CAPÍTULO

7 DISEÑO DE BASES DE DATOS RELACIONALES

Este capítulo continúa el estudio de los problemas de diseño de las bases de datos relacionales. En general, el objetivo del diseño de las bases de datos relacionales es la generación de un conjunto de esquemas relacionales que nos permita almacenar la información sin redundancias innecesarias, pero que también nos permita recuperar fácilmente esa información. Un enfoque es el diseño de esquemas que se hallen en una *forma normal* adecuada. Para determinar si el esquema de una relación se halla en una de las formas normales deseables hace falta información adicional sobre la empresa real que ese está modelando con la base de datos. En este capítulo se introduce el concepto de la dependencia funcional. Luego se definirán las formas normales en términos de las dependencias funcionales y otros tipos de dependencias de datos.

7.1. PRIMERA FORMA NORMAL

La primera de las formas normales que se van a estudiar, la **primera forma normal**, impone un requisito muy elemental a las relaciones; a diferencia de las demás formas normales, no exige información adicional como las dependencias funcionales.

Un dominio es **atómico** si se considera que los elementos del dominio son unidades indivisibles. Se dice que el esquema de una relación *R* está en la **primera forma normal** (1FN) si los dominios de todos los atributos de *R* son atómicos.

Un conjunto de nombres es un ejemplo de valor no atómico. 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 se hallaría en la primera forma normal.

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

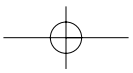
Se da por supuesto que los enteros son atómicos, por lo que el conjunto de enteros es un dominio atómico; el conjunto de todos los conjuntos de enteros es un dominio no atómico. La diferencia estriba en que normalmente no se considera que los enteros tengan subpartes, pero sí se considera que los tienen 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 del dominio en la base de datos.

El dominio de todos los enteros no sería 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 el empleado dentro de ese departamento. Ejemplos de estos números pueden ser *IN*0012 y *EE*1127. Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de una relación tuviera un atributo cuyo dominio consistiera en números de identificación codificados como se ha indicado, el esquema no se hallaría en la primera forma normal.

Cuando se utilizan estos números de identificación, se puede averiguar el departamento de cada empleando 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 estos números de identificación como claves principales: Cada vez que un empleado cambia de departamento hay que cambiar su número de identificación, lo que puede constituir una tarea difícil, o en su defecto el código que interpreta ese número dará un resultado erróneo.

El empleo de atributos con el valor dado por el conjunto puede llevar a diseños con almacenamiento de datos redundantes, 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 una relación independiente *impositor*, puede que un 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, hay que llevar a cabo la actualización en dos lugares; no llevar a cabo las dos actualizaciones puede dejar la base



de datos en un estado inconsistente. La conservación de sólo uno de estos conjuntos evitaría la información repetida, pero complicaría algunas consultas. También es más complicado tanto escribir consultas con los atributos con el valor dado por el conjunto como razonar sobre ellos.

En este capítulo sólo se toman en consideración dominios atómicos y se da por supuesto que las relaciones están en la primera forma normal. Aunque no se haya mencionado antes la primera forma normal, cuando se introdujo el modelo relacional en el Capítulo 3, se afirmó que los valores de los atributos deben ser atómicos.

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 de tipo 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 representa 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 a la forma atómica y desde ella. Por tanto, el soporte de los valores no atómicos puede resultar muy útil en esos dominios. De hecho, los sistemas modernos de bases de datos soportan muchos tipos de valores no atómicos, como se verá en los capítulos 8 y 9. Sin embargo, en este capítulo nos limitaremos a las relaciones en la primera forma normal.

7.2. DIFICULTADES EN EL DISEÑO DE BASES DE DATOS RELACIONALES

Antes de continuar con el estudio de las formas normales hay que examinar lo que puede salir mal en un mal diseño de bases de datos. Entre las propiedades indeseables que puede tener un mal diseño están:

- Repetición de la información
- Imposibilidad de la representación de determinada información

Estos problemas se estudiarán con ayuda de un diseño de base de datos modificado para el ejemplo bancario: A diferencia del esquema de relación utilizado en los capítulos 3 a 6, supóngase que la información relativa a los préstamos se guarda en una sola relación, *empréstito*, que se define mediante el esquema de relación

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

La Figura 7.1 muestra un ejemplo de la relación *empréstito* (*esquema-empréstito*). Cada tupla *t* de la relación *empréstito* tiene el siguiente significado intuitivo:

- *t*[*activo*] es el volumen de activo de la sucursal denominada *t*[*nombre-sucursal*].
- *t*[*ciudad-sucursal*] es la ciudad en la que se ubica la sucursal denominada *t*[*nombre-sucursal*].
- *t*[*número-préstamo*] es el número asignado al préstamo concedido por la sucursal denominada *t*[*nombre-sucursal*] al cliente llamado *t*[*nombre-cliente*].
- *t*[*importe*] es el importe del préstamo cuyo número es *t*[*número-préstamo*].

Supóngase que se desea añadir un nuevo préstamo a la base de datos. Digamos que el préstamo se lo concede la sucursal de Navacerrada a la señora Fernández por un importe de 1500 €. Sea el *número-préstamo* P-31.

(Navacerrada, Aluche, 1.700.000, Fernández, P-31, 1.500)

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*	*número-préstamo*	*importe*
Centro	Arganzuela	9.000.000	Santos	P-17	1.000
Moralzarzal	La Granja	2.100.000	Gómez	P-23	2.000
Navacerrada	Aluche	1.700.000	López	P-15	1.500
Centro	Arganzuela	9.000.000	Sotoca	P-14	1.500
Becerril	Aluche	400.000	Santos	P-93	500
Collado Mediano	Aluche	8.000.000	Abril	P-11	900
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso	P-29	1.200
Segovia	Cerceda	3.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	González	P-18	2.000
Navacerrada	Aluche	1.700.000	Rodríguez	P-25	2.500
Galapagar	Arganzuela	7.100.000	Amo	P-10	2.200

FIGURA 7.1. Relación *empréstito* de ejemplo.

En el diseño hace falta una tupla con los valores de todos los atributos de *Esquema-empréstito*. Por tanto, hay que repetir los datos del activo y de la ciudad de la sucursal de Navacerrada y añadir la tupla a la relación *empréstito*. En general, los datos del activo y de la ciudad deben aparecer una vez para cada préstamo concedido por esa sucursal.

La repetición de la información en el diseño alternativo no es deseable. La repetición de la información desaprovecha el espacio. Además, complica la actualización de la base de datos. Supóngase, por ejemplo, que el activo de la sucursal de Navacerrada cambia de 1.700.000 a 1.900.000. Con el diseño original hay que modificar una tupla de la relación *sucursal*. Con el diseño alternativo hay que modificar muchas tuplas de la relación *empréstito*. Por tanto, las actualizaciones resultan más costosas con el diseño alternativo que con el original. Cuando se lleva a cabo la actualización en la base de datos alternativa hay que asegurarse de que se actualicen *todas* las tuplas correspondientes a la sucursal de Navacerrada, o la base de datos mostrará dos valores diferentes del activo para esa sucursal.

Esa observación resulta fundamental para la comprensión del motivo por el que el diseño alternativo es malo. Se sabe que cada sucursal bancaria tiene un valor único del activo, por lo que dado el nombre de una sucursal se puede identificar de manera única el valor del activo. Por otro lado, se sabe que cada sucursal puede conceder muchos préstamos por lo que, dado el nombre de una sucursal, no se puede determinar de manera única el número de un préstamo. En otras palabras, se dice que se cumple la *dependencia funcional*

$$nombre\text{-}sucursal \rightarrow activo$$

para *Esquema-empréstito*, pero no se espera que se cumpla la dependencia funcional *nombre-sucursal* → *número-préstamo*. El hecho de que cada sucursal tenga un valor concreto del activo y el de que cada sucursal conceda préstamos son independientes y, como se ha visto, esos hechos quedan mejor representados en relaciones distintas. Se verá que se pueden utilizar las dependencias funcionales para la especificación formal cuando el diseño de la base de datos es bueno.

Otro problema del diseño *Esquema-empréstito* es que no se puede representar de manera directa la información relativa a cada sucursal (*nombre-sucursal*, *ciudad-sucursal*, *activo*) a menos que haya como mínimo un préstamo en esa sucursal. Esto se debe a que las tuplas de la relación *empréstito* exigen los valores de *número-préstamo*, *importe* y *nombre-cliente*.

Una solución para este problema es introducir los *valores nulos*, como se hizo para tratar las actualizaciones mediante las vistas. Hay que recordar, no obstante, que los valores nulos resultan difíciles de manejar, como se vio en el Apartado 3.3.4. Si no se desea tratar con los valores nulos se puede crear la información sobre cada sucursal sólo después de que se formule la primera solicitud de préstamo en esa sucursal. Lo que es peor aún, habrá que eliminar esa información cuando se hayan pagado todos los préstamos. Evidentemente, esta situación no resulta deseable ya que, con el diseño original de la base de datos, la información sobre la sucursal estaba disponible independientemente de si se mantenía algún préstamo vivo en la sucursal, y sin recurrir a los valores nulos.

7.3. DEPENDENCIAS FUNCIONALES

Las dependencias funcionales desempeñan un papel fundamental en la diferenciación entre los buenos diseños de bases de datos y los malos. Una **dependencia funcional** es un tipo de restricción que constituye una generalización del concepto de *clave*, como se estudió en los capítulos 2 y 3.

7.3.1. Conceptos básicos

Las dependencias funcionales son restricciones del conjunto de relaciones legales. Permiten expresar hechos sobre la empresa que se modela con la base de datos.

En el Capítulo 2 se definió el concepto de *superclave* de la manera siguiente. Sea R el esquema de una relación. El subconjunto K 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 $t_1 \neq t_2$, $t_1[K] \neq t_2[K]$. Es decir, ningún par de tuplas de una relación legal $r(R)$ puede tener el mismo valor para el conjunto de atributos K.

El concepto de dependencia funcional generaliza la noción de superclave. 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 ocurre que $t_1[\beta] = t_2[\beta]$.

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

Las dependencias funcionales nos permiten expresar las restricciones que no se pueden expresar con las superclaves. Considérese el esquema

Esquema-info-préstamo = (*número-préstamo, nombre-sucursal, nombre-cliente, importe*)

que es una simplificación de *Esquema-empréstito*, que se ha visto anteriormente. El conjunto de dependencias funcionales que se espera que se cumplan en este esquema de relación es

número-préstamo → *importe*
número-préstamo → *nombre-sucursal*

Sin embargo, no se espera que se cumpla la dependencia funcional

número-préstamo → *nombre-cliente*

ya que, en general, cada préstamo se puede conceder a más de un cliente (por ejemplo, a los dos integrantes de una pareja marido-esposa).

Las dependencias funcionales se utilizarán de dos maneras:

1. Para probar las relaciones y ver si son legales según 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 por las relaciones que satisfagan un conjunto dado de dependencias funcionales. Si uno desea restringirse a las relaciones del esquema R que satisfagan el conjunto F de dependencias funcionales, se dice que F **se cumple** en R.

Considérese la relación r de la Figura 7.2, 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 para C, por ejemplo, c_1. De manera parecida, las dos tuplas con un valor para A de a_2 tienen el mismo valor para C, c_2. No hay más pares de tuplas diferentes que tengan el mismo valor para 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 los mismos valores para 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 $t_1[C] = t_2[C]$, pero $t_1[A] \neq t_2[A]$.

r satisface muchas otras dependencias funcionales, incluida, por ejemplo, la dependencia funcional $AB \rightarrow D$. Obsérvese que se utiliza AB como abreviatura de $\{A,B\}$, para adecuarnos al uso estándar. Obsérvese que no hay ningún par de tuplas diferentes t_1 y t_2 tales que $t_1[AB] = t_2[AB]$. Por tanto, si $t_1[AB] = t_2[AB]$, debe ser que $t_1 = t_2$ y, por tanto, $t_1[D] = t_2[D]$. Así, r satisface $AB \rightarrow D$.

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 impliquen 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 parecida, $AB \rightarrow A$ la satisfacen todas las relaciones que impliquen al atributo A. En general, una dependencia funcional de la forma $\alpha \rightarrow \beta$ es **trivial** si $\beta \subseteq \alpha$.

Para distinguir entre los conceptos de que una relación satisfaga una dependencia y que una dependencia se cumpla en un esquema hay que volver al ejemplo del banco. Si se considera la relación *cliente* (en *Esquema-cliente*) de la Figura 7.3, puede verse que se satisface *calle-cliente* → *ciudad-cliente*. Sin embargo, se sabe que en el mundo real dos ciudades pueden tener calles que se llamen igual. Por tanto, resulta posible, en un momento dado, tener un ejemplar de la relación *cliente* en la 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 *Esquema-cliente*.

En la relación *préstamo* (de *Esquema-préstamo*) de la Figura 7.4 se puede ver que se satisface la dependencia *número-préstamo* → *importe*. A diferencia del caso de *ciudad-cliente* y *calle-cliente* de *Esquema-cliente*, se sabe que en la empresa real que se está modelando se exige que cada préstamo tenga un único importe. Por tanto, se desea exigir que la relación *préstamo* satisfaga siempre *número-préstamo* → *importe*. En otras palabras, se exige que la restricción *número-préstamo* → *importe* se cumpla en *Esquema-préstamo*.

En la relación *sucursal* de la Figura 7.5 puede verse que se satisface *nombre-sucursal* → *activo*, igual que ocurre con *activo* → *nombre-sucursal*. Se desea exigir

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 7.2. Relación de ejemplo *r*.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
Santos	Mayor	Peguerinos
Gómez	Carretas	Cerceda
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rupérez	Ramblas	León
Abril	Preciados	Valsaín
Valdivieso	Goya	Vigo
Fernández	Jazmín	León
González	Arenal	La Granja
Rodríguez	Yeserías	Cádiz
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín

FIGURA 7.3. La relación *cliente*.

número-préstamo	*nombre-sucursal*	*importe*
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-15	Navacerrada	1.500
P-14	Centro	1.500
P-93	Becerril	500
P-11	Collado Mediano	900
P-29	Navas de la Asunción	1.200
P-16	Segovia	1.300
P-18	Centro	2.000
P-25	Navacerrada	2.500
P-10	Galapagar	2.200

FIGURA 7.4. La relación *préstamo*.

nombre-sucursal	*ciudad-sucursal*	*activo*
Centro	Arganzuela	9.000.000
Moralzarzal	La Granja	2.100.000
Navacerrada	Aluche	1.700.000
Becerril	Aluche	400.000
Collado Mediano	Aluche	8.000.000
Navas de la Asunción	Alcalá de Henares	300.000
Segovia	Cerceda	3.700.000
Galapagar	Arganzuela	7.100.000

FIGURA 7.5. La relación *sucursal*.

que se cumpla *nombre-sucursal* → *activo* en *Esquema-sucursal*. Sin embargo, no se desea exigir que se cumpla *activo* → *nombre-sucursal*, ya que es posible tener varias sucursales con el mismo valor del activo.

En lo que viene a continuación se da por supuesto que, cuando se diseña una base de datos relacional, se enumeran en primer lugar las dependencias funcionales que se deben cumplir siempre. En el ejemplo del banco, en la lista de dependencias figuran:

- En *Esquema-sucursal*:
  nombre-sucursal → *ciudad-sucursal*
  nombre-sucursal → *activo*
- En *Esquema-cliente*:
  nombre-cliente → *ciudad-cliente*
  nombre-cliente → *calle-cliente*
- En *Esquema-préstamo*:
  número-préstamo → *importe*
  número-préstamo → *nombre-sucursal*
- En *Esquema-prestatario*:
  Ninguna dependencia funcional
- En *Esquema-cuenta*:
  número-cuenta → *nombre-sucursal*
  número-cuenta → *saldo*
- En *Esquema-impositor*:
  Ninguna dependencia funcional

7.3.2. Cierre de un conjunto 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 se cumplen otras dependencias funcionales determinadas. Se dice que esas dependencias funcionales están «implicadas lógicamente» por F.

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 un 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, en la 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 en F. Dado F, se puede calcular F^+ directamente a partir de la definición formal de dependencia funcional. Si F fuera de gran tamaño, este proceso sería prolongado y difícil. Este 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 ejemplo de dependencias.

Los **axiomas**, o reglas de inferencia, proporcionan una técnica más sencilla para el razonamiento sobre las depen-

dencias 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 individuales. Se utiliza ab 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 también se cumple que $\beta \rightarrow \gamma$, entonces se cumple que $\alpha \rightarrow \gamma$.

Los axiomas de Armstrong son **correctos** porque no generan dependencias funcionales incorrectas. Son **completos**, porque, para un conjunto dado F de dependencias funcionales, 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 difícil utilizarlos directamente para el cálculo de F^+. Para simplificar más las cosas se relacionan unas reglas adicionales. Resulta posible utilizar los axiomas de Armstrong para probar que estas reglas son correctas (véanse los ejercicios 7.8, 7.9 y 7.10).

- **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 $\alpha\gamma \rightarrow \delta$.

Apliquemos ahora las reglas al ejemplo del esquema $R = (A, B, C, G, H, I)$ y el 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 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 hallar 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 $CG \rightarrow I$, se infiere que $AG \rightarrow I$.

La Figura 7.6 muestra un procedimiento que demuestra formalmente el modo de utilizar los axiomas de Armstrong para calcular F^+. En este procedimiento, cuando se añade una dependencia funcional a F^+, puede que ya esté presente y, en ese caso, no hay ninguna modificación en F^+. También se verá una manera alternativa de calcular F^+ en el Apartado 7.3.3.

Los términos a la derecha y a la izquierda de una dependencia funcional 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 repeat del procedimiento, salvo la última, añade como mínimo una dependencia funcional a F^+. Por tanto, se garantiza que el procedimiento se termine.

7.3.3. Cierre de un conjunto de atributos

Para comprobar si un conjunto α es una 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.

El algoritmo eficiente para calcular el conjunto de atributos determinados funcionalmente por a no sólo resulta útil para comprobar si a es una 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 7.7 muestra un algoritmo, escrito en pseudocó-

```
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 pareja de dependencias funcionales f1 y f2 de F+
      if f1 y f2 pueden combinarse mediante la transitividad
         añadir la dependencia funcional resultante a F+
until F+ no cambie más
```

FIGURA 7.6. Procedimiento para calcular F^+.

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

FIGURA 7.7. Algoritmo para el cálculo de α^+, el cierre de α bajo F.

digo, 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 7.3.2. Se comienza con *resultado* = *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 ver este hecho, se observa 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** no se añaden atributos nuevos a *resultado*, y se termina el algoritmo.

Veamos ahora el motivo por el que el algoritmo de la Figura 7.7 es correcto. El primer paso es correcto, ya que $\alpha \rightarrow \alpha$ se cumple siempre (por la regla de reflexividad). Se asegura que, para cualquier subconjunto β de *resultado*, $\alpha \rightarrow \beta$. Dado que se inicia el bucle **while** 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 $\alpha \rightarrow \gamma$ (utilizando $\alpha \rightarrow \beta$ 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*.

Resulta que, en el peor de los casos, este algoritmo puede tardar 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; este algoritmo se presenta como parte del Ejercicio 7.14.

Hay varios usos para el algoritmo de cierre de atributos:

- Comprobar si α es una superclave, se calcula α^+ y se comprueba si α^+ 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 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 una dependencia funcional $\gamma \rightarrow S$.

7.3.4. Recubrimiento canónico

Supóngase que se tiene un conjunto de dependencias funcionales F 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.

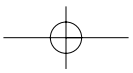
El sistema debe retroceder la actualización si viola alguna dependencia funcional del conjunto F.

Se puede reducir el esfuerzo empleado en 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 el 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 α 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 $AB \rightarrow C$. Otro ejemplo más: 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 prestar atención a la dirección de las implicaciones cuando se utiliza 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 también F implica siempre lógicamente a $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$

He aquí 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 $F' = \{AB \rightarrow D, A \rightarrow E \text{ y } E \rightarrow C\}$. El cierre es $ABCDE$, que incluye a CD, por lo que se infiere que C es raro.

El **recubrimiento canónico** F_c de F es un conjunto de dependencias tales 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 7.8. 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 estas dependencias funcionales.

Se puede demostrar que el recubrimiento canónico de F, F_c, tiene el mismo cierre que F; por tanto, la comprobación de si se satisface F_c es equivalente a la comprobación de 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 lado izquierdo idéntico. 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 $AB \rightarrow C$ porque 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 que toda una dependencia funcional del conjunto sea rara, en el sentido de que su eliminación no modifica 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 contra-

```
Fc = F
repeat
    Utilizar la regla de unión para sustituir las dependencias de Fc de la forma
                α1 → β1 y α1 → β2 con α1 → β1 β2.
    Hallar una dependencia funcional α → β de Fc con un atributo raro en α o en β.
        /* Nota: la comprobación de los atributos raros se lleva a cabo empleando Fc, no F */
    Si se halla algún atributo raro, hay que eliminarlo de α → β.
until Fc ya no cambie.
```

FIGURA 7.8. Cálculo del recubrimiento canónico.

rio, existiera esa dependencia rara en F_c. Los atributos del lado derecho de la dependencia serían raros, lo que no es posible por la definición de los recubrimientos canónicos.

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 eliminarlos a los dos. El algoritmo para hallar el recubrimiento canónico selecciona uno de ellos 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,
   $F_c = \{A \rightarrow B, B \rightarrow C \text{ y } C \rightarrow A\}$ y
   $F_c = \{A \rightarrow B, B \rightarrow AC \text{ y } C \rightarrow B\}$.
2. Si se elimina B, se obtiene el conjunto $\{A \rightarrow C, B \rightarrow AC \text{ y } C \rightarrow AB\}$. Este caso es simétrico al anterior, y genera dos recubrimientos canónicos,
   $F_c = \{A \rightarrow C, C \rightarrow B \text{ y } B \rightarrow A\}$ y
   $F_c = \{A \rightarrow C, B \rightarrow C \text{ y } C \rightarrow AB\}$.

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

7.4. DESCOMPOSICIÓN

El mal diseño del Apartado 7.2 sugiere que se deben *descomponer* los esquemas de relación que tienen muchos atributos en varios esquemas con menos atributos. Una descomposición poco cuidadosa, no obstante, puede llevar a otra modalidad de mal diseño.

Considérese un diseño alternativo en el que se descompone *Esquema-empréstito* en los dos esquemas siguientes:

Esquema-sucursal-cliente = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente*)
Esquema-cliente-préstamo = (*nombre-cliente, número-préstamo, importe*)

Empleando la relación *empréstito* de la Figura 7.1 se crean las relaciones nuevas *sucursal-cliente* (*Esquema-sucursal-cliente*) y *cliente-préstamo* (*Esquema-cliente-préstamo*):

$$\textit{sucursal-cliente} = \Pi_{\textit{nombre-sucursal, ciudad-sucursal, activo, nombre-cliente}} (\textit{empréstito})$$
$$\textit{cliente-préstamo} = \Pi_{\textit{nombre-cliente, número-préstamo, importe}} (\textit{empréstito})$$

Las Figuras 7.9 y 7.10, respectivamente, muestran las relaciones *sucursal-cliente* y *nombre-cliente* resultantes.

Por supuesto, hay casos en los que hace falta reconstruir la relación *préstamo*. Por ejemplo, supóngase que se desea hallar todas las sucursales que tienen préstamos con importes inferiores a 1000 €. Ninguna relación de la base de datos alternativa contiene esos datos. Hay que reconstruir la relación *empréstito*. Parece que se puede hacer escribiendo

$$\textit{sucursal-cliente} \bowtie \textit{cliente-préstamo}$$

La Figura 7.11 muestra el resultado de calcular *sucursal-cliente* $\bowtie$ *préstamo-cliente*. Cuando se compara esta relación con la relación *empréstito* con la cual comenzamos (Figura 7.1) se aprecia una diferencia: aunque cada tupla que aparece en *empréstito* aparece también en *sucursal-cliente* $\bowtie$ *préstamo-cliente*, hay tuplas de *sucursal-cliente* $\bowtie$ *préstamo-cliente* que no

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*
Centro	Arganzuela	9.000.000	Santos
Moralzarzal	La Granja	2.100.000	Gómez
Navacerrada	Aluche	1.700.000	López
Centro	Arganzuela	9.000.000	Sotoca
Becerril	Aluche	400.000	Santos
Collado Mediano	Aluche	8.000.000	Abril
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso
Segovia	Cerceda	3.700.000	López
Centro	Arganzuela	9.000.000	González
Navacerrada	Aluche	1.700.000	Rodríguez
Galapagar	Arganzuela	7.100.000	Amo

FIGURA 7.9. La relación *sucursal-cliente*.

nombre-cliente	*número-préstamo*	*importe*
Santos	P-17	1.000
Gómez	P-23	2.000
López	P-15	1.500
Sotoca	P-14	1.500
Santos	P-93	500
Abril	P-11	900
Valdivieso	P-29	1.200
López	P-16	1.300
González	P-18	2.000
Rodríguez	P-25	2.500
Amo	P-10	2.200

FIGURA 7.10. La relación *cliente-préstamo*.

nombre-sucursal	*ciudad-sucursal*	*activo*	*nombre-cliente*	*número-préstamo*	*importe*
Centro	Arganzuela	9.000.000	Santos	P-17	1.000
Centro	Arganzuela	9.000.000	Santos	P-93	500
Moralzarzal	La Granja	2.100.000	Gómez	P-23	2.000
Navacerrada	Aluche	1.700.000	López	P-15	1.500
Navacerrada	Aluche	1.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	Sotoca	P-14	1.500
Becerril	Aluche	400.000	Santos	P-17	1.000
Becerril	Aluche	400.000	Santos	P-93	500
Collado Mediano	Aluche	8.000.000	Abril	P-11	900
Navas de la Asunción	Alcalá de Henares	300.000	Valdivieso	P-29	1.200
Segovia	Cerceda	3.700.000	López	P-15	1.500
Segovia	Cerceda	3.700.000	López	P-16	1.300
Centro	Arganzuela	9.000.000	González	P-18	2.000
Navacerrada	Aluche	1.700.000	Rodríguez	P-25	2.500
Galapagar	Arganzuela	7.100.000	Amo	P-10	2.200

FIGURA 7.11. La relación *sucursal-cliente* ⋈ *cliente-préstamo*.

están en *empréstito*. En el ejemplo, *sucursal-cliente* ⋈ *préstamo-cliente* tiene las siguientes tuplas adicionales:

(Centro, Arganzuela, 9.000.000, Santos, P-93, 500)
(Navacerrada, Aluche, 1.700.000, López, P-16, 1.300)
(Becerril, Aluche, 400.000, Santos, P-17, 1.000)
(Segovia, Cerceda, 3.700.000, López, P-15, 1.500)

Considérese la consulta «Hallar todas las sucursales que han concedido un préstamo por un importe inferior a 1.500 €». Si se vuelve a la Figura 7.1 se observa que las únicas sucursales con créditos con importe inferior a 1.500 € son Becerril y Collado Mediano. Sin embargo, al aplicar la expresión

$$\Pi_{nombre\text{-}sucursal}\,(\sigma_{importe < 1.000}\,(sucursal\text{-}cliente \bowtie préstamo\text{-}cliente))$$

obtenemos los nombres de tres sucursales: Becerril, Collado Mediano y Centro.

Un examen más detenido de este ejemplo muestra el motivo. Si resulta que un cliente tiene varios préstamos de diferentes sucursales, no se puede decir el préstamo que pertenece a cada sucursal. Por lo tanto, al reunir *sucursal-cliente* y *cliente-préstamo*, no sólo se obtienen las tuplas que se tenía originariamente en *empréstito*, sino también varias tuplas adicionales. Aunque se tengan *más* tuplas en *sucursal-cliente* ⋈ *cliente-préstamo*, realmente se tiene *menos* información. Ya no es posible, en general, representar en la base de datos la información de los clientes que tienen concedidos préstamos en cada sucursal. Debido a esta pérdida de información se dice que la descomposición de *Esquema-empréstito* en *Esquema-sucursal-cliente* y *Esquema-cliente-préstamo* es una **descomposición con pérdida**, o una **descomposición de reunión con pérdida**. Una descomposición que no es una descomposición con pérdida es una **descomposición de reunión sin pérdida**. Queda claro con este ejemplo que una descomposición de reunión con pérdida supone, en general, un mal diseño de base de datos.

Es interesante averiguar el motivo por el que la descomposición es una descomposición con pérdida. Hay un atributo en común entre *Esquema-cliente-sucursal* y *Esquema-cliente-préstamo*:

Esquema-sucursal-cliente ∩ *Esquema-cliente-préstamo* = {*nombre-cliente*}

El único modo de que se pueda representarse una relación entre, por ejemplo, *número-préstamo* y *nombre-sucursal* es mediante *nombre-cliente*. Esta representación no resulta adecuada porque puede que un cliente tenga concedidos varios préstamos, pero esos préstamos no tienen que haberse obtenido necesariamente de la misma sucursal.

Considérese otro diseño alternativo en el que se descompone *Esquema-empréstito* en los dos esquemas siguientes:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

Hay un atributo en común entre estos dos esquemas:

Esquema-sucursal-préstamo ∩ *Esquema-cliente-préstamo* = {*nombre-sucursal*}

Por lo tanto, el único modo de poder representar una relación entre, por ejemplo, *nombre-cliente* y *activo* es

mediante *nombre-sucursal*. La diferencia entre este ejemplo y el anterior es que el activo de una sucursal es el mismo, independientemente del cliente al que se haga referencia, mientras que la sucursal prestamista *sí* depende del cliente al que se haga referencia. Para un valor dado de *nombre-sucursal* dado, hay exactamente un valor de *activo* y un valor de *ciudad-sucursal*, mientras que no se puede hacer una afirmación parecida para *nombre-cliente*. Es decir, se cumple la dependencia funcional:

$$\textit{nombre-sucursal} \rightarrow \textit{activo ciudad-sucursal}$$

pero *nombre-cliente* no determina funcionalmente a *número-préstamo*.

El concepto de reunión sin pérdida resulta fundamental para gran parte del diseño de bases de datos relacionales. Por lo tanto, se volverán a formular los ejemplos anteriores de manera más concisa y formal. Sea R un esquema de relación. Un conjunto de esquemas de relación $\{R_1, R_2, ..., R_n\}$ es una **descomposición** de R si

$$R = R_1 \cup R_2 \cup \ldots \cup R_n$$

Es decir, $\{R_1, R_2, ..., R_n\}$ es una descomposición de R si, para $i = 1, 2,..., n$, cada R_i es un subconjunto de R y cada atributo de R aparece en al menos un R_i.

Sea r una relación del esquema R y $r_i = \prod_{R_i}(r)$ para $i = 1, 2,..., n$. Es decir, $\{r_1, r_2,..., r_n\}$ es la base de datos que resulta de descomponer R en $\{R_1, R_2, ..., R_n\}$. Siempre se cumple que

$$r \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

Para comprobar que esta afirmación es cierta considérese una tupla t de la relación r. Cuando se calculan las relaciones $r_1, r_2, \ldots, r_n$, la tupla t da lugar a una tupla t_i en cada r_i, $i = 1, 2, \ldots, n$. Estas n tuplas se combinan para regenerar t cuando se calcula $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$. Los detalles se dejan para completarlos como ejercicio. Por tanto, cada tupla de r aparece en $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$.

En general, $r \neq r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$. Para mostrarlo consideremos el ejemplo anterior, en el que

- $n = 2$.
- $R =$ *Esquema-empréstito*.
- $R_1 =$ *Esquema-sucursal-cliente*.
- $R_2 =$ *Esquema-cliente-préstamo*
- $r =$ la relación mostrada en la Figura 7.1.
- $r_1 =$ la relación mostrada en la Figura 7.9.
- $r_2 =$ la relación mostrada en la Figura 7.10.
- $r_1 \bowtie r_2 =$ la relación mostrada en la Figura 7.11.

Obsérvese que las relaciones de las Figuras 7.1 y 7.11 no son iguales.

Para tener una descomposición de reunión sin pérdida hay que imponer restricciones en el conjunto de las relaciones posibles. Se descubrió que la descomposición de *Esquema-empréstito* en *Esquema-sucursal* y *Esquema-info-préstamo* era sin pérdida porque se cumple la dependencia funcional

$$\textit{nombre-sucursal} \rightarrow \textit{ciudad-sucursal activo}$$

en *Esquema-sucursal*.

Más adelante en este capítulo se introducirán otras restricciones distintas de las dependencias funcionales. Se dice que una relación es **legal** si satisface todas las reglas, o restricciones, que se hayan impuesto en la base de datos.

Sea C un conjunto de restricciones de la base de datos y R un esquema de relación. Una descomposición $\{R_1, R_2, \ldots, R_n\}$ de R es una **descomposición de reunión sin pérdida** si, para todas las relaciones r del esquema R que son legales bajo C

$$r = \prod_{R_1}(r) \bowtie \prod_{R_2}(r) \bowtie \ldots \bowtie \prod_{R_n}(r)$$

En los siguientes apartados se mostrará el modo de comprobar si una descomposición es una descomposición de reunión sin pérdida. La parte principal de este capítulo trata del problema de la especificación de restricciones para las bases de datos y del modo de obtener descomposiciones de reunión sin pérdida que eviten los inconvenientes representados en los ejemplos de malos diseños de bases de datos que se han visto en este apartado.

7.5. PROPIEDADES DESEABLES DE LA DESCOMPOSICIÓN

Se puede utilizar un conjunto dado de dependencias funcionales para diseñar una base de datos relacional en la que no se halle presente la mayor parte de las propiedades no deseables estudiadas en el Apartado 7.2. Cuando se diseñan estos sistemas puede hacerse necesaria la descomposición de una relación en varias relaciones de menor tamaño.

En este apartado se describen las propiedades deseables de las descomposiciones de los esquemas relacionales. En apartados posteriores se describen maneras concretas de descomponer un esquema relacional para obtener las propiedades deseadas. Estos conceptos se ilustran con el esquema *Esquema-empréstito* del Apartado 7.2:

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

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

nombre-sucursal → *ciudad-sucursal activo*
número-préstamo → *importe nombre-sucursal*

Como se estudió en el Apartado 7.2, *Esquema-empréstito* es un ejemplo de un mal diseño de base de datos. Supóngase que se descompone en las tres relaciones siguientes:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Puede afirmarse que esta descomposición tiene varias propiedades deseables que se estudiarán a continuación. Obsérvese que estos tres esquemas de relación son precisamente los que se utilizaron anteriormente en los capítulos 3 a 5.

7.5.1. Descomposición de reunión sin pérdida

En el Apartado 7.4 se arguyó que, al descomponer una relación en varias relaciones de menor tamaño, resulta fundamental que la descomposición sea una descomposición sin pérdida. Se puede afirmar que la descomposición del Apartado 7.5 es realmente una descomposición sin pérdida. Para demostrar esta afirmación antes hay que presentar un criterio para determinar si una descomposición es una descomposición con pérdida.

Sea R un esquema de relación, y sea F un conjunto de dependencias funcionales de R. R_1 y R_2 forman una descomposición de R. Esta descomposición es una descomposición de reunión sin pérdida de R si al menos una de las siguientes dependencias se halla en F^+:

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

En otras palabras, si $R_1 \cap R_2$ forma una superclave de R_1 o de R_2, la descomposición de R es una descomposición de reunión sin pérdida. Se puede utilizar el cierre de los atributos para comprobar de manera eficiente la existencia de superclaves, como ya se ha visto.

Ahora se demostrará que la descomposición de *Esquema-empréstito* es una descomposición de reunión sin pérdida mostrando una secuencia de pasos que generan la descomposición. Para empezar se descompone *Esquema-empréstito* en dos esquemas:

Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

Dado que *nombre-sucursal* → *ciudad-sucursal activo*, la regla de la aumentatividad para las dependencias funcionales (Apartado 7.3.2) implica que

nombre-sucursal → *nombre-sucursal ciudad-sucursal activo*

Como *Esquema-sucursal* ∩ *Esquema-info-préstamo* = {*nombre-sucursal*}, se concluye que la descomposición inicial es una descomposición de reunión sin pérdida.

A continuación se descompone *Esquema-info-préstamo* en

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Este paso da lugar a una descomposición de reunión sin pérdida ya que *número-préstamo* es un atributo común y *número-préstamo* → *importe nombre-sucursal*.

En el caso general de la descomposición simultánea de una relación en varias partes, la búsqueda de la descomposición de reunión sin pérdida resulta más complicada. Véanse las notas bibliográficas para encontrar referencias sobre este tema.

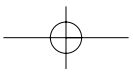
Aunque la prueba de la descomposición binaria es, evidentemente, una condición suficiente para la reunión sin pérdida, sólo constituye una condición necesaria si todas las restricciones son dependencias funcionales. Más adelante se verán otros tipos de restricciones (en especial, un tipo de restricción denominado dependencia multivalorada), que pueden asegurar que una descomposición es una reunión sin pérdida aunque no haya ninguna dependencia funcional.

7.5.2. Conservación de las dependencias

Hay otro objetivo en el diseño de las bases de datos relacionales: la *conservación de las dependencias*. Cuando se lleva a cabo una actualización de la base de datos el sistema debe poder comprobar que la actualización no crea ninguna relación ilegal, es decir, una relación que no satisface todas las dependencias funcionales dadas. Si hay que comprobar de manera eficiente las actualizaciones, se deben diseñar unos esquemas de bases de datos relacionales que permitan la validación de las actualizaciones sin que haga falta calcular las reuniones.

Para decidir si hay que calcular las reuniones para comprobar una actualización hace falta determinar las dependencias funcionales que hay que comprobar verificando cada relación una a una. Sea F un conjunto de dependencias funcionales del esquema R y $R_1, R_2,...,R_n$ 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 una restricción únicamente implican atributos de un esquema de relación, es posible comprobar el cumplimiento de la condición por una dependencia verificando sólo una relación.

Obsérvese que la definición de restricción utiliza todas las dependencias de F^+, no sólo las de F. Por ejem-



plo, 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, entonces, $A \rightarrow C$, ya que $A \rightarrow C$ se halla en F^+, aunque no se halle en *F*.

El conjunto de restricciones $F_1, F_2, ..., F_n$ es el conjunto de dependencias que pueden comprobarse de manera eficiente. Ahora cabe preguntarse si es suficiente comprobar sólo las restricciones. Sea $F' = F_1 \cup F_2 \cup ... \cup F_n$. 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 cada dependencia de *F* está lógicamente implicada por F' y, si se comprueba que se satisface F', se habrá comprobado que se satisface *F*. Se dice que las descomposiciones que tienen propiedad $F'^+ = F^+$ son **descomposiciones que conservan las dependencias**.

La Figura 7.12 muestra un algoritmo para la comprobación de la conservación de las dependencias. La entrada es un conjunto $E = \{R_1, R_2, ..., R_n\}$ de esquemas de relaciones descompuestas y un conjunto *F* de dependencias funcionales. Este algoritmo resulta costoso, ya que exige el cálculo de F^+; se describirá otro algoritmo que es más eficiente después de haber dado un ejemplo de comprobación de la conservación de las dependencias.

Ahora se puede demostrar que la descomposición de *Esquema-empréstito* conserva las dependencias. En lugar de aplicar el algoritmo de la Figura 7.12, se considera una alternativa más sencilla: se considera cada miembro del conjunto de dependencias funcionales *F* que se exige que se cumplan en *Esquema-empréstito* y se demuestra que cada una de ellas puede comprobarse, como mínimo, en una relación de la descomposición.

- Se puede comprobar la dependencia funcional: *nombre-sucursal* → *ciudad-sucursal activo* utilizando *Esquema-sucursal* = (*nombre-sucursal, ciudad-sucursal, activo*).
- Se puede comprobar la dependencia funcional: *número-préstamo* → *importe nombre-sucursal* utilizando *Esquema-préstamo* = (*nombre-sucursal, número-préstamo, importe*).

```
calcular F+;
for each esquema Ri de E 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 (true)
            else return (false);
```

FIGURA 7.12. Comprobación de la conservación de las dependencias.

Si puede comprobarse cada miembro de *F* en una de las relaciones de la descomposición, la descomposición conserva las dependencias. Sin embargo, hay casos en los que, aunque la descomposición conserve las dependencias, hay alguna dependencia de *F* que no puede comprobarse en ninguna relación de la descomposición. Se puede utilizar, por tanto, la prueba alternativa como condición suficiente que resulta sencilla de comprobar; si falla, no se puede concluir que la descomposición no conserve las dependencia; en lugar de eso, habrá que aplicar la prueba general.

Ahora se dará una prueba más eficiente para la conservación de las dependencias, que evita el cálculo de F^+. La idea es comprobar cada dependencia funcional $\alpha \rightarrow \beta$ de *F* empleando una forma modificada del cierre de los atributos para ver si la descomposición la conserva. Se aplica el siguiente procedimiento a cada $\alpha \rightarrow \beta$ de *F*.

```
resultado = α
while (cambios en resultado) do
    for each Ri de la descomposición
        t = (resultado ∩ Ri)+ ∩ Ri
        resultado = resultado ∪ t
```

El cierre de los atributos está tomado con respecto a las dependencias funcionales de *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 se conservan todas las dependencias de *F*.

Obsérvese que, en lugar de calcular previamente la restricción de *F* a R_i y utilizarla para el cálculo del cierre de los atributos de *resultado*, se usa el cierre de los atributos en $(resultado \cap R_i)$ con respecto a *F* y luego se intersecta con R_i para obtener un resultado equivalente. Este procedimiento tarda un tiempo polinómico, en lugar del tiempo exponencial necesario para calcular F^+.

7.5.3. Repetición de la información

La descomposición de *Esquema-empréstito* no sufre el problema de repetición de la información que se estudió en el Apartado 7.2. En *Esquema-empréstito* era necesario repetir la ciudad y el activo de la sucursal para cada préstamo. La descomposición separa los datos de la sucursal y los del préstamo en relaciones diferentes, con lo que elimina esta redundancia. De manera parecida, se observa que, en *Esquema-empréstito*, si se concede un solo préstamo a varios clientes, hay que repetir el importe del préstamo una vez por cada cliente (así como la ciudad y activo de la sucursal). En la descomposición, la relación del esquema *Esquema-prestatario* contiene la relación *número-préstamo, nombre-cliente*, y no la contiene ningún otro esquema. Por tanto, sólo

se tiene una tupla por préstamo para cada cliente en la relación de *Esquema-prestatario*. En las otras relaciones que implican a *número-préstamo* (las de los esquemas *Esquema-préstamo* y *Esquema-prestatario*) solamente hace falta que aparezca una tupla por préstamo.

Evidentemente, la falta de redundancia de la descomposición es algo deseable. El grado hasta el que se puede conseguir esta falta de redundancia viene representado por varias *formas normales*, que se estudiarán en el resto del capítulo.

7.6. FORMA NORMAL DE BOYCE–CODD

Mediante las dependencias funcionales se pueden definir varias *formas normales* que representan «buenos» diseños de bases de datos. En este apartado se tratará de la FNBC (forma normal de Boyce-Codd, que se define a continuación) y, más adelante, en el Apartado 7.7, se tratará de la 3FN (tercera forma normal).

7.6.1. Definición

Una de las formas normales mas deseables que se pueden obtener es la **forma normal de Boyce-Codd** (FNBC). Un esquema de relación R está en FNBC respecto a un conjunto de dependencias funcionales F 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 una superclave del esquema R.

Un diseño de base de datos está en FNBC si cada miembro del conjunto de esquemas de relación que constituye el diseño está en FNBC.

A modo de ejemplo, considérense los siguientes esquemas de relación y sus respectivas dependencias funcionales:

- *Esquema-cliente* = (*nombre-cliente, calle-cliente, ciudad-cliente*)
  nombre-cliente → *calle-cliente ciudad-cliente*
- *Esquema-sucursal* = (*nombre-sucursal, activo, ciudad-sucursal*)
  nombre-sucursal → *activo ciudad-sucursal*
- *Esquema-info-préstamo* = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)
  número-préstamo → *importe nombre-sucursal*

Puede afirmarse que *Esquema-cliente* está en FNBC. Obsérvese que una clave candidata para el esquema es *nombre-cliente*. Las únicas dependencias funcionales no triviales que se cumplen en *Esquema-cliente* tienen a *nombre-cliente* a la izquierda de la flecha. Dado que *nombre-cliente* es una clave candidata, las dependencias funcionales con *nombre-cliente* en la parte izquierda no violan la definición de FNBC. De manera parecida, se puede demostrar fácilmente que el esquema de relación *Esquema-sucursal* está en FNBC.

El esquema *Esquema-info-préstamo*, sin embargo, *no* está en FNBC. En primer lugar, obsérvese que *número-préstamo* no es una superclave de *Esquema-info-préstamo*, ya que *puede* que haya un par de tuplas que representen a un solo préstamo concedido a dos personas, por ejemplo,

(Centro, Sr. Pinilla, P-44, 1.000)
(Centro, Sra. Pinilla, P-44, 1.000)

Como no se ha relacionado ninguna dependencia funcional que descarte el caso anterior, *número-préstamo* no es una clave candidata. Sin embargo, la dependencia funcional *número-préstamo* → *importe* es de tipo no trivial. Por lo tanto, *Esquema-info-préstamo* no satisface la definición de FNBC.

Se puede afirmar que *Esquema-info-préstamo* no está en una forma normal adecuada, ya que sufre del problema de *repetición de información* que se describió en el Apartado 7.2. Se observa que, si hay varios nombres de clientes asociados a un préstamo, en una relación de *Esquema-info-préstamo* es obligatorio repetir el nombre de la sucursal y el importe una vez por cada cliente. Se puede eliminar esta redundancia rediseñando la base de datos de forma que todos los esquemas estén en FNBC. Una manera de abordar este problema es tomar el diseño que no está en FNBC ya existente como punto de partida y descomponer los esquemas que no estén en FNBC. Considérese la descomposición de *Esquema-info-préstamo* en dos esquemas:

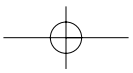
Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Esta descomposición es una descomposición de reunión sin pérdida.

Para determinar si esos esquemas están en FNBC es necesario determinar las dependencias funcionales que se les aplican. En este ejemplo resulta sencillo ver que

número-préstamo → *importe nombre-sucursal*

se aplica a *Esquema-préstamo*, y que sólo se aplican las dependencias funcionales triviales a *Esquema-presta-*



tario. Aunque *número-préstamo* no sea una superclave de *Esquema-info-préstamo*, es una clave candidata para *Esquema-préstamo*. Por tanto, los dos esquemas de la descomposición están en FNBC.

Ahora resulta posible evitar la redundancia en el caso en que haya varios clientes asociados a un mismo préstamo. En la relación de *Esquema-préstamo* hay exactamente una tupla para cada préstamo, y una tupla para cada cliente de cada préstamo en la relación de *Esquema-prestatario*. Por tanto, no hay que repetir el nombre de la sucursal y el importe una vez por cada cliente asociado a un préstamo.

A menudo se puede simplificar la comprobación de una relación para ver si satisface FNBC:

- Para comprobar si la dependencia no trivial $\alpha \rightarrow \beta$ provoca una violación de 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 una superclave de R.
- Para comprobar si el esquema de relación R se halla en FNBC basta con comprobar únicamente las dependencias del conjunto dado F en búsqueda de violaciones de FNBC, en lugar de comprobar todas las dependencias de F^+.

Se puede probar que si ninguna de las dependencias de F provoca una violación de FNBC, entonces ninguna de las dependencias de F^+ la provocará tampoco.

Por desgracia, el último procedimiento no funciona cuando una relación está descompuesta. Es decir, *no* basta con utilizar F al comprobar la relación R_i en la descomposición de R para buscar violaciones de FNBC. Por ejemplo, considérese el esquema de relación $R(A,B,C,D,E)$, con las dependencias funcionales F que contienen $A \rightarrow B$ y $BC \rightarrow D$. Supóngase que estuviera descompuesto en $R_1(A,B)$ y $R_2(A,C,D,E)$. Ahora bien, ninguna de las dependencias de F contiene únicamente atributos de (A,C,D,E), por lo que puede inducir a creer que R_2 satisface FNBC. En realidad, hay una dependencia $AC \rightarrow D$ de F^+ (que puede inferirse empleando la regla de la pseudotransitividad a partir de las dos dependencias de F) que demuestra que R_2 no está en FNBC. Por tanto, puede que se necesite alguna dependencia que esté en F^+, pero que no está en F, para demostrar que la relación descompuesta no está en FNBC.

La prueba alternativa de FNBC resulta a veces más sencilla que el cálculo de todas las dependencias de F^+. Para comprobar si la relación R_i de una descomposición de R está en FNBC hay que aplicar esta prueba:

- En cada subconjunto de atributos a de R_i hay que comprobar que α^+ (el cierre de los atributos de α bajo F) no incluye ningún atributo de $R_i - \alpha$ o que incluye todos los atributos de R_i.

Si algún conjunto de atributos a de R_i viola la condición, considérese la siguiente dependencia funcional, que se puede demostrar que se encuentra en F^+:

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

La dependencia anterior demuestra que R_i viola FNBC y es un «testigo» de esa violación. El algoritmo de descomposición de la FNBC, que se verá en el Apartado 7.6.2, hace uso de este testigo.

7.6.2. Algoritmo de descomposición

Ahora se puede exponer un método general para descomponer los esquemas de relación de manera que satisfagan FNBC. La Figura 7.13 muestra un algoritmo para esta tarea. Si R no está en FNBC se puede descomponer en un conjunto de esquemas en FNBC, $R_1, R_2, ..., R_n$ utilizando este algoritmo. El algoritmo utiliza las dependencias («testigos») que demuestran la violación de FNBC para llevar a cabo la descomposición.

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

Se aplicará el algoritmo de descomposición FNBC al esquema *Esquema-empréstito* que se empleó en el Apartado 7.2 como ejemplo de mal diseño de base de datos:

Esquema-empréstito = (*nombre-sucursal, ciudad-sucursal, activo, nombre-cliente, número-préstamo, importe*)

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

nombre-sucursal → *activo 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 7.13 al ejemplo *Esquema-empréstito* de la manera siguiente:

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

FIGURA 7.13. Algoritmo de descomposición de FNBC.

- La dependencia funcional

  nombre-sucursal → *activo ciudad-sucursal*

  se cumple en *Esquema-empréstito*, pero *nombre-sucursal* no es una superclave. Por tanto, *Esquema-empréstito* no está en FNBC. Se sustituye *Esquema-empréstito* por

  Esquema-sucursal = (*nombre-sucursal, ciudad-sucursal, activo*)
  Esquema-info-préstamo = (*nombre-sucursal, nombre-cliente, número-préstamo, importe*)

- Las únicas dependencias funcionales no triviales que se cumplen en *Esquema-sucursal* incluyen a *nombre-sucursal* a la izquierda de la flecha. Como *nombre-sucursal* es una clave de *Esquema-sucursal*, la relación *Esquema-sucursal* está en FNBC.

- La dependencia funcional

  número-préstamo → *importe nombre-sucursal*

  se cumple en *Esquema-info-préstamo*, pero *número-préstamo* no es una clave de *Esquema-info-préstamo*. Se sustituye *Esquema-info-préstamo* por

  Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)
  Esquema-prestatario = (*nombre-cliente, número-préstamo*)

- *Esquema-préstamo* y *Esquema-prestatario* están en FNBC.

Por tanto, la descomposición de *Esquema-empréstito* da lugar a tres esquemas de relación *Esquema-sucursal, Esquema-préstamo* y *Esquema-prestatario*, cada uno de los cuales está en FNBC. Estos esquemas de relación son los del Apartado 7.5, donde se demostró que la descomposición resultante es, a un tiempo, una descomposición de reunión sin pérdida y una descomposición que preserva las dependencias.

El algoritmo de descomposición FNBC tarda un tiempo exponencial en el tamaño del esquema inicial, ya que el algoritmo para la comprobación de si la relación de la descomposición satisface FNBC puede tardar un tiempo exponencial. Las notas bibliográficas proporcionan referencias de un algoritmo que puede calcular la descomposición FNBC en un tiempo polinómico. No obstante, el algoritmo puede «sobrenormalizar», es decir, descomponer una relación de manera innecesaria.

7.6.3. Conservación de las dependencias

No todas las descomposiciones FNBC conservan las dependencias. A modo de ejemplo, considérese el esquema de relación

Esquema-asesor = (*nombre-sucursal, nombre-cliente, nombre-asesor*)

que indica que el cliente tiene un «asesor personal» en una sucursal determinada. El conjunto F de dependencias funcionales que se exige que se cumpla en *Esquema-asesor* es

nombre-asesor → *nombre-sucursal*
nombre-sucursal nombre-cliente → *nombre-asesor*

Evidentemente, *Esquema-asesor* no está en FNBC, ya que *nombre-asesor* no es una superclave.

Si se aplica el algoritmo de la Figura 7.13 se obtiene la siguiente descomposición FNBC:

Esquema -asesor-sucursal = (*nombre-asesor, nombre-sucursal*)
Esquema -cliente-asesor = (*nombre-cliente, nombre-asesor*)

Los esquemas descompuestos sólo conservan *nombre-asesor* → *nombre-sucursal* (y las dependencias triviales) pero el cierre de {*nombre-asesor* → *nombre-sucursal*} no incluye *nombre-cliente nombre-sucursal* → *nombre-asesor*. La violación de esta dependencia no puede detectarse a menos que se calcule la reunión.

Para ver el motivo de que la descomposición de *Esquema-asesor* en los esquemas *Esquema—asesor-sucursal* y *Esquema-cliente-asesor* no conserva las dependencias se aplica el algoritmo de la Figura 7.12. Se descubre que las restricciones F_1 y F_2 de F para cada esquema son las siguientes:

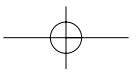
F_1 = { *nombre-asesor* → *nombre-sucursal* }
$F_2 = \varnothing$ (en *Esquema-asesor-cliente* solamente se cumplen las dependencias triviales)

(En aras de la brevedad no se muestran las dependencias funcionales triviales.) Resulta evidente que la dependencia *nombre-cliente nombre-sucursal* → *nombre-asesor* no está en $(F_1 \cup F_2)^+$ aunque *sí* está en F^+. Por tanto, $(F_1 \cup F_2)^+ \neq F^+$ y la descomposición no conserva las dependencias.

Este ejemplo demuestra que no todas las descomposiciones FNBC conservan las dependencias. Lo que es más, resulta evidente que *ninguna* descomposición FNBC de *Esquema-asesor* puede conservar *nombre-cliente nombre-sucursal* → *nombre-asesor*. Por tanto, el ejemplo demuestra que no se pueden cumplir siempre los tres objetivos del diseño:

1. Reunión sin pérdida
2. FNBC
3. Conservación de las dependencias

Recuérdese que la reunión sin pérdida es una condición esencial para la descomposición, para evitar la pérdida de información. Por tanto, es obligatorio abandonar la FNBC o la conservación de la dependencia. En el Apartado 7.7 se presenta una forma normal alterna-



tiva, denominada **tercera forma normal**, que es una pequeña relajación de la FNBC; el motivo del empleo de la tercera forma normal es que en ella siempre hay una descomposición que conserva las dependencias.

Hay situaciones en las que hay más de un modo de descomponer un esquema en su FNBC. Puede que algunas de estas descomposiciones conserven las dependencias, mientras que puede que otras no lo hagan. Por ejemplo, supóngase que se tiene un esquema de relación $R(A, B, C)$ con las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$. A partir de este conjunto se puede obtener la dependencia adicional $A \rightarrow C$. Si se utilizara la dependencia $A \rightarrow B$ (o, de manera equivalente, $A \rightarrow C$) para descomponer R, se acabaría con dos relaciones, $R_1(A, B)$ y $R_2(A, C)$; la dependencia $B \rightarrow C$ no se conservaría.

Si en lugar de eso se empleara la dependencia $B \rightarrow C$ para descomponer R, se acabaría con dos relaciones, $R_1(A, B)$ y $R_2(B, C)$, que están en FNBC, y la descomposición, además, conserva las dependencias. Evidentemente, resulta preferible la descomposición en $R_1(A, B)$ y $R_2(B, C)$. En general, por tanto, el diseñador de la base de datos debería examinar las descomposiciones alternativas y escoger una descomposición que conserve las dependencias siempre que resulte posible.

7.7. TERCERA FORMA NORMAL

Como ya se ha visto, hay esquemas relacionales en que la descomposición FNBC no puede conservar las dependencias. Para estos esquemas hay dos alternativas si se desea comprobar si una actualización viola alguna dependencia funcional:

- Soportar el coste extra del cálculo de las reuniones para buscar violaciones.
- Emplear una descomposición alternativa, la tercera forma normal (3FN), que se presenta a continuación, que hace menos costoso el examen de las actualizaciones. A diferencia de FNBC, las descomposiciones 3FN pueden contener cierta redundancia en el esquema descompuesto.

Se verá que siempre resulta posible hallar una descomposición de reunión sin pérdida que conserve las dependencias que esté en 3FN. La alternativa que se escoja es una decisión de diseño que debe adoptar el diseñador de la base de datos con base en los requisitos de la aplicación.

7.7.1. Definición

FNBC exige que todas las dependencias no triviales sean de la forma $\alpha \rightarrow \beta$ donde α es una superclave. 3FN relaja ligeramente esta restricción permitiendo dependencias funcionales no triviales cuya parte izquierda no sea una superclave.

Un esquema de relación R está en **tercera forma normal (3FN)** 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:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial.
- α es una 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 $\alpha \rightarrow \beta$; cada atributo A de $\alpha \rightarrow \beta$ puede estar contenido en una clave candidata *diferente*.

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

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

La definición de 3FN permite ciertas dependencias funcionales que no se permitían en FNBC. Una dependencia $\alpha \rightarrow \beta$ que sólo satisfaga la tercera alternativa de la definición de 3FN no se permitiría en FNBC, pero sí se permite en 3FN[1].

Volvamos al ejemplo *Esquema-asesor* (Apartado 7.6). Se demostró que este esquema de relación no tiene una descomposición FNBC de reunión sin pérdida que conserve las dependencias. Resulta, sin embargo, que este esquema está en 3FN. Para comprobarlo, obsérvese que {*nombre-cliente, nombre-sucursal*} es una cla-

[1] Estas dependencias son ejemplos de **dependencias transitivas** (véase el Ejercicio 7.25). La definición original de 3FN venía en términos de las dependencias transitivas. La definición que se ha empleado es equivalente, pero más sencilla de comprender.

ve candidata de *Esquema-asesor*, por lo que el único atributo no contenido en una clave candidata de *Esquema-asesor* es *nombre-asesor*. Las únicas dependencias funcionales no triviales de la forma

$$\alpha \rightarrow \textit{nombre-asesor}$$

incluyen {*nombre-cliente, nombre-sucursal*} como parte de α. Dado que {*nombre-cliente, nombre-sucursal*} es una clave candidata, estas dependencias no violan la definición de 3FN.

Como optimización, al buscar 3FN, se pueden considerar sólo las dependencias funcionales del conjunto dado F, en lugar de las de F^+. Además, se pueden descomponer las dependencias de F de manera que su lado derecho consista sólo en atributos sencillos y utilizar el conjunto resultante en lugar de F.

Dada la dependencia $\alpha \rightarrow \beta$, se puede utilizar la misma técnica basada en el cierre de los atributos que se empleó para FNBC para comprobar si α es una superclave. Si α no es una superclave, hay que comprobar si cada atributo de β está contenido en alguna clave candidata de R; esta comprobación resulta bastante más costosa, ya que implica buscar las claves candidatas. De hecho, se ha demostrado que la comprobación de la 3FN resulta NP-duro; por tanto, resulta bastante improbable que haya algún polinomio con complejidad polinómica en el tiempo para esta tarea.

7.7.2. Algoritmo de descomposición

La Figura 7.14 muestra un algoritmo para la búsqueda de descomposiciones de reunión sin pérdida que conserven las dependencias en 3FN. El conjunto de dependencias *Fc* 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 este caso el conjunto está vacío.

```
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 αβ
        then begin
                i := i +1;
                R_i := αβ;
             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 7.14. Descomposición de reunión sin pérdida que conserva las dependencias en 3FN.

Para ilustrar el algoritmo de la Figura 7.14 considérese la siguiente extensión de *Esquema-asesor* del Apartado 7.6:

Esquema-info-asesor = (*nombre-sucursal, nombre-cliente, nombre-asesor, número-sucursal*)

La diferencia principal es que se incluye el número de la sucursal del asesor como parte de la información. Las dependencias funcionales para este esquema de relación son:

nombre-asesor → *nombre-sucursal número-sucursal*
nombre-cliente nombre-sucursal → *nombre-asesor*

El bucle **for** del algoritmo hace que se incluyan los siguientes esquemas en la descomposición:

Esquema-asesor-sucursal = (*nombre-asesor, nombre-sucursal, número-sucursal*)
Esquema-asesor = (*nombre-cliente, nombre-sucursal, nombre-asesor*)

Como *Esquema-asesor* contiene una clave candidata de *Esquema-info-asesor*, el proceso de descomposición ha terminado.

El algoritmo asegura la conservación de las dependencias creando de manera explícita un esquema para cada dependencia del recubrimiento canónico. Asegura que la descomposición sea una descomposición de reunión sin pérdida garantizado que, como mínimo, un esquema contenga una clave candidata del esquema que está descomponiendo. El Ejercicio 7.19 proporciona algunos indicios de la prueba de que esto basta para garantizar una reunión sin pérdida.

Este algoritmo también se denomina **algoritmo de síntesis de 3FN**, ya que toma un conjunto de dependencias y añade los esquemas uno a 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 3FN. Recuérdese que, cuando se busca 3FN, basta con considerar las dependencias funcionales cuyo lado derecho sea un solo atributo. Por tanto, para ver si R_i está en 3FN, hay que convencerse de que cualquier dependencia funcional $\gamma \rightarrow B$ que se cumpla en R_i satisface la definición de 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 $\alpha \rightarrow \beta$ generó R_i. Considérense los tres casos posibles:

- B está tanto en α como en β. En este caso, la dependencia $\alpha \rightarrow \beta$ no habría estado en F_c, ya que B sería raro en β. Por tanto, este caso no puede darse.

- B está en β pero no en α. Considérense dos casos:
  - γ es una superclave. Se satisface la segunda condición de 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 podría haber empleado $\alpha \rightarrow \beta$ –para hacerlo, α debe estar contenido en el cierre de los atributos de γ, lo que no resulta posible, ya que se ha dado por supuesto que γ no es una superclave. Ahora bien, empleando $\alpha \rightarrow (\beta - \{B\})$ y $\gamma \rightarrow B$, se puede obtener $\alpha \rightarrow B$ (ya que $\gamma \subseteq \alpha\beta$ y que γ no puede contener a B porque $\gamma \rightarrow B$ es no trivial). Esto implicaría que B es raro en el lado derecho de $\alpha \rightarrow \beta$, lo que no resulta posible, ya que $\alpha \rightarrow \beta$ está en el recubrimiento canónico F_c. Por tanto, si B está en β, entonces γ debe ser una superclave, y se satisface la segunda condición de 3FN.
- B está en α pero no en β.
  Como $\alpha\beta$ es una clave candidata, se satisface la tercera alternativa de la definición de 3FN.

Resulta de interés que el algoritmo que se ha descrito para la descomposición en 3FN pueda implementarse en tiempo polinómico, aunque la comprobación de una relación dada para ver si satisface 3FN sea NP-duro.

7.7.3. Comparación entre FNBC y 3FN

De las dos formas normales para los esquemas de las bases de datos relacionales, 3FN y FNBC, hay ventajas en 3FN porque se sabe que siempre resulta posible obtener un diseño en 3FN sin sacrificar la reunión sin pérdida o la conservación de las dependencias. Sin embargo, hay inconvenientes en 3FN: si no se eliminan todas las dependencias transitivas de las relaciones de los esquemas, puede que se tengan que emplear valores nulos para representar algunas de las relaciones significativas posibles entre los datos, y está el problema de repetición de la información.

Como ilustración del problema de los valores nulos, considérese de nuevo *Esquema-asesor* y las dependencias funcionales asociadas. Dado que *nombre-asesor* → *nombre-sucursal*, puede que se desee representar las relaciones entre los valores de *nombre-asesor* y los valores de *nombre-sucursal* de la base de datos. No obstante, si se va a hacer eso, o bien debe existir el valor correspondiente de *nombre-cliente* o bien hay que utilizar un valor nulo para el atributo *nombre-cliente*.

Como ilustración del problema de repetición de información, considérese el ejemplo de *Esquema-asesor* de la Figura 7.15. Obsérvese que la información que indica que González trabaja en la sucursal Navacerrada está repetida.

nombre-cliente	*nombre-asesor*	*nombre-sucursal*
Santos	González	Navacerrada
Gómez	González	Navacerrada
López	González	Navacerrada
Sotoca	González	Navacerrada
Pérez	González	Navacerrada
Abril	González	Navacerrada

FIGURA 7.15. Ejemplo de *Esquema-asesor*.

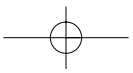
Recuérdese que los objetivos del diseño de bases de datos con dependencias funcionales son:

1. FNBC
2. Reunión sin pérdida
3. Conservación de las dependencias

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

Merece la pena destacar que SQL no proporciona 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**. Resulta posible, aunque un poco complicado, escribir afirmaciones que hagan que se cumpla una dependencia funcional (véase el Ejercicio 7.15); por desgracia, la comprobación de estas afirmaciones 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 SQL estándar, no se podrá comprobar de manera eficiente las dependencias funcionales cuyo lado izquierdo no 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 el coste empleando las vistas materializadas, que la mayor parte de los sistemas de bases de datos soporta. Dada una descomposición 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 estas 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 **única** (α). En el lado negativo hay que contar una sobrecarga espacial y temporal debida a la vista materializada pero, en el lado positivo, el programador de la aplicación no tiene que preocuparse por la escritura de 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 avanzado el libro, en el Apartado 14.5, se describe el modo en que el sis-



tema 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 FNBC que conserve las dependencias, suele resultar preferible optar por FNBC y utilizar técnicas como las vistas materializadas para reducir el coste de la comprobación de las dependencias funcionales.

7.8. CUARTA FORMA NORMAL

No parece que algunos esquemas de relación, aunque se hallen en FNBC, estén lo bastante normalizados, en el sentido de que siguen sufriendo el problema de la repetición de la información.

Considérese nuevamente el ejemplo bancario. Supóngase que, en un diseño alternativo del esquema de la base de datos, se tiene el esquema

Esquema-BC = (*número-préstamo, nombre-cliente, calle-cliente, ciudad-cliente*)

El lector sagaz reconocerá este esquema como un esquema que no está en FNBC debido a la dependencia funcional

nombre-cliente → *calle-cliente ciudad-cliente*

que se estableció anteriormente, y debido a que *nombre-cliente* no es una clave de *Esquema-BC*. Sin embargo, supóngase que el banco está atrayendo a clientes ricos que tienen varios domicilios (por ejemplo, una residencia de invierno y otra de verano). Entonces ya no se deseará hacer que se cumpla la dependencia funcional *nombre-cliente* → *calle-cliente ciudad-cliente*. Si se elimina esta dependencia funcional, se halla que *Esquema-BC* está en FNBC con respecto al conjunto modificado de dependencias funcionales. Sin embargo, aunque *Esquema-BC* esté ahora en FNBC, sigue existiendo el problema de la repetición de la información que se tenía anteriormente.

Para tratar este problema hay que definir una nueva forma de restricción, denominada *dependencia multivalorada*. Como se hizo para 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 FNBC. Se verá que cada esquema 4FN se halla también en FNBC, pero que hay esquemas FNBC que no se hallan en 4FN.

7.8.1. Dependencias multivaloradas

Las dependencias funcionales impiden que ciertas tuplas estén en una relación. Si $A \rightarrow B$, entonces no puede haber dos tuplas con el mismo valor de A y diferentes valores de B. Las dependencias multivaloradas, por otro lado, no impiden la existencia de esas tuplas. En lugar de eso, *exigen* que estén presentes en la relación otras tuplas de una cierta forma. Por este motivo, las dependencias funcionales se denominan a veces **dependencias de generación de igualdad** y las dependencias multivaloradas se conocen como **dependencias de generación de 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 toda 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

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

Esta definición es menos complicada de lo que parece. La Figura 7.16 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* en el 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 *Esquema-BC* y la relación *bc* (*Esquema-BC*) de la Figura 7.17. Hay que repetir el número de préstamo una vez por cada dirección que tenga un cliente, y la dirección en cada préstamo que tenga el cliente. Esta repetición es innecesaria, ya que la relación entre el cliente y su dirección es independiente de la relación entre ese cliente y el préstamo. Si un cliente (por ejemplo, Gómez) tiene un préstamo (por ejemplo, el préstamo número P-23), se desea que ese préstamo esté asociado con todas las direcciones que tenga Gómez. Por tanto, la relación

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

FIGURA 7.16. Representación tabular de $\alpha \twoheadrightarrow \beta$.

número-préstamo	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
P-23	Gómez	Carretas	Cerceda
P-23	Gómez	Mayor	Chinchón
P-93	Pérez	Leganitos	Aluche

FIGURA 7.17. Relación *bc*: ejemplo de redundancia en una relación FNBC.

de la Figura 7.18 es ilegal. Para hacer que esta relación sea legal hay que añadir las tuplas (P-23, Gómez, Mayor, Chinchón) y (P-27, Gómez, Carretas, Cerceda) a la relación *bc* de la Figura 7.18.

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

$$\textit{nombre-cliente} \twoheadrightarrow \textit{calle-cliente ciudad-cliente}$$

(La dependencia multivalorada *nombre-cliente* $\twoheadrightarrow$ *número-préstamo* también se cumplirá. Pronto se verá que son equivalentes.)

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

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.

Obsérvese que, si una relación r no satisface una dependencia multivalorada dada, se puede crear una relación r' que *sí* satisfaga esa dependencia multivalorada añadiendo tuplas a r.

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

número-préstamo	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
P-23	Gómez	Carretas	Cerceda
P-27	Gómez	Mayor	Chinchón

FIGURA 7.18. Una relación *bc* ilegal.

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, cada dependencia funcional es también una dependencia multivalorada.

7.8.2. Definición de la cuarta forma normal

Considérese nuevamente el ejemplo del *Esquema-BC* en el que se cumple la dependencia multivalorada *nombre-cliente* $\twoheadrightarrow$ *calle-cliente ciudad-cliente*, pero no se cumple ninguna dependencia funcional no trivial. Se vio en los primeros párrafos del Apartado 7.8 que, aunque *Esquema-BC* se halla en FNBC, el diseño no es el ideal, ya que hay que repetir la información de la dirección del cliente para cada préstamo. Se verá que se puede utilizar la dependencia multivalorada dada para mejorar el diseño de la base de datos descomponiendo *Esquema-BC* en una descomposición en la **cuarta forma normal**.

Un esquema de relación R está en la **cuarta forma normal** (4FN) con respecto a un conjunto F de dependencias funcionales y multivaloradas si, para todas las dependencias multivaloradas de F^+ 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 una superclave del esquema R.

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

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

Sea R un algoritmo de descomposición y sea R_1, R_2, . . . , R_n una descomposición de R. Para comprobar si cada esquema de relación R_i se halla en 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 F de dependencias funcionales y multivaloradas. La **restricción** de F a R_i es el conjunto F_i que consta de

1. Todas las dependencias funcionales de F^+ que sólo incluyen atributos de R_i
2. Todas las dependencias multivaloradas de la forma
$$\alpha \twoheadrightarrow \beta \cap R_i$$
donde $\alpha \subseteq R_i$ y $\alpha \twoheadrightarrow \beta$ está en F^+.

7.8.3. Algoritmo de descomposición

La analogía entre 4FN y FNBC es aplicable al algoritmo para la descomposición de los esquemas en 4FN. La Figura 7.19 muestra el algoritmo de descomposición en 4FN. Es idéntico al algoritmo de descomposición en FNBC de la Figura 7.13, excepto en que emplea dependencias multivaloradas en lugar de funcionales y en que utiliza la restricción de F^+ a R_i.

Si se aplica el algoritmo de la Figura 7.19 a *Esquema-BC* se halla que *nombre-cliente* $\twoheadrightarrow$ *número-préstamo* es una dependencia multivalorada no trivial, y que *nombre-cliente* no es una superclave de *Esquema-BC*. Siguiendo el algoritmo, se sustituye *Esquema-BC* por dos esquemas:

Esquema-prestatario = (*nombre-cliente*, *número-préstamo*)
Esquema-cliente = (*nombre-cliente, calle-cliente, ciudad-cliente*).

Este par de esquemas, que se halla en 4FN, elimina el problema que se encontró anteriormente con la redundancia de *Esquema-BC*.

Como ocurría cuando se trataba solamente con las dependencias funcionales, resultan interesantes las descomposiciones que son descomposiciones de reunión sin pérdida y que conservan las dependencias. El hecho siguiente relativo a las dependencias multivaloradas y las reuniones sin pérdida muestra que el algoritmo de la Figura 7.19 sólo genera descomposiciones de reunión sin pérdida:

```
resultado := {R};
hecho := falso;
calcular F+; Dado el esquema Ri, supongamos que Fi denota la
restricción de F+ a Ri
while (not hecho) do
    if (hay un esquema Ri en resultado que no se halla en 4FN
con respecto a Fi)
        then begin
            supongamos que α ↠ β es una dependencia
            multivalorada no trivial que se cumple en Ri tal
            que α → Ri no se halla en Fi, y α ∩ β = ∅;
            resultado := (resultado – Ri) ∪ (Ri – β) ∪ (α, β);
        end
    else hecho := verdadero;
```

FIGURA 7.19. Alogaritmo de descomposición en 4FN.

- Sean R un esquema de relación y F un conjunto de dependencias funcionales y multivaloradas de R. Supongamos que R_1 y R_2 forman una descomposición de R. Esta descomposición será una descomposición de reunión sin pérdida de R si y sólo si, como mínimo, una de las siguientes dependencias multivaloradas se halla en F^+:
$$R_1 \cap R_2 \twoheadrightarrow R_1$$
$$R_1 \cap R_2 \twoheadrightarrow R_2$$

Recuérdese que se afirmó en el Apartado 7.5.1 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 una descomposición de reunión sin pérdida de R. El hecho anterior sobre las dependencias multivaloradas es una afirmación más general sobre las reuniones sin pérdida. Dice que, para *toda* descomposición de reunión sin pérdida de R 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 la dependencia al descomponer una relación se vuelve más complejo en presencia de dependencias multivaloradas. El Apartado C.1.2 del apéndice aborda este tema.

7.9. OTRAS 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 tipos de 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, la FNRP y la forma normal de dominios y claves se utilizan muy raramente. El Apéndice C ofrece más detalles sobre estas formas normales.

Conspicua por su ausencia de este estudio de las formas normales es 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 7.26.

7.10. PROCESO GENERAL DEL DISEÑO DE BASES DE DATOS

Hasta ahora se han examinado problemas concretos de las formas normales y de la normalización. En este apartado se estudiará el modo en que se encaja la normalización en proceso general de diseño de bases de datos.

Anteriormente, en este capítulo, a partir del Apartado 7.4, se dio por supuesto que se da un esquema de relación R y que se procede a normalizarlo. Hay varios modos de obtener el esquema R:

1. R puede haberse generado al convertir un diagrama E-R en un conjunto de tablas.
2. R puede haber sido una sola relación que contuviera *todos* los atributos que resultan 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 relaciones, que hay que comprobar para verificar que satisface la forma normal deseada.

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

7.10.1. El modelo E-R y la normalización

Cuando se define con cuidado un diagrama E-R, identificando correctamente todas las entidades, las tablas generadas a partir del diagrama E-R no necesitan más normalización. No obstante, puede haber dependencias funcionales entre los atributos de una entidad. Por ejemplo, supóngase que una 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 estas dependencias surgen de un mal diseño del diagrama E-R. En el ejemplo anterior, si se hiciera 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 una 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 el 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.

7.10.2. El enfoque de la relación universal

El segundo enfoque del diseño de bases de datos es comenzar con un solo esquema de relación que contenga todos los atributos de interés y descomponerlo. Uno de los objetivos al escoger una descomposición era que fuera una descomposición de reunión sin pérdida. Para considerar la carencia de pérdida se dio por supuesto que resulta válido hablar de la reunión de todas las relaciones de la base de datos descompuesta.

Considérese la base de datos de la Figura 7.20, que muestra una descomposición de la relación *info-préstamo*. La figura muestra una situación en la que no se ha determinado todavía el importe del préstamo P-58, pero se desea registrar el resto de los datos del préstamo. Si se calcula la reunión natural de estas relaciones, se descubre que todas las tuplas que hacen referencia al préstamo P-58 desaparecen. En otras palabras, no hay relación *info-préstamo* correspondiente a las relaciones de la Figura 7.20. Las tuplas que desaparecen cuando se calcula la reunión son *tuplas colgantes* (véase el Apartado 6.2.1). Formalmente, sea $r_1(R_1)$, $r_2(R_2)$, . . . , $r_n(R_n)$ un conjunto de relaciones. Una tupla t de la relación r_i es una tupla colgante si t no está en la relación

$$\Pi_{R_i} (r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n)$$

Las tuplas colgantes pueden aparecer en las aplicaciones prácticas de las bases de datos. Representan información incompleta, como en el ejemplo, en que se desea almacenar datos sobre un préstamo que todavía se halla en proceso de negociación. La relación $r_1 \bowtie r_2 \bowtie \cdots \bowtie r_n$ se denomina **relación universal**, ya que implica a todos los atributos del universo definido por $R_1 \cup R_2 \cup \cdots \cup R_n$.

nombre-sucursal	*número-préstamo*
Collado Mediano	P-58

número- préstamo	*importe*

número-préstamo	*nombre-cliente*
P-58	González

FIGURA 7.20. Descomposición de *info-préstamo*.

La única manera de escribir una relación universal para el ejemplo de la Figura 7.20 es incluir los *valores nulos* en la relación universal. Se vio en el Capítulo 3 que los valores nulos presentan varias dificultades. Debido a ello, puede que sea mejor ver las relaciones del diseño descompuesto como representación de la base de datos, en lugar de como la relación universal cuyo esquema se descompuso durante el proceso de normalización. (Las notas bibliográficas discuten la investigación sobre los valores nulos y las relaciones universales.)

Téngase en cuenta que no se puede introducir en la base de datos de la Figura 7.20 toda la información incompleta sin recurrir a los valores nulos. Por ejemplo, no se puede introducir un número de préstamo a menos que se conozca, como mínimo, uno de los datos siguientes:

- El nombre del cliente
- El nombre de la sucursal
- El importe del préstamo

Por tanto, cada descomposición concreta define una forma restringida de información incompleta que es aceptable en la base de datos.

Las formas normales que se han definido generan buenos diseños de bases de datos desde el punto de vista de la representación de la información incompleta. Volviendo una vez más al ejemplo de la Figura 7.20, no sería deseable permitir el almacenamiento del hecho siguiente: «Hay un préstamo (cuyo número se desconoce) para Santos con un importe de 100 €». Esto se debe a que

$$\textit{número-préstamo} \rightarrow \textit{nombre-cliente importe}$$

y, por tanto, la única manera de relacionar *nombre-cliente* e *importe* es mediante *número-préstamo*. Si no se conoce el número del préstamo, no se puede distinguir este préstamo de otros préstamos con números desconocidos.

En otras palabras, no se desea almacenar datos de los cuales se desconocen los atributos claves. Obsérvese que las formas normales que se han definido no nos permiten almacenar ese tipo de información a menos que se utilicen los valores nulos. Por tanto, las formas normales permiten la representación de la información incompleta aceptable mediante las tuplas colgantes, mientras que prohíben el almacenamiento de la información incompleta indeseable.

Otra consecuencia del enfoque de la relación universal del diseño de bases de datos es que los nombres de los atributos deben ser únicos en la relación universal. No se puede utilizar *nombre* para hacer referencia tanto a *nombre-cliente* como a *nombre-sucursal*. Generalmente resulta preferible utilizar nombres únicos, como se ha hecho aquí. No obstante, si se definen de manera directa los esquemas de las relaciones, en vez de en términos de una relación universal, se pueden obtener relaciones en esquemas como los siguientes para el ejemplo del banco:

sucursal-préstamo (*nombre, número*)
préstamo-cliente (*número, nombre*)
importe (*número, importe*)

Obsérvese que, con las relaciones anteriores, expresiones como *sucursal-préstamo* $\bowtie$ *préstamo-cliente* carecen de sentido. En realidad, la expresión *sucursal-préstamo* $\bowtie$ *préstamo-cliente* halla los préstamos concedidos por las sucursales a los clientes que tienen el mismo nombre que la sucursal.

En un lenguaje como SQL, sin embargo, una consulta que implique a *sucursal-préstamo* y a *préstamo-cliente* debe eliminar la ambigüedad en las referencias a *nombre* anteponiendo el nombre de la relación. En estos entornos los diferentes papeles de *nombre* (como nombre de la sucursal y del cliente) resultan menos problemáticos y puede que sean más sencillos de utilizar.

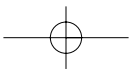
Es opinión de los autores de este libro que la **suposición de un único papel** –que cada nombre de atributo tenga un significado único en la base de datos– suele resultar preferible a la reutilización del mismo nombre en varios papeles. Cuando no se realiza la suposición, el diseñador de la base de datos debe ser especialmente cuidadoso al crear un diseño normalizado de una base de datos relacional.

7.10.3. Desnormalización para el rendimiento

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

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

Una alternativa para calcular 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. No obstante, la información del saldo de la cuenta se repite para cada persona titular de la cuenta, y la aplicación debe actualizar todas las copias, siempre que se actualice el saldo de la cuenta. El proceso de tomar un esquema normalizado y hacerlo no normalizado se denomina **desnormalización**, y los diseñadores lo utilizan para ajustar el rendimiento de los sistemas para dar soporte a las operaciones críticas en el tiempo.



Una alternativa mejor, soportada hoy en día por muchos sistemas de bases de datos, es emplear el esquema normalizado y, de manera adicional, almacenar la reunión o *cuenta* e *impositor* en forma de vista materializada. (Recuérdese que una vista materializada es una vista cuyo resultado se almacena en la base de datos y se actualiza cuando se actualizan las relaciones utilizadas en la vista.) 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 la vista actualizada es labor del sistema de bases de datos, no del programador de la aplicación.

7.10.4. Otros problemas de 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. A continuación se ofrecerán algunos ejemplos; evidentemente, conviene evitar esos diseños.

Considérese la base de datos de una empresa, donde se desea almacenar los beneficios de las compañías de varios años. Se puede utilizar la relación *beneficios*(*id-empresa, año, importe*) para almacenar la información de los beneficios. La única dependencia funcional de esta relación es *id-empresa año* → *importe*, y esta relación se halla en 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 de interés son 2000, 2001 y 2002; se tendrán, entonces, las relaciones de la forma *beneficios-2000*, *beneficios-2001*, *beneficios-2002*, todos los cuales se hallan en el esquema (*id-empresa, beneficios*). Aquí, la única dependencia funcional de cada relación será *id-empresa* → *beneficios*, por lo que estas relaciones también se hallan en FNBC.

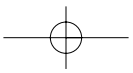
No obstante, este diseño alternativo es, claramente, una mala idea: habría que crear una relación nueva cada año, y también habría que escribir consultas nuevas cada año, para tener en cuenta cada nueva relación. Las consultas también tendrían que ser más complicadas, ya que puede que tengan que hacer referencia a muchas relaciones.

Otra manera más de representar los mismos datos es tener una sola relación *empresa-año*(*id-empresa, beneficios-2000, beneficios-2001, beneficios-2002*). En este caso, las únicas dependencias funcionales van de *id-empresa* hacia los demás atributos, y la relación vuelve a estar en FNBC. Este diseño también es una mala idea, ya que tiene problemas parecidos al 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 tengan que hacer referencia a muchos atributos.

Las representaciones como las de la compañía *empresa-año*, con una columna para cada valor de un atributo, se denominan **de tablas cruzadas**; se emplean ampliamente en las hojas de cálculo, en los informes y en las herramientas de análisis de datos. Aunque estas representaciones resultan útiles para mostrárselas a los usuarios, por las razones que acaban de darse, no resultan deseables en el diseño de bases de datos. Se han propuesto extensiones de SQL para convertir los datos desde una representación relacional normal en una tabla cruzada, para poder mostrarlos.

7.11. 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 introducido el concepto de las dependencias funcionales y se ha mostrado el modo de razonar con ellas. Se ha puesto un énfasis especial en que las dependencias están implicadas lógicamente por un conjunto 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 introducido el concepto de descomposición y se ha mostrado que las descomposiciones deben ser descomposiciones de reunión sin pérdida y que, preferiblemente, deben conservar las dependencias.
- Si la descomposición conserva las dependencias, dada una actualización de la base de datos, todas las dependencias funcionales pueden verificarse a partir de las relaciones individuales, sin calcular la reunión de las relaciones en la descomposición.
- Luego se ha presentado la Forma normal de Boyce–Codd (FNBC); las relaciones en FNBC están libres de las dificultades ya descritas. Se ha descrito un algoritmo para la descomposición de las relaciones en FNBC. Hay relaciones para las que no hay ninguna descomposición FNBC que conserve las dependencias.
- Se han utilizado los recubrimientos canónicos para descomponer una relación en 3FN, que es una pequeña relajación de la condición FNBC. Puede que las relaciones en 3FN tengan alguna redundancia, pero siempre hay una descomposición en 3FN que conserve las dependencias.



- Se ha presentado el concepto de las 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 FNRP y FNDC, eliminan formas más sutiles de redundancia. No obstante, 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 de datos relacionales 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

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

EJERCICIOS

7.1. 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 bases de datos relacionales.

7.2. 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 de reunión sin pérdida si se cumple el siguiente conjunto F de dependencias funcionales:

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

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

7.4. Indíquense todas las dependencias funcionales que satisface la relación de la Figura 7.21.

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 7.21. La relación del Ejercicio 7.4.

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

7.6. Explíquese el modo en que las dependencias funcionales pueden utilizarse para indicar lo siguiente:

- 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*.

7.7. 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 segura mostrando una relación r que satisfaga $\alpha \rightarrow \beta$ y $\gamma \rightarrow \beta$, pero no satisfaga $\alpha \rightarrow \gamma$.

7.8. Utilícense los axiomas de Armstrong para probar la seguridad 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 aumentatividad, utilizando $\alpha \rightarrow \gamma$, y aplíquese luego la regla de transitividad.)

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

7.10. Utilícense los axiomas de Armstrong para probar la corrección de la regla de la pseudotransitividad.

7.11. 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.

7.12. Utilizando las dependencias funcionales del Ejercicio 7.11, calcúlese B^+.

7.13. Utilizando las dependencias funcionales del Ejercicio 7.11, calcúlese el recubrimiento canónico F_c.

7.14. Considérese el algoritmo de la Figura 7.22 para calcular α^+. Demuéstrese que este algoritmo resulta más eficiente que el presentado en la Figura 7.7 (Apartado 7.3.3) y que calcula α^+ de manera correcta.

7.15. 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 una declaración SQL que haga que se cumpla la dependencia funcional. Supóngase que no hay ningún valor nulo.

7.16. Demuéstrese que la siguiente descomposición del esquema R del Ejercicio 7.2 no es una descomposición de reunión sin pérdida:

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

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

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

7.17. 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

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

7.18. Demuéstrese que la descomposición del Ejercicio 7.2 no es una descomposición que conserve las dependencias.

7.19. Demuéstrese que es posible que una descomposición que conserve las dependencias en 3FN sea una descomposición de reunión sin pérdida 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.)

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

7.21. Dese una descomposición de reunión sin pérdida en FNBC del esquema R del Ejercicio 7.2.

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

7.23. Al diseñar una base de datos relacional, indíquese el motivo de que se pueda escoger un diseño que no sea FNBC.

7.24. Dese una descomposición en 3FN de reunión sin pérdida que conserve las dependencias del esquema R del Ejercicio 7.2.

7.25. Sea un atributo *primo* uno 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 cumple $\beta \rightarrow \alpha$. Se dice que A es *dependiente de manera transitiva* de α. Se puede reformular la definición de 3FN de la manera siguiente: un esquema de relación R está en la 3FN con respecto a un 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.

7.26. Una dependencia funcional $\alpha \rightarrow \beta$ se denomina **dependencia parcial** si hay un subconjunto adecuado γ de α tal que $\gamma \rightarrow \beta$. Se dice que β es *parcialmente dependiente* de α. Un 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.
- No es parcialmente dependiente de una clave candidata.

Demuéstrese que cada esquema 3FN se halla en 2FN. (*Sugerencia*: demuéstrese que cada dependencia parcial es una dependencia transitiva.)

7.27. 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 2FN, pero que no se halle en una forma normal de orden

```
resultado := 2
/* cuentadf es un array cuyo elemento iésimo contiene el número de atributos del lado izquierdo de la iésima DF que todavía no
   se sabe que estén en α+ */
for i := 1 to |F| do
    begin
        Supongamos 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 sado izquierdo de la i-ésima DF */
for each atributo A do
    begin
        aparece [A] := NIL;
        for i := 1 to |F| do
            begin
                Supongamos que β → γ denota la iésima DF;
                if A ∈ β then añadir i a aparece [A];
            end
    end
agregar (α);
return (resultado);

procedure agregar (α);
for each atributo A de α do
    begin
        if A ∉ resultado then
            begin
                resultado := resultado ∪ {A};
                for each elemento i de aparece[A] do
                    begin
                        cuentadf [i] :=cuentadf [i] – 1;
                        if cuentadf [i] := 0 then
                            begin
                                supongamos que β → γ denota la i-ésima DF;
                                agregar (γ);
                            end
                    end
            end
    end
```

FIGURA 7.22. Un algoritmo para calcular α^+.

superior. (Véase el Ejercicio 7.26 para obtener la definición de 2FN.)

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

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

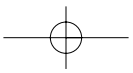
7.30. Explíquese el modo en que pueden aparecer las tuplas colgantes. Explíquense los problemas que pueden provocar.

NOTAS BIBLIOGRÁFICAS

El primer estudio de la teoría del diseño de bases de datos relacionales apareció en un documento pionero de Codd [1970]. En ese documento 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]. Ullman [1988] es una fuente fácilmente accesible de las pruebas de seguridad y de completitud de los axiomas de Armstrong. Ullman [1988] ofrece también un algoritmo para la comprobación de la descomposición de reunión sin pérdida para las descomposiciones generales (no binarias), y muchos otros algoritmos, teoremas y demostraciones relativos a la teoría de las dependencias. Maier [1983] estudia la teoría de las dependencias funcionales. Graham et al. [1986] estudia los aspectos formales del concepto de relación legal.

FNBC se introdujo en Codd [1972]. Las ventajas de FNBC se estudian en Bernstein et al. [1980a]. En Tsou y Fischer [1982] aparece un algoritmo polinómico en el tiempo para la descomposición en FNBC, y también



puede hallarse en Ullman [1988]. Biskup et al. [1979] ofrecen el algoritmo que se ha utilizado aquí para buscar una descomposición en 3FN de reunión sin pérdida que conserve las dependencias. Los resultados fundamentales de la propiedad de reunión sin pérdida aparecen en Aho et al. [1979a].

Las dependencias multivaloradas se estudian en Zaniolo [1976]. Beeri et al. [1977] dan un conjunto de axiomas para las dependencias multivaloradas y demuestran que los axiomas de los autores son seguros y completos. La axiomatización de este libro se basa en la suya. Los conceptos de 4FN, FNRP y FNDC proceden de Fagin [1977], Fagin [1979] y Fagin [1981], respectivamente.

Maier [1983] presenta con detalle la teoría del diseño de bases de datos relacionales. Ullman [1988] y Abiteboul et al. [1995] presentan un tratamiento más teórico de muchas de las dependencias y formas normales aquí presentadas. Véanse las notas bibliográficas del Apéndice C para obtener más referencias de literatura sobre normalización.