gestores de bases de datos, desde los más "domésticos" hasta los más altamente profesionales, y tanto de código libre como de código privativo; algunos ejemplos son: *Base, Access, dBASE, Sybase, Informix, NonStop SQL, MySQL, MongoDB, SQL Server, Oracle JDBC, ODBC, PosgreeSQL,* etc.

2. Diseño de bases de datos

2.1 Objetivos.

La creación de una base de datos es un proceso complejo que parte de la necesidad de almacenar información del mundo real para que su acceso sea rápido y eficiente. Un diseño inapropiado de nuestra base de datos, derivará con seguridad en problemas de redundancia (repetición innecesaria de información), y como consecuencia de ésta, problemas de incoherencia (cuando se tiene una base de datos con elementos duplicados, puede ocurrir que se modifiquen unos y no otros, llegándose una situación en la que no se sabe cuál de ellos es correcto), que producirá que la base de datos encuentre en estado de inconsistencia.

Por el contrario, un buen diseño de nuestra base de datos nos va a procurar mayor rendimiento, mayor velocidad de acceso los datos, y en general, un mejor aprovechamiento de los recursos de los que disponemos, además de evitar los problemas de redundancia e inconsistencia antes mencionados. Parece claro, por tanto, que requerimos de una metodología lo suficientemente potente como para permitirnos realizar un diseño metódico y eficiente.

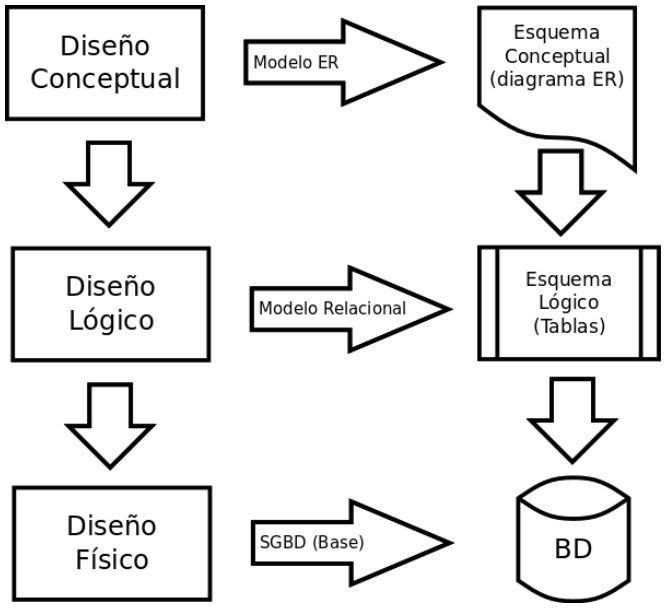
2.2 Fases de diseño

La metodología aplicada en el diseño de bases de datos relacionales, se divide en tres fases:

1.- Fase de **diseño conceptual**: esta fase es independiente del ordenador o del sistema de gestión de base de datos utilizado. En ella, simplemente estudiamos el problema y seleccionamos qué elementos del mundo real vamos a modelar, ayudándonos de la utilización de un modelo de datos conceptual que, para el caso que nos ocupa, será el ***modelo entidad-relación***. Como resultado de esta fase, se generará un *esquema conceptual* de la base de datos.

2.- Fase de **diseño lógico**: partiendo del esquema conceptual obtenido en la fase anterior, llegaremos a un *esquema lógico* de la base de datos. En nuestro caso, transformaremos las entidades y las relaciones identificadas en tablas haciendo uso de la metodología del ***modelo relacional***. En esta fase, se aplican también un conjunto de *reglas de normalización* que veremos más adelante.

3.- Fase de **diseño físico**: esta fase dependerá del sistema gestor de bases de datos utilizado. En este caso, implementamos nuestra base de forma centralizada o distribuida entre varias máquinas, tarea que realizará el gestor de la base de datos empleado.



3. Diseño conceptual: El modelo Entidad-Relación

El modelo **Entidad-Relación** (modelo ER) es el modelo de datos más utilizado para el diseño conceptual de bases de datos. Fue propuesto por *Peter P. Chen* entre los años 1976‑77. Otros muchos autores han investigado y escrito sobre el modelo, proporcionando importantes aportaciones. Así, posteriormente se llegó a desarrollar un modelo ER mejorado al que se conoce como modelo **Entidad-Relación Extendido** (ó ERE).

3.1 Elementos del modelo ER

En el modelo ER básico, se modelan los datos como **entidades**, **relaciones** y **atributos**, permitiendo representar el esquema conceptual de una base de datos de forma gráfica mediante la utilización de los diagramas ER. Pasamos a continuación a describir detalladamente tales elementos:

3.1.1 Entidades

Una **entidad** es todo objeto sobre el que nos interesa almacenar información. De manera más formal, podríamos decir que una entidad es *"una persona, lugar, cosa, concepto, o suceso real o abstracto de interés para la empresa"* ANSI (1977). A cada una de las posibles ocurrencias (cada persona, lugar, cosa o evento concreto) de la entidad se la llama ***ejemplar***. Por ejemplo, la entidad `ALUMNOS` de un centro escolar, o la entidad `CLIENTES` de un banco. En los diagramas ER, las entidades se representan gráficamente como **rectángulos**, y sus nombres son **sustantivos** (normalmente, en plural).

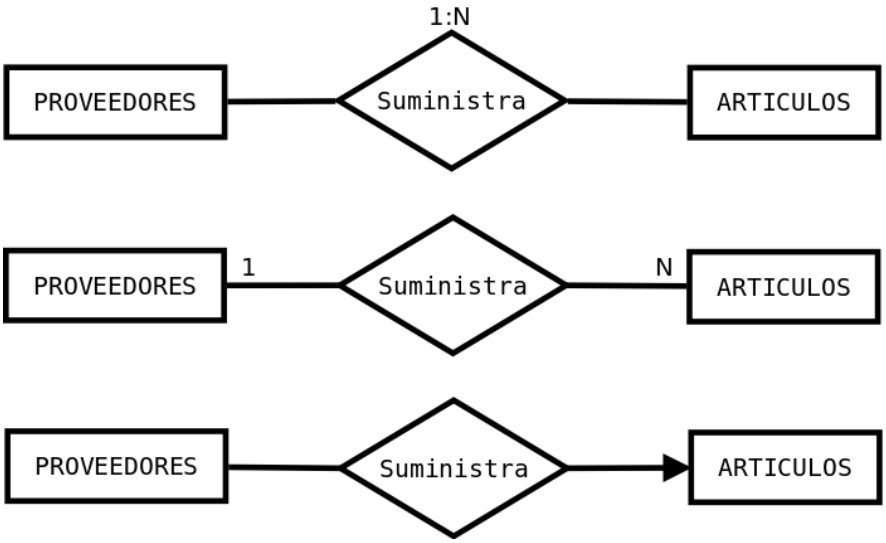
3.1.2 Relaciones

Las **relaciones** (a veces también denominadas *interrelaciones*) representan conexiones, asociaciones o vínculos entre entidades. En los diagramas ER se representa gráficamente como **rombos**, y sus nombres son **verbos**.

Debemos definir los siguientes conceptos:

- **Grado**: Número de entidades que participan de la relación. En función del grado, podríamos hablar de relaciones ***reflexivas*** (una entidad se relaciona consigo misma), relaciones ***binarias*** (la relación vincula dos entidades) ó ***ternarias*** (existen tres entidades implicadas en la relación). En el diseño de relaciones es muy aconsejable que el grado de las relaciones no sea superior a 2.
- **Cardinalidad**: Número de ejemplares de las entidades implicadas que participan en la relación. Según ésta, podemos clasificar las relaciones en los tipos:
  - **1:1** *(de una a una)*: cada ejemplar de una entidad está relacionada únicamente con otro ejemplar de la entidad asociada.
  - **1:N** *(de una a muchas):* cada ejemplar de una entidad puede estar relacionada con muchos ejemplares de la entidad asociada. De **N:1** sería el mismo tipo de relación, pero en sentido contrario.
  - **N:N** *(de muchas a muchas)*: la cantidad de asociaciones entre ejemplares es múltiple.

La forma de representar la cardinalidad de una relación es situándola encima del rumbo que representa dicha relación. Otra forma consiste en situar el número que representa la cardinalidad de cada entidad encima del extremo más cercano de la línea que las une con la relación, o simplemente, utilizar un terminador específico para la línea de unión entre la relación y las entidades. Veamos los siguientes ejemplos:



3.1.3 Atributos

El último de los elementos que se puede también representar el modelo ER, son los atributos. Los **atributos** son los responsables del verdadero almacenamiento de la información o propiedades que nos interesan, tanto de las entidades como de las relaciones. Se representan gráficamente encerrados dentro de una **elipse**, y conectados mediante una línea con la entidad o relación a la que pertenecen.

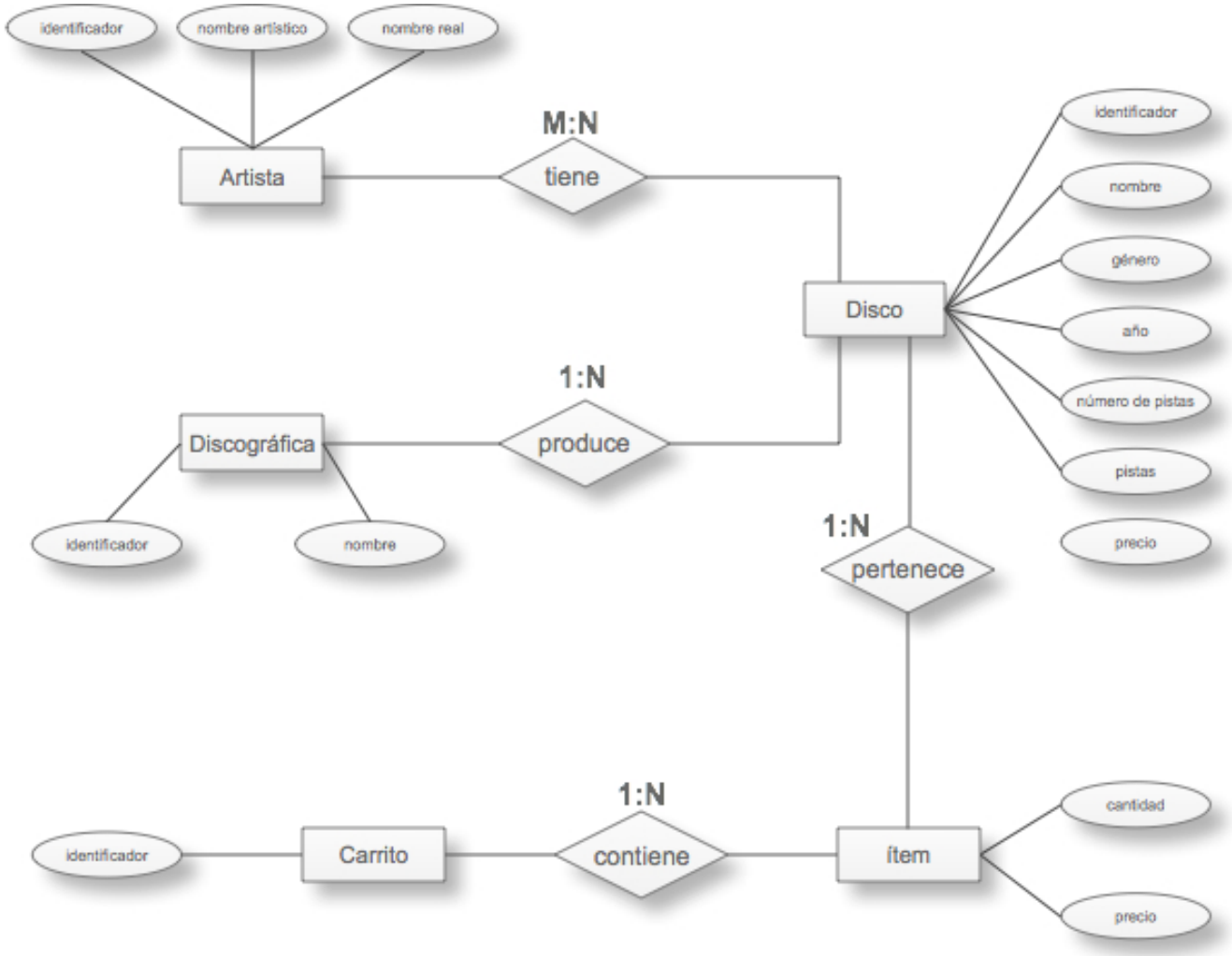
Dentro de los atributos, tenemos de varias clases. Por un lado están los atributos ***descriptores***, es decir, aquellos que aportan información particular de cada ejemplar, y por otro lado, tenemos los ***identificadores***, atributos propios únicamente de la entidades (no de las relaciones) que, además, sirven para identificar de forma **única** a cada ejemplar. Por ejemplo, la `Dirección` de una persona aporta información sobre ella (la describe), pero es su `DNI` quien la identifica.

A los atributos *identificadores* también se les denomina ***claves***, y entre ellos tenemos varios tipos:

- Clave ***candidata***: es un atributo o un conjunto de atributos que identifican de modo único a cada ejemplar de una entidad.
- Clave ***primaria***: es la clave candidata por la que opta finalmente el diseñador de la base de datos. Una entidad puede tener varias claves candidatas, pero sólo una clave primaria.
- Clave ***alterna***: son las claves candidatas por las que no ha adoptado el diseñador como clave primaria.
- Clave ***foránea*** o ***externa***: es un atributo o un conjunto de atributos de una entidad que es, a su vez, *clave primaria* de otra entidad. Las claves foráneas se utilizan para representar relaciones entre datos y profundizaremos en ellas al estudiar cómo transformar el modelo conceptual en el lógico.

3.1.4 Ejemplo de diagrama ER

En el siguiente diagrama ER, se describe el esquema conceptual de una pequeña base de datos desarrollada para tratar las necesidades de venta de discos de una Web de música:



3.2 Elementos del modelo ER Extendido ó ERE

A lo largo de los años, se evidenció que el modelo ER básico no cumplía su propósito con eficacia debido a que se detectaron ciertas limitaciones semánticas. Por ese motivo, se fueron incorporando nuevos elementos:

3.2.1 Entidades fuertes y débiles

Las entidades ***fuertes*** son aquellas que tienen sentido y existencia por sí solas. Las entidades ***débiles*** sin embargo, requieren estar relacionadas con otra entidad de la que dependen. En este sentido, las entidades débiles pueden serlo por dos motivos: bien porque su existencia en la base de datos depende de una entidad *fuerte*, lo que se conoce como ***dependencia en existencia***, o bien porque requieran para su identificación de los atributos identificadores de la otra entidad con la que deben estar relacionadas, lo que se conoce como ***dependencia en identidad*** (que obviamente, siempre implica una *dependencia en existencia*).

Por ejemplo, para que exista una cuenta bancaria, es necesario que exista un titular de manera tal que, si no existe un titular para la cuenta, no tiene sentido la existencia de la cuenta. Dicho de otra forma, si eliminásemos el titular de la cuenta de nuestra base de datos, nos veríamos obligados a borrar todas sus cuentas. Esto demuestra que cuenta bancaria es una entidad débil que depende del titular. En este caso, hablamos de una dependencia en existencia. Otro ejemplo, la entidad `NOTAS` no tendría sentido sin la entidad `ALUMNOS`, tratándose en este caso de una dependencia en identidad. Gráficamente, las entidades *fuertes* se representan con el nombre de la entidad encerrado dentro de un **rectángulo simple**, mientras que las entidades *débiles* se representan encerradas con un **doble rectángulo**.

3.2.2 Restricciones de cardinalidad

A lo largo del tiempo, se evidenció la necesidad de representar el hecho de que no todos los ejemplares de una entidad tenían por qué estar relacionadas con algún otro ejemplar de la entidad asociada. Imaginemos, por ejemplo, la relación *caza* entre dos entidades denominadas `GATOS` y `RATONES`. Podría ocurrir que un gato no cace ningún ratón, y del mismo modo, podría ocurrir que un ratón no fuera cazado por ningún gato. En ambos casos, la participación de dichas entidades es parcial (no total). Para reflejar gráficamente este tipo de situaciones, se desarrollaron las llamadas ***restricciones de cardinalidad***. Así, se definen las cardinalidades ***mínima*** y ***máxima*** como el número mínimo y máximo de ejemplares de una entidad que pueden relacionarse con un único ejemplar de la otra entidad. Gráficamente, las restricciones de cardinalidad se representan por una de las etiquetas siguientes: **(0,1)**, **(1,1)**, **(0,N)** ó **(N,N)**. Para entenderlo, veamos el siguiente diagrama:

En importante saber leer estas etiquetas, y el secreto está en preguntarse sobre un ejemplar de una entidad, leyendo la cardinalidad que está al otro extremo de la relación. Así, podemos concluir que:

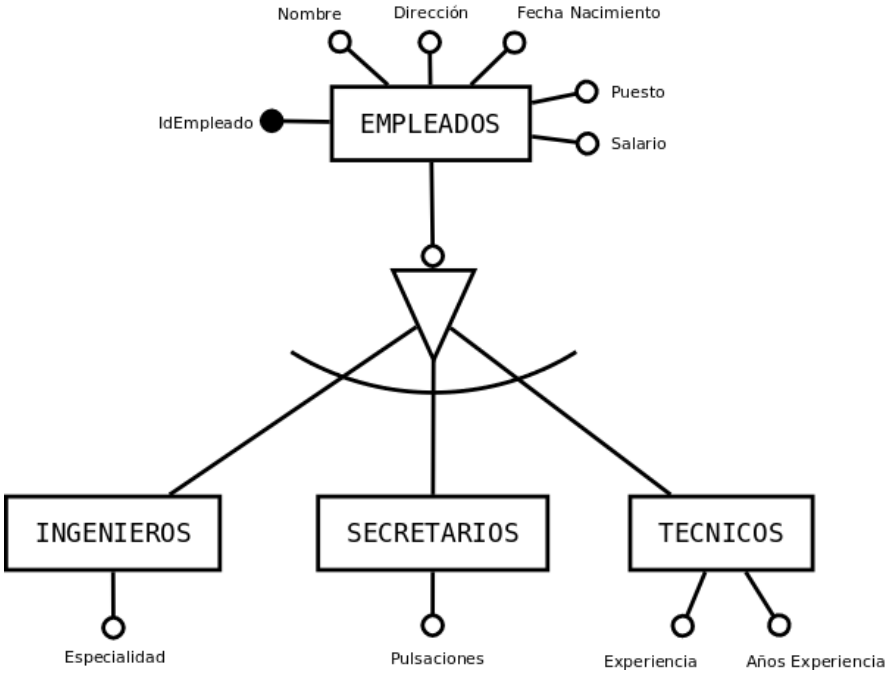
- Un gato puede cazar ningún ratón (**0**,N), o bien, varios (0,**N**).
- Un ratón puede ser cazado por ningún gato (**0**,1), o bien, por un único gato (0,**1**).

Cuando se utiliza esta notación, no se suelen utilizar terminaciones específicas para las líneas que unen las entidades con las relaciones, aunque si existieran, debe entenderse que son las cardinalidades máximas.

3.2.3 Generalizaciones

En ocasiones, puede ocurrirnos que identifiquemos un conjunto de atributos de una entidad que únicamente tienen sentido para un subconjunto de los ejemplares de la misma. Por ejemplo, las `Pulsaciones` de un empleado, pueden tener sentido si dicho empleado en un `SECRETARIO`, pero poca o ninguna si se trata de un `INGENIERO`.

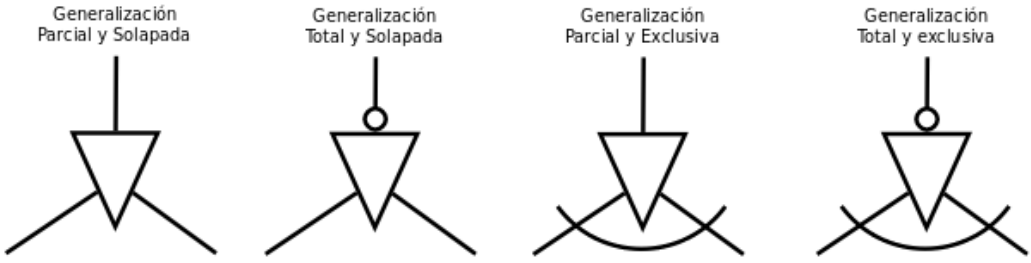
Para poder representar esta realidad, se utilizan las *generalizaciones*. Las ***generalizaciones*** son un mecanismo que permite especializar una entidad "padre", se que denomina ***supertipo***, en una o varias "hijas", a las que se denominará ***subtipos***. Para comprender mejor lo anterior, veamos el siguiente diagrama ERE:



Existen restricciones semánticas relacionadas con las *generalizaciones*, concretamente:

- **Totalidad/parcialidad**: Si no hay ejemplares en el *supertipo* que no pertenezcan a ninguno de los *subtipos*, entonces se trata de una generalización total, y en caso contrario, parcial.
- **Exclusividad/solapamiento**: Si puede haber ejemplares que pertenezcan a más de uno de los *subtipos*, entonces se trata de generalizaciones con solapamiento, en caso de que los *subtipos* sean disjuntos, se habla de generalizaciones exclusivas.

Para representar las mencionadas restricciones, se utilizan los siguientes elementos gráficos:



En el ejemplo anterior, se muestra una generalización total, es decir, no existen `EMPLEADOS` que no sean ni `INGENIEROS` ni `SECRETARIOS` ni `TÉCNICOS`; además, es una generalización exclusiva, es decir, un `EMPLEADO`, o es `INGENIERO`, o es `SECRETARIO` o es `TÉCNICO`, pero no puede ser varias cosas a la vez. Puede ocurrir que un subtipo no tenga atributos propios, en cuyo caso, sólo se representará si dicho subtipo está implicado en alguna relación.

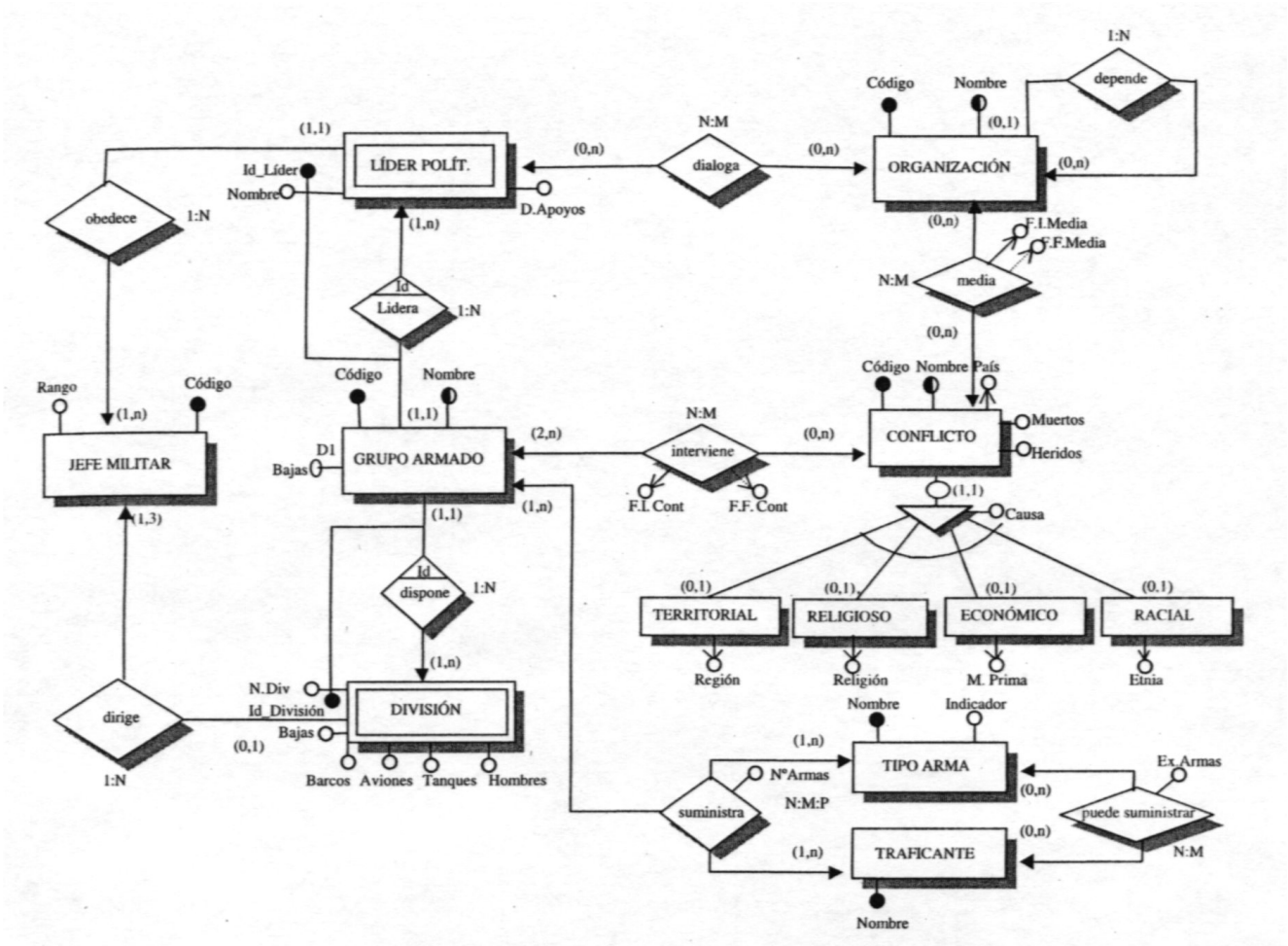
Por otro lado, seguro ha advertido también una nueva forma de notar los atributos. En este sentido, se utiliza:

Mención especial requieren los ***atributos multivaluados***. Estos atributos pueden tomar un conjunto "limitado" de valores. Imaginemos, por ejemplo, el `teléfono` de un `CLIENTE`. El atributo `teléfono` podría no tener ningún valor, o bien, un conjunto de valores (fijo, móvil, otro móvil, etc.). Este tipo de atributos, que podrían contener varios valores, pueden representarse de dos maneras: como un *atributo multivaluado* (utilizando la notación arriba indicada), o bien, como una nueva entidad relacionada con la entidad original (siento, normalmente, la cardinalidad de dicha relación de tipo 1:N). Existen discrepancias en el modo de representarlos, si bien, suele decantarse por la segunda cuando se sabe con seguridad que habrá otras relaciones entre esta nueva entidad y las ya existentes en la base de datos.

Así mismo, en las relaciones que unen una entidad fuerte con otra débil, suele indicarse en la parte superior del rombo que representa la relación, el tipo de dependencia existente entre dichas entidades, pudiendo ser: `Ex` para dependencias en existencia, e `Id` para dependencias en identidad.

3.2.4 Ejemplo de diagrama ERE

En el siguiente diagrama ERE, se describe el esquema conceptual de una base de datos desarrollada para tratar diversas cuestiones sobre conflictos de carácter bélico.



4. Diseño lógico: El modelo Relacional

Una vez finalizada la fase de diseño conceptual, habiendo obtenido como resultado el esquema conceptual de la base de datos (representado mediante su diagrama ERE), hemos de transformar dicho esquema conceptual en su correspondiente esquema lógico. En primer lugar, será necesario elegir el modelo apropiado sobre el que realizar la transformación. Existen muchos, aunque nosotros nos decantaremos por uno de los mundialmente más utilizados, el ***modelo relacional***.

4.1 El modelo relacional

El *modelo de datos relacional* está basado en el concepto matemático de ***relación*** (¡cuidado!, no las confundas con las *relaciones* del modelo entidad-relación). En este caso, hay que entender una *relación* como una ***tabla***, tablas que tienen los siguientes elementos:

- El ***nombre*** de la tabla.
- Los ***atributos*** o ***campos*** de información de los que nos interesa mantener información (las columnas de la tabla). Al número de *atributos* se le denomina ***grado*** de la relación.
- Y las ***tuplas*** o ***registros***, que contienen los valores que toma cada uno de los *atributos* para cada elemento de la relación (las filas de la tabla). Al número de *tuplas* de una relación se le denomina ***cardinalidad***.

Así mismo, las *relaciones* deben cumplir las siguientes **condiciones**:

- Cada relación debe tener un nombre único y distinto de los demás.
- No puede haber dos *atributos* que se llamen igual.
- Los valores de los *atributos* deben ser atómicos, es decir, tomar un solo valor.
- El orden de los *atributos* es irrelevante; no están ordenados.
- Cada *tupla* (= cada fila) es distinta de las demás; no hay *tuplas* duplicadas.
- El orden de las *tuplas* es irrelevante; las *tuplas* no están ordenadas.

4.2 Transformación del modelo entidad-relación al modelo relacional

El proceso de transformación consiste, básicamente, en la creación de un conjunto de tablas en función de las entidades y del tipo de relaciones existentes entre éstas. Más concretamente, podemos afirmar que las reglas básicas para transformar un esquema ER a un esquema relacional, son las siguientes::

- Toda **entidad** se transforma en su correspondiente **tabla** (salvo en el Caso 1 para relaciones del tipo 1:1, como veremos seguidamente).
- Todo **atributo** de una entidad o relación, pasa a ser atributo de la tabla correspondiente. Sin embargo, en el caso especial de tratarse de un atributo multivaluado, éste provocará la modificación del esquema conceptual para dar cabida a una nueva entidad y una nueva relación entre ésta y la entidad original. Tras ello, se procederá como se indica en el resto de apartados para realizar su correspondiente transformación en tablas.
- En el caso de las **relaciones**, para saber cuales se transforman en tablas tendremos que analizar cada caso en función de sus cardinalidades:

- Si tenemos una relación de tipo **1:1** entre dos entidades **Entidad1** y **Entidad2**, distinguimos:
  - **Caso 1**: Que ambas cardinalidades sean **(1,1)**, en cuyo caso fusionaremos ambas entidades creado una única tabla Entidad1-2 con los atributos de ambas y los de la propia relación (si los tuviera). La clave primaria podría ser la de la original Entidad1, o bien, la de la Entidad2.

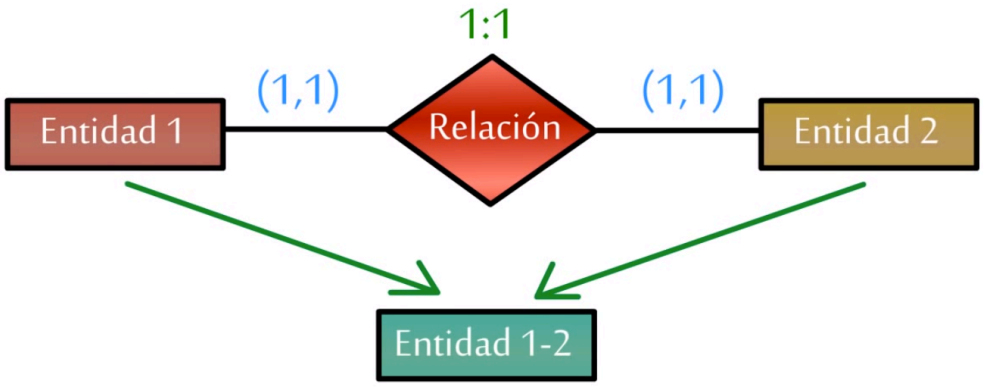


Tabla Entidad 1-2: Entidad1PK, Entidad2PK, Entidad1Atri,

  - **Caso 2**: Que las cardinalidades sean **(0,1) – (1,1)**, en cuyo caso debemos propagar la clave primaria de la entidad (1,1) a la de (0,1), pasando a ser clave foránea de ésta.

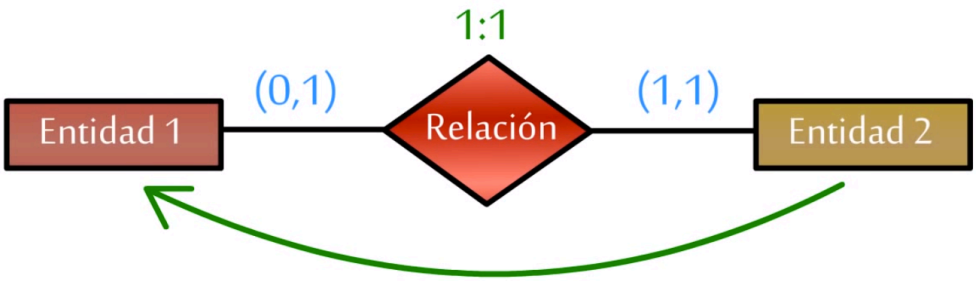


Tabla Entidad 1: Entidad1PK, Entidad1Atri, Entidad2PK
FK

Tabla Entidad 2: Entidad2PK, Entidad2Atri

  - **Caso 3**: Que las cardinales sean ambas **(0,1)**, en cuyo caso se crea una nueva tabla con los atributos de la relación (si los tuviera), donde la clave primaria será la unión de las claves primarias de ambas entidades, siendo además foráneas.

Tabla Entidad 1: Entidad1PK, Entidad1Atri

Tabla Entidad 2: Entidad2PK, Entidad2Atri

Relación: (Entidad1PK, Entidad2PK), RelaciónAtrib
FK FK

- Si tenemos una relación de tipo **1:N** entre dos entidades **Entidad1** y **Entidad2**, hemos de analizar la cardinalidad mínima cercana a la Entidad1, pudiendo presentarse dos casos:

  - **Caso 1**: Que la cardinalidad sea **(1,1)**, en cuyo caso no se crea una tabla nueva, sino que se propaga la clave de la Entidad1 a la tabla de la Entidad2 (siendo clave foránea), llevando también los atributos de la relación (si los tuviera) a la tabla de la Entidad2.

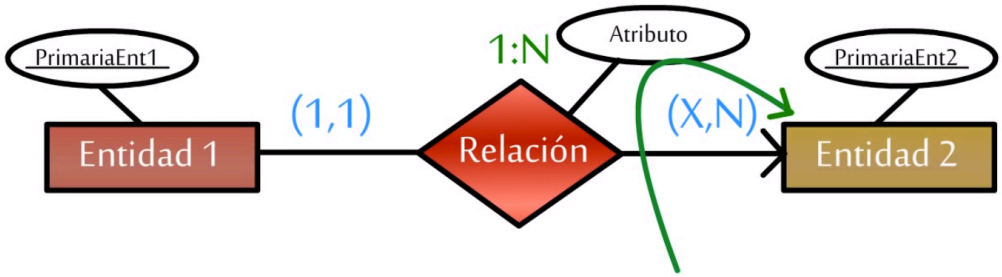


Tabla Entidad 1: PrimariaEnt1

Tabla Entidad 2: PrimariaEnt2, PrimariaEnt1, Atributo
FK

  - **Caso 2**: Que la cardinalidad sea **(0,1)**, en cuyo caso se crea una nueva tabla con la clave de la Entidad2 como primaria y también foránea, la clave de Entidad1 únicamente como foránea, y los atributos de la relación (si los tuviera).

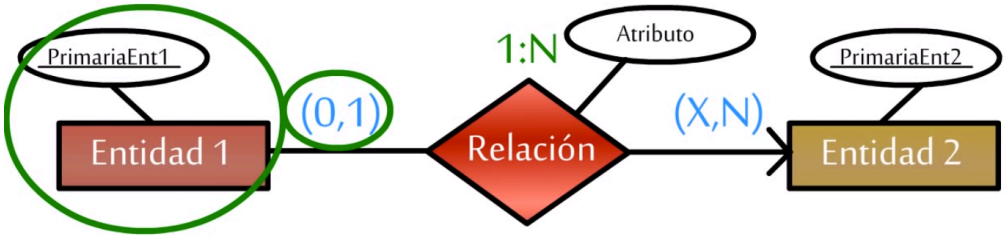
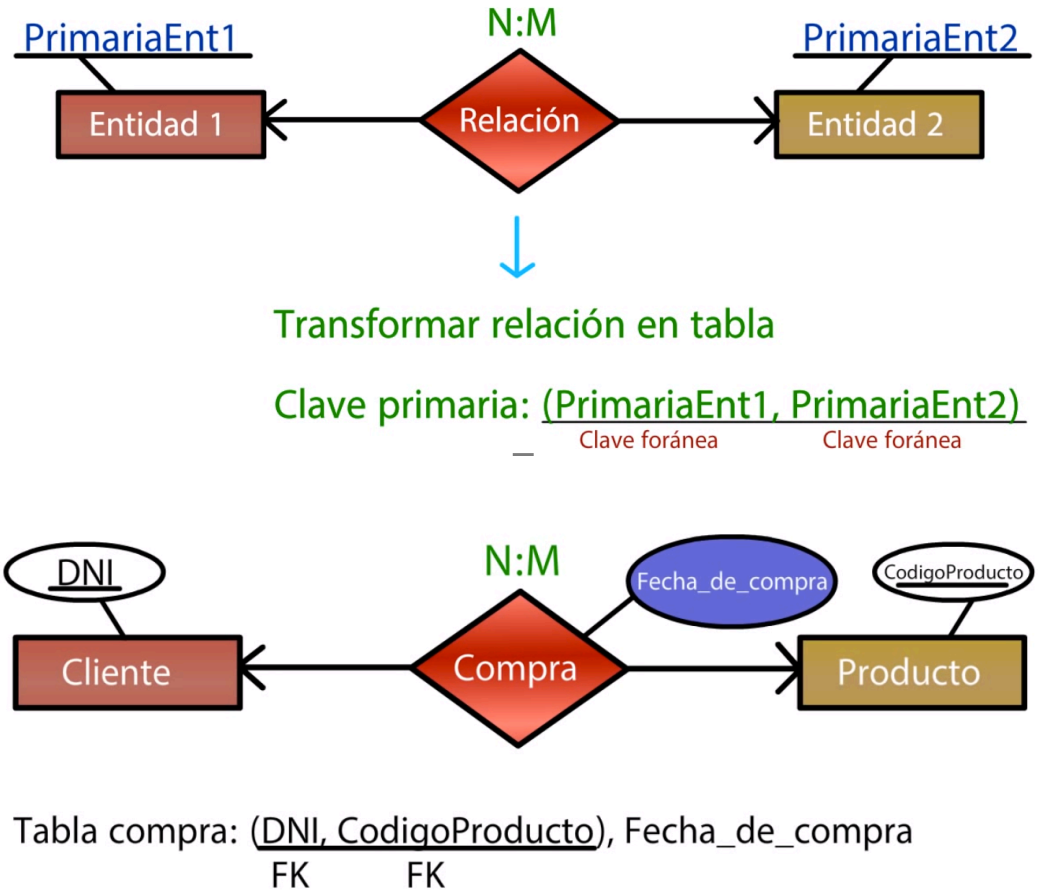


Tabla Entidad 1: PrimariaEnt1

Tabla Entidad 2: PrimariaEnt2

Tabla Relación : PrimariaEnt2, Atributo, PrimariaEnt1
FK FK

- Si tenemos una relación de tipo **N:N** entre dos entidades **Entidad1** y **Entidad2**, se crea una nueva tabla donde la clave primaria será la unión de las claves primarias de las Entidades 1 y 2 (siendo, como siempre, también foráneas), y los atributos de la relación (si los tuviera).



- Finalmente, en relación con las **generalizaciones**, existen distintas alternativas:
  - **Integrar todas las entidades en una única, eliminando los *subtipos***. Esta nueva entidad contendrá todos los atributos del *supertipo* y de todos los de los *subtipos*. La gran ventaja es la simplicidad, pues todo se reduce a una entidad. El gran inconveniente es que se generan demasiados valores nulos para los atributos propios de cada subtipo.
  - **Eliminación del *supertipo***. Esta alternativa es únicamente posible si se trata de una generalización total y exclusiva, y consiste en transferir los atributos del *supertipo* a cada uno de los *subtipos*. Esta opción cuenta, sin embargo, con importantes inconvenientes:
    - Se introduce redundancia en la información, pues los atributos del *supertipo* se repiten en cada uno de los subtipos.
    - El número de relaciones aumenta, pues si el *supertipo* tiene relaciones, éstas pasan a cada uno de los subtipos.
  - **Insertar una relación 1:1 entre la *supertipo* y las *subtipos***, transformándose cada una de las entidades en su correspondiente tabla, y propagándose la clave primaria de la *supertipo* en cada *subtipo*. Esta es la mejor de las opciones y la que nosotros adoptaremos, siendo también conviene incluir en la tabla generada para la *supertipo* un atributo discriminante (que aporte información) de las subtipos.

4.3 Restricciones del modelo relacional

Los datos almacenados en una base de datos deben cumplir una serie de reglas para garantizar que son correctos. Estas reglas se denominan restricciones y las hay de dos tipos:

- Restricciones inherentes al modelo
- Restricciones semánticas

4.3.1 Restricciones inherentes al modelo

Indican las características propias de las relaciones que han de cumplirse obligatoriamente: no hay dos *tuplas* iguales, el orden de las *tuplas* y de los atributos no es relevante, cada atributo sólo puede tomar un único valor del dominio al que pertenece y ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor nulo.

4.3.2 Restricciones semánticas o de usuario

Éstas hacen que las ocurrencias (= las filas) sean válidas y son las siguientes:

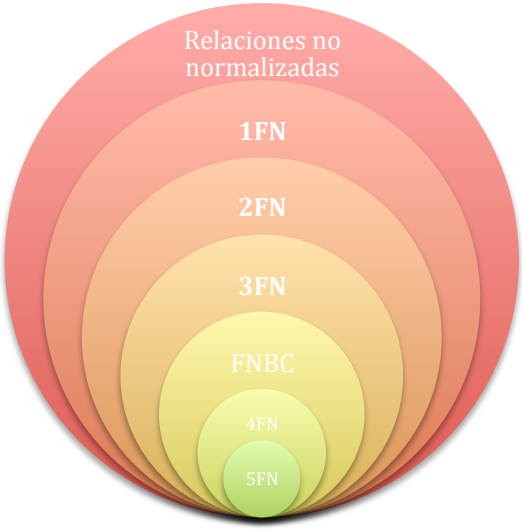
1. **La restricción de clave primaria** (PRIMARY KEY): Permite declarar uno o varios atributos como clave primaria de una relación.
2. **La restricción de unicidad** (UNIQUE): permite definir claves alternativas. Los valores de los atributos no pueden repetirse.
3. **La restricción de obligatoriedad** (NOT NULL): permite declarar los atributos que no pueden tomar valores nulos.
4. **La restricción de verificación** (CHECK): esta restricción permite especificar las condiciones de validación que deban cumplir los valores de los atributos. Cada vez que se realice una inserción o modificación de un atributo con este tipo de restricción, se comprobará que se cumpla la condición.
5. **Integridad referencial o restricción de clave ajena** (FOREIGN KEY): se utiliza para enlazar relaciones mediante sus claves primarias/foráneas. La integridad referencial implica que los valores de la clave foránea en la relación *destino* se corresponden con los de la clave primaria en la relación *origen*. La gestión de este tipo de integridad es fundamental, e implica además distintos modos de actuar ante los borrados y/o las modificaciones, pudiendo ser:
    a. **Borrado y/o modificación en cascada** (CASCADE): El borrado o modificación de una fila en la relación *origen*, ocasiona el borrado o modificación de todas las filas relacionadas en la relación *destino*.
    b. **Borrado y/o modificación restringida** (RESTRICT): En este caso, no se permite borrar y/o modificar las filas de la relación *origen* si existen filas relacionadas en la relación *destino*.
    c. **Borrado y/o modificación con puesta a nulos** (SET NULL): Esta restricción permite poner la clave foránea en la relación *destino* a NULL si se produce el borrado y/o la modificación de la clave primaria en la relación *origen*.
    d. **Borrado y/o modificación con puesta a valor por defecto** (SET DEFAULT): En este caso, el valor que toma la clave foránea es el valor por defecto que se haya especificado en la creación de la tabla.

4.4 Normalización

4.4.1 Introducción

En la primera parte del diseño lógico, se produce la transformación del esquema conceptual de la base de datos en su correspondiente esquema lógico. El resultado será un conjunto de tablas interrelacionadas con sus correspondientes atributos y la identificación de aquellos que resulten ser claves primarias y foráneas.

Con la obtención de dicho esquema lógico no se completa la fase de diseño pues, resultará necesario aplicar una serie de restricciones a las tablas identificadas en las que se conoce como ***reglas de normalización***. La aplicación de dichas reglas tiene por objetivo verificar la calidad de nuestro diseño, sirviendo además, para evitar redundancias y, con ello, los posibles problemas que podríamos tener al actualizar las tablas (cada regla elimina un tipo de redundancia).



En un principio, *Codd* definió las tres primeras reglas o **formas normales** (1FN, 2FN y 3FN). Posteriormente, en 1974 la tercera forma normal fue revisada por *Boyce* y *Codd*, que aumentó el número de restricciones, pasándose a llamar *forma normal de Boyce/Codd* (FNBC) la nueva revisión. Por último, en los años 1977 y 1979, *Robert Fagin* definió las cuarta y quinta formas normales (4FN y 5FN).

Las formas normales se aplican desde la 1FN hasta la 5FN por orden, de forma que superar una significa también haber superado la anterior (a excepción, evidentemente, de la primera). Así, si una base de datos se encuentra, por ejemplo, en 3FN, significará que ha superado las tres primeras formas normales (1FN, 2FN y 3FN). A medida que vamos aplicando formas normales, las tablas se van fragmentado lo que, evidentemente, afectará al rendimiento de nuestra base de datos. En estos apuntes, no analizaremos la 4FN ni la 5FN por su elevada complejidad y porque habitualmente no se utilizan salvo en diseños muy concretos.

4.4.2 Definición: Dependencia funcional

Las ***dependencias funcionales*** son relaciones semánticas entre los datos, que no se pueden definir ni demostrar de una forma matemática o empírica (si bien, su tratamiento sí es puramente matemático). La única forma de conocerlas es mediante un atento estudio del problema real y entendiendo correctamente el sentido y significado de cada atributo.

Dados 2 atributos (o conjuntos de atributos) ***A*** y ***B***, se dice que *B depende funcionalmente de A,* y se nota como ***A→B****, cuando el valor de A implica, determina, o indica el valor de B*. Dicho de otro modo, para cada valor de *A* existe un único valor de *B* asociado. En este caso, se dice que *A* es determinante o descriptor de *B*. Sí, además, ocurre que *A* también depende funcionalmente de *B*, es decir, *B→A*, se dice que la *dependencia funcional* entre *A* y *B* es *mutua*, y se nota como *A↔B*.

Se dice que existe *dependencia funcional completa* entre *A* y *B*, cuando *B* depende funcionalmente de *A* pero no así de ningún subconjunto de atributos de *A*, es decir: $\nexists\, a \subset A \;/\; a \rightarrow B$.

Finalmente, dados 3 atributos (o conjuntos de atributos) *A, B y C*, si ocurre que *A→B* y *B→C*, entonces se cumple también que *A→C*, es decir, *C depende transitivamente* de *A,* lo que se nota como *A⇒C*.

4.4.3 Formas normales

Primera forma normal (1FN)

Se dice que una relación está en primera forma normal si y sólo si **los tributos únicamente pueden contener valores atómicos**, entendiéndose como *atómico* aquel valor que resulta indivisible *(E. F. Codd, 1970)*. Esta definición de atomicidad resulta, sin embargo, ambigua para la mayoría de los autores. En consecuencia, en la práctica se considera que una relación se encuentra en primera forma normal cuando **carece de *atributos repetitivos***, es decir, atributos que pueden tener más de un valor a la vez para una misma *tupla*. Para ilustrar tal situación imagínese, por ejemplo, que deseemos almacenar para cada cliente varios teléfonos, de forma que un diseñador principiante podría considerar almacenar la información de éstos en un único atributo *teléfonos*.

Si una relación contiene atributos repetitivos no estará en 1FN, siendo necesario eliminarlos. La forma de eliminar los atributos repetitivos más elegante, efectiva y aconsejable, consiste en poner el atributo repetitivo en una relación aparte, heredando la clave primaria de la relación en la que se encontraba.

Segunda forma normal (2FN)

Una relación se encuentra en segunda forma normal (2FN) si y sólo si: está en 1FN, y **cada atributo no clave es completamente dependiente de la clave primaria**. Observemos que, para que una relación se encuentre en 2FN, debe ocurrir que todo atributo no clave debe representar un hecho acerca de la **totalidad** de la clave. En consecuencia, la 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple (tiene un solo atributo), entonces también está en 2FN.

Una relación que no se encuentre en 2FN dará problemas en todos los procesos de inserción, eliminación, y actualización, debido a la redundancia de información. Para ilustrar la segunda forma normal, imaginemos una tabla en la que queremos almacenar los datos del conjunto de piezas que tenemos en stock, así como los datos del almacén donde se encuentran. El esquema podría ser el siguiente:

`PIEZAS-ALMACEN (Pieza, Almacén, NúmeroPiezas, DirecciónAlmacén)`

Como podemos comprobar, ocurre que `(Pieza,Almacén)→NúmeroPiezas`, pero también ocurre la dependencia `Almacén→DirecciónAlmacén`, luego `DirecciónAlmacén` no depende completamente de la clave primaria elegida `(Pieza,Almacén)`, sino de un subconjunto de ésta, no encontrándose en 2FN.

Para hacer de la relación supere la segunda forma normal, se deberá realizar una descomposición creando:

- Una tabla con la clave primaria y todos los atributos no claves que dependan completamente de ella. En el ejemplo expuesto sería:
  `PIEZAS (Pieza, Almacén, NúmeroPiezas).`
- Una segunda tabla con la parte de la clave primaria que tiene dependencias no completas, junto con los atributos no claves afectados. En el ejemplo sería:
  `ALMACENES (Almacén, DirecciónAlmacén).`

En la práctica, la segunda forma normal sirve para verificar si se ha elegido adecuadamente la clave primaria.

Tercera forma normal (3FN)

Una relación se encuentra en tercera forma normal (3FN) si y sólo si: está en 2FN, y **no existen atributos no claves que dependan transitivamente de la clave primaria**. Observemos que, para que una relación se encuentre 3FN, debe ocurrir que todo atributo no clave debe representar un hecho **sólo** acerca de la clave (y de la totalidad de la clave por 2FN). La 3FN se aplica a relaciones que tienen, al menos, dos atributos no claves, pues de lo contrario, ya estaría en 3FN.

Una relación que no se encuentra en 3FN dará, al igual que ocurría para una tabla que no estuviese en 2FN, problemas en los procesos de inserción, eliminación y actualización. Para ilustrar la tercera forma normal, imaginemos una tabla en la que queremos almacenar los datos de un conjunto de vehículos, su modelo y potencia. El esquema sería:

`VEHÍCULOS (Matrícula, Modelo, Potencia)`

Como podemos comprobar `Matricula→Modelo` y también `Matrícula→Potencia`, pero ocurre que `Modelo→Potencia`, luego ocurre que existe un atributo no clave, `Potencia`, que depende transitivamente de la clave primaria `Matrícula`.

Para hacer de la relación supere la tercera forma normal, se deberá realizar una descomposición creando:

- Una tabla con la clave primaria y los atributos no claves que no dependan transitivamente de ésta. En el ejemplo expuesto sería:
  `VEHÍCULOS (Matrícula, Modelo).`
- Una segunda tabla para cada atributo que presentara dependencia transitiva con una copia de su determinante (el atributo un conjunto de atributos de los que depende funcionalmente). En el ejemplo expuesto sería:
  `MODELOS (Modelo, Potencia)`.

Forma normal de *Boyce-Codd* (FNBC)

Una relación se encuentra en forma normal de *Boyce-Codd* (FNBC) si y sólo si: está en 3FN, y **todo determinante es una clave candidata.** La formulación de FNBC viene a endurecer la formulación de la 3FN. Obsérvese que una relación que esté en 3FN no estará en FNBC cuando exista un atributo perteneciente a una clave compuesta que dependa de un atributo no clave.

Para ilustrar la forma normal de *Boyce-Codd*, imaginemos una tabla en la que queremos almacenar los datos de un conjunto de direcciones con sus códigos postales y ciudades, de forma tal que el esquema sería:

`CALLEJERO (Dirección, Ciudad, CP)`

Como podemos comprobar, ocurren las dependencias funcionales: `(Dirección, Ciudad)→CP` y `CP→Ciudad`. En éste último caso, ocurre que `CP` es determinante de `Ciudad`, pero `CP` por sí solo no es clave candidata (puede serlo únicamente en asociación con la `Dirección`), por tanto, esta relación no supera la FNBC.

Para hacer de la relación supere la forma normal de *Boyce-Codd*, y como sucede con el resto de las formas normales, debemos proceder realizando una descomposición de la relación actual. Sin embargo, en función de las dependencias funcionales existentes tal descomposición podría no garantiza la conservación de las mismas. Por ejemplo, en el ejemplo expuesto, podría realizarse la siguiente descomposición:

```
DIRECCIONES (Dirección, CP)
CODIGOS (CP, Ciudad)
```

Obsérvese, sin embargo, que se pierde la dependencia funcional `(Dirección, Ciudad)→CP`, si bien, esta pérdida es superable puesto que estaba originada por una mala elección de la clave primaria.