PARTE 3

Teoría y metodología del diseño de bases de datos

CAPÍTULO **10**

Dependencias funcionales y normalización en bases de datos relacionales

En los Capítulos del 5 al 9, mostramos varios aspectos del modelo relacional y los lenguajes asociados a él. Cada *esquema de relación* consta de un número de atributos, mientras que un *esquema de base de datos relacional* está compuesto por un número de esquemas de relación. Hasta ahora sólo hemos utilizado el sentido común del diseñador de la base de datos para agrupar los atributos y formar así un esquema de relación, o bien hemos utilizado un diseño de esquema de base de datos a partir de un modelo de datos conceptual como ER o EER, o algún otro. Estos modelos hacen que el diseñador identifique los tipos de entidad y de relación y sus respectivos atributos, lo que nos lleva a un agrupamiento natural y lógico de los atributos en relaciones cuando van seguidos por los procedimientos de mapeado del Capítulo 7. Sin embargo, aún necesitamos algún tipo de medida formal que nos indique por qué un agrupamiento de atributos en el esquema de una relación puede ser mejor que otro. Hasta el momento, en nuestro debate sobre el diseño conceptual de los Capítulos 3 y 4 y su asignación en el modelo relacional del Capítulo 7, no hemos desarrollado ningún método que nos indique la idoneidad de la calidad del diseño, aparte de la intuición del diseñador. En este capítulo vamos a ver parte de la teoría desarrollada con el objetivo de evaluar esquemas relacionales encaminados a la calidad del diseño; es decir, mediremos formalmente por qué un conjunto de agrupaciones de atributos en un esquema de relación es mejor que otro.

Hay dos niveles a los que podemos explicar la *bondad* de los esquemas de relación. El primero es el **nivel lógico** (o **conceptual**): cómo los usuarios interpretan los esquemas de relación y el significado de sus atributos. Disponer de un buen esquema de relación a este nivel permite a los usuarios comprender con claridad el significado de los datos en la relaciones y, por consiguiente, formular sus consultas correctamente. El segundo nivel es el de **implementación** (o **almacenamiento**): de qué modo se almacenan y actualizan las tuplas en una relación base. Este nivel se aplica sólo a esquemas de relación base (cómo se almacenarán físicamente los ficheros), mientras que a nivel lógico nos interesan tanto las relaciones base como las vistas (relaciones virtuales). La teoría de diseño de una base de datos relacional desarrollada en este capítulo se aplica fundamentalmente a las *relaciones base,* aunque algunos criterios de idoneidad también se utilizan en las vistas (consulte la Sección 10.1).

Como ocurre con otros muchos problemas de diseño, el de una base de datos debe llevarse a cabo usando dos metodologías: ascendente (*bottom-up*) o descendente (*top-down*). Una **metodología de diseño de tipo ascendente**, llamada también *diseño por síntesis*, tiene como punto de partida las relaciones básicas entre *atributos*

individuales, y los usa para construir los esquemas de relación. Esta metodología no es muy popular en la práctica[1], ya que tiene el problema de tener que recopilar al principio una gran cantidad de relaciones binarias entre los atributos. Como contrapartida, en una **metodología descendente**, conocida también como *diseño por análisis*, se empieza con varios agrupamientos de atributos de una relación que están juntos de forma natural, como por ejemplo, en una factura, un formulario o un informe. Las relaciones son entonces analizadas individual y colectivamente, lo que conduce a una descomposición posterior que permite conocer todas las propiedades deseables. La teoría descrita en este capítulo es aplicable a ambas metodologías de diseño, aunque es más práctica cuando se emplea en la de tipo descendente.

Iniciamos este capítulo comentando de manera informal algunos criterios en la Sección 10.1 para determinar un buen o mal esquema de relación. En la Sección 10.2 definimos el concepto de *dependencia funcional,* una restricción formal entre los atributos que es la herramienta principal para la medida formal de la idoneidad del agrupamiento de atributos en los esquemas de relación. También se estudian y analizan las propiedades de las dependencias funcionales. La Sección 10.3 se centra en las formas normales y en el proceso de normalización usando dependencias funcionales. Las formas normales sucesivas están definidas para cumplir el conjunto de restricciones deseables expresadas mediante dependencias funcionales. El procedimiento de normalización consiste en la aplicación de una serie de comprobaciones de las relaciones para cumplir con unos requisitos cada vez más restrictivos y descomponer las relaciones cuando sea necesario. En la Sección 10.4 tratamos las definiciones más generales de las formas normales que pueden aplicarse directamente a un diseño concreto y que no precisan de un análisis paso a paso y una normalización.

El Capítulo 11 continúa con el desarrollo de la teoría para un buen diseño del esquema relacional. Comentamos las propiedades deseables de la descomposición relacional (propiedad de reunión no aditiva y de preservación de dependencia funcional) y después consideramos la metodología de tipo ascendente en el diseño de una base de datos que consiste en un conjunto de algoritmos. Estos algoritmos asumen como entrada un conjunto dado de dependencias funcionales y consiguen un diseño relacional en una forma normal de destino a la vez que añade las propiedades deseables antes comentadas. También se presenta un algoritmo general que verifica si una descomposición tiene o no la propiedad de reunión sin pérdida (Algoritmo 11.1). El Capítulo 11 contiene además la definición de tipos de dependencias adicionales y formas normales avanzadas que lleva más allá la *idoneidad* de un esquema de relación.

El lector que sólo está interesado en una introducción informal a la normalización puede saltarse las Secciones 10.2.3, 10.2.4 y 10.2.5. Si no se estudia el Capítulo 11 en un curso, recomendamos una introducción rápida a las propiedades deseables de la descomposición mostradas de la Sección 11.1 y un debate de la propiedad NJB, además del Capítulo 10.

10.1 Directrices de diseño informales para los esquemas de relación

Antes de entrar en detalles con la teoría formal del diseño de bases de datos relacionales, vamos a ver en esta sección cuatro *medidas informales* de calidad para el diseño de un esquema de relación:

- La semántica de los atributos.
- La reducción de información redundante en las tuplas.
- La reducción de los valores NULL en las tuplas.
- Prohibición de la posibilidad de generar tuplas falsas.

Como podremos ver, estas medidas no siempre son independientes entre sí.

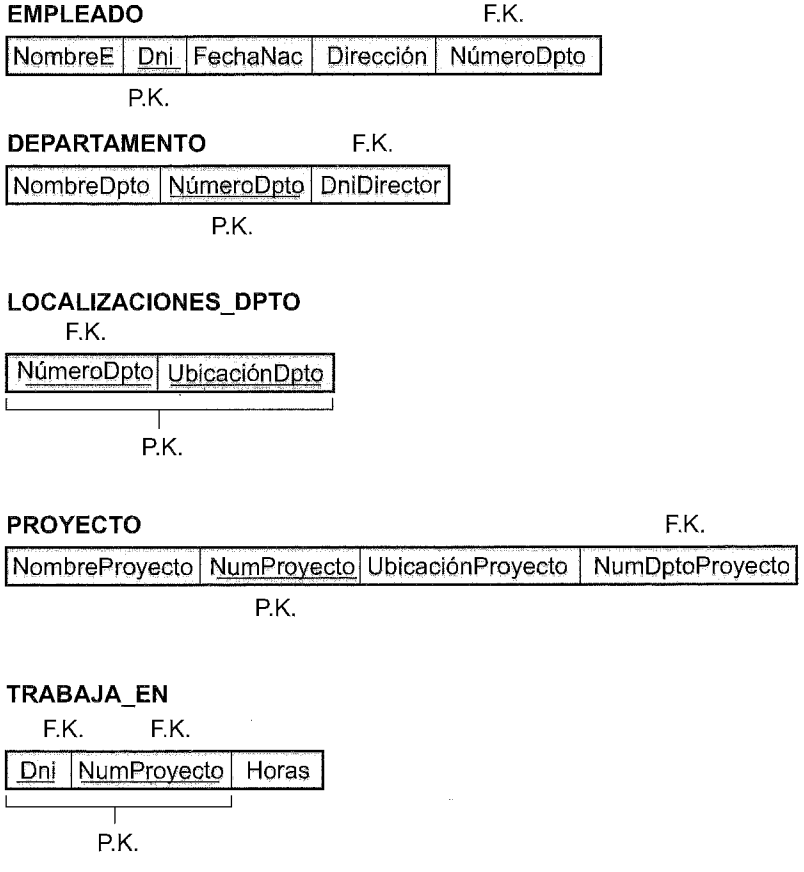
[1] El modelo relacional binario es una excepción en la que se usa esta metodología en la práctica. Un ejemplo del mismo es la metodología NIAM (Verheijen y VanBekkum 1982).

10.1.1 Impartir una semántica clara a los atributos de las relaciones

Siempre que agrupamos atributos para formar un esquema de relación asumimos que pertenecen a una relación que tiene cierta similitud con el mundo real y una interpretación propia asociada a ellos. La semántica de una relación hace referencia a la interpretación de los valores de un atributo en una tupla. En el Capítulo 5 vimos que una relación puede interpretarse como un conjunto de hechos. Si el diseño conceptual descrito en los Capítulos 3 y 4 se lleva a cabo cuidadosamente y el procedimiento de mapeado del Capítulo 7 se sigue sistemáticamente, el diseño del esquema relacional debería tener un significado claro.

En general, cuanto más sencillo es explicar la semántica de la relación, mejor será el diseño del esquema de relación. Para ilustrar esto, considere la Figura 10.1, una versión simplificada del esquema de base de datos relacional EMPRESA de la Figura 5.5, y la Figura 10.2, que muestra un ejemplo de estado de relación. El significado del esquema de relación EMPLEADO es muy simple: cada tupla representa a un empleado, con valores que contienen su nombre (NombreE), su Documento Nacional de Identidad (Dni), su fecha de nacimiento (FechaNac), su dirección (Dirección) y el número del departamento en el que trabaja (NúmeroDpto). El atributo NúmeroDpto es una *foreign key* que representa una *relación implícita* entre EMPLEADO y DEPARTAMENTO. Las semánticas de los esquemas DEPARTAMENTO y PROYECTO son también muy directas: cada tupla DEPARTAMENTO representa a una entidad departamento, y cada tupla PROYECTO es una entidad proyecto. El atributo DniDirector de DEPARTAMENTO relaciona un departamento con el empleado que es director del mismo, mientras que NumDptoProyecto de PROYECTO asocia un proyecto con el departamento que lo gestiona; ambos atributos son *foreign keys*. La facilidad con la que se pueda explicar el significado de los atributos de una relación es una *medida informal* de lo bien que está diseñada esa relación.

Figura 10.1. Una versión simplificada del esquema de base de datos relacional EMPRESA.



La semántica de los otros dos esquemas de relación de la Figura 10.1 es algo más compleja. Cada tupla de LOCALIZACIONES_DPTO consta de un número de departamento (NúmeroDpto) y *una* de las localizaciones del departamento (UbicaciónDpto). Cada tupla de TRABAJA_EN contiene el DNI del empleado (DniEmpleado), el número de uno de los proyectos en los que trabaja (NumProyecto) y el número de horas semanales que le dedica al mismo (Horas). Sin embargo, ambos esquemas tienen una interpretación bien definida y sin ambigüedad. El esquema LOCALIZACIONES_DPTO representa un atributo multivalor de DEPARTAMENTO, mientras que TRABAJA_EN es una relación M:N entre EMPLEADO y PROYECTO. Por consiguiente, todo el esquema de relaciones de la Figura 10.1 podría considerarse como fácil de explicar y, por tanto, bueno desde el punto de vista de contar con una semántica clara. De esta forma, podemos formular la siguiente directriz informal de diseño.

Directriz 1

Diseñar un esquema de relación para que sea fácil explicar su significado. No combine atributos de varios tipos de entidad y de relación en una única relación. Intuitivamente, si un esquema de relación se corres-

Figura 10.2. Ejemplo del estado de la base de datos para el esquema de base de datos relacional de la Figura 10.1.

EMPLEADO

NombreE	Dni	FechaNac	Dirección	NúmeroDpto
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98	5
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9	5
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38	4
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67	4
Ojeda Ordóñez, Fernando.	666884444	15-09-1962	Portillo, s/n	5
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6	5
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90	4
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1	1

DEPARTAMENTO

NombreDpto	NúmeroDpto	DniDirector
Investigación	5	333445555
Administración	4	987654321
Sede central	1	888665555

LOCALIZACIONES_DPTO

NúmeroDpto	UbicaciónDpto
1	Madrid
4	Gijón
5	Valencia
5	Sevilla
5	Madrid

PROYECTO

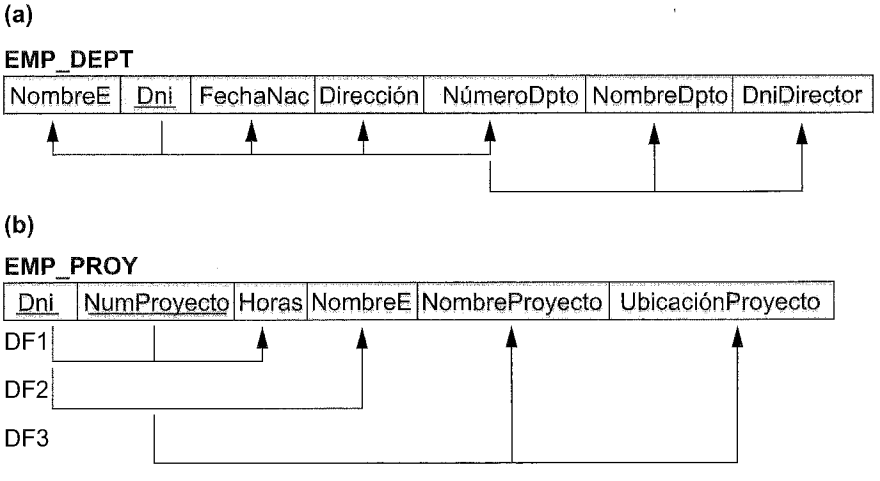
NombreProyecto	NumProyecto	UbicaciónProyecto	NumDptoProyecto
ProductoX	1	Valencia	5
ProductoY	2	Sevilla	5
ProductoZ	3	Madrid	5
Computación	10	Gijón	4
Reorganización	20	Madrid	1
Comunicaciones	30	Gijón	4

Figura 10.2. (*Continuación*).

TRABAJA_EN

Dni	NumProyecto	Horas
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

Figura 10.3. Dos esquemas de relación con anomalías en la actualización. (a) EMP_DEPT y (b) EMP_PROY



ponde con un tipo de entidad o de relación, es correcto interpretar y explicar su significado. Por contra, si la relación está compuesta por una mezcla de múltiples entidades y relaciones, se producirá una ambigüedad semántica y la relación no podrá explicarse con claridad.

Los esquemas de relación de las Figuras 10.3(a) y 10.3(b) tienen también semánticas claras (el lector debe ignorar por el momento las líneas que aparecen bajo las relaciones; se utilizan para documentar la notación de dependencia funcional que explicamos en la Sección 10.2). Una tupla en el esquema de relación EMP_DEPT de la Figura 10.3(a) representa a un solo empleado, aunque incluye información adicional: el

nombre del departamento en el que trabaja (NombreDpto) y el DNI del director de ese departamento (DniDirector). En la relación EMP_PROY de la Figura 10.3(b), cada tupla relaciona un empleado con un proyecto, aunque incluye también el nombre del empleado (NombreE), el del proyecto (NombreProyecto) y la localización de éste (UbicaciónProyecto). Aunque desde el punto de vista lógico no existe nada erróneo en estas dos relaciones, se considera que tienen un diseño pobre porque violan la directriz 1 al mezclar atributos de dos entidades del mundo real; EMP_DEPT combina atributos de empleados y departamentos, mientras que EMP_PROY combina atributos de empleados y proyectos y la relación TRABAJA_EN. Deberían utilizarse como vistas, aunque esto provocaría problemas cuando se usasen como relaciones base, tal y como veremos en la siguiente sección.

10.1.2 Información redundante en tuplas y anomalías en la actualización

Uno de los objetivos de un esquema de diseño es reducir el espacio de almacenamiento utilizado por las relaciones base (y, por tanto, por los ficheros correspondientes). El agrupamiento de atributos en esquemas de relación tiene un efecto significativo sobre el espacio de almacenamiento. Por ejemplo, compare el espacio empleado por las dos relaciones base EMPLEADO y DEPARTAMENTO de la Figura 10.2 con el necesario para EMP_DEPT de la Figura 10.4, que es el resultado de aplicar la operación NATURAL JOIN a EMPLEADO y DEPARTAMENTO. En EMP_DEPT, los valores de atributo pertenecientes a un departamento particular (NúmeroDpto, NombreDpto, DniDirector) están repetidos para *cada empleado que trabaja en ese departamento.* Por contra, la información de cada departamento sólo aparece una vez en la relación DEPARTAMENTO de la Figura 10.2. Por cada empleado que trabaja en ese departamento, sólo se repite el número de departamento (NúmeroDpto) en la relación EMPLEADO como una *foreign key*. A la relación EMP_PROY pueden aplicársele comentarios similares (véase la Figura 10.4), que aumenta la relación TRABAJA_EN con atributos adicionales procedentes de EMPLEADO y PROYECTO.

Otro serio problema que aparece cuando se usan las relaciones de la Figura 10.4 como relaciones base son las **anomalías en la actualización**, las cuales pueden clasificarse en anomalías de inserción, de borrado y de modificación.[2]

Anomalías de inserción. Las anomalías de inserción pueden diferenciarse en dos tipos, que se ilustran con los siguientes ejemplos basados en la relación EMP_DEPT:

- Para insertar una nueva tupla en EMP_DEPT, debemos incluir los valores correspondientes al departamento en el que dicho empleado trabaja, o valores NULL en el caso de que no lo haga para ninguno. Por ejemplo, para insertar una nueva tupla para un empleado que trabaja en el departamento número 5, debemos introducir correctamente los valores de atributo del departamento 5, de modo que sean *coherentes* con los valores correspondientes del resto de tuplas de EMP_DEPT. En el diseño de la Figura 10.2 no tenemos que preocuparnos por el problema de la coherencia ya que sólo indicamos el número de departamento en la tupla de empleado; el resto de valores de atributo del departamento 5 sólo se graban una vez en la base de datos, como una única tupla de la relación DEPARTAMENTO.
- Es complicado insertar un nuevo departamento que aún no tenga ningún empleado en la relación EMP_DEPT. La única forma de hacerlo es colocando valores NULL en los atributos correspondiente al empleado. Esto genera un problema, ya que el DNI es la clave principal de EMP_DEPT, y se supone que cada tupla representa a una entidad empleado, no a una entidad departamento. Además, cuando se asigna el primer empleado a ese departamento, ya no necesitaremos nunca más esta tupla con valores NULL. Este problema no se da en el diseño de la Figura 10.2 porque un departamento se introduce en

[2] Estas anomalías fueron identificadas por Codd (1972a) para justificar la necesidad de normalización en las relaciones, como ya comentaremos en la Sección 10.3.

Figura 10.4. Ejemplo de los estados de EMP_DEPT y EMP_PROY resultantes de aplicar una NATURAL JOIN a las relaciones de la Figura 10.2. Éstas deberían almacenarse como relaciones base por motivos de rendimiento.

EMP_DEPT

Redundancia

NombreE	Dni	FechaNac	Dirección	NúmeroDpto	NombreDpto	DniDirector
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98	5	Investigación	333445555
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9	5	Investigación	333445555
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38	4	Administración	987654321
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67	4	Administración	987654321
Ojeda Ordóñez, Fernando.	666884444	15-09-1962	Portillo, s/n	5	Investigación	333445555
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6	5	Investigación	333445555
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90	4	Administración	987654321
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1	1	Sede central	888665555

EMP_PROY

Redundancia Redundancia

Dni	NumProyecto	Horas	NombreE	NombreProyecto	UbicaciónProyecto
123456789	1	32.5	Pérez Pérez, José	ProductoX	Valencia
123456789	2	7.5	Pérez Pérez, José	ProductoY	Sevilla
666884444	3	40.0	Ojeda Ordóñez, Fernando.	ProductoZ	Madrid
453453453	1	20.0	Oliva Avezuela, Aurora	ProductoX	Valencia
453453453	2	20.0	Oliva Avezuela, Aurora	ProductoY	Sevilla
333445555	2	10.0	Campos Sastre, Alberto	ProductoY	Sevilla
333445555	3	10.0	Campos Sastre, Alberto	ProductoZ	Madrid
333445555	10	10.0	Campos Sastre, Alberto	Computación	Gijón
333445555	20	10.0	Campos Sastre, Alberto	Reorganización	Madrid
999887777	30	30.0	Jiménez Celaya, Alicia	Comunicaciones	Gijón
999887777	10	10.0	Jiménez Celaya, Alicia	Computación	Gijón
987987987	10	35.0	Pajares Morera, Luís	Computación	Gijón
987987987	30	5.0	Pajares Morera, Luís	Comunicaciones	Gijón
987654321	30	20.0	Sainz Oreja, Juana	Comunicaciones	Gijón
987654321	20	15.0	Sainz Oreja, Juana	Reorganización	Madrid
888665555	20	Null	Ochoa Paredes, Eduardo	Reorganización	Madrid

la relación DEPARTAMENTO independientemente de que existan o no empleados trabajando en él, y siempre que un empleado se asigne a ese departamento se inserta una tupla en EMPLEADO.

Anomalías de borrado. El problema de las anomalías de borrado está relacionado con la segunda anomalía de inserción comentada anteriormente. Si eliminamos de EMP_DEPT una tupla empleado que representa al último que trabaja para un departamento dado, la información concerniente a ese departamento se pierde de la base de datos. Este problema no ocurre en la base de datos de la Figura 10.2 porque las tuplas DEPARTAMENTO se almacenan de forma separada.

Anomalías de modificación. En EMP_DEPT, si cambiamos el valor de uno de los atributos de un departamento particular (por ejemplo, el director del departamento 5), debemos actualizar las tuplas de todos los empleados que trabajan en ese departamento; en caso de no hacerlo, la base de datos se volverá inconsistente. Si falla la actualización de alguna tupla, el mismo departamento tendrá dos valores diferentes como director en distintas tuplas de empleado, lo que será incorrecto.[3]

Basándonos en las tres anomalías precedentes, podemos enunciar la siguiente directriz.

Directriz 2

Diseñar los esquemas de relación base de forma que no se presenten anomalías de inserción, borrado o actualización en las relaciones. En caso de que aparezca alguna de ellas, anótela claramente y asegúrese de que los programas que actualizan la base de datos operarán correctamente.

La segunda directriz es coherente, en cierto modo, con una reafirmación de la primera directriz. Podemos ver también la necesidad de una metodología más formal para evaluar si un diseño cumple estas directrices. Las Secciones de la 10.2 a la 10.4 muestran estas necesidades formales. Es importante indicar que estas directrices pueden, a veces, *tener que violarse* para *mejorar el rendimiento* de ciertas consultas. Por ejemplo, si una consulta importante recupera información relativa al departamento de un empleado junto con atributos de ese empleado, podría usarse el esquema EMP_DEPT como relación base. Sin embargo, deben indicarse y justificarse las anomalías de EMP_DEPT (por ejemplo, usando *triggers* o procedimientos almacenados que llevarían a cabo actualizaciones automáticas) de modo que, si se actualiza la relación base, no nos encontremos con incoherencias. En general, es aconsejable usar relaciones base que estén libres de anomalías y especificar vistas que incluyan las concatenaciones necesarias para recuperar los atributos que se referencian frecuentemente en las consultas. Esto reduce el número de términos JOIN especificados en la consulta, simplificando la escritura correcta de la consulta y, en muchos casos, mejorando el rendimiento.[4]

10.1.3 Valores NULL en las tuplas

En algunos diseños podemos agrupar muchos atributos en una relación "muy grande". Si muchos de los atributos no se aplican a todas las tuplas de la relación, nos encontraremos con muchos valores NULL en esas tuplas. Esto puede desperdiciar espacio de almacenamiento y puede inducir a problemas a la hora de entender el significado de los atributos con la especificación de operaciones JOIN a nivel lógico.[5] Otro problema con los NULL es cómo contabilizarlos cuando se aplican operaciones de agregación como COUNT o SUM. Las operaciones SELECT o JOIN implican comparaciones. Si hay presentes valores NULL, los resultados serán impredecibles.[6] Además, los NULL pueden tener múltiples interpretaciones:

- El atributo *no se aplica* a esta tupla.
- El valor de atributo de esta tupla es *desconocido*.
- El valor es *conocido pero está ausente*, es decir, aún no se ha grabado.

---

[3] Esto no es tan serio como otros problemas, ya que todas las tuplas pueden actualizarse con una sola sentencia SQL.

[4] El rendimiento de una consulta especificada en una vista que es la concatenación de varias relaciones base depende de cómo el DBMS implementa la vista. Muchos RDBMSs materializan las vistas usadas frecuentemente de forma que no se tengan que llevar a cabo las concatenaciones más habituales. El DBMS es responsable de la actualización de la vista materializada (ya sea inmediata o periódicamente) siempre que las relaciones base se modifiquen.

[5] Esto se debe a que las concatenaciones externas e internas producen resultados diferentes cuando existen valores NULL implicados en ellas. Los usuarios deben, por tanto, tener cuidado con los distintos significados de cada tipo de concatenación. Lo que resulta razonable para usuarios sofisticados, puede ser difícil para otros.

[6] En la Sección 8.5.1 presentamos varias comparaciones que implican valores NULL donde el resultado (en la lógica de tres valores) es TRUE, FALSE y UNKNOWN.

El tener la misma representación para todos los NULL compromete los diferentes significados que pueden tener. Por consiguiente, podemos establecer otra directriz.

Directriz 3

Hasta donde sea posible, evite situar en una relación base atributos cuyos valores sean NULL frecuentemente. En caso de no poderse evitar, asegúrese de que se aplican sólo en casos excepcionales y no los aplique a la mayor parte de las tuplas de la relación.

Utilizar el espacio eficientemente y evitar concatenaciones son los dos criterios principales que determinan si incluir las columnas que pueden tener valores NULL en una relación o tener una relación separada para esas columnas (con las columnas clave apropiadas). Por ejemplo, si sólo el 10 por ciento de los empleados tienen oficinas individuales, no es razón suficiente para la inclusión de un atributo NúmeroOficina en la relación EMPLEADO; en lugar de ello, se puede crear una relación OFICINAS_EMPS(DniEmpleado, NúmeroOficina) que incluya las tuplas de los empleados con oficinas individuales.

10.1.4 Generación de tuplas falsas

Considere los dos esquemas de relación EMP_LOCS y EMP_PROY1 de la Figura 10.5(a), la cual puede usarse en lugar de la relación simple EMP_PROY de la Figura 10.3(b). Una tupla en EMP_LOCS significa que el empleado cuyo nombre es NombreE trabaja en *algún proyecto* cuya localización es UbicaciónProyecto. Una tupla EMP_PROY1 se refiere al hecho de que el empleado cuyo Documento Nacional de Identidad es Dni trabaja un número de Horas por semana en el proyecto cuyo nombre, número y ubicación son NombreProyecto, NumProyecto y UbicaciónProyecto. La Figura 10.5(b) muestra el estado de relación de EMP_LOCS y EMP_PROY1 correspondiente a la relación EMP_PROY de la Figura 10.4, la cual se obtiene aplicando la operación PROYECCIÓN (π) adecuada a EMP_PROY [ignore por ahora las líneas discontinuas de la Figura 10.5(b)].

Supongamos que utilizamos EMP_PROY1 y EMP_LOCS como relaciones base en lugar de EMP_PROY. Esto produce un diseño de esquema incorrecto algo peculiar porque no podemos recuperar la información originaria de EMP_PROY desde EMP_PROY1 y EMP_LOCS. Si intentamos llevar a cabo una operación CONCATENACIÓN NATURAL en estas relaciones, el resultado produce muchas más tuplas que las existentes en el conjunto original de EMP_PROY. En la Figura 10.6, sólo se muestra la aplicación de la concatenación a las tuplas que están *por encima* de las líneas discontinuas de la Figura 10.5(b) (para reducir el tamaño de la relación resultante). Las tuplas adicionales que no se encontraban en EMP_PROY reciben el nombre de **tuplas falsas** (***spurious tuples***) porque representa información falsa que no es válida. Las tuplas falsas están marcadas con asteriscos (*) en la Figura 10.6.

No es aconsejable descomponer EMP_PROY en EMP_LOCS y EMP_PROY1 porque cuando deshacemos la CONCATENACIÓN usando una CONCATENACIÓN NATURAL, no obtenemos la información original correcta. Esto es así porque, en este caso, UbicaciónProyecto es el atributo que relaciona EMP_LOCS y EMP_PROY1, y no es ni una clave principal ni una *foreign key* en EMP_LOCS o EMP_PROY1. Ahora estamos en condiciones de definir otra directriz de diseño.

Directriz 4

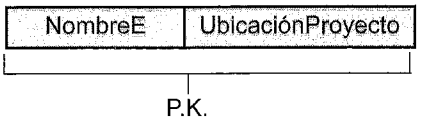
Diseñar los esquemas de relación de forma que puedan concatenarse con condiciones de igualdad en los atributos que son parejas de clave principal y *foreign key* de forma que se garantice que no se van a generar tuplas falsas. Evite las relaciones que contienen atributos coincidentes que no son combinaciones de *foreign key* y clave principal porque la concatenación de estos atributos puede producir tuplas falsas.

Esta directriz informal debe ser, obviamente, redefinida de una manera más adecuada. En el Capítulo 11 trataremos una condición formal llamada propiedad de reunión no aditiva que garantiza que ciertas concatenaciones no producen tuplas falsas.

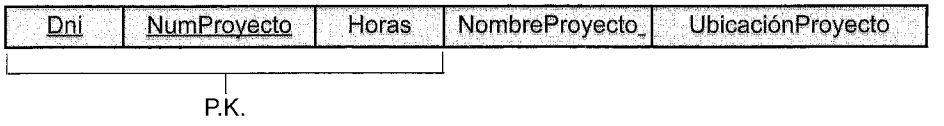
Figura 10.5. Diseño particularmente pobre de la relación EMP_PROY de la Figura 10.3(b). (a) Los dos esquemas de relación EMP_LOCS y EMP_PROY1. (b) El resultado de proyectar la extensión de EMP_PROY de la Figura 10.4 a las relaciones EMP_LOCS y EMP_PROY1.

(a)

EMP_LOCS



EMP_PROJ1



(b)

EMP_LOCS

NombreE	Ubicación-Proyecto
Pérez Pérez, José	Valencia
Pérez Pérez, José	Surgarland
Ojeda Ordóñez, Fernando.	Madrid
Oliva Avezuela, Aurora	Valencia
Oliva Avezuela, Aurora	Surgarland
Campos Sastre, Alberto	Surgarland
Campos Sastre, Alberto	Madrid
Campos Sastre, Alberto	Gijón
Jiménez Celaya, Alicia	Gijón
Pajares Morera, Luis	Gijón
Sainz Oreja, Juana	Gijón
Sainz Oreja, Juana	Madrid
Ochoa Paredes, Eduardo	Madrid

EMP_PROJ1

Dni	NumProyecto	Horas	NombreProyecto	Ubicación-Proyecto
123456789	1	32.5	ProductoX	Valencia
123456789	2	7.5	ProductoY	Sevilla
666884444	3	40.0	ProductoZ	Madrid
453453453	1	20.0	ProductoX	Valencia
453453453	2	20.0	ProductoY	Sevilla
333445555	2	10.0	ProductoY	Sevilla
333445555	3	10.0	ProductoZ	Madrid
333445555	10	10.0	Computación	Gijón
333445555	20	10.0	Reorganización	Madrid
999887777	30	30.0	Comunicaciones	Gijón
999887777	10	10.0	Computación	Gijón
987987987	10	35.0	Computación	Gijón
987987987	30	5.0	Comunicaciones	Gijón
987654321	30	20.0	Comunicaciones	Gijón
987654321	20	15.0	Reorganización	Madrid
888665555	20	NULL	Reorganización	Madrid

10.1.5 Resumen y explicación acerca de las directrices de diseño

En las Secciones de la 10.1.1 a la 10.1.4, hemos visto situaciones que provocan esquemas de relación problemáticos, y hemos propuesto unas directrices informales para definir un buen diseño relacional. Los problemas que hemos apuntado, que pueden detectarse sin la intervención de herramientas de análisis adicionales, son los siguientes:

- Anomalías que causan trabajo redundante durante la inserción y modificación de una relación, y que pueden causar pérdidas accidentales de información durante el borrado de la misma.
- Desaprovechamiento del espacio de almacenamiento debido a valores NULL y la dificultad de llevar a cabo operaciones de selección, agregación y concatenación debido a estos valores.

Figura 10.6. Resultado de aplicar una CONCATENACIÓN NATURAL a las tuplas que se encuentran por encima de las líneas discontinuas en EMP_PROY1 y EMP_LOCS de la Figura 10.5. Las tuplas falsas generadas aparecen marcadas con asteriscos.

	Dni	NumProyecto	Horas	NombreProyecto	UbicaciónProyecto	NombreE
	123456789	1	32.5	ProductoX	Valencia	Pérez Pérez, José
*	123456789	1	32.5	ProductoX	Valencia	Oliva Avezuela, Aurora
	123456789	2	7.5	ProductoY	Sevilla	Pérez Pérez, José
*	123456789	2	7.5	ProductoY	Sevilla	Oliva Avezuela, Aurora
*	123456789	2	7.5	ProductoY	Sevilla	Campos Sastre, Alberto
	666884444	3	40.0	ProductoZ	Madrid	Ojeda Ordóñez, Fernando.
*	666884444	3	40.0	ProductoZ	Madrid	Campos Sastre, Alberto
*	453453453	1	20.0	ProductoX	Valencia	Pérez Pérez, José
	453453453	1	20.0	ProductoX	Valencia	Oliva Avezuela, Aurora
*	453453453	2	20.0	ProductoY	Sevilla	Pérez Pérez, José
	453453453	2	20.0	ProductoY	Sevilla	Oliva Avezuela, Aurora
*	453453453	2	20.0	ProductoY	Sevilla	Campos Sastre, Alberto
*	333445555	2	10.0	ProductoY	Sevilla	Pérez Pérez, José
*	333445555	2	10.0	ProductoY	Sevilla	Oliva Avezuela, Aurora
	333445555	2	10.0	ProductoY	Sevilla	Campos Sastre, Alberto
*	333445555	3	10.0	ProductoZ	Madrid	Ojeda Ordóñez, Fernando.
	333445555	3	10.0	ProductoZ	Madrid	Campos Sastre, Alberto
	333445555	10	10.0	Computación	Gijón	Campos Sastre, Alberto
*	333445555	20	10.0	Reorganización	Madrid	Ojeda Ordóñez, Fernando.
	333445555	20	10.0	Reorganización	Madrid	Campos Sastre, Alberto

*
*
*

- Generación de datos incorrectos y falsos durante las concatenaciones en relaciones base incorrectamente relacionadas.

En el resto de este capítulo vamos a presentar conceptos y teorías formales que pueden utilizarse para definir de forma más precisa la *idoneidad* y la *mala calidad* de un esquema de relación *individual*. En primer lugar comentaremos la dependencia funcional como una herramienta de análisis. A continuación especificaremos las tres formas normales y la BCNF (Forma normal de Boyce-Codd, *Boyce-Codd Normal Form*) para un esquema de relación. En el Capítulo 11, definimos formas normales adicionales que están basadas en dependencias de tipos de datos adicionales llamadas dependencias multivalor y dependencias de concatenación.

10.2 Dependencias funcionales

El concepto básico más importante en la teoría de diseño de un esquema relacional es el de una dependencia funcional. En esta sección definiremos formalmente el concepto, mientras que en la Sección 10.3 veremos cómo usarlo para definir formas normales para los esquemas de relación.

10.2.1 Definición de dependencia funcional

Una dependencia funcional es una restricción que se establece entre dos conjuntos de atributos de la base de datos. Supongamos que nuestro esquema de base de datos relacional tiene n atributos $A_1, A_2, \ldots, A_n$; pense-

mos que la base de datos completa está descrita por un único esquema de relación **universal** $R = \{A_1, A_2, \ldots, A_n\}$.[7] No sugerimos que vamos a almacenar la base de datos como una única tabla universal; usamos este concepto sólo en el desarrollo de la teoría formal de las dependencias de datos.[8]

> **Definición**. Una dependencia funcional, denotada por $X \rightarrow Y$, entre dos conjuntos de atributos X e Y que son subconjuntos de R, especifica una *restricción* en las posibles tuplas que pueden formar un estado de relación r de R. La restricción dice que dos tuplas t_1 y t_2 en r que cumplen que $t_1[X] = t_2[X]$, deben cumplir también que $t_1[Y] = t_2[Y]$.

Esto significa que los valores del componente Y de una tupla de r dependen de, o *están determinados por,* los valores del componente X; alternativamente, los valores del componente X de una tupla únicamente (o **funcionalmente**) *determinan* los valores del componente Y. Decimos también que existe una dependencia funcional de X hacia Y, o que Y es **funcionalmente dependiente** de X. La abreviatura de dependencia funcional es **DF**, o **FD** o **f.d.** (del inglés, *functional dependency*). El conjunto de atributos X recibe el nombre de **lado izquierdo** de la DF, mientras que Y es el **lado derecho**.

Por tanto, X determina funcionalmente Y si para toda instancia r del esquema de relación R, no es posible que r tenga dos tuplas que coincidan en los atributos de X y no lo hagan en los atributos de Y. Observe lo siguiente:

- Si una restricción de R indica que no puede haber más de una tupla con un valor X concreto en cualquier instancia de relación $r(R)$, es decir, que X es una **clave candidata** de R, se cumple que $X \rightarrow Y$ para cualquier subconjunto de atributos Y de R [ya que la restricción de clave implica que dos tuplas en cualquier estado legal $r(R)$ no tendrán el mismo valor de X].
- Si $X \rightarrow Y$ en R, esto no supone que $Y \rightarrow X$ en R.

Una dependencia funcional es una propiedad de la **semántica** o **significado de los atributos**. Los diseñadores de la base de datos utilizarán su comprensión de la semántica de los atributos de R (esto es, cómo se relacionan unos con otros) para especificar las dependencias funcionales que deben mantenerse en *todos* los estados de relación (extensiones) r de R. Siempre que la semántica de dos conjuntos de atributos de R indique que debe mantenerse una dependencia funcional, la especificamos como una restricción. Las extensiones de relación $r(R)$ que satisfacen la restricción de dependencia funcional reciben el nombre de **estados de relación legales** (o **extensiones legales**) de R. Por tanto, el uso fundamental de las dependencias funcionales es describir más en profundidad un esquema de relación R especificando restricciones de sus atributos que *siempre deben cumplirse.* Ciertas DF pueden especificarse sin hacer referencia a una relación específica. Por ejemplo, {Provincia, NumPermisoConducir} → Dni debe mantenerse para cualquier adulto que viva en España. También es posible que ciertas dependencias funcionales puedan dejar de existir en el mundo real si cambia la relación. Por ejemplo, en Estados Unidos la DF CódigoPostal → CodÁrea se utiliza como una relación entre los códigos postales y los códigos de los números telefónicos, pero con la proliferación de los códigos de área telefónica ya no es tan cierta.

Considere el esquema de relación EMP_PROY de la Figura 10.3(b); desde el punto de vista de la semántica de los atributos sabemos que deben mantenerse las siguientes dependencias funcionales:

a. Dni → NombreE

b. NumProyecto → {NombreProyecto, UbicaciónProyecto}

c. {Dni, NúmeroDpto} → Horas

---

[7] Este concepto de una relación universal es importante cuando se explican los algoritmos para el diseño de una base de datos relacional en el Capítulo 11.

[8] Esta presunción implica que cada atributo de la base de datos debe tener un nombre diferente. En el Capítulo 5 prefijamos nombres de atributo derivados de nombres de relación para lograr la unicidad siempre que los atributos de distintas relaciones tuvieran el mismo nombre.

Figura 10.7. Un estado de relación IMPARTIR con una *posible* dependencia funcional TEXTO → CURSO. Sin embargo, PROFESOR → CURSO no es posible.

IMPARTIR

Profesor	Curso	Texto
Smith	Estructuras de datos	Bartram
Smith	Administración de datos	Martin
Hall	Compiladores	Hoffman
Brown	Estructuras de datos	Horowitz

Estas dependencias funcionales especifican que (a) el valor del Documento Nacional de Identidad de un empleado (Dni) determina de forma inequívoca su nombre (NombreE), (b) el valor de un número de proyecto (NumProyecto) determina de forma única el nombre del mismo (NombreProyecto) y su ubicación (UbicaciónProyecto), y (c) una combinación de valores Dni y NumProyecto determina el número de horas semanales que el empleado ha trabajado en el proyecto (Horas). Alternativamente, decimos que NombreE está determinado funcionalmente por (o es funcionalmente dependiente de) Dni, o que *dado un Dni concreto, conocemos el valor de NombreE,* etc.

Una dependencia funcional es una *propiedad del esquema de relación R*, y no un estado de relación legal particular *r* de *R*. Por consiguiente, una DF *no puede* ser inferida automáticamente a partir de una extensión de relación *r*, sino que alguien que conozca la semántica de los atributos de *R* debe definirla explícitamente. Por ejemplo, la Figura 10.7 muestra un estado particular del esquema de relación IMPARTIR. Aunque a primera vista pudiéramos pensar que Texto → Curso, no podemos confirmarlo a menos que sepamos que se cumple *para todos los estados legales posibles* de IMPARTIR. Sin embargo, basta con demostrar un *único ejemplo en contra* para desautorizar una dependencia funcional. Por ejemplo, ya que 'Smith' enseña tanto 'Estructura de datos' como 'Administración de datos', podemos concluir que Profesor *no* determina funcionalmente a Curso.

La Figura 10.3 muestra una **notación diagramática** para visualizar las DFs: cada una de ellas aparece como una línea horizontal. Los atributos del lado izquierdo de la DF están conectados por líneas verticales a la línea que representa la DF, mientras que los del lado derecho lo están a los atributos mediante flechas que apuntan hacia ellos [véanse las Figuras 10.3(a) y 10.3(b)].

10.2.2 Reglas de inferencia para las dependencias funcionales

Decimos que *F* es el conjunto de dependencias funcionales especificadas en un esquema de relación *R*. Habitualmente, el diseñador del esquema especifica las dependencias funcionales que son *semánticamente obvias*; sin embargo, es habitual que otras muchas dependencias funcionales se encuentren en *todas* las instancias de relación legales entre los conjuntos de atributos que pueden derivarse y satisfacen las dependencias de *F*. Esas otras dependencias pueden *inferirse* o *deducirse* de las DF de *F*. En la vida real, es imposible especificar todas las dependencias funcionales posibles para una situación concreta. Por ejemplo, si cada departamento tiene un director, de manera que NúmeroDpto determina de forma única DniDirector (NúmeroDpto → DniDirector), y un director tiene un único número de teléfono TeléfonoDirector (DniDirector → TeléfonoDirector), entonces ambas dependencias juntas suponen que NúmeroDpto → TeléfonoDirector. Esto es una DF inferida y *no* tiene que declararse explícitamente. Por tanto, formalmente es útil definir un concepto llamado *clausura* (*closure*) que incluye todas las posibles dependencias que pueden inferirse de un conjunto *F* dado.

> **Definición.** Formalmente, el conjunto de todas las dependencias que incluyen *F*, junto con las dependencias que pueden inferirse de F, reciben el nombre de **clausuras** de *F*; está designada mediante F^+.

Por ejemplo, suponga que especificamos el siguiente conjunto *F* de dependencias funcionales obvias en el esquema de relación de la Figura 10.3(a):

F = {Dni → {NombreE, FechaNac, Dirección, NúmeroDpto},
NúmeroDpto → {NombreDpto, DniDirector} }

Las siguientes son algunas de las dependencias funcionales adicionales que se pueden *inferir* de F:

Dni → {NombreDpto, DniDirector}
Dni → Dni
NúmeroDpto → NombreDpto

Una DF $X \rightarrow Y$ es **inferida de** un conjunto de dependencias F especificado en R si $X \rightarrow Y$ se cumple en *todo* estado de relación legal r de R; es decir, siempre que r satisfaga todas las dependencias en F, $X \rightarrow Y$ también se cumple en r. La clausura F^+ de F es el conjunto de todas las dependencias funcionales que pueden inferirse de F. Para determinar una manera sistemática de inferir dependencias, debemos descubrir un conjunto de **reglas de inferencia** que puedan usarse para deducir nuevas dependencias a partir de un conjunto de dependencias concreto. A continuación vamos a considerar algunas de estas reglas de inferencia. Usamos la notación $F \models X \rightarrow Y$ para indicar que la dependencia funcional $X \rightarrow Y$ se infiere del conjunto de dependencias funcionales F.

En la siguiente explicación, usaremos una notación abreviada para hablar de las dependencias funcionales. Por conveniencia, concatenamos las variables de atributo y eliminamos las comas. Por tanto, la DF $\{X,Y\} \rightarrow Z$ se expresa de forma abreviada como $XY \rightarrow Z$, mientras que la DF $\{X, Y, Z\} \rightarrow \{U, V\}$ se indica como $XYZ \rightarrow UV$. Las reglas de la RI1 a la RI6 son reglas de inferencia bien conocidas para las dependencias funcionales:

RI1 (regla reflexiva)[9]: Si $X \supseteq Y$, entonces $X \rightarrow Y$.

RI2 (regla de aumento)[10]: $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

RI3 (regla transitiva): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

RI4 (regla de descomposición, o proyectiva): $\{X \rightarrow YZ\} \models X \rightarrow Y$.

RI5 (regla de unión, o aditiva): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.

RI6 (regla seudotransitiva): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

La regla reflexiva (RI1) especifica que un conjunto de atributos siempre se determina a sí mismo o cualquiera de sus subconjuntos, lo que es obvio. Ya que la RI1 genera dependencias que siempre son verdaderas, éstas reciben el nombre de *triviales*. Formalmente, una dependencia funcional $X \rightarrow Y$ es **trivial** si $X \supseteq Y$; en cualquier otro caso es **no trivial**. La regla de aumento (RI2) dice que añadir el mismo conjunto de atributos a ambos lados de una dependencia genera otra dependencia válida. Según la RI3, las dependencias funcionales son transitivas. La regla de descomposición (RI4) especifica que podemos eliminar atributos del lado derecho de una dependencia; si aplicamos esta regla repetidamente podemos descomponer la DF $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ en el conjunto de dependencias $\{X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n\}$. La regla de unión (RI5) nos permite realizar lo contrario: podemos combinar un conjunto de dependencias $\{X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n\}$ en una única DF $X \rightarrow \{A_1, A_2, \ldots, A_n\}$.

Una *nota preventiva* acerca del uso de estas reglas. Aunque $X \rightarrow A$ y $X \rightarrow B$ implican $X \rightarrow AB$ por la regla de unión antes comentada, $X \rightarrow A$ e $Y \rightarrow B$ no *implican* que $XY \rightarrow AB$. Además, $XY \rightarrow A$ *no* implica necesariamente ni $X \rightarrow A$ ni $Y \rightarrow A$.

Cada una de las reglas de inferencia anteriores puede probarse a partir de la definición de dependencia funcional, bien por comprobación directa o bien **por contradicción**. Una comprobación por contradicción asume

[9] La regla reflexiva puede expresarse también como $X \rightarrow X$, es decir, cualquier conjunto de atributos se determina, funcionalmente, a sí mismo.

[10] La regla de aumento puede definirse también como $\{X \rightarrow Y\} \models XZ \rightarrow Y$, es decir, incrementar los atributos del lado izquierdo de una DF produce otra DF correcta.

que la regla no se cumple y muestra que ésta no es posible. Vamos a demostrar ahora que las tres primeras reglas son válidas. La segunda comprobación es por contradicción.

Comprobación de la RI1. Suponga que $X \supseteq Y$, y que dos tuplas t_1 y t_2 existen en alguna instancia de relación r de R como $t_1[X] = t_2[X]$. Por tanto, $t_1[Y] = t_2[Y]$ porque $X \supseteq Y$; por tanto, $X \rightarrow$ Y debe cumplirse en r.

Comprobación de la RI2 por contradicción. Asumimos que $X \rightarrow Y$ se cumple en una instancia de relación r de R, pero no así $XZ \rightarrow YZ$. Entonces, deben existir dos tuplas t_1 y t_2 en r tales que (1) $t_1[X] = t_2[X]$, (2) $t_1[Y] = t_2[Y]$, (3) $t_1[XZ] = t_2[XZ]$ y (4) $t_1[YZ]$? $t_2[YZ]$. Esto no es posible porque desde los puntos (1) y (3) deducimos el (5) $t_1[Z] = t_2[Z]$, y desde los puntos (2) y (5) deducimos el (6) $t_1[YZ] = t_2[YZ]$, contradiciendo el (4).

Comprobación de la RI3. Asumimos que (1) $X \rightarrow Y$ y (2) $Y \rightarrow Z$ se cumplen en una relación r. Entonces, por cada dos tuplas t_1 y t_2 en r tales que $t_1[X] = t_2[X]$, debemos tener (3) $t_1[Y] = t_2[Y]$, desde la asunción del punto (1); por consiguiente, también debemos tener un punto (4) $t_1[Z] = t_2[Z]$, desde el (3) y asumiendo el (2); por tanto, $X \rightarrow Z$ debe cumplirse en r.

Usando argumentos de comprobación similares, podemos probar las reglas de inferencia de la RI4 a la RI6 y cualquier otra regla de inferencia válida. Sin embargo, una forma más simple de demostrar que una regla de inferencia es válida para las dependencias funcionales es probarla usando algunas de las que ya hemos visto que lo son. Por ejemplo, podemos comprobar las reglas RI4 a RI6 usando las reglas RI1, RI2 y RI3 de la siguiente forma.

Comprobación de RI4 (usando RI1, RI2 y RI3).

1. $X \rightarrow YZ$ (dada).
2. $YZ \rightarrow Y$ (usando RI1 y sabiendo que $YZ \supseteq Y$).
3. $X \rightarrow Y$ (usando RI3 en 1 y 2).

Comprobación de RI5 (usando RI1, RI2 y RI3).

1. $X \rightarrow Y$ (dada).
2. $X \rightarrow Z$ (dada).
3. $X \rightarrow XY$ (usando RI2 en 1 aumentando con X; observe que $XX = X$).
4. $XY \rightarrow YZ$ (usando RI2 en 2 aumentando con Y).
5. $X \rightarrow YZ$ (usando RI3 en 3 y 4).

Comprobación de RI6 (usando las RI1, RI2 y RI3).

1. $X \rightarrow Y$ (dada).
2. $WY \rightarrow Z$ (dada).
3. $WX \rightarrow WY$ (usando RI2 en 1 aumentando con W).
4. $WX \rightarrow Z$ (usando RI3 en 3 y 2).

Armstrong(1974) demostró que las reglas de inferencia de la RI1 a la RI3 son sólidas y completas. Por **sólida** queremos decir que, dado un conjunto de dependencias funcionales F especificado en un esquema de relación R, cualquier dependencia que podamos inferir de F usando RI1, RI2 y RI3 se cumple en cada estado de relación r de R que *satisfaga* las *dependencias* de F. Por **completa** decimos que, usando las tres primeras reglas repetidamente para inferir dependencias hasta que ya no se pueda determinar ninguna más, se genera el conjunto completo de *todas las dependencias posibles* que pueden inferirse a partir de F. En otras palabras, el conjunto de dependencias F^+, que hemos llamado **clausura** de F, pueden determinarse a partir de F usando

sólo las reglas de inferencia de la RI1 a la RI3. Estas tres reglas se conocen como **reglas de inferencia de Armstrong**.[11]

Habitualmente, los diseñadores de bases de datos especifican, en primer lugar, el conjunto de dependencias funcionales F que pueden determinarse fácilmente a partir de la semántica de los atributos de R; por tanto, se usan las reglas RI1, RI2 y RI3 para inferir dependencias funcionales adicionales que también se almacenarán en R. Una forma semántica para determinar estas dependencias funcionales adicionales es determinar, en primer lugar, cada conjunto de atributos X que aparece en la parte izquierda de alguna dependencia funcional de F para, a continuación, determinar el conjunto de *todos los atributos* que son dependientes en X.

Definición. Para cada conjunto de atributos X como éste, determinamos el conjunto X^+ de atributos que están funcionalmente determinados por X basados en F; X^+ recibe el nombre de **clausura de X bajo F**. Puede usarse el Algoritmo 10.1 para calcular X^+.

Algoritmo 10.1. Determinación de X^+, la clausura de X bajo F:

$X^+ := X$;
  repetir
    antigua$X^+ := X^+$;
    por cada dependencia funcional $Y \rightarrow Z$ en F ejecutar
      si $X^+ \supseteq Y$ entonces $X^+ := X^+ \cup Z$;
hasta que (X^+ = antiguaX^+);

El Algoritmo 10.1 empieza asignado a X^+ todos los atributos de X. Según RI1, sabemos que todos esos atributos son funcionalmente dependientes de X. Usando las reglas de inferencia RI3 y RI4, añadimos atributos a X^+, usando cada dependencia funcional en F. Continuamos por todas las dependencias de F (el bucle *repetir*) hasta que no se añadan más atributos a X^+ *durante un ciclo completo* (del bucle *por cada*) a través de las dependencias de F. Por ejemplo, consideremos el esquema de relación EMP_PROY de la Figura 10.3(b); según la semántica de los atributos, especificamos el siguiente conjunto F de dependencias funcionales que deben cumplirse en EMP_PROY:

F = {Dni → NombreE,
  NumProyecto → {NombreProyecto, UbicaciónProyecto},
  {Dni, NumProyecto} → Horas}

Usando el Algoritmo 10.1, calculamos los siguientes conjuntos de clausura con respecto a F:

{Dni} += {Dni, NombreE}
{NumProyecto} += {NumProyecto, NombreProyecto, UbicaciónProyecto}
{Dni, NumProyecto} += {Dni, NumProyecto, NombreE, NombreProyecto, UbicaciónProyecto, Horas}

Intuitivamente, el conjunto de atributos del lado derecho de cada una de las líneas anteriores representa a todos los atributos que son funcionalmente dependientes del conjunto de atributos del lado izquierdo en base al conjunto F dado.

10.2.3 Equivalencia de los conjuntos de dependencias funcionales

En esta sección vamos a tratar la equivalencia de dos conjuntos de dependencias funcionales. En primer lugar vamos a ver algunas definiciones preliminares.

[11] En la actualidad se conocen como **axiomas de Armstrong**. En un sentido estrictamente matemático, los *axiomas* (hechos dados) son las dependencias funcionales de F, ya que asumimos que son correctos, mientras que las reglas RI1, RI2 y RI3 son las *reglas de inferencia* para determinar nuevas dependencias funcionales (nuevos hechos).

Definición. Un conjunto de dependencias funcionales F se dice que **cubre** a otro conjunto de dependencias funcionales E si toda DF de E está también en F^+, es decir, si toda dependencia de E puede ser inferida a partir de F; alternativamente, podemos decir que E está **cubierto por** F.

Definición. Dos conjuntos de dependencias funcionales E y F son **equivalentes** si $E^+ = F^+$. Por consiguiente, equivalencia significa que cada DF de E puede inferirse a partir de F, y viceversa; es decir, E es equivalente a F si se cumplen las condiciones E cubre a F *y* F cubre a E.

Podemos determinar si F cubre a E calculando X^+ *respecto a* F para cada DF $X \rightarrow Y$ *en* E, y comprobando después si X^+ incluye todos los atributos de Y. Si es el caso para *toda* DF en E, entonces F cubre a E. Determinamos si E y F son equivalentes comprobando que E cubre a F y que F cubre a E.

10.2.4 Conjuntos mínimos de dependencias funcionales

Informalmente, una **cobertura mínima** de un conjunto de dependencias funcionales E es un conjunto de dependencias funcionales F que satisface la propiedad de que cada dependencia de E está en la clausura F^+ de F. Además, esta propiedad se pierde si se elimina cualquier dependencia del conjunto F; F no debe tener redundancias, y las dependencias de F están en una forma estándar. Podemos definir formalmente que un conjunto de dependencias funcionales F es **mínimo** si satisface las siguientes condiciones:

1. Toda dependencia en F tiene un único atributo en su lado derecho.
2. No podemos reemplazar ninguna dependencia $X \rightarrow A$ de F por otra dependencia $Y \rightarrow A$, donde Y es un subconjunto propio de X, y seguir teniendo un conjunto de dependencias equivalente a F.
3. No podemos eliminar ninguna dependencia de F y seguir teniendo un conjunto de dependencias equivalente a F.

Podemos concebir un conjunto mínimo de dependencias como un conjunto de dependencias de una *forma estándar* o *canónica* y *sin redundancias*. La condición 1 sólo representa cada dependencia en forma canónica con un único atributo en el lado derecho.[12] Las condiciones 2 y 3 garantizan que no habrá redundancia en las dependencias ya sea por tener atributos redundantes en el lado izquierdo de una dependencia (condición 2) o por tener una dependencia que puede inferirse a partir del renombrado de las DF en F (condición 3).

Definición. Una cobertura mínima de un conjunto de dependencias funcionales E es un conjunto mínimo de dependencias (en forma canónica estándar y sin redundancia) equivalente a E. Con el Algoritmo 10.2, siempre podemos buscar, *al menos*, *una* cobertura mínima F por cada conjunto de dependencias E.

Si hay varios conjuntos de DF cualificados como coberturas mínimas de E por la definición anterior, es costumbre utilizar criterios adicionales para minimizar. Por ejemplo, podemos elegir el conjunto mínimo con el *menor número de dependencias* o con la *longitud total* más pequeña (la longitud total de un conjunto de dependencias se calcula concatenando las dependencias y tratándolas como una cadena de caracteres larga).

Algoritmo 10.2. Localizar una cobertura mínima F para un conjunto de dependencias funcionales E.

1. Establecer $F := E$.
2. Reemplazar cada dependencia funcional $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ en F por las n dependencias funcionales $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
3. Por cada dependencia funcional $X \rightarrow A$ en F
   por cada atributo B que es un elemento de X
   si $\{ \{F - \{X \rightarrow A\}\} \cup \{(X - \{B\}) \rightarrow A\} \}$ es equivalente a F,
   entonces reemplazar $X \rightarrow A$ por $(X - \{B\}) \rightarrow A$ en F.

[12] Es una forma estándar de simplificar las condiciones y los algoritmos que garantizan la no existencia de redundancia en F. Con la regla de inferencia RI4, podemos convertir una dependencia simple con múltiples atributos en su parte derecha en un conjunto de dependencias con atributos simples en la parte derecha.

4. Por cada dependencia funcional $X \rightarrow A$ sobrante en F
si $\{F - \{X \rightarrow A\}\}$ es equivalente a F,
entonces eliminar $X \rightarrow A$ de F.

Para ilustrar el algoritmo anterior, podemos utilizar lo siguiente:

Partiendo del siguiente conjunto de DF, $E : \{B \rightarrow A, D \rightarrow A, AB \rightarrow D\}$, tenemos que localizar su cobertura mínima.

- Todas las dependencias anteriores están en forma canónica; ya hemos completado el paso 1 del Algoritmo 10.2 y podemos proceder con el paso 2. En este paso necesitamos determinar si $AB \rightarrow D$ tiene algún atributo redundante en el lado izquierdo, es decir, ¿puede sustituirse por $B \rightarrow D$ o $A \rightarrow D$?
- Ya que B $\rightarrow$ A, aumentando con B en ambos lados (RI2), tenemos que $BB \rightarrow AB$, o $B \rightarrow AB$ (i). Sin embargo, $AB \rightarrow D$ como se ha dado (ii).
- Por la regla transitiva (RI3), obtenemos de (i) y (ii), $B \rightarrow D$. Por consiguiente $AB \rightarrow D$ podría sustituirse por $B \rightarrow D$.
- Ahora tenemos un conjunto equivalente al E original, el $E' : \{B \rightarrow A, D \rightarrow A, B \rightarrow D\}$. Ya no son posibles más reducciones en el paso 2, ya que todas las DF tienen un único atributo en el lado izquierdo.
- En el paso 3 buscamos una DF redundante en E'. Usando la regla transitiva en $B \rightarrow D$ y $D \rightarrow A$, derivamos $B \rightarrow A$. Por tanto, $B \rightarrow A$ es redundante en E' y puede eliminarse.
- Así pues, la cobertura mínima de E es $\{B \rightarrow D, D \rightarrow A\}$.

En el Capítulo 11 veremos cómo pueden sintetizarse las relaciones de un conjunto de dependencias E localizando primero la cobertura mínima F para E.

10.3 Formas normales basadas en claves principales

Una vez estudiadas las dependencias funcionales y algunas de sus propiedades, estamos preparados para usarlas a la hora de especificar algunos aspectos de la semántica de un esquema de relaciones. Asumimos que contamos con un conjunto de dependencias funcionales para cada relación, y que cada una de estas relaciones dispone de una clave principal; esta información combinada con las verificaciones (condiciones) de las formas normales conduce al *proceso de normalización* del diseño del esquema relacional. Los proyectos de diseño relacional más prácticos siguen una de estas aproximaciones:

- Realizar un diseño de esquema conceptual usando un modelo conceptual como el ER o el EER y asignar el diseño conceptual a un conjunto de relaciones.
- Diseñar las relaciones en base a un conocimiento externo derivado de una implementación existente de ficheros o formularios o informes.

Siguiendo alguno de estos métodos, es útil evaluar las relaciones de idoneidad y descomponerlas todo lo necesario hasta obtener formas normales elevadas usando la teoría de normalización presentada en este capítulo y en el siguiente. En esta sección nos centraremos en las tres primeras formas normales de un esquema de relaciones y en la intuición que se esconde tras ellas, y comentaremos el modo en que fueron desarrolladas históricamente. En la Sección 10.4 veremos las definiciones más generales de estas formas normales, las cuales tienen en cuenta todas las claves candidatas de una relación en lugar de considerar sólo la clave principal.

Empezaremos tratando de manera informal las formas normales y la motivación que se esconde tras su desarrollo, sin olvidarnos de revisar algunas de las definiciones propuestas en el Capítulo 5 y que son necesarias aquí. A continuación trataremos la primera forma normal (1FN) en la Sección 10.3.4, y presentaremos las definiciones de la segunda (2FN) y tercera (3FN) formas normales, que están basadas en claves principales, en las Secciones 10.3.5 y 10.3.6, respectivamente.

10.3.1 Normalización de relaciones

El proceso de normalización, tal y como fue propuesto en un principio por Codd (1972a), hace pasar un esquema de relación por una serie de comprobaciones para *certificar* que satisface una determinada **forma normal**. El proceso, que sigue un método descendente evaluando cada relación contra el criterio de las formas normales y descomponiendo las relaciones según sea necesario, puede considerarse como un *diseño relacional por análisis*. Inicialmente, Codd propuso tres formas normales: la primera, la segunda y la tercera. Una definición más estricta de la 3FN, llamada BCNF (Forma normal Boyce-Codd, *Boyce-Codd Normal Form*) fue propuesta posteriormente por Boyce y Codd. Todas estas formas normales estaban basadas en una única herramienta analítica: las dependencias funcionales entre los atributos de una relación. Más adelante se propusieron una cuarta (4FN) y una quinta (5FN) formas normales basadas en los conceptos de dependencias multivalor y dependencias de concatenación, respectivamente; ambas se explican en el Capítulo 11. Al final de dicho capítulo, comentaremos también el modo en que las relaciones 3FN pueden sintetizarse a partir de un conjunto de DFs dado. Este acercamiento recibe el nombre de *diseño relacional por síntesis*.

La **normalización de datos** puede considerarse como un proceso de análisis de un esquema de relación, basado en sus DF y sus claves principales, para obtener las propiedades deseables de (1) minimizar la redundancia y (2) minimizar las anomalías de inserción, borrado y actualización comentadas en la Sección 10.1.2. Los esquemas de relación no satisfactorios que no cumplen ciertas condiciones (las **pruebas de forma normal**) se decomponen en esquemas de relación más pequeños que cumplen esas pruebas y que, por consiguiente, cuentan con las propiedades deseables. De este modo, el procedimiento de normalización ofrece a los diseñadores de bases de datos lo siguiente:

- Un marco formal para el análisis de los esquemas de relación basado en sus claves y en las dependencias funcionales entre sus atributos.
- Una serie de pruebas de forma normal que pueden efectuarse sobre esquemas de relación individuales, de modo que la base de datos relacional pueda **normalizarse** hasta el grado deseado.

  Definición. La **forma normal** de una relación hace referencia a la forma normal más alta que cumple, e indica por tanto el grado al que ha sido normalizada.

Las formas normales, consideradas *aisladas* de otros factores, no garantizan un buen diseño de base de datos. Generalmente, no basta con comprobar por separado que cada esquema de relación de la base de datos está, digamos, en BCNF o en 3FN. En lugar de ello, el proceso de normalización por descomposición debe confirmar también la existencia de las propiedades adicionales que los esquemas relacionales, en conjunto, deben poseer. Dos de estas propiedades son las siguientes:

- La **propiedad de reunión sin pérdida** o **reunión no aditiva**, que garantiza que no se presentará el problema de las tuplas falsas comentado en la Sección 10.1.4, respecto a los esquema de relación creados después de la descomposición.
- La **propiedad de conservación de las dependencias**, que asegura que todas las dependencias funcionales están representadas en alguna relación individual resultante tras la descomposición.

La propiedad de reunión no aditiva es extremadamente crítica y **debe cumplirse a cualquier precio**, mientras que la de conservación de las dependencias, aunque deseable, puede sacrificarse a veces, tal y como veremos en la Sección 11.1.2. Aplazaremos al Capítulo 11 la presentación de los conceptos formales y las técnicas que garantizan las dos propiedades anteriores.

10.3.2 Uso práctico de las formas normales

Los proyectos de diseño más prácticos obtienen información de diseños de bases de datos previos, de modelos heredados o de ficheros existentes. La normalización se lleva a la práctica de forma que los diseños resultantes sean de máxima calidad y cumplan las propiedades deseables antes comentadas. Aunque se han

definido varias formas normales superiores, como la 4FN y la 5FF que veremos en el Capítulo 11, su utilidad práctica es cuestionable cuando las restricciones en las que se basan son difíciles de comprender o detectar por parte de los diseñadores de la base de datos y son los usuarios los que deben descubrirlas. Así, el diseño de base de datos, tal y como se realiza en la actualidad en la industria, presta especial atención a la normalización hasta la 3FN, la BCNF o la 4FN.

Otro punto que conviene resaltar es que los diseñadores de bases de datos *no necesitan* normalizar hasta la forma normal más alta posible. Por razones de rendimiento, las relaciones podrían dejarse en un estado de normalización menor, como el 2FN, como se ha comentado al final de la Sección 10.1.2.

> **Definición.** El proceso para almacenar la concatenación de relaciones de forma normal superiores como relación base (que se encuentra en una forma normal inferior) recibe el nombre de **desnormalización**.

10.3.3 Definiciones de claves y atributos que participan en las claves

Antes de avanzar más, vamos a revisar de nuevo las definiciones de las claves de un esquema de relación mostradas en el Capítulo 5.

> **Definición.** Una **superclave** de un esquema de relación $R = \{A_1, A_2, \ldots, A_n\}$ es un conjunto de atributos $S \subseteq R$ con la propiedad de que no habrá un par de tuplas t_1 y t_2 en ningún estado de relación permitido r de R tal que $t_1[S] = t_2[S]$. Una **clave** K es una superclave con la propiedad adicional de que la eliminación de cualquier atributo de K provocará que K deje de ser una superclave.

La diferencia entre una clave y una superclave es que la primera tiene que ser mínima, es decir, si tenemos una clave $K = \{A_1, A_2, \ldots, A_k\}$ de R, entonces $K - \{A_i\}$ no es una clave de R para ningún A_i, 1 ? i ? k. En la Figura 10.1, {Dni} es una clave de EMPLEADO, mientras que {Dni}, {Dni, NombreE}, {Dni, NombreE, FechaNac} y cualquier otro conjunto de atributos que incluya Dni, son superclaves.

Si un esquema de relación tiene más de una clave, cada una de ellas se denomina **clave candidata**. Una de ellas se elige *arbitrariamente* como **clave principal**, mientras que el resto son claves secundarias. Todo esquema de relación debe contar con una clave principal. En la Figura 10.1, {Dni} es la única clave candidata de EMPLEADO, por lo que también será la clave principal.

> **Definición.** Un atributo del esquema de relación R recibe el nombre de **atributo primo** de R si es miembro de *alguna de las claves candidatas* de R. Un atributo es **no primo** si no es miembro de ninguna clave candidata.

En la Figura 10.1, tanto Dni como NúmeroDpto son atributos primos de TRABAJA_EN, mientras que el resto de atributos de TRABAJA_EN no lo son.

Ahora vamos a presentar las tres primeras formas normales: 1FN, 2FN y 3FN. Todas ellas fueron propuestas por Codd (1972a) como una secuencia para alcanzar el estado deseable de relaciones 3FN tras pasar por los estados intermedios 1FN y 2FN en caso de ser necesario. Como veremos, la 2FN y la 3FN atacan diferentes problemas. Sin embargo, por motivos históricos, es habitual seguirlas en ese orden; así pues, por definición, una relación 3FN *también satisface* la 2FN.

10.3.4 Primera forma normal

La **primera forma normal (1FN)** está considerada como una parte de la definición formal de una relación en el modelo relacional básico;[13] históricamente, fue definida para prohibir los atributos multivalor, los

[13] Esta condición desaparece en el modelo relacional anidado y en los ORDBMS (*Sistemas de objetos relacionales, Object-Relational Systems*), los cuales permiten relaciones no normalizadas (consulte el Capítulo 22).

atributos compuestos y sus combinaciones. Afirma que el dominio de un atributo sólo debe incluir valores *atómicos* (simples, indivisibles) y que el valor de cualquier atributo en una tupla debe ser un *valor simple* del dominio de ese atributo. Por tanto, 1FN prohíbe tener un conjunto de valores, una tupla de valores o una combinación de ambos como valor de un atributo para una *tupla individual*. En otras palabras, 1FN prohíbe las *relaciones dentro de las relaciones* o las *relaciones como valores de atributo dentro de las tuplas*. Los únicos valores de atributo permitidos por 1FN son los **atómicos** (o **indivisibles**).

Considere el esquema de relación DEPARTAMENTO de la Figura 10.1, cuya clave principal es NúmeroDpto, y suponga que lo ampliamos incluyendo el atributo UbicacionesDpto [véase la Figura 10.8(a)]. Asumimos que cada departamento puede tener *un número de* localizaciones. En la Figura 10.8 podemos ver el esquema DEPARTAMENTO y un estado de relación de ejemplo. Como puede apreciarse, esto no está en la 1FN porque UbicacionesDpto no es un atributo atómico, como queda demostrado por la primera tupla de la Figura 10.8(b). Hay dos formas de ver el atributo UbicacionesDpto:

- El dominio de UbicacionesDpto contiene valores atómicos, pero algunas tuplas pueden tener un conjunto de estos valores. En este caso, UbicacionesDpto no es funcionalmente dependiente de la clave principal NúmeroDpto.
- El dominio de UbicacionesDpto contiene conjuntos de valores y, por tanto, no es atómico. En este caso, NúmeroDpto → UbicacionesDpto porque cada conjunto está considerado como un miembro único de un dominio de atributo.[14]

En cualquier caso, la relación DEPARTAMENTO de la Figura 10.8 no está en 1FN; de hecho, no está calificada ni como relación según nuestra definición de relación de la Sección 5.1. Existen tres formas principales de alcanzar la primera forma normal para una relación como ésta:

1. Eliminar el atributo UbicacionesDpto que viola la 1FN y colocarlo en una relación aparte LOCALIZACIONES_DPTO junto con la clave principal NúmeroDpto de DEPARTAMENTO. La clave principal de esta relación es la combinación {NúmeroDpto, UbicaciónDpto}, como puede verse en la Figura 10.2. En LOCALIZACIONES_DPTO existe una tupla distinta por *cada localización* de un departamento. Esto descompone la no relación 1FN en dos relaciones 1FN.
2. Expandir la clave de forma que exista una tupla separada en la relación DEPARTAMENTO original por cada localización de un DEPARTAMENTO, como puede verse en la Figura 10.8(c). En este caso, la clave principal resulta de la combinación {NúmeroDpto, UbicaciónDpto}. Esta solución tiene la desventaja de introducir *redundancia* en la relación.
3. Si se conoce un *número máximo de valores* para el atributo (por ejemplo, si se sabe que *pueden existir, al menos, tres localizaciones* para un departamento), sustituir el atributo UbicacionesDpto por tres atributos atómicos: UbicaciónDpto1, UbicaciónDpto2 y UbicaciónDpto3. Esta solución tiene el inconveniente de introducir *valores NULL* en el caso de que algún departamento tenga menos de tres localizaciones. Introduce además una semántica confusa acerca de la ordenación entre los valores de localización, lo que no era lo que originalmente se pretendía. La consulta sobre este atributo se hace más complicada; por ejemplo, considere cómo escribir la siguiente consulta: *enumere los departamentos que tienen 'Valencia' como una de sus localizaciones.*

De las tres posibles soluciones, la primera es la que se considera como la mejor porque no introduce redundancia y es completamente general, sin estar limitada por un número máximo de valores. De hecho, si elegimos la segunda solución, el proceso posterior de normalización nos conducirá a la primera solución.

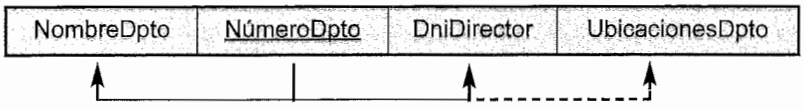
La primera forma normal también inhabilita los atributos multivalor que son compuestos. Reciben el nombre de **relaciones anidadas** porque cada tupla puede tener una relación *dentro de ella.* La Figura 10.9 muestra a

[14] En este caso, podemos considerar que el dominio de UbicacionesDpto es el **conjunto potencia** del conjunto de localizaciones individuales, es decir, el dominio está compuesto por todos los posibles subconjuntos del conjunto de ubicaciones individuales.

Figura 10.8. Normalización en 1FN. (a) Un esquema de relación que no está en 1FN. (b) Ejemplo de un estado de relación DEPARTAMENTO. (c) Versión 1FN de la misma relación con redundancia.

(a)

DEPARTAMENTO



(b)

DEPARTAMENTO

NombreDpto	NúmeroDpto	DniDirector	UbicacionesDpto
Investigación	5	333445555	{Valencia, Sevilla, Madrid}
Administración	4	987654321	{Gijón}
Sede central	1	888665555	{Madrid}

(c)

DEPARTAMENTO

NombreDpto	NúmeroDpto	DniDirector	UbicaciónDpto
Investigación	5	333445555	Valencia
Investigación	5	333445555	Sevilla
Investigación	5	333445555	Madrid
Administración	4	987654321	Gijón
Sede central	1	888665555	Madrid

qué podría parecerse la relación EMP_PROY en caso de permitirse la anidación. Cada tupla representa una entidad empleado, y una relación PROYS(NumProyecto, Horas) *dentro de cada tupla* implica los proyectos de cada empleado y las horas semanales dedicadas a cada proyecto. El esquema de esta relación EMP_PROY puede representarse de la siguiente manera:

EMP_PROY(Dni, NombreE, {PROYS(NumProyecto, Horas)})

Las llaves { } identifican el atributo PROYS como de tipo multivalor, y enumeramos los atributos componente que forman PROYS entre paréntesis ( ). Las últimas tendencias para el soporte de objetos complejos (consulte el Capítulo 20) y datos XML (consulte el Capítulo 27) intentan permitir y formalizar relaciones anidadas dentro de sistemas de bases de datos relacionales, las cuales fueron prohibidas en principio por la 1FN.

Observe que Dni es la clave principal de la relación EMP_PROY en las Figuras 10.9(a) y (b), mientras que NumProyecto es la clave **parcial** de la relación anidada, esto es, dentro de cada tupla, la relación anidada debe contar con valores únicos de NumProyecto. Para normalizar esto en 1FN, llevamos los atributos de la relación anidada a una nueva relación y *propagamos la clave principal*; la clave principal de la nueva relación combinará la clave parcial con la clave principal de la relación original. La descomposición y la propagación de la clave principal producen los esquemas EMP_PROY1 y EMP_PROY2 [véase la Figura 10.9(c)].

Este procedimiento puede aplicarse recursivamente a una relación con varios niveles de anidamiento para **desanidar** la relación en un conjunto de relaciones 1FN. Esto resulta útil para convertir un esquema de relación no normalizado con varios niveles de anidamiento en relaciones 1FN. La existencia de más de un atributo multivalor en una relación debe ser manipulado con cuidado. Como ejemplo, considere la siguiente relación no 1FN:

PERSONA (Dn#, {NumPermisoConducir#} , {Teléfono#})

Figura 10.9. Normalización de relaciones anidadas en 1FN. (a) Esquema de la relación EMP_PROY con un atributo de *relación anidada* PROYS. (b) Ejemplo de extensión de la relación EMP_PROY mostrando relaciones anidadas dentro de cada tupla. (c) Descomposición de EMP_PROY en las relaciones EMP_PROY1 y EMP_PROY2 por la propagación de la clave principal.

(a)

EMP_PROY

		Proyectos	
Dni	NombreE	NumProyecto	Horas

(b)

EMP_PROY

Dni	NombreE	NumProyecto	Horas
123456789	Pérez Pérez, José	1 2	32.5 7.5
666884444	Ojeda Ordóñez, Fernando.	3	40.0
453453453	Oliva Avezuela, Aurora	1 2	20.0 20.0
333445555	Campos Sastre, Alberto	2 3 10 20	10.0 10.0 10.0 10.0
999887777	Zelaya, Alicia J.	30 10	30.0 10.0
987987987	Pajares Morera, Luis	10 30	35.0 5.0
987654321	Sainz Oreja, Juana	20 30	20.0 15.0
888665555	Ochoa Paredes, Eduardo	20	NULL

(c)

EMP_PROY1

Dni	NombreE

EMP_PROY2

Dni	NumProyecto	Horas

Esta relación representa el hecho de que una persona tiene varios vehículos y teléfonos. Si se sigue la segunda opción antes comentada, obtendremos una relación repleta de claves:

PERSONA_EN_1FN (Dn#, NumPermisoConducir#, Teléfono#)

Para evitar introducir relaciones extrañas entre el NumPermisoConducir# y Teléfono#, están representadas todas las posibles combinaciones de valores por cada Dn#, dando lugar a redundancias. Esto conduce a los problemas manipulados por las dependencias multivalor y 4FN, de las que hablaremos en el Capítulo 11. La forma correcta de tratar con los dos atributos multivalor de PERSONA es descomponerlos en dos relaciones separadas, usando la primera estrategia comentada anteriormente: P1(Dn#, NumPermisoConducir#) y P2(Dn#, Teléfono#).

10.3.5 Segunda forma normal

La **segunda forma normal** (**2FN**) está basada en el concepto de *dependencia funcional total*. Una dependencia funcional $X \rightarrow Y$ es **total** si la eliminación de cualquier atributo A de X implica que la dependencia deje de ser válida, es decir, para cualquier atributo $A \in X$, $(X - \{A\})$ *no* determina funcionalmente a Y. Una dependencia funcional $X \rightarrow Y$ es **parcial** si al eliminarse algún atributo $A \in X$ de X la dependencia sigue siendo válida, es decir, para algún $A \in X$, $(X - \{A\}) \rightarrow Y$. En la Figura 10.3(b), {Dni, NumProyecto} → Horas es una dependencia completa (ni Dni → Horas ni NumProyecto → Horas son válidas). Sin embargo, la dependencia {Dni, NumProyecto} → NombreE es parcial porque se cumple Dni → NombreE.

> **Definición.** Un esquema de relación R está en 2FN si todo atributo no primo A en R es *completa y funcionalmente dependiente* de la clave principal de R.

La comprobación para 2FN implica la verificación de las dependencias funcionales cuyos atributos del lado izquierdo forman parte de la clave principal. Si ésta contiene un único atributo, no es necesario aplicar la verificación. La relación EMP_PROY de la Figura 10.3(b) está en 1FN pero no en 2FN. El atributo no primo NombreE viola la 2FN debido a DF2, lo mismo que hacen los atributos no primos NombreProyecto y UbicaciónProyecto debido a DF3. Las dependencias funcionales DF2 y DF3 hacen que NombreE, NombreProyecto y UbicaciónProyecto sean parcialmente dependientes de la clave principal {Dni, NumProyecto} de EMP_PROY, violando la comprobación de 2FN.

Si un esquema de relación está en 2FN, puede ser *normalizado en segundo lugar* o *normalizado 2FN* en un número de relaciones 2FN en las que los atributos no primos sólo están asociados con la parte de la clave principal de la que son completa y funcionalmente dependientes. Por consiguiente, las dependencias funcionales FD1, FD2 y FD3 de la Figura 10.3(b) inducen a la descomposición de EMP_PROY en los tres esquemas de relación EP1, EP2 y EP3 mostrados en la Figura 10.10(a), cada uno de los cuales está es 2FN.

10.3.6 Tercera forma normal

La **tercera forma normal** (**3FN**) se basa en el concepto de *dependencia transitiva*. Una dependencia funcional $X \rightarrow Y$ en un esquema de relación R es una **dependencia transitiva** si existe un conjunto de atributos Z que ni es clave candidata ni un subconjunto de ninguna clave de R,[15] y se cumple tanto $X \rightarrow Z$ como $Z \rightarrow Y$. La dependencia Dni → DniDirector es transitiva a través de NúmeroDpto en EMP_DEPT en la Figura 10.3(a) porque se cumplen las dependencias Dni → NúmeroDpto y NúmeroDpto → DniDirector *y* NúmeroDpto no es una clave por sí misma ni un subconjunto de la clave de EMP_DEPT. Intuitivamente, podemos ver que la dependencia de DniDirector en NúmeroDpto no es deseable en EMP_DEPT ya que NúmeroDpto no es una clave de EMP_DEPT.

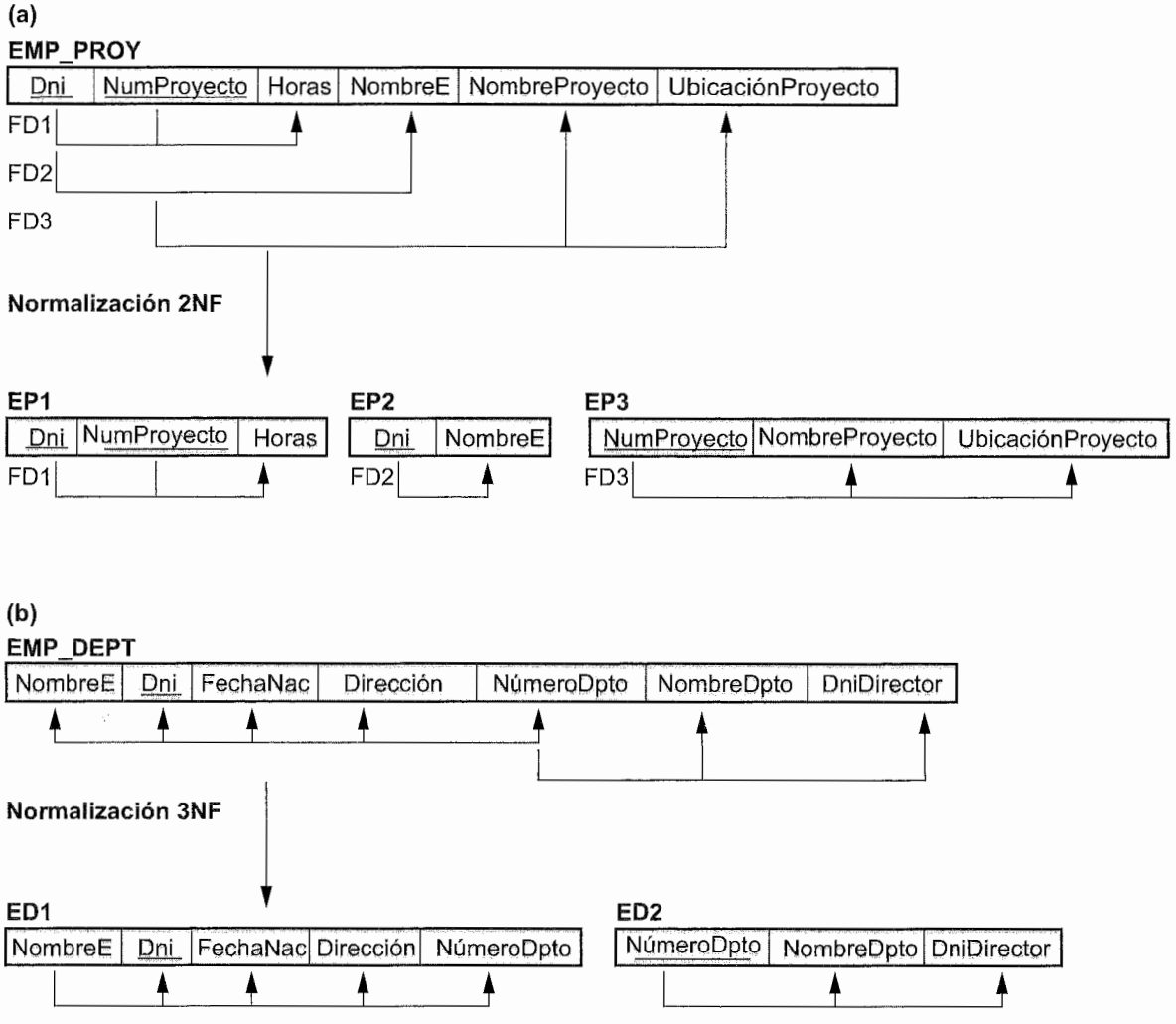
> **Definición.** Según la definición original de Codd, un esquema de relación R está en **3FN** si satisface 2FN *y* ningún atributo no primo de R es transitivamente dependiente en la clave principal.

El esquema de relación EMP_DEPT de la Figura 10.3(a) está en 2FN, ya que no existen dependencias no parciales en una clave. Sin embargo, EMP_DEPT no está en 3FN debido a la dependencia transitiva de DniDirector (y también de NombreDpto) en Dni a través de NúmeroDpto. Podemos normalizar EMP_DEPT descomponiéndola en los dos esquemas de relación 3FN ED1 y ED2 mostrados en la Figura 10.10(b). Intuitivamente, vemos que ED1 y ED2 representan entidades independientes de empleados y departamentos. Una operación CONCATENACIÓN NATURAL en ED1 y ED2 recuperará la relación EMP_DEPT original sin generar tuplas falsas.

Intuitivamente podemos ver que cualquier dependencia funcional en la que el lado izquierdo es parte de la clave principal (subconjunto propio), o cualquier dependencia funcional en la que el lado izquierdo es un

[15] Ésta es la definición general de una dependencia transitiva. Ya que en esta sección sólo nos interesan las claves principales, permitimos las dependencias transitivas donde X es la clave principal pero Z podría ser (un subconjunto de) una clave candidata.

Figura 10.10. Normalización en 2FN y 3FN. (a) Normalizando EMP_PROY en relaciones 2FN. (b) Normalizando EMP_DEPT en relaciones 3FN.



atributo no clave, implica una DF *problemática*. La normalización 2FN y 3FN elimina estas DFs descomponiendo la relación original en nuevas relaciones. En términos del proceso de normalización, no es necesario quitar las dependencias parciales antes que las dependencias transitivas, pero históricamente, la 3FN se ha definido con la presunción de que una relación se ha verificado antes para 2FN que para 3FN. La Tabla 10.1 resume de manera informal las tres formas normales basándose en sus claves principales, las pruebas empleadas en cada caso y el *remedio* correspondiente, o la normalización realizada para conseguir la forma normal.

10.4 Definiciones generales de la segunda y tercera formas normales

En general, diseñaremos nuestro esquema de relaciones de forma que no tengan ni dependencias parciales ni transitivas porque causan las anomalías de actualización comentadas en la Sección 10.1.2. Los pasos para la normalización en relaciones 3FN que hemos visto hasta ahora imposibilitan las dependencias parciales y

Tabla 10.1. Resumen de las formas normales en función a sus claves principales y la normalización correspondiente.

Forma normal	Prueba	Remedio (normalización)
Primera (1FN)	La relación no debe tener atributos multivalor o relaciones anidadas.	Generar nuevas relaciones para cada atributo multivalor o relación anidada.
Segunda (2FN)	Para relaciones en las que la clave principal contiene varios atributos, un atributo no clave debe ser funcionalmente dependiente en una parte de la clave principal.	Descomponer y configurar una nueva relación por cada clave parcial con su(s) atributo(s) dependiente(s). Asegurarse de mantener una relación con la clave principal original y cualquier atributo que sea completa y funcionalmente dependiente de ella.
Tercera (3FN)	La relación no debe tener un atributo no clave que esté funcionalmente determinado por otro atributo no clave (o por un conjunto de atributos no clave). Esto es, debe ser una dependencia transitiva de un atributo no clave de la clave principal.	Descomponer y configurar una relación que incluya el(los) atributo(s) no clave que determine(n) funcionalmente otro(s) atributo(s) no clave.

transitivas en la *clave principal*. Estas definiciones, sin embargo, no tienen en cuenta otras claves candidatas de una relación, en caso de que existan. En esta sección vamos a ver unas definiciones más generales de las formas 2FN y 3FN que sí consideran *todas* esas claves candidatas. Observe que esto no afecta a la definición de la 1FN ya que es independiente de claves y dependencias funcionales. Un **atributo primo** es, de forma general, un atributo que forma parte de *cualquier clave candidata*. Ahora consideraremos las dependencias funcionales y transitivas *respecto a todas las claves candidatas* de una relación.

10.4.1 Definición general de la segunda forma normal

> **Definición.** Un esquema de relación R está en **segunda forma normal (2FN)** si cada atributo no primo A en R no es parcialmente dependiente de *ninguna* clave de R.[16]

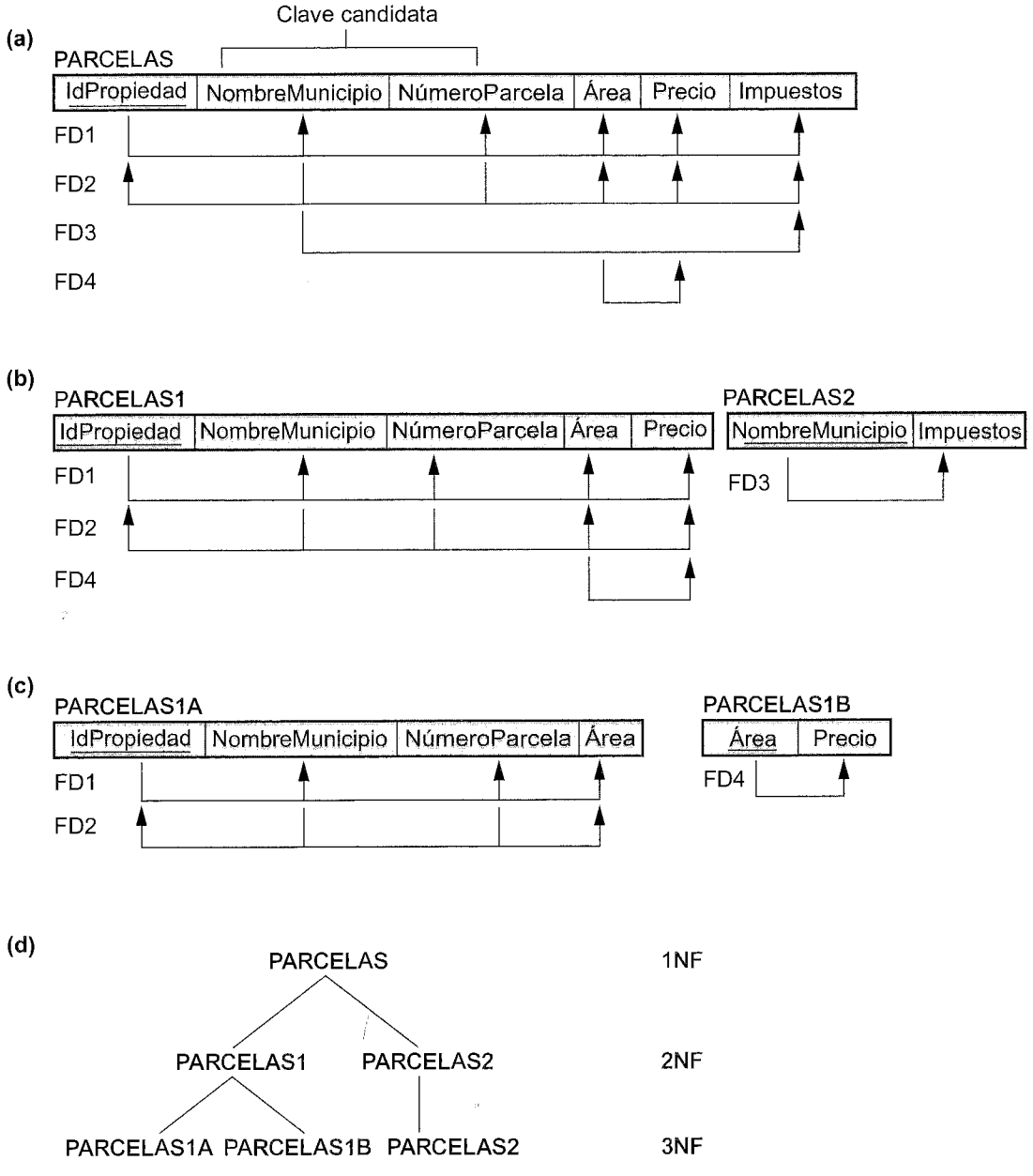
La prueba para 2FN implica la verificación de las dependencias funcionales cuyos atributos de la izquierda forman *parte de* la clave principal. Si la clave principal contiene un único atributo, no es preciso aplicar la comprobación a todo. Considere el esquema de relación PARCELAS de la Figura 10.11(a), que describe los terrenos en venta en distintas provincias. Suponga que existen dos claves candidatas: IdPropiedad y {NombreMunicipio, NúmeroParcela}, es decir, los números de parcela son únicos en cada provincia, mientras que los IdPropiedad son únicos para toda una provincia.

En base a las dos claves candidatas IdPropiedad y {NombreMunicipio, NúmeroParcela}, sabemos que se cumplen las dependencias funcionales FD1 y FD2 de la Figura 10.11(a). Elegimos IdPropiedad como clave principal, razón por la que aparece subrayada en la Figura 10.11(a), aunque no es preciso tomar ninguna consideración especial con esta clave en relación con la otra clave candidata. Supongamos que también se cumplen las dos siguientes dependencias funcionales en PARCELAS:

FD3: NombreProvincia → Impuestos
FD4: Área → Precio

[16] Esta definición puede redefinirse de la siguiente forma: un esquema de relación R está en 2FN si cada atributo no primo A en R es completa y funcionalmente dependiente en *cada* clave de R.

Figura 10.11. Normalización en 2FN y 3FN. (a) La relación PARCELAS con sus dependencias funcionales de la FD1 a la FD4. (b) Descomposición en la relaciones 2FN PARCELAS1 y PARCELAS2. (c) Descomposición de PARCELAS1 en las relaciones 3FN PARCELAS1A y PARCELAS1B. (d) Resumen de la normalización progresiva de PARCELAS.



La dependencia FD3 dice que los impuestos son fijos para cada municipio (no varían de una parcela a otra en el mismo municipio), mientras que FD4 dice que el precio de una parcela lo determina su área sin tener en cuenta el municipio en el que se encuentre (asumimos que éste es el precio de la parcela por motivos fiscales).

El esquema de relación PARCELAS viola la definición general de 2FN porque Impuestos es parcialmente dependiente de la clave candidata {NombreMunicipio, NúmeroParcela}, debido a FD3. Para normalizar

PARCELAS a 2FN, la descomponemos en las dos relaciones PARCELAS1 y PARCELAS2, mostradas en la Figura 10.11(b). Construimos PARCELAS1 eliminando de PARCELAS el atributo Impuestos que viola 2FN y colocándolo junto a NombreProvincia (el lado izquierdo de FD3 que provoca la dependencia parcial) en otra relación PARCELAS2. Tanto PARCELAS1 como PARCELAS2 están en 2FN. Observe que FD4 no viola 2FN y persiste en PARCELAS1.

10.4.2 Definición general de la tercera forma normal

Definición. Un esquema de relación R está en **tercera forma normal (3FN)** si, siempre que una dependencia funcional *no trivial* $X \rightarrow A$ se cumple en R, ya sea (a) X una superclave de R, o (b) A un atributo primo de R.

Según esta definición, PARCELAS2 (Figura 10.11(b)) está en 3FN. Sin embargo, FD4 de PARCELAS1 viola 3FN porque Área no es una superclave y Precio no es un atributo primo en PARCELAS1. Para normalizar PARCELAS1 a 3FN, la descomponemos en los esquemas de relación PARCELAS1A y PARCELAS1B de la Figura 10.11(c). Construimos PARCELAS1A eliminando el atributo Precio de PARCELAS1, que viola 3FN, y colocándolo junto con Área (el lado izquierdo de FD4 que provoca la dependencia transitiva) en otra relación PARCELAS1B. Tanto PARCELAS1A como PARCELAS1B están en 3FN.

Dos son los puntos a destacar en este ejemplo y en la definición general de la 3FN:

- PARCELAS1 viola 3FN porque Precio depende transitivamente de cada una de las claves candidatas de PARCELAS1 a través del atributo no primo Área.
- Esta definición general puede aplicarse *directamente* para comprobar si un esquema de relación está en 3FN; esto no implica pasar primero por 2FN. Si aplicamos la definición anterior de 3FN a PARCELAS con las dependencias FD1 a FD4, encontramos que *tanto* FD3 como FD4 violan 3FN. Por consiguiente, podríamos descomponer PARCELAS en PARCELAS1A, PARCELAS1B y PARCELAS2 directamente. Por consiguiente, las dependencias transitiva y parcial que violan 3FN pueden eliminarse *en cualquier orden.*

10.4.3 Interpretando la definición general de la tercera forma normal

Un esquema de relación R viola la definición general de 3FN si una dependencia funcional $X \rightarrow A$ se cumple en R y viola las *dos* condiciones (a) y (b) de la 3FN. La violación del apartado (b) significa que A es un atributo no primo, mientras que la vulneración del (a) implica que X no es una superclave de ninguna clave de R; por consiguiente, X podría ser no primo o ser un subconjunto propio de una clave de R. Si X es no primo, lo que tenemos es una dependencia transitiva que viola 3FN, mientras que si X es un subconjunto propio de una clave de R, lo que aparece es una dependencia parcial que viola 3FN (y también 2FN). Por tanto, podemos declarar una **definición alternativa general de 3FN** de la siguiente manera:

Definición alternativa. Un esquema de relación R está en 3FN si cada atributo no primo de R cumple las siguientes condiciones:

- Es completa y funcionalmente dependiente de cada clave de R.
- No depende transitivamente de cada clave de R.

10.5 Forma normal de Boyce-Codd

La **BCNF (Forma normal de Boyce-Codd, *Boyce-Codd Normal Form*)** se propuso como una forma más simple de la 3FN, aunque es más estricta que ésta. Es decir, toda relación que esté en BCNF lo está también

en 3FN; sin embargo, una relación 3FN *no está necesariamente* en BCNF. Intuitivamente, podemos ver la necesidad de una forma normal más estricta que la 3FN si volvemos al esquema de relación PARCELAS de la Figura 10.11(a), la cual tiene cuatro dependencias funcionales: de la FD1 a la FD4. Supongamos que tenemos cientos de parcelas en la relación, pero que sólo están en dos municipios: Getafe y Alcorcón. Supongamos también que el tamaño de las parcelas de Alcorcón es de sólo 0,5; 0,6; 0,7; 0,8; 0,9 y 1 hectárea, mientras que las de Getafe están restringidas a 1,1 y 2 hectáreas. En una situación como ésta deberemos contar con una dependencia funcional adicional FD5: Área → NombreMunicipio. Si queremos añadirla al resto de dependencias, el esquema de relación PARCELAS1A seguirá estando en 3FN porque NombreMunicipio es un atributo primo.

El área de una parcela que determina el municipio, según se especifica en la FD5, puede representarse mediante 16 tuplas en una relación separada R(Área, NombreMunicipio), ya que sólo existen 16 posibles valores de Área. Esta representación reduce la redundancia de tener que repetir la misma información en los miles de tuplas PARCELAS1A. La BCNF es una *forma normal más estricta* que prohibiría PARCELAS1A y sugeriría la necesidad de descomponerla.

> **Definición**. Un esquema de relación R está en **BCNF** si siempre que una dependencia funcional *no trivial* $X \rightarrow A$ se cumple en R, entonces X es una superclave de R.

La definición formal de BCNF difiere ligeramente de la de 3FN. La única diferencia entre ellas es la condición (b) de 3FN, la cual permite que A sea primo, lo que no se consiente en BCNF. En nuestro ejemplo, FD5 viola BCNF en PARCELAS1A porque ÁREA no es una superclave de PARCELAS1A. Observe que FD5 satisface la 3FN en PARCELAS1A porque NombreMunicipio es un atributo primo (condición b), pero esta condición no existe en la definición de BCNF. Podemos descomponer PARCELAS1A en las dos relaciones BCNF PARCELAS1AX y PARCELAS1AY mostradas en la Figura 10.12(a). Esta descomposición pierde la dependencia funcional FD2 porque sus atributos no coexisten en la misma relación tras la descomposición.

En la práctica, casi todos los esquemas de relación que están en 3FN lo están también en BCNF. Sólo si se cumple $X \rightarrow A$ en un esquema de relación R, no siendo X una superclave y siendo A un atributo primo,

Figura 10.12. Forma normal de Boyce-Codd. (a) Normalización BCNF de PARCELAS1A en la que se pierde la dependencia funcional FD2 en la descomposición. (b) Una relación esquemática con FDs; está en 3FN pero no en BCNF.

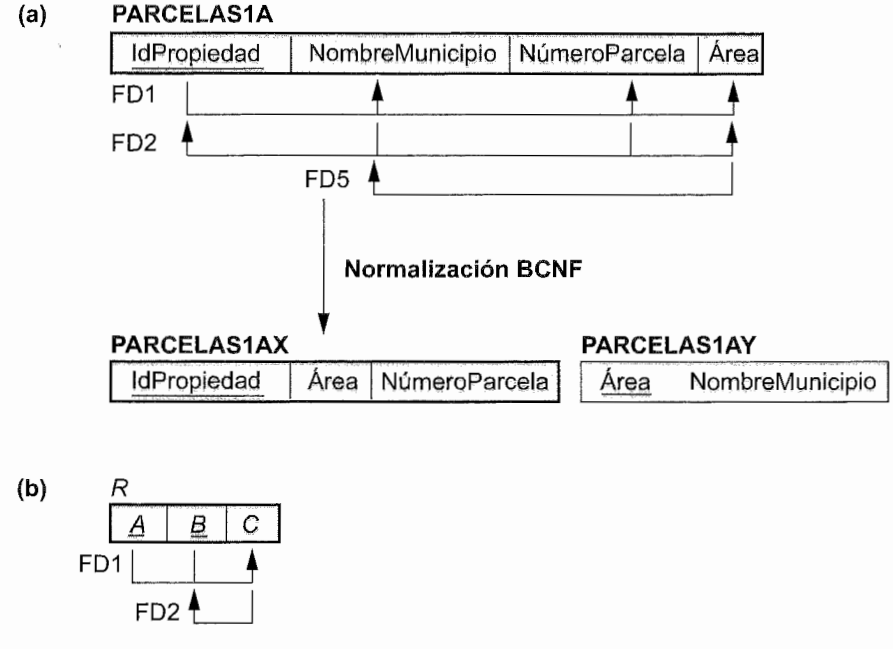


Figura 10.13. Una relación ENSEÑAR que está en 3FN pero no en BCNF.

Estudiante	Curso	Profesor
Campos	Bases de datos	Marcos
Pérez	Bases de datos	María
Pérez	Sistemas operativos	Amanda
Pérez	Teoría	Sergio
Ochoa	Bases de datos	Marcos
Ochoa	Sistemas operativos	Aurora
Morera	Bases de datos	Eduardo
Celaya	Bases de datos	María
Campos	Sistemas operativos	Amanda

estará en 3FN pero no en BCNF. El esquema de relación *R* mostrado en la Figura 10.12(b) ilustra el caso general de una relación como ésta. De forma ideal, el diseño de una base de datos relacional debe afanarse por cumplir la BCNF o la 3FN en cada esquema de relación. Alcanzar sólo la normalización 1FN o 2FN no se considera adecuado, ya que fueron desarrolladas históricamente como una pasarela hacia la 3FN y la BCNF.

La Figura 10.13 es otro ejemplo que muestra una relación ENSEÑAR con las siguientes dependencias:

FD1: {Estudiante, Curso} → Profesor
FD2:[17] Profesor → Curso

Observe que {Estudiante, Curso} es una clave candidata para esta relación y que las dependencias mostradas siguen el patrón de la Figura 10.12(b), con Estudiante como *A*, Curso como *B* y Profesor como *C*. Por consiguiente, esta relación está en 3FN pero no en BCNF. La descomposición de este esquema de relación en dos esquemas no es muy correcta porque puede dividirse en uno de los tres siguientes pares:

1. {Estudiante, Profesor} y {Estudiante, Curso}.
2. {Curso, Profesor} y {Curso, Estudiante}.
3. {Profesor, Curso} y {Profesor, Estudiante}.

Las tres descomposiciones *pierden* la dependencia funcional FD1. De ellas, la *descomposición deseable* es la número 3 porque no generará tuplas falsas tras una concatenación.

Una prueba para determinar si una descomposición es no aditiva (sin pérdida) se explica en la Sección 11.1.4 bajo la Propiedad NJB. En general, una relación que no está en BCNF debería descomponerse para cumplir esta propiedad, a la vez que se renuncia a preservar todas las dependencias funcionales en las relaciones resultantes, como ocurre en este ejemplo. El Algoritmo 11.3 hace esto y debería emplearse antes que la descomposición 3 de ENSEÑAR, la cual produce dos relaciones en BCNF como éstas:

(Profesor, Curso) y (Profesor, Estudiante)

Observe que si designamos (Profesor, Estudiante) como clave principal de la relación ENSEÑAR, la DF Profesor → Curso provoca una dependencia parcial (no completamente funcional) de Curso en una parte de esta clave. Esta DF podría eliminarse como parte de una segunda normalización generando exactamente las

[17] Esta dependencia significa que *cada profesor que enseña un curso* es una restricción para esta aplicación.

dos mismas relaciones. Esto es un ejemplo de cómo alcanzar el mismo diseño BCNF a través de rutas de normalización alternativas.

10.6 Resumen

En este Capítulo hemos tratado varios de los peligros a los que nos podemos enfrentar a la hora de diseñar bases de datos relacionales usando argumentos intuitivos. Identificamos de manera informal algunas de las medidas para indicar si un esquema de relación es *idóneo* o *inadecuado*, y facilitamos algunas líneas maestras informales para un buen diseño. Estas directrices están basadas en la realización de un cuidadoso diseño conceptual en el modelo ER y EER, siguiendo correctamente el procedimiento de asignación del Capítulo 7 para asociar entidades y relaciones. Una ejecución correcta de estas directrices, y la ausencia de redundancia, evitará las anomalías en la inserción/borrado/actualización, y la generación de datos falsos. Recomendamos limitar los valores NULL que causan problemas durante las operaciones SELECCIÓN, CONCATENACIÓN y de agregación. A continuación presentamos algunos conceptos formales que nos permiten realizar diseños relacionales de forma descendente mediante el análisis individual de la relaciones. Definimos este proceso de diseño mediante el análisis y la descomposición introduciendo el proceso de normalización.

Definimos el concepto de dependencia funcional y comentamos algunas de sus propiedades. Las dependencias funcionales especifican restricciones semánticas entre los atributos de un esquema de relación. Mostramos cómo se pueden inferir dependencias adicionales de un conjunto de dependencias funcionales usando reglas de inferencia. Definimos los conceptos de clausura y cubierta en relación con las dependencias funcionales. Después mostramos qué es la cubierta mínima de un conjunto de dependencias y ofrecimos un algoritmo para calcularla. También enseñamos el modo de verificar si dos conjuntos de dependencias funcionales son equivalentes.

A continuación describimos el proceso de normalización para lograr buenos diseños mediante la verificación de las relaciones. Ofrecimos un tratamiento de normalización sucesiva basado en una clave principal predefinida en cada relación, y después "aflojamos" este requisito y mostramos una serie de definiciones más generales de la segunda (2FN) y tercera (3FN) formas normales que tienen en cuenta todas las claves candidatas de una relación. Presentamos ejemplos que ilustran cómo, utilizando la definición general de la 3FN, una relación concreta puede analizarse y descomponerse para, eventualmente, producir un conjunto de relaciones en 3FN.

Finalmente, presentamos la BCNF (Forma normal de Boyce-Codd, *Boyce-Codd Normal Form*) y explicamos que se trata de una forma más estricta que la 3FN. Ilustramos también cómo la descomposición de una relación que no está en BCNF debe realizarse considerando los requisitos de descomposición no aditiva. El Capítulo 11 presenta los algoritmos de síntesis y descomposición para un diseño de bases de datos relacional basado en dependencias funcionales. En relación con la descomposición, abordamos los conceptos de reunión no aditiva (sin pérdida) y *conservación de las dependencias*, los cuales están implementados por alguno de estos algoritmos. Otros temas incluidos en el Capítulo 11 son las dependencias multivalor, las de concatenación y la cuarta y quinta formas normales, las cuales consideran estas dependencias.

Preguntas de repaso

10.1. Argumente la semántica de atributo como una medida informal de la idoneidad de un esquema de relación.

10.2. Comente las anomalías de inserción, borrado y modificación. ¿Por qué están consideradas como malas? Ilustre sus comentarios con ejemplos.

10.3. ¿Por qué deben evitarse en la medida de lo posible lo valores NULL en una relación? Comente el problema de las tuplas falsas y cómo pueden prevenirse.

10.4. Enuncie las directrices informales para un esquema de relación que hemos comentado. Ilustre cómo la violación de estas líneas maestras podría ser dañina.

10.5. ¿Qué es una dependencia funcional? ¿Cuáles son las posibles fuentes de información que definen las dependencias funcionales que se cumplen entre los atributos de un esquema de relación?

10.6. ¿Por qué no podemos inferir automáticamente una dependencia funcional de un estado de relación particular?

10.7. ¿Cuál es el papel de las reglas de inferencia de Armstrong (las que van de la RI1 a la RI3) en el desarrollo de la teoría del diseño relacional?

10.8. ¿Qué implican la integridad y la solidez de las reglas de inferencia de Armstrong?

10.9. ¿Cuál es el significado de la clausura de un conjunto de dependencias funcionales? Ilústrelo con un ejemplo.

10.10. ¿Cuándo son equivalentes dos dependencias funcionales? ¿Cómo podemos determinar esta equivalencia?

10.11. ¿Qué es un conjunto mínimo de dependencias funcionales? ¿Debe tener cada conjunto de dependencias un conjunto equivalente mínimo? ¿Es siempre único?

10.12. ¿A qué hace referencia el término *relación no normalizada*? ¿Cómo se desarrollaron históricamente las formas normales desde la primera hasta la de Boyce-Codd?

10.13. Defina las tres primeras formas normales cuando sólo se consideran las claves principales. ¿En qué difieren las definiciones generales de la 2FN y la 3FN, las cuales consideran todas las claves de una relación, de las que sólo consideran las claves principales?

10.14. ¿Qué dependencias no deseables se evitan cuando una relación está en 2FN?

10.15. ¿Qué dependencias no deseables se evitan cuando una relación está en 3FN?

10.16. Defina la forma normal de Boyce-Codd. ¿En qué difiere de la 3FN? ¿Por qué se la considera una forma más estricta de la 3FN?

Ejercicios

10.17. Suponga que tenemos los siguientes requisitos para la base de datos de una universidad que se utiliza para controlar los certificados de estudios de los estudiantes:

a. La universidad controla, por cada estudiante, su NombreEstudiante, su NúmeroEstudiante, su Dni, su DirecciónActualEstudiante y su TeléfonoActualEstudiante, su DirecciónPermanenteEstudiante y su TeléfonoPermanenteEstudiante, su FechaNac, su Sexo, su Curso ('primer año'. 'segundo año', . . . , 'graduado') y su Especialidad. Tanto el Dni como el NúmeroEstudiante tienen valores únicos para cada estudiante.

b. Cada departamento está descrito mediante un NombreDpto, un CódigoDpto, un NúmeroOficina, un TeléfonoOficina y un Colegio. Tanto el nombre como el código tienen valores únicos para cada departamento.

c. Cada curso tienen un NombreCurso, una DescripciónCurso, un NúmeroCurso, el NúmeroHorasSemestre, el Nivel y el DepartamentoImparte. El número de curso es único por cada uno de ellos.

d. Cada Sección tiene un profesor (NombreProfesor), un Semestre, un Año, un CursoSección y un NumSección. El número de sección diferencia cada una de las secciones del mismo curso que se imparten durante el mismo semestre/año; sus valores son 1, 2, 3, ... , hasta alcanzar el número de secciones impartidas durante cada semestre.

e. Un registro de nota hace referencia a un Estudiante (Dni), una sección particular y una Nota.

Diseñar un esquema de base de datos relacional para esta aplicación. Primero, muestre todas las dependencias funcionales que deben cumplirse entre los atributos. A continuación, diseñe el esquema de relaciones para la base de datos que están en 3FN o BCNF. Especifique los atributos clave

de cada relación. Anote cualquier requisito no especificado y tome las decisiones necesarias para suministrar la especificación completa.

10.18. Confirme o rechace las siguientes reglas de inferencia para dependencias funcionales. Puede realizarse una comprobación mediante un argumento de prueba o usando las reglas de inferencia de la RI1 a la RI3. Los rechazos deben llevarse a cabo probando una instancia de relación que satisfaga las condiciones y dependencias funcionales en el lado izquierdo de la regla de inferencia pero que no satisfaga las dependencias del lado derecho.

a. $\{W \rightarrow Y, X \rightarrow Z\} \models \{WX \rightarrow Y\}$

b. $\{X \rightarrow Y\}$ y $Y \supseteq Z \models \{X \rightarrow Z\}$

c. $\{X \rightarrow Y, X \rightarrow W, WY \rightarrow Z\} \models \{X \rightarrow Z\}$

d. $\{XY \rightarrow Z, Y \rightarrow W\} \models \{XW \rightarrow Z\}$

e. $\{X \rightarrow Z, Y \rightarrow Z\} \models \{X \rightarrow Y\}$

f. $\{X \rightarrow Y, XY \rightarrow Z\} \models \{X \rightarrow Z\}$

g. $\{X \rightarrow Y, Z \rightarrow W\} \models \{XZ \rightarrow YW\}$

h. $\{XY \rightarrow Z, Z \rightarrow X\} \models \{Z \rightarrow Y\}$

i. $\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow YZ\}$

j. $\{XY \rightarrow Z, Z \rightarrow W\} \models \{X \rightarrow W\}$

10.19. Considere los dos siguientes conjuntos de dependencias funcionales: $F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ y $G = \{A \rightarrow CD, E \rightarrow AH\}$. Compruebe si son equivalentes.

10.20. Considere el esquema de relación EMP_DEPT de la Figura 10.3(a) y el siguiente conjunto G de dependencias funcionales en EMP_DEPT: G = {Dni → {NombreE, FechaNac, Dirección, NúmeroDpto}, NúmeroDpto → {NombreDpto, DniDirector} }. Calcule las clausuras $\{\text{Dni}\}^+$ y $\{\text{NúmeroDpto}\}^+$ con respecto a G.

10.21. ¿Es mínimo el conjunto de dependencias funcionales G del Ejercicio 10.20? En caso negativo, intente localizar uno que sea equivalente a G y demuestre que ambos lo son.

10.22. ¿Qué anomalías de actualización ocurren en las relaciones EMP_PROY y EMP_DEPT de las Figuras 10.3 y 10.4?

10.23. ¿En qué forma normal está el esquema de relación PARCELAS de la Figura 10.11(a) con respecto a las interpretaciones que *sólo tienen en cuenta la clave principal*? ¿Estaría en la misma forma normal si se utilizaran sus definiciones generales?

10.24. Demuestre que cualquier esquema de relación con dos atributos está en BCNF.

10.25. ¿Por qué pueden producirse dos tuplas falsas en el resultado de la concatenación de las relaciones EMP_PROY1 y EMP_ LOCS de la Figura 10.5 (el resultado se muestra en la Figura 10.6)?

10.26. Considere la relación universal $R = \{A, B, C, D, E, F, G, H, I, J\}$ y el conjunto de dependencias funcionales $F = \{ \{A, B\} \rightarrow \{C\}, \{A\} \rightarrow \{D, E\}, \{B\} \rightarrow \{F\}, \{F\} \rightarrow \{G,H\}, \{D\} \rightarrow \{I, J\} \}$. ¿Cuál es la clave para R? Descomponga R en relaciones 2FN y después en 3FN.

10.27. Repita el Ejercicio 10.26 para el siguiente conjunto diferente de dependencias funcionales $G = \{ \{A, B\} \rightarrow \{C\}, \{B, D\} \rightarrow \{E, F\}, \{A, D\} \rightarrow \{G, H\}, \{A\} \rightarrow \{I\}, \{H\} \rightarrow \{J\} \}$.

10.28. Considere la relación dada en la tabla de la página siguiente:

a. Dada la extensión previa (estado), ¿cuáles de las siguientes dependencias *podrían cumplirse* en la relación anterior? Si la dependencia no se cumple, explique por qué *especificando las tuplas que provocan la violación.*

i. A → B, ii. B → C, iii. C → B, iv. B → A, v. C → A

A	B	C	TUPLA#
10	b1	c1	#1
10	b2	c2	#2
11	b4	c1	#3
12	b3	c4	#4
13	b1	c1	#5
14	b3	c4	#6

b. ¿Tiene la relación anterior alguna clave candidata? En caso afirmativo, ¿cuál es? En caso negativo, ¿por qué no la tiene?

10.29. Considere una relación $R(A, B, C, D, E)$ con las siguientes dependencias:

$AB \rightarrow C$, $CD \rightarrow E$, $DE \rightarrow B$

¿Es AB una clave candidata de esta relación? En caso negativo, ¿lo es ABD? Razone su respuesta.

10.30. Considere la relación R, que tiene atributos que guardan programaciones de cursos y secciones en una universidad; R = {NúmeroCurso, NumSección, DeptOfertante, HorasCrédito, NivelCurso, DniProfesor, Semestre, Año, HorasDía, NúmeroSala, NúmeroDeEstudiantes}. Supongamos que en R se mantienen las siguientes dependencias funcionales:

{NúmeroCurso} → {DeptOfertante, HorasCrédito, NivelCurso}
{NúmeroCurso, NumSección, Semestre, Año} → {HorasDía, NúmeroSala, NúmeroDeEstudiantes, DniProfesor}
{NúmeroSala, HorasDía, Semestre, Año} → {DniProfesor, NúmeroCurso, NumSección}

Intente determinar qué conjuntos de atributos forman las claves de R. ¿Cómo se podría normalizar esta relación?

10.31. Considere las siguientes relaciones para una aplicación de base de datos para el procesamiento de pedidos de la empresa ABC, Inc.

PEDIDO (NúmeroPedido, FechaPedido, NúmeroCliente, CosteTotal)
LÍNEA_PEDIDO(NúmeroPedido, CódigoObjeto, CantidadSolicitada, PrecioTotal, PorcentajeDto)

Asumimos que cada línea de pedido tiene un descuento diferente. El PrecioTotal se refiere a una línea, FechaPedido es la fecha en la que el pedido se realizó y el CosteTotal es el coste total del mismo. Si aplicamos una concatenación natural en la relaciones LÍNEA_PEDIDO y PEDIDO, ¿qué aspecto tendría el esquema de relación resultante? ¿Cuál sería su clave? Muestre las DF resultantes de esta relación. ¿Está en 2FN? ¿Está en 3FN? Razone sus respuestas.

10.32. Considere la siguiente relación:

VENTA_COCHE(NúmeroCoche, FechaVenta, NúmeroVendedor, PorcentajeComisión, Descuento)

Asumimos que un coche lo pueden vender varios vendedores, por lo que {NúmeroCoche, NúmeroVendedor} es la clave principal. Las siguientes son otras dependencias adicionales

FechaVenta → Descuento y
NúmeroVendedor → PorcentajeComisión

Basándonos en la clave principal anterior, ¿está la relación en 1FN, 2FN o 3FN? Razone su respuesta. ¿Cómo podría normalizarla completamente?

Considere la siguiente relación para registrar los libros publicados:

LIBRO(TítuloLibro, Autor, TipoLibro, ListaPrecio, AfiliaciónAutor, Editorial)

AfiliaciónAutor hace referencia a la afiliación del autor. Suponga que existen las siguientes dependencias:

TituloLibro → Editorial, TipoLibro

TipoLibro → ListaPrecio

Autor → AfiliaciónAutor

a. ¿En qué forma normal está la relación? Razone su respuesta.

b. Aplique una normalización hasta que la relación no pueda descomponerse más. Explique las razones que se esconden tras cada descomposición.

10.34. Este ejercicio le pide que convierta sentencias de empresa en dependencias. Considere la relación DISCO (NúmeroSerie, Fabricante, Modelo, Versión, Capacidad, Distribuidor). Cada tupla de esta relación contiene información acerca de un disco con un NúmeroSerie único, fabricado por un fabricante, que tiene un modelo particular, distribuido con un número de versión concreto, que tiene una determinada capacidad y que es vendido por un distribuidor concreto. Por ejemplo, la tupla Disco ('1978619', 'WesternDigital', 'A2235X', '765234', 500, 'CompUSA') especifica que WesternDigital fabricó un disco con el número de serie 1978619, el modelo A2235X y la versión 765234; su capacidad es de 500GB y lo vende CompUSA.

Escriba cada una de las siguientes dependencias como una DF:

a. El fabricante y el número de serie identifican el disco de forma única.

b. Un modelo está registrado por un fabricante y, por consiguiente, no puede utilizarlo ningún otro.

c. Todos los discos de una versión particular son del mismo modelo.

d. Todos los discos de un determinado modelo de un fabricante particular tienen exactamente la misma capacidad.

10.35. Demuestre que $AB \rightarrow D$ está en la clausura de:

$$\{AB \rightarrow C, CE \rightarrow D, A \rightarrow E\}$$

10.36. Considere la relación siguiente:

R (NúmeroDoctor, NúmeroPaciente, Fecha, Diagnóstico, CódigoTratamiento, Coste)

En la relación anterior, cada tupla describe la visita de un paciente a un médico junto con el código del tratamiento diagnosticado y un coste diario. Asumimos que el diagnóstico está determinado (únicamente) para cada paciente por un médico. Asumimos también que cada código de tratamiento tiene un coste fijo (independientemente del paciente). ¿Esta relación está en 2FN? Justifique su respuesta y descompóngala si fuera necesario. A continuación, argumente si es necesaria una normalización a 3FN y, en caso afirmativo, realícela.

10.37. Considere la siguiente relación:

VENTA_COCHE (IdCoche, Extra, PrecioExtra, FechaVenta, PrecioDescontado)

Esta relación hace referencia a los extras instalados en un coche (por ejemplo, el control de velocidad) y que se vendieron a un distribuidor, y la lista y los precios descontados de los extras.

Si IdCoche → FechaVenta y Extra → PrecioExtra
e IdCoche, Extra → PrecioDescontado,

argumente usando la definición generalizada de la 3FN que esta relación no está en 3FN. A continuación, y desde sus conocimientos de la 2FN, indique por qué tampoco está en 2FN.

10.38. Considere una versión descompuesta de la relación:

PRECIO_ACTUAL_EXTRA (IdCoche, Extra, PrecioDescontado)

COCHE(IdCoche, FechaVenta)
EXTRA(Extra, PrecioExtra)

Usando el algoritmo de comprobación de descomposición sin pérdida (Algoritmo 11.1), determine si esta descomposición es realmente sin pérdidas.

Ejercicios de práctica

Nota. Los siguientes ejercicios utilizan el sistema DBD (Diseñador de base de datos, *Data Base Designer*) descrito en el manual de prácticas. El esquema relacional R y el conjunto de dependencias funcionales F tienen que codificarse como listas. Como ejemplo, R y F para el problema 10.26 están codificados de este modo:

$R = [a, b, c, d, e, f, g, h, i, j]$
$F = [[[a, b],[c]],$
$[[a],[d, e]],$
$[[b],[f]],$
$[[f],[g, h]],$
$[[d],[i, j]]]$

Ya que DBD está implementado en Prolog, el uso de términos en mayúsculas está reservado a las variables del lenguaje y, por consiguiente, las constantes en minúsculas se emplean para codificar los atributos. Para obtener más detalles acerca del uso del sistema DBD, consulte el manual de prácticas.

10.39. Usando el sistema DBD, verifique sus respuestas a los siguientes ejercicios:

a. 10.19
b. 10.20
c. 10.21
d. 10.26 (sólo 3FN)
e. 10.27
f. 10.29
g. 10.30

Bibliografía seleccionada

Las dependencias funcionales fueron presentadas originalmente por Codd (1970). Las definiciones originales de la primera, segunda y tercera formas normales fueron definidas en Codd (1972a), donde puede encontrarse también un comentario acerca de las anomalías de actualización. La forma normal de Boyce-Codd fue definida en Codd (1974). La definición alternativa de la tercera forma normal fue dada en Ullman (1988), así como la definición de BCNF que hemos mostrado aquí. Ullman (1988), Maier (1983) y Atzeni y De Antonellis (1993) contienen muchos de los teoremas y demostraciones referentes a las dependencias funcionales.

Armstrong (1974) muestra la solidez y la integridad de las reglas de inferencia de la RI1 a la RI3. El Capítulo 11 contiene referencias adicionales sobre la teoría del diseño relacional.

CAPÍTULO 11

Algoritmos de diseño de bases de datos relacionales y dependencias adicionales

El Capítulo 10 presentó una técnica de **diseño relacional descendente** y una serie de conceptos usados extensamente en el diseño de bases de datos comerciales de hoy en día. El procedimiento supone la generación de un esquema conceptual ER o EER y su mapeo posterior al modelo relacional mediante alguno de los procedimientos descritos en el Capítulo 7. Las claves principales se asignan a cada relación en base a dependencias funcionales conocidas. El proceso consecuente podría recibir el nombre de **diseño relacional por análisis**, en el que las relaciones diseñadas inicialmente a partir del procedimiento anterior (o aquéllas heredadas de ficheros, formularios y otras fuentes) se analizan para detectar dependencias funcionales no deseadas. Estas dependencias son eliminadas por el procedimiento de normalización sucesivo descrito en la Sección 10.3 junto con las definiciones de las formas normales relacionadas, las cuales son los mejores estados de diseño de las relaciones individuales. En la Sección 10.3 asumimos que las claves primarias fueron asignadas a relaciones individuales; en la Sección 10.4 se presentó un tratamiento de normalización más general en el que se tenían en cuenta todas las claves candidatas de cada relación.

En este Capítulo usamos la teoría de las formas normales y las dependencias funcionales desarrolladas en el último capítulo mientras se mantienen tres diferentes planteamientos. En primer lugar, se describen las propiedades deseables de las concatenaciones no aditivas (sin pérdida) y la conservación de dependencias funcionales. Se presenta también un algoritmo general para comprobar la no aditividad de las concatenaciones entre un conjunto de relaciones. En segundo lugar, se muestra un acercamiento al **diseño relacional por síntesis** de dependencias funcionales, un **acercamiento de diseño ascendente** que presupone que se ha tomado como entrada el conocimiento de las dependencias funcionales a través de los conjuntos de atributos en el UoD (Universo de discurso, *Universe of Discourse*). Presentamos algoritmos para obtener las formas normales deseables (la 3FN y la BCNF) y para conseguir una, o ambas, propiedades de no aditividad de las concatenaciones y la conservación de la dependencia funcional. Aunque la aproximación por síntesis es teóricamente interesante como un acercamiento formal, no debe usarse en la práctica con diseños de bases de datos de gran tamaño debido a la dificultad de proporcionar todas las posibles dependencias funcionales directas antes de intentar el diseño. Sin embargo, con el desarrollo presentado en el Capítulo 10, las descomposiciones sucesivas y los refinamientos del diseño se hacen más manejables y pueden evolucionar con el tiempo. El último planteamiento de este capítulo es comentar los nuevos tipos de dependencias, como la MVD (Dependencia multivalor, *Multi-Valued Dependency*) y la cuarta forma normal basada en la eliminación de las MVD; por último, se ofrece una introducción breve a la dependencia de concatenación (*Join Dependency*) y a la quinta forma normal relacionada con ella.

En la Sección 11.1, primero se tratan las dos **propiedades de descomposición** deseables, la de conservación de la dependencia y la de concatenación sin pérdida (o no aditiva), que se utilizan en el diseño de algoritmos para obtener descomposiciones correctas. Es importante indicar que *no basta* con comprobar los esquemas de relación *independientemente unos de otros* por cumplir las formas normales superiores como la 2FN, la 3FN y la BCNF. Las relaciones resultantes deben satisfacer colectivamente estas dos propiedades adicionales para calificar un diseño como bueno. La Sección 11.2 está dedicada al desarrollo de algoritmos de diseño relacional que comienzan con un esquema de relación gigante llamado **relación universal**, que es una relación hipotética que contiene todos los atributos. Esta relación se descompone (o, en otras palabras, las dependencias funcionales dadas se sintetizan) en relaciones que satisfacen una determinada forma normal (por ejemplo, la 3FN o la BCNF) y que también cumple alguna de las propiedades deseables.

En la Sección 11.3 trataremos la MVD, y a continuación la JD (Dependencia de concatenación, *Join Dependency*) en la Sección 11.4. Estos dos tipos de dependencias representan restricciones que *no pueden* capturarse por las dependencias funcionales. Su eliminación produce relaciones en cuarta (4FN, carentes de MVDs no deseadas) y quinta formas normales (5FN, carentes de JDs no deseadas).

Completamos la explicación de las dependencias entre datos introduciendo las dependencias de inclusión y las dependencias de plantilla. Las primeras pueden representar restricciones de integridad referencial y de clase/subclase a través de las relaciones. Las segundas son una forma de representar cualquier restricción generalizada en los atributos. Describiremos también algunas situaciones en las que es necesaria la definición de un procedimiento o una función para declarar y verificar una dependencia funcional entre los atributos.

Por último, veremos la DKNF (Forma normal de dominio clave, *Domain-Key Normal Form*), que está considerada como la forma normal más general.

En un curso de introducción a las bases de datos, es posible saltarse alguna, o todas, las Secciones desde la 11.4 a la 11.6.

11.1 Propiedades de las descomposiciones relacionales

En la Sección 11.1.1 ofrecemos ejemplos para evidenciar que examinar una relación *individual* para verificar si está en una forma normal superior no es garantía, por sí misma, de un buen diseño; antes bien, un *conjunto de relaciones* que, juntas, forman el esquema de una base de datos relacional debe poseer ciertas propiedades adicionales para garantizar un buen diseño. En las Secciones 11.1.2 y 11.1.3 comentaremos dos de estas propiedades: la conservación (o preservación) de la dependencia y la concatenación no aditiva. La Sección 11.1.4 está dedicada a las descomposiciones binarias, mientras que la Sección 11.1.5 se centra en las de concatenación no aditiva.

11.1.1 Descomposición de una relación e insuficiencia de formas normales

Los algoritmos de diseño de una base de datos relacional que se presentan en la Sección 11.2 se inician a partir de un único **esquema de relación universal** $R = \{A_1, A_2, \ldots, A_n\}$ que incluye *todos* los atributos de la base de datos. Implícitamente hacemos la conjetura de **relación universal**, que especifica que cada nombre de atributo es único. El conjunto F de dependencias funcionales que se debe cumplir en los atributos de R está especificado por los diseñadores de la base de datos y disponible a través de los algoritmos de diseño. Al usar dependencias funcionales, los algoritmos descomponen el esquema de relación universal R en un conjunto de esquemas de relación $D = \{R_1, R_2, \ldots, R_m\}$ que se convertirán en el esquema de la base de datos relacional; D recibe el nombre de **descomposición** de R.

Debemos asegurarnos de que en la descomposición, cada atributo de R aparezca en, al menos, una relación R_i de forma que no se *pierdan* atributos; formalmente, tenemos:

$$\bigcup_{i=1}^{m} R_i = R$$

Esto es lo que se conoce como condición de **conservación de atributos** de una descomposición.

Otro de los objetivos es conseguir que cada relación R_i individual de la descomposición D esté en BCNF o 3FN. Sin embargo, esta condición no es suficiente por sí misma para garantizar un buen diseño de la base de datos. Debemos considerar la descomposición de la relación universal de una forma general, además de buscar en las relaciones individuales. Para ilustrar este punto, consideremos la relación EMP_LOCS(NombreE, UbicaciónProyecto) de la Figura 10.5, la cual está en 3FN y en BCNF. De hecho, cualquier esquema de relación con sólo dos atributos está automáticamente en BCNF[1]. Aunque EMP_LOCS está en BCNF, sigue generando tuplas falsas cuando se concatena con EMP_PROY (Dni, NumProyecto, Horas, NombreProyecto, UbicaciónProyecto), la cual no está en BCNF (compruebe el resultado de la concatenación natural en la Figura 10.6). Por tanto, EMP_LOCS representa un esquema de relación malo debido a su enrevesada semántica por la que UbicaciónProyecto da la localización de *uno de los proyectos* en los que trabaja un empleado. Al concatenar EMP_LOCS con PROYECTO(NombreProyecto, NumProyecto, UbicaciónProyecto, NumDptoProyecto) de la Figura 10.2 (que *está* en BCNF) también se generan tuplas falsas. Esto acentúa la necesidad de encontrar otros criterios que, junto con las condiciones de la 3FN o la BCNF, prevengan estos malos diseños. En las siguientes tres subsecciones comentaremos algunas de estas condiciones adicionales que deben cumplirse en una descomposición D.

11.1.2 Propiedad de conservación de la dependencia de una descomposición

Resultaría útil si cada dependencia funcional $X \rightarrow Y$ especificada en F apareciera directamente en uno de los esquemas de relación R_i de la descomposición D, o pudiera inferirse a partir de las dependencias que aparecen en alguna R_i. Informalmente, esto es lo que se conoce como *condición de conservación de la dependencia.* Lo que queremos es mantener las dependencias porque cada una de ellas en F representa una restricción de la base de datos. Si alguna de estas dependencias no está representada en alguna relación individual R_i de la descomposición, no podemos imponer esta restricción en referencia a una relación concreta. Puede que tengamos que concatenar varias relaciones para incluir todos los atributos implicados en esta dependencia.

No es necesario que las dependencias exactas especificadas en F aparezcan en las relaciones individuales de la descomposición D. Basta con que la unión de las dependencias que se mantienen en las relaciones individuales de D sea equivalente a F. Ahora estamos en condiciones de definir estos conceptos de una manera más formal.

> **Definición.** Dado un conjunto de dependencias F en R, la **proyección** de F en R_i, especificada por $\pi_{Ri}(F)$ donde R_i es un subconjunto de R, es el conjunto de dependencias $X \rightarrow Y$ en F^+ tales que los atributos en $X \cup Y$ se encuentran todos en R_i. Por consiguiente, la proyección de F en cada esquema de relación R_i en la descomposición D es el conjunto de dependencias funcionales en F^+, la clausura de F, tal que todos sus atributos del lado izquierdo y derecho están en R_i. Decimos que una descomposición $D = \{R_1, R_2, \ldots, R_m\}$ de R **conserva las dependencias** respecto a F si la unión de las proyecciones de F en cada R_i en D es equivalente a F; es decir, $((\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$.

Si una descomposición no es de dependencia conservada, algunas de las dependencias se **pierden** en la descomposición. Para comprobar que una dependencia perdida se mantiene, debemos tomar la CONCATENACIÓN de dos o más relaciones en la descomposición para obtener una relación que incluya todos los atributos de la izquierda y de la derecha de la dependencia perdida, y después verificar que la dependencia persiste en el resultado de la CONCATENACIÓN (una opción que no es práctica).

---

[1] Como ejercicio, el lector deberá probar que esta sentencia es verdadera.

La Figura 10.12(a) muestra un ejemplo de descomposición que no conserva las dependencias, ya que se pierde la dependencia funcional DF2 cuando se descompone PARCELAS1A en {PARCELAS1AX, PARCELAS1AY}. Sin embargo, las descomposiciones de la Figura 10.11 conservan las dependencias. De forma similar, para el ejemplo de la Figura 10.13, no importa la descomposición que se elija para la relación ENSEÑAR(Estudiante, Curso, Profesor) de las tres que se ofrecen en el texto, se pierden una o ambas dependencias presentadas originalmente. A continuación, formulamos una afirmación que está relacionada con esta propiedad sin proporcionar ninguna prueba.

> **Afirmación 1.** Siempre es posible buscar una descomposición D con las dependencias conservadas respecto a F de modo que cada relación Ri en D esté en 3FN.

En la Sección 11.2.1 se describe el Algoritmo 11.2, que crea una descomposición con las dependencias conservadas $D = \{R_1, R_2, \ldots, R_m\}$ de una relación universal R basada en un conjunto de dependencias funcionales F, de modo que cada R_i en D esté en 3FN.

11.1.3 Propiedad no aditiva (sin pérdida) de una descomposición

Otra propiedad que puede poseer una descomposición D es la de concatenación no aditiva, la cual garantiza que no se generarán tuplas falsas cuando se aplica una operación de CONCATENACIÓN NATURAL (NATURAL JOIN) a las relaciones de la descomposición. En la Sección 10.1.4 ya ilustramos este problema con el ejemplo de las Figuras 10.5 y 10.6. Ya que ésta es una propiedad de una descomposición de esquemas de relación, la condición de que no existan tuplas falsas debe mantenerse en *cada estado de relación legal,* es decir, en cada relación que satisface las dependencias funcionales en F. Por consiguiente, la propiedad de concatenación sin pérdida está siempre definida respecto a un conjunto específico F de dependencias.

> **Definición**. Formalmente, una descomposición $D = \{R_1, R_2, \ldots, R_m\}$ de R tiene **la propiedad de concatenación sin pérdida (no aditiva)** respecto al conjunto de dependencias F en R si, por *cada* estado de relación r de R que satisface F, se mantiene lo siguiente, donde $*$ es la CONCATENACIÓN NATURAL de todas las relaciones en D: $*(\pi_{R1}(r), \ldots, \pi_{Rm}(r)) = r$.

La palabra "pérdida" en *sin pérdida* hace referencia a una *pérdida de información*, no de tuplas. Si una descomposición no tiene esta propiedad, podríamos obtener tuplas falsas adicionales una vez aplicadas las operaciones PROYECCIÓN (π) y CONCATENACIÓN NATURAL ($*$); estas tuplas adicionales representan información errónea o incorrecta. Preferimos el término concatenación no aditiva porque describe la situación de una forma más exacta. Aunque el término concatenación sin pérdida ha sido muy popular en la literatura, nosotros usaremos de ahora en adelante el término "concatenación no aditiva" porque es más explicativo y menos ambiguo. Esta propiedad garantiza que no se producirán tuplas falsas tras la aplicación de las operaciones PROYECCIÓN y CONCATENACIÓN. Sin embargo, puede que haya ocasiones en las que empleemos el concepto diseño con pérdida para referirnos a un diseño que representa una pérdida de información (consulte el ejemplo que aparece al final del Algoritmo 11.2).

La descomposición de EMP_PROY(Dni, NumProyecto, Horas, NombreE, NombreProyecto, UbicaciónProyecto) de la Figura 10.3 en EMP_LOCS(NombreE, UbicaciónProyecto) y EMP_PROY1(Dni, NumProyecto, Horas, NombreProyecto, UbicaciónProyecto) de la Figura 10.5 obviamente no cuenta con la propiedad de concatenación no aditiva, tal y como se muestra en la Figura 10.6. Usaremos un procedimiento general para comprobar si cualquier descomposición D de una relación en n relaciones es no aditiva respecto a un conjunto dado de dependencias funcionales F en la relación; es el Algoritmo 11.1 que se muestra más abajo. Es posible aplicar una verificación más simple para determinar si la descomposición es no aditiva para las descomposiciones binarias; esta verificación se describe en la Sección 11.1.4.

> **Algoritmo 11.1.** Verificación de la propiedad de concatenación no aditiva:
>
> **Entrada**: Una relación universal R, una descomposición $D = \{R_1, R_2, \ldots, R_m\}$ de R y un conjunto F de dependencias funcionales.

Nota: Al final de algunos de los pasos podrá encontrar algunos comentarios explicativos que siguen el formato: (* *comentario* *)

1. Cree una matriz inicial S con una fila i por cada relación R_i en D, y una columna j por cada atributo A_j en R.
2. Asigne $S(i, j) := b_{ij}$ en todas las entradas de la matriz. (* cada b_{ij} es un símbolo distinto asociado a índices (i, j) *)
3. Por cada fila i que representa un esquema de relación R_i

   {por cada columna j que representa un atributo A_j

   {si la (relación R_i incluye un atributo A_j) entonces asignar $S(i, j) := a_j$;};};

   (* cada a_j es un símbolo distinto asociado a un índice (j) *)
4. Repetir el siguiente bucle hasta que una *ejecución completa del mismo* no genere cambios en S

   {por cada dependencia funcional $X \rightarrow Y$ en F

   {para todas las filas de S *que tengan los mismos símbolos* en las columnas correspondientes a los atributos de X

   {hacer que los símbolos de cada columna que se corresponden con un atributo de Y sean los mismos en todas esas filas siguiendo este patrón: si cualquiera de las filas tiene un símbolo a para la columna, hacer que el resto de filas tengan el *mismo* símbolo a en la columna. Si no existe un símbolo a para el atributo en ninguna de las filas, elegir uno de los símbolos b para el atributo que aparezcan en una de las filas y ajustar el resto de filas a ese valor;} ;} ;};
5. Si una fila está compuesta enteramente por símbolos a, entonces la descomposición tiene la propiedad de concatenación no aditiva; en caso contrario, no la tiene.

Dada una relación R que está descompuesta en un número de relaciones $R_1, R_2, \ldots, R_m$, el Algoritmo 11.1 empieza con la matriz S que consideramos que es algún estado de relación r de R. La fila i en S representa una tupla t_i (correspondiente a la relación R_i) que tiene símbolos a en las columnas que se corresponden con los atributos de R_i y símbolos b en el resto. El algoritmo transforma entonces las filas de esta matriz (en el bucle del paso 4) de modo que representen tuplas que satisfagan todas las dependencias funcionales en F. Al final del paso 4, dos filas cualesquiera de S (que representan a dos tuplas de r) que coinciden en sus valores de atributos izquierdos X de una dependencia funcional $X \rightarrow Y$ en F coincidirán también en los valores de sus atributos derechos Y. Puede verse que tras aplicar el bucle del paso 4, si una fila de S sólo cuenta con símbolos a, entonces la descomposición D tiene la propiedad de concatenación no aditiva respecto a F.

Si, por otro lado, ninguna fila termina sólo con símbolos a, D no satisface la propiedad de concatenación no aditiva. En este caso, la relación r representada por S al final del algoritmo será un ejemplo de estado de relación r de R que satisface las dependencias en F pero que no cumple la condición de concatenación no aditiva. Por consiguiente, esta relación sirve como un **contraejemplo** que prueba que D no tiene esta propiedad respecto a F. Observe que los símbolos a y b no tienen ningún significado especial al final del algoritmo.

La Figura 11.1(a) muestra cómo aplicar el Algoritmo 11.1 a la descomposición del esquema de relación EMP_PROY de la Figura 10.3(b) en los dos esquemas de relación EMP_PROY1 y EMP_LOCS de la Figura 10.5(a). El bucle del paso 4 no puede cambiar ninguna de las *bes* por *aes*; por tanto, la matriz S resultante no cuenta con ninguna fila en la que sólo haya símbolos a, y por tanto la descomposición no tiene la propiedad de concatenación sin pérdida.

La Figura 11.1(b) es otro ejemplo de descomposición de EMP_PROY (en EMPLEADO, PROYECTO y TRABAJA_EN) que tiene la propiedad de concatenación no aditiva, mientras que la Figura 11.1(c) muestra la forma de aplicar el algoritmo para esta descomposición. Una vez que una fila sólo está compuesta por símbolos a, sabemos que la descomposición tiene la propiedad de concatenación no aditiva, y podemos dejar de aplicar las dependencias funcionales (paso 4 del algoritmo) a la matriz S.

Figura 11.1. Comprobación de la concatenación no aditiva para *n* descomposiciones. (a) Caso 1: la descomposición de EMP_PROY en EMP_PROY1 y EMP_LOCS hace que la comprobación falle. (b) Una descomposición de EMP_PROY que tiene la propiedad de concatenación sin pérdida. (c) Caso 2: la descomposición de EMP_PROY en EMPLEADO, PROYECTO y TRABAJA_EN satisface la comprobación.

(a) R = {Dni, NombreE, NumProyecto, NombreProyecto, UbicaciónProyecto, Horas} $D = \{R_1, R_2\}$
R_1 = EMP_LOCS = {NombreE, UbicaciónProyecto}
R_2 = EMP_PROY1 = {Dni, NumProyecto, Horas, NombreProyecto, UbicaciónProyecto}

F = {Dni ➔ NombreE; NumProyecto ➔ {NombreProyecto, UbicaciónProyecto}; {Dni, NumProyecto} ➔ Horas}

	Dni	NombreE	NumProyecto	NombreProyecto	UbicaciónProyecto	Horas
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(Ningún cambio en la matriz después de aplicar las dependencias funcionales.)

(b) **EMP**

Dni	NombreE

PROYECTO

NumProyecto	NombreProyecto	UbicaciónProyecto

TRABAJA_EN

Dni	NumProyecto	Horas

(c) R = {Dni, NombreE, NumProyecto, NombreProyecto, UbicaciónProyecto, Horas} $D = \{R_1, R_2, R_3\}$
R_1 = EMP = {Dni, NombreE}
R_2 = PROY = {NumProyecto, NombreProyecto, UbicaciónProyecto}
R_3 = TRABAJA_EN = {Dni, NumProyecto, Horas}

F ={Dni ➔ NombreE; NumProyecto ➔ {NombreProyecto, UbicaciónProyecto};{Dni, NumProyecto} ➔ Horas}

	Dni	NombreE	NumProyecto	NombreProyecto	UbicaciónProyecto	Horas
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Matriz S original al comienzo del algoritmo.)

	Dni	NombreE	NumProyecto	NombreProyecto	UbicaciónProyecto	Horas
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matriz S después de aplicar las dos primeras dependencias funcionales;
la última fila sólo contiene símbolos "a", por lo que paramos)

11.1.4 Comprobación de la propiedad de concatenación no aditiva en descomposiciones binarias

El Algoritmo 11.1 nos permite comprobar si una descomposición D particular en n relaciones cumple la propiedad de concatenación no aditiva respecto a un conjunto de dependencias funcionales F. Existe un caso especial llamado **descomposición binaria**: la descomposición de una relación R en dos relaciones. Vamos a dar un método más simple que el Algoritmo 11.1 que, aunque es muy cómodo de utilizar, sólo puede aplicarse a las descomposiciones binarias.

Propiedad NJB (Comprobación de concatenación no aditiva para descomposiciones binarias). Una descomposición $D = \{R_1, R_2\}$ de R tiene la propiedad de concatenación sin pérdida (no aditiva) respecto a un conjunto de dependencias funcionales F en R *si y sólo si*:

- La DF $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ está en F^+, o bien,
- La DF $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ está en F^+.

Es preciso comprobar que esta propiedad se cumple con respecto a nuestros ejemplos informales de normalización sucesiva de las Secciones 10.3 y 10.4.

11.1.5 Descomposiciones de concatenación no aditiva sucesivas

En las Secciones 10.3 y 10.4 vimos una descomposición sucesiva de relaciones durante el proceso de la segunda y tercera normalización. Para comprobar que estas descomposiciones son no aditivas, necesitamos garantizar otra propiedad, lo que desemboca en la Afirmación 2.

Afirmación 2 (Conservación de la no aditividad en descomposiciones sucesivas). Si una descomposición $D = \{R_1, R_2, \ldots, R_m\}$ de R tiene la propiedad de concatenación no aditiva respecto a un conjunto de dependencias funcionales F en R, y si una descomposición $D_i = \{Q_1, Q_2, \ldots, Q_k\}$ de R_i también tiene la propiedad en relación a la proyección de F en R_i, entonces la descomposición $D_2 = \{R_1, R_2, \ldots, R_{i-1}, Q_1, Q_2, \ldots, Q_k, R_{i+1}, \ldots, R_m\}$ de R cuenta a su vez con la propiedad de concatenación no aditiva respecto a F.

11.2 Algoritmos para el diseño de un esquema de base de datos relacional

Ahora vamos a ver tres algoritmos para la creación de una descomposición relacional a partir de una relación universal. Cada algoritmo tiene propiedades específicas, como veremos más adelante.

11.2.1 Descomposición de la conservación de dependencias en esquemas 3FN

El Algoritmo 11.2 crea una descomposición de la conservación de dependencias $D = \{R_1, R_2, \ldots, R_m\}$ de una relación universal R basada en un conjunto de dependencias funcionales F, de modo que cada R_i en D esté en 3FN. Sólo garantiza la propiedad de conservación de las dependencias, no la de concatenación no aditiva. El primer paso del Algoritmo 11.2 es localizar una cobertura mínima G para F; para ello, puede utilizarse el Algoritmo 10.2. Pueden existir varias coberturas mínimas para un conjunto F (como quedará demostrado en el ejemplo que sigue al Algoritmo 11.2). En estos casos, los algoritmos pueden producir potencialmente múltiples diseños alternativos.

Algoritmo 11.2. Síntesis relacional en 3FN con conservación de las dependencias:

Entrada: Una relación universal R y un conjunto de dependencias funcionales F en los atributos de R.

1. Localizar una cobertura mínima G para F (utilice el Algoritmo 10.2);
2. Por cada X izquierdo de una dependencia funcional que aparezca en G, crear un esquema de relación en D con los atributos $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, donde $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k$ son las únicas dependencias en G que tienen X como parte izquierda (X es la clave de esta relación);
3. Situar cualquier atributo que sobre (los que no se hayan podido colocar en ninguna relación) en un esquema de relación simple que asegure la propiedad de conservación de los atributos.

Ejemplo del Algoritmo 11.2. Considere la siguiente relación universal:

U(DniEmpleado, NumProy, SalarioEmpleado, TlfEmpleado, NúmeroDpto, NombreProyecto, UbicaciónProyecto)

DniEmpleado, SalarioEmpleado y TlfEmpleado hacen referencia al Documento Nacional de Identidad, el salario y el teléfono del empleado. NumProy, NombreProyecto y UbicaciónProyecto son el número, el nombre y la localización del proyecto. NúmeroDpto es el número de departamento.

Las siguientes dependencias están presentes:

DF1: DniEmpleado→SalarioEmpleado, TlfEmpleado, NúmeroDpto
DF2: NumProy→NombreProyecto, UbicaciónProyecto
DF3: DniEmpleado, NumProy→SalarioEmpleado, TlfEmpleado, NúmeroDpto, NombreProyecto, UbicaciónProyecto

En virtud de la DF3, el conjunto de atributos {DniEmpleado, NumProy} es una clave de la relación universal. Así pues, F, el conjunto de DFs dado, incluye {DniEmpleado → SalarioEmpleado, TlfEmpleado, NúmeroDpto; NumProy→NombreProyecto, UbicaciónProyecto; DniEmpleado, NumProy→SalarioEmpleado, TlfEmpleado, NúmeroDpto, NombreProyecto, UbicaciónProyecto}.

Aplicando el Algoritmo 10.2 de cobertura mínima, en el paso 3 vemos que NumProy es un atributo redundante en DniEmpleado, NumProy→SalarioEmpleado, TlfEmpleado, NúmeroDpto. Además, DniEmpleado es redundante en DniEmpleado, NumProy→NombreProyecto, UbicaciónProyecto. Por consiguiente, la cobertura mínima constará de DF1 y DF2 (DF3 es completamente redundante) de la siguiente forma (si agrupamos los atributos con la misma LHS en una DF):

Cobertura mínima G: {DniEmpleado→SalarioEmpleado, TlfEmpleado, NúmeroDpto; NumProy→NombreProyecto, UbicaciónProyecto}

Aplicando el Algoritmo 11.2 a G, obtenemos un diseño 3FN que consta de dos relaciones con las claves DniEmpleado y NumProy:

R_1 (DniEmpleado, SalarioEmpleado, TlfEmpleado, NúmeroDpto)

R_2 (NumProy, NombreProyecto, UbicaciónProyecto)

Un lector observador habrá visto fácilmente que estas dos relaciones han perdido la información original contenida en la clave de la relación universal U (digamos que existen ciertos empleados que trabajan en ciertos proyectos en una relación de muchos-a-muchos). Así, mientras que el algoritmo conserva las dependencias originales, no garantiza que se mantenga toda la información. Por tanto, el diseño resultante es *pobre*.

Afirmación 3. Todo esquema de relación creado por el Algoritmo 11.2 está en 3FN (no daremos aquí una comprobación formal;[2] la comprobación está condicionada a que G sea un conjunto mínimo de dependencias).

Es obvio que todas las dependencias en G son conservadas por el algoritmo, ya que cada una de ellas aparece en una de las relaciones R_i de la descomposición D. Ya que G es equivalente a F, todas las dependencias

[2] Si desea una demostración, consulte Maier (1983) o Ullman (1982).

en F también se conservan directamente en la descomposición, o puede derivarse de ellas usando las reglas de inferencia de la Sección 10.2.2 en las relaciones resultantes, lo que asegura la propiedad de conservación de las dependencias. El Algoritmo 11.2 recibe el nombre de **algoritmo de síntesis relacional**, porque cada esquema de relación R_i en la descomposición se sintetiza (construye) a partir del conjunto de dependencias funcionales en G con la misma X izquierda.

11.2.2 Descomposición de concatenación no aditiva en esquemas BCNF

El siguiente algoritmo descompone una relación universal $R = \{A_1, A_2, \ldots, A_n\}$ en una descomposición $D = \{R_1, R_2, \ldots, R_m\}$ de forma que cada R_i esté en BCNF y que la descomposición D tenga la propiedad de concatenación sin pérdida respecto a F. El Algoritmo 11.3 utiliza la propiedad NJB y la Afirmación 2 (conservación de la no aditividad en descomposiciones sucesivas) para crear una descomposición de concatenación no aditiva $D = \{R_1, R_2, \ldots, R_m\}$ de una relación universal R basada en un conjunto de dependencias funcionales F, de forma que cada R_i en D esté en BCNF.

Algoritmo 11.3. Descomposición relacional en BCNF con la propiedad de concatenación no aditiva:

Entrada: Una relación universal R y un conjunto de dependencias funcionales F en los atributos de R.

1. Establecer $D := \{R\}$;
2. Mientras que exista un esquema de relación Q en D que no sea BCNF hacer lo siguiente:

```
{
  elegir un esquema de relación Q en D que no esté en BCNF;
  localizar una dependencia funcional X→Y en Q que viole BCNF;
  reemplazar Q en D con dos esquemas de relación (Q – Y) y (X ∪ Y);
} ;
```

Cada pasada por el bucle del Algoritmo 11.3 descompone un esquema de relación Q que no está en BCNF en dos esquemas de relación. Según la propiedad NJB para las descomposiciones binarias y la Afirmación 2, la descomposición D tiene la propiedad de concatenación no aditiva. Al final del algoritmo, todos los esquemas de relación en D estarán en BCNF. El lector puede comprobar que el ejemplo de normalización de las Figuras 10.11 y 10.12 siguen básicamente este algoritmo. Las dependencias funcionales DF3, DF4 y la tardía DF5 violan la BCNF, por lo que la relación PARCELAS se descompone apropiadamente en relaciones BCNF, lo que satisface la propiedad de concatenación no aditiva. De forma parecida, si aplicamos el algoritmo al esquema de relación ENSEÑAR de la Figura 10.13, éste se descompone en ENSEÑAR1(Profesor, Estudiante) y ENSEÑAR2(Profesor, Curso) porque la dependencia DF2: Profesor→Curso viola la BCNF.

En el segundo paso del Algoritmo 11.3, es necesario determinar si un esquema de relación Q está en BCNF o no. Un método para lograrlo es comprobar, por cada dependencia funcional $X \rightarrow Y$ en Q, si X^+ falla al incluir todos los atributos en Q, para así determinar si X es o no una (súper)clave en Q. Otra técnica está basada en una observación que dice que siempre que un esquema de relación Q viola la BCNF existe un par de atributos A y B en Q que hacen que $\{Q - \{A, B\}\} \rightarrow A$; procesando la clausura $\{Q - \{A, B\}\}^+$ para cada par de atributos $\{A, B\}$ de Q, y verificando si esa clausura incluye A (o B), podemos determinar si Q está en BCNF.

11.2.3 Conservación de las dependencias y descomposición de concatenación no aditiva (sin pérdida) en esquemas 3FN

Si queremos que una descomposición tenga la propiedad de concatenación no aditiva y que conserve las dependencias, tenemos que ajustar los esquemas de relación a 3FN en lugar de a BCNF. Una sencilla modificación del Algoritmo 11.2, que se muestra en el Algoritmo 11.4, consigue que una descomposición D de R cumpla lo siguiente:

- Conservar las dependencias.
- Tener la propiedad de concatenación no aditiva.
- Que cada esquema de relación resultante de la descomposición esté en 3FN.

Ya que el Algoritmo 11.4 logra las dos propiedades deseables, en lugar de sólo la de conservación de las dependencias funcionales garantizada por el Algoritmo 11.2, podemos considerarlo como un algoritmo preferente.

Algoritmo 11.4. Síntesis relacional en 3FN con conservación de las dependencias y propiedad de concatenación no aditiva:

Entrada: Una relación universal R y un conjunto de dependencias funcionales F en los atributos de R.

1. Localizar una cobertura mínima G para F (utilice el Algoritmo 10.2).
2. Por cada X izquierdo de una dependencia funcional que aparezca en G, crear un esquema de relación en D con los atributos $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, donde $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k$ son las únicas dependencias en G que tienen X como parte izquierda (X es la clave de esta relación).
3. Si ninguno de los esquemas de relación en D contienen una clave de R, crear un esquema de relación más en D que contenga los atributos que forman una clave de R.[3] (Puede utilizarse el Algoritmo 11.4(a) para localizar una clave.)
4. Eliminar las relaciones redundantes del conjunto de relaciones resultante en el esquema de base de datos relacional. Una relación R se considera como redundante si es una proyección de otra relación S en el esquema; alternativamente, R está abarcada por S.[4]

El paso 3 del Algoritmo 11.4 implica la identificación de una clave K de R. Puede usarse el Algoritmo 11.4(a) para identificarla basándonos en el conjunto dado de dependencias funcionales F. Empezamos asignando K a todos los atributos de R; a continuación, eliminamos un atributo cada vez y comprobamos si los que quedan siguen formando una superclave. Observe que el conjunto de dependencias funcionales usado para determinar una clave en el Algoritmo 11.4(a) podría ser F o G, ya que ambos son equivalentes. Observe también que el Algoritmo 11.4(a) sólo determina *una clave* de entre todas las posibles claves candidatas de R; la que se devuelva dependerá del orden en el que se eliminan los atributos de R en el paso 2.

Algoritmo 11.4(a). Localización de una clave K para R dado un conjunto F de dependencias funcionales:

Entrada: Una relación universal R y un conjunto de dependencias funcionales F en los atributos de R.

1. Establecer $K := R$.
2. Por cada atributo A en K

   {procesar $(K - A)^+$ respecto a F;
   si $(K - A)^+$ contiene todos los atributos en R, entonces establecer $K := K - \{A\}$ };

Ejemplo 1 del Algoritmo 11.4. Vamos a retomar el ejemplo dado al final del Algoritmo 11.2. La cobertura mínima G se mantiene como antes. El segundo paso produce las relaciones R_1 y R_2 igual que antes. Sin embargo, ahora en el paso 3 se generará una relación correspondiente a la clave {DniEmpleado, NumProy}. Por consiguiente, el diseño resultante contiene:

---

[3] El paso 3 del Algoritmo 11.2 no es necesario en el Algoritmo 11.4 para conservar los atributos, porque la clave incluirá todos los que no están colocados; se trata de los atributos que no participan en ninguna dependencia funcional.

[4] Observe que hay un tipo adicional de dependencia: R es una proyección de la concatenación de dos o más relaciones en el esquema. Este tipo de redundancia está considerada como de concatenación, como se verá más adelante en la Sección 11.4. De este modo, técnicamente, podría seguir existiendo sin desestabilizar el estado 3FN del esquema.

R_1 (DniEmpleado, SalarioEmpleado, TlfEmpleado, NúmeroDpto)

R_2 (NumProy, NombreProyecto, UbicaciónProyecto)

R_3 (DniEmpleado, NumProy)

Este diseño consigue las dos propiedades deseables de conservación de las dependencias y concatenación no aditiva.

Ejemplo 2 del Algoritmo 11.4 (caso *X*). Considere el esquema de relación PARCELAS1A de la Figura 10.12(a). Asumimos que es una relación universal que tiene las siguientes dependencias funcionales:

DF1: IdPropiedad→NúmeroParcela, NombreMunicipio, Área

DF2: NúmeroParcela, NombreMunicipio→Área, IdPropiedad

DF3: Área→NombreMunicipio

Los nombres que recibieron en la Figura 10.12(a) eran DF1, DF2 y DF5. El significado de los atributos anteriores y la implicación de las dependencias funcionales se explicaron en la Sección 10.4. Para facilitar la referencia, abreviaremos los atributos como IdPropiedad = P, NúmeroParcela = L, NombreMunicipio = C y Área=A, y representaremos las dependencias funcionales como el conjunto:

F : { P → LCA, LC → AP, A → C }.

Si aplicamos el Algoritmo 10.2 de cobertura mínima a F, (en el paso 2) representamos primero el conjunto F como:

F : {P → L, P → C, P → A, LC → A, LC → P, A → C}.

En el conjunto F, podemos inferir P→A de P→LC y LC→A; P→A se da por transitividad y es, por consiguiente, redundante. De este modo, una posible cobertura mínima sería:

Cobertura mínima GX: {P→LC, LC → AP, A → C}.

En el paso 2 del Algoritmo 11.4 se produce el diseño X (antes de eliminar relaciones redundantes) como:

Diseño X: R_1 (P, L, C), R_2 (L, C, A, P) y R_3 (A, C).

En el paso 4 del algoritmo, vemos que R_3 está incluida en R_2, es decir, R_3 es siempre una proyección de R_2 al igual que R_1. Por tanto, ambas relaciones son redundantes. De este modo, el esquema 3FN que logra las dos propiedades deseables es (tras eliminar las relaciones redundantes):

Diseño X: R_2 (L, C, A, P).

o, en otras palabras, es idéntica a la relación PARCELAS1A (NúmeroParcela, NombreMunicipio, Área, IdPropiedad) que en la Sección 10.4.2 determinamos que estaba en 3FN.

Ejemplo 2 del Algoritmo 11.4 (caso *Y*). Empezando con PARCELAS1A como la relación universal ,y dado el mismo conjunto de dependencias funcionales, el segundo paso del Algoritmo 10.2 de cobertura mínima produce, como antes:

F: {P → C, P → A, P → L, LC → A, LC → P, A → C}.

La DF LC → A podría considerarse como redundante porque LC → P y P → A implica LC → A por transitividad. Lo mismo podríamos decir de P→C, ya que P → A y A → C implican P → C también por transitividad. Esto proporciona una cobertura mínima diferente:

Cobertura mínima GY: { P → LA, LC → P, A → C }.

El diseño alternativo Y producido por el algoritmo es ahora:

Diseño Y: S_1 (P, A, L), S_2 (L, C, P) y S_3 (A, C).

Observe que este diseño tiene tres relaciones 3FN, de la cuales ninguna puede considerarse como redundante por la condición del paso 4. Todas las DF del conjunto F original se mantienen. El lector habrá observado que fuera de las tres relaciones anteriores, S_1 y S_3 se produjeron como el diseño BCNF según el procedimiento de la Sección 10.5 (lo que implica que S_2 es redundante en presencia de S_1 y S_3). Sin embargo, no podemos eliminar S_2 del conjunto de las tres relaciones 3FN ya que no es una proyección ni de S_1 ni S_3. El diseño Y permanece, por consiguiente, como uno de los posibles resultados de la aplicación del Algoritmo 11.4 en la relación universal que genera las relaciones 3FN.

Es importante saber que la teoría de las descomposiciones de concatenación no aditiva se basan en la suposición de que *no se permiten valores NULL en los atributos de concatenación*. La siguiente sección trata algunos de los problemas que los NULL pueden causar en las descomposiciones relacionales.

11.2.4 Problemas con los valores NULL y las tuplas colgantes

A la hora de diseñar un esquema de bases de datos relacional debemos prestar mucha atención a los problemas derivados de los NULL. Hasta ahora, no existe ningún diseño relacional satisfactorio que incluya estos valores. Un problema aparece cuando alguna tupla tiene valores NULL en los atributos que se usarán para concatenar las relaciones individuales en la descomposición. Para demostrar esto, considere la base de datos de la Figura 11.2(a), donde se muestran las relaciones EMPLEADO y DEPARTAMENTO. Las dos últimas tuplas de empleado ('Palomo' y 'Benítez') representan a dos empleados recientemente contratados que aún no están asignados a un departamento (asumimos que esta situación no viola ninguna restricción de integridad). Ahora vamos a suponer que queremos recuperar una lista de valores (NombreE, NombreDpto) de todos los empleados. Si aplicamos la CONCATENACIÓN NATURAL en EMPLEADO y DEPARTAMENTO (Figura 11.2[b]), las dos tuplas antes mencionadas *no* aparecerán en el resultado. La operación CONCATENACIÓN EXTERNA, comentada en el Capítulo 6, puede resolver este problema. Recuerde que si tomamos la CONCATENACIÓN EXTERNA IZQUIERDA de EMPLEADO con DEPARTAMENTO, las tuplas de EMPLEADO que tengan NULL en los atributos de concatenación aparecerán en el resultado unidas con una tupla *imaginaria* de DEPARTAMENTO que tiene valores NULL en todos sus atributos. La Figura 11.2(c) muestra el resultado.

En general, siempre que el esquema de una base de datos relacional esté diseñado de forma que dos o más relaciones estén interrelacionadas mediante *foreign keys*, debemos prestar especial atención para localizar potenciales valores NULL en dichas claves, ya que esto puede provocar pérdidas inesperadas de información en las consultas que implican concatenación de ellas. Además, si aparecen NULLs en otros atributos, como el salario, debe evaluarse con cuidado su efecto en funciones predefinidas como SUM y AVERAGE.

Un problema relacionado es el de las *tuplas colgantes*, que se pueden generar si llevamos a cabo una descomposición demasiado lejos. Supongamos que descomponemos la relación EMPLEADO de la Figura 11.2(a) más allá de EMPLEADO_1 y EMPLEADO_2 (véanse las Figuras 11.3[a] y 11.3[b]).[5] Si aplicamos una operación CONCATENACIÓN NATURAL a ambas relaciones, obtenemos la relación EMPLEADO original. Sin embargo, podemos usar la representación alternativa de la Figura 11.3(c), donde *no se incluye una tupla* en EMPLEADO_3 si el empleado aún no ha sido asignado a ningún departamento (en lugar de generar una con valores NULL en NumDptoProyecto como ocurre en EMPLEADO_2). Si usamos EMPLEADO_3 en lugar de EMPLEADO_2 y aplicamos una CONCATENACIÓN NATURAL en EMPLEADO_1 y EMPLEADO_3, las tuplas correspondientes a Palomo y Benítez no aparecerán en el resultado; esto es lo que se conoce como **tuplas colgantes**, porque sólo están representadas en una de las dos relaciones que representan a empleados y que, por tanto, se perderían si se aplicara una operación CONCATENACIÓN (INTERNA).

[5] Esto ocurre a veces si se aplica una fragmentación vertical a la relación en el contexto de una base de datos distribuida (consulte el Capítulo 25).

Figura 11.2. Problemas que aparecen en las concatenaciones con valores NULL. (a) Alguna tupla EMPLEADO tiene NULLs en el atributo de concatenación NumDptoProyecto. (b) Resultado de aplicar una CONCATENACIÓN NATURAL a las relaciones EMPLEADO y DEPARTAMENTO. (c) Resultado de aplicar una CONCATENACIÓN EXTERNA IZQUIERDA a EMPLEADO y DEPARTAMENTO.

(a)

EMPLEADO

NombreE	Dni	FechaNac	Dirección	NúmeroDpto
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98	5
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9	5
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38	4
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67	4
Ojeda Ordóñez, Fernando	666884444	15-09-1962	Portillo, s/n	5
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6	5
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90	4
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1	1
Palomo González, Andrés	999775555	26-04-1965	Alameda, 3	NULL
Benítez Gallego, Carlos	888664444	09-01-1963	Paseo del río, 45	NULL

DEPARTAMENTO

NombreDpto	NúmeroDpto	DniDirector
Investigación	5	333445555
Administración	4	987654321
Sede Central	1	888665555

(b)

NombreE	Dni	FechaNac	Dirección	NúmeroDpto	NombreDpto	DniDirector
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98	5	Investigación	333445555
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9	5	Investigación	333445555
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38	4	Administración	987654321
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67	4	Administración	987654321
Ojeda Ordóñez, Fernando	666884444	15-09-1962	Portillo, s/n	5	Investigación	333445555
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6	5	Investigación	333445555
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90	4	Administración	987654321
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1	1	Sede central	888665555

(c)

NombreE	Dni	FechaNac	Dirección	NúmeroDpto	NombreDpto	DniDirector
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98	5	Investigación	333445555
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9	5	Investigación	333445555
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38	4	Administración	987654321
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67	4	Administración	987654321
Ojeda Ordóñez, Fernando	666884444	15-09-1962	Portillo, s/n	5	Investigación	333445555
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6	5	Investigación	333445555
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90	4	Administración	987654321
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1	1	Sede Central	888665555
Palomo González, Andrés	999775555	26-04-1965	Alameda, 3	NULL	NULL	NULL
Benítez Gallego, Carlos	888665555	09-01-1963	Paseo del río, 45	NULL	NULL	NULL

Figura 11.3. El problema de la tupla colgante. (a) La relación EMPLEADO_1 (incluye todos los atributos de EMPLEADO de la Figura 11.2[a], excepto NumDptoProyecto). (b) La relación EMPLEADO_2 (incluye los atributos NumDptoProyecto con valores NULL). (c) La relación EMPLEADO_3 (incluye NumDptoProyecto, pero no las tuplas en las que este atributo tiene valores NULL).

(a) EMPLEADO_1

NombreE	Dni	FechaNac	Dirección
Pérez Pérez, José	123456789	09-01-1965	Eloy I, 98
Campos Sastre, Alberto	333445555	08-12-1955	Avda. Ríos, 9
Jiménez Celaya, Alicia	999887777	19-07-1968	Gran Vía, 38
Sainz Oreja, Juana	987654321	20-06-1941	Cerquillas, 67
Ojeda Ordóñez, Fernando	666884444	15-09-1962	Portillo, s/n
Oliva Avezuela, Aurora	453453453	31-07-1972	Antón, 6
Pajares Morera, Luis	987987987	29-03-1969	Enebros, 90
Ochoa Paredes, Eduardo	888665555	10-11-1937	Las Peñas, 1
Palomo González, Andrés	999775555	26-04-1965	Alameda, 3
Benítez Gallego, Carlos	888665555	09-01-1963	Paseo del río, 45

(b) EMPLEADO_2

Dni	NúmeroDpto
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLEADO_3

Dni	NúmeroDpto
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

11.2.5 Normalización de algoritmos

Uno de los problemas que existen en los algoritmos de normalización que hemos descrito es que el diseñador de la base de datos debe especificar primero *todas* las dependencias funcionales relevantes entre los atributos de la base de datos. Esto no es una tarea simple en bases de datos grandes con cientos de atributos. Un error a la hora de especificar una o dos dependencias importantes puede desembocar en un diseño ilegible. Otro problema es que estos algoritmos son, en general, *no deterministas*. Por ejemplo, los *algoritmos de síntesis* (Algoritmos 11.2 y 11.4) precisan de la especificación de una cobertura mínima G para un conjunto de dependencias funcionales F. Debido a que pueden existir muchas de estas coberturas mínimas para F, como quedó demostrado en el Ejemplo 2 del Algoritmo 11.4, éste puede generar diferentes diseños dependiendo de la cobertura usada. Algunos de estos diseños pueden no ser adecuados. El *Algoritmo de descomposición* (Algoritmo 11.3) depende del orden en el que se le suministran las dependencias funcionales para verificar la violación de la BCNF. De nuevo, es posible que se generen muchos diseños diferentes para el mismo conjunto de dependencias funcionales, en función del orden en el que son consideradas dichas dependencias en la

comprobación de la BCNF. Algunos de los diseños pueden ser bastante superiores, mientras que otros pueden no ser adecuados.

No siempre es posible encontrar una descomposición en los esquemas de relación que conserve las dependencias y permita que cada esquema de relación en la descomposición esté en BCNF (en lugar de en 3FN como ocurre en el Algoritmo 11.4). Podemos verificar individualmente los esquemas de relación 3FN en la descomposición para ver si cada una de ellas satisface la BCNF. Si algún esquema de relación R_i no está en BCNF, podemos elegir entre efectuar más descomposiciones o dejarlo en 3FN (con el riesgo de posibles anomalías de actualización).

Para ilustrar los puntos anteriores, vamos a retomar la relación PARCELAS1A de la Figura 10.12(a). Esta relación está en 3FN, pero no en BCNF como quedó demostrado en la Sección 10.5. Vimos también que al empezar con las dependencias funcionales (DF1, DF2 y DF5 de la Figura 10.12[a]) usando el planteamiento ascendente para el diseño y la aplicación del Algoritmo 11.4, es posible la generación de cualquier relación PARCELAS1A como diseño 3FN (lo que llamamos previamente diseño X), u otro diseño alternativo Y compuesto por tres relaciones (S_1, S_2, S_3), cada una de ellas en 3FN. Observe que si verificamos el diseño Y para BCNF, S_1, S_2 y S_3 se giran para estar individualmente en BCNF. Sin embargo, cuando se comprueba el diseño X para BCNF, la verificación falla. Produce las dos relaciones S_1 y S_3 aplicando el Algoritmo 11.3 (debido a la violación de la dependencia funcional A $\rightarrow$ C). Por tanto, el procedimiento de diseño ascendente del Algoritmo 11.4 para diseñar relaciones 3FN, conseguir las dos propiedades deseables y después aplicar el 11.3 para obtener la BCNF con la propiedad de concatenación no aditiva (sacrificando la conservación de la dependencia funcional), produce S_1, S_2 y S_3 como diseño BCNF final por un camino (diseño Y) y S_1 y S_3 por otro (diseño X). Esto ocurre por las distintas coberturas mínimas del conjunto de dependencias funcionales original. Observe que S_2 es una relación redundante en el diseño Y; sin embargo, no viola la restricción de concatenación no aditiva. Es fácil ver que S_2 es una relación válida y significativa que tiene juntas las dos claves candidatas (L, C), y P. La Tabla 11.1 resume las propiedades de los algoritmos vistos hasta el momento en este capítulo.

Tabla 11.1 Resumen de los algoritmos explicados en las Secciones 11.1 y 11.2.

Algoritmo	Entrada	Salida	Propiedades/Objetivo	Comentarios
11.1	Una descomposición D de R y un conjunto F de dependencias funcionales	Resultado booleano: sí o no para la propiedad de concatenación no aditiva	Verificación de la descomposición de concatenación no aditiva	Consulte la verificación más simple para las descomposiciones binarias en la Sección 11.1.4
11.2	Conjunto de dependencias funcionales F	Un conjunto de relaciones en 3FN	Conservación de la dependencia	No garantiza cumplir la propiedad de concatenación sin pérdida
11.3	Conjunto de dependencias funcionales F	Un conjunto de relaciones en BCNF	Descomposición de concatenación no aditiva	No garantiza la conservación de la dependencia
11.4	Conjunto de dependencias funcionales F	Un conjunto de relaciones en 3FN	Concatenación no aditiva *y* descomposición por conservación de la dependencia	Puede no alcanzarse la BCNF, pero sí se consiguen *todas* las propiedades y la 3FN
11.4a	Esquema de relación R con un conjunto de dependencias funcionales F	Clave K de R	Localizar una clave K (que es un subconjunto de R)	Toda la relación R es siempre una superclave predeterminada

11.3 Dependencias multivalor y cuarta forma normal

Hasta ahora sólo hemos tratado la dependencia funcional que es, con mucho, el tipo de dependencia más importante en la teoría del diseño de una base de datos relacional. Sin embargo, en muchos casos, las relaciones tienen restricciones que no pueden especificarse como dependencias funcionales. En esta sección vamos a tratar el concepto de MVD (dependencia multivalor) y a definir la *cuarta forma normal,* la cual se basa en esta dependencia. Las dependencias multivalor son una consecuencia de la 1NF (consulte la Sección 10.3.4), que prohíbe que una tupla tenga un *conjunto de valores.* Si tenemos dos o más atributos multivalor *independientes* en el mismo esquema de relación, tenemos el problema de tener que repetir cada valor de uno de los atributos con cada valor del otro atributo para mantener la consistencia del estado de la relación y la independencia entre los atributos implicados. Esta restricción está especificada por una dependencia multivalor.

Por ejemplo, consideremos la relación EMP de la Figura 11.4(a). Una tupla de esta relación representa el hecho de que un empleado cuyo nombre es NombreE trabaja en el proyecto cuyo nombre es NombreProyecto y tiene un subordinado llamado NombreSubordinado. Un empleado puede trabajar en varios proyectos y tener varios subordinados, y sus proyectos y subordinados son independientes unos de otros.[6] Para mantener la coherencia de esta relación, debemos tener una tupla separada para representar cada combinación de subordinado y proyecto del empleado. Esta restricción se especifica como una dependencia multivalor en EMP. De manera informal, siempre que dos relaciones 1:N *independientes* $A{:}B$ y $A{:}C$ se mezclen en la misma relación se alcanza $R(A, B, C)$ y una MVD.

11.3.1 Definición formal de una dependencia multivalor

Definición. Una dependencia multivalor $X \twoheadrightarrow Y$ especificada en un esquema de relación R, donde X e Y son subconjuntos de R, especifica las siguientes restricciones en cualquier relación r de R: si dos tuplas t_1 y t_2 existen en r de modo que $t_1[X] = t_2[X]$, entonces también deberían existir otras dos tuplas t_3 y t_4 en r con las siguientes propiedades,[7] donde utilizamos Z para indicar $(R - (X \cup Y))$:[8]

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$.
- $t_3[Y] = t_1[Y]$ y $t_4[Y] = t_2[Y]$.
- $t_3[Z] = t_2[Z]$ y $t_4[Z] = t_1[Z]$.

Siempre que se cumpla $X \twoheadrightarrow Y$, decimos que X **multidetermina** Y. Debido a la simetría de la definición, cuando se cumpla $X \twoheadrightarrow Y$ en R, también lo hace en $X \twoheadrightarrow Z$. Por tanto, $X \twoheadrightarrow Y$ implica $X \twoheadrightarrow Z$, por lo que a veces se escribe como $X \twoheadrightarrow Y|Z$.

La definición formal especifica que dado un valor particular de X, el conjunto de valores de Y determinados por este valor de X está completamente determinado por X y *no depende* de los valores de los atributos Z restantes de R. De este modo, siempre que existan dos tuplas con distintos valores Y pero el mismo valor X, los valores de Y deben repetirse en tuplas separadas por cada *valor distinto de Z* que se produzca con ese mismo valor de X. Esto hace que Y sea un atributo multivalor de las entidades representadas por tuplas en R.

En la Figura 11.4(a), las MVD NombreE $\twoheadrightarrow$ NombreProyecto y NombreE $\twoheadrightarrow$ NombreSubordinado (o NombreE $\twoheadrightarrow$ NombreProyecto | NombreSubordinado) se cumplen en la relación EMP. El empleado con el

---

[6] En un diagrama ER, cada una podría representarse como un atributo multivalor o como un tipo de entidad débil (consulte el Capítulo 3).

[7] Las tuplas t_1, t_2, t_3 y t_4 no son necesariamente distintas.

[8] Z es un atajo para los atributos de R una vez eliminados de R los atributos en $(X \cup Y)$.

Figura 11.4. Cuarta y quinta formas normales.
(a) La relación EMP con dos MVD: NombreE →→NombreProyecto y NombreE →→NombreSubordinado.
(b) Descomposición de la relación EMP en dos relaciones 4FN, PROYECTOS_EMP y SUBORDINADOS_EMP.
(c) La relación SUMINISTRO, sin MVD, está en 4FN, pero no en 5FN si tiene la JD(R_1, R_2, R_3).
(d) Descomposición de la relación SUMINISTRO en las relaciones 5FN R_1, R_2, R_3.

(a) EMP

NombreE	NombreProyecto	NombreSubordinado
Pérez	X	Juan
Pérez	Y	Ana
Pérez	X	Ana
Pérez	Y	Juan

(b) PROYECTOS_EMP

NombreE	NombreProyecto
Pérez	X
Pérez	Y

SUBORDINADOS_EMP

NombreE	NombreSubordinado
Pérez	Juan
Pérez	Ana

(c) SUMINISTRO

NombreS	NombrePieza	NombreProy
Pérez	Cerrojo	ProyX
Pérez	Tuerca	ProyY
Alberto	Cerrojo	ProyY
María	Tuerca	ProyZ
Alberto	Clavo	ProyX
Alberto	Cerrojo	ProyX
Pérez	Cerrojo	ProyY

(d) *R*1

NombreS	NombrePieza
Pérez	Cerrojo
Pérez	Tuerca
Alberto	Cerrojo
María	Tuerca
Alberto	Clavo

***R*2**

NombreS	NombreProy
Pérez	ProyX
Pérez	ProyY
Alberto	ProyY
María	ProyZ
Alberto	ProyX

***R*3**

NombrePieza	NombreProy
Cerrojo	ProyX
Tuerca	ProyY
Cerrojo	ProyY
Tuerca	ProyZ
Clavo	ProyX

NombreE 'Pérez' trabaja en los proyectos cuyo NombreProyecto es 'X' e 'Y' y tiene dos subordinados con un NombreSubordinado 'Juan' y 'Ana'. Si sólo almacenamos las dos primeras tuplas de EMP (<'Pérez', 'X', 'Juan'> y <'Pérez', 'Y', 'Ana'>), podríamos mostrar de forma incorrecta asociaciones entre el proyecto 'X' y 'Juan' y entre el proyecto 'Y' y 'Ana'; podría equivaler a una relación falsa entre proyecto y subordinado, ya que no se pretende algo parecido a esto en esta relación. Por tanto, debemos almacenar las otras dos tuplas (<'Pérez', 'X', 'Ana'> y <' Pérez', 'Y', 'Juan'>) para mostrar que {'X', 'Y'} y {'Juan', 'Ana'} sólo están asociadas con 'Pérez', es decir, que no existe una asociación entre NombreProyecto y NombreSubordinado, lo que significa que los dos atributos son independientes.

Una MVD $X \twoheadrightarrow Y$ en R recibe el nombre de **MVD trivial** si (a) Y es un subconjunto de X, o (b) $X \cup Y = R$. Por ejemplo, la relación PROYECTOS_EMP de la Figura 11.4(b) tiene la MVD trivial NombreE $\twoheadrightarrow$ NombreSubordinado. Una MVD que no satisface ni (a) ni (b) se conoce como **MVD no trivial**. Una MVD trivial se cumplirá en *cualquier* estado de relación r de R; recibe el nombre de trivial porque no especifica ninguna restricción significativa en R.

Si tenemos una MVD no trivial en una relación, puede que tengamos que repetir valores en las tuplas. En la relación EMP de la Figura 11.4(a), los valores 'X' e 'Y' de NombreProyecto están repetidos en cada valor de NombreDpto (o, por simetría, los valores 'Juan' y 'Ana' de NombreSubordinado están repetidos con cada uno de los valores de NombreProyecto). Esta redundancia es, claramente, indeseable. Sin embargo, el esquema EMP está en BCNF porque *no* se almacenan dependencias en él. Por consiguiente, tenemos que definir una cuarta forma normal que sea más estricta que la BCNF y prohíba esquemas de relación como EMP. Primero trataremos algunas de las propiedades de las MVD y consideraremos cómo están relacionadas con las dependencias funcionales. Observe que las relaciones que contienen MVDs no triviales tienden a ser **relaciones todo clave**, es decir, su clave está compuesta por todos sus atributos juntos. Adicionalmente, es raro que relaciones de este tipo con una ocurrencia combinatoria de valores repetidos se produzca en la práctica. Sin embargo, el reconocimiento de las MVD como una dependencia potencialmente problemática es esencial en el diseño relacional.

11.3.2 Reglas de inferencia para dependencias funcionales y multivalor

Al igual que ocurre con las dependencias funcionales (DF), se han desarrollado las reglas de inferencia para las MVD. Lo mejor es desarrollar un entorno unificado que incluya tanto las DF como las MVD, de forma que ambos tipos de restricciones pueden ser tratados en conjunto. Las reglas de inferencia de la RI1 a la RI8 componen un lógico y completo conjunto para inferir dependencias funcionales y multivalor a partir de un conjunto concreto de dependencias. Asumimos que todos los atributos están incluidos en una *relación universal* $R = \{A_1, A_2, ..., A_n\}$ y que X, Y, Z y W son subconjuntos de R.

RI1 (regla reflexiva para las DF): si $X \supseteq Y$, entonces $X \rightarrow Y$.

RI2 (regla de aumento para las DF): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

RI3 (regla transitiva para las DF): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

RI4 (regla de complementación para las MVD): $\{X \twoheadrightarrow Y\} \models \{X \twoheadrightarrow (R - (X \cup Y))\}$.

RI5 (regla de aumento para las MVD): si $X \twoheadrightarrow Y$ y $W \supseteq Z$, entonces $WX \twoheadrightarrow YZ$.

RI6 (regla transitiva para las MVD): $\{X \twoheadrightarrow Y, Y \twoheadrightarrow Z\} \models X \twoheadrightarrow (Z - Y)$.

RI7 (regla de duplicación de las DF a las MVD): $\{X \rightarrow Y\} \models X \twoheadrightarrow Y$.

RI8 (regla de coalescencia para las DF y las MVD): si $X \twoheadrightarrow Y$ y existe W con las propiedades de que (a) $W \cap Y$ está vacío, (b) $W \rightarrow Z$ y (c) $Y \supseteq Z$, entonces $X \rightarrow Z$.

Las reglas de la RI1 a la RI3 son las reglas de inferencia de Armstrong sólo para las DF. De la RI4 a la RI6 pertenecen sólo a las MVD. Por último, la RI7 y la RI8 relacionan las DF y las MVD. En particular, la RI7 dice que una dependencia funcional es un *caso especial* de una dependencia multivalor, es decir, cada DF es también una MVD porque satisface la definición formal de una MVD. Sin embargo, esta equivalencia tiene un truco: una DF $X \rightarrow Y$ es una MVD $X \twoheadrightarrow Y$ con la *restricción adicional implícita* de que a lo sumo un valor de Y está asociado con cada valor de X.[9] Dado un conjunto F de dependencias funcionales y multivalor especificadas en $R = \{A_1, A_2, ..., A_n\}$, podemos usar las reglas de la RI1 a la RI8 para inferir el conjunto

[9] Es decir, el conjunto de valores de Y determinados por un valor de X está restringido a ser un conjunto simple con un único valor. Por tanto, en la práctica, nunca vemos una DF como una MVD.

(completo) de todas las dependencias (funcionales o multivalor) F^+ que se almacenarán en cada relación r de R que satisface F. De nuevo, llamamos a F^+ la **clausura** de F.

11.3.3 Cuarta forma normal

Ahora vamos a presentar la definición de la **cuarta forma normal** (**4FN**), la cual se viola cuando una relación tiene dependencias multivalor no deseadas y, por tanto, puede usarse para identificar y descomponer relaciones de este tipo.

> **Definición.** Un esquema de relación R está en 4FN respecto a un conjunto de dependencias F (que incluye dependencias funcionales y multivalor) si, por cada dependencia multivalor *no trivial* $X \twoheadrightarrow Y$ en F^+, X es una superclave de R.

La relación EMP de la Figura 11.4(a) no tiene DF, ya que es una relación todo-clave. Ya que las restricciones BCNF están formuladas sólo en términos de DFs, todas las relaciones todo-clave están siempre en BCNF por defecto. Por consiguiente, EMP está en BCNF pero no en 4FN porque en las MVD no triviales NombreE $\twoheadrightarrow$ NombreProyecto y NombreE $\twoheadrightarrow$ NombreSubordinado, y NombreE no es una superclave de EMP. Descomponemos EMP en PROYECTOS_EMP y SUBORDINADOS_EMP (véase la Figura 11.4[b]). Ambas están en 4FN, ya que las MVD NombreE $\twoheadrightarrow$ NombreProyecto en PROYECTOS_EMP y NombreE $\twoheadrightarrow$ NombreSubordinado en SUBORDINADOS_EMP son triviales. No se almacena ninguna otra MVD no trivial ni en PROYECTOS_EMP ni en SUBORDINADOS_EMP. Tampoco existe ninguna DF en estos esquemas de relación.

Para ilustrar la importancia de la 4FN, la Figura 11.5(a) muestra la relación EMP en el que un nuevo empleado, 'López', cuenta con tres subordinados ('Jesús', 'Dolores' y 'Roberto') y trabaja en cuatro proyectos diferentes ('W', 'X', 'Y' y 'Z'). Existen 16 tuplas EMP en la Figura 11.5(a). Si descomponemos la relación en PROYECTOS_EMP y SUBORDINADOS_EMP, tal y como puede verse en la Figura 11.5(b), sólo tenemos que almacenar un total de 11 tuplas en ambas relaciones. La descomposición, no sólo ahorra espacio de almacenamiento, sino que también evita las anomalías de actualización asociadas a las dependencias multivalor. Por ejemplo, si 'López' empieza a trabajar en un nuevo proyecto 'P', debemos insertar *tres* tuplas en EMP (una por cada subordinado). Si olvidamos hacerlo con alguno de ellos, la relación viola la MVD y se vuelve inconsistente porque implica de forma incorrecta una relación entre un proyecto y un subordinado.

Si la relación tiene MVDs no triviales, entonces las operaciones de inserción, borrado y actualización en tuplas únicas pueden provocar la modificación de otras tuplas. Si la actualización se realiza incorrectamente, el significado de la relación podría cambiar. Sin embargo, tras la normalización en la 4FN, estas anomalías desaparecen. Por ejemplo, para incorporar la información de que 'López' será asignado al proyecto 'P', sólo insertaremos una tupla en la relación 4FN PROYECTOS_EMP.

La relación EMP de la Figura 11.4(a) no está en 4FN porque representa a dos relaciones 1:N *independientes*: una entre los empleados y los proyectos en los que trabajan, y otra entre los empleados y sus subordinados. Hay veces en las que tenemos una relación entre tres entidades que depende de esas tres entidades, como ocurre con la relación SUMINISTRO de la Figura 11.4(c) (considere por ahora sólo las tuplas de la Figura 11.4[c] que están *por encima* de la línea de puntos). En este caso, una tupla representa a un proveedor que suministra una pieza específica a un *proyecto particular,* por lo que no existen MVDs no triviales. De este modo, la relación todo-clave SUMINISTRO ya está en 4FN y no debe descomponerse.

11.3.4 Descomposición de concatenación no aditiva en relaciones 4FN

Siempre que se descompone un esquema de relación R en $R_1 = (X \cup Y)$ y $R_2 = (R - Y)$ basándonos en una MVD $X \twoheadrightarrow Y$ contenida en R, la descomposición tiene la propiedad de la concatenación no aditiva. Puede

Figura 11.5. Descomposición de un estado de la relación EMP que no está en 4FN. (a) Relación EMP con tuplas adicionales. (b) Las dos relaciones 4FN correspondientes, PROYECTOS_EMP y SUBORDINADOS_EMP.

(a) EMP

NombreE	NombreProyecto	NombreSubordinado
Pérez	X	Juan
Pérez	Y	Ana
Pérez	X	Ana
Pérez	Y	Juan
López	W	Jesús
López	X	Jesús
López	Y	Jesús
López	Z	Jesús
López	W	Dolores
López	X	Dolores
López	Y	Dolores
López	Z	Dolores
López	W	Roberto
López	X	Roberto
López	Y	Roberto
López	Z	Roberto

(b) PROYECTOS_EMP

NombreE	NombreProyecto	NombreSubordinado
Pérez	X	Ana
Pérez	Y	Juan
López	W	Jesús
López	X	Jesús
López	Y	Jesús
López	Z	Jesús

SUBORDINADOS_EMP

NombreE	NombreSubordinado
Pérez	Ana
Pérez	Juan
López	Jesús
López	Jesús
López	Jesús
López	Jesús

demostrarse que esto es una condición necesaria y suficiente para descomponer un esquema en otros dos que tienen la propiedad de concatenación no aditiva, como hace la propiedad NJB', que es una generalización superior de la propiedad NJB mostrada anteriormente. La propiedad NJB sólo trata con las DFs, mientras que la NJB' trata tanto con las DFs como con las MVDs (recuerde que una DF es también una MVD).

Propiedad NJB'. Los esquemas de relación R_1 y R_2 forman una descomposición de concatenación no aditiva de R respecto a un conjunto F de dependencias funcionales *y* multivalor si, y solo si,

$(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$

o, por simetría, si y sólo si,

$(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$.

Podemos usar una pequeña modificación del Algoritmo 11.3 para desarrollar el 11.5, el cual crea una descomposición de concatenación no aditiva en esquemas de relación que están en 4FN (en lugar de en BCNF). Al igual que ocurre con el Algoritmo 11.3, el 11.5 *no* produce necesariamente una descomposición que conserve las DF.

Algoritmo 11.5. Descomposición relacional en relaciones 4FN con la propiedad de concatenación no aditiva:

Entrada: Una relación universal R y un conjunto de dependencias funcionales y multivalor F.

1. Establecer $D := \{ R \}$;
2. Mientras exista un esquema de relación Q en D que no esté en 4FN, hacer lo siguiente:

   { elegir un esquema de relación Q en D que no esté en 4FN;
   localizar una MVD no trivial $X \twoheadrightarrow Y$ en Q que viole la 4FN;
   sustituir Q en D por dos esquemas de relación $(Q - Y)$ y $(X \cup Y)$;
   };

11.4 Dependencias de concatenación y quinta forma normal

Hemos visto que LJ1 y LJ1' dan a un esquema de relación R la condición de estar descompuesto en otros dos esquemas R_1 y R_2, donde la descomposición tiene la propiedad de concatenación no aditiva. Sin embargo, en algunos casos, puede que no se produzca una descomposición de concatenación no aditiva de R en *dos* esquemas de relación, sino en *más de dos*. Además, puede que no exista una dependencia funcional en R que viole ninguna forma normal hasta la BCNF, y puede que no haya ninguna MVD no trivial en R que viole la 4FN. En este caso, hemos de recurrir a otra dependencia llamada *dependencia de concatenación* y, en caso de existir, llevar a cabo una *descomposición multivía* en la quinta forma normal (5FN). Es importante indicar que una dependencia de este tipo es una restricción con una semántica muy peculiar que resulta muy difícil de detectar en la práctica; por consiguiente, la normalización a 5FN es muy rara verla en la práctica.

Definición. Una **JD (Dependencia de concatenación**, *Join Dependency*), expresada por JD(R_1, R_2, . . . , R_n), especificada en un esquema de relación R, indica una restricción en los estados r de R que dice que cada estado legal r de R debe tener una descomposición de concatenación no aditiva en $R1$, $R2$, . . . , Rn; es decir, por cada r tenemos:

$$* (\pi_{R1}(r), \pi_{R2}(r), ..., \pi_{Rn}(r)) = r$$

Observe que una MVD es un caso especial de JD donde $n = 2$. Esto es, una JD indicada como JD(R_1, R_2) implica una MVD $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$ (o, por simetría, $(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$). Una dependencia de concatenación JD(R_1, R_2,..., R_n), especificada en un esquema de relación R, es una JD **trivial** si uno de los esquemas de relación R_i de la JD(R_1, R_2,..., R_n) es igual a R. Una dependencia de este tipo se dice que es trivial porque tiene la propiedad de concatenación no aditiva en cualquier estado de relación r de R y, por consiguiente, no especifica ninguna restricción en R. Ahora podemos definir la quinta forma normal, la cual recibe también el nombre de forma normal de proyección-concatenación.

Definición. Un esquema de relación R está en **quinta forma normal (5FN)** (o en **PJNF [Forma normal de proyección-concatenación, *Project-Join Normal Form*]**) respecto a un conjunto F de dependencias funcionales, multivalor y de concatenación si, por cada dependencia de concatenación no aditiva JD(R_1, R_2,..., R_n) en F^+ (es decir, implicada por F), cada R_i es una superclave de R.

Para ver un ejemplo de JD, consideremos de nuevo la relación todo-clave SUMINISTRO de la Figura 11.4(c). Supongamos que siempre se cumple la siguiente restricción adicional: siempre que un proveedor s suminis-

tra una pieza p, y un proyecto j utiliza la pieza p, y el proveedor s suministra *al menos una* pieza del proyecto j, *entonces* el proveedor s será también suministrador de p al proyecto j. Esta restricción puede replantearse de otras formas y especificar una dependencia de concatenación JD(R_1, R_2, R_3) entre las tres proyecciones R_1(NombreS, NombrePieza), R_2(NombreS, NombreProy) y R_3(NombrePieza, NombreProy) de SUMINISTRO. Si esta restricción se cumple, las tuplas por encima de la línea de puntos de la Figura 11.4(c) deben existir en cualquier estado legal de la relación SUMINISTRO que también contenga las tuplas por encima de esa línea. La Figura 11.4(d) muestra cómo la relación SUMINISTRO *con la dependencia de concatenación* se descompone en tres relaciones R_1, R_2 y R_3 que están en 5FN. Observe que la aplicación de una CONCATENACIÓN NATURAL a *dos* relaciones *cualquiera produce tuplas falsas,* pero no así la misma operación sobre *las tres relaciones*. El lector deberá verificar este hecho en la relación de ejemplo de la Figura 11.4(c) y sus proyecciones de la Figura 11.4(d). Esto es así porque sólo existe la JD, pero no se han especificado las MVD. Observe también que la JD(R_1, R_2, R_3) está indicada en *todos* los estados de relación legales, no sólo en los mostrados en la Figura 11.4(c).

Descubrir todas las JD en bases de datos reales con cientos de atributos es casi imposible. Esto sólo puede conseguirse con un alto grado de intuición sobre los datos por parte del diseñador.

11.5 Dependencias de inclusión

Las dependencias de inclusion fueron definidas para formalizar los dos tipos de restricciones interrelacionales:

- La *foreign key* (o restricción de la integridad referencial) no puede especificarse como una dependencia funcional o multivalor porque relaciona atributos entre relaciones.
- La restricción entre dos relaciones que representan una relación clase/subclase (consulte el Capítulo 4 y la Sección 7.2) tiene también una definición no formal en términos de dependencias funcionales, multivalor o de concatenación.

  Definición. Una **dependencia de inclusión** $R.X < S.Y$ entre dos conjuntos de atributos (X del esquema de relación R e Y del esquema de relación S) especifica la restricción de que, en cualquier momento en que r es un estado de relación de R y s lo es de S, debemos tener:

  $$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

La relación $\subseteq$ (subconjunto) no tiene que ser necesariamente un subconjunto. Obviamente, los conjuntos de atributos en los que se especifica la dependencia de inclusión (X de R e Y de S) deben tener el mismo número de atributos. Además, los dominios de los pares de atributos correspondientes deben ser compatibles. Por ejemplo, si $X = \{A_1, A_2, ..., A_n\}$ e $Y = \{B_1, B_2, ..., B_n\}$, una posible correspondencia es contar con dom(A_i) *compatible con* dom(B_i) para $1 \leq i \leq n$. En este caso, decimos que A_i **se corresponde con** B_i.

Por ejemplo, podemos especificar las siguientes dependencias de inclusión en el esquema relacional de la Figura 10.1:

DEPARTAMENTO.DniDirector < EMPLEADO.Dni

TRABAJA_EN.Dni < EMPLEADO.Dni

EMPLEADO.NúmeroDpto < DEPARTAMENTO.NúmeroDpto

PROYECTO.NumDptoProyecto < DEPARTAMENTO.NúmeroDpto

TRABAJA_EN.NumProyecto < PROYECTO.NumProyecto

LOCALIZACIONES_DPTO.NúmeroDpto < DEPARTAMENTO.NúmeroDpto

Todas las dependencias de inclusión anteriores representan **restricciones de integridad referencial**. También podemos usar dependencias de inclusión para representar **relaciones clase/subclase**. Por ejemplo, en el esquema relacional de la Figura 7.6, podemos especificar las siguientes dependencias de inclusión:

EMPLEADO.Dni < PERSONA.Dni

ALUMNO.Dni < PERSONA.Dni

ESTUDIANTE.Dni < PERSONA.Dni

Al igual que ocurre con otros tipos de dependencias, tenemos las IDIR (Reglas de inferencia de las dependencias de inclusión, *Inclusion Dependency Inference Rules*). Las siguientes son tres ejemplos de ellas:

IDIR1 (reflexividad): $R.X < R.X$.

IDIR2 (correspondencia de atributo): si $R.X < S.Y$, donde $X = \{A_1, A_2,..., A_n\}$ e $Y = \{B_1, B_2,..., B_n\}$ y A_i *se corresponde con* B_i, entonces $R.A_i < S.B_i$ para $1 \leq i \leq n$.

IDIR3 (transitividad): si $R.X < S.Y$ y $S.Y < T.Z$, entonces $R.X < T.Z$.

Las reglas de inferencia anteriores evidenciaron que eran correctas y completas para las dependencias de inclusión. Hasta ahora, no se han desarrollado formas normales basadas en ellas.

11.6 Otras dependencias y formas normales

11.6.1 Dependencias de plantilla

Las dependencias de plantilla ofrecen una técnica de representación de restricciones en las relaciones que, normalmente, no tienen definiciones formales ni sencillas. Sin importar cuántos tipos de dependencias desarrollemos, siempre surge alguna restricción peculiar basada en la semántica de los atributos de las relaciones que no puede ser representada por ninguna de ellas. La idea que se esconde tras las dependencias de plantilla es especificar una plantilla (o ejemplo) que defina cada restricción o dependencia.

Existen dos tipos de plantilla: la de generación de tuplas y la de generación de restricciones. Una plantilla consta de un número de **tuplas de hipótesis** que están pensadas para mostrar un ejemplo de los tipos de tuplas que pueden aparecer en una o más relaciones. La otra parte de la plantilla es la **conclusión**. En las plantillas de generación de tuplas, la conclusión es un *conjunto de tuplas* que deben existir también en las relaciones en las que estén presentes las tuplas de hipótesis. Para las plantillas de generación de restricciones, la conclusión de la plantilla es una *condición* que debe cumplirse en las tuplas de hipótesis.

La Figura 11.6 muestra cómo definir como plantillas las dependencias funcionales, multivalor y de inclusión. La Figura 11.7 indica la forma de especificar la restricción de que *el salario de un empleado no puede ser superior al de su supervisor directo* en el esquema de relación EMPLEADO de la Figura 5.5.

11.6.2 Dependencias funcionales basadas en funciones aritméticas y procedimientos

A veces, los atributos de una relación pueden estar relacionados mediante alguna función aritmética o alguna otra relación funcional más compleja. Siempre y cuando un valor único de Y esté asociado con cada X, podemos considerar que existe la DF $X \rightarrow Y$. Por ejemplo, en la relación:

LÍNEA_PEDIDO(NúmeroPedido, NúmeroObjeto, Cantidad, PrecioUnitario, PrecioTotal, PrecioDescuento)

cada tupla representa un elemento con una cantidad y el precio por unidad de ese elemento. En esta relación, (Cantidad, PrecioUnitario) →PrecioTotal según la fórmula

PrecioTotal = PrecioUnitario * Cantidad.

Por tanto, existe un único valor PrecioTotal para cada pareja (Cantidad, PrecioUnitario), y esto se adapta a la definición de dependencia funcional.

Figura 11.6. Plantilla para algunos tipos de dependencias comunes.
(a) Plantilla para la dependencia funcional X→ Y.
(b) Plantilla para la dependencia multivalor X →→Y.
(c) Plantilla para la dependencia de inclusión R.X < S.Y.

(a)

R = {A, B, C, D}

Hipótesis

a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2

Conclusión

$c_1 = c_2$ y $d_1 = d_2$

X = {A, B}
Y = {C, D}

(b)

R = {A, B, C, D}

Hipótesis

a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2

Conclusión

a_1	b_1	c_2	d_1
a_1	b_1	c_1	d_2

X = {A, B}
Y = {C}

(c)

R = {A, B, C, D}

Hipótesis

a_1	b_1	c_1	d_1

S = {E, F, G}

Conclusión

c_1	d_1	g

X = {C, D}
Y = {E, F}

Además, puede existir un procedimiento que tenga en cuenta los descuentos, el tipo de elemento, etcétera, y que procese un precio con descuento a partir de la cantidad total solicitada para ese elemento. Por consiguiente, podemos decir que:

(NúmeroObjeto, Cantidad, PrecioUnitario) →PrecioDescuento, o bien,
(NúmeroObjeto, Cantidad, PrecioTotal) →PrecioDescuento.

Para verificar la DF anterior, tenemos que hacer entrar en juego un procedimiento más complejo llamado CALCULAR_PRECIO_TOTAL. Aunque las DF anteriores están presentes en la mayoría de relaciones, no se les presta una atención especial durante la normalización.

11.6.3 Forma normal de dominio clave

No existe una regla estricta y rápida para definir formas normales sólo hasta la 5FN. Históricamente, el proceso de normalización y el de descubrimiento de dependencias no deseadas fueron llevados a cabo a través de la 5FN, pero ha sido posible definir formas normales más estrictas que tienen en cuenta tipos adicionales de dependencias y restricciones. La idea que se esconde tras la **DKFN** (**Forma normal de dominio clave**, ***Domain-Key Normal Form***) es la de especificar (al menos, teóricamente) la *última forma normal* que tiene en cuenta todos los posibles tipos de dependencias y restricciones. Un esquema de relación se dice que está en **DKNF** si todas las restricciones y dependencias que deben persistir en los estados de relación válidos pueden cumplirse simplemente haciendo cumplir las restricciones clave y de dominio de la relación. Para una relación en DKNF, le es muy sencillo cumplir todas las restricciones de la base de datos comprobando simplemente que cada atributo de una tupla está en el dominio apropiado y que se cumple cada restricción clavé.

Sin embargo, debido a la dificultad para incluir restricciones complejas en una relación DKNF, su utilidad práctica está limitada, ya que puede resultar muy complicado especificar restricciones de integridad genera-

Figura 11.7. Plantillas para la restricción de que el salario de un empleado debe ser menor que el de su supervisor.

EMPLEADO = {Nombre, Dni, . . . , Sueldo, DniSupervisor

	a	b	c	d
Hipótesis	e	d	f	g
Conclusión			c < f	

les. Por ejemplo, consideremos las relaciones COCHE(Marca, Identificador) (donde Identificador es el número de identificación del vehículo) y FABRICANTE(Identificador, País). Una restricción general podría tener la siguiente forma: *si la Marca es 'Toyota' o 'Lexus', el primer carácter de Identificador es una 'J' si el país de fabricación es 'Japón'; si la Marca es 'Honda' o 'Acura', el segundo carácter de Identificador es una 'J' si el país de fabricación es 'Japón'*. No existe una forma simplificada para representar esta restricción a no ser con un procedimiento (o una afirmación general). El procedimiento CALCULAR_PRECIO_TOTAL anterior es un ejemplo de procedimiento necesario para forzar restricciones de integridad de este tipo.

11.7 Resumen

En este capítulo hemos visto varios algoritmos de normalización. Los a*lgoritmos de síntesis relacional* (como el 11.2 y el 11.4) crean relaciones 3FN a partir de un esquema de relación universal basado en un conjunto de dependencias funcionales dado especificado por el diseñador de la base de datos. Los algoritmos de descomposición relacional (como el 11.3 y el 11.5) crean relaciones BCNF (o 4FN) mediante una descomposición no aditiva sucesiva de relaciones no normalizadas en dos relaciones. Primero estudiamos dos propiedades importantes de las descomposiciones: la de concatenación no aditiva y la de conservación de las dependencias. Hemos mostrado un algoritmo para la descomposición no aditiva (Algoritmo 11.1) y hemos hablado de las descomposiciones binarias (NJB). Vimos que es posible sintetizar esquemas de relación 3FN que cumplen las dos propiedades anteriores; sin embargo, en el caso de la BCNF, es posible tener como única meta la no aditividad de las concatenaciones: la conservación de la dependencia *puede no* estar necesariamente garantizada. En este caso, la condición de concatenación no aditiva es un objetivo prioritario.

A continuación, definimos otros tipos de dependencias y formas normales. Las dependencias multivalor, que provienen de una combinación incorrecta de dos o más atributos multivalor independientes en la misma relación y que producen una expansión combinatoria de las tuplas, se utilizan para definir la cuarta forma normal (4FN). Las dependencias de concatenación, que indican una descomposición sin pérdida multivía de una relación, inducen la definición de la quinta forma normal (5FN), que también se conoce como PJNF. Vimos también las dependencias de inclusión, que se utilizan para especificar restricciones de integridad referencial y de clase/subclase, y las dependencias de plantilla, que pueden utilizarse para especificar tipos de restricciones arbitrarias. Para forzar ciertas restricciones de una dependencia funcional, apuntamos la necesidad de contar con funciones aritméticas o con procedimientos más complejos. Concluimos con un breve comentario acerca de la DKNF.

Preguntas de repaso

11.1. ¿Cuál es el significado de la condición de conservación de los atributos en una descomposición?

11.2. ¿Por qué son insuficientes las formas normales como condición para un buen diseño de un esquema?

11.3. ¿Qué es la propiedad de conservación de la dependencia de una descomposición? ¿Por qué es importante?

11.4. ¿Por qué no podemos garantizar la producción de esquemas de relación BCNF a partir de descomposiciones de dependencia conservada de esquemas de relación que no sean BCNF? Ofrezca un contraejemplo que ilustre este punto.

11.5. ¿Qué es la propiedad de concatenación sin pérdida (o no aditiva)? ¿Por qué es importante?

11.6. Entre las propiedades de conservación de la dependencia y pérdida, ¿cuál debe satisfacerse definitivamente? ¿Por qué?

11.7. Comente los problemas relacionados con el valor NULL y las tuplas colgantes.

11.8. ¿Qué es una dependencia multivalor? ¿Qué tipo de restricción especifica? ¿Cuándo se origina?

11.9. Ilustre cómo el proceso de creación de relaciones en la primera forma normal puede inducir dependencias multivalor. ¿Cómo debe realizarse adecuadamente la primera normalización para evitar las MVD?

11.10. Defina la cuarta forma normal. ¿Cuándo se viola? ¿Por qué es útil?

11.11. Defina las dependencias de concatenación y la quinta forma normal. ¿Por qué se conoce también a la 5FN como PJNF?

11.12. ¿Qué tipo de restricción pretende representar las dependencias de inclusión?

11.13. ¿En qué difieren las dependencias de plantilla de los otros tipos de dependencias que hemos estudiado?

11.14. ¿Por qué se dice que la DKNF es la última forma normal?

Ejercicios

11.15. Demuestre que los esquemas de relación generados por el Algoritmo 11.2 están en 3FN.

11.16. Demuestre que, si la matriz S resultante del Algoritmo 11.1 no tiene una fila en la que todos sus símbolos son a, proyectando S en la descomposición y concatenándola de vuelta siempre producirá, al menos, una tupla falsa.

11.17. Demuestre que los esquemas de relación generados por el Algoritmo 11.3 están en BCNF.

11.18. Demuestre que los esquemas de relación generados por el Algoritmo 11.4 están en BCNF.

11.19. Especifique una dependencia de plantilla para las dependencias de concatenación.

11.20. Especifique todas las dependencias de inclusión del esquema relacional de la Figura 5.5.

11.21. Pruebe que una dependencia funcional satisface la definición formal de una dependencia multivalor.

11.22. Considere el ejemplo de normalización de la relación PARCELAS de la Sección 10.4. Determine si su descomposición en {PARCELAS1AX, PARCELAS1AY, PARCELAS1B, PARCELAS2} tiene la propiedad de concatenación sin pérdida, aplicando el Algoritmo 11.1, y también usando la prueba bajo la Propiedad NJB.

11.23. Demuestre cómo las MVD NombreE $\twoheadrightarrow$ NombreProyecto y NombreE $\twoheadrightarrow$ NombreSubordinado de la Figura 11.4(a) pueden producirse durante la normalización a 1NF de una relación, donde los atributos NombreProyecto y NombreSubordinado son multivalor.

11.24. Aplique el Algoritmo 11.4(a) a la relación del Ejercicio 10.26 para determinar una clave para R. Cree un conjunto mínimo de dependencias G que sea equivalente a F, y aplique el algoritmo de síntesis (Algoritmo 11.4) para descomponer R en relaciones 3FN.

11.25. Repita el Ejercicio 11.24 para las dependencias funcionales del Ejercicio 10.27.

11.26. Aplique el algoritmo de descomposición (Algoritmo 11.3) a la relación R y el conjunto de dependencias F del Ejercicio 10.26. Repítalo para las dependencias G del Ejercicio 10.27.

11.27. Aplique el Algoritmo 11.4(a) a las relaciones de los Ejercicios 10.29 y 10.30 para determinar una clave de R. Aplique el algoritmo de síntesis (Algoritmo 11.4) para descomponer R en relaciones 3FN, y el de descomposición (Algoritmo 11.3) para descomponer R en relaciones BCNF.

11.28. Escriba programas que implementen los Algoritmos 11.3 y 11.4.

11.29. Considere las siguientes descomposiciones para el esquema de relación R del Ejercicio 10.26. Determine si cada descomposición tiene (1) la propiedad de conservación de dependencia y (2) la propiedad de concatenación sin pérdida, respecto a F. Determine también en qué forma normal está cada relación de la descomposición.

a. $D_1 = \{R_1, R_2, R_3, R_4, R_5\}$; $R_1 = \{A, B, C\}$, $R_2 = \{A, D, E\}$, $R_3 = \{B, F\}$, $R_4 = \{F, G, H\}$, $R_5 = \{D, I, J\}$

b. $D_2 = \{R_1, R_2, R_3\}$; $R_1 = \{A, B, C, D, E\}$, $R_2 = \{B, F, G, H\}$, $R_3 = \{D, I, J\}$

c. $D_3 = \{R_1, R_2, R_3, R_4, R_5\}$; $R_1 = \{A, B, C, D\}$, $R_2 = \{D, E\}$, $R_3 = \{B, F\}$, $R_4 = \{F, G, H\}$, $R_5 = \{D, I, J\}$

11.30. Considere la relación FRIGORÍFICO(NúmeroModelo, Año, Precio, PlantaFabricación, Color), que de forma abreviada es FRIGORÍFICO (M, Y, P, MP, C), y el siguiente conjunto F de dependencias funcionales: $F = \{M \rightarrow MP, \{M, Y\} \rightarrow P, MP \rightarrow C\}$

a. Evalúe cada una de las siguientes como una clave candidata de FRIGORÍFICO, dando razones de si puede ser o no una clave: {M} , {M, Y} , {M, C}.

b. Basándonos en la determinación de claves anterior, indique si la relación FRIGORÍFICO está en 3FN y en BCNF, dando las razones apropiadas.

c. Considere la descomposición de FRIGORÍFICO en $D = \{R_1(M, Y, P), R_2(M, MP, C)\}$. ¿Es esta descomposición sin pérdida? Demuéstrelo (puede consultar la comprobación que se encuentra bajo la propiedad LJ1 de la Sección 11.1.4).

11.31. Considere la relación LIBRO(TítuloLibro, Autor, Edición, Año) que tiene los siguientes datos:

TítuloLibro	Autor	Edición	Año
Fundamentos de bases de datos	Navathe	3	2000
Fundamentos de bases de datos	Elmasri	3	2000
Fundamentos de bases de datos	Elmasri	4	2004
Fundamentos de bases de datos	Navathe	4	2004

a. Basándonos en el sentido común, ¿cuáles son las posibles claves candidatas de esta relación?

b. ¿Tienen los datos de la tabla una o más dependencias funcionales? (No liste las DFs aplicando las reglas de derivación.) En caso afirmativo, ¿cuáles son? Especifique cómo eliminaría las dependencias por descomposición.

c. ¿Tiene la relación resultante una MVD? En caso afirmativo, ¿qué es?

d. ¿Qué aspecto tendrá la descomposición final?

11.32. Considere la siguiente relación:

VIAJE(IdViaje, FechaInicio, CiudadesVisitadas, TarjetasUsadas)

Esta relación hace referencia a los viajes de negocio efectuados por los comerciales de una compañía. Suponga que VIAJE tiene una única FechaInicio, pero que implica muchas Ciudades y los comerciales pueden utilizar varias tarjetas de crédito en el viaje. Realice una simulación para la carga de la tabla.

a. Comente qué DFs y/o MVDs existen en esta relación.

b. Demuestre cómo la normalizaría.

Ejercicios de práctica

Nota. Estos ejercicios utilizan el sistema DBD (Diseñador de base de datos) descrito en el manual de laboratorio. El esquema relacional R y el conjunto de dependencias funcionales F tienen que codificarse como listas. Como ejemplo, R y F están codificados del siguiente modo en el problema 10.26:

$R = [a, b, c, d, e, f, g, h, i, j]$

$F = [[[a, b],[c]],$
$[[a],[d, e]],$
$[[b],[f]],$
$[[f],[g, h]],$
$[[d],[i, j]]]$

Ya que el DBD está implementado en Prolog, el uso de términos en mayúsculas está reservado a las variables del lenguaje y, por consiguiente, se utilizan las constantes en minúsculas para codificar los atributos. Para obtener más información acerca del uso del sistema DBD, por favor consulte el manual del laboratorio.

11.33. Usando el sistema DBD, verifique sus respuestas a los siguientes ejercicios:

a. 11.22
b. 11.24
c. 11.25
d. 11.26
e. 11.27
f. 11.29 (a) y (b)
g. 11.30 (a) y (c)

Bibliografía seleccionada

Los libros de Maier (1983) y de Atzeni y De Antonellis (1992) incluyen un tratamiento global de la teoría de la dependencia relacional. El algoritmo de descomposición (Algoritmo 11.3) se debe a Bernstein (1976). El Algoritmo 11.4 está basado en la algoritmo de normalización presentado en Biskup y otros (1979). Tsou y Fischer (1982) aportan un algoritmo polinómico de tiempo para la descomposición BCNF.

La teoría de la conservación de las dependencias y las concatenaciones sin pérdida aparecieron en Ullman (1988), donde aparecen algunas de las comprobaciones de los algoritmos aquí comentados. La propiedad de concatenación sin pérdida se analiza en Aho y otros (1979). Los algoritmos para determinar las claves de una relación desde las dependencias funcionales aparecen en Osborn (1976); la verificación de la BCNF se comenta en Osborn (1979), mientras que la de la 3FN aparece en Tsou y Fischer (1982). Los algoritmos para el diseño de relaciones BCNF se brindan en Wang (1990) y en Hernández y Chan (1991).

Las dependencias multivalor y la cuarta forma normal están definidas en Zaniolo (1976) y Nicolas (1978). Muchas de las formas normales avanzadas son debidas a Fagin: la cuarta forma normal en Fagin (1977), la PJNF en Fagin (1979) y la DKNF en Fagin (1981). El conjunto completo de reglas para las dependencias funcionales y multivalor aparecen en Beeri y otros (1977). Las dependencias de concatenación se comentan en Rissanen (1977) y en Aho y otros (1979). Las reglas de inferencia para las dependencias de concatenación las facilita Sciore (1982). Las dependencias de inclusión son comentadas por Casanova y otros (1981), y analizadas en Cosmadakis y otros (1990). Su uso para la optimización de esquemas relacionales se trata en Casanova y otros (1989). Las dependencias de plantilla se tratan en Sadri y Ullman (1982). El resto de dependencias se comentan en Nicolas (1978), Furtado (1978) y Mendelzon y Maier (1979). Abiteboul y otros (1995) proporciona un tratamiento teórico de muchas de las ideas presentadas en este capítulo y en el Capítulo 10.