EL MODELO RELACIONAL

En capítulos anteriores hemos estudiado que existen distintos modelos según los cuales la información puede ser almacenada y relacionada entre sí. Actualmente, para la mayoría de las aplicaciones de gestión que utilizan bases de datos, el modelo más empleado es el modelo relacional, por su gran versatilidad, potencia y por los formalismos matemáticos sobre los que se basa.

Este modelo permite representar la información del mundo real de una manera intuitiva, introduciendo conceptos cotidianos y fáciles de entender por cualquier inexperto. Asimismo, mantiene información sobre las propias características de la base de datos (metadatos), que facilitan las modificaciones, disminuyendo los problemas ocasionados en las aplicaciones ya desarrolladas. Por otro lado, incorpora mecanismos de consulta muy potentes, totalmente independientes del S.G.B.D., e incluso de la organización física de los datos; el propio S.G.B.D. es el encargado de optimizar estas preguntas en formato estándar, a sus características propias de almacenamiento.

El modelo relacional fue propuesto por E.F. Codd en los laboratorios de IBM en California. Se trata de un modelo lógico [Irene Luque Ruiz- Ed. Ra-ma], que establece una estructura sobre los datos, aunque posteriormente éstos puedan ser almacenados de múltiples formas para aprovechar características físicas concretas de la máquina sobre la que se implante la base de datos realmente. Es algo así como guardar unos libros en una biblioteca; dependiendo del número de salas de la biblioteca, del tamaño y forma de cada una de ellas, su número de estanterías, y en definitiva, de las características físicas del recinto, podremos disponer los libros de una forma u otra para hacer más cómoda y fácil su consulta y acceso. Los libros son los mismos, pero pueden ubicarse de muy distintas formas.

Vamos a estudiar entonces, las características concretas de este modelo de datos, sin entrar para nada en cómo los almacena físicamente cada ordenador, o cada S.G.B.D.

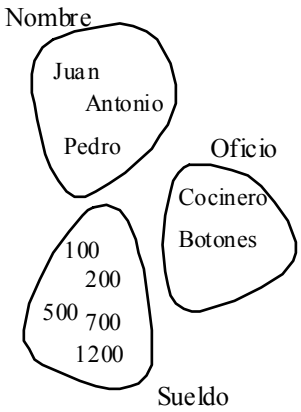
Estructura general.

El nombre de modelo *relacional* viene de la estrecha relación que existe entre el elemento básico de este modelo, y el concepto matemático de relación. Podemos decir que una relación R sobre los conjuntos D_1, D_2, .., D_n, se define como:

$$R \subseteq D_1 \times D_2 \times ... \times D_n$$

donde los conjuntos D_1, D_2, .., D_n pueden ser cualesquiera, e incluso estar repetidos.

Los conjuntos pueden ser cualesquiera, aunque en el momento en que se trabaja con ordenadores, hay que imponer ciertas restricciones de discretización.



Nombre	Oficio	Sueldo
- Juan	Cocinero	100
- Pedro	Botones	700
- Juan	Cocinero	200
- Pedro	Cocinero	700
- Juan	Cocinero	1200

Esto es una relación.
Dado que se trata de un conjunto no puede haber elementos repetidos.

Si nos fijamos en el dibujo adjunto, podemos ver que una de estas relaciones no es más que una lista de líneas, donde cada línea está dividida en trozos.

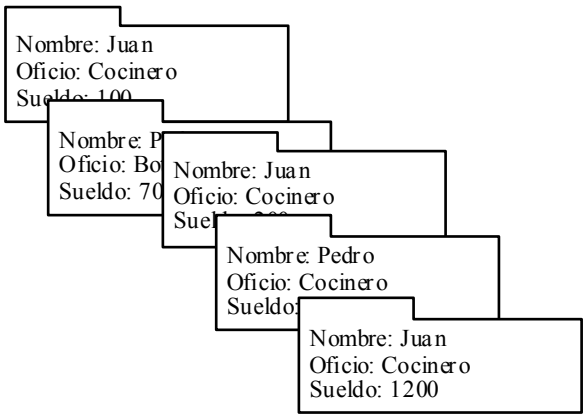
Para observar bien el porqué ha surgido el método relacional, pensemos en cómo almacenaríamos las líneas de la lista anterior, si los ordenadores no existiesen.

Para almacenar estas líneas, tendríamos que emplear papel, de manera que en un folio escribiríamos todas las líneas de la lista, empezando por la primera y continuando en el folio secuencialmente; si el folio se acaba, cogemos otro, y hacemos la misma operación, de forma que, al final, la lista está escrita o almacenada en varios folios. Este método, que es el más directo, tiene el problema de qué ocurre cuando se desean introducir nuevas líneas. Inicialmente, la tarea parece fácil, pues nos basta con seguir escribiendo líneas tras la última línea de la última página, e ir tomando nuevos folios a mediada que las páginas se vayan llenando. No obstante, este método sólo es adecuado si las líneas no están ordenadas según un criterio. Si las líneas ya están ordenadas, y deseamos introducir una nueva, ¿cómo lo hacemos?, ¿escribiendo una línea por enmedio con letra más pequeña?, o bien ¿escribiendo de nuevo todas las líneas, pero esta vez con la que se desea insertar? Está claro que ninguna de estas posibilidades es una solución factible.

Otra posibilidad de registrar esas líneas es utilizando una ficha de cartón para cada una de ellas. Cada ficha de cartón será parecida a las que el alumno rellena para el profesor de cada asignatura a la que asiste, con la variante de que en lugar de poner el nombre, apellidos, dirección, etc. del alumno, se pondrá la información que nos interese guardar. de esta manera cada línea queda almacenada en una ficha de cartón. Si se desea insertar una nueva ficha basta con rellenarla y meterla en su posición adecuada. De la misma forma se puede proceder a la hora de eliminar alguna ficha.

Pues bien, este último método es el que, a grandes rasgos, intenta utilizar el modelo relacional. El modelo relacional representa las listas de líneas mediante registros o fichas cada una de las cuales puede ser manejada individualmente y con independencia de las demás. No obstante, a efectos de facilitar la visualización, puede ser posible ver todas las líneas juntas como si de una

lista se tratase. Ver figura.



De esta manera, tendremos varios tipos de fichas: fichas de clientes, de proveedores, de facturas, de albaranes, de reservas, de empleados, de apuntes contables, etc., cada una de las cuales podemos almacenar en un cajón o en un fichero independiente. De cada tipo de ficha tendremos un montón de fichas rellenas: 100 ó 200 clientes, 4 ó 5 proveedores, miles de facturas, etc. Por tanto, es interesante distinguir entre un tipo de ficha, que no hace referencia a ninguna ficha rellena en concreto (p.ej. una ficha de clientes), y una ficha concreta, rellena con unos datos concretos (la ficha de Juan el Cocinero).

Pues bien, el modelo relacional plasma en un ordenador este mismo esquema, aprovechando las enormes características de computación y almacenamiento de las máquinas actuales.

Concepto de tabla. Dominios y atributos.

Una tabla en el modelo relacional viene a ser como una de las listas descritas anteriormente. Una **tabla** o **relación** es una matriz rectangular que almacena líneas con una estructura concreta:

- La primera línea de una tabla, es una cabecera que indica el nombre de cada columna. O sea, cada columna tiene asignado un nombre único, e indica que los ítemes almacenados en esa columna deben pertenecer a un conjunto de valores concreto: Números, Letras, Frases, etc.
- Cada línea (excepto la primera) recibe el nombre de **tupla**, y almacena ítemes concretos para cada columna.
- Todas las filas deben ser diferentes entre sí.
- El orden de las filas y de las columnas carece de importancia a efectos del S.G.B.D. Este hecho es el que verdaderamente diferencia las tablas relacionales del concepto matemático de relación, en el que el orden de las columnas es fundamental.

En la figura aparece una tabla de

PLATOS

Código	Nombre	Precio	Menú
15	Solomillo a la pimienta	1000	No
23	Fondue Neûchatel	1500	No
17	Migas con chocolate	850	Sí
34	Ajo blanco con uvas	850	Sí
12	Paella valenciana	1100	Sí
21	Mono a la cantonesa	7500	No
07	Sopa de aleta de tiburón	525	Sí

Platos. Toda tabla debe poseer al menos la primera línea, que identifica el nombre de la columna. En este caso, nuestra tabla o relación contiene las columnas Código, Nombre, Precio y Menú (que indica si ese plato pertenece o no a las opciones del menú del día).

El **grado** de esta tabla es el número de campos que posee, en nuestro caso 4. La **cardinalidad** es el número de tuplas concretas que almacena: 7. Nótese que el grado de una tabla es independiente del momento en que observemos la tabla (el número de columnas es invariable, y queda definido en el momento en que se crea la tabla), mientras que la cardinalidad depende de la situación real que represente la tabla y puede variar en el tiempo; p.ej., la lista de platos puede cambiar todos los meses, o según la estación del año, para adecuarse a los gustos propios de cada una de ellas.

La primera fila de una tabla es la más importante, ya que nos da su estructura. Esta columna identifica los nombres de campo o **atributos** de que consta cada tabla. En otras palabras, cada tupla está formada por un conjunto de información estructurada en elementos más simples llamados atributos. El nombre del atributo debe describir el significado de la información que representa. En la tabla *Platos,* el atributo *Precio* tendrá como cometido almacenar el valor en pesetas con el que ese plato se vende al público. A menudo suele ser necesario añadir una pequeña descripción a cada atributo que aclare su naturaleza: ¿el precio lleva I.V.A. o no?

En el atributo *Menú* no está claro que es lo que se almacena. Una descripción del mismo aclararía más las cosas: *Indica si el cliente puede pedir este plato o no en el menú del día.*

Por otro lado, está claro que un atributo en una tupla no puede tomar un valor cualquiera, p.ej., no tiene sentido que en el precio del *Ajo blanco con uvas* se guarde una palabra como pueda ser *Gerente*. Para evitar este tipo de situaciones anómalas en la medida de lo posible, obligaremos a que cada atributo sólo pueda tomar los valores pertenecientes a un conjunto de valores previamente establecido, o sea, un atributo tiene asociado un **dominio** de valores. En el caso anterior se tiene que el atributo *Precio* sólo puede tomar valores numéricos, mientras que el *Nombre* sólo puede contener frases textuales.

Los dominios a que puede pertenecer un atributo, suelen depender de los que proporcione el S.G.B.D. que empleemos. Suelen ser comunes dominios como: Texto, Número entero, Número decimal, Fecha, Hora, Sí/No, etc.

Por otro lado, un dominio como pueda ser Número entero, es un dominio cuyo conjunto de valores es infinito, y dado que trabajamos con ordenadores, es imprescindible poner un límite que permita almacenar un valor concreto debido a las limitaciones de memoria, y sobre todo al hecho de que toda tupla debe poseer el mismo tamaño. Tomemos como ejemplo el *Nombre* del *Plato,* que es evidentemente del tipo Texto; las limitaciones del ordenador nos impiden almacenar nombres ilimitadamente largos, como puedan ser «Magret de pato al vinagre de grosella con guarnición de higos malagueños al vino moscatel Pedro Ximénez». Es necesario, por tanto, establecer una cota al número máximo de letras que podemos teclear, por lo que el dominio del atributo *Nombre* puede ser Texto de 20 caracteres.

Así, la estructura completa de la tabla *Platos* quedaría como sigue:

Platos:

Código. Número entero.
Número con el que el plato aparece en la carta.
Nombre. Texto de 20 caracteres.
Nombre del plato.
Precio. Número decimal simple.
Precio del plato. no incluye I.V.A. ni descuentos, etc.
Menú. Sí/No.
Indica si ese plato puede ser consumido dentro del precio del menú del día.

La información anterior son los metadatos que definen la estructura de la tabla.

Algunos S.G.B.D. simplifican la tarea de indicar el dominio de un atributo, asignando un dominio por defecto, o estableciendo una jerarquía de dominios. P.ej., Microsoft Access® toma como dominio por defecto el Texto, y en concreto de 50 caracteres. Se pueden cambiar los dominios a uno cualquiera dentro de las siguientes posibilidades: Texto, Numérico, Fecha/Hora, Moneda, Sí/No, Memo, Autonumérico, Objeto OLE, o Hipervínculo. A su vez, cada uno de estos tipos se puede dividir en otras clases, una de las cuales es la que se toma por defecto; p.ej. el tipo Numérico tiene a su vez los subtipos: Byte, Entero, Entero largo, Simple y Doble. Si sólo escogemos Numérico, por defecto se toma el subtipo Entero largo, que permite guardar números enteros (sin parte decimal) desde el -2.147.483.648 hasta el 2.147.483.647. Si el subtipo por defecto no es el que se quiere puede especificarse otro.

Independientemente del dominio a que pertenezca un atributo, cualquier atributo puede tomar un valor especial que designa la ausencia de dato; es el valor NULO (en inglés *NULL* o *NIL)*. Cuando, por cualquier circunstancia, se desconoce el valor de un atributo, como p.ej. la dirección de un cliente, o bien ese atributo carece de sentido (el atributo Teléfono de un empleado que no tiene teléfono en su casa), podemos asociar a dicho atributo este valor especial, común a cualquier dominio. El equivalente en nuestro símil de las fichas de cartón sería dejar ese hueco en blanco. Hay que hacer notar la diferencia entre el valor NULO, y el valor «espacios en blanco»; el ordenador considera un espacio en blanco como cualquier otro carácter, por lo que a efectos computacionales, la introducción de caracteres «espacios en blanco» es distinta al concepto de «ausencia de información». Los espacios en blanco sí son datos, y pertenecen al dominio Texto.

Restricciones.

En el apartado anterior observamos que cada atributo está obligado a tomar un valor

perteneciente a un dominio concreto, siendo imposible el que guarde otro distinto. Esto supone una restricción sobre los atributos

Otras restricciones ya comentadas son:

- La imposibilidad de repetir tuplas en una misma tabla.
- La imposibilidad de establecer un orden en las tuplas, aunque algunos S.G.B.D. relajen un poco esta restricción.

En este apartado vamos a introducir otras restricciones más importantes que posee el modelo relacional, así como los conceptos implicados. Por último veremos las posibilidades que tiene el usuario para definir restricciones en función de las características propias de su trabajo.

Reglas fundamentales. Claves.

El modelo relacional intenta representar con una tabla a un tipo de objetos de la vida real, como puedan ser *Empleados, Clientes,* etc., e incluso considera las relaciones entre estos objetos como objetos en sí mismos. Para ello, designa cada tipo de objetos por un conjunto de atributos que son los que darán la «forma particular» a cada objeto. Volvamos al caso de nuestra tabla de *Platos*.

Para representar a un *Plato,* hemos escogido los atributos: *Código, Nombre, Precio* y *Menú*. Un plato concreto puede ser p.ej., (17, «Migas con chocolate», 850, Sí). Este plato concreto, como cualquier otro objeto distinguible del mundo real posee unas características que lo distinguen de los demás de su mismo tipo, tal y como lo hace el ADN de una persona. El ADN conforma la esencia o clave de toda persona. Es más, todo objeto posee algo concreto que lo hace diferente; incluso puede poseer más de una cosa por la que se le puede diferenciar: una persona puede distinguirse también por su nacionalidad y su D.N.I., lo que conforma otra clave de esa persona.

Como en una tabla, las tuplas pueden estar en cualquier orden, no podemos referenciar una tupla concreta mediante su posición entre las demás, y necesitamos alguna forma de seleccionar una tupla concreta. La forma de hacerlo es mediante una clave.

Una **clave** es un atributo o conjunto de atributos cuyo valor es único y diferente para cada tupla.

Cada tabla puede poseer más de una clave. P.ej., en nuestro caso de la tabla *Platos*, la clave puede ser tanto el atributo *Código*, como el atributo *Nombre*. Tenemos dos claves potenciales, también llamadas **claves candidatas**.

De entre todas las claves candidatas, el administrador, cuando define la tabla, debe decidir cuál de ellas va a ser la **clave principal o clave primaria**, mientras que el resto de las claves pasan a denominarse **claves alternativas o claves alternas**. La distinción entre clave principal y claves alternativas, es sólo a efectos de acceso interno a los datos, y para que el S.G.B.D. adopte ciertas decisiones sobre cómo almacenar esos datos en los sistemas de almacenamiento masivos.

Por otro lado, la clave de una tabla debe ser **propia**, o sea, ninguno de los atributos que la forman debe ser superfluo. Siguiendo con la tabla de *Platos*, si tomamos el grupo de atributos (*Código, Precio*), vemos que posee todas las características necesarias para considerarlo como una clave candidata. sin embargo, el atributo *Código* por sí sólo ya es una clave candidata, con lo cual el hecho de añadir el atributo *Precio* y crear una clave nueva, no aporta información de identificación, ya que el resto de los atributos (el *Código* en este caso) identifica por sí sólo a una tupla de la tabla de *Platos*. Así pues, el atributo *Precio* es superfluo en el grupo de atributos (*Código, Precio*), con lo que dicho grupo no es una clave candidata. Para distinguir cuando un grupo de atributos es clave o no, basta con probar con ir eliminando uno a uno cada uno de los atributos del grupo; si los atributos restantes siguen poseyendo las propiedades de clave, el atributo eliminado es superfluo, por lo que el grupo de atributos de partida no es clave propia.

Más adelante veremos como esta situación se deriva del hecho de que el valor del atributo quesobra (el superfluo) viene determinado por los valores de los otros atributos (la clave propia), en lo que se ha dado en llamar **dependencia funcional**.

El concepto de **clave** es tan importante, que da lugar a una serie de restricciones fundamentales sobre la base de datos. Son la **regla de identificación única** y la **regla de integridad referencial**.

Regla de identificación única.

Esta restricción ya fue estudiada en el tema de los diagramas E-R. En esencia, los conceptos de clave de una entidad en el diagrama E-R, y de clave de una tabla coinciden plenamente. Así pues, al igual que en aquel momento, podemos enunciar esta regla de la misma forma:

«En ninguna tupla de una tabla, ninguno de los atributos que formen parte de la clave primaria de una relación podrá tomar un valor nulo. El valor de la clave será único para cada tupla de la tabla».

Esta regla nos dice que una vez escogida la clave primaria de una tabla, y dado que ninguno de los atributos que la componen es superfluo, no podemos dejar de lado el valor de ninguno de ellos para identificar unívocamente a una tupla. O sea, el que todos los atributos de la tabla sean necesarios está en contradicción con que alguno de ellos pueda carecer de valor. Ningún atributo de la clave puede ser vacío porque en tal caso no sería una característica identificativa propia del objeto a que pertenece.

Regla de integridad referencial.

Esta otra regla, aunque también tiene una relación directa con el concepto de clave, está quizás más vinculada a la forma en que un diagrama E-R se convierte en un esquema relacional,

por lo que puede no ser bien comprendida en este punto. No obstante, por completitud se da aquí su enunciado, aunque su explicación la dejaremos para más adelante.

«Si una tupla de una tabla A posee atributos ($a_1..a_n$) que hacen referencia a la clave primaria de otra tupla de una tabla B, dichos atributos poseen, o bien valores nulos, o bien valores ($v_1..v_n$) que se corresponden con la clave de una tupla concreta de B».

Restricciones en la base de datos.

Las restricciones que hemos visto hasta ahora, son restricciones de clave aplicables a todas las bases de datos que siguen el modelo relacional, con independencia de su rango de aplicación o los datos que contengan. Restricciones de este tipo son también las que restringen los valores que puede contener una tupla, la imposibilidad de repetir tuplas o incluso de establecer un orden entre las tuplas de una tabla.

En este punto vamos a estudiar las reglas que el administrador impone a los datos que debe contener la base de datos. Son restricciones propias de la base de datos, y de la información que se pretende representar como por ejemplo que no haya pedidos que servir cuya fecha límite sea anterior a la fecha actual, que el total debido por un cliente no supere el riesgo máximo permitido, etc.

Son restricciones que en general limitan la estructura de la información almacenada en la base de datos, suministrando por tanto información sobre las características que respetan los datos guardados.

Este tipo de restricciones podemos dividirlas en tres grupos principales:

- Restricciones de atributo.
- Restricciones de tupla.
- Restricciones de tabla.
- Restricciones de base de datos.
- Restricciones por usuario.

Antes de hablar de las restricciones, es interesante echar un vistazo a conceptos importantes propios de la lógica matemática, y su aplicación a la informática.

Expresiones. Operadores relacionales. Predicados.

Cuando hablamos de restricciones, tenemos en mente una serie normas o reglas que deben cumplir los datos, ya sea por sí solos, o en relación con los demás. El problema reside en como expresar esas normas de manera que el ordenador pueda comprenderlas.

Para solucionar este problema, es necesario que bajemos un poco al nivel de la máquina, para convertir nuestras ideas en una secuencia de símbolos (sumas, restas, preguntas sobre si fulano es mayor que zutano, etc.) que dan lugar a una expresión final que sólo puede tomar dos

valores: o es verdad o es falso. Esta expresión final es lo que se llama **predicado**, que no tiene nada que ver con el análisis sintáctico de una frase en lingüística (sujeto, predicado, y esas cosas).

El predicado más simple que se puede formar es mediante la comparación entre dos cosas. Podemos comparar números para saber si son iguales, o uno es mayor que otro, podemos comparar frases (sólo desde el punto de vista lexicográfico, ya que el ordenador no comprende el significado que nosotros le damos a dichas frases) para saber si contienen palabras repetidas, si una frase subsume a otra, o simplemente para saber si una precede a otra o no según el orden alfabético. De esta forma aparecen los operadores relacionales, que fundamentalmente son:

> Mayor que
< Menor que
= Igual que
<= Menor o igual que (#)
>= Mayor o igual que ($)
<> Distinto a (¥)

Con estos operadores podemos preguntar si un número es mayor que otro o no, p.ej.:

$$6 > 8$$

a lo que el ordenador nos responderá: NO, o lo que es lo mismo FALSO.

En el ejemplo anterior hemos empleado tan sólo constantes: el 6 y el 8 son valores constantes. La verdadera potencia de este tipo de consultas se produce en el momento en que intervienen también variables o, en nuestro caso, atributos. P.ej.:

$$Descuento < 100$$

En este caso, el ordenador responderá VERDADERO o FALSO según el valor que en ese momento represente el atributo *Descuento*. En este caso, el atributo *Descuento* almacena el porcentaje de descuento a aplicar sobre una factura que posee una determinada *Base Imponible* en pesetas. Si en una tupla, el atributo *Descuento* tiene el valor 80, el predicado anterior tendrá como valor VERDADERO; pero si en otra tupla *Descuento* vale 120, el predicado valdrá FALSO.

Así, a las constantes, y a las variables o atributos, las llamamos **expresiones**. Aún más interesante es el hecho de poder emplear expresiones matemáticas más complejas, como por ejemplo, si queremos que el *Descuento* nunca sea mayor que la mitad del *IVA* aplicado, supuesto que ambos atributos almacenan un porcentaje, se emplearía el siguiente predicado:

$$Descuento < IVA / 2$$

Las expresiones empleados pueden llegar a ser muy complejas, pudiendo involucrar a varios atributos a la vez. Si por ejemplo tenemos además del descuento normal, un atributo *DPP* (Descuento por Pronto Pago), que no almacena un porcentaje si no una cantidad en pesetas, si queremos que este descuento no supere nunca al *Descuento* normal a que hacemos referencia desde elprincipio, nos encontramos con el problema de que *Descuento* posee un porcentaje mientras que *DPP* posee una cantidad en pesetas: NO SON DIRECTAMENTE COMPARABLES.

Para solucionar el problema, es necesario convertir alguno de los dos a la magnitud del

otro: o convertimos *Descuento* a una cantidad en pesetas o pasamos *DPP* a un porcentaje. En cualquiera de los dos casos, intervendrá el atributo *Base Imponible* imprescindible para esta conversión. Para convertir el *Descuento* en una cantidad concreta en pesetas, usaremos la expresión: (*Descuento* / 100) * *Base Imponible*, con lo que el predicado quedará:

DPP <= (*Descuento* / 100) * *Base Imponible*

Por otro lado, para convertir el *DPP* en un porcentaje, emplearemos la expresión: (*DPP* / *Base Imponible*) * 100, con lo que el predicado quedará:

(*DPP* / *Base Imponible*) * 100 <= *Descuento*

Como vemos, estos predicados expresan reglas que pueden cumplirse o no en una determinada tupla de una tabla. **Cuando exigimos que dicho predicado se cumpla siempre estamos hablando de una restricción.**

Hay algunas situaciones en las que estos predicados simples no son suficiente para expresar la restricción que queremos imponer. P.ej., ¿cómo expresar que el IVA sólo puede tomar los valores 3, 7, 16 ó 33? Con lo que hemos visto hasta ahora, podemos expresar por separado esta situación: *IVA* = 3, *IVA* = 7, *IVA* = 16, e *IVA* = 33. El problema radica en cómo expresar que deben cumplirse varios predicados simples simultáneamente.

Otra situación distinta aparece cuando, en lugar de exigir que se cumpla todo a la vez, queremos que se cumpla al menos una de ellas. P.ej. cómo expresar que uno de llos dos descuentos debe ser menor que el iva aplicado; por separado tenemos que *DPP* < *IVA*, *Descuento* < *IVA*. El problema radica en cómo expresar que debe cumplirse al menos un predicado simple.

Para solucionar estos problemas se emplean las conectivas u operadores lógicos. Cuando queremos que un predicado P_1 se cumpla **a la vez** que otro P_2, se emplea la conectiva **Y** (en inglés **AND**), dando lugar a P_1 AND P_2. Esta conectiva es asociativa, por lo que siguiendo el ejemplo anterior quedaría:

IVA = 3 AND *IVA* = 7 AND *IVA* = 16 AND *IVA* = 33

Cuando queremos que se cumpla **al menos** uno de los predicados P_1 o P_2 se emplea la conectiva **O** (en inglés **OR**), dando lugar a P_1 OR P_2. Esta conectiva es asociativa, y siguiendo con el segundo ejemplo quedaría:

DPP < *IVA* OR *Descuento* < *IVA*

Un tercer tipo de conectiva menos empleada es aquella que permite indicar que un predicado NO debe cumplirse; es la conectiva **NO** (en inglés **NOT**), y se aplica a un sólo predicado (al igual que un signo menos para indicar que un número es negativo). Por ejemplo, si queremos indicar que el *Descuento* nunca puede ser del 23% se haría algo así como:

NOT (*Descuento* = 23)

lo que puede conseguirse de forma similar mediante

Descuento <> 23

Por último, otra conectiva muy interesenta es la que obliga a que un predicado sea verdadero siempre y cuando se verifique otro previo. Es la conectiva **IMPLICA** (en inglés **IMPLIES**). P.ej., si no queremos aplicar descuentos superiores al 40% en facturas cuya base

imponible sea menor de 1000 ptas. crearemos el siguiente predicado:

Base Imponible < 1000 IMPLIES *Descuento* <= 40

lo que podemos leer como: «Si la *Base Imponible* es menor de 1000 **entonces** el *Descuento* no puede ser mayor de 40». Si la *Base Imponible* fuese mayor de 1000, entonces da igual el valor del *Descuento*: el predicado tomará el valor VERDADERO. Este predicado sólo se incumple (toma el valor de FALSO), cuando el primer predicado se cumple y el segundo no.

Un predicado que emplea cualquiera de estas conectivas recibe el nombre de **predicado complejo**.

Las conectivas anteriores pueden mezclarse entre sí utilizando paréntesis para dejar bien claro la forma en que los predicados simples deben agruparse entre sí.

De esta forma, de aquí en adelante, una restricción es ni más ni menos un predicado que debe tomar siempre el valor VERDADERO.

NOTA: Más adelante sería interesante hablar del concepto de función: Mes(), Hora(), etc.

Restricciones de atributo.

Hemos hablado del concepto de dominio, que se corresponde con los valores que puede tomar un atributo de una tabla. Normalmente, los dominios de que se dispone en cualquier S.G.B.D. son dominios muy generales, y que abarcan un conjunto de valores muy amplio que en la mayoría de los casos puede satisfacer las necesidades del administrador. No obstante, hay situaciones en las que se desea restringir aún más el conjunto de estos valores. P.ej., si en una tabla de *Albaranes* deseamos guardar un atributo *Descuento* que contendrá el porcentaje de dscuento a aplicar, está claro que *Descuento* no puede poseer un valor superior al 100%. Si el S.G.B.D. sólo permite decir si un atributo posee un valor entero o dcimal, ¿cómo indicar esta restricción sobre el campo *Descuento*?

La solución viene dada a través de las restricciones de atributo. Una restricción de atributo es un predicado en el que sólo pueden intervenir constantes y, por supuesto, dicho atributo.

De esta forma, la restricción del ejemplo anterior, quedaría:

Descuento < 100

Por supuesto, podemos emplear predicador más complejos. Si tenemos una tabla de *Clientes*, y deseamos saber su sexo para poder mandarles cartas personalizadas, emplearemos un atributo *Sexo* cuyo tipo será Texto de 1 carácter. Está claro que los únicos valores que vamos a permitir serán «V» (Varón) y «M» (Mujer). Esto puede conseguirse con el siguiente predicado:

Sexo = "V" OR *Sexo* = "M"

Hay algunos S.G.B.D. en los este tipo de restricciones puede indicarse de una forma más intuitiva. Dado que el único atributo que puede intervenir es aquél al que se refiere la propia restricción, podemos quitarlo de la expresión, dándolo por implícito:

= "V" OR = "M"

Restricciones de tupla.

En el caso anterior, los valores que podía tomar un determinado atributo, dependen exclusivamente de su propia condición, o sea, con independencia del resto de los atributos de la tupla a que pertenece. No siempre ocurre así, sino que a veces, los valores de ciertos atributos de una tupla deben poseer valores consistentes **entre sí**, y no de forma independiente. P.ej., si tenemos una tabla de *Clientes* y queremos almacenar datos de cada uno de ellos para hacerles la nómina, podemos incluir entre otros, los atributos de *Número de Hijos* y de *Retención IRPF*. Está claro que existe una relación entre ambos valores, de manera que si alguien no tiene hijos, su retención de IRPF no puede ser menor que un porcentaje mínimo establecido por ley, pongamos el 10%.

No podemos considerar esta restricción como una restricción de atributo, ya que existe una dualidad a la hora de considerar a qué atributo pertenece la restricción. Podemos verlo como una restricción de *Retención IRPF* que no puede ser menor de 10 si *Número de hijos* es igual a 0.

Pero, ¿por qué no verlo como una restricción de *Número de Hijos*? *Número de Hijos* no puede ser 0 si *Retención IRPF* es menor de 10.

Esta dualidad, producida por una restricción mutua entre dos o más atributos de una misma tupla es lo que da lugar a una restricción de tupla.

El predicado que soluciona este problema sería:

Número de hijos = 0 IMPLIES *Retención IRPF* > 10

Restricciones de tabla.

En otras situaciones, no es el valor de un atributo el que depende de los de los demás de la tupla a que pertenece, sino que es la tabla en sí la que depe preservar unas propiedades globales para que la información que posee sea consistente.

Por ejemplo, supongamos que estamos gestionando las prácticas que los alumnos de Hostelería deben realizar en distintos hoteles y rstaurantes de la costa. Estas prácticas vienen determinadas mediante una serie de turnos que cada alumno tiene asignado en distintas empresas. Para conseguir una enseñanza de calidad deseamos que cada alumno efectúe prácticas por al menos 50 horas, y por supuesto, no debe darse la circunstancia de que ningún alumno tenga turnos coincidentes, esto es, en lo que coincidan fecha y hora de la práctica en distintos lugares.

Alumno	Fecha	Inicio	Final	Lugar
Ana Bernal Gracia	12/2/98	12:00	17:00	Rte. Eurogallo
Juan Yáñez Pi	13/2/98	8:00	18:00	H. Esturión
Juan Yáñez Pi	14/2/98	12:00	18:00	Rte. La Plata
Ana Bernal Gracia	19/2/98	8:00	17:00	Rte. Eurogallo
Juan Yáñez Pi	14/2/98	8:00	18:00	Rte. Baco
Ana Bernal Gracia	26/2/98	12:00	18:00	H. Málaga
Juan Yáñez Pi	22/2/98	8:00	18:00	H. Esturión
Ana Bernal Gracia	3/3/98	8:00	17:00	H. Málaga
Ana Bernal Gracia	10/3/98	12:00	17:00	H. Mar Azul
Juan Yáñez Pi	10/3/98	8:00	18:00	H. Valhala
Ana Bernal Gracia	15/3/98	12:00	17:00	Rte. Eurogallo
Juan Yáñez Pi	22/3/98	8:00	18:00	Rte. Eurogallo
Juan Yáñez Pi	23/3/98	8:00	18:00	H. Estigia
Juan Yáñez Pi	30/3/98	8:00	18:00	Rte. Odín
Juan Yáñez Pi	7/4/98	8:00	18:00	Rte. Odín
Ana Bernal Gracia	7/4/98	12:00	17:00	H. Málaga

La figura siguiente representa una

situación que no queremos que ocurra. En esta tabla hemos representado tan sólo los datos referentes a dos de nuestros alumnos. Uno de ellos, *Juan Yáñez Pi* tiene asignadas 86 horas de prácticas, mientras que *Ana Bernal Gracia* tan sólo tiene 44 horas asignadas, incumpliendo la restricción que queremos imponer.

Pero aún hay más, las tupla marcadas con una flecha entran en contradicción, pues nos indican que de 12:00 a 18:00 Juan Yáñez Pi debe realizar prácticas en dos sitios distintos: el restaurante «La Plata», y el restaurante «Baco», lo cual es igualmente inadmisible.

Con objeto de evitar este tipo de situaciones, aparecen las restricciones de tabla. Una restricción de tabla es un predicado que engloba todas o parte de las tuplas de una misma tabla. su valor de VERDAD o FALSEDAD no puede ser encontrado si no es con el examen de dichas tuplas.

La forma de expresar este tipo de restricciones implica la utilización de expresiones y consultas que, por ahora, quedan fuera de nuestra visión. Baste decir que será necesario utilizar toda una batería de métodos de consulta que veremos más adelante en el apartado de SQL.

Restricciones de base de datos.

Las restricciones de base de datos son a las restricciones de tablas, lo que las de tupla son a las de atributo. Si en una restricción de atributo sólo podía intervenir un atributo, y en las de tupla podían intervenir varios (siempre de la misma tupla), las restricciones de base de datos son iguales que las de tabla con la salvedad de que pueden intervenir más de una tabla (de la misma base de datos, evidentemente).

Paquetes

Nombre	Destino	Comienzo	Duración	Plazas
Alpinismo	Katmandú	23/7/98	25	3
Aventura	Nairobi	15/4/98	12	5
Disney	Paris	10/9/98	5	15
Zares	Moscú	7/8/98	7	11

Supongamos p.ej. que trabajamos en una agencia de viajes en la que ofertamos paquetes de viajes; de cada paquete de viajes el número de plazas es limitado. Así, podemos almacenar información de los paquetes disponibles en una tabla como la de la figura.

A medida que vamos vendiendo los paquetes, iremos guardando en otra tabla el cliente que ha comprado el paquete, y el nombre del paquete comprado. De manera que tras algún tiempo, podemos tener la siguiente tabla de ventas.

Ventas

Paquete	Nombre
Disney	Juan Alonso Cortés
Alpinismo	Benito Galdeano Ruiz
Alpinismo	Francisco Ché Ordóñez
Aventura	Alfredo Jerez Tomelloso
Disney	Benito Beltrán Sánchez
Zares	Juan Ramírez Pray
Alpinismo	Antonio Cabeza Cano
Aventura	Pedro Calabrús Giménez
Disney	Pablo Chacón Rojas
Zares	Ernesto Troya Díaz
Alpinismo	Elías Sáenz de Heredia
Disney	Manuel Gálvez Robles

A poco que observemos esta tabla, vemos que hemos vendido cuatro paquetes de *Alpinismo*, cuando el número de plazas disponibles es tan sólo de tres. Estamos infringiendo una restricción de la base de

datos, y lo que es más, para darnos cuenta de ello hemos tenido que inspeccionar los datos de más de una tabla.

Para poder detectar automáticamente a través del S.G.B.D. este tipo de situaciones hemos de indicar una restricción de base de datos.

Al igual que ocurría en el caso anterior, el predicado correspondiente a esta restricción es complejo de especificar, y su estudio está supeditado al conocimiento de SQL, que veremos más adelante.

Restricciones de usuario.

Estas restricciones son muy diferentes a las anteriores, y suelen estar referidas a la inserción de datos pr parte de un usuario concreto.

Como veremos más adelante, cuando se crea una gran base de datos, su manejo y gestión no suele realizarlo una sola persona, sino que sobre ella trabajan muchas personas a la vez, cada una de las cuales suele efectuar unas operaciones concretas: el recepcionista suele dar entrada a los clientes (check in), les hace la factura cuando se marchan (check out), etc., mientras que el contable se centra exclusivamente en los asientos contables y no necesita saber nada de los clientes.

De esta forma, podemos hacer incluso que el S.G.B.D. sepa QUIÉN está manejando la base de datos en cada momento, mediante un sistema en el que antes de acceder a la base de datos, el S.G.B.D. solicita una identificación al usuario. En función del perfil que tenga asignado cada usuario (privilegios), podemos hacer que los datos que introduzca en la base de datos estén restringidos en todos los sentidos.

Por ejemplo, podemos permitir que cualquier persona haga un apunte en la factura de un cliente de un hotel. Sin embargo, si este apunte supera las 100000 ptas., sólo puede hacerse con el consentimiento de algún usuario especial como pueda ser p.ej. el gerente. De esta forma, el valor de un apunte en una factura está condicionado a quien es el usuario que inserta dicho apunte.

Este tipo de reglas suele ser muy complejo de manejar a través del S.G.B.D., y suelen sustituirse mediante los llamados **permisos de usuario**.

Conversión de diagramas E-R a esquemas relacionales.

En el tema 2 estudiamos un modelo conceptual de datos que nos permitía describir la información que se desea almacenar en una base de datos cualquiera: el modelo Entidad-Relación. La ventaja de este modelo es que es independiente del modelo lógico sobre el que se vaya a implantar finalmente dicha base de datos. Por otro lado, cuando dicho modelo lógico es el modelo relacional, resulta bastante sencillo pasar del diagrama E-R al esquema relacional mediante unas cuantas reglas sencillas y fáciles de aplicar.

Antes de comenzar es necesario resaltar las diferencias existentes entre estos dos modelos.

De una parte, el modelo E-R trabaja a nivel conceptual, estableciendo cuáles son las entidades fuertes y débiles que intervienen en nuestra base de datos, y las relaciones existentes entre ellas; sin embargo, no hace referencia alguna a la forma en que estos «objetos» se almacenan en ninguna base de datos, entre otras cosas por se trata **sólo de un modelo conceptual**. Por contra, el modelo relacional lo que trata es de representar la información en la forma en que se va a almacenar en la memoria del ordenador (o al menos en la forma en que el usuario la ve). Para ello se vale casi únicamente del concepto de **tabla**.

Por tanto, lo que se pretende con este apartado es pasar de describir conceptualmente el mundo mediante entidades y relaciones, a describirlo lógicamente mediante tablas.

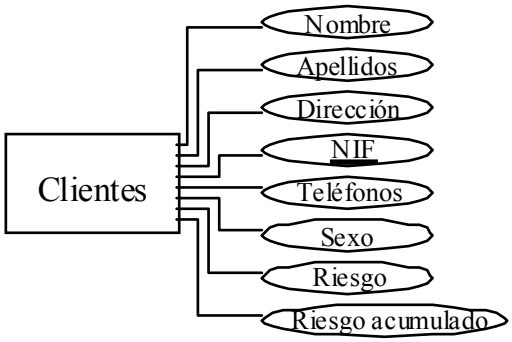
Utilización de atributos atómicos.

El modelo relacional impone una condición sobre las características que debe poseer cada columna de una tabla relacional. De hecho, el lector habrá notado que todas las columnas poseen datos simples, también llamados **atómicos**, de manera que cada uno de ellos representa una información unitaria y que no puede descomponerse en bloques de información más pequeños, (al menos desde el punto de vista del S.G.B.D.).

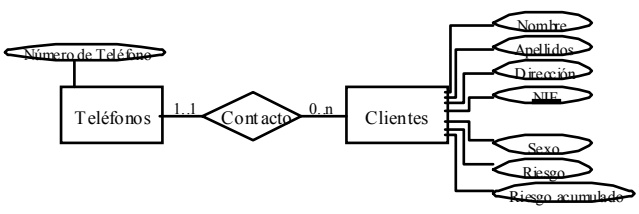
Por otro lado, el modelo E-R no impone esta restricción, sino que no dice nada sobre las características de los atributos, de manera que estos pueden contener información compuesta, o múltiple.

En el ejemplo adjunto, podemos observar una entidad *Clientes* que posee varios atributos, entre ellos *Teléfonos*; nada impide en el modelo E-R que *Teléfonos* almacene un solo número de teléfono, sino que podría almacenar varios según las necesidades con que se haya concebido la entidad.

Sin embargo, este atributo (llamado **atributo múltiple**, pues consiste en un mismo atributo atómico *Teléfono,* duplicado un número **indeterminado** de veces), no tiene representación correspondiente en el modelo relacional, por lo que es necesario algún tipo de transformación que convierta atributos múltiples en atributos atómicos.



Esta transformación es muy sencilla, y consiste únicamente en crear una entidad débil llamada *Teléfonos*, y relacionarla con *Clientes* a través de una relación débil. Esta nueva entidad poseerá un único atributo atómico *Número de Teléfono*. Así, el atributo múltiple se convierte en una entidad, ahora ya con atributos atómicos, de manera que cada *Cliente* se relacionará con



0, 1 ó muchas instancias de esta nueva entidad, con lo que el resultado es el mismo, con la salvedad de que en este nuevo diagrama sólo intervienen atributos atómicos.

Si el atributo múltiple pertenece a una relación, la solución es la misma, excepto por el hecho de que no es necesario crear una nueva relación, sino que la que poseía el atributo múltiple nos sirve además para conectar con la nueva entidad débil creada.

Algo parecido ocurre con los llamados **atributos compuestos**. Un atributo compuesto es aquél que puede dividirse en bloques de información más pequeños, pero a diferencia de los múltiples, estos bloques pequeños no son homogéneos, sino diferentes entre sí. El caso más común es el del atributos *Domicilio* de algunas entidades. Normalmente en el *Domicilio* de un *Cliente* o de un *Proveedor*, no es necesario distinguir entre calle, número, portal, planta, etc. Si en nuestro diseño deseamos distinguir efectivamente entre todos esos **componentes** del *Domicilio* entonces la única solución es crear atributos atómicos para cada uno de ellos.

Esta situación suele presentarse en raras ocasiones, y normalmente el diseño suele hacerse directamente en base a los atributos atómicos que pretenden distinguirse.

Conversión de entidades y relaciones a tablas.

Una vez preparados los atributos de las entidades y relaciones, la conversión del diagrama E-R al modelo relacional pasa por dos etapas: una en la que se convierten las entidades, y otra en la que se convierten las relaciones. No obstante, las tablas que se van obteniendo no adoptan su forma definitva hasta que se ha acabado el proceso.

Conversión de entidades a tablas.

«Una entidad A con atributos $a_1..a_n$ se convierte en una tabla de nombre A, y nombres de columna o atributos $a_1..a_n$. Si la clave de la entidad A está formada por los atributos $a_i..a_{i+k}$, la clave de la tabla correspondiente estará formada por dichos atributos».

En definitiva, podemos decir que existe una correspondencia directa entre el concepto de entidad del diagrama E-R (una vez eliminados los atributos múltiples y los compuestos), y el concepto de tabla relacional.

Clientes

Nombre	Apellidos	Dirección	NIF	Sexo	Riesgo	Riesgo acumulado

P.ej., siguiendo con el caso anterior, la entidad *Clientes* se convertiría en la tabla adjunta, en la que no hay ningún dato insertado.

Si la entidad es débil, será necesario incluir también los atributos correspondientes a su entidad fuerte, indispensables para poder establecer una clave identificativa en la tabla así

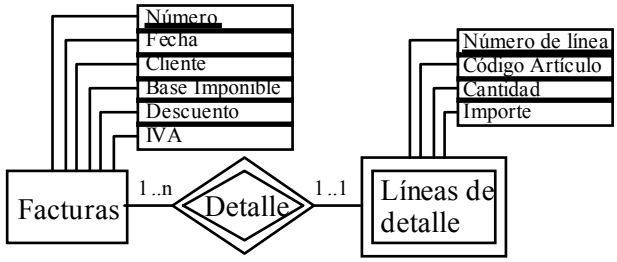
formada.

Conversión de relaciones binarias a tablas.

Conversión de relaciones es-un y relaciones débiles.

La conversión de relaciones es-un y relaciones binarias débiles en general, no conlleva realmente la aplicación de ninguna regla, ya que la propia conversión de la entidad débil en tabla convierte automática e implícitamente la relación débil también.

Supongamos por ejemplo el diagrama que nos permite representar las facturas propias de cualquier negocio. Dado que el número de líneas de detalle de una factura es indeterminado, es necesario crear una relación débil que relacione cada factura con los detalles que en ella se facturan, tal y como se ve en el diagrama adjunto.



Cuando convertimos las entidades *Facturas* y *Líneas de Detalle* en sus tablas correspondientes, obtenemos las de la figura, en la que se observa que la tabla de *Líneas de Detalle* hereda los atributos que forman la clave de *Facturas*.

Facturas

Número	Fecha	Cliente	Base Imponible	Descuento	IVA

Líneas de Detalle

Número de Factura	Número de Línea	Código Artículo	Cantidad	Importe

Con este método está claro cuales son las instancias de *Líneas de Detalle* que se relacionan con cada *Factura* concreta, ya que partiendo del *Número* de la *Factura* buscamos todas las tuplas de *Líneas de Detalle* en las que coincida su atributo *Número de Factura*. Por otro lado, averiguar a qué *Factura* pertenece una *Línea de Detalle* es trivial, todo caso que se conoce la clave de dicha *Factura* a través de *Número de Factura*.

De esta forma, la relación débil *Detalle* queda representada en el modelo relacional por la inclusión de la clave de la relación fuerte en la tabla de la débil.

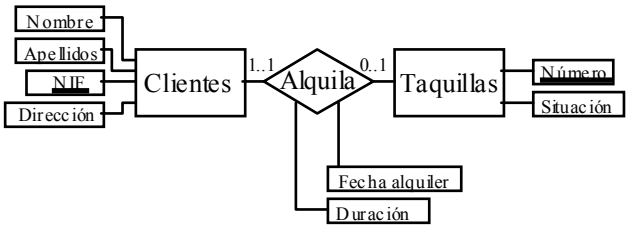
Una relación es-un se trata de la misma forma que cualquier otra relación binaria de debilidad.

Conversión de relaciones uno a uno.

La conversión de una relación uno a uno, no da lugar a una tabla nueva, sino que modifica una de las dos tablas correspondientes a las entidades que relaciona.

«Una relación R del tipo **uno a uno** con atributos $r_1..r_n$ que relaciona entidades A y B de claves $a_i..a_{i+k}$ y $b_j..b_{j+m}$, modifica la tabla de la entidad A, añadiéndole como atributos los de la clave de B, y los suyos propios, esto es $b_j..b_{j+m}$ y $r_1..r_n$».

Por ejemplo, supongamos el diagrama E-R de la figura que representa a una entidad *Clientes* y a una entidad *Taquillas*, en un sistema en el que queremos representar parte de un gimnasio, de manera que un cliente alquila una taquilla, y una taquilla sólo puede pertenecer como mucho a un cliente. Esta situación se representa mediante la relación *Alquila*, que en tal caso es del tipo **uno a uno**.



Tras convertir las entidades *Clientes* y *Taquillas* en tablas, se obtienen las de la figura adjunta.

Clientes

Nombre	Apellidos	NIF	Dirección

Taquillas

Número	Situación

Si ahora aplicamos la regla dada anteriormente, nos damos cuenta de su ambigüedad, en el sentido de que hace referencia a una entidad A y otra B. En nuestro caso, da lo mismo cual consideremos como entidad A (si a *Clientes* o a *Taquillas*), ya que el proceso a seguir es idéntico escojamos la que escojamos.

Supongamos que la entidad A es *Clientes*. en tal caso para convertir la relación *Alquila* con atributos *Fecha alquiler* y *Duración*, ampliaremos la tabla de *Clientes* con la clave de *Taquillas*, o sea *Número*, y los atributos de la relación, dando lugar a la figura siguiente.

Clientes

Nombre	Apellidos	NIF	Dirección	Número	Fecha alquiler	Duración

Como resultado de esta conversión, hemos transformado una tabla añadiéndole atributos que permiten seguir la relación existente entre un *Cliente* y una *Taquilla*. Podemos saber directamente qué *Taquilla* tiene asignada un *Cliente* sin más que consultar su clave, en este caso el atributo *Número*, que, por el hecho de ser clave, identifica de forma única una tupla en la tabla de *Taquillas*. Asimismo, acompañamos la adición de esta clave con la adición de los atributos propios de la relación, con lo que podemos saber qué *Taquilla* ha alquilado cada *Cliente*, en qué *Fecha alquiler* y por cuánta *Duración*.

Por otro lado, para saber a partir de un *Número* de *Taquilla*, qué *Cliente* la ha alquilado, basta con inspeccionar todas las tuplas de *Clientes* en busca de uno cuyo atributo *Número* coincida con el que estamos buscando.

Por tanto, lo que en el diagrama E-R no era más que un dibujo que relacionaba instancias de una entidad, lo hemos convertido en tablas y atributos insertados en ellas que nos permiten «seguir el hilo» de las instancias relacionadas.

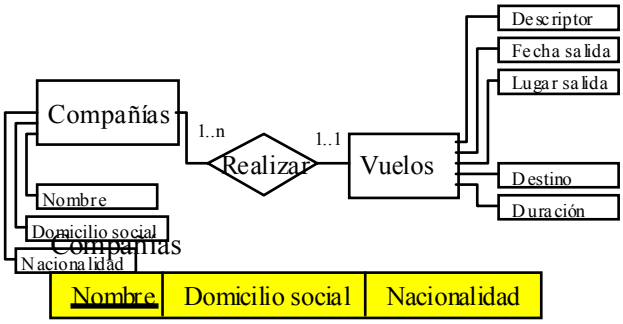
Esta operación, en la que la clave de una tabla «emigra» a otra, da lugar a lo que se llama **clave foránea**, que no es más que el conjunto de atributos que conforman la clave migrada.

Conversión de relaciones uno a muchos.

Cuando la relación que se desea convertir es del tipo uno a muchos, la solución es muy parecida a la del punto anterior, y consiste en migrar una de las claves a la tabla correspondiente a la otra entidad.

«Una relación R del tipo **uno a muchos** con atributos $r_1..r_n$ que relaciona entidades A y B de claves $a_i..a_{i+k}$ y $b_j..b_{j+m}$ de manera que una instancia de A se puede relacionar con muchas de B, modifica la tabla de la entidad B, añadiéndole como atributos los de la clave de A, y los suyos propios, esto es $a_i..a_{i+k}$ y $r_1..r_n$».

Típico ejemplo de esta situación es el diagrama E-R que representa la relación entre una lista de vuelos comerciales y las compañías aéreas que los realizan. Esto puede verse en la figura adjunta.



Tras haber convertido las entidades en tablas se obtienen las de la figura siguiente.

Compañías

Nombre	Domicilio social	Nacionalidad

Vuelos

Descriptor	Fecha salida	Lugar salida	Duración	Destino

En esta situación, para convertir la relación *Realizar* al modelo relacional, observamos que una *Compañía* se relaciona con muchos *Vuelos*, por lo que siguiendo la regla anterior, *Compañía* hace las veces de entidad A, y *Vuelos* hace las veces de entidad B. Por tanto, para convertir la relación, basta con incluir la clave de *Compañías* en la tabla de *Vuelos*, dando lugar al siguiente esquema.

Compañías

Nombre	Domicilio social	Nacionalidad

Vuelos

Descriptor	Fecha salida	Lugar salida	Duración	Destino	Nombre

En este caso, la relación *Realizar* carece de atributos propios por lo que no se añaden más atributos a la tabla de *Vuelos*.

En este punto es interesante hacer notar que en el diagrama E-R existe la posibilidad de tener entidades distintas con atributos distintos pero con el mismo nombre; p.ej., puede ser común tener la entidad *Clientes* con un atributo *NIF*, y a la vez tener la entidad *Empleados* con un atributo también llamdo *NIF*. Esto es posible porque cuando se hace referencia a *NIF*, es necesario también indicar la entidad a que nos referimos: *Clientes* o *Empleados*.

Sin embargo, en el momento de efectuar la conversión del diagrama a las tablas relacionales, vemos que en ciertas situaciones es necesario migrar las claves de unas entidades a otras, lo cual puede dar conflictos de nombres. Por ejemplo, ¿qué ocurriría si el atributo que forma la clave (destinado a guardar el código del vuelo: IB-713, AV-098, etc.), en lugar de llamarse *Descriptor* se llamase *Nombre*? Está claro que cuando se migrase la clave de la *Compañía* a la tabla de *Vuelos* habría un problema, pues tendríamos dos atributos con el mismo nombre.

Pues bien, tanto si se produce esa situación como si no, cuando se migra la clave de una tabla a otra, nada nos impide renombrar los atributos en su nueva ubicación. Por ejemplo, en el caso anterior, la tabla *Vuelos* podría haber quedado como se ve en la figura: el atributo *Nombre* ha pasado a llamarse *Nombre de Compañía*.

Lo que sí está claro, en cualquier caso, es que el atributo *Nombre de Compañía* sigue siendo clave foránea, aunque tenga distinto nombre.

Conversión de relaciones muchos a muchos.

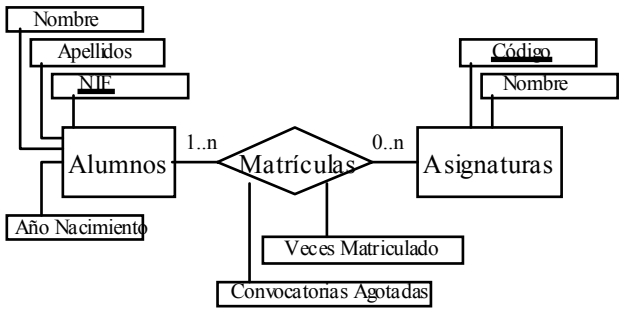
Este es el caso más general de conversión de relaciones, pudiendo incluso aplicarse en las relaciones uno a uno y uno a muchos. El único motivo por el que no se da esta regla como única regla general es la eficiencia, ya que como veremos implica la creación de tablas nuevas y la duplicación de información en gran cantidad. De hecho, también las reglas anteriores pueden ser refinadas con objeto de conseguir tablas más compactas y menos redundantes, pero a costa de enrarecer la comprensión de las mismas, razón por la que no las incluímos aquí.

«Una relación R del tipo **muchos a muchos** con atributos $r_1..r_n$ que relaciona entidades A y B de claves $a_i..a_{i+k}$ y $b_j..b_{j+m}$ respectivamente, se convierte en una tabla llamada R y compuesta por los atributos de las claves de A y B, así como por los atributos propios de la relación R, esto es $a_i..a_{i+k}$, $b_j..b_{j+m}$, y $r_1..r_n$. Los atributos $a_i..a_{i+k}$, $b_j..b_{j+m}$ forman la clave de la nueva tabla».

En los casos anteriores hemos visto como para convertir una relación uno a muchos (entre A y B) ampliamos una de las tablas correspondientes a una de las entidades que intervienen: la de B. Esto se puede hacer así porque cada instancia de B sólo puede relacionarse con una instancia de A, lo que en el esquema relacional puede plasmarse con la inserción de una sola clave foránea entre los atributos de la tabla asociada a B. Esta decisión no es simétrica, como ocurre en las relaciones uno a uno, o sea, no podemos ampliar la tabla de A con la clave foránea de B, ya que una instancia de A se puede relacionar con muchas de B, y necesitaríamos un número **indeterminado** de claves foráneas en A, lo cual no es lícito en el modelo relacional.

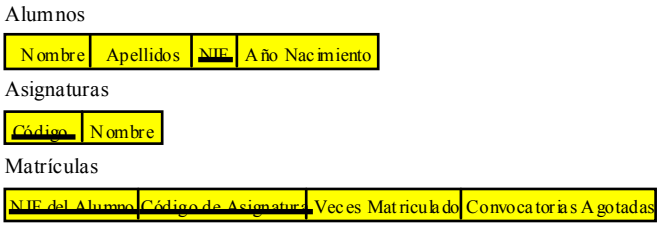
En el caso de las relaciones muchos a muchos ocurre lo mismo, pero esta vez respecto a las dos entidades A y B. No podemos ampliar ninguna de las tablas asociadas porque necesitaríamos un número indeterminado de claves foráneas. Por tanto, la solución pasa por crear una nueva tabla con el único objetivo de contener los pares de instancias que se relacionan; evidentemente, en lugar de repetir toda la información de cada instancia, se almacena tan sólo la información identificativa: la clave.

Para ilustrar esto, supongamos que queremos representar la información relativa a los alumnos de una Facultad y las asignaturas en que se halan matriculados. El diagrama E-R que representa esto puede verse en la figura.



Dado que la relación *Matrículas* es muchos a muchos, según la regla anterior, la conversión implica crear una nueva tabla con el mismo nombre, o sea *Matrículas*, y con los atributos *Veces Matriculado* y *Convocatorias Agotadas*, así como las claves de *Alumnos* y *Asignaturas*, o sea, *NIF* y *Código*, que podemos renombrar como *NIF del Alumno* y *Código de Asignatura*, quedando las tablas de la figura siguiente.

Con este esquema de tablas, para saber en qué asignaturas se ha matriculado un alumno concreto, basta con buscar todas las veces que aparezca su NIF en la tabla *Matrículas*; cada tupla en la que aparezca contendrá además la clave de una de las asignaturas en la que está matriculado. Para saber el nombre de cada asigntura utilizaremos el *Código de Asignatura* como clave para buscar el nombre en la tabla *Asignaturas*.



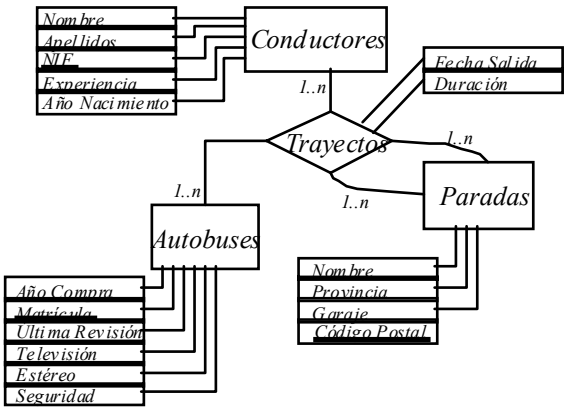
Es interesante hacer notar la necesidad de los atributos asociados a la relación, tal y como explicábamos en el capítulo de diagramas E-R.

Conversión de relaciones no binarias a tablas.

Hasta ahora hemos visto la forma de proceder para convertir sólo relaciones binarias. Muchos autores se quedan sólo aquí, ya que es posible convertir cualquier relación no binaria en varias binarias. No obstante, el método más general es permitir la existencia de relaciones que aglutinen a más de dos entidades, por lo que veremos aquí el método para convertirlas en tablas.

«Una relación R no binaria con atributos $r_1..r_n$ que relaciona entidades E_1, $E_2..E_n$ de claves $e_{i_1}..e_{i_1\%\,k_1}$, $e_{i_2}..e_{i_2\%\,k_2}$, $..e_{i_n}..e_{i_n\%\,k_n}$ respectivamente, se convierte en una tabla llamada R y compuesta por los atributos de las claves de E_1, $E_2..E_n$, así como por los atributos propios de la relación R, esto es $e_{i_1}..e_{i_1\%\,k_1}$, $e_{i_2}..e_{i_2\%\,k_2}$, $..e_{i_n}..e_{i_n\%\,k_n}$ y $r_1..r_n$. Los atributos $e_{i_1}..e_{i_1\%\,k_1}$, $e_{i_2}..e_{i_2\%\,k_2}$, $..e_{i_n}..e_{i_n\%\,k_n}$ forman la clave de la nueva tabla».

Esta regla es igual que la de conversión de relaciones binarias muchos a muchos, sólo que extendida a relaciones no binarias. Supongamos una Agencia de Transporte de Pasajeros por Carretera en la que tenemos varias delegaciones repartidas por todo el territorio nacional, y deseamos llevar un registro de cada viaje efectuado: Origen, Destino, Conductor y Autobús. Para ello, disponemos de las entidades *Conductores*, *Autobuses*, y *Paradas*. El registro que queremos viene dado por la relación *Trayectos*, tal y como vemos en la figura.



En este diagrama vemos que la entidad *Paradas* interviene dos veces en *Trayectos*, una como origen, y otra como destino. Asimismo vemos que la multiplicidad de todas las entidades en *Trayectos* es **a muchos**, aunque este hecho es indiferente a la hora de proceder a convertir *Trayectos* en una tabla relacional.

Después de aplicar las reglas de conversión de entidades, y la regla que nos ocupa en este punto, obtenemos las tablas de la figura.

Nótese que cuando una entidad interviene más de una vez en una relación que se convierte en tabla, es obligatoriamente necesario renombrar su clave foránea, ya que de otra forma tendríamos atributos con nombres duplicados. Es lo que ocurre con la entidad *Paradas*, cuya clave pasa a llamarse *CP Origen* o *CP Destino* según como participe en el *Trayecto*.

Conductores

Nombre	Apellidos	NIF	Experiencia	Año Nacimiento

Autobuses

Año Compra	Matrícula	Última Revisión	Televisión	Estéreo	Seguridad

Paradas

Nombre	Provincia	Garaje	Código Postal

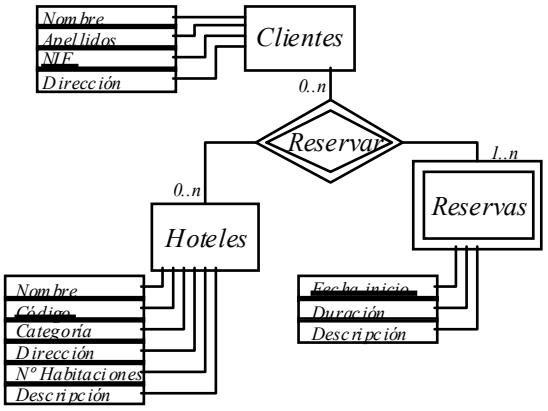
Trayectos

CP Origen	CP Destino	NIF Conductor	Matrícula	Fecha Salida	Duración

Conversión de relaciones débiles.

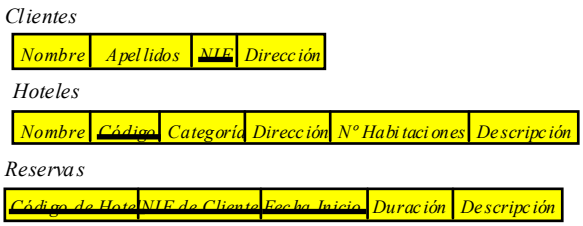
Como hemos visto, hay situaciones en las que una entidad débil no sólo depende de una fuerte, sino que puede depender de dos o más.

Supongamos por ejemplo que manejamos una cadena de hoteles, y que deseamos tener registradas las reservas efectuadas por los clientes. Esta necesidad queda solucionada por el diagrama E-R de la figura. Puede observarse como la entidad *Reservas* no puede sustituirse por la inclusión en *Reservar* de todos sus



atributos, ya que en tal caso un *Cliente* no podría efectuar dos reservas en el mismo *Hotel* en fechas distintas.

Estando así las cosas, las tablas correspondientes a las entidades *Clientes*, *Hoteles* y *Reservas*, serían como las de la figura siguiente, en la que podemosver como *Reservas* ha heredado los atributos de las claves de *Clientes* y *Hoteles*. De esta forma, estamos representando implícitamente la relación *Reservar*, ya que las claves heredadas por *Reservas* nos permiten «seguir elhilo» hasta las instancias de *Clientes* y de *hoteles* a que pertenecen. Igualmente podemos emplear estas claves foráneas para saber las reservadas efectuadas en un *Hotel* determinado o por un *Cliente* concreto.



Normalización de esquemas relacionales.

Como se ha podido comprobar a lo largo de este estudio sobre los diagramas E-R y sobre el modelo relacional, la creación de un conjunto de tablas a partir de unas necesidades dadas no es ni mucho menos una labor algorítmica en la que sólo existe una solución posible. De hecho, distintos administradores expertos pueden proponer distintos conjuntos detablas como solución almismo problema, sin que ninguna de ellas tenga que primar sobre las demás como mejor aproximación. Simplemente cada esquema propuesto se centrará en mejorar el comportamiento respecto a determinadas características del problema, solucionando el resto con una eficiencia aceptable aunque, aquizás, no óptima en todos los casos.

En cualquier caso, y a pesar de que el proceso no es ni mucho menos automático, existen una serie de reglas generales, que aplicadas a cualquier esquema relacional, sea cual sea el problema que intente solucionar, aumentan la legibilidad y la potencia de las tablas, a la vez que disminuyen la redundancia, y los posibles problemas de concordancia entre los datos, incrementando también la utilidad de la base de datos, y la confianza que podemos depositar en la información representada. La aplicación de estas reglas a las tablas relacionales, y su transformación en un nuevo conjunto de tablas que, aunque representan lo mismo, posee propiedades de mayor calidad, es lo que se ha dado en llamar **Normalización**.

La normalización se fundamenta en dos conceptos principales: las dependencias funcionales (en todas sus variedades), y los determinantes. En cualquier texto que trate sobre el tema, podemos encontrar fundamentalmente seis reglas a aplicar sobre las tablas, de forma que la aplicación de cada una de ellas hace que el esquema pase a estar en una **forma normal** concreta. Así, la aplicación de la 1ª reglas hara que las tablas se hallen en 1ª forma normal; la aplicación de las reglas 1ª y 2ª, pasan a 2ª forma normal, y así sucesivamente. Por tanto, cada forma normal asegura una características de mayor calidad en las tablas obtenidas.

Sin embargo, nosotros sólo trabajaremos con dos de estas reglas: la que da lugar a la 1ª forma normal, y la llamada forma normal de Boyce-Codd. a partir de aquí podemos aplicar todavía la 4ª y 5ª formas normales, pero ello está fuera de nuestros objetivos, ya que implica conceptos de mayor nivel innecesarios para nuestros propósitos.

Comenzaremos, por tanto con los conceptos de **dependencias funcionales** y de **determinantes**.

Dependencias funcionales.

Las dependencias funcionales son de primordial importancia a la hora de encontrar y eliminar la redundancia de los datos almacenados en las tablas de una base de datos relacional. aunque existen varios tipos de dd. ff., nosotros sólo vamos a trabajar con el tipo básico, que es el que nos permitirá llegar hasta la forma normal de Boyce-Codd.

Podemos definir una dependencia funcional de la siguiente manera:

«

Determinantes.

1ª Forma Normal.

Forma Normal de Boyce-Codd.