CAPÍTULO 7

Diseño de bases de datos relacionales

En este capítulo se considera el problema de diseñar el esquema de una base de datos relacional. Muchos de los problemas que conlleva son parecidos a los de diseño que se han considerado en el Capítulo 6 en relación con el modelo E-R.

En general, el objetivo del diseño de una base de datos relacional es la generación de un conjunto de esquemas de relación que permita almacenar la información sin redundancias innecesarias, pero que también permita recuperarla fácilmente. Esto se consigue mediante el diseño de esquemas que se hallen en la *forma normal* adecuada. Para determinar si el esquema de una relación se halla en una de las formas normales deseables es necesario obtener información sobre la empresa real que se está modelando con la base de datos. Parte de esa información se halla en un diagrama E-R bien diseñado, pero puede ser necesaria información adicional sobre la empresa.

En este capítulo se introduce un enfoque formal al diseño de bases de datos relacionales basado en el concepto de dependencia funcional. Posteriormente se definen las formas normales en términos de las dependencias funcionales y de otros tipos de dependencias de datos. En primer lugar, sin embargo, se examina el problema del diseño relacional desde el punto de vista de los esquemas derivados de un diseño entidad-relación dado.

7.1 Características de los buenos diseños relacionales

El estudio del diseño entidad-relación llevado a cabo en el Capítulo 6 ofrece un excelente punto de partida para el diseño de bases de datos relacionales. Ya se vio en el Apartado 6.9 que es posible generar directamente un conjunto de esquemas de relación a partir del diseño E-R. Evidentemente, la adecuación (o no) del conjunto de esquemas resultante depende, en primer lugar, de la calidad del diseño E-R. Más adelante en este capítulo se estudiarán maneras precisas de evaluar la adecuación de los conjuntos de esquemas de relación. No obstante, es posible llegar a un buen diseño empleando conceptos que ya se han estudiado.

Para facilitar las referencias, en la Figura 7.1 se repiten los esquemas del Apartado 6.9.7.

7.1.1 Alternativa de diseño: esquemas grandes

A continuación se examinan las características de este diseño de base de datos relacional y algunas alternativas. Supóngase que, en lugar de los esquemas *prestatario* y *préstamo*, se considerase el esquema:

$$prestatario\_préstamo = (id\_cliente, número\_préstamo, importe)$$

Esto representa el resultado de la reunión natural de las relaciones correspondientes a *prestatario* y a *préstamo*. Parece una buena idea, ya que es posible expresar algunas consultas empleando menos reuniones, hasta que se considera detenidamente la entidad bancaria que condujo al diseño E-R. Obsérvese

sucursal = (*nombre_sucursal*, *ciudad_sucursal*, *activos*)
cliente = (*id_cliente*, *nombre_cliente*, *calle_cliente*, *ciudad_cliente*)
préstamo = (*número_préstamo*, *importe*)
cuenta = (*número_cuenta*, *saldo*)
empleado = (*id_empleado*, *nombre_empleado*, *número_teléfono*, *fecha_contratación*)
nombre_subordinado = (*id_empleado*, *nombre_subordinado*)
sucursal_cuenta = (*número_cuenta*, *nombre_sucursal*)
sucursal_préstamo = (*número_préstamo*, *nombre_sucursal*)
prestatario = (*id_cliente*, *número_préstamo*)
impositor = (*id_cliente*, *número_cuenta*)
asesor = (*id_cliente*, *id_empleado*, *tipo*)
trabaja_para = (*id_empleado_trabajador*, *id_empleado_jefe*)
pago = (*número_préstamo*, *número_pago*, *fecha_pago*, *importe_pago*)
cuenta_ahorro = (*número_cuenta*, *tasa_interés*)
cuenta_corriente = (*número_cuenta*, *importe_descubierto*)

Figura 7.1 Los esquemas bancarios del Apartado 6.9.7.

que el conjunto de relaciones *prestatario* es varios a varios. Esto permite que cada cliente tenga varios préstamos y, también, que se puedan conceder préstamos conjuntamente a varios clientes. Se tomó esta decisión para que fuese posible representar los préstamos realizados conjuntamente a matrimonios o a consorcios de personas (que pueden participar en una aventura empresarial conjunta). Ése es el motivo de que la clave primaria del esquema *prestatario* consista en *id_cliente* y *número_préstamo* en lugar de estar formada sólo por *número_préstamo*.

Considérese un préstamo que se concede a uno de esos consorcios y considérense las tuplas que deben hallarse en la relación del esquema *prestatario_préstamo*. Supóngase que el préstamo P-100 se concede al consorcio formado por los clientes Santiago (con identificador de cliente 23652), Antonio (con identificador de cliente 15202) y Jorge (con identificador de cliente 23521) por un importe de 10.000 €.

La Figura 7.2 muestra la forma en que es posible representar esta situación empleando *préstamo* y *prestatario* y con el diseño alternativo usando *prestatario_préstamo*. La tupla (P-100, 10.000) de la relación del esquema *préstamo* se reúne con tres tuplas de la relación del esquema *prestatario*, lo que genera tres tuplas de la relación del esquema *prestatario_préstamo*. Obsérvese que en *prestatario_préstamo* es necesario repetir una vez el importe del préstamo por cada cliente integrante del consorcio que solicita el préstamo. Es importante que todas estas tuplas concuerden en lo relativo al importe del préstamo P-100, ya que, en caso contrario, la base de datos quedaría en un estado inconsistente. En el diseño original que utiliza *préstamo* y *prestatario* se almacenaba el importe de cada préstamo exactamente una vez. Esto sugiere que el empleo de *prestatario_préstamo* no es buena idea, ya que se almacena el importe de cada préstamo de manera redundante y se corre el riesgo de que algún usuario actualice el importe del préstamo en una de las tuplas pero no en todas, y, por tanto, genere inconsistencia.

Considérese ahora otra alternativa. Sea *préstamo_importe_sucursal* = (*número_préstamo*, *importe*, *nombre_sucursal*), creada a partir de *préstamo* y de *sucursal_préstamo* (mediante una reunión de las relaciones correspondientes). Esto se parece al ejemplo que se acaba de considerar, pero con una importante diferencia. En este caso, *número_préstamo* es la clave primaria de los dos esquemas, *sucursal_préstamo* y *préstamo* y, por tanto, también lo es de *préstamo_importe_sucursal*. Esto se debe a que el conjunto de relaciones *sucursal_préstamo* está definido como varios a uno, a diferencia del conjunto de relaciones *prestatario* del ejemplo anterior. Para cada préstamo sólo hay una sucursal asociada. Por tanto, cada número de préstamo concreto sólo aparece una vez en *sucursal_préstamo*. Supóngase que el préstamo P-100 está asociado con la sucursal de Soria. La Figura 7.3 muestra la manera en que esto se representa empleando *préstamo_importe_sucursal*. La tupla (P-100, 10.000) de la relación del esquema *préstamo* se reúne con una sola tupla de la relación del esquema *sucursal_préstamo*, lo que sólo genera una tupla de la relación del esquema *préstamo_importe_sucursal*. No hay repetición de información en *préstamo_importe_sucursal* y, por tanto, se evitan los problemas que se encontraron en el ejemplo anterior.

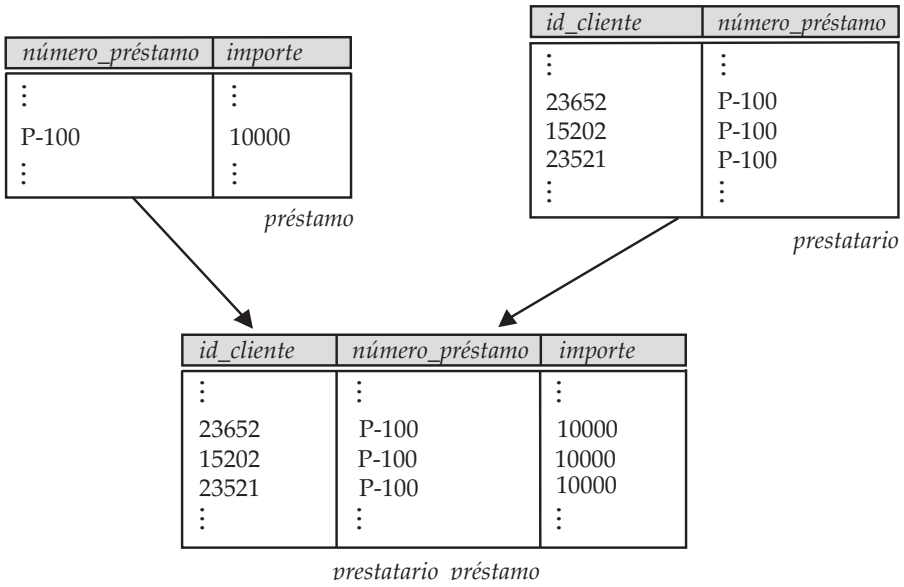

Figura 7.2 Lista parcial de las tuplas de las relaciones *préstamo*, *prestatario* y *prestatario_préstamo*.

Antes de acceder finalmente al empleo de *préstamo_importe_sucursal* en lugar de *sucursal_préstamo* y de *préstamo*, hay otro aspecto que se debe considerar. Puede que se desee registrar en la base de datos un préstamo y su sucursal asociada antes de que se haya determinado su importe. En el diseño antiguo, el esquema *sucursal_préstamo* puede hacerlo pero, con el diseño revisado *préstamo_importe_sucursal*, habría que crear una tupla con un valor nulo para *importe*. En algunos casos los valores nulos pueden crear problemas, como se vio en el estudio de SQL. No obstante, si se decide que en este caso no suponen ningún problema, se puede aceptar el empleo del diseño revisado.

Los dos ejemplos que se acaban de examinar muestran la importancia de la naturaleza de las claves primarias al decidir si las combinaciones de esquemas tienen sentido. Se han presentado problemas —concretamente, la repetición de información—cuando el atributo de reunión (*número_préstamo*) no era la clave primaria de *los dos* esquemas que se combinaban.

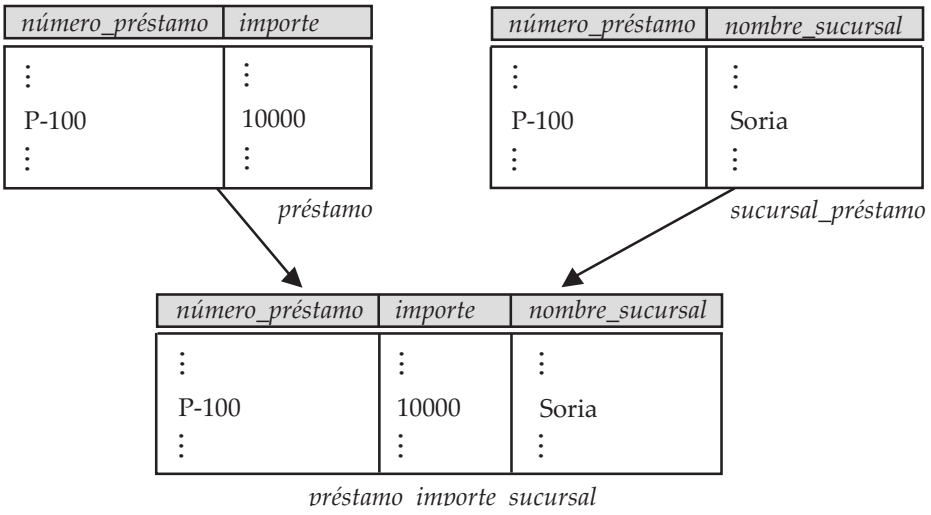

Figura 7.3 Lista parcial de las tuplas de las relaciones *préstamo*, *sucursal_préstamo* y *préstamo_importe_sucursal*.

7.1.2 Alternativa de diseño: esquemas pequeños

Supóngase nuevamente que, de algún modo, se ha comenzado a trabajar con el esquema *prestatario_préstamo*. ¿Cómo se reconoce que necesita que haya repetición de la información y que hay que dividirlo en dos esquemas, *prestatario* y *préstamo*? Como no se dispone de los esquemas *prestatario* y *préstamo*, se carece de la información sobre la clave primaria que se empleó para describir el problema de *prestatario_préstamo*.

Mediante la observación del contenido de las relaciones reales del esquema *prestatario_préstamo* se puede percibir la repetición de información resultante de tener que relacionar el importe del préstamo una vez por cada prestatario asociado con cada uno de ellos. Sin embargo, se trata de un proceso en el que no se puede confiar. Las bases de datos reales tienen gran número de esquemas y un número todavía mayor de atributos. El número de tuplas puede ser del orden de millones. Descubrir la repetición puede resultar costoso. Hay un problema más fundamental todavía en este enfoque. No permite determinar si la carencia de repeticiones es meramente un caso especial "afortunado" o la manifestación de una regla general. En el ejemplo anterior, ¿cómo se *sabe* que en la entidad bancaria cada préstamo (identificado por su número correspondiente) sólo *debe* tener un importe? ¿El hecho de que el préstamo P-100 aparezca tres veces con el mismo importe es sólo una coincidencia? Estas preguntas no se pueden responder sin volver a la propia empresa y comprender sus reglas de funcionamiento. En concreto, hay que averiguar si el banco exige que cada préstamo (identificado por su número correspondiente) *sólo* tenga un importe.

En el caso de *prestatario_préstamo*, el proceso de creación del diseño E-R logró evitar la creación de ese esquema. Sin embargo, esta situación fortuita no se produce siempre. Por tanto, hay que permitir que el diseñador de la base de datos especifique normas como "cada valor de *número_préstamo* corresponde, como máximo, a un *importe*", incluso en los casos en los que *número_préstamo* no sea la clave primaria del esquema en cuestión. En otras palabras, hay que escribir una norma que diga "si hay un esquema (*número_préstamo, importe*), entonces *número_préstamo* puede hacer de clave primaria". Esta regla se especifica como la **dependencia funcional** *número_préstamo* → *importe*. Dada esa regla, ya se tiene suficiente información para reconocer el problema del esquema *prestatario_préstamo*. Como *número_préstamo* no puede ser la clave primaria de *prestatario_préstamo* (porque puede que cada préstamo necesite varias tuplas de la relación del esquema *prestatario_préstamo*), tal vez haya que repetir el importe del préstamo.

Observaciones como éstas y las reglas (especialmente las dependencias funcionales) que generan permiten al diseñador de la base de datos reconocer las situaciones en que hay que dividir, o descomponer, un esquema en dos o más. No es difícil comprender que la manera correcta de descomponer *prestatario_préstamo* es dividirlo en los esquemas *prestatario* y *préstamo*, como en el diseño original. Hallar la descomposición correcta es mucho más difícil para esquemas con gran número de atributos y varias dependencias funcionales. Para trabajar con ellos hay que confiar en una metodología formal que se desarrolla más avanzado este capítulo.

No todas las descomposiciones de los esquemas resultan útiles. Considérese el caso extremo en que todo lo que tengamos sean esquemas con un solo atributo. No se puede expresar ninguna relación interesante de ningún tipo. Considérese ahora un caso menos extremo en el que se decida descomponer el esquema *empleado* en

$$empleado1 = (\underline{id\_empleado}, nombre\_empleado)$$
$$empleado2 = (\underline{nombre\_empleado}, número\_teléfono, fecha\_contratación)$$

El problema con esta descomposición surge de la posibilidad de que la empresa tenga dos empleados que se llamen igual. Esto no es improbable en la práctica, ya que muchas culturas tienen algunos nombres muy populares y, además, puede que los niños se llamen como sus padres. Por supuesto, cada persona tiene un identificador de empleado único, que es el motivo de que se pueda utilizar *id_empleado* como clave primaria. A modo de ejemplo, supóngase que dos empleados llamados Koldo trabajan para el mismo banco y tienen las tuplas siguientes de la relación del esquema *empleado* del diseño original:

(1234567890, Koldo, 918820000, 29/03/1984)
(9876543210, Koldo, 918699999, 16/01/1981)

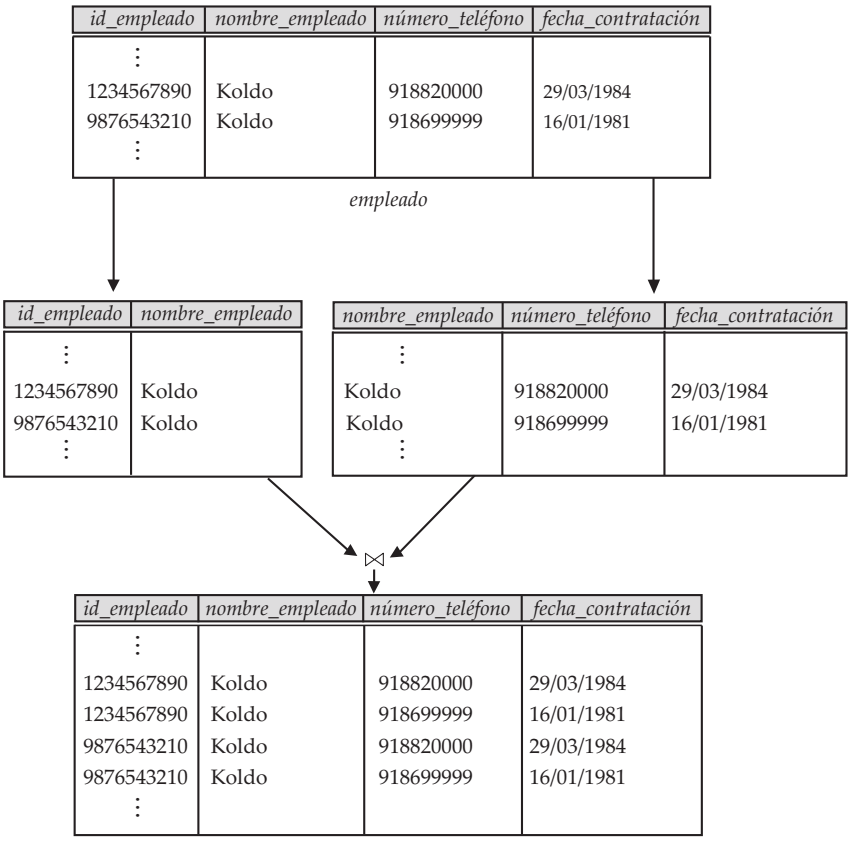

Figura 7.4 Pérdida de información debida a una mala descomposición.

La Figura 7.4 muestra estas tuplas, las tuplas resultantes al utilizar los esquemas procedentes de la descomposición y el resultado que se obtendría si se intentara volver a generar las tuplas originales mediante una reunión natural. Como se ve en la figura, las dos tuplas originales aparecen en el resultado junto con dos tuplas nuevas que mezclan de manera incorrecta valores de fechas correspondientes a los dos empleados llamados Koldo. Aunque se dispone de más tuplas, se tiene en realidad menos información en el sentido siguiente: se puede indicar que un determinado número de teléfono y una fecha de contratación dada corresponden a alguien llamado Koldo, pero no se puede distinguir a cuál de ellos. Por tanto, la descomposición propuesta es incapaz de representar algunos datos importantes de la entidad bancaria. Evidentemente, es preferible evitar este tipo de descomposiciones. Estas descomposiciones se denominan **descomposiciones con pérdidas**, y, a la inversa, aquéllas que no tienen pérdidas se denominan **descomposiciones sin pérdidas**.

7.2 Dominios atómicos y la primera forma normal

El modelo E-R permite que los conjuntos de entidades y los de relaciones tengan atributos con algún tipo de subestructura. Concretamente, permite atributos multivalorados, como *nombre_subordinado* de la Figura 6.25, y atributos compuestos (como el atributo *dirección* con los atributos componentes *calle* y *ciudad*). Cuando se crean tablas a partir de los diseños E-R que contienen ese tipo de atributos, se elimina esa subestructura. Para los atributos compuestos, se deja que cada atributo componente sea un atributo de pleno derecho. Para los atributos multivalorados, se crea una tupla por cada elemento del conjunto multivalorado.

En el modelo relacional se formaliza esta idea de que los atributos no tienen subestructuras. Un dominio es **atómico** si se considera que los elementos de ese dominio son unidades indivisibles. Se dice que

el esquema de la relación R está en la **primera forma normal** (1FN) si los dominios de todos los atributos de R son atómicos.

Los conjuntos de nombres son ejemplos de valores no atómicos. Por ejemplo, si el esquema de la relación *empleado* incluyera el atributo *hijos*, los elementos de cuyo dominio son conjuntos de nombres, el esquema no estaría en la primera forma normal.

Los atributos compuestos, como el atributo *dirección* con los atributos componentes *calle* y *ciudad*, tienen también dominios no atómicos.

Se da por supuesto que los números enteros son atómicos, por lo que el conjunto de los números enteros es un dominio atómico; el conjunto de todos los conjuntos de números enteros es un dominio no atómico. La diferencia estriba en que normalmente no se considera que los enteros tengan subpartes, pero sí se considera que las tienen los conjuntos de enteros—es decir, los enteros que componen el conjunto. Pero lo importante no es lo que sea el propio dominio, sino el modo en que se utilizan los elementos de ese dominio en la base de datos. El dominio de todos los enteros sería no atómico si se considerara que cada entero es una lista ordenada de cifras.

Como ilustración práctica del punto anterior, considérese una organización que asigna a los empleados números de identificación de la manera siguiente: las dos primeras letras especifican el departamento y las cuatro cifras restantes son un número único para cada empleado dentro de ese departamento. Ejemplos de estos números pueden ser $CS0012$ y $EE1127$. Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de la relación tuviera un atributo cuyo dominio consistiera en números de identificación así codificados, el esquema no se hallaría en la primera forma normal.

Cuando se utilizan números de identificación de este tipo, se puede averiguar el departamento de cada empleado escribiendo código que analice la estructura de los números de identificación. Ello exige programación adicional, y la información queda codificada en el programa de aplicación en vez de en la base de datos. Surgen nuevos problemas si se utilizan esos números de identificación como claves primarias: cada vez que un empleado cambie de departamento, habrá que modificar su número de identificación en todos los lugares en que aparezca, lo que puede constituir una tarea difícil; en caso contrario, el código que interpreta ese número dará un resultado erróneo.

El empleo de atributos con el valor definido por un conjunto puede llevar a diseños con almacenamiento de datos redundante; lo que, a su vez, puede dar lugar a inconsistencias. Por ejemplo, en lugar de representar la relación entre las cuentas y los clientes como la relación independiente *impositor*, puede que el diseñador de bases de datos esté tentado a almacenar un conjunto de *titulares* con cada cuenta, y un conjunto de *cuentas* con cada cliente. Siempre que se cree una cuenta, o se actualice el conjunto de titulares de una cuenta, habrá que llevar a cabo la actualización en dos lugares; no llevar a cabo esas dos actualizaciones puede dejar la base de datos en un estado inconsistente. Guardar sólo uno de esos conjuntos evitaría la información repetida, pero complicaría algunas consultas.

Algunos tipos de valores no atómicos pueden resultar útiles, aunque deben utilizarse con cuidado. Por ejemplo, los atributos con valores compuestos suelen resultar útiles, y los atributos con el valor determinado por un conjunto también resultan útiles en muchos casos, que es el motivo por el que el modelo E-R los soporta. En muchos dominios en los que las entidades tienen una estructura compleja, la imposición de la representación en la primera forma normal supone una carga innecesaria para el programador de las aplicaciones, que tiene que escribir código para convertir los datos a su forma atómica. También hay sobrecarga en tiempo de ejecución por la conversión de los datos entre su forma habitual y su forma atómica. Por tanto, el soporte de los valores no atómicos puede resultar muy útil en ese tipo de dominios. De hecho, los sistemas modernos de bases de datos soportan muchos tipos de valores no atómicos, como se verá en el Capítulo 9. Sin embargo, en este capítulo nos limitamos a las relaciones en la primera forma normal y, por tanto, todos los dominios son atómicos.

7.3 Descomposición mediante dependencias funcionales

En el Apartado 7.1 se indicó que hay una metodología formal para evaluar si un esquema relacional debe descomponerse. Esta metodología se basa en los conceptos de clave y de dependencia funcional.

7.3.1 Claves y dependencias funcionales

Las claves y, más en general, las dependencias funcionales son restricciones de la base de datos que exigen que las relaciones cumplan determinadas propiedades. Las relaciones que cumplen todas esas restricciones son **legales**.

En el Capítulo 6 se definió el concepto de *superclave* de la manera siguiente. Sea R el esquema de una relación. El subconjunto C de R es una **superclave** de R si, en cualquier relación legal $r(R)$, para todos los pares t_1 y t_2 de tuplas de r tales que $t_1 \neq t_2$, $t_1[C] \neq t_2[C]$. Es decir, ningún par de tuplas de una relación legal $r(R)$ puede tener el mismo valor del conjunto de atributos C.

Mientras que una clave es un conjunto de atributos que identifica de manera unívoca toda una tupla, una dependencia funcional permite expresar restricciones que identifican de manera unívoca el valor de determinados atributos. Considérese el esquema de una relación R y sean $\alpha \subseteq R$ y $\beta \subseteq R$. La **dependencia funcional** $\alpha \rightarrow \beta$ se cumple para el esquema R si, en cualquier relación legal $r(R)$, para todos los pares de tuplas t_1 y t_2 de r tales que $t_1[\alpha] = t_2[\alpha]$, también se cumple que $t_1[\beta] = t_2[\beta]$.

Empleando la notación para la dependencia funcional, se dice que C es superclave de R si $C \rightarrow R$. Es decir, C es superclave si, siempre que $t_1[C] = t_2[C]$, también se cumple que $t_1[R] = t_2[R]$ (es decir, $t_1 = t_2$).

Las dependencias funcionales permiten expresar las restricciones que no se pueden expresar con superclaves. En el Apartado 7.1.2 se consideró el esquema

$$\textit{prestatario\_préstamo} = (\underline{\textit{id\_cliente, número\_préstamo}}, \textit{importe})$$

en el que se cumple la dependencia funcional *número_préstamo* → *importe*, ya que por cada préstamo (identificado por *número_préstamo*) hay un solo importe. El hecho de que el par de atributos (*id_cliente, número_préstamo*) forma una superclave para *prestatario_préstamo* se denota escribiendo

$$\textit{id\_cliente, número\_préstamo} \rightarrow \textit{id\_cliente, número\_préstamo, importe}$$

o, de manera equivalente,

$$\textit{id\_cliente, número\_préstamo} \rightarrow \textit{prestatario\_préstamo}$$

Las dependencias funcionales se emplean de dos maneras:

1. Para probar las relaciones y ver si son legales de acuerdo con un conjunto dado de dependencias funcionales. Si una relación r es legal según el conjunto F de dependencias funcionales, se dice que r **satisface** F.
2. Para especificar las restricciones del conjunto de relaciones legales. Así, *sólo* habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales. Si uno desea restringirse a las relaciones del esquema R que satisfacen el conjunto F de dependencias funcionales, se dice que F se **cumple** en R.

Considérese la relación r de la Figura 7.5, para ver las dependencias funcionales que se satisfacen. Obsérvese que se satisface $A \rightarrow C$. Hay dos tuplas que tienen un valor para A de a_1. Estas tuplas tienen el mismo valor de C—por ejemplo, c_1. De manera parecida, las dos tuplas con un valor para A de a_2

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Figura 7.5 Relación de ejemplo r.

tienen el mismo valor de C, c_2. No hay más pares de tuplas diferentes que tengan el mismo valor de *A*. Sin embargo, la dependencia funcional $C \rightarrow A$ no se satisface. Para verlo, considérense las tuplas $t_1 = (a_2, b_3, c_2, d_3)$ y $t_2 = (a_3, b_3, c_2, d_4)$. Estas dos tuplas tienen el mismo valor de C, c_2, pero tienen valores diferentes para *A*, a_2 y a_3, respectivamente. Por tanto, se ha hallado un par de tuplas t_1 y t_2 tales que $t_1[C] = t_2[C]$, but $t_1[A] \neq t_2[A]$.

Se dice que algunas dependencias funcionales son **triviales** porque las satisfacen todas las relaciones. Por ejemplo, $A \rightarrow A$ la satisfacen todas las relaciones que implican al atributo *A*. La lectura literal de la definición de dependencia funcional deja ver que, para todas las tuplas t_1 y t_2 tales que $t_1[A] = t_2[A]$, se cumple que $t_1[A] = t_2[A]$. De manera análoga, $AB \rightarrow A$ la satisfacen todas las relaciones que implican al atributo *A*. En general, las dependencias funcionales de la forma $\alpha \rightarrow \beta$ son **triviales** si se cumple la condición $\beta \subseteq \alpha$.

Es importante darse cuenta de que una relación dada puede, en cualquier momento, satisfacer algunas dependencias funcionales cuyo cumplimiento no sea necesario en el esquema de la relación. En la relación *cliente* de la Figura 2.4 puede verse que se satisface *calle_cliente* → *ciudad_cliente*. Sin embargo, se sabe que en el mundo real dos ciudades diferentes pueden tener calles que se llamen igual. Por tanto, es posible, en un momento dado, tener un ejemplar de la relación *cliente* en el que no se satisfaga *calle _cliente* → *ciudad_cliente*. Por consiguiente, no se incluirá *calle_cliente* → *ciudad_cliente* en el conjunto de dependencias funcionales que se cumplen en la relación *cliente*.

Dado un conjunto de dependencias funcionales *F* que se cumple en una relación *r*, es posible que se pueda inferir que también se deban cumplir en esa misma relación otras dependencias funcionales. Por ejemplo, dado el esquema $r = (A, B, C)$, si se cumplen en *r* las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$, se puede inferir que también $A \rightarrow C$ se debe cumplir en *r*. Ya que, dado cualquier valor de *A*, sólo puede haber un valor correspondiente de *B* y, para ese valor de *B*, sólo puede haber un valor correspondiente de C. Más adelante, en el Apartado 7.4.1, se estudia la manera de realizar esas inferencias.

Se utiliza la notación F^+ para denotar el **cierre** del conjunto *F*, es decir, el conjunto de todas las dependencias funcionales que pueden inferirse dado el conjunto *F*. Evidentemente, F^+ es un superconjunto de *F*.

7.3.2 Forma normal de Boyce–Codd

Una de las formas normales más deseables que se pueden obtener es la **forma normal de Boyce-Codd** (**FNBC**). Elimina todas las redundancias que se pueden descubrir a partir de las dependencias funcionales aunque, como se verá en el Apartado 7.6, puede que queden otros tipos de redundancia. El esquema de relación *R* está en la FNBC respecto al conjunto *F* de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, al menos, una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial (es decir, $\beta \subseteq \alpha$).
- α es superclave del esquema *R*.

Los diseños de bases de datos están en la FNBC si cada miembro del conjunto de esquemas de relación que constituye el diseño se halla en la FNBC.

Ya se ha visto que el esquema *prestatario_préstamo* = (*id_cliente, número_préstamo, importe*) no se halla en FNBC. La dependencia funcional *número_préstamo* → *importe* se cumple en *prestatario_préstamo*, pero *número_préstamo* no es superclave (ya que, como se recordará, los préstamos se pueden conceder a consorcios formados por varios clientes). En el Apartado 7.1.2 se vio que la descomposición de *prestatario _préstamo* en *prestatario* y *préstamo* es mejor diseño. El esquema *prestatario* se halla en la FNBC, ya que no se cumple en él ninguna dependencia funcional que no sea trivial. El esquema *préstamo* tiene una dependencia funcional no trivial que se cumple, *número_préstamo* → *importe*, pero *número_préstamo* es superclave (realmente, en este caso, la clave primaria) de *préstamo*. Por tanto, *préstamo* se halla en la FNBC.

Ahora se va a definir una regla general para la descomposición de esquemas que no se hallen en la FNBC. Sea R un esquema que no se halla en la FNBC. En ese caso, queda, al menos, una dependencia funcional no trivial $\alpha \rightarrow \beta$ tal que α no es superclave de R. En el diseño se sustituye R por dos esquemas:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

En el caso anterior de *prestatario_préstamo*, $\alpha =$ *número_préstamo*, $\beta =$ *importe* y *prestatario_préstamo* se sustituye por

- $(\alpha \cup \beta)$ = (*número_préstamo*, *importe*)
- $(R - (\beta - \alpha))$ = (*id_cliente*, *número_préstamo*)

En este caso resulta que $\beta - \alpha = \beta$. Hay que definir la regla tal y como se ha hecho para que trate correctamente las dependencias funcionales que tienen atributos que aparecen a los dos lados de la flecha. Las razones técnicas para esto se tratarán más adelante, en el Apartado 7.5.1.

Cuando se descomponen esquemas que no se hallan en la FNBC, puede que uno o varios de los esquemas resultantes no se hallen en la FNBC. En ese caso, hacen falta más descomposiciones, cuyo resultado final es un conjunto de esquemas FNBC.

7.3.3 FNBC y la conservación de las dependencias

Se han visto varias maneras de expresar las restricciones de consistencia de las bases de datos: restricciones de clave primaria, dependencias funcionales, restricciones **check**, asertos y disparadores. La comprobación de estas restricciones cada vez que se actualiza la base de datos puede ser costosa y, por tanto, resulta útil diseñar la base de datos de manera que las restricciones se puedan comprobar de manera eficiente. En concreto, si la comprobación de las dependencias funcionales puede realizarse considerando sólo una relación, el coste de comprobación de esa restricción será bajo. Se verá que la descomposición en la FNBC puede impedir la comprobación eficiente de determinadas dependencias funcionales.

Para ilustrar esto, supóngase que se lleva a cabo una modificación aparentemente pequeña en el modo en que opera la entidad bancaria. En el diseño de la Figura 6.25 cada cliente sólo puede tener un empleado como "asesor personal". Esto se deduce de que el conjunto de relaciones *asesor* es varios a uno de *cliente* a *empleado*. La "pequeña" modificación que se va a llevar a cabo es que cada cliente pueda tener más de un asesor personal pero, como máximo, uno por sucursal.

Esto se puede implementar en el diagrama E-R haciendo que el conjunto de relaciones *asesor* sea varios a varios (ya que ahora cada cliente puede tener más de un asesor personal) y añadiendo un nuevo conjunto de relaciones, *trabaja_en*, entre *empleado* y *sucursal*, que indique los pares empleado-sucursal, en los que el empleado trabaja en la sucursal. Hay que hacer *trabaja_en* varios a uno de *empleado* a *sucursal*, ya que cada sucursal puede tener varios empleados, pero cada empleado sólo puede trabajar en una sucursal. La Figura 7.6 muestra un subconjunto de la Figura 6.25 con estas adiciones.

No obstante, hay un fallo en este diseño. Permite que cada cliente tenga dos (o más) asesores personales que trabajen en la misma sucursal, algo que el banco no permite. Lo ideal sería que hubiera un solo conjunto de relaciones al que se pudiera hacer referencia para hacer que se cumpla esta restricción. Esto exige que se considere una manera diferente de modificar el diseño E-R. En lugar de añadir el conjunto de relaciones *trabaja_en*, se sustituye el conjunto de relaciones *asesor* por la relación ternaria *sucursal_asesor*, que implica a los conjuntos de entidades *cliente*, *empleado* y *sucursal* y es varios a uno del par *cliente*, *empleado* a *sucursal*, como puede verse en la Figura 7.7. Como este diseño permite que un solo conjunto de relaciones represente la restricción, supone una ventaja significativa respecto del primer enfoque considerado.

Sin embargo, la diferencia entre estos dos enfoques no es tan clara. El esquema derivado de *sucursal_asesor* es

sucursal_asesor = (*id_cliente*, *id_empleado*, *nombre_sucursal*, *tipo*)

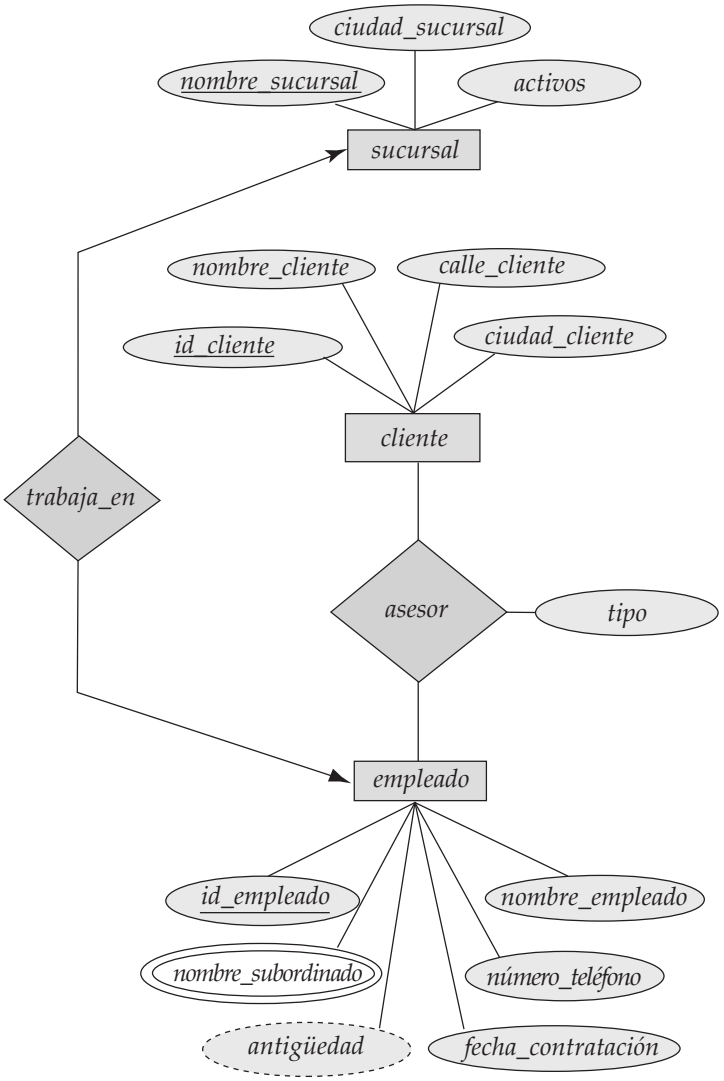

Figura 7.6 Los conjuntos de relaciones *asesor* y *trabaja_en*.

Como cada empleado sólo puede trabajar en una sucursal, en la relación del esquema *sucursal_asesor* sólo puede haber un valor de *nombre_sucursal* asociado con cada valor de *id_empleado*; es decir:

$$id\_empleado \rightarrow nombre\_sucursal$$

No obstante, no queda más remedio que repetir el nombre de la sucursal cada vez que un empleado participa en la relación *sucursal_asesor*. Es evidente que *sucursal_asesor* no se halla en la FNBC, ya que *id_empleado* no es superclave. De acuerdo con la regla para la descomposición en la FNBC, se obtiene:

(*id_cliente*, *id_empleado*, *tipo*)
(*id_empleado*, *nombre_sucursal*)

Este diseño, que es exactamente igual que el primer enfoque que utilizaba el conjunto de relaciones *trabaja_en*, dificulta que se haga cumplir la restricción de que cada cliente pueda tener, como máximo, un asesor en cada sucursal. Esta restricción se puede expresar mediante la dependencia funcional

$$id\_cliente, nombre\_sucursal \rightarrow id\_empleado$$

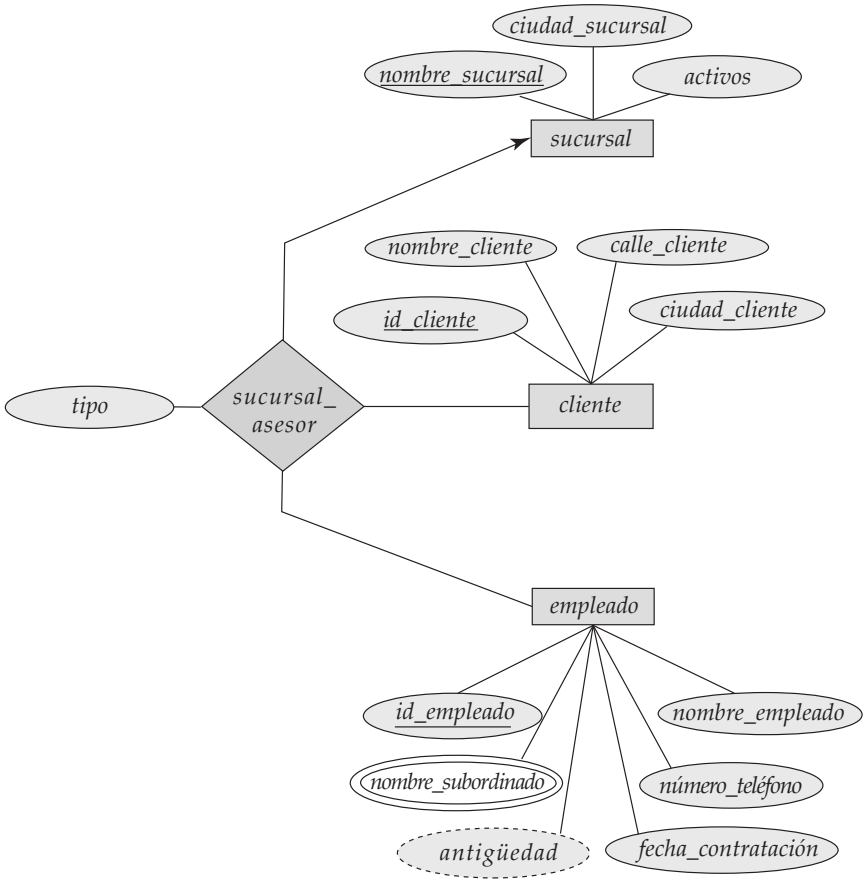

Figura 7.7 El conjunto de relaciones *sucursal_asesor*.

y hacer notar que en el diseño FNBC escogido no hay ningún esquema que incluya todos los atributos que aparecen en esta dependencia funcional. Como el diseño hace computacionalmente difícil que se haga cumplir esta dependencia funcional, se dice que el diseño no **conserva las dependencias**[1]. Como la conservación de las dependencias suele considerarse deseable, se considera otra forma normal, más débil que la FNBC, que permita que se conserven las dependencias. Esa forma normal se denomina tercera forma normal[2].

7.3.4 Tercera forma normal

La FNBC exige que todas las dependencias no triviales sean de la forma $\alpha \rightarrow \beta$, donde α es una superclave. La tercera forma normal (3NF) relaja ligeramente esta restricción al permitir dependencias funcionales no triviales cuya parte izquierda no sea una superclave. Antes de definir la 3NF hay que recordar que las claves candidatas son superclaves mínimas—es decir, superclaves de las que ningún subconjunto propio sea también superclave.

El esquema de relación R está en **tercera forma normal** respecto a un conjunto F de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, al menos, una de las siguientes condiciones:

1. Técnicamente es posible que una dependencia cuyos atributos no aparezcan en su totalidad en ningún esquema se siga haciendo cumplir de manera implícita, debido a la presencia de otras dependencias que la impliquen lógicamente. Este caso se abordará más adelante, en el Apartado 7.4.5.
2. Puede que se haya observado que se ha omitido la segunda forma normal. Sólo tiene significación histórica y, en la práctica, no se utiliza.

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial.
- α es superclave de R.
- Cada atributo A de $\beta - \alpha$ está contenido en alguna clave candidata de R.

Obsérvese que la tercera condición no dice que una sola clave candidata deba contener todos los atributos de $\beta - \alpha$; cada atributo A de $\beta - \alpha$ puede estar contenido en una clave candidata *diferente*.

Las dos primeras alternativas son iguales que las dos alternativas de la definición de la FNBC. La tercera alternativa de la definición de la 3FN parece bastante poco intuitiva, y no resulta evidente el motivo de su utilidad. Representa, en cierto sentido, una relajación mínima de las condiciones de la FNBC que ayuda a garantizar que cada esquema tenga una descomposición que conserve las dependencias en la 3FN. Su finalidad se aclarará más adelante, cuando se estudie la descomposición en la 3FN.

Obsérvese que cualquier esquema que satisfaga la FNBC satisface también la 3FN, ya que cada una de sus dependencias funcionales satisfará una de las dos primeras alternativas. Por tanto, la FNBC es una forma normal más restrictiva que la 3FN.

La definición de la 3FN permite ciertas dependencias funcionales que no se permiten en la FNBC. Las dependencias $\alpha \rightarrow \beta$ que sólo satisfacen la tercera alternativa de la definición de la 3FN no se permiten en la FNBC, pero sí en la 3FN[3].

Considérese nuevamente el ejemplo *sucursal_asesor* y la dependencia funcional

$$id\_empleado \rightarrow nombre\_sucursal$$

que hacía que el esquema no se hallara en FNBC. Obsérvese que, en este caso, $\alpha = id\_empleado$, $\beta = nombre\_sucursal$ y $\beta - \alpha = nombre\_sucursal$. Resulta que *nombre_sucursal* está contenido en una clave candidata y que, por tanto, *sucursal_asesor* se halla en la 3FN. Probar esto, no obstante, exige un pequeño esfuerzo.

Se sabe que, además de las dependencias funcionales

$$id\_empleado \rightarrow nombre\_sucursal$$
$$id\_cliente, nombre\_sucursal \rightarrow id\_empleado$$

que se cumplen, la dependencia funcional

$$id\_cliente, id\_empleado \rightarrow sucursal\_asesor$$

se cumple como consecuencia de que (*id_cliente, id_empleado*) es la clave primaria. Esto hace de (*id_cliente, id_empleado*) clave candidata. Por supuesto, no contiene *nombre_sucursal*, por lo que hay que ver si hay otras claves candidatas. Se concluye que el conjunto de atributos (*id_cliente, nombre_sucursal*) es clave candidata. Veamos el motivo.

Dados unos valores concretos de *id_cliente* y de *nombre_sucursal*, se sabe que sólo hay un valor asociado de *id_empleado*, ya que

$$id\_cliente, nombre\_sucursal \rightarrow id\_empleado$$

Pero entonces, para esos valores concretos de *id_cliente* y de *id_empleado*, sólo puede haber una tupla asociada de *sucursal_asesor*, ya que

$$id\_cliente, id\_empleado \rightarrow sucursal\_asesor$$

Por tanto, se ha argumentado que (*id_cliente, nombre_sucursal*) es superclave. Como resulta que ni *id_cliente* ni *nombre_sucursal* por separado son superclaves, (*id_cliente, nombre_sucursal*) es clave candida-

3. Estas dependencias son ejemplos de **dependencias transitivas** (véase el Ejercicio práctico 7.14). La definición original de la 3NF se hizo en términos de las dependencias transitivas. La definición que aquí se utiliza es equivalente, pero más fácil de comprender.

ta. Dado que esta clave candidata contiene *nombre_sucursal*, la dependencia funcional

$$id\_empleado \rightarrow nombre\_sucursal$$

no viola las reglas de la 3FN.

La demostración de que *sucursal_asesor* se halla en la 3NF ha supuesto cierto esfuerzo. Por esta razón (y por otras), resulta útil adoptar un enfoque estructurado y formal del razonamiento sobre las dependencias funcionales, las formas normales y la descomposición de los esquemas, lo cual se hará en el Apartado 7.4.

Se ha visto el equilibrio que hay que guardar entre la FNBC y la 3NF cuando no hay ningún diseño FNBC que conserve las dependencias. Estos equilibrios se describen con más detalle en el Apartado 7.5.3; en ese apartado también se describe un enfoque de la comprobación de dependencias mediante vistas materializadas que permite obtener las ventajas de la FNBC y de la 3NF.

7.3.5 Formas normales superiores

Puede que en algunos casos el empleo de las dependencias funcionales para la descomposición de los esquemas no sea suficiente para evitar la repetición innecesaria de información. Considérese una ligera variación de la definición del conjunto de entidades *empleado* en la que se permita que los empleados tengan varios números de teléfono, alguno de los cuales puede ser compartido entre varios empleados. Entonces, *número_teléfono* será un atributo multivalorado y, de acuerdo con las reglas para la generación de esquemas a partir de los diseños E-R, habrá dos esquemas, uno por cada uno de los atributos multivalorados *número_teléfono* y *nombre_subordinado*:

(*id_empleado, nombre_subordinado*) (*id_empleado, número_teléfono*)

Si se combinan estos esquemas para obtener

(*id_empleado, nombre_subordinado, número_teléfono*)

se descubre que el resultado se halla en la FNBC, ya que sólo se cumplen dependencias funcionales no triviales. En consecuencia, se puede pensar que ese tipo de combinación es una buena idea. Sin embargo, se trata de una mala idea, como puede verse si se considera el ejemplo de un empleado con dos subordinados y dos números de teléfono. Por ejemplo, sea el empleado con *id_empleado* 999999999 que tiene dos subordinados llamados "David" y "Guillermo" y dos números de teléfono, 512555123 y 512555432. En el esquema combinado hay que repetir los números de teléfono una vez por cada subordinado:

(999999999, David, 512555123)
(999999999, David, 512555432)
(999999999, Guillermo, 512555123)
(999999999, Guillermo, 512555432)

Si no se repitieran los números de teléfono y sólo se almacenaran la primera y la última tupla, se habrían guardado el nombre de los subordinados y los números de teléfono, pero las tuplas resultantes implicarían que David correspondería al 512555123 y que Guillermo correspondería al 512555432. Como sabemos, esto es incorrecto.

Debido a que las formas normales basadas en las dependencias funcionales no son suficientes para tratar con situaciones como ésta, se han definido otras dependencias y formas normales. Se estudian en los Apartados 7.6 y 7.7.

7.4 Teoría de las dependencias funcionales

Se ha visto en los ejemplos que resulta útil poder razonar de manera sistemática sobre las dependencias funcionales como parte del proceso de comprobación de los esquemas para la FNBC o la 3NF.

7.4.1 Cierre de los conjuntos de dependencias funcionales

No es suficiente considerar el conjunto dado de dependencias funcionales. También hay que considerar *todas* las dependencias funcionales que se cumplen. Se verá que, dado un conjunto F de dependencias funcionales, se puede probar que también se cumple alguna otra dependencia funcional. Se dice que F "implica lógicamente" esas dependencias funcionales.

De manera más formal, dado un esquema relacional R, una dependencia funcional f de R está **implicada lógicamente** por un conjunto de dependencias funcionales F de R si cada ejemplar de la relación $r(R)$) que satisface F satisface también f.

Supóngase que se tiene el esquema de relación $R = (A, B, C, G, H, I)$ y el conjunto de dependencias funcionales

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

La dependencia funcional

$$A \rightarrow H$$

está implicada lógicamente. Es decir, se puede demostrar que, siempre que el conjunto dado de dependencias funcionales se cumple en una relación, también se debe cumplir $A \rightarrow H$. Supóngase que t_1 y t_2 son tuplas tales que

$$t_1[A] = t_2[A]$$

Como se tiene que $A \rightarrow B$, se deduce de la definición de dependencia funcional que

$$t_1[B] = t_2[B]$$

Entonces, como se tiene que $B \rightarrow H$, se deduce de la definición de dependencia funcional que

$$t_1[H] = t_2[H]$$

Por tanto, se ha demostrado que, siempre que t_1 y t_2 sean tuplas tales que $t_1[A] = t_2[A]$, debe ocurrir que $t_1[H] = t_2[H]$. Pero ésa es exactamente la definición de $A \rightarrow H$.

Sea F un conjunto de dependencias funcionales. El **cierre** de F, denotado por F^+, es el conjunto de todas las dependencias funcionales implicadas lógicamente por F. Dado F, se puede calcular F^+ directamente a partir de la definición formal de dependencia funcional. Si F fuera de gran tamaño, ese proceso sería largo y difícil. Este tipo de cálculo de F^+ requiere argumentos del tipo que se acaba de utilizar para demostrar que $A \rightarrow H$ está en el cierre del conjunto de dependencias de ejemplo.

Los **axiomas**, o reglas de inferencia, proporcionan una técnica más sencilla para el razonamiento sobre las dependencias funcionales. En las reglas que se ofrecen a continuación se utilizan las letras griegas (α, β, γ, ...) para los conjuntos de atributos y las letras latinas mayúsculas desde el comienzo del alfabeto para los atributos. Se emplea $\alpha\beta$ para denotar $\alpha \cup \beta$.

Se pueden utilizar las tres reglas siguientes para hallar las dependencias funcionales implicadas lógicamente. Aplicando estas reglas *repetidamente* se puede hallar todo F^+, dado F. Este conjunto de reglas se denomina **axiomas de Armstrong** en honor de la persona que las propuso por primera vez.

- **Regla de la reflexividad**. Si α es un conjunto de atributos y $\beta \subseteq \alpha$, entonces se cumple que $\alpha \rightarrow \beta$.
- **Regla de la aumentatividad**. Si se cumple que $\alpha \rightarrow \beta$ y γ es un conjunto de atributos, entonces se cumple que $\gamma\alpha \rightarrow \gamma\beta$.
- **Regla de la transitividad**. Si se cumple que $\alpha \rightarrow \beta$ y que $\beta \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \gamma$.

Los axiomas de Armstrong son **correctos**, ya que no generan dependencias funcionales incorrectas. Son **completos**, ya que, para un conjunto de dependencias funcionales F dado, permiten generar todo

F^+. Las notas bibliográficas proporcionan referencias de las pruebas de su corrección y de su completitud.

Aunque los axiomas de Armstrong son completos, resulta pesado utilizarlos directamente para el cálculo de F^+. Para simplificar más las cosas se dan unas reglas adicionales. Se pueden utilizar los axiomas de Armstrong para probar que son correctas (véanse los Ejercicios prácticos 7.4 y 7.5 y el Ejercicio 7.21).

- **Regla de la unión**. Si se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \beta\gamma$.
- **Regla de la descomposición**. Si se cumple que $\alpha \rightarrow \beta\gamma$, entonces se cumple que $\alpha \rightarrow \beta$ y que $\alpha \rightarrow \gamma$.
- **Regla de la pseudotransitividad**. Si se cumple que $\alpha \rightarrow \beta$ y que $\gamma\beta \rightarrow \delta$, entonces se cumple que $\alpha\gamma \rightarrow \delta$.

Se aplicarán ahora las reglas al ejemplo del esquema $R = (A, B, C, G, H, I)$ y del conjunto F de dependencias funcionales $\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. A continuación se relacionan varios miembros de F^+:

- $A \rightarrow H$. Dado que se cumplen $A \rightarrow B$ y $B \rightarrow H$, se aplica la regla de transitividad. Obsérvese que resultaba mucho más sencillo emplear los axiomas de Armstrong para demostrar que se cumple que $A \rightarrow H$, que deducirlo directamente a partir de las definiciones, como se ha hecho anteriormente en este apartado.
- $CG \rightarrow HI$. Dado que $CG \rightarrow H$ y $CG \rightarrow I$, la regla de la unión implica que $CG \rightarrow HI$.
- $AG \rightarrow I$. Dado que $A \rightarrow C$ y $CG \rightarrow I$, la regla de pseudotransitividad implica que se cumple que $AG \rightarrow I$.

  Otra manera de averiguar que se cumple que $AG \rightarrow I$ es la siguiente. Se utiliza la regla de aumentatividad en $A \rightarrow C$ para inferir que $AG \rightarrow CG$. Aplicando la regla de transitividad a esta dependencia y a $CG \rightarrow I$, se infiere que $AG \rightarrow I$.

La Figura 7.8 muestra un procedimiento que demuestra formalmente el modo de utilizar los axiomas de Armstrong para calcular F^+. En este procedimiento puede que la dependencia funcional ya esté presente en F^+ cuando se le añade y, en ese caso, no hay ninguna modificación en F^+. En el Apartado 7.4.2 se verá una manera alternativa de calcular F^+.

Los términos a la derecha y a la izquierda de las dependencias funcionales son subconjuntos de R. Dado que un conjunto de tamaño n tiene 2^n subconjuntos, hay un total de $2 \times 2^n = 2^{n+1}$ dependencias funcionales posibles, donde n es el número de atributos de R. Cada iteración del bucle repetir del procedimiento, salvo la última, añade como mínimo una dependencia funcional a F^+. Por tanto, está garantizado que el procedimiento termine.

```
F+ = F
repeat
    for each dependencia funcional f de F+
        aplicar las reglas de reflexividad y de aumentatividad a f
        añadir las dependencias funcionales resultantes a F+
    for each par de dependencias funcionales f1 y f2 de F+
        if f1 y f2 se pueden combinar mediante la transitividad
            añadir la dependencia funcional resultante a F+
until F+ deje de cambiar
```

Figura 7.8 Procedimiento para calcular F^+.

7.4.2 Cierre de los conjuntos de atributos

Se dice que un atributo B está **determinado funcionalmente** por α si $\alpha \rightarrow B$. Para comprobar si un conjunto α es superclave hay que diseñar un algoritmo para el cálculo del conjunto de atributos determinados funcionalmente por α. Una manera de hacerlo es calcular F^+, tomar todas las dependencias funcionales con α como término de la izquierda y tomar la unión de los términos de la derecha de todas esas dependencias. Sin embargo, hacer esto puede resultar costoso, ya que F^+ puede ser de gran tamaño.

Un algoritmo eficiente para el cálculo del conjunto de atributos determinados funcionalmente por α no sólo resulta útil para comprobar si α es superclave, sino también para otras tareas, como se verá más adelante en este apartado.

Sea α un conjunto de atributos. Al conjunto de todos los atributos determinados funcionalmente por α bajo un conjunto F de dependencias funcionales se le denomina **cierre** de α bajo F; se denota mediante α^+. La Figura 7.9 muestra un algoritmo, escrito en pseudocódigo, para calcular α^+. La entrada es un conjunto F de dependencias funcionales y el conjunto α de atributos. La salida se almacena en la variable *resultado*.

Para ilustrar el modo en que trabaja el algoritmo se utilizará para calcular $(AG)^+$ con las dependencias funcionales definidas en el Apartado 7.4.1. Se comienza con $result = AG$. La primera vez que se ejecuta el bucle **while** para comprobar cada dependencia funcional se halla que

- $A \rightarrow B$ hace que se incluya B en *resultado*. Para comprobarlo, obsérvese que $A \rightarrow B$ se halla en F y $A \subseteq$ *resultado* (que es AG), por lo que *resultado* := *resultado* $\cup B$.
- $A \rightarrow C$ hace que *resultado* se transforme en $ABCG$.
- $CG \rightarrow H$ hace que *resultado* se transforme en $ABCGH$.
- $CG \rightarrow I$ hace que *resultado* se transforme en $ABCGHI$.

La segunda vez que se ejecuta el bucle **while** ya no se añade ningún atributo nuevo a *resultado*, y se termina el algoritmo.

Veamos ahora el motivo de que el algoritmo de la Figura 7.9 sea correcto. El primer paso es correcto, ya que $\alpha \rightarrow \alpha$ se cumple siempre (por la regla de reflexividad). Se afirma que, para cualquier subconjunto β de *resultado*, $\alpha \rightarrow \beta$. Dado que el bucle **while** se inicia con $\alpha \rightarrow$ *resultado* como cierto, sólo se puede añadir γ a *resultado* si $\beta \subseteq$ *resultado* y $\beta \rightarrow \gamma$. Pero, entonces, *resultado* $\rightarrow \beta$ por la regla de reflexividad, por lo que $\alpha \rightarrow \beta$ por transitividad. Otra aplicación de la transitividad demuestra que $\alpha \rightarrow \gamma$ (empleando $\alpha \rightarrow \beta$ y $\beta \rightarrow \gamma$). La regla de la unión implica que $\alpha \rightarrow$ *resultado* $\cup \gamma$, por lo que α determina funcionalmente cualquier resultado nuevo generado en el bucle **while**. Por tanto, cualquier atributo devuelto por el algoritmo se halla en α^+.

Resulta sencillo ver que el algoritmo halla todo α^+. Si hay un atributo de α^+ que no se halle todavía en *resultado*, debe haber una dependencia funcional $\beta \rightarrow \gamma$ para la que $\beta \subseteq$ *resultado* y, como mínimo, un atributo de γ no se halla en *resultado*.

En el peor de los casos es posible que este algoritmo tarde un tiempo proporcional al cuadrado del tamaño de F. Hay un algoritmo más rápido (aunque ligeramente más complejo) que se ejecuta en un tiempo proporcional al tamaño de F; ese algoritmo se presenta como parte del Ejercicio práctico 7.8.

Existen varias aplicaciones del algoritmo de cierre de atributos:

```
resultado := α;
while (cambios en resultado) do
        for each dependencia funcional β → γ in F do
            begin
                if β ⊆ resultado then resultado := resultado ∪ γ;
            fin
```

Figura 7.9 Algoritmo para el cálculo de α^+, el cierre de α bajo F.

- Para comprobar si α es superclave, se calcula α^+ y se comprueba si contiene todos los atributos de *R*.
- Se puede comprobar si se cumple la dependencia funcional $\alpha \rightarrow \beta$ (o, en otras palabras, si se halla en F^+), comprobando si $\beta \subseteq \alpha^+$. Es decir, se calcula α^+ empleando el cierre de los atributos y luego se comprueba si contiene a β. Esta prueba resulta especialmente útil, como se verá más adelante en este mismo capítulo.
- Ofrece una manera alternativa de calcular F^+: para cada $\gamma \subseteq R$ se halla el cierre γ^+ y, para cada $S \subseteq \gamma^+$, se genera la dependencia funcional $\gamma \rightarrow S$.

7.4.3 Recubrimiento canónico

Supóngase que se tiene un conjunto *F* de dependencias funcionales de un esquema de relación. Siempre que un usuario lleve a cabo una actualización de la relación, el sistema de bases de datos debe asegurarse de que la actualización no viole ninguna dependencia funcional, es decir, que se satisfagan todas las dependencias funcionales de *F* en el nuevo estado de la base de datos.

El sistema debe retroceder la actualización si viola alguna dependencia funcional del conjunto *F*.

Se puede reducir el esfuerzo dedicado a la comprobación de las violaciones comprobando un conjunto simplificado de dependencias funcionales que tenga el mismo cierre que el conjunto dado. Cualquier base de datos que satisfaga el conjunto simplificado de dependencias funcionales satisfará también el conjunto original y viceversa, ya que los dos conjuntos tienen el mismo cierre. Sin embargo, el conjunto simplificado resulta más sencillo de comprobar. En breve se verá el modo en que se puede crear ese conjunto simplificado. Antes, hacen falta algunas definiciones.

Se dice que un atributo de una dependencia funcional es **raro** si se puede eliminar sin modificar el cierre del conjunto de dependencias funcionales. La definición formal de los **atributos raros** es la siguiente. Considérese un conjunto *F* de dependencias funcionales y la dependencia funcional $\alpha \rightarrow \beta$ de *F*.

- El atributo *A* es raro en $alpha$ si $A \in \alpha$ y *F* implica lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
- El atributo *A* es raro en β si $A \in \beta$ y el conjunto de dependencias funcionales $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica lógicamente a *F*.

Por ejemplo, supóngase que se tienen las dependencias funcionales $AB \rightarrow C$ y $A \rightarrow C$ de F. Entonces, B es raro en $AB \rightarrow C$. Como ejemplo adicional, supóngase que se tienen las dependencias funcionales $AB \rightarrow CD$ y $A \rightarrow C$ de F. Entonces, C será raro en el lado derecho de $AB \rightarrow CD$.

Hay que tener cuidado con la dirección de las implicaciones al utilizar la definición de los atributos raros: si se intercambian el lado derecho y el izquierdo, la implicación se cumplirá *siempre*. Es decir, $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$ siempre implica lógicamente a F, y F también implica lógicamente siempre a $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$.

A continuación se muestra el modo de comprobar de manera eficiente si un atributo es raro. Sea *R* el esquema de la relación y F el conjunto dado de dependencias funcionales que se cumplen en R. Considérese el atributo A de la dependencia $\alpha \rightarrow \beta$.

- Si $A \in \beta$, para comprobar si A es raro hay que considerar el conjunto
  $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$
  y comprobar si $\alpha \rightarrow A$ puede inferirse a partir de F'. Para ello hay que calcular α^+ (el cierre de α) bajo F'; si α^+ incluye a A, entonces A es raro en β.
- Si $A \in \alpha$, para comprobar si A es raro, sea $\gamma = \alpha - \{A\}$, hay que comprobar si se puede inferir que $\gamma \rightarrow \beta$ a partir de F. Para ello hay que calcular γ^+ (el cierre de γ) bajo F; si γ^+ incluye todos los atributos de β, entonces A es raro en α.

Por ejemplo, supóngase que F contiene $AB \rightarrow CD$, $A \rightarrow E$ y $E \rightarrow C$. Para comprobar si C es raro en $AB \rightarrow CD$, hay que calcular el cierre de los atributos de AB bajo $F' = \{AB \rightarrow D, A \rightarrow E$ y $E \rightarrow C\}$. El cierre es $ABCDE$, que incluye a CD, por lo que se infiere que C es raro.

$F_c = F$
repeat
  Utilizar la regla de unión para sustituir las dependencias de F_c de la forma
    $\alpha_1 \rightarrow \beta_1$ y $\alpha_1 \rightarrow \beta_2$ con $\alpha_1 \rightarrow \beta_1 \beta_2$.
  Hallar una dependencia funcional $\alpha \rightarrow \beta$ de F_c con un atributo
    raro en α o en β.
    /* Nota: la comprobación de los atributos raros se lleva a cabo empleando F_c, no F */
  Si se halla algún atributo raro, hay que eliminarlo de $\alpha \rightarrow \beta$.
until F_c ya no cambie.

Figura 7.10 Cálculo del recubrimiento canónico.

El **recubrimiento canónico** F_c de F es un conjunto de dependencias tal que F implica lógicamente todas las dependencias de F_c y F_c implica lógicamente todas las dependencias de F. Además, F_c debe tener las propiedades siguientes:

- Ninguna dependencia funcional de F_c contiene atributos raros.
- El lado izquierdo de cada dependencia funcional de F_c es único. Es decir, no hay dos dependencias $\alpha_1 \rightarrow \beta_1$ y $\alpha_2 \rightarrow \beta_2$ de F_c tales que $\alpha_1 = \alpha_2$.

El recubrimiento canónico del conjunto de dependencias funcionales F puede calcularse como se muestra en la Figura 7.10. Es importante destacar que, cuando se comprueba si un atributo es raro, la comprobación utiliza las dependencias del valor actual de F_c, y **no** las dependencias de F. Si una dependencia funcional sólo contiene un atributo en su lado derecho, por ejemplo, $A \rightarrow C$, y se descubre que ese atributo es raro, se obtiene una dependencia funcional con el lado derecho vacío. Hay que eliminar esas dependencias funcionales.

Se puede demostrar que el recubrimiento canónico de F, F_c, tiene el mismo cierre que F; por tanto, comprobar si se satisface F_c es equivalente a comprobar si se satisface F. Sin embargo, F_c es mínimo en un cierto sentido—no contiene atributos raros, y combina las dependencias funcionales con el mismo lado izquierdo. Resulta más económico comprobar F_c que comprobar el propio F.

Considérese el siguiente conjunto F de dependencias funcionales para el esquema (A, B, C):

$$A \rightarrow BC$$
$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

Calcúlese el recubrimiento canónico de F.

- Hay dos dependencias funcionales con el mismo conjunto de atributos a la izquierda de la flecha:

  $$A \rightarrow BC$$
  $$A \rightarrow B$$

  Estas dependencias funcionales se combinan en $A \rightarrow BC$.
- A es raro en $AB \rightarrow C$, ya que F implica lógicamente a $(F - \{AB \rightarrow C\}) \cup \{B \rightarrow C\}$. Esta aseveración es cierta porque $B \rightarrow C$ ya se halla en el conjunto de dependencias funcionales.
- C es raro en $A \rightarrow BC$, ya que $A \rightarrow BC$ está implicada lógicamente por $A \rightarrow B$ y $B \rightarrow C$.

Por tanto, el recubrimiento canónico es

$$A \rightarrow B$$
$$B \rightarrow C$$

Dado un conjunto F de dependencias funcionales, puede suceder que toda una dependencia funcional del conjunto sea rara, en el sentido de que eliminarla no modifique el cierre de F. Se puede demostrar que el recubrimiento canónico F_c de F no contiene esa dependencia funcional rara. Supóngase que, por el contrario, esa dependencia rara estuviera en F_c. Los atributos del lado derecho de la dependencia serían raros, lo que no es posible por la definición de recubrimiento canónico.

Puede que el recubrimiento canónico no sea único. Por ejemplo, considérese el conjunto de dependencias funcionales $F = \{A \rightarrow BC, B \rightarrow AC \text{ y } C \rightarrow AB\}$. Si se aplica la prueba de rareza a $A \rightarrow BC$ se descubre que tanto B como C son raros bajo F. Sin embargo, sería incorrecto eliminar los dos. El algoritmo para hallar el recubrimiento canónico selecciona uno de los dos y lo elimina. Entonces:

1. Si se elimina C, se obtiene el conjunto $F' = \{A \rightarrow B, B \rightarrow AC \text{ y } C \rightarrow AB\}$. Ahora B ya no es raro en el lado derecho de $A \rightarrow B$ bajo F'. Siguiendo con el algoritmo se descubre que A y B son raros en el lado derecho de $C \rightarrow AB$, lo que genera dos recubrimientos canónicos:

$$F_c = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$$
$$F_c = \{A \rightarrow B, B \rightarrow AC, C \rightarrow B\}$$

2. Si se elimina B, se obtiene el conjunto $\{A \rightarrow C, B \rightarrow AC \text{ y } C \rightarrow AB\}$. Este caso es simétrico del anterior y genera dos recubrimientos canónicos:

$$F_c = \{A \rightarrow C, C \rightarrow B, B \rightarrow A\}$$
$$F_c = \{A \rightarrow C, B \rightarrow C, C \rightarrow AB\}$$

Como ejercicio, el lector debe intentar hallar otro recubrimiento canónico de F.

7.4.4 Descomposición sin pérdida

Sean R un esquema de relación y F un conjunto de dependencias funcionales de R. Supóngase que R_1 y R_2 forman una descomposición de R. Sea $r(R)$ una relación con el esquema R. Se dice que la descomposición es una **descomposición sin pérdidas** si, para todos los ejemplares legales de la base de datos (es decir, los ejemplares de la base de datos que satisfacen las dependencias funcionales especificadas y otras restricciones),

$$\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$$

En otros términos, si se proyecta r sobre R_1 y R_2 y se calcula la reunión natural del resultado de la proyección, se vuelve a obtener exactamente r. Las descomposiciones que no son sin pérdidas se denominan **descomposiciones con pérdidas**. Los términos **descomposición de reunión sin pérdidas** y **descomposición de reunión con pérdidas** se utilizan a veces en lugar de descomposición sin pérdidas y descomposición con pérdidas.

Se pueden utilizar las dependencias funcionales para probar si ciertas descomposiciones son sin pérdidas. Sean R, R_1, R_2 y F como se acaban de definir. R_1 y R_2 forman una descomposición sin pérdidas de R si, como mínimo, una de las dependencias funcionales siguientes se halla en F^+:

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

En otros términos, si $R_1 \cap R_2$ forma una superclave de R_1 o de R_2, la descomposición de R es una descomposición sin pérdidas. Se puede utilizar el cierre de los atributos para buscar superclaves de manera eficiente, como ya se ha visto antes.

Para ilustrar esto, considérese el esquema

$$\textit{prestatario\_préstamo} = (\underline{\textit{id\_cliente, número\_préstamo}}, \textit{importe})$$

que se descompuso en el Apartado 7.1.2 en

$$\textit{prestatario} = (\textit{id\_cliente}, \underline{\textit{número\_préstamo}})$$
$$\textit{préstamo} = (\underline{\textit{número\_préstamo}}, \textit{importe})$$

```
calcular F+;
for each esquema Ri in D do
    begin
        Fi : = la restricción de F+ a Ri;
    end
F' := ∅
for each restricción Fi do
    begin
        F' = F' ∪ Fi
    end
calcular F'+;
if (F'+ = F+) then return (cierto)
              else return (falso);
```

Figura 7.11 Comprobación de la conservación de las dependencias.

En este caso *prestatario* $\cap$ *préstamo* = *número_préstamo* y *número_préstamo* $\rightarrow$ *importe*, lo que satisface la regla de la descomposición sin pérdidas.

Para el caso general de la descomposición simultánea de un esquema en varios, la comprobación de la descomposición sin pérdidas es más complicada. Véanse las notas bibliográficas para tener referencias sobre este tema.

Mientras que la comprobación de la descomposición binaria es, claramente, una condición suficiente para la descomposición sin pérdidas, sólo se trata de una condición necesaria si todas las las restricciones son dependencias funcionales. Más adelante se verán otros tipos de restricciones (especialmente, un tipo de restricción denominada dependencia multivalorada, que se estudia en el Apartado 7.6.1), que pueden garantizar que una descomposición sea sin pérdidas aunque no esté presente ninguna dependencia funcional.

7.4.5 Conservación de las dependencias

Resulta más sencillo caracterizar la conservación de las dependencias empleando la teoría de las dependencias funcionales que mediante el enfoque ad hoc que se utilizó en el Apartado 7.3.3.

Sea F un conjunto de dependencias funcionales del esquema R y $R_1, R_2, \ldots, R_n$ una descomposición de R. La **restricción** de F a R_i es el conjunto F_i de todas las dependencias funcionales de F^+ que *sólo* incluyen atributos de R_i. Dado que todas las dependencias funcionales de cada restricción implican a los atributos de un único esquema de relación, es posible comprobar el cumplimiento de esas dependencias verificando sólo una relación.

Obsérvese que la definición de restricción utiliza todas las dependencias de F^+, no sólo las de F. Por ejemplo, supóngase que se tiene $F = \{A \rightarrow B, B \rightarrow C\}$ y que se tiene una descomposición en AC y AB. La restricción de F a AC es, por tanto, $A \rightarrow C$, ya que $A \rightarrow C$ se halla en F^+, aunque no se halle en F.

El conjunto de restricciones $F_1, F_2, \ldots, F_n$ es el conjunto de dependencias que pueden comprobarse de manera eficiente. Ahora cabe preguntarse si es suficiente comprobar sólo las restricciones. Sea $F' = F_1 \cup F_2 \cup \cdots \cup F_n$. F' es un conjunto de dependencias funcionales del esquema R pero, en general $F' \neq F$. Sin embargo, aunque $F' \neq F$, puede ocurrir que $F'^+ = F^+$. Si esto último es cierto, entonces, todas las dependencias de F están implicadas lógicamente por F' y, si se comprueba que se satisface F', se habrá comprobado que se satisface F. Se dice que las descomposiciones que tienen la propiedad $F'^+ = F^+$ son **descomposiciones que conservan las dependencias**.

La Figura 7.11 muestra un algoritmo para la comprobación de la conservación de las dependencias. La entrada es el conjunto $D = \{R_1, R_2, \ldots, R_n\}$ de esquemas de relaciones descompuestas y el conjunto F de dependencias funcionales. Este algoritmo resulta costoso, ya que exige el cálculo de F^+. En lugar de aplicar el algoritmo de la Figura 7.11, se consideran dos alternativas.

En primer lugar, hay que tener en cuenta que si se puede comprobar cada miembro de F en una de las relaciones de la descomposición, la descomposición conserva las dependencias. Se trata de una manera sencilla de demostrar la conservación de las dependencias; no obstante, no funciona siempre. Hay casos en los que, aunque la descomposición conserve las dependencias, hay alguna dependencia de F que no

se puede comprobar para ninguna relación de la descomposición. Por tanto, esta prueba alternativa sólo se puede utilizar como condición suficiente que es fácil de comprobar; si falla, no se puede concluir que la descomposición no conserve las dependencias; en vez de eso, hay que aplicar la prueba general.

A continuación se da una comprobación alternativa de la conservación de las dependencias, que evita tener que calcular F^+. La idea intuitiva subyacente a esta comprobación se explicará tras presentarla. La comprobación aplica el procedimiento siguiente a cada $\alpha \rightarrow \beta$ de F.

resultado = α
while (cambios en *resultado*) **do**
  for each R_i de la descomposición
    t = (*resultado* $\cap$ R_i)$^+$ $\cap$ R_i
    resultado = *resultado* $\cup$ t

En este caso, el cierre de los atributos se halla bajo el conjunto de dependencias funcionales F. Si *resultado* contiene todos los atributos de β, se conserva la dependencia funcional $\alpha \rightarrow \beta$. La descomposición conserva las dependencias si, y sólo si, el procedimiento prueba que se conservan todas las dependencias de F.

Las dos ideas claves subyacentes a la comprobación anterior son las siguientes.

- La primera idea es comprobar cada dependencia funcional $\alpha \rightarrow \beta$ de F para ver si se conserva en F' (donde F' es tal y como se define en la Figura 7.11). Para ello, se calcula el cierre de α bajo F'; la dependencia se conserva exactamente cuando el cierre incluye a β. La descomposición conserva la dependencia si (y sólo si) se comprueba que se conservan todas las dependencias de F.
- La segunda idea es emplear una forma modificada del algoritmo de cierre de atributos para calcular el cierre bajo F', sin llegar a calcular antes F'. Se desea evitar el cálculo de F', ya que resulta bastante costoso. Téngase en cuenta que F' es la unión de las F_i, donde F_i es la restricción de F sobre R_i. El algoritmo calcula el cierre de los atributos de (*resultado* $\cap$ R_i) con respecto a F, intersecta el cierre con R_i y añade el conjunto de atributos resultante a *resultado*; esta secuencia de pasos es equivalente al cálculo del cierre de *resultado* bajo F_i. La repetición de este paso para cada *i* del interior del bucle mientras genera el cierre de *resultado* bajo F'.

  Para comprender el motivo de que este enfoque modificado al cierre de atributos funcione correctamente, hay que tener en cuenta que para cualquier $\gamma \subseteq R_i$, $\gamma \rightarrow \gamma^+$ es una dependencia funcional de F^+, y que $\gamma \rightarrow \gamma^+ \cap R_i$ es una dependencia funcional que se halla en F_i, la restricción de F^+ a R_i. A la inversa, si $\gamma \rightarrow \delta$ estuvieran en F_i, δ sería un subconjunto de $\gamma^+ \cap R_i$.

Esta comprobación tarda un tiempo que es polinómico, en lugar del exponencial necesario para calcular F^+.

7.5 Algoritmos de descomposición

Los esquemas de bases de datos del mundo real son mucho mayores que los ejemplos que caben en las páginas de un libro. Por este motivo, hacen falta algoritmos para la generación de diseños que se hallen en la forma normal adecuada. En este apartado se presentan algoritmos para la FNBC y para la 3NF.

7.5.1 Descomposición en la FNBC

Se puede emplear la definición de la FNBC para comprobar directamente si una relación se halla en esa forma normal. Sin embargo, el cálculo de F^+ puede resultar una tarea tediosa. En primer lugar se van a describir pruebas simplificadas para verificar si una relación dada se halla en la FNBC. En caso de que no lo esté, se puede descomponer para crear relaciones que sí estén en la FNBC. Más avanzado este apartado se describirá un algoritmo para crear descomposiciones sin pérdidas de las relaciones, de modo que esas descomposiciones se hallen en la FNBC.

7.5.1.1 Comprobación de la FNBC

La comprobación de relaciones para ver si satisfacen la FNBC se puede simplificar en algunos casos:

- Para comprobar si la dependencia no trivial $\alpha \rightarrow \beta$ provoca alguna violación de la FNBC hay que calcular α^+ (el cierre de los atributos de α) y comprobar si incluye todos los atributos de R; es decir, si es superclave de R.
- Para comprobar si el esquema de relación R se halla en la FNBC basta con comprobar sólo si las dependencias del conjunto F dado violan la FNBC, en vez de comprobar todas las dependencias de F^+.

  Se puede probar que, si ninguna de las dependencias de F provoca violaciones de la FNBC, ninguna de las dependencias de F^+ lo hace tampoco.

Por desgracia, el último procedimiento no funciona cuando se descomponen relaciones. Es decir, en las descomposiciones *no* basta con emplear F cuando se comprueba si la relación R_i de R viola la FNBC. Por ejemplo, considérese el esquema de relación $R\ (A, B, C, D, E)$, con las dependencias funcionales F que contienen $A \rightarrow B$ and $BC \rightarrow D$. Supóngase que se descompusiera en $R_1(A, B)$ y en $R_2(A, C, D, E)$. Ahora ninguna de las dependencias de F contiene únicamente atributos de (A, C, D, E), por lo que se podría creer erróneamente que R_2 satisface la FNBC. De hecho, hay una dependencia $AC \rightarrow D$ de F^+ (que se puede inferir empleando la regla de la pseudotransitividad con las dos dependencias de F), que prueba que R_2 no se halla en la FNBC. Por tanto, puede que haga falta una dependencia que esté en F^+, pero no en F, para probar que una relación descompuesta no se halla en la FNBC.

Una comprobación alternativa de la FNBC resulta a veces más sencilla que calcular todas las dependencias de F^+. Para comprobar si una relación R_i de una descomposición de R se halla en la FNBC, se aplica esta comprobación:

- Para cada subconjunto α de atributos de R_i se comprueba que α^+ (el cierre de los atributos de α bajo F) no incluye ningún atributo de $R_i - \alpha$ o bien incluye todos los atributos de R_i.

Si algún conjunto de atributos α de R_i viola esta condición, considérese la siguiente dependencia funcional, que se puede probar que se halla en F^+:

$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$.

La dependencia anterior muestra que R_i viola la FNBC.

7.5.1.2 Algoritmo de descomposición de la FNBC

Ahora se puede exponer un método general para descomponer los esquemas de relación de manera que satisfagan la FNBC. La Figura 7.12 muestra un algoritmo para esta tarea. Si R no está en la FNBC, se puede descomponer en un conjunto de esquemas en la FNBC, $R_1, R_2, \ldots, R_n$ utilizando este algoritmo. El algoritmo utiliza las dependencias que demuestran la violación de la FNBC para llevar a cabo la descomposición.

```
resultado := {R};
hecho := falso;
calcular F+;
while (not hecho) do
    if (hay algún esquema Ri de resultado que no se halle en la FNBC)
      then begin
              sea α → β una dependencia funcional no trivial que se cumple
              en Ri tal que α → Ri no se halla en F+, y α ∩ β = ∅;
              resultado := (result − Ri) ∪ (Ri − β) ∪ ( α, β);
           end
      else hecho := cierto;
```

Figura 7.12 Algoritmo de descomposición en la FNBC.

La descomposición que genera este algoritmo no sólo está en la FNBC, sino que también es una descomposición sin pérdidas. Para ver el motivo de que el algoritmo sólo genere descomposiciones sin pérdidas hay que darse cuenta de que, cuando se sustituye el esquema R_i por $((R_i - \beta)$ y (α, β), se cumple que $\alpha \rightarrow \beta$ y que $(R_i - \beta) \cap (\alpha, \beta) = \alpha$.

Si no se exigiera que $\alpha \cap \beta = \emptyset$, los atributos de $\alpha \cap \beta$ no aparecerían en el esquema $((R_i - \beta)$, y ya no se cumpliria la dependencia $\alpha \rightarrow \beta$.

Es fácil ver que la descomposición de *prestatario_préstamo* del Apartado 7.3.2 se obtiene de la aplicación del algoritmo. La dependencia funcional *número_préstamo* → *importe* satisface la condición $\alpha \cap \beta = \emptyset$ y, por tanto, se elige para descomponer el esquema.

El algoritmo de descomposición en la FNBC tarda un tiempo exponencial en relación con el tamaño del esquema inicial, ya que el algoritmo para comprobar si las relaciones de la descomposición satisfacen la FNBC puede tardar un tiempo exponencial. Las notas bibliográficas ofrecen referencias de un algoritmo que puede calcular la descomposición en la FNBC en un tiempo polinómico. Sin embargo, puede que ese algoritmo "sobrenormalice", es decir, descomponga relaciones sin que sea necesario.

A modo de ejemplo, más prolijo, del empleo del algoritmo de descomposición en la FNBC, supóngase que se tiene un diseño de base de datos que emplea el siguiente esquema *empréstito*:

empréstito = (*nombre_sucursal, ciudad_sucursal, activos, nombre_cliente, número_préstamo, importe*)

El conjunto de dependencias funcionales que se exige que se cumplan en *empréstito* es

nombre_sucursal → *activos ciudad_sucursal*
número_préstamo → *importe nombre_sucursal*

Una clave candidata para este esquema es {*número_préstamo, nombre_cliente*}.

Se puede aplicar el algoritmo de la Figura 7.12 al ejemplo *empréstito* de la manera siguiente:

- La dependencia funcional

  nombre_sucursal → *activos ciudad_sucursal*

  se cumple, pero *nombre_sucursal* no es superclave. Por tanto, *empréstito* no se halla en la FNBC. Se sustituye *empréstito* por

  sucursal = (*nombre_sucursal, ciudad_sucursal, activos*)
  info_préstamo = (*nombre_sucursal, nombre_cliente, número_préstamo, importe*)

- Entre las pocas dependencias funcionales no triviales que se cumplen en *sucursal* está *nombre_sucursal*, al lado izquierdo de la flecha. Dado que *nombre_sucursal* es clave de *sucursal*, la relación *sucursal* se halla en la FNBC.

- La dependencia funcional

  número_préstamo → *importe nombre_sucursal*

  se cumple en *info_préstamo*, pero *número_préstamo* no es clave de *info_préstamo*. Se sustituye *info_préstamo* por

  sucursal_préstamo = (*número_préstamo, nombre_sucursal, importe*)
  prestatario = (*nombre_cliente, número_préstamo*)

- *sucursal_préstamo* y *prestatario* están en la FNBC.

Por tanto, la descomposición de *empréstito* da lugar a los tres esquemas de relación *sucursal*, *sucursal_préstamo* y *prestatario*, cada uno de los cuales se halla en la FNBC. Se puede comprobar que la descomposición es sin pérdidas y que conserva las dependencias.

```
sea F_c un recubrimiento canónico de F;
i := 0;
for each dependencia funcional α → β de F_c do
   if ninguno de los esquemas R_j, j = 1, 2, ..., i contiene α β
      then begin
              i := i + 1;
              R_i := α β;
           end
if ninguno de los esquemas R_j, j = 1, 2, ..., i contiene una clave candidata de R
   then begin
           i := i + 1;
           R_i := cualquier clave candidata de R;
      end
return (R_1, R_2, ..., R_i)
```

Figura 7.13 Descomposición en la 3NF sin pérdidas que conserva las dependencias.

Téngase en cuenta que, aunque el esquema *sucursal_préstamo* anterior se halle en la FNBC, se puede decidir descomponerlo aún más empleando la dependencia funcional *número_préstamo* → *importe* para obtener los esquemas

préstamo = (*número_préstamo*, *importe*) *sucursal_préstamo* = (*número_préstamo*, *nombre_sucursal*)

Estos esquemas corresponden a los que se han utilizado en este capítulo.

7.5.2 Descomposición en la 3FN

La Figura 7.13 muestra un algoritmo para la búsqueda de descomposiciones en la 3FN sin pérdidas y que conserven las dependencias. El conjunto de dependencias F_c utilizado en el algoritmo es un recubrimiento canónico de F. Obsérvese que el algoritmo considera el conjunto de esquemas $R_j, j = 1, 2, \ldots, i$; inicialmente $i = 0$ y, en ese caso, el conjunto está vacío.

Se aplicará este algoritmo al ejemplo del Apartado 7.3.3 en el que se demostró que

sucursal_asesor = (*id_cliente*, *id_empleado*, *nombre_sucursal*, *tipo*)

se halla en la 3NF, aunque no se halle en la FNBC. El algoritmo utiliza las dependencias funcionales de F, que, en este caso, también son F_c:

(*id_cliente*, *id_empleado* → *nombre_sucursal*, *tipo*)
id_empleado → *nombre_sucursal*

y considera dos esquemas en el bucle **for**. A partir de la primera dependencia funcional el algoritmo genera como R_1 el esquema (*id_cliente, id_empleado, nombre_sucursal, tipo*). A partir de la segunda, el algoritmo genera el esquema (*id_empleado, nombre_sucursal*), pero no lo crea como R_2, ya que se halla incluido en R_1 y, por tanto, falla la condición **if**.

El algoritmo garantiza la conservación de las dependencias mediante la creación explícita de un esquema para cada dependencia del recubrimiento canónico. Asegura que la descomposición sea sin pérdidas al garantizar que, como mínimo, un esquema contenga una clave candidata del esquema que se está descomponiendo. El Ejercicio práctico 7.12 proporciona algunos indicios de la prueba de que esto basta para garantizar una descomposición sin pérdidas.

Este algoritmo también se denomina **algoritmo de síntesis de la 3NF**, ya que toma un conjunto de dependencias y añade los esquemas de uno en uno, en lugar de descomponer el esquema inicial de manera repetida. El resultado no queda definido de manera única, ya que cada conjunto de dependencias funcionales puede tener más de un recubrimiento canónico y, además, en algunos casos, el resultado del algoritmo depende del orden en que considere las dependencias de F_c.

Si una relación R_i está en la descomposición generada por el algoritmo de síntesis, entonces R_i está en la 3FN. Recuérdese que, cuando se busca la 3FN, basta con considerar las dependencias funcionales cuyo lado derecho sea un solo atributo. Por tanto, para ver si R_i está en la 3FN, hay que convencerse de que cualquier dependencia funcional $\gamma \rightarrow B$ que se cumpla en R_i satisface la definición de la 3FN.

Supóngase que la dependencia que generó R_i en el algoritmo de síntesis es $\alpha \rightarrow \beta$. Ahora bien, B debe estar en α o en β, ya que B está en R_i y $\alpha \rightarrow \beta$ ha generado R_i. Considérense los tres casos posibles:

- B está tanto en α como en β. En ese caso, la dependencia $\alpha \rightarrow \beta$ no habría estado en F_c, ya que B sería rara en β. Por tanto, este caso no puede darse.
- B está en β pero no en α. Considérense dos casos:
  - γ es superclave. Se satisface la segunda condición de la 3FN.
  - γ no es superclave. Entonces, α debe contener algún atributo que no se halle en γ. Ahora bien, como $\gamma \rightarrow B$ se halla en F^+, debe poder obtenerse a partir de F_c mediante el algoritmo del cierre de atributos de γ. La obtención no puede haber empleado $\alpha \rightarrow \beta$—si lo hubiera hecho, α debería estar contenida en el cierre de los atributos de γ, lo que no es posible, ya que se ha dado por supuesto que γ no es superclave. Ahora bien, empleando $\alpha \rightarrow (\beta - \{B\})$ y $\gamma \rightarrow B$, se puede obtener que $\alpha \rightarrow B$ (debido a que $\gamma \subseteq \alpha\beta$ y a que γ no puede contener a B porque $\gamma \rightarrow B$ no es trivial). Esto implicaría que B es raro en el lado derecho de $\alpha \rightarrow \beta$, lo que no es posible, ya que $\alpha \rightarrow \beta$ está en el recubrimiento canónico F_c. Por tanto, si B está en β, γ debe ser superclave, y se debe satisfacer la segunda condición de la 3FN.
- B está en α pero no en β.
  Como α es clave candidata, se satisface la tercera alternativa de la definición de la 3FN.

Es interesante que el algoritmo que se ha descrito para la descomposición en la 3FN pueda implementarse en tiempo polinómico, aunque la comprobación de una relación dada para ver si satisface la 3FN sea NP-dura (lo que significa que es muy improbable que se invente nunca un algoritmo de tiempo polinómico para esta tarea).

7.5.3 Comparación de la FNBC y la 3FN

De las dos formas normales para los esquemas de las bases de datos relacionales, la 3FN y la FNBC, la 3FN es más conveniente, ya que se sabe que siempre es posible obtener un diseño en la 3FN sin sacrificar la ausencia de pérdidas ni la conservación de las dependencias. Sin embargo, la 3FN presenta inconvenientes: puede que haya que emplear valores nulos para representar algunas de las relaciones significativas posibles entre los datos y existe el problema de la repetición de la información.

Los objetivos del diseño de bases de datos con dependencias funcionales son:

1. FNBC
2. Ausencia de pérdidas
3. Conservación de las dependencias

Como no siempre resulta posible satisfacer las tres, puede que nos veamos obligados a escoger entre la FNBC y la conservación de las dependencias con la 3FN.

Merece la pena destacar que el SQL no ofrece una manera de especificar las dependencias funcionales, salvo para el caso especial de la declaración de las superclaves mediante las restricciones **primary key** o **unique**. Es posible, aunque un poco complicado, escribir asertos que hagan que se cumpla una dependencia funcional determinada (véase el Ejercicio práctico 7.9); por desgracia, la comprobación de los asertos resultaría muy costosa en la mayor parte de los sistemas de bases de datos. Por tanto, aunque se tenga una descomposición que conserve las dependencias, si se utiliza el SQL estándar, sólo se podrán comprobar de manera eficiente las dependencias funcionales cuyo lado izquierdo sea una clave.

Aunque puede que la comprobación de las dependencias funcionales implique una reunión si la descomposición no conserva las dependencias, se puede reducir su coste empleando vistas materializadas, que se pueden utilizar en la mayor parte de los sistemas de bases de datos. Dada una descomposición en la FNBC que no conserve las dependencias, se considera cada dependencia de un recubrimiento mínimo F_c que no se conserva en la descomposición. Para cada una de esas dependencias $\alpha \rightarrow \beta$, se define una vista materializada que calcula una reunión de todas las relaciones de la descomposición y proyecta el resultado sobre $\alpha\beta$. La dependencia funcional puede comprobarse fácilmente en la vista materializada

mediante una restricción **unique**(α). La parte negativa es que hay una sobrecarga espacial y temporal debida a la vista materializada, pero la positiva es que el programador de la aplicación no tiene que preocuparse de escribir código para hacer que los datos redundantes se conserven consistentes en las actualizaciones; es labor del sistema de bases de datos conservar la vista materializada, es decir, mantenerla actualizada cuando se actualice la base de datos (más adelante, en el Apartado 14.5, se describe el modo en que el sistema de bases de datos puede llevar a cabo de manera eficiente el mantenimiento de las vistas materializadas).

Por tanto, en caso de que no se pueda obtener una descomposición en la FNBC que conserve las dependencias, suele resultar preferible optar por la 3NF y emplear técnicas como las vistas materializadas para reducir el coste de la comprobación de las dependencias funcionales.

7.6 Descomposición mediante dependencias multivaloradas

No parece que algunos esquemas de relación, aunque se hallen en la FNBC, estén suficientemente normalizados, en el sentido de que siguen sufriendo el problema de la repetición de información. Considérese nuevamente el ejemplo bancario. Supóngase que, en un diseño alternativo del esquema de la base de datos bancaria, se tiene el esquema

$$\textit{préstamo\_cliente} = (\underline{\textit{número\_préstamo, id\_cliente}}, \textit{nombre\_cliente, calle\_cliente, ciudad\_cliente})$$

El lector avispado reconocerá este esquema como no correspondiente a la FNBC, debido a la dependencia funcional

$$\textit{id\_cliente} \rightarrow \textit{nombre\_cliente, calle\_cliente ciudad\_cliente}$$

y a que *id_cliente* no es clave de *préstamo_cliente*. No obstante, supóngase que el banco está atrayendo clientes ricos que tienen varios domicilios (por ejemplo, una residencia de invierno y otra de verano). Entonces, ya no se desea que se cumpla la dependencia funcional *id_cliente* →*calle_cliente ciudad_cliente*, aunque, por supuesto, se sigue deseando que se cumpla *id_cliente* → *nombre_cliente* (es decir, el banco no trata con clientes que operan con varios alias). A partir del algoritmo de descomposición en la FNBC se obtienen dos esquemas:

$$R_1 = (\underline{\textit{id\_cliente}}, \textit{nombre\_cliente})$$
$$R_2 = (\underline{\textit{número\_préstamo, id\_cliente}}, \textit{calle\_cliente, ciudad\_cliente})$$

Los dos se encuentran en la FNBC (recuérdese que no sólo puede cada cliente tener concedido más de un préstamo, sino que también se pueden conceder préstamos a grupos de personas y, por tanto, no se cumplen ni *id_cliente* → *número_préstamo* ni *número_préstamo* → *id_cliente*).

Pese a que R_2 se halla en la FNBC, hay redundancia. Se repite la dirección de cada residencia de cada cliente una vez por cada préstamo que tenga concedido ese cliente. Este problema se puede resolver descomponiendo más aún R_2 en:

id_cliente_préstamo = (*número_préstamo, id_cliente*)
residencia_cliente = (*id_cliente, calle_cliente, ciudad_cliente*)

pero no hay ninguna restricción que nos lleve a hacerlo.

Para tratar este problema hay que definir una nueva modalidad de restricción, denominada *dependencia multivalorada*. Al igual que se hizo con las dependencias funcionales, se utilizarán las dependencias multivaloradas para definir una forma normal para los esquemas de relación. Esta forma normal, denominada **cuarta forma normal** (4FN), es más restrictiva que la FNBC. Se verá que cada esquema en la 4FN también se halla en la FNBC, pero que hay esquemas en la FNBC que no se hallan en la 4FN.

7.6.1 Dependencias multivaloradas

Las dependencias funcionales impiden que ciertas tuplas estén en una relación dada. Si $A \rightarrow B$, entonces no puede haber dos tuplas con el mismo valor de A y diferentes valores de B. Las dependencias

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

Figura 7.14 Representación tabular de $\alpha \twoheadrightarrow \beta$.

multivaloradas, por otro lado, no impiden la existencia de esas tuplas. Por el contrario, *exigen* que estén presentes en la relación otras tuplas de una forma determinada. Por este motivo, las dependencias funcionales se denominan a veces **dependencias que generan igualdades**; y las dependencias multivaloradas, **dependencias!que generan tuplas**.

Sea R un esquema de relación y sean $\alpha \subseteq R$ y $\beta \subseteq R$. La **dependencia multivalorada**

$$\alpha \twoheadrightarrow \beta$$

se cumple en R si, en cualquier relación legal $r(R)$ para todo par de tuplas t_1 y t_2 de r tales que $t_1[\alpha] = t_2[\alpha]$ existen unas tuplas t_3 y t_4 de r tales que

$$\begin{aligned} &t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\ &t_3[\beta] = t_1[\beta] \\ &t_3[R - \beta] = t_2[R - \beta] \\ &t_4[\beta] = t_2[\beta] \\ &t_4[R - \beta] = t_1[R - \beta] \end{aligned}$$

Esta definición es menos complicada de lo que parece. La Figura 7.14 muestra una representación tabular de t_1, t_2, t_3 y t_4. De manera intuitiva, la dependencia multivalorada $\alpha \twoheadrightarrow \beta$ indica que la relación entre α y β es independiente de la relación entre α y $R - \beta$. Si todas las relaciones del esquema R satisfacen la dependencia multivalorada $\alpha \twoheadrightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$ es una dependencia multivalorada *trivial* del esquema R. Por tanto, $\alpha \twoheadrightarrow \beta$ es trivial si $\beta \subseteq \alpha$ o $\beta \cup \alpha = R$.

Para ilustrar la diferencia entre las dependencias funcionales y las multivaloradas, considérense de nuevo el esquema R_2 y la relación de ejemplo de ese esquema mostrada en la Figura 7.15. Hay que repetir el número de préstamo una vez por cada dirección que tenga el cliente, y la dirección en cada préstamo que tenga concedido ese cliente. Esta repetición es innecesaria, ya que la relación entre cada cliente y su dirección es independiente de la relación entre ese cliente y el préstamo. Si un cliente con identificador de cliente 99123 tiene concedido un préstamo (por ejemplo, el préstamo número P-23), se desea que ese préstamo esté asociado con todas las direcciones de ese cliente. Por tanto, la relación de la Figura 7.16 es ilegal. Para hacer que esa relación sea legal hay que añadir a la relación de la Figura 7.16 las tuplas (P-23, 99123, Mayor, Chinchón) y (P-27, 99123, Carretas, Cerceda).

Si se compara el ejemplo anterior con la definición de dependencia multivalorada, se ve que se desea que se cumpla la dependencia multivalorada

id_cliente $\twoheadrightarrow$ *calle_cliente ciudad_cliente*

También se cumplirá la dependencia multivalorada *id_cliente* $\twoheadrightarrow$ *número_préstamo*. Pronto se verá que son equivalentes.

Al igual que las dependencias funcionales, las dependencias multivaloradas se utilizan de dos maneras:

número_préstamo	*id_cliente*	*calle_cliente*	*ciudad_cliente*
P-23	99123	Carretas	Cerceda
P-23	99123	Mayor	Chinchón
P-93	15106	Leganitos	Aluche

Figura 7.15 Ejemplo de redundancia en un esquema en la FNBC.

número_préstamo	*id_cliente*	*calle_cliente*	*ciudad_cliente*
P-23	99123	Carretas	Cerceda
P-27	99123	Mayor	Chinchón

Figura 7.16 La relación ilegal R_2.

1. Para verificar las relaciones y determinar si son legales bajo un conjunto dado de dependencias funcionales y multivaloradas.
2. Para especificar restricciones del conjunto de relaciones legales; de este modo, *sólo* habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales y multivaloradas.

Téngase en cuenta que, si una relación *r* no satisface una dependencia multivalorada dada, se puede crear una relación *r'* que *sí* la satisfaga añadiendo tuplas a *r*.

Supóngase que *D* denota un conjunto de dependencias funcionales y multivaloradas. El **cierre** D^+ de *D* es el conjunto de todas las dependencias funcionales y multivaloradas implicadas lógicamente por *D*. Al igual que se hizo con las dependencias funcionales, se puede calcular D^+ a partir de *D*, empleando las definiciones formales de dependencia funcional y multivalorada. Con este razonamiento se puede trabajar con dependencias multivaloradas muy sencillas. Afortunadamente, parece que las dependencias multivaloradas que se dan en la práctica son bastante sencillas. Para dependencias complejas es mejor razonar con conjuntos de dependencias mediante un sistema de reglas de inferencia (el Apartado C.1.1 del apéndice describe un sistema de reglas de inferencia para las dependencias multivaloradas).

A partir de la definición de dependencia multivalorada se puede obtener la regla siguiente:

- Si $\alpha \rightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$.

En otras palabras, todas las dependencias funcionales son también dependencias multivaloradas.

7.6.2 Cuarta forma normal

Considérese nuevamente el ejemplo del esquema en la FNBC

$$R_2 = (\underline{\textit{número\_préstamo, id\_cliente}}, \textit{calle\_cliente, ciudad\_cliente})$$

en el que se cumple la dependencia multivalorada *id_cliente* $\twoheadrightarrow$ *calle_cliente ciudad_cliente*. Se vio en los primeros párrafos del Apartado 7.6 que, aunque este esquema se halla en la FNBC, el diseño no es el ideal, ya que hay que repetir la información sobre la dirección del cliente para cada préstamo. Se verá que se puede utilizar esta dependencia multivalorada para mejorar el diseño de la base de datos, realizando la descomposición del esquema en la **cuarta forma normal**.

Un esquema de relación R está en la **cuarta forma normal** (4FN) con respecto a un conjunto D de dependencias funcionales y multivaloradas si, para todas las dependencias multivaloradas de D^+ de la forma $\alpha \twoheadrightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, como mínimo, una de las condiciones siguientes

- $\alpha \twoheadrightarrow \beta$ es una dependencia multivalorada trivial.
- α es superclave del esquema R.

El diseño de una base de datos está en la 4FN si cada componente del conjunto de esquemas de relación que constituye el diseño se halla en la 4FN.

Téngase en cuenta que la definición de la 4FN sólo se diferencia de la definición de la FNBC en el empleo de las dependencias multivaloradas en lugar de las dependencias funcionales. Todos los esquemas en la 4FN están en la FNBC. Para verlo hay que darse cuenta de que, si un esquema R no se halla en la FNBC, hay una dependencia funcional no trivial $\alpha \rightarrow \beta$ que se cumple en R en la que α no es superclave. Como $\alpha \rightarrow \beta$ implica $\alpha \twoheadrightarrow \beta$, R no puede estar en la 4FN.

```
resultado := {R};
hecho := falso;
calcular D+; dado el esquema Ri, Di denotará la restricción de D+ a Ri
while (not hecho) do
    if (hay un esquema Ri en resultado que no se halla en la 4FN con respecto a Di)
      then begin
              sea α ↠ β una dependencia multivalorada no trivial que se cumple
              en Ri tal que α → Ri no se halla en Di, y α ∩ β = ∅;
              resultado := (resultado − Ri) ∪ (Ri − β) ∪ (α, β);
           end
      else hecho := cierto;
```

Figura 7.17 Algoritmo de descomposición en la 4FN.

Sea R un esquema de relación y sean $R_1, R_2, \ldots, R_n$ una descomposición de R. Para comprobar si cada esquema de relación R_i de la descomposición se halla en la 4FN hay que averiguar las dependencias multivaloradas que se cumplen en cada R_i. Recuérdese que, para un conjunto F de dependencias funcionales, la restricción F_i de F a R_i son todas las dependencias funcionales de F^+ que *sólo* incluyen los atributos de R_i. Considérese ahora un conjunto D de dependencias funcionales y multivaloradas. La **restricción** de D a R_i es el conjunto D_i consistente en

1. Todas las dependencias funcionales de D^+ que sólo incluyen atributos de R_i
2. Todas las dependencias multivaloradas de la forma

$$\alpha \twoheadrightarrow \beta \cap R_i$$

donde $\alpha \subseteq R_i \text{y} \alpha \twoheadrightarrow \beta$ está en D^+.

7.6.3 Descomposición en la 4FN

La analogía entre la 4FN y la FNBC es aplicable al algoritmo para descomponer esquemas en la 4FN. La Figura 7.17 muestra el algoritmo de descomposición en la 4FN. Es idéntico al algoritmo de descomposición en la FNBC de la Figura 7.12, salvo en que emplea dependencias multivaloradas en lugar de funcionales y en que utiliza la restricción de D^+ a R_i.

Si se aplica el algoritmo de la Figura 7.17 a (*número_préstamo, id_cliente, calle_cliente, ciudad_cliente*), se descubre que *id_cliente* ↠ *número_préstamo* es una dependencia multivalorada no trivial, y que *id_cliente* no es superclave del esquema. De acuerdo con el algoritmo, se sustituye el esquema original por estos dos esquemas:

id_cliente_préstamo = (*número_préstamo*, *id_cliente*)
residencia_cliente = (*id_cliente*, *calle_cliente*, *ciudad_cliente*)

Este par de esquemas, que se hallan en la 4FN, elimina la redundancia que se encontró anteriormente.

Como ocurría cuando se trataba solamente con las dependencias funcionales, resultan interesantes las descomposiciones que carecen de pérdidas y que conservan las dependencias. La siguiente circunstancia relativa a las dependencias multivaloradas y a la ausencia de pérdidas muestra que el algoritmo de la Figura 7.17 sólo genera descomposiciones sin pérdidas:

- Sean R un esquema de relación y D un conjunto de dependencias funcionales y multivaloradas de R. Supóngase que R_1 y R_2 forman una descomposición de R. Esa descomposición de R carecerá de pérdidas si, y sólo si, como mínimo, una de las siguientes dependencias multivaloradas se halla en D^+:

$$R_1 \cap R_2 \twoheadrightarrow R_1$$
$$R_1 \cap R_2 \twoheadrightarrow R_2$$

Recuérdese que se afirmó en el Apartado 7.4.4 que, si $R_1 \cap R_2 \rightarrow R_1$ o $R_1 \cap R_2 \rightarrow R_2$, entonces R_1 y R_2 son descomposiciones de R sin pérdidas. Esta circunstancia relativa a las dependencias multivaloradas es una declaración más general sobre la carencia de pérdidas. Afirma que, para *toda* descomposición de R sin pérdidas en dos esquemas R_1 y R_2, debe cumplirse una de las dos dependencias $R_1 \cap R_2 \twoheadrightarrow R_1$ o $R_1 \cap R_2 \twoheadrightarrow R_2$.

El problema de la conservación de las dependencias al descomponer relaciones se vuelve más complejo en presencia de las dependencias multivaloradas. El Apartado C.1.2 del apéndice aborda este asunto.

7.7 Más formas normales

La cuarta forma normal no es, de ningún modo, la forma normal "definitiva". Como ya se ha visto, las dependencias multivaloradas ayudan a comprender y a abordar algunas formas de repetición de la información que no pueden comprenderse en términos de las dependencias funcionales. Hay restricciones denominadas **dependencias de reunión** que generalizan las dependencias multivaloradas y llevan a otra forma normal denominada **forma normal de reunión por proyección (FNRP)** (la FNRP se denomina en algunos libros **quinta forma normal**). Hay una clase de restricciones todavía más generales, que lleva a una forma normal denominada **forma normal de dominios y claves** (FNDC).

Un problema práctico del empleo de estas restricciones generalizadas es que no sólo es difícil razonar con ellas, sino que tampoco hay un conjunto de reglas de inferencia seguras y completas para razonar sobre las restricciones. Por tanto, FNRP y FNDC se utilizan muy rara vez. El Apéndice C ofrece más detalles sobre estas formas normales.

Destaca por su ausencia en este estudio de las formas normales la **segunda forma normal** (2FN). No se ha estudiado porque sólo es de interés histórico. Simplemente se definirá, y se permitirá al lector experimentar con ella, en el Ejercicio práctico 7.15.

7.8 Proceso de diseño de las bases de datos

Hasta ahora se han examinado aspectos detallados de las formas normales y de la normalización. En este apartado se estudiará el modo de encajar la normalización en el proceso global de diseño de las bases de datos.

Anteriormente, en este capítulo, a partir del Apartado 7.3, se ha dado por supuesto que se tenía un esquema de relación R y que se procedía a normalizarlo. Hay varios modos de obtener ese esquema R:

1. R puede haberse generado al convertir un diagrama E-R en un conjunto de esquemas de relación.
2. R puede haber sido una sola relación que contenía *todos* los atributos que resultaban de interés. El proceso de normalización divide a R en relaciones más pequeñas.
3. R puede haber sido el resultado de algún diseño ad hoc de las relaciones, que hay que comprobar para asegurarse de que satisface la forma normal deseada.

En el resto de este apartado se examinarán las implicaciones de estos enfoques. También se examinarán algunos aspectos prácticos del diseño de las bases de datos, incluida la desnormalización para el rendimiento y ejemplos de mal diseño que no son detectados por la normalización.

7.8.1 El modelo ER y la normalización

Cuando se definen con cuidado los diagramas E-R, identificando correctamente todas las entidades, los esquemas de relación generados a partir de ellos no deben necesitar mucha más normalización. No obstante, puede haber dependencias funcionales entre los atributos de alguna entidad. Por ejemplo, supóngase que la entidad *empleado* tiene los atributos *número_departamento* y *dirección_departamento*, y que hay una dependencia funcional *número_departamento* → *dirección_departamento*. Habrá que normalizar la relación generada a partir de *empleado*.

La mayor parte de los ejemplos de este tipo de dependencias surge de un mal diseño del diagrama E-R. En el ejemplo anterior, si se hubiera diseñado correctamente el diagrama E-R, se habría creado una entidad *departamento* con el atributo *dirección_departamento* y una relación entre *empleado* y *departamento*.

De manera parecida, puede que una relación que implique a más de dos entidades no se halle en la forma normal deseable. Como la mayor parte de las relaciones son binarias, estos casos resultan relativamente raros (de hecho, algunas variantes de los diagramas E-R hacen realmente difícil o imposible especificar relaciones no binarias).

Las dependencias funcionales pueden ayudar a detectar un mal diseño E-R. Si las relaciones generadas no se hallan en la forma normal deseada, el problema puede solucionarse en el diagrama E-R. Es decir, la normalización puede llevarse a cabo formalmente como parte del modelado de los datos. De manera alternativa, la normalización puede dejarse a la intuición del diseñador durante el modelado E-R, y puede hacerse formalmente sobre las relaciones generadas a partir del modelo E-R.

El lector atento se habrá dado cuenta de que para que se pudiera ilustrar la necesidad de las dependencias multivaloradas y de la cuarta forma normal hubo que comenzar con esquemas que no se obtuvieron a partir del diseño E-R. En realidad, el proceso de creación de diseños E-R tiende a generar diseños 4FN. Si se cumple alguna dependencia multivalorada y no la implica la dependencia funcional correspondiente, suele proceder de alguna de las fuentes siguientes:

- Una relación de varios a varios.
- Un atributo multivalorado de un conjunto de entidades.

En las relaciones de varios a varios cada conjunto de entidades relacionado tiene su propio esquema y hay un esquema adicional para el conjunto de relaciones. Para los atributos multivalorados se crea un esquema diferente que consta de ese atributo y de la clave primaria del conjunto de entidades (como en el caso del atributo *nombre_subordinado* del conjunto de entidades *empleado*).

El enfoque de las relaciones universales para el diseño de bases de datos relacionales parte de la suposición de que sólo hay un esquema de relación que contenga todos los atributos de interés. Este esquema único define la manera en que los usuarios y las aplicaciones interactúan con la base de datos.

7.8.2 Denominación de los atributos y de las relaciones

Una característica deseable del diseño de bases de datos es la **asunción de un rol único**, lo que significa que cada nombre de atributo tiene un significado único en toda la base de datos. Esto evita que se utilice el mismo atributo para indicar cosas diferentes en esquemas diferentes. Por ejemplo, puede que, de otra manera, se considerara el empleo del atributo *número* para el número de préstamo en el esquema *préstamo* y para el número de cuenta en el esquema *cuenta*. La reunión de una relación del esquema *préstamo* con otra de *cuenta* carece de significado ("información de los pares préstamo-cuenta en los que coincide el número de préstamo y el de cuenta"). Aunque los usuarios y los desarrolladores de aplicaciones pueden trabajar con esmero para garantizar el empleo del *número* correcto en cada circunstancia, tener nombres de atributo diferentes para el número de préstamo y para el de cuenta sirve para reducir los errores de los usuarios. De hecho, se ha observado la suposición de un rol único en los diseños de bases de datos de este libro, y su aplicación es una buena práctica general que conviene seguir.

Aunque es una buena idea hacer que los nombres de los atributos incompatibles sean diferentes, si los atributos de relaciones diferentes tienen el mismo significado, puede ser conveniente emplear el mismo nombre de atributo. Por ejemplo, se han utilizado los nombres de atributo *id_clente* e *id_empleado* en los conjuntos de entidades *cliente* y *empleado* (y en sus relaciones). Si deseáramos generalizar esos conjuntos de entidades mediante la creación del conjunto de entidades *persona*, habría que renombrar el atributo. Por tanto, aunque no se tenga actualmente una generalización de *cliente* y de *empleado*, si se prevé esa posibilidad es mejor emplear el mismo nombre en los dos conjuntos de relaciones (y en sus relaciones).

Aunque, técnicamente, el orden de los nombres de los atributos en los esquemas no tiene ninguna importancia, es costumbre relacionar en primer lugar los atributos de la clave primaria. Esto facilita la lectura de los resultados predeterminados (como los generados por **select ***).

En los esquemas de bases de datos de gran tamaño los conjuntos de relaciones (y los esquemas derivados) se suelen denominar mediante la concatenación de los nombres de los conjuntos de entidades a los que hacen referencia, quizás con guiones o caracteres de subrayado intercalados. En este libro se han utilizado algunos nombres de este tipo, por ejemplo, *sucursal_cuenta* y *sucursal_préstamo*. Se han utilizado los nombres *prestatario* o *impositor* en lugar de otros concatenados de mayor longitud como *cliente*

_préstamo o *cliente_cuenta*. Esto resultaba aceptable, ya que no es difícil recordar las entidades asociadas a unas pocas relaciones. No siempre se pueden crear nombres de relaciones mediante la mera concatenación; por ejemplo, la relación jefe o trabaja-para entre empleados no tendría mucho sentido si se llamara *empleado_empleado*. De manera parecida, si hay varios conjuntos de relaciones posibles entre un par de conjuntos de entidades, los nombres de las relaciones deben incluir componentes adicionales para identificar cada relación.

Las diferentes organizaciones tienen costumbres diferentes para la denominación de las entidades. Por ejemplo, a un conjunto de entidades de clientes se le puede denominar *cliente* o *clientes*. En los diseños de las bases de datos de este libro se ha decidido utilizar la forma singular. Es aceptable tanto el empleo del singular como el del plural, siempre y cuando la convención se utilice de manera consistente en todas las entidades.

A medida que los esquemas aumentan de tamaño, con un número creciente de relaciones, el empleo de una denominación consistente de los atributos, de las relaciones y de las entidades facilita mucho la vida de los diseñadores de bases de datos y de los programadores de aplicaciones.

7.8.3 Desnormalización para el rendimiento

A veces, los diseñadores de bases de datos escogen un esquema que tiene información redundante; es decir, que no está normalizado. Utilizan la redundancia para mejorar el rendimiento de aplicaciones concretas. La penalización sufrida por no emplear un esquema normalizado es el trabajo adicional (en términos de tiempos de codificación y de ejecución) de mantener consistentes los datos redundantes.

Por ejemplo, supóngase que hay que mostrar el nombre del titular junto con el número de cuenta y con el saldo cada vez que se accede a la cuenta. En el esquema normalizado esto exige una reunión de *cuenta* con *impositor*.

Una alternativa al cálculo de la reunión sobre la marcha es almacenar una relación que contenga todos los atributos de *cuenta* y de *impositor*. Esto hace más rápida la visualización de la información de la cuenta. Sin embargo, la información del saldo de la cuenta se repite para cada uno de los titulares, y la aplicación debe actualizar todas las copias cada vez que se actualice el saldo. El proceso de tomar un esquema normalizado y hacer que no esté normalizado se denomina **desnormalización**, y los diseñadores lo utilizan para ajustar el rendimiento de los sistemas para que den soporte a las operaciones críticas en el tiempo.

Una opción mejor, soportada hoy en día por muchos sistemas de bases de datos, es emplear el esquema normalizado y, además, almacenar la reunión de *cuenta* e *impositor* en forma de vista materializada (recuérdese que las vistas materializadas son vistas cuyo resultado se almacena en la base de datos y se actualiza cuando se actualizan las relaciones utilizadas en ellas). Al igual que la desnormalización, el empleo de las vistas materializadas supone sobrecargas de espacio y de tiempo; sin embargo, presenta la ventaja de que conservar actualizadas las vistas es labor del sistema de bases de datos, no del programador de la aplicación.

7.8.4 Otros problemas del diseño

Hay algunos aspectos del diseño de bases de datos que la normalización no aborda y, por tanto, pueden llevar a un mal diseño de la base de datos. Los datos relativos al tiempo o a intervalos temporales presentan varios de esos problemas. A continuación se ofrecen algunos ejemplos; evidentemente, conviene evitar esos diseños.

Considérese una base de datos empresarial, en la que se desea almacenar los beneficios de varias compañías a lo largo de varios años. Se puede utilizar la relación *beneficios* (*id_empresa, año, importe*) para almacenar la información sobre los beneficios. La única dependencia funcional de esta relación es *id_empresa, año* → *importe*, y se halla en la FNBC.

Un diseño alternativo es el empleo de varias relaciones, cada una de las cuales almacena los beneficios de un año diferente. Supóngase que los años que nos interesan son 2000, 2001 y 2002; se tendrán, entonces, relaciones de la forma *beneficios_2000*, *beneficios_2001* y *beneficios_2002*, todas las cuales se hallan en el esquema (*id_empresa, beneficios*). En este caso, la única dependencia funcional de cada relación será *id_empresa* → *beneficios*, por lo que esas relaciones también se hallan en la FNBC.

No obstante, este diseño alternativo es, claramente, una mala idea—habría que crear una relación nueva cada año, y también habría que escribir consultas nuevas todos los años, para tener en cuenta cada nueva relación. Las consultas también serían más complicadas, ya que probablemente tendrían que hacer referencia a muchas relaciones.

Otra manera de representar esos mismos datos es tener una sola relación *año_empresa* (*id_empresa, beneficios_2000, beneficios_2001, beneficios_2002*). En este caso, las únicas dependencias funcionales van de *id_empresa* hacia los demás atributos y, una vez más, la relación se halla en la FNBC. Este diseño también es una mala idea, ya que tiene problemas parecidos a los del diseño anterior—es decir, habría que modificar el esquema de la relación y escribir consultas nuevas cada año. Las consultas también serían más complicadas, ya que puede que tuvieran que hacer referencia a muchos atributos.

Las representaciones como las de la relación *año_empresa*, con una columna para cada valor de cada atributo, se denominan de **referencias cruzadas**; se emplean mucho en las hojas de cálculo, en los informes y en las herramientas de análisis de datos. Aunque esas representaciones resultan útiles para mostrárselas a los usuarios, por las razones que se acaban de dar, no resultan deseables en el diseño de bases de datos. Se han propuesto extensiones del SQL para pasar los datos de la representación relacional normal a la de referencias cruzadas, para su visualización.

7.9 Modelado de datos temporales

Supóngase que en el banco se conservan datos que no sólo muestran la dirección actual de cada cliente, sino también todas las direcciones anteriores de las que el banco tenga noticia. Se pueden formular consultas como "Averiguar todos los clientes que vivían en Vigo en 1981". En ese caso, puede que se tengan varias direcciones por cliente. Cada dirección tiene asociadas una fecha de comienzo y otra de finalización, que indican el periodo en que el cliente residió en esa dirección. Se puede utilizar un valor especial para la fecha de finalización, por ejemplo, nulo, o un valor que se halle claramente en el futuro, como 31/12/9999, para indicar que el cliente sigue residiendo en esa dirección.

En general, los **datos temporales** son datos que tienen asociado un intervalo de tiempo durante el cual son **válidos**[4]. Se utiliza el término **instantánea** de los datos para indicar su valor en un momento determinado. Por tanto, una instantánea de los datos de los clientes muestra el valor de todos los atributos de los clientes, como la dirección, en un momento concreto.

El modelado de datos temporales es una cuestión interesante por varios motivos. Por ejemplo, supóngase que se tiene una entidad *cliente* con la que se desea asociar una dirección que varía con el tiempo. Para añadir información temporal a una dirección hay que crear un atributo multivalorado, cada uno de cuyos valores es un valor compuesto que contiene una dirección y un intervalo de tiempo. Además de los valores de los atributos que varían con el tiempo, puede que las propias entidades tengan un periodo de validez asociado. Por ejemplo, la entidad cuenta puede tener un periodo de validez desde la fecha de apertura hasta la de cancelación. Las relaciones también pueden tener asociados periodos de validez. Por ejemplo, la relación *impositor* entre un cliente y una cuenta puede registrar el momento en que el cliente pasó a ser titular de la cuenta. Por tanto, habría que añadir intervalos de validez a los valores de los atributos, a las entidades y a las relaciones. La adición de esos detalles a los diagramas E-R hace que sean muy difíciles de crear y de comprender. Ha habido varias propuestas para extender la notación E-R para que especifique de manera sencilla que un atributo o una relación varía con el tiempo, pero no hay ninguna norma aceptada al respecto.

Cuando se realiza un seguimiento del valor de los datos a lo largo del tiempo, puede que dejen de cumplirse dependencias funcionales que se suponía que se cumplían, como

$$id\_cliente \rightarrow calle\_cliente\ ciudad\_cliente$$

En su lugar, se cumpliría la restricción siguiente (expresada en castellano): "*id_cliente* sólo tiene un valor de *calle_cliente* y de *ciudad_cliente* para cada momento *t* dado".

Las dependencias funcionales que se cumplen en un momento concreto se denominan dependencias funcionales temporales. Formalmente, la **dependencia funcional temporal** $X \xrightarrow{\tau} Y$ se cumple en el

4. Hay otros modelos de datos temporales que distinguen entre **periodo de validez** y **momento de la transacción**; el último registra el momento en que se registró un hecho en la base de datos. Para simplificar se prescinde de estos detalles.

esquema de relación R si, para todos los ejemplares legales r de R, todas las instantáneas de r satisfacen la dependencia funcional $X \rightarrow Y$.

Se puede extender la teoría del diseño de bases de datos relacionales para que tenga en cuenta las dependencias funcionales temporales. Sin embargo, el razonamiento con las dependencias funcionales normales ya resulta bastante difícil, y pocos diseñadores están preparados para trabajar con las dependencias funcionales temporales.

En la práctica, los diseñadores de bases de datos recurren a enfoques más sencillos para diseñar las bases de datos temporales. Un enfoque empleado con frecuencia es diseñar toda la base de datos (incluidos los diseños E-R y relacional) ignorando las modificaciones temporales (o, lo que es lo mismo, tomando sólo una instantánea en consideración). Tras esto, el diseñador estudia las diferentes relaciones y decide las que necesitan que se realice un seguimiento de su variación temporal.

El paso siguiente es añadir información sobre los periodos de validez a cada una de esas relaciones, añadiendo como atributos el momento de inicio y el de finalización. Por ejemplo, supóngase que se tiene la relación

asignatura (*id_asignatura*, *denominación_asignatura*)

que asocia la denominación de cada asignatura con la asignatura correspondiente, que queda identificada mediante su identificador de asignatura. La denominación de la asignatura puede variar con el tiempo, lo cual se puede tener en cuenta añadiendo un rango de validez; el esquema resultante sería:

asignatura (*id_asignatura*, *denominación_asignatura*, *comienzo, final*)

Un ejemplar de esta relación puede tener dos registros (CS101, "Introducción a la programación", 01/01/1985, 31/12/2000) y (CS101,"Introdución a C", 01/01/2001, 31/12/9999). Si se actualiza la relación para cambiar la denominación de la asignatura a "Introducción a Java", se actualizaría la fecha "31/12/9999" a la correspondiente al momento hasta el que fue válido el valor anterior ("Introducción a C"), y se añadiría una tupla nueva que contendría la nueva denominación ("Introducción a Java"), con la fecha de inicio correspondiente.

Si otra relación tuviera una clave externa que hiciera referencia a una relación temporal, el diseñador de la base de datos tendría que decidir si la referencia se hace a la versión actual de los datos o a los datos de un momento concreto. Por ejemplo, una relación que registre la asignación de aulas actual para cada asignatura puede hacer referencia de manera implícita al valor temporal actual asociado con cada *id_asignatura*. Por otro lado, los registros del expediente de cada estudiante deben hacer referencia a la denominación de la asignatura en el momento en que la cursó ese estudiante. En este último caso, la relación que hace la referencia también debe almacenar la información temporal, para identificar cada registro concreto de la relación *asignatura*.

La clave primaria original de una relación temporal ya no identificaría de manera unívoca a cada tupla. Para solucionar este problema se pueden añadir a la clave primaria los atributos de fecha de inicio y de finalización. Sin embargo, persisten algunos problemas:

- Es posible almacenar datos con intervalos que se solapen, pero la restricción de clave primaria no puede detectarlo. Si el sistema soporta un tipo nativo *periodo de validez*, puede detectar y evitar esos intervalos temporales que se solapan.
- Para especificar una clave externa que haga referencia a una relación así, las tuplas que hacen la referencia tienen que incluir los atributos de momento inicial y final como parte de su clave externa, y los valores deberán coincidir con los de la tupla a la que hacen referencia. Además, si la tupla a la que hacen referencia se actualiza (y el momento final que se hallaba en el futuro se actualiza), esa actualización debe propagarse a todas las tuplas que hacen referencia a ella.

  Si el sistema soporta los datos temporales de alguna manera mejor, se puede permitir que la tupla que hace la referencia especifique un momento, en lugar de un rango temporal, y confiar en que el sistema garantice que hay una tupla en la relación a la que hace referencia cuyo periodo de validez contenga ese momento. Por ejemplo, un registro de un expediente puede especificar *id _asignatura* y un momento (por ejemplo, la fecha de comienzo de un trimestre), lo que basta para identificar el registro correcto de la relación *asignatura*.

Como caso especial frecuente, si todas las referencias a los datos temporales se hacen exclusivamente a los datos actuales, una solución más sencilla es no añadir información temporal a la relación y, en su lugar, crear la relación *historial* correspondiente que contenga esa información temporal, para los valores del pasado. Por ejemplo, en la base de datos bancaria, se puede utilizar el diseño que se ha creado, ignorando las modificaciones temporales, para almacenar únicamente la información actual. Toda la información histórica se traslada a las relaciones históricas. Por tanto, la relación *cliente* sólo puede almacenar la dirección actual, mientras que la relación *historial_cliente* puede contener todos los atributos de *cliente*, con los atributos adicionales *momento_inicial* y *momento_final*.

Aunque no se ha ofrecido ningún método formal para tratar los datos temporales, los problemas estudiados y los ejemplos ofrecidos deben ayudar al lector a diseñar bases de datos que registren datos temporales. Más adelante, en el Apartado 24.2, se tratan otros aspectos del manejo de datos temporales, incluidas las consultas temporales.

7.10 Resumen

- Se han mostrado algunas dificultades del diseño de bases de datos y el modo de diseñar de manera sistemática esquemas de bases de datos que eviten esas dificultades. Entre esas dificultades están la información repetida y la imposibilidad de representar cierta información.
- Se ha mostrado el desarrollo del diseño de bases de datos relacionales a partir de los diseños E-R, cuándo los esquemas se pueden combinar con seguridad y cuándo se deben descomponer. Todas las descomposiciones válidas deben ser sin pérdidas.
- Se han descrito las suposiciones de dominios atómicos y de primera forma normal.
- Se ha introducido el concepto de las dependencias funcionales y se ha utilizado para presentar dos formas normales, la forma normal de Boyce–Codd (FNBC) y la tercera forma normal (3NF).
- Si la descomposición conserva las dependencias, dada una actualización de la base de datos, todas las dependencias funcionales pueden verificarse a partir de las diferentes relaciones, sin necesidad de calcular la reunión de las relaciones de la descomposición.
- Se ha mostrado la manera de razonar con las dependencias funcionales. Se ha puesto un énfasis especial en señalar las dependencias que están implicadas lógicamente por conjuntos de dependencias. También se ha definido el concepto de recubrimiento canónico, que es un conjunto mínimo de dependencias funcionales equivalente a un conjunto dado de dependencias funcionales.
- Se ha descrito un algoritmo para la descomposición de las relaciones en la FNBC. Hay relaciones para las cuales no hay ninguna descomposición en la FNBC que conserve las dependencias.
- Se han utilizado los recubrimientos canónicos para descomponer las relaciones en la 3NF, que es una pequeña relajación de las condiciones de la FNBC. Las relaciones en la 3NF pueden tener alguna redundancia, pero siempre hay una descomposición en la 3NF que conserva las dependencias.
- Se ha presentado el concepto de dependencias multivaloradas, que especifican las restricciones que no pueden especificarse únicamente con las dependencias funcionales. Se ha definido la cuarta forma normal (4FN) con las dependencias multivaloradas. El Apartado C.1.1 del apéndice da detalles del razonamiento sobre las dependencias multivaloradas.
- Otras formas normales, como la FNRP y la FNDC, eliminan formas más sutiles de redundancia. Sin embargo, es difícil trabajar con ellas y se emplean rara vez. El Apéndice C ofrece detalles de estas formas normales.
- Al revisar los temas de este capítulo hay que tener en cuenta que el motivo de que se hayan podido definir enfoques rigurosos del diseño de bases de datos relacionales es que el modelo relacional de datos descansa sobre una base matemática sólida. Ésa es una de las principales ventajas del modelo relacional en comparación con los otros modelos de datos que se han estudiado.

Términos de repaso

- Modelo E-R y normalización.
- Descomposición.
- Dependencias funcionales.
- Descomposición sin pérdidas.
- Dominios atómicos.
- Primera forma normal (1FN).
- Relaciones legales.
- Superclave.
- R satisface F.
- F se cumple en R.
- Forma normal de Boyce–Codd (FNBC).
- Conservación de las dependencias.
- Tercera forma normal (3NF).
- Dependencias funcionales triviales.
- Cierre de un conjunto de dependencias funcionales.
- Axiomas de Armstrong.
- Cierre de los conjuntos de atributos.
- Restricción de F a R_i.
- Recubrimiento canónico.
- Atributos raros.
- Algoritmo de descomposición en la FNBC.
- Algoritmo de descomposición en la 3NF.
- Dependencias multivaloradas.
- Cuarta forma normal (4FN).
- Restricción de las dependencias multivaloradas.
- Forma normal de reunión por proyección (FNRP).
- Forma normal de dominios y claves (FNDC).
- Relación universal.
- Suposición de un rol único.
- Desnormalización.

Ejercicios prácticos

7.1 Supóngase que se descompone el esquema $R = (A, B, C, D, E)$ en

$$(A, B, C)$$
$$(A, D, E)$$

Demuéstrese que esta descomposición es una descomposición sin pérdidas si se cumple el siguiente conjunto F de dependencias funcionales:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

7.2 Indíquense todas las dependencias funcionales que satisface la relación de la Figura 7.18.

7.3 Explíquese el modo en que se pueden utilizar las dependencias funcionales para indicar:

- Existe un conjunto de relaciones de uno a uno entre los conjuntos de entidades *cuenta* y *cliente*.
- Existe un conjunto de relaciones de varios a uno entre los conjuntos de entidades *cuenta* y *cliente*.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Figura 7.18 La relación del Ejercicio práctico 7.2.

7.4 Empléense los axiomas de Armstrong para probar la corrección de la regla de la unión. *Sugerencia*: utilícese la regla de la aumentatividad para probar que, si $\alpha \rightarrow \beta$, entonces $\alpha \rightarrow \alpha\beta$. Aplíquese nuevamente la regla de la aumentatividad, utilizando $\alpha \rightarrow \gamma$, y aplíquese luego la regla de la transitividad.

7.5 Empléense los axiomas de Armstrong para probar la corrección de la regla de la pseudotransitividad.

7.6 Calcúlese el cierre del siguiente conjunto F de relaciones funcionales para el esquema de relación $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Indíquense las claves candidatas de R.

7.7 Utilizando las dependencias funcionales del Ejercicio práctico 7.6, calcúlese el recubrimiento canónico F_c.

7.8 Considérese el algoritmo de la Figura 7.19 para calcular α^+. Demuéstrese que este algoritmo resulta más eficiente que el presentado en la Figura 7.9 (Apartado 7.4.2) y que calcula α^+ de manera correcta.

7.9 Dado el esquema de base de datos $R(a, b, c)$ y una relación *r* del esquema R, escríbase una consulta SQL para comprobar si la dependencia funcional $b \rightarrow c$ se cumple en la relación *r*. Escríbase también un aserto de SQL que haga que se cumpla la dependencia funcional. Supóngase que no hay ningún valor nulo.

7.10 Sea $R_1, R_2, \ldots, R_n$ una descomposición del esquema U. Sea $u(U)$ una relación y sea $r_i = \Pi_{R_I}(u)$. Demuéstrese que

$$u \subseteq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$$

7.11 Demuéstrese que la descomposición del Ejercicio práctico 7.1 no es una descomposición que conserve las dependencias.

7.12 Demuéstrese que es posible asegurar que una descomposición que conserve las dependencias en la 3FN sea una descomposición sin pérdidas garantizando que, como mínimo, un esquema contenga una clave candidata para el esquema que se está descomponiendo. *Sugerencia*: demuéstrese que la reunión de todas las proyecciones en los esquemas de la descomposición no puede tener más tuplas que la relación original.

7.13 Dese un ejemplo de esquema de relación R' y de un conjunto F' de dependencias funcionales tales que haya, al menos, tres descomposiciones sin pérdidas distintas de R' en FNBC.

7.14 Sea atributo *primo* el que aparece, como mínimo, en una clave candidata. Sean α y β conjuntos de atributos tales que se cumple $\alpha \rightarrow \beta$, pero no se cumple $\beta \rightarrow \alpha$. Sea A un atributo que no esté en α ni en β y para el que se cumpla que $\beta \rightarrow \alpha$. Se dice que A es *dependiente de manera transitiva* de α. Se puede reformular la definición de la 3FN de la manera siguiente: el esquema de relación R está en la 3FN con respecto al conjunto F de dependencias funcionales si no hay atributos no primos A en R para los cuales A sea dependiente de manera transitiva de una clave de R.

Demuéstrese que esta nueva definición es equivalente a la original.

7.15 La dependencia funcional $\alpha \rightarrow \beta$ se denomina **dependencia parcial** si hay un subconjunto propio γ de α tal que $\gamma \rightarrow \beta$. Se dice que β es *parcialmente dependiente* de α. El esquema de relación R está en la **segunda forma normal** (2FN) si cada atributo A de R cumple uno de los criterios siguientes:

- Aparece en una clave candidata.

```
resultado := ∅;
/* cuentadf es un array cuyo elemento i-ésimo contiene
   el número de atributos del lado izquierdo de la i-ésima
   DF que todavía no se sabe que estén en α+ */
for i := 1 to |F| do
   begin
      Supóngase que β → γ denota la iésima DF;
      cuentadf [i] := |β|;
   end
/* aparece es un array con una entrada por cada atributo. La
entrada del atributo A es una lista de enteros. Cada entero
i de la lista indica que A aparece en el lado izquierdo de
la i-iésima DF */
for each atributo A do
   begin
      aparece [A] := NIL;
      for i := 1 to |F| do
         begin
            Supóngase que β → γ denota la i-ésima DF;
            if A ∈ β then añadir i a aparece [A];
         end
   end
agregar (α);
return (resultado);

procedure agregar (α);
for each atributo A in α do
   begin
      if A ∉ resultado then
         begin
            resultado := resultado ∪ {A};
            for each elemento i in aparece[A] do
               begin
                  cuentafd [i] := cuentafd [i] − 1;
                  if cuentafd [i] := 0 then
                     begin
                        supóngase que β → γ denota la i-ésima FD;
                        agregar (γ);
                     end
               end
         end
   end
```

Figura 7.19 Algoritmo para calcular α^+.

- No es parcialmente dependiente de una clave candidata.

Demuéstrese que cada esquema en la 3FN se halla en la 2FN. *Sugerencia*: demuéstrese que todas las dependencias parciales son dependencias transitivas.

7.16 Dese un ejemplo de esquema de relación R y un conjunto de dependencias tales que R se halle en la FNBC, pero no en la 4FN.

Ejercicios

7.17 Explíquese lo que se quiere decir con *repetición de la información* e *imposibilidad de representación de la información*. Explíquese el motivo por el que estas propiedades pueden indicar un mal diseño de las bases de datos relacionales.

7.18 Indíquese el motivo de que ciertas dependencias funcionales se denominen dependencias funcionales *triviales*.

7.19 Utilícese la definición de dependencia funcional para argumentar que cada uno de los axiomas de Armstrong (reflexividad, aumentatividad y transitividad) es correcto.

7.20 Considérese la siguiente regla propuesta para las dependencias funcionales: si $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, entonces $\alpha \rightarrow \gamma$. Pruébese que esta regla *no* es correcta mostrando una relación *r* que satisfaga $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, pero no $\alpha \rightarrow \gamma$.

7.21 Utilícense los axiomas de Armstrong para probar la corrección de la regla de la descomposición.

7.22 Utilizando las dependencias funcionales del Ejercicio práctico 7.6, calcúlese B^+.

7.23 Demuéstrese que la siguiente descomposición del esquema R del Ejercicio práctico 7.1 no es una descomposición sin pérdidas:

$$(A, B, C)$$
$$(C, D, E)$$

Sugerencia: dese un ejemplo de una relación *r* del esquema R tal que

$$\Pi_{A,B,C}(r) \bowtie \Pi_{C,D,E}(r) \neq r$$

7.24 Indíquense los tres objetivos de diseño de las bases de datos relacionales y explíquese el motivo de que cada uno de ellos sea deseable.

7.25 Dese una descomposición sin pérdidas en la FNBC del esquema R del Ejercicio práctico 7.1.

7.26 Al diseñar una base de datos relacional, indíquese el motivo de que se pueda escoger un diseño que no esté en la FNBC.

7.27 Dese una descomposición sin pérdidas en la 3FN que conserve las dependencias del esquema R del Ejercicio práctico 7.1.

7.28 Dados los tres objetivos del diseño de bases de datos relacionales, indíquese si hay alguna razón para diseñar un esquema de base de datos que se halle en la 2FN, pero que no se halle en ninguna forma normal de orden superior (véase el Ejercicio práctico 7.15 para obtener la definición de la 2FN).

7.29 Dado el esquema relacional $r(A, B, C, D)$, ¿implica lógicamente $A \twoheadrightarrow BC$ a $A \twoheadrightarrow B$ y a $A \twoheadrightarrow C$? En caso positivo, pruébese; en caso contrario, dese un contraejemplo.

7.30 Explíquese el motivo de que la 4FN sea una forma normal más deseable que la FNBC.

Notas bibliográficas

El primer estudio de la teoría del diseño de bases de datos relacionales apareció en un artículo pionero de Codd [1970]. En ese artículo, Codd introducía también las dependencias funcionales y la primera, la segunda y la tercera formas normales.

Los axiomas de Armstrong se introdujeron en Armstrong [1974]. A finales de los años setenta se produjo un desarrollo significativo de la teoría de las bases de datos relacionales. Esos resultados se recogen en varios textos sobre la teoría de las bases de datos, como Maier [1983], Atzeni y Antonellis [1993] y Abiteboul et al. [1995].

La FNBC se introdujo en Codd [1972]. Biskup et al. [1979] dan el algoritmo que se ha utilizado para encontrar descomposiciones en la 3FN que conserven las dependencias. Los resultados fundamentales de la propiedad de la descomposición sin pérdidas aparecen en Aho et al. [1979a].

Beeri et al. [1977] dan un conjunto de axiomas para las dependencias multivaluadas y prueba que los axiomas de los autores son correctos y completos. Los conceptos de 4FN, de FNRP y de FNDC son de Fagin [1977], Fagin [1979] y de Fagin [1981], respectivamente. Véanse las notas bibliográficas del Apéndice C para tener más referencias sobre la literatura relativa a la normalización.

Jensen et al. [1994] presenta un glosario de conceptos relacionados con las bases de datos temporales. Gregersen y Jensen [1999] presenta un resumen de las extensiones del modelo E-R para el tratamiento de datos temporales. Tansel et al. [1993] trata la teoría, el diseño y la implementación de las bases de datos temporales. Jensen et al. [1996] describe las extensiones de la teoría de la dependencia a los datos temporales.