# Parte 2
# Diseño de bases de datos

Los sistemas de bases de datos están diseñados para gestionar grandes cantidades de información. Estas grandes cantidades de información no se encuentran aisladas. Forman parte del funcionamiento de alguna empresa cuyo producto final puede ser la información obtenida de la base de datos o algún producto o servicio para el que la base de datos solo desempeña un papel secundario.

Los dos primeros capítulos de esta parte se centran en el diseño de los esquemas de las bases de datos. El modelo entidad-relación (E-R) descrito en el Capítulo 7 es un modelo de datos de alto nivel. En lugar de representar todos los datos en tablas, distingue entre los objetos básicos, denominados *entidades*, y las *relaciones* entre esos objetos. Suele utilizarse como un primer paso en el diseño de los esquemas de las bases de datos.

El diseño de las bases de datos relacionales —el diseño del esquema relacional— se trató informalmente en los capítulos anteriores. No obstante, existen criterios para distinguir los buenos diseños de bases de datos de los malos. Estos se formalizan mediante varias «formas normales» que ofrecen diferentes compromisos entre la posibilidad de inconsistencias y la eficiencia de determinadas consultas. El Capítulo 8 describe el diseño formal de los esquemas de las relaciones.

El diseño de un entorno completo de aplicaciones para bases de datos, que responda a las necesidades de la empresa que se modela, exige prestar atención a un conjunto de aspectos más amplio, muchos de los cuales se tratan en el Capítulo 9. Este capítulo describe en primer lugar el diseño de las interfaces basadas en Web para aplicaciones. Finalmente el capítulo proporciona una detallada discusión sobre la seguridad en los niveles de aplicación y de base de datos.

# Diseño de bases de datos y el modelo E-R

07

Hasta este punto se ha dado por supuesto un determinado esquema de la base de datos y se ha estudiado la manera de expresar las consultas y las actualizaciones. Ahora se va a considerar en primer lugar la manera de diseñar el esquema de la base de datos. En este capítulo nos centraremos en el modelo de datos entidad-relación (E-R), que ofrece una manera de identificar las entidades que se van a representar en la base de datos y el modo en que se relacionan entre sí. Finalmente, el diseño de la base de datos se expresará en términos del diseño de bases de datos relacionales y del conjunto de restricciones asociado. En este capítulo se mostrará la manera en que el diseño E-R puede transformarse en un conjunto de esquemas de relación y el modo en que se pueden incluir algunas de las restricciones en ese diseño. Luego, en el Capítulo 8, se considerará en detalle si el conjunto de esquemas de relación representa un diseño de la base de datos bueno o malo y se estudiará el proceso de creación de buenos diseños usando un conjunto de restricciones más amplio. Estos dos capítulos tratan los conceptos fundamentales del diseño de bases de datos.

## 7.1. Visión general del proceso de diseño

La tarea de creación de aplicaciones de bases de datos es una labor compleja, que implica varias fases, como el diseño del esquema de la base de datos, el diseño de los programas que tienen acceso a los datos y los actualizan, así como el diseño del esquema de seguridad para controlar el acceso a los datos. Las necesidades de los usuarios desempeñan un papel central en el proceso de diseño. En este capítulo nos centraremos en el diseño del esquema de la base de datos, aunque más adelante en este capítulo se esbozarán brevemente algunas de las otras tareas.

El diseño de un entorno completo de aplicaciones de bases de datos que responda a las necesidades de la empresa que se está modelando exige prestar atención a un amplio conjunto de consideraciones. Estos aspectos adicionales del uso esperado de la base de datos influyen en gran variedad de opciones de diseño en los niveles físico, lógico y de vistas.

### 7.1.1. Fases del diseño

Para aplicaciones pequeñas puede resultar factible para un diseñador de bases de datos que comprenda los requisitos de la aplicación decidir directamente sobre las relaciones que hay que crear, sus atributos y las restricciones sobre las relaciones. Sin embargo, un proceso de diseño tan directo resulta difícil para las aplicaciones reales, ya que a menudo son muy complejas. Frecuentemente no existe una sola persona que comprenda todas las necesidades de datos de la aplicación. El diseñador de la base de datos debe interactuar con los usuarios para comprender las necesidades de la aplicación; realizar una representación de alto nivel de esas necesidades, que pueda ser comprendida por los usuarios, y luego traducir esos requisitos a niveles inferiores del diseño. Los modelos de datos de alto nivel sirven a los diseñadores de bases de datos ofreciéndoles un marco conceptual en el que especificar de forma sistemática los requisitos de datos de los usuarios de la base de datos y una estructura para la base de datos que satisfaga esos requisitos.

- La fase inicial del diseño de las bases de datos es la caracterización completa de las necesidades de datos de los posibles usuarios de la base de datos. El diseñador de la base de datos debe interactuar intensamente con los expertos y los usuarios del dominio para realizar esta tarea. El resultado de esta fase es una especificación de requisitos del usuario. Aunque existen técnicas para representar en diagramas los requisitos de los usuarios, en este capítulo solo se describirán textualmente esos requisitos.
- A continuación, el diseñador elige el modelo de datos y, aplicando los conceptos del modelo de datos elegido, traduce estos requisitos en un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de **diseño conceptual** proporciona una visión detallada de la empresa. Se suele emplear el modelo entidad-relación, que se estudiará en el resto de este capítulo, para representar el diseño conceptual. En términos del modelo entidad-relación, el esquema conceptual especifica las entidades que se representan en la base de datos, sus atributos, las relaciones entre ellas y las restricciones que las afectan. Generalmente, la fase de diseño conceptual da lugar a la creación de un diagrama entidad-relación que ofrece una representación gráfica del esquema.

  El diseñador revisa el esquema para confirmar que realmente se satisfacen todos los requisitos y que no entran en conflicto entre sí. También puede examinar el diseño para eliminar características redundantes. Su atención en este momento se centra en describir los datos y sus relaciones, más que en especificar los detalles del almacenamiento físico.
- Un esquema conceptual completamente desarrollado indica también los requisitos funcionales de la empresa. En la **especificación de requisitos funcionales** los usuarios describen los tipos de operaciones (o transacciones) que se llevarán a cabo sobre los datos. Algunos ejemplos de operaciones son la modificación o actualización de datos, la búsqueda y recuperación de datos concretos y el borrado de datos. En esta fase de diseño conceptual, el diseñador puede revisar el esquema para asegurarse de que satisface los requisitos funcionales.
- El paso desde el modelo abstracto de datos a la implementación de la base de datos se divide en dos fases de diseño finales.
  - En la **fase de diseño lógico**, el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de bases de datos que se va a usar. El modelo de implementación de los datos suele ser el modelo relacional, y este paso suele consistir en la traducción del esquema conceptual definido mediante el modelo entidad-relación en un esquema de relación.

– Finalmente, el diseñador usa el esquema de base de datos resultante propio del sistema en la siguiente **fase de diseño físico**, en la que se especifican las características físicas de la base de datos. Entre estas características están la forma de organización de los archivos y las estructuras de almacenamiento interno; se estudian en los Capítulos 10 y 11.

El esquema físico de la base de datos puede modificarse con relativa facilidad una vez creada la aplicación. Sin embargo, las modificaciones del esquema lógico suelen resultar más difíciles de llevar a cabo, ya que pueden afectar a varias consultas y actualizaciones dispersas por todo el código de la aplicación. Por tanto, es importante llevar a cabo con cuidado la fase de diseño de la base de datos antes de crear el resto de la aplicación de bases de datos.

### 7.1.2. Alternativas de diseño

Una parte importante del proceso de diseño de las bases de datos consiste en decidir la manera de representar los diferentes tipos de «cosas», como personas, lugares, productos y similares. Se usa el término *entidad* para hacer referencia a cualquiera de esos elementos claramente identificables. En el ejemplo de base de datos de una universidad, son entidades los profesores, estudiantes, departamentos, asignaturas y oferta de asignaturas.[1] Las diferentes entidades se relacionan entre sí de diversas formas, relaciones que han de ser capturadas en el diseño de la base de datos. Por ejemplo, un estudiante se matricula en una oferta de asignaturas, mientras que un profesor enseña una oferta de asignaturas; matricula y enseña son ejemplos de relaciones entre entidades.

Al diseñar el esquema de una base de datos hay que asegurarse de que se evitan dos peligros importantes:

1. **Redundancia:** un mal diseño puede repetir información. Por ejemplo, si se guarda el identificador de asignatura y el nombre de la asignatura con cada oferta de esta, el nombre se guardaría de forma redundante (es decir, varias veces, de forma innecesaria) con cada oferta de asignaturas. Sería suficiente guardar solo el identificador de asignatura con cada oferta de asignaturas y asociar el nombre con el identificador de la misma una única vez en una entidad asignatura.

   También puede haber redundancia en un esquema relacional. En el ejemplo de la universidad, se tiene una relación de información de sección y una relación distinta con la información de asignatura. Suponga que en su lugar tuviésemos una sola relación en la que se repitiese toda la información de la asignatura (id de asignatura, nombre de la asignatura, nombre del departamento, créditos) para cada sección (oferta) de cada asignatura. Claramente, la información de las asignaturas se guardaría de forma redundante.

   El mayor problema con esta representación redundante de la información es que las copias de una determinada información se pueden convertir en inconsistentes si dicha información se actualiza sin tener la precaución de actualizar todas las copias de la misma. Por ejemplo, las distintas ofertas de asignaturas pueden tener el mismo identificador de asignatura, pero diferentes nombres. No quedaría claro cuál es el nombre correcto de la asignatura. De forma ideal, la información debería estar exclusivamente en un solo lugar.
2. **Incompletitud:** un mal diseño puede hacer que determinados aspectos de la empresa resulten difíciles o imposibles de modelar. Por ejemplo, supóngase que para el caso anterior solo se dispone de entidades para la oferta de asignaturas, sin tener entidades que se correspondan con las asignaturas. De forma equivalente en términos de las relaciones, suponga que tenemos una sola relación en la que se repite toda la información de asignatura para cada sección en que se oferta una asignatura. Sería imposible representar la información sobre una nueva asignatura, a no ser que dicha asignatura esté en una oferta. Se podría intentar solventar la situación guardando valores nulos en la información de sección. Este parche no solo resulta poco atractivo, sino que puede evitarse mediante restricciones de clave primaria.

Evitar los malos diseños no es suficiente. Puede haber gran número de buenos diseños entre los que haya que escoger. Como ejemplo sencillo, considérese un cliente que compra un producto. ¿La venta de este producto es una relación entre el cliente y el producto? Dicho de otra manera, ¿es la propia venta una entidad que está relacionada con el cliente y con el producto? Esta elección, aunque simple, puede suponer una importante diferencia en cuanto a que los aspectos de la empresa se pueden modelar bien. Considerando la necesidad de tomar decisiones como esta para el gran número de entidades y de relaciones que hay en las empresas reales, no es difícil ver que el diseño de bases de datos puede constituir un problema arduo. En realidad, se verá que exige una combinación de conocimientos y de «buen gusto».

## 7.2. El modelo entidad-relación

El modelo de datos **entidad–relación (E-R)** se desarrolló para facilitar el diseño de bases de datos permitiendo la especificación de un *esquema de la empresa* que representa la estructura lógica global de la base de datos.

El modelo E-R resulta de gran utilidad a la hora de trasladar los significados e interacciones de las empresas del mundo real a esquemas conceptuales. Gracias a esta utilidad, muchas herramientas de diseño de bases de datos permiten dibujar estos conceptos del modelo E-R. El modelo de datos E-R emplea tres conceptos básicos: los conjuntos de entidades, los conjuntos de relaciones y los atributos, que se tratarán en primer lugar. El modelo entidad-relación también tiene asociada una representación en forma de diagramas, los diagramas E-R, que se tratarán más adelante en este capítulo.

### 7.2.1. Conjuntos de entidades

Una **entidad** es una «cosa» u «objeto» del mundo real que es distinguible de todos los demás objetos. Por ejemplo, cada persona de una universidad es una entidad. Una entidad tiene un conjunto de propiedades, y los valores de algún conjunto de propiedades pueden identificar cada entidad de forma unívoca. Por ejemplo, una persona puede tener una propiedad *persona_id* que identifica unívocamente a dicha persona. Es decir, un valor como 677-89-9011 en *persona_id* identificará de forma única a una persona concreta de la universidad. De forma similar, se puede pensar en las asignaturas como entidades, y *asignatura_id* identifica de forma única a una entidad asignatura en la universidad. Las entidades pueden ser concretas, como las personas o los libros, o abstractas, como las asignaturas, las ofertas de asignaturas o una reserva de un vuelo.

Un **conjunto de entidades** agrupa entidades del mismo tipo que comparten propiedades o atributos. El conjunto de todas las personas que son profesores de una universidad dada, por ejemplo, se puede definir como el conjunto de entidades *profesor*. De forma similar, el conjunto de entidades *estudiante* podría representar al conjunto de todos los estudiantes de la universidad.

En el proceso de modelado normalmente hablamos de *conjunto de entidades* en abstracto, sin aludir a ningún conjunto de entidades individuales. El término **extensión** de un conjunto de entidades se refiere a la colección real de entidades que pertenecen al conjunto

1 Una asignatura puede haberse impartido en varios semestres, así como varias veces en un mismo semestre. Nos referiremos a esta oferta de asignaturas como una sección.

de entidades en cuestión. Es decir, el conjunto de los profesores reales de la universidad forma la extensión del conjunto de entidades *profesor*. La distinción anterior es similar a la diferencia entre una relación y un ejemplar de relación, como se vio en el Capítulo 2.

Los conjuntos de entidades no son necesariamente disjuntos. Por ejemplo, es posible definir el conjunto de entidades de todas las personas de la universidad (*persona*). Una entidad *persona* puede ser una entidad *profesor;* una entidad *estudiante*, ambas o ninguna.

Cada entidad se representa mediante un conjunto de **atributos.** Los atributos son propiedades descriptivas que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información parecida relativa a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo. Posibles atributos del conjunto de entidades *profesor* son: *ID*, *nombre*, *nombre_dept* y *sueldo*. En la vida real habría más atributos, como el número de la calle, el número del piso, la provincia, el código postal y el país, pero se omiten para no complicar el ejemplo. Posibles atributos del conjunto de entidades *asignatura* son *asignatura_id*, *nombre_asig*, *nombre_dept* y *créditos*.

Cada entidad tiene un **valor** para cada uno de sus atributos. Por ejemplo, una entidad *profesor* concreta puede tener el valor 12121 para *ID*, el valor Wu para *nombre*, el valor Finanzas para *nombre_dept* y el valor 90000 para *sueldo*.

El atributo *ID* se usa para identificar unívocamente a los profesores, dado que puede haber más de un profesor con el mismo nombre. En Estados Unidos, muchas empresas consideran adecuado usar el número *seguridad_social* de las personas[2] como atributo cuyo valor identifica unívocamente a esa persona. En general, la empresa tendría que crear y asignar un identificador unívoco a cada profesor.

Por tanto, las bases de datos incluyen una serie de conjuntos de entidades, cada una de las cuales contiene cierto número de entidades del mismo tipo. La Figura 7.1 muestra parte de una base de datos de la universidad solo con algunos de los atributos de los dos conjuntos de entidades que se muestran.

Las bases de datos de una universidad pueden incluir otros conjuntos de entidades. Por ejemplo, además del seguimiento de los profesores y de los estudiantes, la universidad también tiene información de las asignaturas, que se representan mediante el conjunto de entidades *asignatura* con los atributos *asignatura_id*, *nombre_asig*, *nombre_dept* y *créditos*. En un entorno real, una base de datos de una universidad puede contener docenas de conjuntos de entidades.

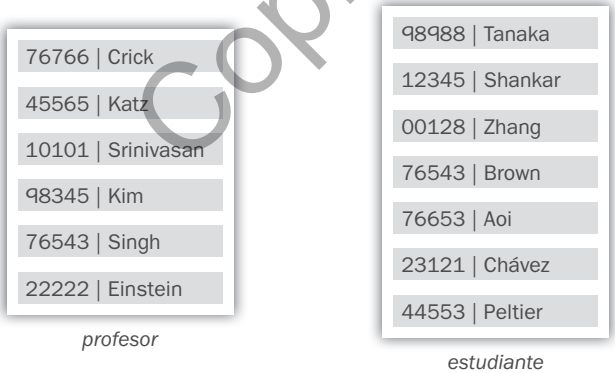



**Figura 7.1.** Conjuntos de entidades *profesor* y *estudiante*.

2 En los Estados Unidos, el Gobierno asigna a cada persona un número único, denominado número de la seguridad social, para su identificación. Se supone que cada individuo tiene un único número de seguridad social y no hay dos personas que puedan tener el mismo número.

## 7.2.2. Conjuntos de relaciones

Una **relación** es una asociación entre varias entidades. Por ejemplo, se puede definir una relación *tutor* que asocie al profesor Katz con el estudiante Shankar. Esta relación especifica que Katz es el tutor del estudiante Shankar.

Un **conjunto de relaciones** es un conjunto de relaciones del mismo tipo. Formalmente, es una relación matemática con $n \geq 2$ de conjuntos de entidades (posiblemente no distintos). Si $E_1, E_2 \ldots, E_n$ son conjuntos de entidades, entonces un conjunto de relaciones $R$ es un subconjunto de:

$$\{(e_1, e_2 \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2 \ldots, e_n \in E_n\}$$

donde $(e_1, e_2 \ldots, e_n)$ es una relación.

Considérense las dos entidades *profesor* y *estudiante* de la Figura 7.1. Se define el conjunto de relaciones *tutor* para denotar la asociación entre los profesores y los estudiantes. La Figura 7.2 muestra esta asociación.

Como ejemplo adicional, considérense los dos conjuntos de entidades *estudiante* y *sección*. Se puede definir el conjunto de relaciones *matricula* para denotar la asociación entre un estudiante y las secciones de asignaturas en que ese estudiante está matriculado.

La asociación entre conjuntos de entidades se conoce como *participación*; es decir, los conjuntos de entidades $E_1, E_2 \ldots, E_n$ **participan** en el conjunto de relaciones $R$. Un **ejemplar de la relación** de un esquema E-R representa una asociación entre las entidades citadas en la empresa real que se está modelando. Como ilustración, la entidad *profesor* Katz, que tiene el identificador *ID* 45565, y la entidad *estudiante* Shankar, que tiene el *ID* 12345, participan en el ejemplar de relación *tutor*. Este ejemplar de relación representa que en la universidad el profesor Katz es el tutor del estudiante Shankar.

La función que desempeña una entidad en una relación se denomina **rol** de esa entidad. Como los conjuntos de entidades que participan en un conjunto de relaciones, generalmente, son distintos, los roles están implícitos y no se suelen especificar. Sin embargo, resultan útiles cuando el significado de una relación necesita aclaración. Tal es el caso cuando los conjuntos de entidades de una relación no son distintos; es decir, el mismo conjunto de entidades participa en un conjunto de relaciones más de una vez, con diferentes roles. En este tipo de conjunto de relaciones, que a veces se denomina conjunto de relaciones **recursivo,** son necesarios los nombres explícitos para los roles con el fin de especificar la manera en que cada entidad participa en cada ejemplar de la relación. Por ejemplo, considérese un conjunto de entidades *asignatura* que almacena información de todas las asignaturas de la universidad. Para indicar la situación donde una asignatura (C2) es un prerrequisito de otra asignatura (C1) se tiene el conjunto de relaciones *prerreq,* que se modela con pares de entidades *asignatura*. La primera asignatura del par tiene el rol de la asignatura C1, mientras que la segunda toma el rol del curso prerrequisito C2. De esta forma, todas las relaciones de *prerreq* se caracterizan por pares (C1, C2); se excluyen los pares (C2, C1).

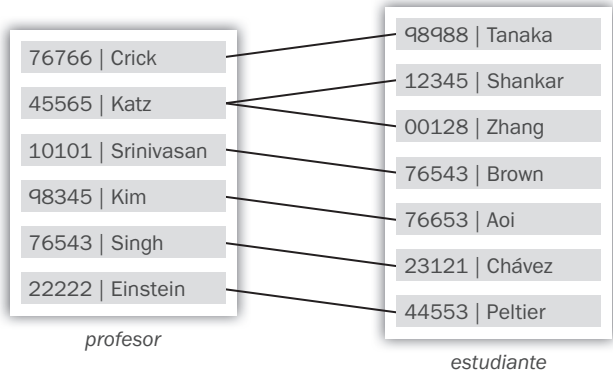



**Figura 7.2.** Conjunto de relaciones *tutor*.

Una relación puede también tener atributos denominados **atributos descriptivos.** Considérese el conjunto de relaciones *tutor* con los conjuntos de entidades *profesor* y *estudiante*. Se puede asociar el atributo *fecha* con esta relación para especificar la fecha cuando un profesor se convirtió en tutor del estudiante. La relación *tutor* entre las entidades correspondientes al profesor Katz y el estudiante Shankar tiene el valor «10 Junio 2007» en el atributo *fecha*, lo que significa que Katz se convirtió en el tutor de Shankar el 10 de junio de 2007.

La Figura 7.3 muestra el conjunto de relaciones *tutor* con el atributo descriptivo *fecha*. Fíjese que Katz tutela a dos estudiantes con distintas fechas de comienzo.

Como ejemplo adicional de los atributos descriptivos de las relaciones, supóngase que se tienen los conjuntos de entidades *estudiante* y *sección* que participan en el conjunto de relaciones *matriculado*. Puede que se desee almacenar el atributo descriptivo *nota* con la relación, para registrar la nota del estudiante en clase. También se puede guardar un atributo descriptivo *para_nota* si el estudiante se ha matriculado en la asignatura para obtener créditos o solo como oyente.

Cada ejemplar de una relación de un conjunto de relaciones determinado debe identificarse unívocamente a partir de sus entidades participantes, sin usar los atributos descriptivos. Para comprender esto, supóngase que deseemos representar todas las fechas en las que un profesor se ha convertido en tutor de un estudiante concreto. El atributo monovalorado *fecha* solo puede almacenar una fecha. No se pueden representar varias fechas mediante varios ejemplares de la relación entre el mismo profesor y el mismo estudiante, ya que los ejemplares de la relación no quedarían identificados unívocamente usando solo las entidades participantes. La forma correcta de tratar este caso es crear el atributo multivalorado *fecha*, que puede almacenar todas las fechas.


**Figura 7.3.** *fecha* como atributo del conjunto de relaciones *tutor*.

Puede haber más de un conjunto de relaciones que implique a los mismos conjuntos de entidades. En nuestro ejemplo, los conjuntos de entidades *profesor* y *estudiante* participan en el conjunto de relaciones *tutor*. Además, supóngase que cada estudiante deba tener otro profesor que sirva como tutor del departamento. Entonces los conjuntos de entidades *profesor* y *estudiante* pueden participar en otro conjunto de relaciones: *tutor_dept*.

Los conjuntos de relaciones *tutor* y *tutor_dept* proporcionan un ejemplo de conjunto de relaciones **binario,** es decir, uno que implica dos conjuntos de entidades. La mayor parte de los conjuntos de relaciones de los sistemas de bases de datos son binarios. A veces, no obstante, los conjuntos de relaciones implican a más de dos conjuntos de entidades.

Por ejemplo, considérense los conjuntos de entidades *proyecto*, que representa todos los proyectos de investigación llevados a cabo en la universidad. Considere los conjuntos de entidades *profesor*, *estudiante* y *proyecto*. Cada proyecto puede tener asociados varios profesores y varios estudiantes. Más aún, cada estudiante que trabaja en un proyecto debe tener asociado un profesor que le guía en el proyecto. Por ahora se van a ignorar las primeras dos relaciones, entre proyecto y profesor y entre proyecto y estudiante. Vamos a centrarnos en la información sobre qué profesor está guiando a qué estudiante en un proyecto dado. Para representar esta información, relacionamos los tres conjuntos de entidades mediante el conjunto de relaciones *proy_direc*, que indica qué estudiante concreto participa en un proyecto bajo la dirección de un profesor.

Fíjese que un estudiante podría tener distintos profesores como guías en distintos proyectos, lo que no se puede capturar mediante una relación binaria entre estudiantes y profesores.

El número de conjuntos de entidades que participan en un conjunto de relaciones es también el **grado** de ese conjunto de relaciones. Los conjuntos de relaciones binarios tienen grado 2; los conjuntos de relaciones ternarios tienen grado 3.

### 7.2.3. Atributos

Para cada atributo hay un conjunto de valores permitidos, denominados **dominio** o **conjunto de valores** de ese atributo. El dominio del atributo *asignatura_id* puede ser el conjunto de todas las cadenas de texto de una cierta longitud. Análogamente, el dominio del atributo *semestre* puede ser el conjunto de todas las cadenas de caracteres del conjunto {Otoño, Invierno, Primavera, Verano}.

Formalmente, cada atributo de un conjunto de entidades es una función que asigna el conjunto de entidades a un dominio. Dado que el conjunto de entidades puede tener varios atributos, cada entidad se puede describir mediante un conjunto de pares (atributo, valor), un par por cada atributo del conjunto de entidades. Por ejemplo, una entidad *profesor* concreta se puede describir mediante el conjunto {(*ID*, 76766), (*nombre*, Crick), (*nombre_dept*, Biología), (*sueldo*, 72000)}, lo que significa que esa entidad describe a una persona llamada Crick cuyo *ID* de profesor es 76766, que es miembro del departamento de Biología con un sueldo de 72.000 €. Ahora se puede ver que existe una integración del esquema abstracto con la empresa real que se está modelando. Los valores de los atributos que describen cada entidad constituyen una parte significativa de los datos almacenados en la base de datos.

Cada atributo, tal y como se usa en el modelo E-R, se puede caracterizar por los siguientes tipos de atributo:

- Atributos **simples** y **compuestos.** En los ejemplos considerados hasta ahora los atributos han sido simples; es decir, no estaban divididos en subpartes. Los atributos **compuestos,** en cambio, se pueden dividir en subpartes (es decir, en otros atributos). Por ejemplo, el atributo *nombre* puede estar estructurado como un atributo compuesto consistente en *nombre*, *primer_apellido* y *segundo_apellido*. Usar atributos compuestos en un esquema de diseño es una buena elección si el usuario desea referirse a un atributo completo en algunas ocasiones y, en otras, solamente a algún componente del atributo. Supóngase que se desease añadir un componente dirección al conjunto de entidades estudiante. La dirección se puede definir como el atributo compuesto *dirección* con los atributos *calle*, *ciudad*, *provincia* y *código postal*.[3] Los atributos compuestos pueden ayudar a agrupar todos los atributos, consiguiendo que los modelos sean más claros.

  Obsérvese también que los atributos compuestos pueden aparecer como una jerarquía. En el atributo compuesto *dirección*, el componente *calle* puede dividirse a su vez en *calle_nombre*, *calle_número* y *piso*. La Figura 7.4 muestra estos ejemplos de atributos compuestos para el conjunto de entidades *profesor*.

3 Supondremos el formato de dirección usado en España y otros muchos países, que incluye un número de código postal.

- Atributos **monovalorados** y **multivalorados.** Todos los atributos que se han especificado en los ejemplos anteriores tienen un único valor para cada entidad concreta. Por ejemplo, el atributo *ID* de *estudiante* para una entidad estudiante específica hace referencia a un único número de *ID* de estudiante. Se dice que estos atributos son **monovalorados.** Puede haber ocasiones en las que un atributo tenga un conjunto de valores para una entidad concreta. Considérese un conjunto de entidades *profesor* con el atributo *número_teléfono*. Cada profesor puede tener cero, uno o varios números de teléfono, y profesores diferentes pueden tener diferente cantidad de teléfonos. Se dice que este tipo de atributo es **multivalorado.** Como ejemplo adicional al conjunto de entidades *profesor* podríamos añadir un atributo que agrupara la lista de *nombre_dependiente*, con todos los que dependen de él. Este atributo es multivalorado, ya que para un profesor dado puede tener cero, uno o más dependiendo de él.

  Para indicar que un atributo es multivalorado se encierra entre llaves, por ejemplo {*número_teléfono*} o {*nombre_dependiente*}. Si resulta necesario, se pueden establecer apropiadamente límites inferior y superior al número de valores en el atributo **multivalorado.** Por ejemplo, una universidad puede limitar a dos el número de teléfonos que se guardan por profesor. El establecimiento de límites en este caso expresa que el atributo *número_teléfono* del conjunto de entidades *profesor* puede tener entre cero y dos valores.
- Atributos **derivados.** El valor de este tipo de atributo se puede obtener a partir del valor de otros atributos o entidades relacionados. Por ejemplo, suponga que el conjunto de entidades *profesor*, que tiene un atributo *estudiantes_tutelados*, representa el número de estudiantes que tiene como tutor. Ese atributo se puede obtener contando el número de entidades *estudiante* asociadas con ese profesor.

  Como ejemplo adicional, supóngase que el conjunto de entidades *profesor* tiene el atributo *edad*, que indica la edad del profesor. Si el conjunto de entidades *profesor* tiene también un atributo *fecha_de_nacimiento*, se puede calcular *edad* a partir de *fecha_de_nacimiento* y de la fecha actual. Por tanto, *edad* es un atributo derivado. En este caso, *fecha_de_nacimiento* puede considerarse un atributo *básico*, o *almacenado*. El valor de los atributos derivados no se almacena, sino que se calcula cada vez que hace falta.

Los atributos toman valores **nulos** cuando las entidades no tienen ningún valor para ese atributo. El valor *nulo* también puede indicar «no aplicable» , es decir, que el valor no existe para esa entidad. Por ejemplo, una persona puede no tener un segundo apellido (si no es español). *Nulo* puede también designar que el valor del atributo es desconocido. Un valor desconocido puede ser *falta* (el valor existe pero no se tiene esa información) o *desconocido* (no se sabe si ese valor existe realmente o no).

Por ejemplo, si el valor *nombre* de un *profesor* dado es *nulo*, se da por supuesto que el valor falta, ya que todos los profesores deben tener nombre. Un valor *nulo* para el atributo *piso* puede significar que la dirección no incluye un piso (no aplicable), que existe el valor piso pero no se conoce cuál es (falta), o que no se sabe si el valor piso forma parte o no de la dirección del profesor (desconocido).

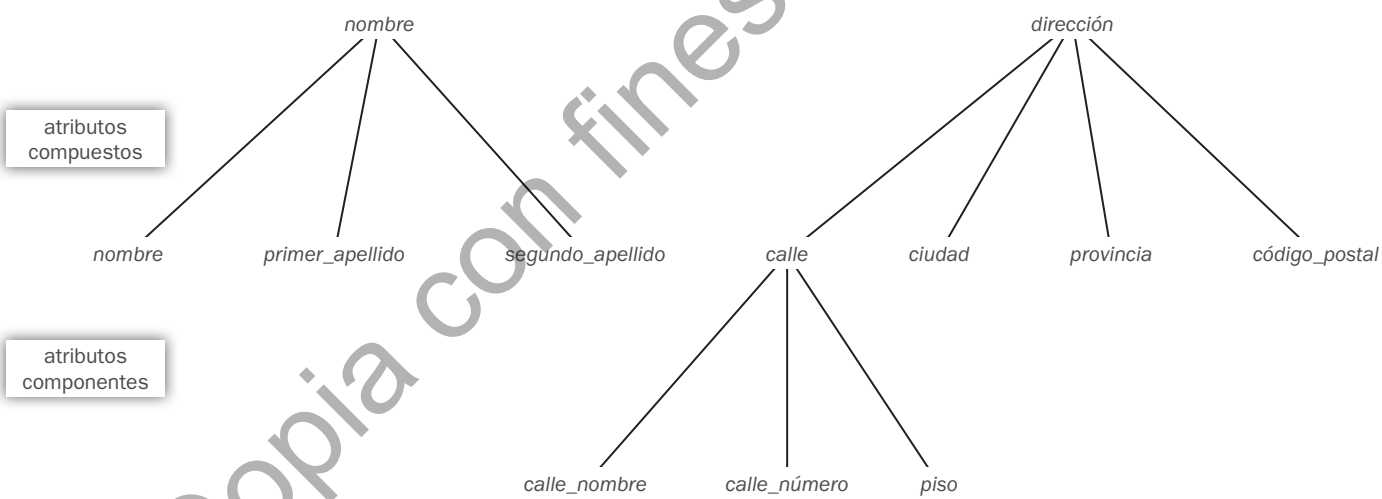



**Figura 7.4.** Atributos compuestos de profesor *nombre* y *dirección*.



## 7.3. Restricciones

Un esquema de empresa E-R puede definir ciertas restricciones que el contenido de la base de datos debe respetar. En este apartado se examinan la correspondencia de cardinalidades y las restricciones de participación

### 7.3.1. Correspondencia de cardinalidades

La **correspondencia de cardinalidades,** o razón de cardinalidad, expresa el número de entidades a las que otra entidad se puede asociar mediante un conjunto de relaciones.

La correspondencia de cardinalidades resulta muy útil para describir conjuntos de relaciones binarias, aunque pueda contribuir a la descripción de conjuntos de relaciones que impliquen más de dos conjuntos de entidades. En este apartado se centrará la atención únicamente en los conjuntos de relaciones binarias.

Para un conjunto de relaciones binarias $R$ entre los conjuntos de entidades $A$ y $B$, la correspondencia de cardinalidades debe ser una de las siguientes:

- **Uno a uno.** Cada entidad de $A$ se asocia, *a lo sumo*, con una entidad de $B$, y cada entidad de $B$ se asocia, *a lo sumo*, con una entidad de $A$ (véase la Figura 7.5a).
- **Uno a varios.** Cada entidad de $A$ se asocia con cualquier número (cero o más) de entidades de $B$. Cada entidad de $B$, sin embargo, se puede asociar, *a lo sumo*, con una entidad de $A$ (véase la Figura 7.5b).

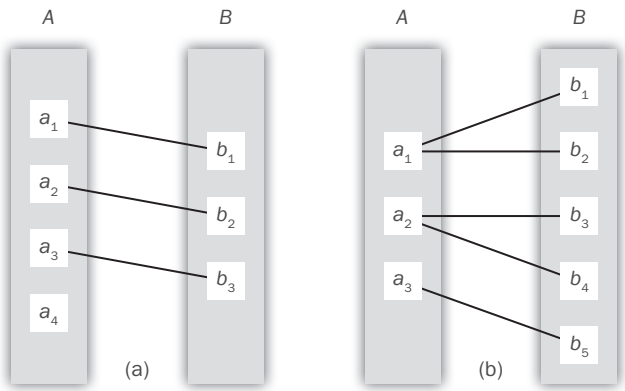



**Figura 7.5.** Correspondencia de cardinalidades (a) Uno a uno. (b) Uno a varios.

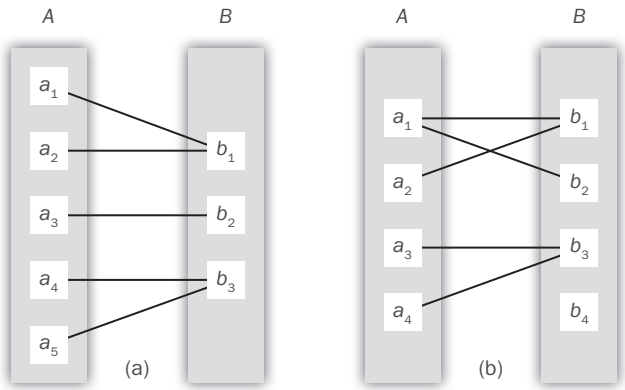



**Figura 7.6.** Correspondencia de cardinalidades. (a) Varios a uno. (b) Varios a varios.

- **Varios a uno.** Cada entidad de *A* se asocia, *a lo sumo*, con una entidad de *B*. Cada entidad de *B*, sin embargo, se puede asociar con cualquier número (cero o más) de entidades de *A* (véase la Figura 7.6a).
- **Varios a varios.** Cada entidad de *A* se asocia con cualquier número (cero o más) de entidades de *B*, y cada entidad de *B* se asocia con cualquier número (cero o más) de entidades de *A* (véase la Figura 7.6b).

La correspondencia de cardinalidades adecuada para un conjunto de relaciones dado depende, obviamente, de la situación del mundo real que el conjunto de relaciones modele.

Como ilustración, considere el conjunto de relaciones *tutor*. Si, en una universidad dada, cada estudiante solo puede tener como tutor a un profesor y cada profesor puede tutelar a varios estudiantes, entonces el conjunto de relaciones de *profesor* a *estudiante* es uno a varios. Si un estudiante puede ser tutelado por varios profesores (como en el caso de tutela compartida), el conjunto de relaciones es varios a varios.

### 7.3.2. Restricciones de participación

Se dice que la participación de un conjunto de entidades *E* en un conjunto de relaciones *R* es **total** si cada entidad de *E* participa, al menos, en una relación de *R*. Si solo algunas entidades de *E* participan en relaciones de *R*, se dice que la participación del conjunto de entidades *E* en la relación *R* es **parcial.** En la Figura 7.5a, la participación de *B* en el conjunto de relaciones es total, mientras que la participación de *A* en el conjunto de relaciones es parcial. En la Figura 7.5b la participación de ambos, tanto *A* como *B*, en el conjunto de relaciones es total.

Por ejemplo, se puede esperar que cada entidad *estudiante* esté relacionada al menos con un profesor mediante la relación *tutor*. Por tanto, la participación de *estudiante* en el conjunto de relaciones *tutor* es total. En cambio, un profesor puede no ser tutor de ningún estudiante. Por tanto, es posible que solo algunas de las entidades *profesor* estén relacionadas con el conjunto de entidades *estudiante* mediante la relación *tutor*, y la participación de *profesor* en la relación *tutor* es, por tanto, parcial.

### 7.3.3. Claves

Es necesario tener una forma de especificar la manera de distinguir las entidades pertenecientes a un conjunto de entidades dado. Conceptualmente, cada entidad es distinta; desde el punto de vista de las bases de datos, sin embargo, la diferencia entre ellas se debe expresar en términos de sus atributos.

Por lo tanto, los valores de los atributos de cada entidad deben ser tales que permitan *identificar unívocamente* a esa entidad. En otras palabras, no se permite que ningún par de entidades de un conjunto de entidades tenga exactamente el mismo valor en todos sus atributos.

La noción de una *clave* para un esquema de relación, tal como se definió en la Sección 2.3, aplica directamente a los conjuntos de entidades. Es decir, una clave para una entidad es un conjunto de atributos que es suficiente para distinguir unas entidades de otras. Los conceptos de superclave, clave candidata y clave primaria se aplican a los conjuntos de entidades solo si son aplicables a los esquemas de relación.

Las claves también ayudan a identificar relaciones de forma única y, por tanto, permiten distinguir unas relaciones de otras. A continuación se definen las nociones correspondientes de claves para las relaciones.

La clave primaria de un conjunto de entidades nos permite distinguir entre varias entidades del conjunto. Necesitamos un mecanismo similar para distinguir entre las distintas relaciones de un conjunto de relaciones.

Sea *R* un conjunto de relaciones que implica a los conjuntos de entidades $E_1$ $E_2$ …, $E_n$. Por ejemplo, *clave-primaria* $(E_i)$ indica el conjunto de atributos que forman la clave primaria de la entidad $E_i$ . Suponga que los nombres de los atributos de todas las claves primarias son únicos. La composición de las claves primarias para un conjunto de relaciones depende del conjunto de atributos asociados con el conjunto de relaciones *R*.

Si el conjunto de relaciones *R* no tiene atributos asociados, entonces el conjunto de atributos

$$\textit{clave-primaria}(E_1) \cup \textit{clave-primaria}(E_2) \cup \ldots \cup \textit{clave-primaria}(E_n)$$

describe una relación individual en el conjunto *R*.

Si el conjunto de relaciones *R* tiene los atributos $a_1$ , $a_2$ …, $a_m$ asociados, entonces el conjunto de atributos

$$\textit{clave-primaria}\ (E_1) \cup \textit{clave-primaria}\ (E_2) \cup \ldots \cup \textit{clave-primaria}(E_n) \cup \{a_1, a_2 \ldots, a_m\}$$

describe una relación individual en el conjunto *R*.

En ambos casos, el conjunto de atributos

$$\textit{clave-primaria}\ (E_1) \cup \textit{clave-primaria}\ (E_2) \cup \ldots \cup \textit{clave-primaria}\ (E_n)$$

forma una superclave del conjunto de relaciones.

Si los nombres de los atributos de las claves primarias no son únicos entre los conjuntos de entidades, los atributos se renombran para distinguirlos; el nombre del conjunto de entidades junto con el nombre del atributo formarán un nombre único. Si un conjunto de entidades participa más de una vez en un conjunto de relaciones (como en la relación *prerreq* en la Sección 7.2.2), en su lugar se usa el rol del nombre, en lugar del nombre del conjunto de entidades, para formar un nombre de atributo que sea único.

La estructura de las claves primarias de un conjunto de relaciones depende de la correspondencia de cardinalidad del conjunto de relaciones. Como ejemplo, suponga el conjunto de entidades *profesor* y *estudiante*, y el conjunto de relaciones *tutor*, con el atributo *fecha*, de la Sección 7.2.2. Suponga que el conjunto de relaciones es de varios a varios. Entonces la clave primaria de *tutor* consistirá en la unión de las claves primarias de *profesor* y *estudiante*. Si la relación es de varios a uno de *estudiante* a *profesor;* es decir, cada estudiante puede tener como mucho un tutor, entonces la clave primaria de *tutor* es simplemente la clave primaria de *estudiante*. Sin embargo, si un profesor puede tutelar solo a un estudiante; es decir, si la relación *tutor* es varios a uno de *profesor* a *estudiante*, entonces la clave primaria de *tutor* es simplemente la clave primaria de *profesor*. Para las relaciones uno a uno se puede usar como clave primaria cualquiera de las claves candidatas.

Para las relaciones no binarias, si no existen restricciones de cardinalidad entonces la superclave formada como se ha descrito anteriormente en esta sección es la única clave candidata, y se elige como clave primaria. La elección de la clave primaria es más complicada si existen restricciones de cardinalidad. Como no se ha tratado cómo especificar las restricciones de cardinalidad para relaciones no binarias, no se va a tratar este tema en este capítulo. Se considerará con más detalle en las Secciones 7.5.5 y 8.4.

## 7.4. Eliminar atributos redundantes de un conjunto de entidades

Cuando se diseña una base de datos usando el modelo E-R, normalmente se comienza identificando los conjuntos de entidades que deberían incluirse. Por ejemplo, en la organización de la universidad que ha tratado hasta ahora, se decidió incluir entidades como *estudiante*, *profesor*, etc. Una vez decididos los conjuntos de entidades hay que elegir sus atributos. Estos atributos se supone que representan los distintos valores que se desea recoger en la base de datos. Para la organización de la universidad se decidió que el conjunto de entidades *profesor* incluyese los atributos *ID*, *nombre*, *nombre_dept* y *sueldo*. Podrían haberse incluido atributos como *número_teléfono*, *número_despacho*, *web_personal*, etc. La elección sobre qué atributos incluir es decisión del diseñador, que tiene el conocimiento adecuado de la estructura de la empresa.

Una vez elegidas las entidades y sus correspondientes atributos, se forman los conjuntos de relaciones entre las distintas entidades. Estos conjuntos de relaciones pueden llevar a situaciones en las que los atributos de algunos conjuntos de entidades sean redundantes y se necesite eliminarlos de los conjuntos de entidades originales. Para mostrarlo, suponga los conjuntos de entidades *profesor* y *departamento*:

- El conjunto de entidades *profesor* incluye los atributos *ID*, *nombre*, *nombre_dept*, y *sueldo*, siendo *ID* la clave primaria.
- El conjunto de entidades *departamento* incluye los atributos *nombre_dept*, *edificio* y *presupuesto*, siendo *nombre_dept* la clave primaria.

Se modela que cada profesor tiene asociado un departamento utilizando el conjunto de relaciones *profesor_dept* relacionando *profesor* y *departamento*.

El atributo *nombre_dept* aparece en ambos conjuntos de entidades. Como es la clave primaria del conjunto de entidades *departamento*, es redundante en el conjunto de entidades *profesor* y habría que eliminarla.

Eliminar el atributo *nombre_dept* del conjunto de entidades *profesor* parece ser un tanto contraintuitivo, ya que la relación *profesor* que se utilizó en el capítulo anterior tenía un atributo *nombre_dept*. Como se verá más adelante, cuando se crea un esquema relacional de un diagrama E-R, el atributo *nombre_dept* de hecho se añade a la relación *profesor*, pero solo si cada profesor tiene asociado como mucho un departamento. Si un profesor tiene asociado más de un departamento, la relación entre profesores y departamentos se registra en una relación distinta *profesor_dept*.

Al realizar la conexión entre profesores y departamentos de manera uniforme como una relación, en lugar de como un atributo de *profesor*, consigue que la relación sea explícita y ayuda a evitar una suposición prematura de que cada profesor está asociado solo con un departamento.

De manera similar, el conjunto de entidades *estudiante* está relacionado con el conjunto de entidades *departamento* mediante la relación *estudiante_dept*, y por tanto no es necesario el atributo *nombre_dept* en *estudiante*.

Como ejemplo adicional, suponga la oferta de cursos (secciones) junto con las franjas horarias de dicha oferta. Cada franja horaria viene identificada con un *franja_horaria_id*, y tiene asociadas un conjunto de horas semanales, cada una identificada por el día de la semana, la hora de inicio y la hora de finalización. Se decide modelar el conjunto de horas semanales como un atributo multivalorado. Suponga que se modelan los conjuntos de entidades *sección* y *franja horaria* de la siguiente forma:

- El conjunto de entidades *sección* incluye los atributos *asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *aula* y *franja_horaria_id*, con (*asignatura_id*, *secc_id*, *año*, *semestre*) como clave primaria.
- El conjunto de entidades *franja_horaria* incluye los atributos *franja_horaria_id*, que es la clave primaria,[4] y un atributo compuesto multivalorado {(*día*, *hora_inicio*, *hora_fin*)}.[5]

Estas entidades se relacionan mediante el conjunto de relaciones *secc_franja_horaria*.

El atributo *franja_horaria_id* aparece en ambos conjuntos de entidades. Como es clave primaria en el conjunto de entidades *franja_horaria*, resulta redundante en el conjunto de entidades *sección* y, por tanto, hay que eliminarlo.

Como ejemplo final, suponga que se tiene un conjunto de entidades *aula*, con los atributos *edificio*, *núm_aula* y *capacidad*, siendo *edificio* y *núm_aula* las claves primarias. Suponga también que se desea establecer un conjunto de relaciones *secc_aula* que relaciona *sección* con *aula*. Entonces los atributos {*edificio*, *núm_aula*} resultan redundantes en el conjunto de entidades *sección*.

En un buen diseño entidad-relación no existen atributos redundantes. Para el ejemplo de la universidad, a continuación se da la lista de conjuntos de entidades y sus atributos, donde se subrayan las claves primarias.

- **aula:** con atributos (*edificio*, *núm_aula*, *capacidad*).
- **departamento:** con atributos (*nombre_dept*, *edificio*, *presupuesto*).
- **asignatura:** con atributos (*asignatura_id*, *nombre_asig*, *créditos*).
- **profesor:** con atributos (*ID*, *nombre*, *sueldo*).
- **sección:** con atributos (*asignatura_id*, *secc_id*, *semestre*, *año*).
- **estudiante:** con atributos (*ID*, *nombre*, *tot_créd*).
- **franja_horaria:** con atributos (*franja_horaria_id*, {(*día*, *hora_inicio*, *hora_fin*)}).

4 Más adelante se verá que para la clave primaria de la relación creada del conjunto de entidades *franja_horaria* se incluye *día* y *hora_inicio*; sin embargo, *día* y *hora_inicio* no forman parte de la clave primaria del conjunto de entidades *franja_horaria*.

5 Opcionalmente, podríamos darle un nombre como *reunión* al atributo compuesto que contiene *día*, *hora_inicio* y *hora_fin*.

Los conjuntos de relaciones del diseño son los siguientes:

- **profesor_dept:** relaciona profesores con departamentos.
- **estudiante_dept:** relaciona estudiantes con departamentos.
- **enseña:** relaciona profesores con secciones.
- **matricula:** relaciona estudiantes con secciones, con el atributo descriptivo *nota*.
- **asignatura_dept:** relaciona asignaturas con departamentos.
- **secc_asignatura:** relaciona secciones con asignaturas.
- **secc_aula:** relaciona secciones con aulas.
- **secc_franja_horaria:** relaciona secciones con franjas horarias.
- **tutor:** relaciona estudiantes con profesores.
- **prerreq:** relaciona asignaturas con prerrequisitos de asignaturas.

Se puede verificar que ninguno de los conjuntos de entidades tiene elementos redundantes. Más aún, se puede comprobar que toda la información (distinta de las restricciones) del esquema relacional de la bases de datos de la universidad, que se vio anteriormente en la Figura 2.8 del Capítulo 2, queda recogida en el diseño anterior, pero con varios atributos del diseño relacional que se han sustituido por relaciones en el diseño E-R.

## 7.5. Diagramas entidad-relación

Como se vio brevemente en la Sección 1.3.3, los **diagramas E-R** pueden expresar gráficamente la estructura lógica general de las bases de datos. Los diagramas E-R son sencillos y claros, cualidades que pueden ser responsables en gran parte de la popularidad del modelo E-R.

### 7.5.1. Estructura básica

Un diagrama E-R consta de los siguientes componentes principales:

- **Rectángulos divididos en dos partes,** que representan conjuntos de entidades. La primera parte, que en este libro aparece con fondo gris, contiene el nombre de los conjuntos de entidades. La segunda parte contiene los nombres de todos los atributos del conjunto de entidades.
- **Rombos,** que representan conjuntos de relaciones.
- **Rectángulos sin dividir**, que representan los atributos de un conjunto de relaciones. Los atributos que forman parte de la clave primaria aparecen subrayados.
- **Líneas,** que unen conjuntos de entidades con conjuntos de relaciones.
- **Líneas discontinuas,** que unen atributos de un conjunto de relaciones con conjuntos de relaciones.
- **Líneas dobles,** que indican la participación total de una entidad en un conjunto de relaciones.
- **Rombos dobles,** que representan conjuntos de relaciones identificadas que se unen a conjuntos de entidades débiles (los conjuntos de relaciones identificadas y los conjuntos de entidades débiles se verán más adelante en la Sección 7.5.6).

Considere el diagrama E-R de la Figura 7.7, que consta de dos conjuntos de entidades, *profesor* y *estudiante* relacionadas mediante el conjunto de relaciones *tutor*. Los atributos asociados con *profesor* son *ID*, *nombre* y *sueldo*. Los atributos asociados con *estudiante* son *ID*, *nombre* y *tot_créd*. En la Figura 7.7, los atributos del conjunto de entidades que son miembros de la clave primaria se pueden ver subrayados.

Si un conjunto de relaciones tiene algunos atributos asociados, entonces dichos atributos se encierran en un rectángulo y se une el rectángulo con una línea discontinua al rombo que representa el conjunto de relaciones. Por ejemplo, en la Figura 7.8 se tiene el atributo descriptivo *fecha* asociado con el conjunto de relaciones *tutor* para indicar la fecha en que el profesor se convirtió en tutor.

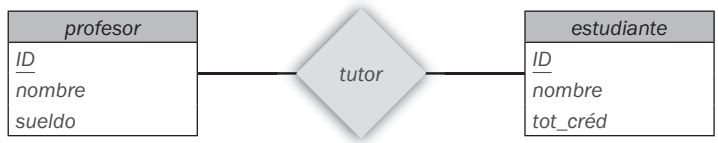



**Figura 7.7.** Diagrama E-R correspondiente a profesores y estudiantes.

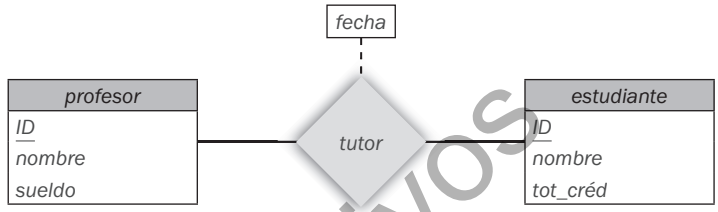



**Figura 7.8.** Diagrama E-R con un atributo asociado al conjunto de relaciones.

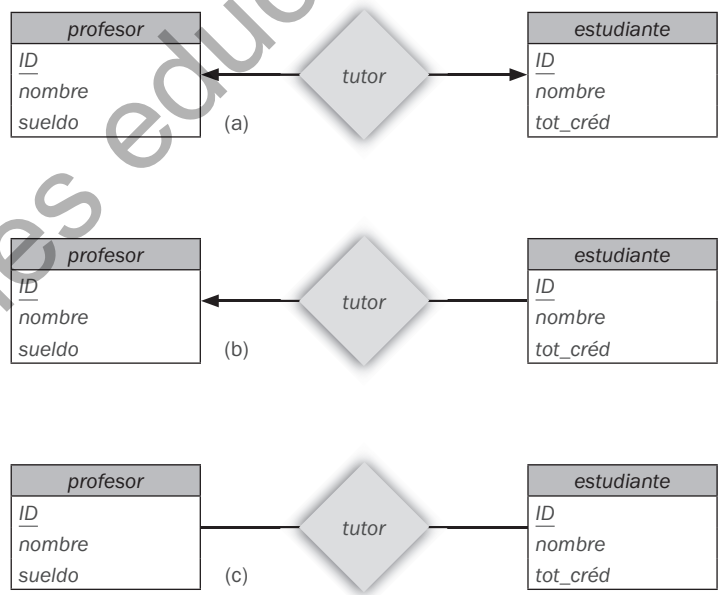



**Figura 7.9.** Relaciones. (a) Uno a uno. (b) Uno a varios. (c) Varios a varios.

### 7.5.2. Correspondencia de cardinalidades

El conjunto de relaciones *tutor*, entre los conjuntos de entidades *profesor* y *estudiante*, puede ser uno a uno, uno a varios, varios a uno o varios a varios. Para distinguir entre estos tipos, se dibuja una línea dirigida (→) o una línea sin dirección (—) entre el conjunto de relaciones y el conjunto de entidades dado, de la siguiente forma:

- **Uno a uno:** se dibuja una línea dirigida desde el conjunto de relaciones *tutor* tanto al conjunto de entidades *profesor* como a *estudiante* (véase la Figura 7.9a). Esto indica que un profesor puede ser tutor de, como mucho, un estudiante, y un estudiante puede tener, como mucho, un tutor.
- **Uno a varios:** se dibuja una línea dirigida desde el conjunto de relaciones *tutor* al conjunto de entidades *profesor* y una línea sin dirección al conjunto de entidades *estudiante* (véase la Figura 7.9b). Esto indica que un profesor puede ser tutor de varios estudiantes, pero un estudiante puede tener, como mucho, un tutor.

- **Varios a uno:** se dibuja una línea sin dirección desde el conjunto de relaciones *tutor* al conjunto de entidades *profesor* y una línea dirigida al conjunto de entidades *estudiante*. Esto indica que un profesor puede ser tutor de, como mucho, un estudiante, pero un estudiante puede tener varios tutores.
- **Varios a varios:** se dibuja una línea sin dirección desde el conjunto de relaciones *tutor* a los conjuntos de entidades *profesor* y *estudiante* (véase la Figura 7.9c). Esto indica que un profesor puede ser tutor de varios estudiantes y un estudiante puede tener varios tutores.

Los diagramas E-R también proporcionan una forma de indicar restricciones más complejas sobre el número de veces que una entidad participa en una relación de un conjunto de relaciones. Una línea puede tener asociada una cardinalidad mínima y máxima, de la forma $l…h$, donde $l$ es el mínimo y $h$ el máximo de cardinalidad. Un valor mínimo de 1 indica la participación total del conjunto de entidades en el conjunto de relaciones; es decir, todas las entidades del conjunto de entidades tienen al menos una relación con el conjunto de relaciones. Un valor máximo de 1 indica que la entidad participa, como mucho, en una relación, mientras que un valor máximo de $*$ indica que no existe límite.

Por ejemplo, considere la Figura 7.10. La línea entre *tutor* y *estudiante* tiene una restricción de cardinalidad de 1…1, lo que significa que el mínimo y el máximo de cardinalidad es 1. Es decir, todos los estudiantes tienen que tener exactamente un tutor. El límite 0…$*$ en la línea entre *tutor* y *profesor* indica que un profesor puede tener cero o más estudiantes. Es decir, la relación *tutor* es uno a varios de *profesor* a *estudiante* y, por tanto, la participación de *estudiante* en *tutor* es una relación total, lo que implica que un estudiante tiene que tener un tutor.

Resulta fácil malinterpretar el 0…$*$ de la parte izquierda y pensar que la relación *tutor* es de varios a uno desde *profesor* a *estudiante*, que es exactamente la interpretación contraria a la correcta.

Si en ambos lados el valor máximo es de 1, la relación es uno a uno. Si se hubiese especificado un límite de cardinalidad de 1…$*$ en el lado izquierdo se podría decir que los profesores deben tutelar al menos a un estudiante.

El diagrama E-R de la Figura 7.10 se podría haber dibujado de forma alternativa con una línea doble de *estudiante* a *tutor* y con una línea dirigida de *tutor* a *profesor*, en lugar de indicar las restricciones de cardinalidad. Este diagrama alternativo indica exactamente las mismas restricciones que las indicadas en la figura.

### 7.5.3. Atributos complejos

En la Figura 7.11 se muestra cómo se pueden representar los atributos compuestos en la notación E-R. En este caso, un atributo compuesto nombre, con los componentes *nombre*, *primer_apellido* y *segundo_apellido* sustituye al atributo simple *nombre* de *profesor*. Como ejemplo adicional, suponga que se desea añadir una dirección al conjunto de entidades *profesor*. La dirección se define como un atributo compuesto *dirección* con los atributos *calle*, *ciudad*, *provincia* y *código postal*. El atributo *calle* a su vez es un atributo compuesto cuyos atributos son *calle_nombre*, *calle_número* y *piso*.

En la Figura 7.11 también se muestra un atributo multivalorado *número_teléfono*, indicado por «{*número_teléfono*}», y un atributo derivado *edad*, indicado con «*edad*( )».

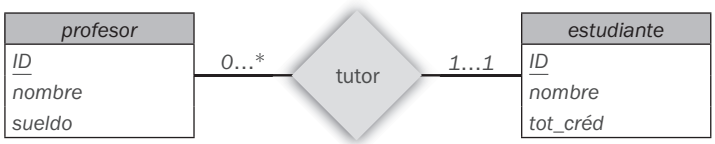



**Figura 7.10.** Límites de cardinalidad en los conjuntos de relaciones.

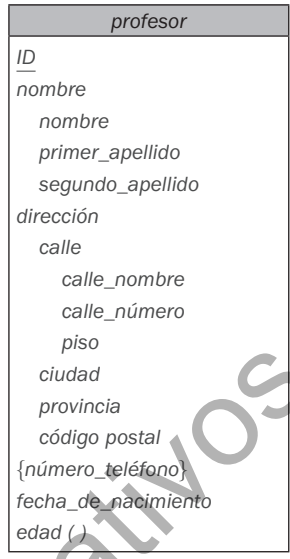



**Figura 7.11.** Diagrama E-R con atributos compuestos, multivalorados y derivados.

### 7.5.4. Roles

Los roles se indican en un diagrama E-R etiquetando las líneas que conectan los rombos con los rectángulos. En la Figura 7.12 se muestran los roles *asignatura_id* y *prerreq_id* entre el conjunto de entidades *asignatura* y el conjunto de relaciones *prerreq*.

### 7.5.5. Conjunto de relaciones no binarias

En un diagrama E-R se pueden especificar fácilmente conjuntos de relaciones no binarias. En la Figura 7.13 se pueden ver tres conjuntos de entidades *profesor*, *estudiante* y *proyecto*, relacionados mediante el conjunto de relaciones *proy_direc*.

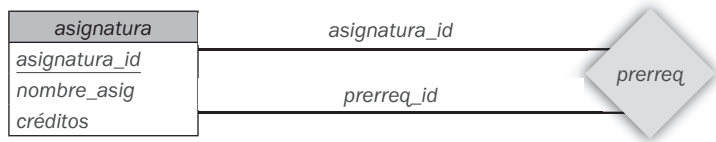



**Figura 7.12.** Diagrama E-R con indicadores de roles.

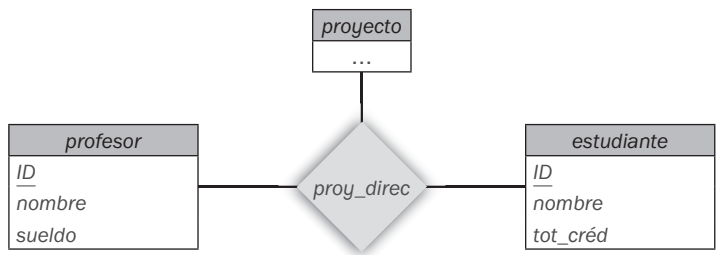



**Figura 7.13.** Diagrama E-R con una relación ternaria.

En el caso de conjuntos de relaciones no binarias se pueden especificar algunos tipos de relaciones varios a uno. Supóngase que un *estudiante* puede tener, a lo sumo, un profesor como director de un proyecto. Esta restricción se puede especificar mediante una flecha que apunte a *profesor* en la línea desde *proy_direc*.

Como máximo se permite una flecha desde cada conjunto de relaciones, ya que los diagramas E-R con dos o más flechas salientes de cada conjunto de relaciones no binarias se pueden interpretar de dos formas. Suponga que hay un conjunto de relaciones $R$ entre los conjuntos de entidades $A_1$, $A_2$ …, $A_n$ y que las únicas flechas están en las líneas hacia los conjuntos de entidades $A_{i+1}$, $A_{i+2}$ …, $A_n$.

Entonces, las dos interpretaciones posibles son:

1. Una combinación concreta de entidades de $A_1, A_2 \ldots, A_i$ se puede asociar, a lo sumo, con una combinación de entidades de $A_{i+1}$, $A_{i+2} \ldots, A_n$. Por tanto, la clave primaria de la relación $R$ se puede crear mediante la unión de las claves primarias de $A_1, A_2 \ldots, A_i$.
2. Para cada conjunto de entidades $A_k$ , $i < k \leq n$, cada combinación de las entidades de los otros conjuntos de entidades se puede asociar, a lo sumo, con una entidad de $A_k$ . Cada conjunto $\{A_1, A_2 \ldots, A_{k-1}, A_{k+1} \ldots, A_n\}$, para $i < k \leq n$, forma, entonces, una clave candidata.

Cada una de estas interpretaciones se ha usado en diferentes libros y sistemas. Para evitar confusiones, solo se permite una flecha saliente de cada conjunto de relaciones, en cuyo caso las dos interpretaciones son equivalentes. En el Capítulo 8 (Sección 8.4) se estudia el concepto de *dependencia funcional*, que permite especificar cualquiera de estas dos interpretaciones sin ambigüedad.

### 7.5.6. Conjunto de entidades débiles

Considere una entidad *sección* identificada unívocamente por un identificador de asignatura, semestre, año e identificador de sección. Claramente, las entidades sección están relacionadas con las entidades asignatura. Suponga que se crea un conjunto de relaciones *secc_asignatura* entre los conjuntos de entidades *sección* y *asignatura*.

Ahora, observe que la información en *secc_asignatura* es redundante, puesto que *sección* ya tiene un atributo *asignatura_id* que identifica el curso con el que dicha asignatura está relacionada. Una opción para combatir esta redundancia es evitar la relación *secc_asignatura*; sin embargo, haciendo esto la relación entre *sección* y *asignatura* se convierte en implícita mediante un atributo, lo que no es deseable.

Una forma alternativa de tratar con esta redundancia es no guardar el atributo *asignatura_id* en la entidad *sección* y guardar solo el resto de los atributos *secc_id*, *año* y *semestre*.[6] Sin embargo, el conjunto de entidades *sección* no tiene entonces suficientes atributos para identificar una entidad *sección* de forma unívoca; aunque cada entidad *sección* es única, las secciones para diferentes asignaturas pueden compartir el mismo *secc_id*, *año* y *semestre*. Para tratar con este problema se crea la relación *secc_asignatura* como una relación especial que proporciona información extra, en este caso el *asignatura_id*, requerido para identificar las entidades *sección* de forma única.

La noción de *conjunto de entidades débil* formaliza la intuición anterior. Se denomina **conjunto de entidades débiles** a los conjuntos de entidades que no tienen suficientes atributos para formar una clave primaria. Los conjuntos de entidades que tienen una clave primaria se denominan **conjuntos de entidades fuertes.**

Para que un conjunto de entidades débiles tenga sentido debe estar asociado con otro conjunto de entidades, denominado **conjunto de entidades identificadoras** o **propietarias.** Cada entidad débil debe asociarse con una entidad identificadora; es decir, se dice que el conjunto de entidades débiles **depende existencialmente** del conjunto de entidades identificadoras. Se dice que el conjunto de entidades identificadoras es **propietario** del conjunto de entidades débiles al que identifica. La relación que asocia el conjunto de entidades débiles con el conjunto de entidades identificadoras se denomina **relación identificadora.**

La relación identificadora es varios a uno del conjunto de entidades débiles al conjunto de entidades identificadoras, y la participación del conjunto de entidades débiles en la relación es total. El conjunto de relaciones identificadoras no debería tener atributos descriptivos, ya que cualquiera de estos atributos se puede asociar con el conjunto de entidades débiles.

En nuestro ejemplo, el conjunto de entidades identificadoras para *sección* es *asignatura*, y la relación *secc_asignatura*, que asocia las entidades *sección* con sus correspondientes entidades *asignatura*, es la relación identificadora.

Aunque los conjuntos de entidades débiles no tienen clave primaria, hace falta un medio para distinguir entre todas las entidades del conjunto de entidades débiles que dependen de una entidad fuerte concreta. El **discriminador** de un conjunto de entidades débiles es un conjunto de atributos que permite que se haga esta distinción. Por ejemplo, el discriminador del conjunto de entidades débiles *sección* consta de los atributos *secc_id*, *año* y *semestre*, ya que, para cada asignatura, este conjunto de atributos identifica de forma única una única sección para dicha asignatura. El discriminador del conjunto de entidades débiles se denomina *clave parcial* del conjunto de entidades.

La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades identificadoras y el discriminador del conjunto de entidades débiles. En el caso del conjunto de entidades *sección*, su clave primaria es {*asignatura_id*, *secc_id*, *año*, *semestre*}, donde *asignatura_id* es la clave primaria del conjunto de entidades identificadoras; es decir, *asignatura* y {*secc_id*, *año*, *semestre*} distingue las entidades *sección* de una misma asignatura.

Se podría haber elegido que *secc_id* fuera globalmente único entre todas las asignaturas que se ofertan en la universidad, en cuyo caso el conjunto de entidades *sección* tendría una clave primaria. Sin embargo, conceptualmente una *sección* seguiría dependiendo de una *asignatura* para que exista, lo que se hace explícito constituyéndola como conjunto de entidades débiles.

En los diagramas E-R, un conjunto de entidades débiles se dibuja con un rectángulo, como un conjunto de entidades fuerte, pero con dos diferencias:

- El discriminador de una entidad débil se subraya con una línea discontinua en lugar de con una continua.
- El conjunto de relaciones que conecta los conjuntos de entidades débiles con el conjunto de entidades fuertes identificadoras se dibuja con un rombo doble.

En la Figura 7.14, el conjunto de entidades débiles *sección* depende del conjunto de entidades fuertes *asignatura* mediante el conjunto de relaciones *secc_asignatura*.

La figura también muestra el uso de las líneas dobles para indicar *participación total*; la participación de un conjunto de entidades (débiles) *sección* en la relación *secc_asignatura* es total, lo que significa que todas las secciones deben estar relacionadas a través de *secc_asignatura* con alguna asignatura. Finalmente, la flecha desde *secc_asignatura* a *asignatura* indica que todas las secciones están relacionadas con una única asignatura.

Los conjuntos de entidades débiles pueden participar en otras relaciones, aparte de en la relación identificadora. Por ejemplo, la entidad *sección* puede participar en una relación con el conjunto de entidades *franja_horaria*, identificando los tiempos en que tiene lugar la reunión para una determinada sección de asignatura. Los conjuntos de entidades débiles pueden participar como propietarios de una relación identificadora con otro conjunto de entidades débiles. También es posible tener conjuntos de entidades débiles con más de un conjunto de entidades identificadoras. Cada entidad débil se identificaría mediante una combinación de entidades, una de cada conjunto de entidades identificadoras. La clave primaria de la entidad débil consistiría en la unión de las claves primarias de los conjuntos de entidades identificadoras y el discriminador del conjunto de entidades débiles.

6 Tenga en cuenta que el esquema relacional que eventualmente se cree del conjunto de entidades *sección* no tiene el atributo *asignatura_id*, por las razones que se verán más claras posteriormente, aunque se haya eliminado el atributo *asignatura_id* del conjunto de entidades *sección*.


**Figura 7.14.** Diagrama E-R con un conjunto de entidades débiles.

En algunos casos, puede que el diseñador de la base de datos decida expresar un conjunto de entidades débiles como atributo compuesto multivalorado del conjunto de entidades propietarias. En el ejemplo, esta alternativa exigiría que el conjunto de entidades *asignatura* tuviera el atributo compuesto y multivalorado *sección*. Los conjuntos de entidades débiles se pueden modelar mejor como atributos si solo participan en la relación identificadora y tienen pocos atributos. A la inversa, las representaciones de los conjuntos de entidades débiles modelarán mejor las situaciones en las que esos conjuntos participen en otras relaciones, aparte de en la relación identificadora, y tengan muchos atributos. Queda claro que *sección* no cumple con los requisitos para ser modelado como un atributo compuesto multivalorado, y resulta más apropiado modelarlo como un conjunto de entidades débiles.

### 7.5.7. Diagrama E-R para la universidad

En la Figura 7.15 se muestra el diagrama E-R que se corresponde con el ejemplo de la universidad que se ha estado utilizando hasta ahora. Este diagrama E-R es equivalente a la descripción textual del modelo E-R de la universidad que se ha visto en la Sección 7.4, pero con algunas restricciones adicionales, y *sección* ahora como entidad débil.

En la base de datos de la universidad se tiene la restricción de que cada profesor ha de estar asociado exactamente a un departamento. Por ello, existe una línea doble en la Figura 7.15 entre *profesor* y *profesor_dept*, indicando la participación total de *profesor* en *profesor_dept*; es decir, todos los profesores tienen que estar asociados a un departamento. Más aún, existe una flecha desde *profesor_dept* a *departamento*, lo que indica que todos los profesores pueden tener, a lo sumo, un departamento asociado.

De forma similar, los conjuntos de entidades *asignatura* y *estudiante* tienen líneas dobles con los conjuntos de relaciones *asignatura_dept* y *estudiante_dept*, respectivamente, así como el conjunto de entidades *sección* con el conjunto de relaciones *secc_franja_horaria*. Las primeras dos relaciones, de hecho, tiene una línea dirigida apuntando a la otra relación, *departamento*, mientras que la tercera relación tiene un línea dirigida apuntando a *franja_horaria*.

Más aún, en la Figura 7.15 se muestra que el conjunto de relaciones *matricula* tiene un atributo descriptivo *nota*, y que todos los estudiantes tienen al menos un tutor. En la figura también se muestra que *sección* es un conjunto de entidades débiles, con atributos *secc_id*, *semestre* y *año* que forman el discriminador; *secc_asignatura* es el conjunto de relaciones identificadoras que relacionan el conjunto de entidades débiles *sección* con el conjunto de entidades fuertes *asignatura*.

En la Sección 7.6 se verá cómo se puede usar este diagrama E-R para derivar los distintos esquemas de relación que usamos.

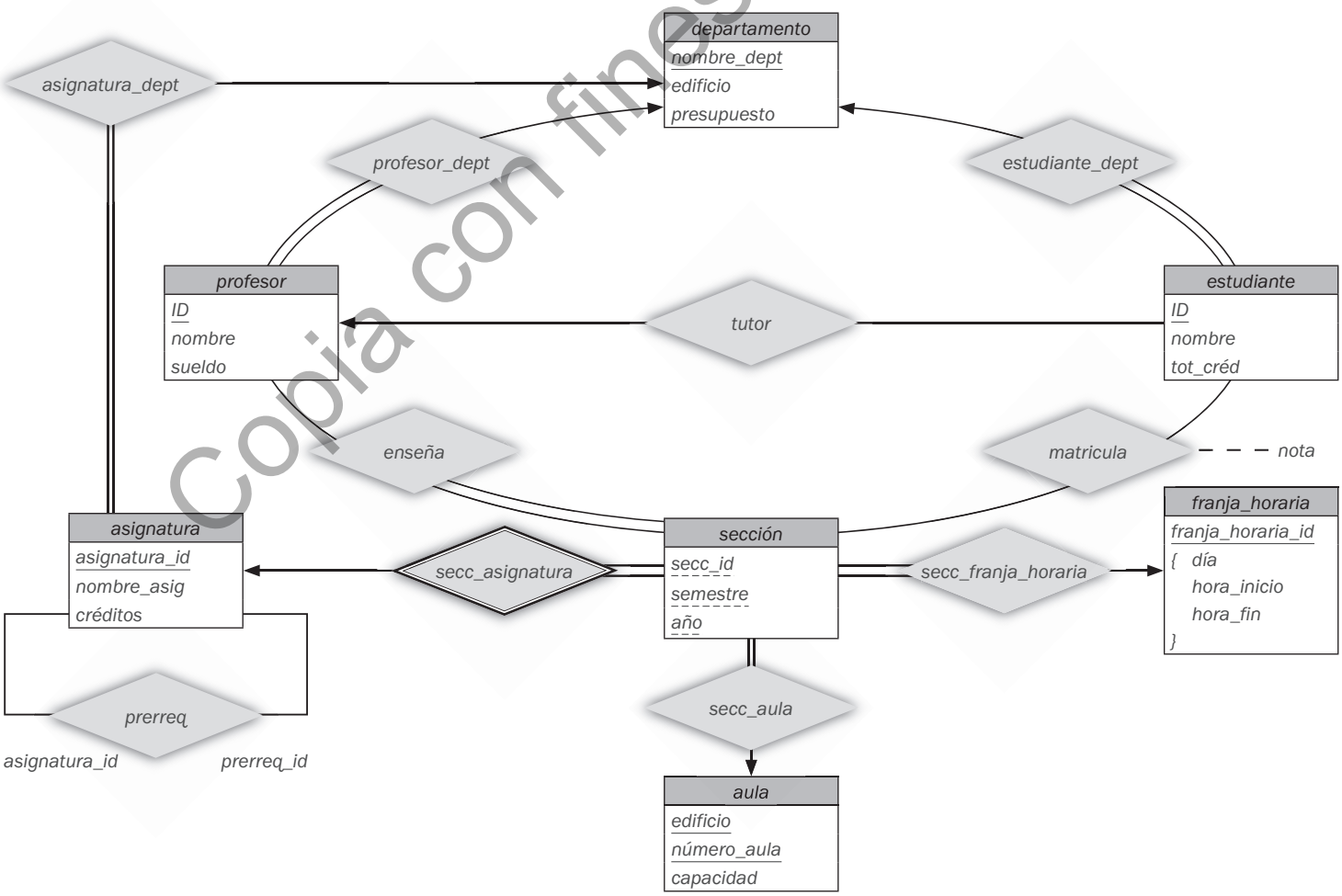



**Figura 7.15.** Diagrama E-R para una universidad.

## 7.6. Reducción a esquemas relacionales

Las bases de datos que se ajustan a un esquema de bases de datos E-R se pueden representar mediante conjuntos de esquemas de relación. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay un solo esquema de relación a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente.

Tanto el modelo E-R de bases de datos como el relacional son representaciones abstractas y lógicas de empresas del mundo real. Como los dos modelos usan principios de diseño parecidos, los diseños E-R se pueden convertir en diseños relacionales.

En esta sección se describe la manera de representar los esquemas E-R mediante esquemas de relación y el modo de asignar las restricciones que surgen del modelo E-R a restricciones de los esquemas de relación.

### 7.6.1. Representación de los conjuntos de entidades fuertes con atributos simples

Sea $E$ un conjunto de entidades fuertes con los atributos descriptivos $a_1$, $a_2$ ..., $a_n$. Esta entidad se representa mediante un esquema denominado $E$ con $n$ atributos distintos. Cada tupla de las relaciones de este esquema corresponde a una entidad del conjunto de entidades $E$.

Para los esquemas derivados de los conjuntos de entidades fuertes, la clave primaria del conjunto de entidades sirve de clave primaria de los esquemas resultantes. Esto se deduce directamente del hecho de que cada tupla corresponde a una entidad concreta del conjunto de entidades.

Como ejemplo, considere el conjunto de entidades *préstamo* del diagrama E-R de la Figura 7.15. Este conjunto de entidades tiene tres atributos: *ID, nombre* y *tot_cred*. Este conjunto de entidades se representa mediante un esquema denominado *estudiante*, con tres atributos:

*estudiante (ID, nombre, tot_créd)*

Observe que, como *ID* es la clave primaria del conjunto de entidades, también es la clave primaria del esquema de la relación.

Siguiendo con el ejemplo, para el diagrama E-R de la Figura 7.15 todos los conjuntos de entidades fuertes, excepto *franja_horaria*, tienen solo atributos simples. Los esquemas que se derivan de estos conjuntos de entidades fuertes son:

*aula (edificio, número_aula, capacidad)*
*departamento (nombre_dept, edificio, presupuesto)*
*asignatura (asignatura_id, nombre_asig, créditos)*
*profesor (ID, nombre, sueldo)*
*estudiante (ID, nombre, tot_créd)*

Como puede observar, tanto el esquema *profesor* como el esquema *estudiante* son diferentes de los esquemas que se han usado en los capítulos anteriores (no contienen el atributo *nombre_dept*). Este aspecto se tratará en breve.

### 7.6.2. Representación de los conjuntos de entidades fuertes con atributos complejos

Cuando un conjunto de entidades fuertes tiene un conjunto de atributos que no son simples, las cosas son un poco más complicadas. Hemos tratado los atributos compuestos creando atributos separados para cada uno de los atributos componentes; no creamos un atributo separado para la composición de sus atributos compuestos. Para mostrar esto, considere la versión del conjunto de entidades *profesor* de la Figura 7.11. Para el atributo compuesto *nombre*, el esquema generado para *profesor* contiene los atributos *nombre*, *primer_apellido* y *segundo_apellido*; no existen atributos separados en el esquema para *nombre*. De forma similar, para el atributo compuesto *dirección* el esquema generado contiene los atributos *calle*, *ciudad*, *provincia* y *código postal*. Como *calle* es un atributo compuesto, se sustituye por *calle_nombre*, *calle_número* y *piso*. Se verá en la Sección 8.2.

Los atributos multivalorados se tratan de forma diferente al resto de atributos. Ya se ha visto que los atributos de un diagrama E-R se corresponden directamente con los atributos de los esquemas de relación apropiados. Sin embargo, los atributos multivalorados son una excepción; se crean nuevos esquemas de relación para ellos, como se verá en breve.

Los atributos derivados no se representan explícitamente en el modelo de datos relacional. Sin embargo, se representan como «métodos» en otros modelos de datos, como el modelo de datos objeto-relacional, que se describirá en el Capítulo 22.

El esquema relacional derivado de la versión del conjunto de entidades *profesor* con atributos compuestos, sin incluir el atributo multivalorado, es por tanto:

*profesor (ID, nombre, primer_apellido, segundo_apellido, calle_nombre, calle_número, piso, ciudad, provincia, código_postal, fecha_de_nacimiento)*

Para un atributo multivalorado $M$, se crea un esquema de relación $R$ con un atributo $A$ que corresponde a $M$ y a los atributos correspondientes a la clave primaria del conjunto de entidades o de relaciones del que $M$ es atributo.

Como ejemplo, considere el diagrama E-R de la Figura 7.11, donde se muestra el conjunto de entidades *profesor* con el atributo multivalorado *número_teléfono*. La clave primaria de *profesor* es *ID*. Para este atributo multivalorado se crea el esquema de relación

*profesor_teléfono (ID, número_teléfono)*

Cada número de teléfono de un profesor se representa como una única tupla de la relación de este esquema. Por tanto, si existe un profesor con *ID* 22222 y números de teléfono 655-1234 y 655-4321, la relación *profesor_teléfono* debería contener las dos tuplas (22222, 555-1234) y (22222, 555-4321).

Se crea una clave primaria del esquema de la relación consistente en todos los atributos del esquema. En el ejemplo anterior, la clave primaria consiste en todos los atributos de la relación *profesor_teléfono*.

Además, se crea una restricción de clave externa para el esquema de la relación, a partir del atributo multivalorado generado por efecto de la clave primaria del conjunto de entidades que hace referencia a la relación generada desde el conjunto de entidades. En el ejemplo anterior, la restricción de clave externa en la relación *profesor_teléfono* sería que el atributo *ID* hiciera referencia a la relación *profesor*.

En el caso de que el conjunto de entidades constara solo de dos atributos, un atributo de clave primaria $B$ y un único atributo multivalorado $M$, el esquema de relación para el conjunto de entidades debería contener solamente un atributo, el atributo de clave primaria $B$. Se puede eliminar esta relación manteniendo el esquema de relación con el atributo $B$ y el atributo $A$ que corresponda con $M$.

Como ejemplo, considere el conjunto de entidades *franja_horaria* que se muestra en la Figura 7.15. Aquí, *franja_horaria_id* es la clave primaria del conjunto de entidades *franja_horaria* y existe un único atributo multivalorado que también resulta que es compuesto. El conjunto de entidades se puede representar por el siguiente esquema creado a partir del atributo compuesto multivalorado:

*franja_horaria(franja_horaria_id, día, hora_inicio, hora_fin)*

Aunque no se represente como una restricción en el diagrama E-R, se sabe que no puede haber dos convocatorias de clases que empiecen a la misma hora el mismo día de la semana, pero terminen a horas diferentes; según esta restricción, *hora_fin* se ha eliminado de la clave primaria del esquema *franja_horaria*.

La relación creada a partir del conjunto de entidades solo tendría un único atributo *franja_horaria_id*; la optimización al eliminar esta relación tiene la ventaja de la simplificación del esquema de la base de datos resultante, aunque tiene la desventaja relativa a las claves externas, que se tratará brevemente en la Sección 7.6.4.

### 7.6.3. Representación de los conjuntos de entidades débiles

Sea $A$ un conjunto de entidades débiles con los atributos $a_1, a_2 \ldots, a_m$. Sea $B$ el conjunto de entidades fuertes del que $A$ depende. La clave primaria de $B$ consiste en los atributos $b_1, b_2 \ldots, b_n$. El conjunto de entidades $A$ se representa mediante el esquema de relación denominado $A$ con un atributo por cada miembro del conjunto:

$$\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$$

Para los esquemas derivados de conjuntos de entidades débiles la combinación de la clave primaria del conjunto de entidades fuertes y del discriminador del conjunto de entidades débiles sirve de clave primaria del esquema. Además de crear una clave primaria, también se crea una restricción de clave externa para la relación $A$, que especifica que los atributos $b_1, b_2 \ldots, b_n$ hacen referencia a la clave primaria de la relación $B$. La restricción de clave externa garantiza que por cada tupla que representa a una entidad débil existe la tupla correspondiente que representa a la entidad fuerte correspondiente.

Como ejemplo, considere el conjunto de entidades débiles *sección* del diagrama E-R de la Figura 7.15. Este conjunto de entidades tiene los atributos: *secc_id*, *semestre* y *año*. La clave primaria del conjunto de entidades *sección*, de la que depende *sección*, es *asignatura_id*. Por tanto, *sección* se representa mediante un esquema con los siguientes atributos:

*sección*(<u>*asignatura_id*</u>, <u>*secc_id*</u>, <u>*semestre*</u>, <u>*año*</u>)

La clave primaria consiste en la clave primaria del conjunto de entidades *asignatura* y el discriminador de *sección*, que es *secc_id*, *semestre* y *año*. También se crea una restricción de clave externa para el esquema *sección*, con el atributo *asignatura_id*, que hace referencia a la clave primaria del esquema *asignatura*, y la restricción de integridad «on delete cascade».[7] Por la especificación «on delete cascade» sobre la restricción de clave externa, si una entidad de asignatura se borra, entonces también se borran las entidades *sección* asociadas.

### 7.6.4. Representación de conjuntos de relaciones

Sea $R$ un conjunto de relaciones, sea $a_1, a_2 \ldots, a_m$ el conjunto de atributos formado por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en $R$, y sean $b_1, b_2 \ldots, b_n$ los atributos descriptivos de $R$ (si los hay). El conjunto de relaciones se representa mediante el esquema de relación $R$, con un atributo por cada uno de los miembros del conjunto:

$$\{a_1, a_2 \ldots, a_m\} \cup \{b_1, b_2 \ldots, b_n\}$$

Ya se ha descrito, en la Sección 7.3.3, la manera de escoger la clave primaria de un conjunto de relaciones binarias. Como se vio en ese apartado, tomar todos los atributos de las claves primarias de todos los conjuntos de entidades primarias sirve para identificar una tupla concreta pero, para los conjuntos de relaciones uno a uno, varios a uno y uno a varios, da como resultado un conjunto de atributos mayor del que hace falta para la clave primaria. En su lugar, la clave primaria se escoge de la siguiente forma:

- Para las relaciones binarias varios a varios la unión de los atributos de clave primaria de los conjuntos de entidades participantes pasa a ser la clave primaria.
- Para los conjuntos de relaciones binarias uno a uno la clave primaria de cualquiera de los conjuntos de entidades puede escogerse como clave primaria de la relación. La elección puede realizarse de manera arbitraria.
- Para los conjuntos de relaciones binarias varios a uno o uno a varios la clave primaria del conjunto de entidades de la parte «varios» de la relación sirve de clave primaria.
- Para los conjuntos de relaciones $n$-arias sin flechas en las líneas la unión de los atributos de clave primaria de los conjuntos de entidades participantes pasa a ser la clave primaria.
- Para los conjuntos de relaciones $n$-arias con una flecha en una de las líneas, las claves primarias de los conjuntos de entidades que no están en el lado «flecha» del conjunto de relaciones sirven de clave primaria del esquema. Recuerde que solo se permite una flecha saliente por cada conjunto de relaciones.

También se crean restricciones de clave externa para el esquema de relación $R$ de la siguiente forma: para cada conjunto de entidades $E_i$ relacionado con el conjunto de relaciones $R$ se crea una restricción de clave externa de la relación $R$, con los atributos de $R$ que se derivan de los atributos de clave primaria de $E_i$ que hacen referencia a la clave primaria del esquema de relación que representa $E_i$.

Como ejemplo, considere el conjunto de relaciones *tutor* del diagrama E-R de la Figura 7.15. Este conjunto de relaciones implica a los dos conjuntos de entidades siguientes:

- *profesor* con clave primaria *ID*.
- *estudiante* con clave primaria *ID*.

Como el conjunto de relaciones no tiene ningún atributo, el esquema *tutor* tiene dos atributos, las claves primarias de *profesor* y *estudiante*. Como el conjunto de relaciones de *tutor* es del tipo varios a uno, se renombran a *p_ID* y *e_ID*.

También creamos dos restricciones de clave externa para la relación *tutor*, con el atributo *p_ID* referenciando a la clave primaria de *profesor* y el atributo *e_ID* referenciando a la clave primaria de *estudiante*.

Continuando con el ejemplo, para el diagrama E-R de la Figura 7.15, los esquemas derivados del conjunto de relaciones se muestran en la Figura 7.16.

Observe que para el caso de los conjuntos de relaciones *prerreq*, los indicadores de rol asociados con las relaciones se usan como nombre de atributo, ya que ambos roles se refieren a la misma relación *asignatura*.

De forma similar al caso de *tutor*, la clave primaria de cada una de las relaciones *secc_asignatura*, *secc_franja_horaria*, *secc_aula*, *profesor_dept*, *estudiante_dept* y *asignatura_dept* consta de las claves primarias de uno solo de los dos conjuntos de entidades relacionadas, ya que las relaciones correspondientes son del tipo varios a uno.

Las claves externas no se muestran en la Figura 7.16, pero para cada relación de la figura hay dos restricciones de clave externa, referenciando las dos relaciones creadas de los dos conjuntos de entidades relacionados. Por ejemplo, *secc_asignatura* tiene claves externas que remiten a *sección* y *aula*, *enseña* tiene claves externas que hacen referencia a *profesor* y *sección*, y *matricula* tiene claves externas que refieren a *estudiante* y *sección*.

7 La función «on delete cascade» de las restricciones de clave externa en SQL se describen en la Sección 4.4.5.

La optimización que nos ha permitido crear solo un único esquema de relación del conjunto de entidades *franja_horaria*, que es un atributo multivalorado, evita la creación de claves externas desde el esquema de relación *secc_franja_horaria* a la relación creada desde el conjunto de entidades *franja_horaria*, ya que se elimina la relación creada desde el conjunto de entidades *franja_horaria*. Se mantiene la relación creada desde el atributo multivalorado, y se nombra como *franja_horaria*, pero esta relación puede potencialmente no tener tuplas correspondientes a una *franja_horaria_id* o puede tener múltiples tuplas correspondientes a una *franja_horaria_id*, por lo que *franja_horaria_id* en *secc_franja_horaria* no puede referenciar esta relación.

El lector avispado puede preguntarse por qué hemos visto los esquemas *secc_asignatura*, *secc_franja_horaria*, *secc_aula*, *profesor_dept*, *estudiante_dept* y *asignatura_dept* en capítulos anteriores. La razón es que el algoritmo que se ha presentado hasta aquí genera algunos esquemas que se pueden eliminar o combinarse con otros. Estos aspectos se tratarán a continuación.

*enseña (ID, asignatura_id, secc_id, semestre, año)*

*matricula (ID, asignatura_id, secc_id, semestre, año, nota)*

*prerreq (asignatura_id, prerreq_id)*

*tutor (e_ID, p_ID)*

*secc_asignatura (asignatura_id, secc_id, semestre, año)*

*secc_franja_horaria (asignatura_id, secc_id, semestre, año, franja_horaria_id)*

*secc_aula (asignatura_id, secc_id, semestre, año, edificio, número_aula)*

*profesor_dept (ID, nombre_dept)*

*estudiante_dept (ID, nombre_dept)*

*asignatura_dept (asignatura_id, nombre_dept)*

**Figura 7.16.** Esquemas derivados del conjunto de relaciones del diagrama E-R de la Figura 7.15.

#### 7.6.4.1. Redundancia de esquemas

Los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con el conjunto correspondiente de entidades fuertes se tratan de manera especial. Como se hizo notar en la Sección 7.5.6, estas relaciones son del tipo varios a uno y no tienen atributos descriptivos. Además, la clave primaria de los conjuntos de entidades débiles incluye la clave primaria de los conjuntos de entidades fuertes. En el diagrama E-R de la Figura 7.14, el conjunto de entidades débiles *sección* depende del conjunto de entidades fuertes *asignatura* a través del conjunto de relaciones *secc_asignatura*. La clave primaria de *sección* es {*asignatura_id*, *secc_id*, *semestre*, *año*} y la clave primaria de *asignatura* es *asignatura_id*. Como *secc_asignatura* no tiene atributos descriptivos, el esquema *secc_asignatura* tiene los atributos, *asignatura_id*, *secc_id*, *semestre* y *año*. El esquema del conjunto de entidades *sección* tiene los atributos *asignatura_id*, *secc_id*, *semestre* y *año* (entre otros). Cada combinación (*asignatura_id*, *secc_id*, *semestre*, *año*) de una relación de *secc_asignatura* también se halla presente en el esquema de relación *sección*, y viceversa. Por tanto, el esquema *secc_asignatura* es redundante.

En general, el esquema de los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con su conjunto correspondiente de entidades fuertes es redundante y no hace falta que esté presente en el diseño de la base de datos relacional basado en el diagrama E-R.

#### 7.6.4.2. Combinación de esquemas

Considérese un conjunto *AB* de relaciones varios a uno del conjunto de entidades *A* al conjunto de entidades *B*. Usando el esquema de construcción de esquemas de relación descrito previamente se consiguen tres esquemas: *A*, *B* y *AB*. Entonces, los esquemas *A* y *B* se pueden combinar para formar un solo esquema consistente en la unión de los atributos de los dos esquemas. La clave primaria del esquema combinado es la clave primaria del conjunto de entidades en cuyo esquema se combinó el conjunto de relaciones.

Como ejemplo, examinemos las distintas relaciones del diagrama E-R de la Figura 7.15, que satisfacen los criterios anteriores:

- *profesor_dept*. Los esquemas *profesor* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *profesor_dept* se puede combinar con el esquema *profesor*. El esquema *profesor* resultado consta de los atributos {*ID*, *nombre*, *nombre_dept*, *sueldo*}.
- *estudiante_dept*. Los esquemas *estudiante* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *estudiante_dept* se puede combinar con el esquema *estudiante*. El esquema *estudiante* resultado consta de los atributos {*ID*, *nombre*, *nombre_dept*, *tot_créd*}.
- *asignatura_dept*. Los esquemas *asignatura* y *departamento* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *asignatura_dept* se puede combinar con el esquema *asignatura*. El esquema *asignatura* resultado consta de los atributos {*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*}.
- *secc_aula*. Los esquemas *sección* y *aula* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *secc_aula* se puede combinar con el esquema *sección*. El esquema *sección* resultado consta de los atributos {*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*}.
- *secc_franja_horaria*. Los esquemas *sección* y *franja_horaria* corresponden a los conjuntos de entidades *A* y *B*, respectivamente. Entonces, el esquema *secc_franja_horaria* se puede combinar con el esquema *sección* obtenido en el paso previo. El esquema *sección* resultado consta de los atributos {*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*, *franja_horaria_id*}.

En el caso de las relaciones uno a uno, el esquema de relación del conjunto de relaciones puede combinarse con el esquema de cualquiera de los conjuntos de entidades.

Se pueden combinar esquemas aunque la participación sea parcial, usando los valores nulos. En el ejemplo anterior, si *profesor_dept* fuese parcial se almacenarían valores nulos para el atributo *nombre_dept* de los profesores que no tuviesen asociado un departamento.

Por último, vamos a considerar las restricciones de clave externa que habrían aparecido en el esquema que representa a los conjuntos de relaciones. Habría habido restricciones de clave externa referenciando a cada uno de los conjuntos de entidades que participan en los conjuntos de relaciones. Se eliminan las restricciones referenciando los conjuntos de entidades en aquellos esquemas en que se combina el esquema del conjunto de relaciones, y se añade la otra restricción de clave externa al esquema combinado. Por ejemplo, *profesor_dept* tiene una restricción de clave externa del atributo *nombre_dept* referenciando la relación *departamento*. Esta restricción externa se añade a la relación *profesor* cuando el esquema *profesor_dept* se combina en *profesor*.

## 7.7. Aspectos del diseño entidad-relación

Los conceptos de conjunto de entidades y de conjunto de relaciones no son precisos, y es posible definir un conjunto de entidades y las relaciones entre ellas de diferentes formas. En esta sección se examinan aspectos básicos del diseño de esquemas de bases de datos E-R. La Sección 7.10 trata el proceso de diseño con más detalle.

### 7.7.1. Uso de conjuntos de entidades en lugar de atributos

Considérese el conjunto de entidades *profesor* con el atributo adicional *número_teléfono* (Figura 7.17a). Se puede argumentar que un *teléfono* es una entidad en sí misma con los atributos *número_teléfono* y *ubicación*; la ubicación puede ser la oficina o el domicilio en el que el teléfono está instalado, y se pueden representar los teléfonos móviles (celulares) mediante el valor «móvil». Si se adopta este punto de vista, no añadimos el atributo *número_teléfono* a *profesor*. En su lugar se crea:

- El conjunto de entidades *teléfono* con los atributos *número_teléfono* y *ubicación*.
- Un conjunto de relaciones *profesor_teléfono*, que denota la asociación entre los profesores y sus teléfonos.

  Esta alternativa es la que se muestra en la Figura 7.17b.

¿Cuál es entonces la diferencia principal entre estas dos definiciones de profesor? Tratar el teléfono como un atributo *número_teléfono* implica que cada profesor tiene exactamente un número de teléfono. Tratar el teléfono como una entidad *teléfono* permite que los empleados tengan varios números de teléfono (incluyendo ninguno) asociados. Sin embargo, se puede definir fácilmente en su lugar *número_teléfono* como atributo multivalorado para permitir varios teléfonos por profesor.

La diferencia principal es que tratar el teléfono como entidad modela mejor una situación en la que se puede querer guardar información extra sobre el teléfono, como su ubicación, su tipo (móvil, teléfono IP o fijo) o las personas que lo comparten. Por tanto, tratar el teléfono como entidad es más general que tratarlo como atributo y resulta adecuado cuando la generalidad pueda ser útil.

En cambio, no sería apropiado tratar el atributo *nombre* (de un profesor) como entidad; es difícil argumentar que *nombre* sea una entidad en sí misma (a diferencia de lo que ocurre con *teléfono*). Así pues, resulta adecuado tener *nombre* como atributo del conjunto de entidades *profesor*.

Por tanto, se suscitan dos preguntas: ¿qué constituye un atributo? y ¿qué constituye un conjunto de entidades? Desafortunadamente no hay respuestas sencillas. Las distinciones dependen principalmente de la estructura de la empresa real que se esté modelando y de la semántica asociada con el atributo en cuestión.

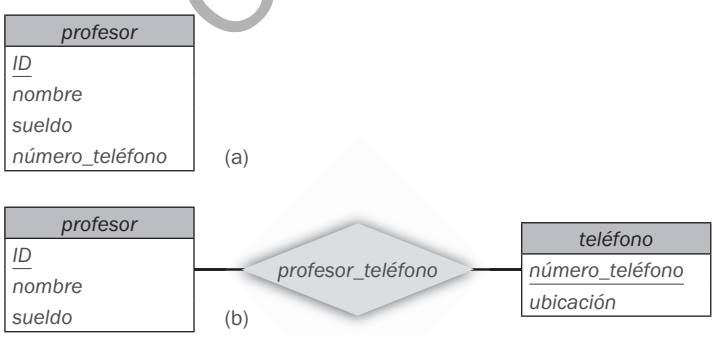



**Figura 7.17.** Alternativas para añadir *teléfono* al conjunto de entidades *profesor*.

Un error común es usar la clave primaria de un conjunto de entidades como atributo de otro conjunto de entidades en lugar de usar una relación. Por ejemplo, es incorrecto modelar el *ID* de un estudiante como atributo de *profesor*, aunque cada profesor solo tutele a un estudiante. La relación *tutor* es la forma correcta de representar la conexión entre estudiantes y profesores, ya que hace explícita su conexión, en lugar de dejarla implícita mediante un atributo.

Otro error relacionado con este que se comete a veces es escoger los atributos de clave primaria de los conjuntos de entidades relacionados como atributos del conjunto de relaciones. Por ejemplo, *ID* (el atributo de clave primaria de *estudiante*) e *ID* (la clave primaria de *profesor*) no deben aparecer como atributos de la relación *tutor*. Esto no es adecuado, ya que los atributos de clave primaria están implícitos en el conjunto de relaciones.[8]

### 7.7.2. Uso de conjuntos de entidades en lugar de conjuntos de relaciones

No siempre está claro si es mejor expresar un objeto mediante un conjunto de entidades o mediante un conjunto de relaciones. En la Figura 7.15 se usa el conjunto de relaciones *matricula* para modelar la situación en la que un estudiante se matricula de una (sección de) asignatura. Una alternativa es imaginar que existe un registro de los registro-asignatura de cada curso en que se matricula un estudiante. Entonces tenemos un conjunto de entidades que representan los registros registro-asignatura. Llamemos a este conjunto de entidades *registro*. Las entidades *registro* están relacionadas con, exactamente, un estudiante y una sección, por lo que tenemos dos conjuntos de relaciones, uno que relaciona registro-asignatura con estudiantes y otro que relaciona registro-asignatura con secciones. En la Figura 7.18 se muestran los conjuntos de entidades *sección* y *estudiante* de la Figura 7.15 con los conjuntos de relaciones *matricula* sustituidos por un conjunto de entidades y dos conjuntos de relaciones:

- *registro*, el conjunto de entidades que representan los rubros de registro-asignatura.
- *sección_reg*, el conjunto de relaciones que relacionan *registro* y *asignatura*.
- *estudiante_reg*, el conjunto de relaciones que relacionan *registro* y *estudiante*.

Se han utilizado líneas dobles para indicar la participación total de las entidades *registro*.

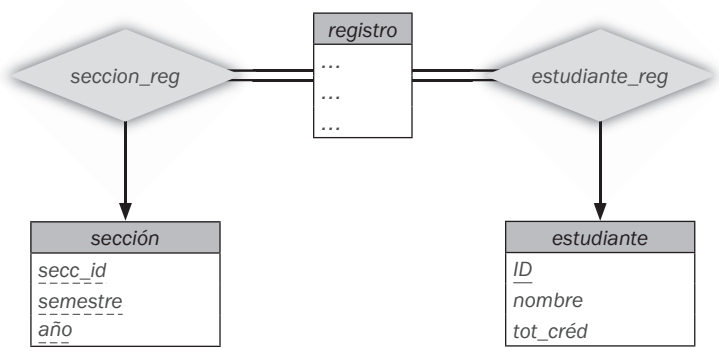



**Figura 7.18.** Sustitución de *matricula* por registro y dos conjuntos de relaciones.

8 Cuando se crea un esquema de relación a partir del esquema E-R, los atributos pueden aparecer en un esquema generado desde el conjunto de relaciones *tutor*, como se verá más adelante; no obstante, no deben aparecer en el conjunto de relaciones *tutor*.

Tanto la forma de la Figura 7.15 como la Figura 7.18 representan con precisión la información de la universidad, pero el uso de *matricula* es más compacto y probablemente preferible. Sin embargo, si la oficina de matriculación asocia otra información con un registro de matricula, podría ser mejor hacer de ello una entidad propia.

Un criterio para determinar si se debe usar un conjunto de entidades o un conjunto de relaciones puede ser escoger un conjunto de relaciones para describir las acciones que se produzcan entre entidades. Este enfoque también puede ser útil para decidir si ciertos atributos se pueden expresar mejor como relaciones.

### 7.7.3. Conjuntos de relaciones binarias y *n*-arias

Las relaciones en las bases de datos suelen ser binarias. Puede que algunas relaciones que no parecen ser binarias se puedan representar mejor mediante varias relaciones binarias. Por ejemplo, se puede crear la relación ternaria *padres*, que relaciona a cada hijo con su padre y con su madre. Sin embargo, esa relación se puede representar mediante dos relaciones binarias, *padre* y *madre*, que relacionan a cada hijo con su padre y con su madre por separado. El uso de las dos relaciones *padre* y *madre* permite el registro de la madre del niño aunque no se conozca la identidad del padre; si se usara en la relación ternaria *padres* se necesitaría un valor nulo. En este caso es preferible usar conjuntos de relaciones binarias.

De hecho, siempre es posible sustituir los conjuntos de relaciones no binarias ($n$-aria, para $n > 2$) por varios conjuntos de relaciones binarias. Por simplificar, considere el conjunto de relaciones abstracto ternario ($n = 3$) $R$ y los conjuntos de entidades $A$, $B$ y $C$. Se sustituye el conjunto de relaciones $R$ por un conjunto de entidades $E$ y se crean tres conjuntos de relaciones como se muestra en la Figura 7.19:

- $R_A$, que relaciona $E$ y $A$.
- $R_B$, que relaciona $E$ y $B$.
- $R_C$, que relaciona $E$ y $C$.

Si el conjunto de relaciones $R$ tiene atributos, estos se asignan al conjunto de entidades $E$; además, se crea un atributo de identificación especial para $E$ (ya que se deben poder distinguir las diferentes entidades de cada conjunto de entidades con base en los valores de sus atributos). Para cada relación $(a_i, b_i, c_i)$ del conjunto de relaciones $R$ se crea una nueva entidad $e_i$ del conjunto de entidades $E$. Luego, en cada uno de los tres nuevos conjuntos de relaciones, se inserta una relación del modo siguiente:

- $(e_i, a_i)$ en $R_A$.
- $(e_i, b_i)$ en $R_B$.
- $(e_i, c_i)$ en $R_C$.

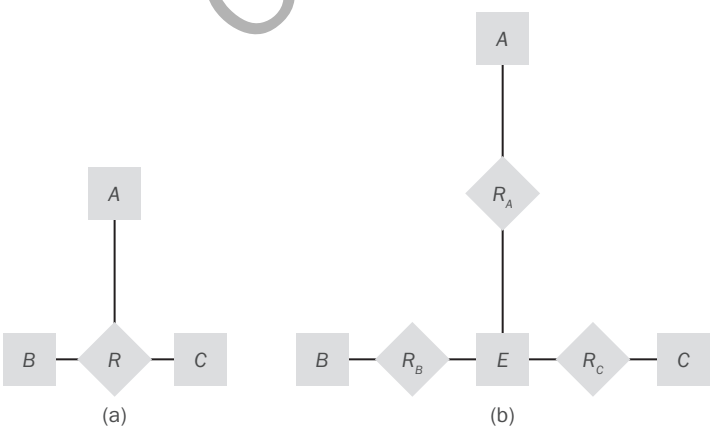



**Figura 7.19.** Relaciones ternarias frente a tres relaciones binarias.

Este proceso se puede generalizar de forma directa a los conjuntos de relaciones $n$-arias. Por tanto, conceptualmente, se puede restringir el modelo E-R para que solo incluya conjuntos de relaciones binarias. Sin embargo, esta restricción no siempre es deseable.

- Es posible que sea necesario crear un atributo de identificación para que el conjunto de entidades represente el conjunto de relaciones. Este atributo, junto con los conjuntos de relaciones adicionales necesarios, incrementa la complejidad del diseño y (como se verá en la Sección 7.6) los requisitos globales de almacenamiento.
- Un conjunto de relaciones $n$-arias muestra más claramente que varias entidades participan en una sola relación.
- Puede que no haya forma de traducir las restricciones a la relación ternaria en restricciones a las relaciones binarias. Por ejemplo, considérese una restricción que dice que $R$ es del tipo varios a uno de $A$, $B$ a $C$; es decir, cada par de entidades de $A$ y de $B$ se asocia, a lo sumo, con una entidad de $C$. Esta restricción no se puede expresar mediante restricciones de cardinalidad sobre los conjuntos de relaciones $R_A$, $R_B$, y $R_C$.

Considere el conjunto de relaciones *proy_direc* de la Sección 7.2.2, que relaciona *profesor*, *estudiante* y *proyecto*. No se puede dividir directamente *proy_direc* en relaciones binarias entre *profesor* y *proyecto* y entre *profesor* y *estudiante*. Si se hiciese, se podría registrar que el profesor Katz trabaja en los proyectos A y B con los estudiantes Shankar y Zhang; sin embargo, no se podría registrar que Katz trabaja en el proyecto A con el estudiante Shankar y trabaja en el proyecto B con el estudiante Zhang, pero no trabaja en el proyecto A con Zhang ni en el proyecto B con Shankar.

El conjunto de relaciones *proy_direc* se puede dividir en relaciones binarias mediante la creación de nuevos conjuntos de entidades como se ha descrito anteriormente. Sin embargo, no sería muy natural.

### 7.7.4. Ubicación de los atributos de las relaciones

La razón de cardinalidad de una relación puede afectar a la ubicación de sus atributos de las relaciones. Por tanto, los atributos de los conjuntos de relaciones uno a uno o uno a varios pueden estar asociados con uno de los conjuntos de entidades participantes, en lugar de con el conjunto de relaciones. Por ejemplo, especifiquemos que *tutor* es un conjunto de relaciones uno a varios tal que cada profesor puede tutelar a varios estudiantes, pero cada estudiante solo puede tener como tutor a un único profesor. En este caso, el atributo *fecha*, que especifica la fecha en que el profesor se convirtió en tutor del estudiante, podría estar asociado con el conjunto de entidades *estudiante*, como se muestra en la Figura 7.20 (para simplificar la figura solo se muestran algunos de los atributos de los dos conjuntos de entidades). Dado que cada entidad *estudiante* participa en una relación con un ejemplar de *profesor*, como máximo, hacer esta designación de atributos tendría el mismo significado que colocar *fecha* en el conjunto de relaciones *tutor*. Los atributos de un conjunto de relaciones uno a varios solo se pueden recolocar en el conjunto de entidades de la parte «varios» de la relación. Para los conjuntos de entidades uno a uno, los atributos de la relación se pueden asociar con cualquiera de las entidades participantes.

La decisión de diseño sobre la ubicación de los atributos descriptivos en estos casos —como atributo de la relación o de la entidad— debe reflejar las características de la empresa que se modela. El diseñador puede elegir mantener *fecha* como atributo de *tutor* para expresar explícitamente que la fecha se refiere a la relación de tutela y no a otros aspectos del estatus del estudiante en la universidad (por ejemplo, la fecha de aceptación en la universidad).

La elección de la ubicación de atributos es más sencilla para los conjuntos de relaciones varios a varios. Volviendo al ejemplo, espe-

cifiquemos el caso, quizá más realista, de que *tutor* sea un conjunto de relaciones varios a varios que expresa que un profesor puede tutelar a uno o más estudiantes y que un estudiante puede tener como tutor a uno o más profesores. Si hay que expresar la fecha en que un profesor se convirtió en tutor de un determinado estudiante, *fecha* debe ser atributo del conjunto de relaciones *tutor*, en lugar de serlo de cualquiera de las entidades participantes. Si *fecha* fuese un atributo de *estudiante*, por ejemplo, no se podría determinar qué profesor se convirtió en tutor en dicha fecha. Cuando un atributo se determina mediante la combinación de los conjuntos de entidades participantes, en lugar de por cada entidad por separado, ese atributo debe estar asociado con el conjunto de relaciones varios a varios. La Figura 7.3 muestra la ubicación de *fecha* como atributo de la relación; de nuevo, para simplificar la figura solo se muestran algunos de los atributos de los dos conjuntos de entidades.

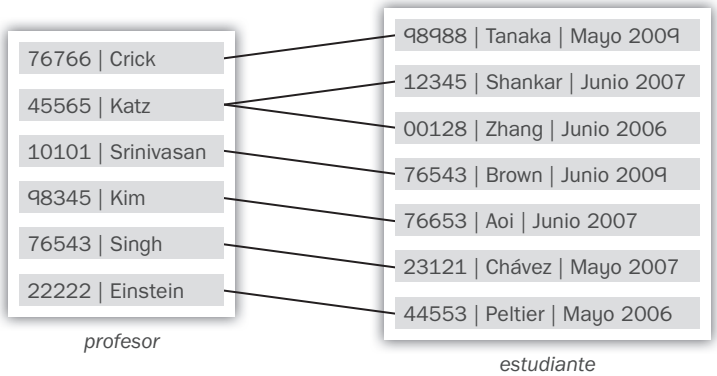



**Figura 7.20.** *fecha* como un atributo del conjunto de entidades *estudiante*.

## 7.8. Características del modelo E-R extendido

Aunque los conceptos básicos del modelo E-R pueden modelar la mayor parte de las características de las bases de datos, algunos aspectos de estas se pueden expresar mejor mediante ciertas extensiones del modelo E-R básico. En este apartado se estudian las características E-R extendidas de especialización, generalización, conjuntos de entidades de nivel superior e inferior, herencia de atributos y agregación.

Para facilitar el contenido se usará un esquema de base de datos algo más elaborado. En particular, se modelarán las distintas personas de la universidad definiendo un conjunto de entidades *persona*, con atributos *ID*, *nombre* y *dirección*.

### 7.8.1. Especialización

Los conjuntos de entidades pueden incluir subgrupos de entidades que se diferencian de alguna forma de las demás entidades del conjunto. Por ejemplo, un subconjunto de entidades de un conjunto de entidades puede tener atributos que no sean compartidos por todas las entidades del conjunto de entidades. El modelo E-R ofrece un medio para representar estos grupos de entidades diferentes.

Como ejemplo, el conjunto de entidades *persona* se puede clasificar como uno de los siguientes:

- *empleado*.
- *estudiante*.

Cada uno de estos tipos de persona se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *persona* más otros posibles atributos adicionales. Por ejemplo, las entidades *empleado* se pueden describir además mediante el atributo *sueldo*, mientras que las entidades *estudiante* se pueden describir además mediante el atributo *tot_créd*. El proceso de establecimiento de subgrupos dentro del conjunto de entidades se denomina **especialización.** La especialización de *persona* permite distinguir entre las personas basándonos en si son empleados o estudiantes: en general, cada persona puede ser empleado, estudiante, las dos cosas o ninguna de ellas.

Como ejemplo adicional, supóngase que la universidad desea dividir a los estudiantes en dos categorías: graduados y pregraduados. Los estudiantes graduados tienen una oficina asignada. Los estudiantes pregraduados tienen asignada una residencia. Cada uno de estos tipos de estudiantes se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *estudiante* más otros atributos adicionales.

La universidad puede crear dos especializaciones de *estudiante*, por ejemplo, *graduado* y *pregraduado*. Como ya se ha visto, las entidades *estudiante* se describen mediante los atributos *ID*, *nombre*, *dirección* y *tot_créd*. El conjunto de entidades *graduado* tendría todos los atributos de *estudiante* y el atributo adicional *número_oficina*. El conjunto de entidades *pregraduado* tendría todos los atributos de *estudiante* y el atributo adicional *residencia*.

La especialización se puede aplicar repetidamente para refinar el esquema de diseño. Por ejemplo, los empleados de la universidad se pueden clasificar como uno de los siguientes:

- *profesor*.
- *secretaria*.

Cada uno de estos tipos de empleado se describe mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *empleado* y otros adicionales. Por ejemplo, las entidades *profesor* se pueden describir, además, por el atributo *rango*; mientras que las entidades *secretaria* se describen por el atributo *horas_semana*. Además, las entidades *secretaria* pueden participar en la relación *secretaria_para*, entre las entidades *secretaria* y *empleado*, que identifica a los empleados a los que ayuda una secretaria.

Un conjunto de entidades se puede especializar en más de una característica distintiva. En este ejemplo, la característica distintiva entre las entidades *empleado* es el trabajo que desempeña cada empleado. Otra especialización coexistente se puede basar en si es un trabajador temporal o fijo, lo que da lugar a los conjuntos de entidades *empleado_temporal* y *empleado_fijo*. Cuando se forma más de una especialización en un conjunto de entidades, cada entidad concreta puede pertenecer a varias especializaciones. Por ejemplo, un empleado dado puede ser un empleado temporal y a su vez secretaria.

En términos de los diagramas E-R, la especialización se representa mediante una punta de flecha hueca desde la entidad especializada a la otra entidad (véase la Figura 7.21). Esta relación se denomina relación ES (ISA – «is a», es un/una) y representa, por ejemplo, que un profesor «es» un/una empleado/a.

La forma en que se dibuja la especialización en un diagrama E-R depende de si una entidad puede pertenecer a varios conjuntos de entidades especializados o si debe pertenecer, como mucho, a un conjunto de entidades especializado. En el primer caso (se permiten varios conjuntos) se llama **especialización solapada,** mientras que en el segundo (se permite uno como mucho) se denomina **especialización disjunta.** Para una especialización solapada (como ocurre en el caso de *estudiante* y *empleado* como especialización de *persona*), se usan dos flechas separadas. Para una especialización disjunta (como en el caso de *profesor* y *secretaria* como especialización de *empleado*), se usa una única flecha. La relación de especialización también se puede llamar relación **superclase-subclase.** Los conjuntos de entidades de mayor y menor nivel son conjuntos de entidades normales, es decir, rectángulos que contienen el nombre del conjunto de entidades.

### 7.8.2. Generalización

El refinamiento a partir del conjunto de entidades inicial en sucesivos niveles de subgrupos de entidades representa un proceso de diseño **descendente (top-down)** en el que las distinciones se hacen explícitas. El proceso de diseño también puede proceder de forma **ascendente (bottom-up),** en la que varios conjuntos de entidades se sintetizan en un conjunto de entidades de nivel superior basado en características comunes. El diseñador de la base de datos puede haber identificado primero:

- El conjunto de entidades *profesor*, con los atributos *profesor_id*, *profesor_nombre*, *profesor_sueldo* y *rango*.
- El conjunto de entidades *secretaria*, con los atributos *secretaria_id*, *secretaria_nombre*, *secretaria_sueldo* y *horas_semana*.

Existen similitudes entre el conjunto de entidades *profesor* y el conjunto de entidades *secretaria* en el sentido de que tienen varios atributos que, conceptualmente, son iguales en los dos conjuntos de entidades: los atributos para el identificador, el nombre y el sueldo. Esta similitud se puede expresar mediante la **generalización,** que es una relación de contención que existe entre el conjunto de entidades de *nivel superior* y uno o varios conjuntos de entidades de *nivel inferior*. En este ejemplo, *empleado* es el conjunto de entidades de nivel superior y *profesor* y *secretaria* son conjuntos de entidades de nivel inferior. En este caso, los atributos que son conceptualmente iguales tienen nombres diferentes en los dos conjuntos de entidades de nivel inferior. Para crear generalizaciones, los atributos deben tener un nombre común y representarse mediante la entidad de nivel superior *persona.* Se pueden usar los nombres de atributos *ID*, *nombre*, *dirección*, como se vio en el ejemplo de la Sección 7.8.1.

Los conjuntos de entidades de nivel superior e inferior también se pueden denominar con los términos **superclase** y **subclase,** respectivamente. El conjunto de entidades *persona* es la superclase de las subclases *estudiante* y *empleado*.

A efectos prácticos, la generalización es una inversión simple de la especialización. Se aplicarán ambos procesos, combinados, en el transcurso del diseño del esquema E-R de una empresa. En términos del propio diagrama E-R no se distingue entre especialización y generalización. Los niveles nuevos de representación de las entidades se distinguen (especialización) o sintetizan (generalización) cuando el esquema de diseño llega a expresar completamente la aplicación de la base de datos y los requisitos del usuario de la base de datos. Las diferencias entre los dos enfoques se pueden caracterizar mediante su punto de partida y su objetivo global.

La especialización parte de un único conjunto de entidades; destaca las diferencias entre las entidades del conjunto mediante la creación de diferentes conjuntos de entidades de nivel inferior. Esos conjuntos de entidades de nivel inferior pueden tener atributos o participar en relaciones que no se aplican a todas las entidades del conjunto de entidades de nivel superior. Realmente, la razón de que el diseñador aplique la especialización es poder representar esas características distintivas. Si *estudiante* y *empleado* tuvieran exactamente los mismos atributos que las entidades *persona* y participaran en las mismas relaciones en las que participan las entidades *persona*, no habría necesidad de especializar el conjunto de entidades *persona*.

La generalización parte del reconocimiento de que varios conjuntos de entidades comparten algunas características comunes (es decir, se describen mediante los mismos atributos y participan en los mismos conjuntos de relaciones). Con base en esas similitudes, la generalización sintetiza esos conjuntos de entidades en un solo conjunto de nivel superior. La generalización se usa para destacar las similitudes entre los conjuntos de entidades de nivel inferior y para ocultar las diferencias; también permite una economía de representación, ya que no se repiten los atributos compartidos.

### 7.8.3. Herencia de los atributos

Una propiedad crucial de las entidades de nivel superior e inferior creadas mediante la especialización y la generalización es la **herencia de los atributos.** Se dice que los atributos de los conjuntos de entidades de nivel superior son **heredados** por los conjuntos de entidades de nivel inferior. Por ejemplo, *estudiante* y *empleado* heredan los atributos de *persona*. Así, *estudiante* se describe mediante sus atributos *ID*, *nombre* y *dirección* y, adicionalmente, por el atributo *tot_créditos*; *empleado* se describe mediante sus atributos *ID*, *nombre* y *dirección* y, adicionalmente, por el atributo *salario*. La herencia de los atributos se aplica en todos los niveles de los conjuntos de entidades de nivel inferior; así, *profesor* y *secretaria*, que son subclases de *empleado*, heredan los atributos *ID*, *nombre* y *dirección* de *persona*, además de heredar *sueldo* de *empleado*.

Los conjuntos de entidades de nivel inferior (o subclases) también heredan la participación en los conjuntos de relaciones en los que participa su entidad de nivel superior (o superclase). Al igual que con la herencia de los atributos, la herencia de participación también se aplica en todos los niveles de los conjuntos de entidades de nivel inferior. Por ejemplo, suponga que el conjunto de entidades *persona* participa en una relación *persona_dept* con *departamento*. Entonces, los conjuntos de entidades *estudiante*, *empleado*, *profesor* y *secretaria*, que son subclases de los conjuntos de entidades *persona*, también participan en la relación *persona_dept* con *departamento*. Los conjuntos de entidades anteriores pueden participar en cualquier relación en la que participe el conjunto de entidades *persona*.

Tanto si se llega a una porción dada del modelo E-R mediante la especialización como si se hace mediante la generalización, el resultado es básicamente el mismo:

- Un conjunto de entidades de nivel superior con los atributos y las relaciones que se aplican a todos sus conjuntos de entidades de nivel inferior.
- Conjuntos de entidades de nivel inferior con características distintivas que solo se aplican en un conjunto dado de entidades de nivel inferior.

En lo que sigue, aunque a menudo solo se haga referencia a la generalización, las propiedades que se estudian corresponden completamente a ambos procesos.

La Figura 7.21 describe una **jerarquía** de conjuntos de entidades. En la figura, *empleado* es un conjunto de entidades de nivel inferior de *persona* y un conjunto de entidades de nivel superior de los conjuntos de entidades *profesor* y *secretaria*. En las jerarquías, un conjunto de entidades dado solo puede estar implicado como conjunto de entidades de nivel inferior en una relación ES; es decir, los conjuntos de entidades de este diagrama solo tienen **herencia única.** Si un conjunto de entidades es un conjunto de entidades de nivel inferior en más de una relación ES, el conjunto de entidades tiene **herencia múltiple** y la estructura resultante se denomina *retículo*.

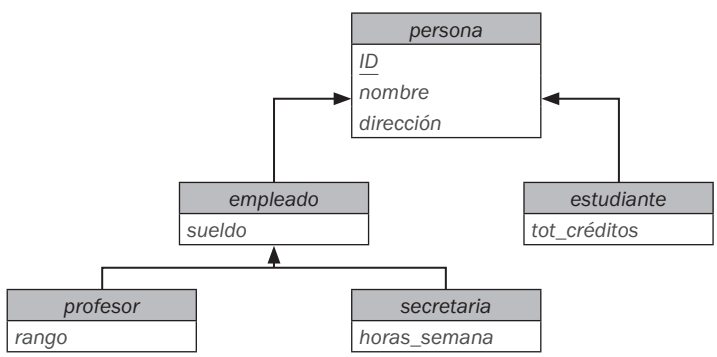



**Figura 7.21.** Especialización y generalización.

### 7.8.4. Restricciones a las generalizaciones

Para modelar una empresa con más precisión, el diseñador de la base de datos puede decidir imponer ciertas restricciones sobre una generalización concreta. Un tipo de restricción implica la determinación de las entidades que pueden formar parte de un conjunto de entidades de nivel inferior dado. Esa pertenencia puede ser una de las siguientes:

- **Definida por la condición.** En los conjuntos de entidades de nivel inferior definidos por la condición, la pertenencia se evalúa en función del cumplimiento de una condición o predicado explícito por la entidad. Por ejemplo, supóngase que el conjunto de entidades de nivel superior *estudiante* tiene el atributo *tipo_estudiante*. Todas las entidades *estudiante* se evalúan según el atributo *tipo_estudiante* que las define. Solo las entidades que satisfacen la condición *tipo_estudiante* = «graduado» pueden pertenecer al conjunto de entidades de nivel inferior *estudiante_graduado*. Todas las entidades que satisfacen la condición *tipo_estudiante* = «pregraduado» se incluyen en *estudiante_pregraduado*. Dado que todas las entidades de nivel inferior se evalúan en función del mismo atributo (en este caso, *tipo_estudiante*), se dice que este tipo de generalización está **definida por el atributo.**
- **Definida por el usuario.** Los conjuntos de entidades de nivel inferior definidos por el usuario no están restringidos por una condición de pertenencia; más bien, el usuario de la base de datos asigna las entidades a un conjunto de entidades dado. Por ejemplo, supóngase que, después de tres meses de trabajo, los empleados de la universidad se asignan a uno de los cuatro grupos de trabajo. En consecuencia, los grupos se representan como cuatro conjuntos de entidades de nivel inferior del conjunto de entidades de nivel superior *empleado*. No se asigna cada empleado a una entidad grupo concreta automáticamente de acuerdo con una condición explícita que lo defina. En vez de eso, la asignación al grupo la lleva a cabo el usuario que toma la decisión persona a persona. La asignación se implementa mediante una operación que añade cada entidad a un conjunto de entidades.

Un segundo tipo de restricciones tiene relación con la pertenencia de las entidades a más de un conjunto de entidades de nivel inferior de la generalización. Los conjuntos de entidades de nivel inferior pueden ser de uno de los tipos siguientes:

- **Disjuntos.** La *restricción sobre la condición de disjunción* exige que cada entidad no pertenezca a más de un conjunto de entidades de nivel inferior. En el ejemplo, la entidad *estudiante* solo puede cumplir una condición del atributo *tipo_estudiante*; una entidad puede ser un estudiante graduado o pregraduado, pero no ambas cosas a la vez.
- **Solapados.** En las *generalizaciones solapadas* la misma entidad puede pertenecer a más de un conjunto de entidades de nivel inferior de la generalización. Como ejemplo, considere el grupo de trabajo de empleados y suponga que algunos empleados participan en más de un grupo de trabajo. Cada empleado, por tanto, puede aparecer en más de uno de los conjuntos de entidades grupo, que son conjuntos de entidades de nivel inferior de *empleado*. Por tanto, la generalización es solapada.

En la Figura 7.21 supondremos que una persona puede ser a la vez un empleado y un estudiante. Se mostrará esta generalización solapada con flechas separadas: una de empleado a persona y otra desde estudiante a persona. Sin embargo, la generalización de un profesor y una secretaria es disjunta. Se mostrará utilizando flechas simples.

Una última restricción, la **restricción de completitud** sobre una generalización o especialización, especifica si una entidad del conjunto de entidades de nivel superior debe pertenecer, al menos, a uno de los conjuntos de entidades de nivel inferior de la generalización o especialización. Esta restricción puede ser de uno de los tipos siguientes:

- **Generalización** o **especialización total.** Cada entidad de nivel superior debe pertenecer a un conjunto de entidades de nivel inferior.
- **Generalización** o **especialización parcial.** Puede que alguna entidad de nivel superior no pertenezca a ningún conjunto de entidades de nivel inferior.

La generalización parcial es la predeterminada. Se puede especificar la generalización total en los diagramas E-R añadiendo la palabra «total» en el diagrama y dibujando una línea discontinua desde la palabra a la correspondiente cabeza de flecha hueca en la que aplica (para una generalización total), o al conjunto de cabezas de flecha huecas a las que aplica (para una generalización solapada).

La generalización de *estudiante* es total: todas las entidades *estudiante* deben ser graduados o pregraduados. Como el conjunto de entidades de nivel superior al que se llega mediante la generalización suele estar compuesto únicamente de entidades de los conjuntos de entidades de nivel inferior, la restricción de completitud para los conjuntos de entidades de nivel superior generalizados suele ser total. Cuando la generalización es parcial, las entidades de nivel superior no están limitadas a aparecer en los conjuntos de entidades de nivel inferior. Los conjuntos de entidades grupo de trabajo ilustran una especialización parcial. Como los empleados solo se asignan a cada grupo después de llevar tres meses en el trabajo, puede que algunas entidades *empleado* no pertenezcan a ninguno de los conjuntos de entidades grupo de nivel inferior.

Los conjuntos de entidades equipo se pueden caracterizar mejor como especialización de *empleado* parcial y solapada. La generalización de *estudiante_graduado* y *estudiante_pregraduado* en *estudiante* es una generalización total y disjunta. Las restricciones de completitud y sobre la condición de disjunción, sin embargo, no dependen una de la otra. Las características de las restricciones también pueden ser parcial — disjunta y total— y solapada.

Es evidente que algunos requisitos de inserción y de borrado son consecuencia de las restricciones que se aplican a una generalización o especialización dada. Por ejemplo, cuando se impone una restricción de completitud total, las entidades insertadas en un conjunto de entidades de nivel superior se deben insertar, al menos, en uno de los conjuntos de entidades de nivel inferior. Con una restricción de definición por condición, todas las entidades de nivel superior que cumplen la condición se deben insertar en ese conjunto de entidades de nivel inferior. Finalmente, las entidades que se borren de los conjuntos de entidades de nivel superior se deben borrar también de todos los conjuntos de entidades de nivel inferior asociados a los que pertenezcan.

### 7.8.5. Agregación

Una limitación del modelo E-R es que no es posible expresar relaciones entre las relaciones. Para ilustrar la necesidad de estos constructores, considérese la relación ternaria *proy_direc*, que ya se ha visto anteriormente, entre *profesor*, *estudiante* y *proyecto* (véase la Figura 7.13).

Suponga ahora que se desea que cada profesor que dirige a un estudiante en un proyecto realice un informe de evaluación mensual. Se modela el informe de evaluación como una entidad *evaluación*, con una clave primaria *evaluación_id*. Una alternativa para registrar la combinación (*estudiante*, *proyecto*, *profesor*) a la que corresponde una evaluación es crear un conjunto de relaciones cuaternaria *eval_para* entre *estudiante*, *proyecto*, *profesor* y *evaluación*. Se requiere una relación cuaternaria, una relación binaria entre *estudiante* y *evaluación*, por ejemplo, no permitiría repre-

sentar la combinación (*proyecto*, *profesor*) a la que corresponde la *evaluación*. Utilizando los constructores del modelo E-R básico se obtiene el diagrama de la Figura 7.22 (para simplificar se han omitido los atributos de los conjuntos de entidades).

Parece que los conjuntos de relaciones *proy_direc* y *eval_para* se pueden combinar en un solo conjunto de relaciones. No obstante, no se deben combinar en una sola relación, ya que puede que algunas combinaciones *estudiante*, *proyecto*, *profesor* no tengan una *evaluación* asociada.

No obstante, hay información redundante en la figura obtenida, ya que cada combinación *estudiante*, *proyecto*, *profesor* de *eval_para* también tiene que estar en *proy_direc*. Si *evaluación* fuese un valor en lugar de una entidad, se podría hacer que *evaluación* fuese un atributo multivalorado de la relación *proy_direc*. Sin embargo, esta alternativa no puede ser una opción si una *evaluación* puede estar relacionada con otras entidades; por ejemplo, los informes de evaluación pueden estar asociados con una *secretaria* que es la responsable de procesar el informe de evaluación para realizar los pagos escolares.

La mejor forma de modelar una situación como la descrita es usar la agregación. La **agregación** es una abstracción a través de la cual las relaciones se tratan como entidades de nivel superior. Así, para este ejemplo, se considera el conjunto de relaciones *proy_direc* (que relaciona los conjuntos de entidades *estudiante*, *proyecto*, *profesor*) como el conjunto de entidades de nivel superior denominado *proy_direc*. Ese conjunto de entidades se trata de la misma forma que cualquier otro conjunto de entidades. Se puede crear entonces la relación binaria *eval_para* entre *proy_direc* y *evaluación* para representar para qué combinación (*estudiante, proyecto, profesor*) es la evaluación. La Figura 7.23 muestra una notación para la agregación que se usa habitualmente para representar esta situación.

## 7.8.6. Reducción a esquemas relacionales

Ya estamos en disposición de describir cómo las características del modelo E-R extendido se pueden trasladar a esquemas relacionales.

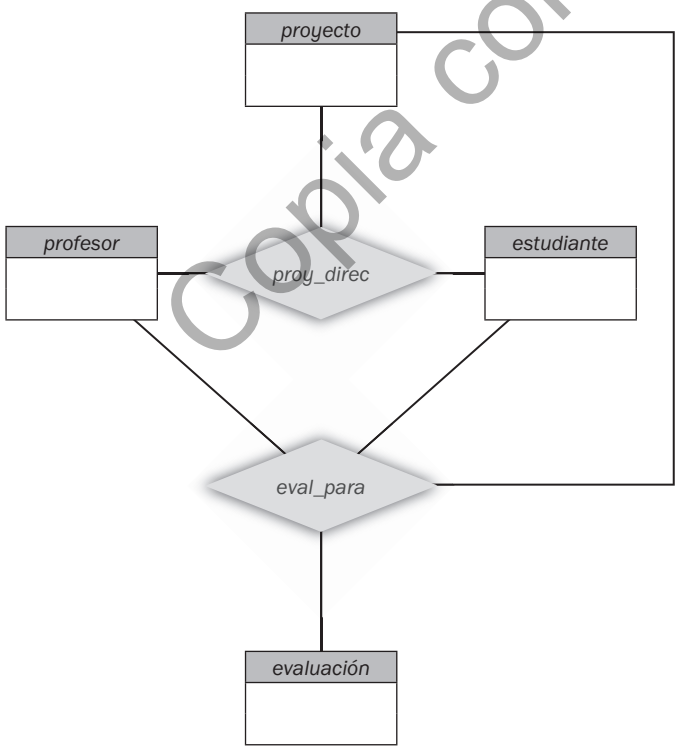



**Figura 7.22.** Diagrama E-R con relaciones redundantes.

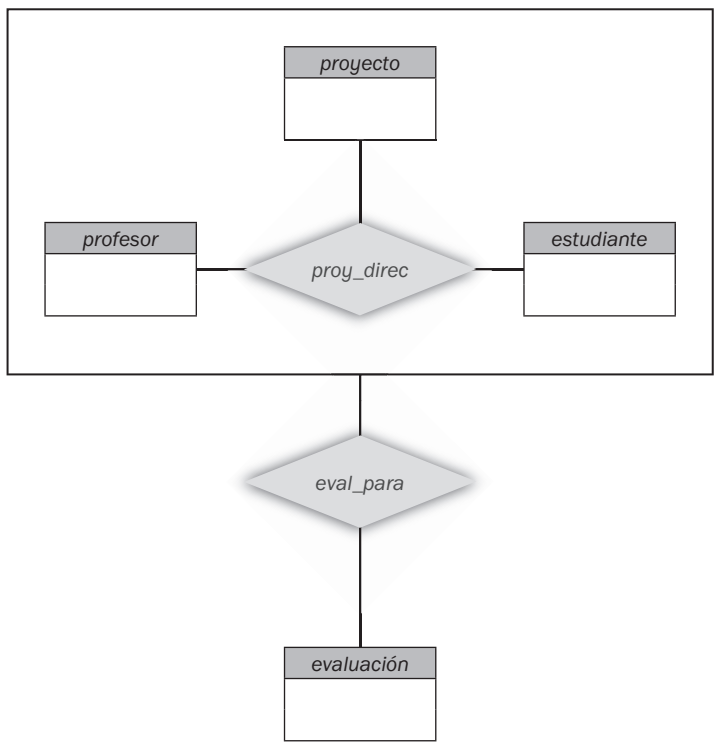



**Figura 7.23.** Diagrama E-R con agregación.

### 7.8.6.1. Representación de la generalización

Existen dos métodos diferentes para designar los esquemas de relación de los diagramas E-R que incluyen generalización. Aunque en esta discusión se hace referencia a la generalización de la Figura 7.21, se simplifica incluyendo solo la primera capa de los conjuntos de entidades de nivel inferior, es decir, *empleado* y *estudiante*. Se da por supuesto que *ID* es la clave primaria de *persona*.

1. Se crea un esquema para el conjunto de entidades de nivel superior. Para cada conjunto de entidades de nivel inferior se crea un esquema que incluye un atributo para cada uno de los atributos de ese conjunto de entidades más un atributo por cada atributo de la clave primaria del conjunto de entidades de nivel superior. Así, para el diagrama E-R de la Figura 7.21, se tienen tres esquemas:

   *persona (ID, nombre, calle, ciudad)*

   *empleado (ID, sueldo)*

   *estudiante (ID, tot_créd)*

   Los atributos de clave primaria del conjunto de entidades de nivel superior pasan a ser atributos de clave primaria del conjunto de entidades de nivel superior y de todos los conjuntos de entidades de nivel inferior. En el ejemplo anterior se pueden ver subrayados.

   Además, se crean restricciones de clave externa para los conjuntos de entidades de nivel inferior, con sus atributos de clave primaria que hacen referencia a la clave primaria de la relación creada a partir del conjunto de entidades de nivel superior. En el ejemplo anterior, el atributo *ID* de *empleado* haría referencia a la clave primaria de *persona*, y similar para *estudiante*.
2. Es posible una representación alternativa si la generalización es disjunta y completa; es decir, si no hay ninguna entidad miembro de dos conjuntos de entidades de nivel inferior directamente por debajo de un conjunto de entidades de nivel superior, y si todas las entidades del conjunto de entidades de nivel superior también pertenecen a uno de los conjuntos de entidades de nivel inferior. En este caso no se crea un esquema para el conjunto de

entidades de nivel superior. En vez de eso, para cada conjunto de entidades de nivel inferior se crea un esquema que incluye un atributo por cada atributo de ese conjunto de entidades más un atributo por *cada* atributo del conjunto de entidades de nivel superior. Entonces, para el diagrama E-R de la Figura 7.21, se tienen dos esquemas:

*empleado (ID, nombre, calle, ciudad, sueldo)*

*estudiante (ID, nombre, calle, ciudad, tot_cred)*

Estos dos esquemas tienen *ID*, que es el atributo de clave primaria del conjunto de entidades de nivel superior *persona*, como clave primaria.

Un inconveniente del segundo método es la definición de las restricciones de clave externa. Para ilustrar el problema, supóngase que se tiene un conjunto de relaciones *R* que implica al conjunto de entidades *persona*. Con el primer método, al crear un esquema de relación *R* a partir del conjunto de relaciones, también se define una restricción de clave externa para *R*, que hace referencia al esquema *persona*. Desafortunadamente, con el segundo método no se tiene una única relación a la que pueda hacer referencia la restricción de clave externa de *R*. Para evitar este problema hay que crear un esquema de relación *persona* que contenga, al menos, los atributos de clave primaria de la entidad *persona*.

Si se usara el segundo método para una generalización solapada, algunos valores se almacenarían varias veces de manera innecesaria. Por ejemplo, si una persona es a la vez empleado y estudiante, los valores de *calle* y de *ciudad* se almacenarían dos veces.

Si la generalización fuera disjunta pero no completa; es decir, si alguna persona no fuera ni empleado ni estudiante, entonces haría falta un esquema adicional

*persona (ID, nombre, calle, ciudad)*

para representar a esas personas. Sin embargo, aún permanecería el problema con las restricciones de clave externa indicados anteriormente. En un intento de soslayar el problema, suponga que a los empleados y a los estudiantes se les representa adicionalmente en la relación *persona*. Desafortunadamente, la información de nombre, calle y ciudad no debería guardarse de forma redundante en la relación *persona* y en la relación *estudiante* para los estudiantes, y de forma similar en la relación *persona* y en la relación *empleado* para los empleados. Esto sugiere guardar la información de nombre, calle y ciudad solo en la relación *persona* y eliminar esta información de *estudiante* y de *empleado*. Si lo hacemos, el resultado es exactamente el obtenido con el primer método presentado.

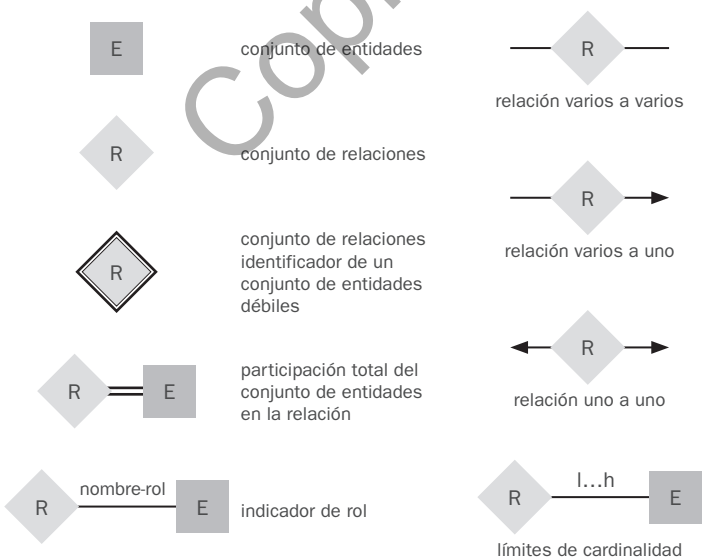



**Figura 7.24.** Símbolos usados en la notación E-R.

#### 7.8.6.2. Representación de la agregación

El diseño de esquemas para los diagramas E-R que incluyen agregación es sencillo. Considérese el diagrama de la Figura 7.23. El esquema del conjunto de relaciones *eval_para* entre la agregación de *proy_direc* y el conjunto de entidades *evaluación* incluye un atributo para cada atributo de las claves primarias del conjunto de entidades *evaluación* y del conjunto de relaciones *proy_direc*. También incluye un atributo para los atributos descriptivos, si los hay, del conjunto de relaciones *eval_para*. Entonces, se transforman los conjuntos de relaciones y de entidades de la entidad agregada siguiendo las reglas que se han definido anteriormente.

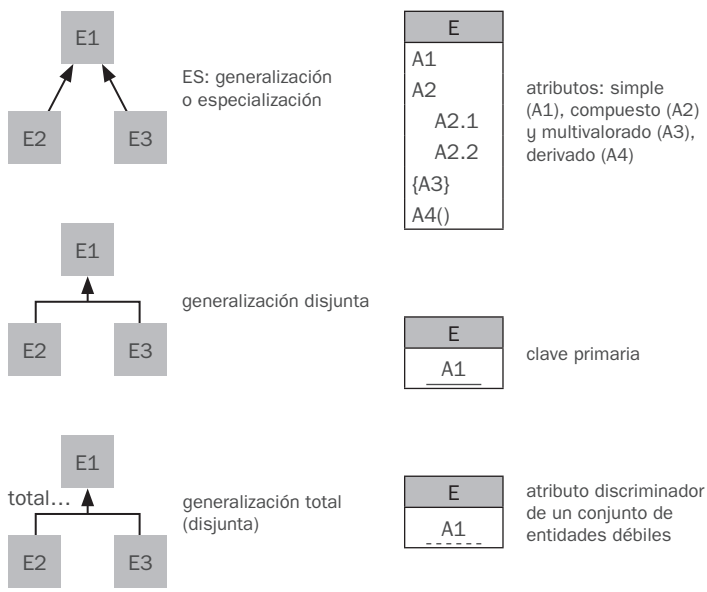
Las reglas que se han visto anteriormente para la creación de restricciones de clave primaria y de clave externa para los conjuntos de relaciones, se pueden aplicar también a los conjuntos de relaciones que incluyen agregación, tratando la agregación como cualquier otra entidad. La clave primaria de la agregación es la clave primaria del conjunto de relaciones que la define. No hace falta ninguna relación más para que represente la agregación; en vez de eso, se usa la relación creada a partir de la relación definidora.

## 7.9. Notaciones alternativas para el modelado de datos

Una representación con un diagrama del modelo de datos de una aplicación es una parte importante del diseño del esquema de la base de datos. La creación de un esquema de la base de datos requiere no solo expertos en el modelado de datos, sino también expertos en el dominio que conozcan los requisitos de la aplicación pero que puede que no estén familiarizados con el modelado de datos. Que una representación gráfica sea intuitiva es importante, ya que facilita la comunicación de información entre estos dos grupos de expertos.

Se han propuesto distintas notaciones alternativas para el modelado de datos, siendo los diagramas E-R y los diagramas de clases de UML los más utilizados. No existe un estándar universal para la notación de los diagramas E-R y en cada libro se utilizan notaciones diferentes. Se ha elegido una notación en particular para esta sexta edición que difiere de la notación empleada en ediciones anteriores, por razones que se explicarán más adelante en esta sección.

En el resto de la sección se estudiarán algunas de las notaciones gráficas E-R alternativas, así como la notación de los diagramas de clases de UML. Para ayudar en la comparación de nuestra notación con las alternativas, la Figura 7.24 resume el conjunto de símbolos que se ha usado en los diagramas E-R.



### 7.9.1. Notaciones E-R alternativas

La Figura 7.25 indica algunas de las notaciones de diagramas E-R alternativas que se usan habitualmente. Una representación alternativa de los atributos o entidades es mostrarlos en óvalos conectados al rectángulo que representa la entidad; los atributos de claves primarias se indican subrayándolos. La notación anterior se muestra en la parte superior de la figura. Los atributos de las relaciones se pueden representar de forma parecida, conectando los óvalos al rombo que representa la relación.

Las restricciones de cardinalidad en las relaciones se pueden indicar de varias formas diferentes, como se muestra en la Figura 7.25. En una alternativa que se muestra en la parte izquierda de la figura, las etiquetas $*$ y 1 en los segmentos que salen de las relaciones se usan a menudo para denotar relaciones varios a varios, uno a uno y varios a uno. El caso de uno a varios es simétrico con el de varios a uno, y no se muestra.

En otra notación alternativa que se muestra en la parte derecha de la figura, los conjuntos de relaciones se representan mediante líneas entre los conjuntos de entidades, sin rombos; por tanto, solo se podrán modelar relaciones binarias. Las restricciones de cardinalidad en esta notación se muestran mediante la notación «pata de gallo», como en la figura. En una relación *R* entre *E1* y *E2* la «pata de gallo» en ambos lados indica una relación varios a varios, mientras que una «pata de gallo» solo en el lado de *E1* indica una relación varios a uno desde *E1* a *E2*. La participación total se especifica en esta notación mediante una barra vertical. Fíjese, sin embargo, que en una relación *R* entre entidades *E1* y *E2* si la participación de *E1* en *R* es total, la barra vertical se sitúa en el lado opuesto, adyacente a la entidad *E2*. De forma similar, la participación parcial se indica usando un círculo, de nuevo en el lado opuesto.

En la parte inferior de la Figura 7.25 se muestra una representación alternativa de la generalización, usando triángulos en lugar de cabezas de flecha huecas.

En anteriores ediciones de este libro, hasta la quinta edición, se han usado óvalos para representar los atributos, con triángulos para representar la generalización, como se muestra en la Figura 7.25. La notación utilizando óvalos para los atributos y rombos para las relaciones es similar a la forma original de los diagramas E-R usados por Chen en su artículo que introdujo la notación del modelado E-R. Esta notación se conoce actualmente como notación de Chen.

El Instituto Nacional de Estados Unidos para Normalización y Tecnología ha definido en 1993 un estándar llamado IDEFIX. IDEFIX usa la notación «pata de gallo» con barras verticales al lado de las relaciones para denotar la participación total y círculos huecos para denotar la participación parcial, e incluye otras notaciones que no se han incluido.

Con el crecimiento del uso del lenguaje de marcado unificado (UML: *Unified Markup Language*), que se describe en la Sección 7.9.2, se ha decidido actualizar la notación E-R para acercarla al formato de los diagramas de clases de UML; las conexiones se verán más claras en la Sección 7.9.2. En comparación con nuestra notación previa, la nueva notación proporciona una representación más compacta de los atributos y está más cercana a las notaciones que admiten muchas de las herramientas de modelado E-R, además de ser más cercana a la notación de los diagramas de clases de UML.

Existen varias herramientas para construir diagramas E-R, cada una de ellas con sus propias variantes de notación. Algunas de las herramientas proporcionan la posibilidad de elegir entre varias notaciones E-R. Para más información consulte las referencias en las notas bibliográficas.

Una de las diferencias clave entre los conjuntos de entidades en un diagrama E-R y los esquemas de relación creados desde dichas entidades es que los atributos en el esquema relacional que se corresponden a relaciones E-R, como el atributo *nombre_dept* de *profesor*, no se muestran en el conjunto de entidades del diagrama E-R. Algunas herramientas de modelado de datos permiten que los usuarios elijan entre vistas de la misma entidad, una vista sin dichos atributos y otra vista relacional con dichos atributos.

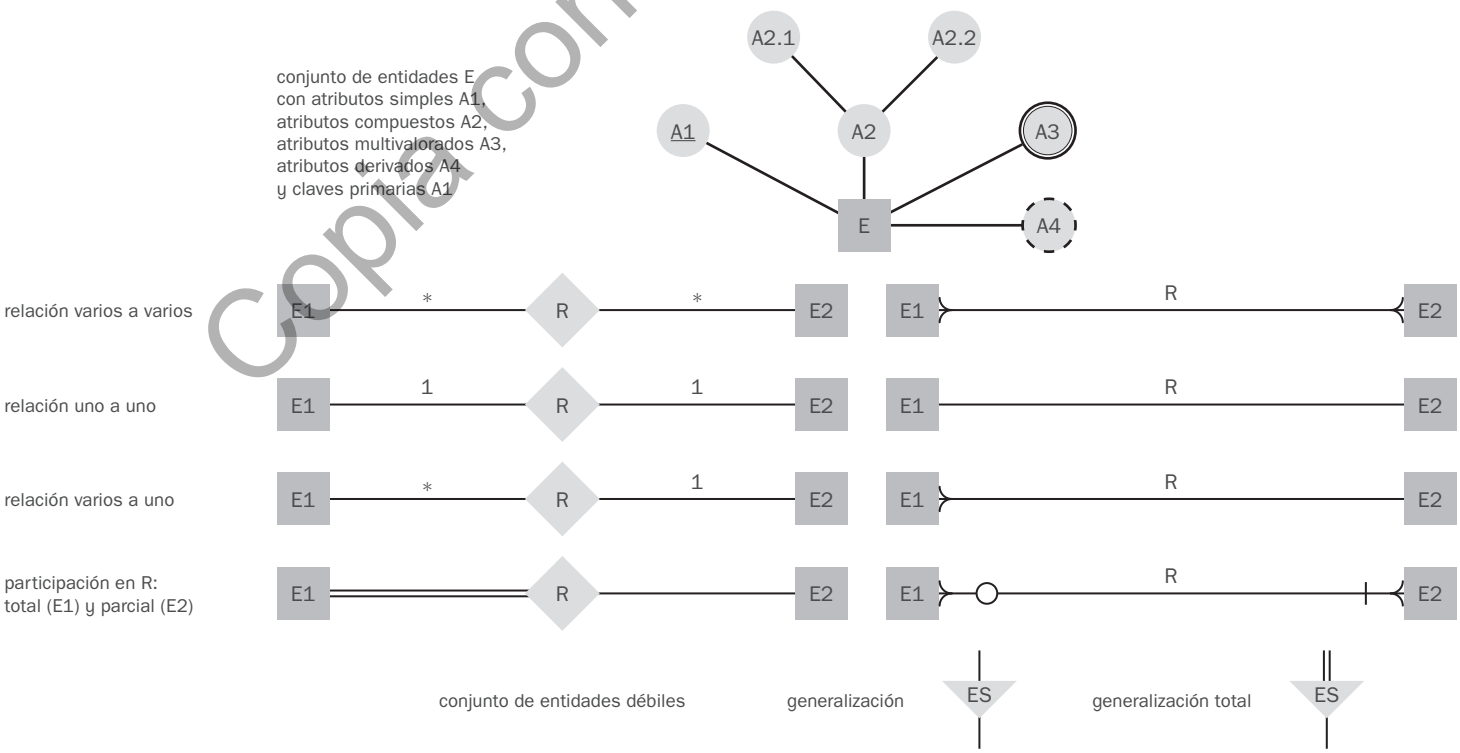



**Figura 7.25.** Notaciones E-R alternativas.

### 7.9.2. El lenguaje de modelado unificado UML

Los diagramas entidad-relación ayudan a modelar el componente de representación de datos de los sistemas de software. La representación de datos, sin embargo, solo forma parte del diseño global del sistema. Otros componentes son los modelos de interacción del usuario con el sistema, la especificación de los módulos funcionales del sistema y su interacción, etc. El **lenguaje de modelado unificado** (Unified Modeling Language, UML) es una norma desarrollada bajo los auspicios del Grupo de Administración de Objetos (Object Management Group, OMG) para la creación de especificaciones de diferentes componentes de los sistemas de software. Algunas de las partes del UML son:

- **Diagramas de clase.** Los diagramas de clase son parecidos a los diagramas E-R. Más adelante en esta sección se mostrarán algunas características de los diagramas de clase y del modo en que se relacionan con los diagramas E-R.
- **Diagramas de caso de uso.** Los diagramas de caso de uso muestran la interacción entre los usuarios y el sistema, en especial los pasos de las tareas que llevan a cabo los usuarios (como retirar dinero o matricularse en una asignatura).
- **Diagramas de actividad.** Los diagramas de actividad describen el flujo de tareas entre los diferentes componentes del sistema.
- **Diagramas de implementación.** Los diagramas de implementación muestran los componentes del sistema y sus interconexiones, tanto en el nivel de los componentes de software como en el de hardware.

Aquí no se pretende ofrecer un tratamiento detallado de las diferentes partes del UML. Véanse las notas bibliográficas para encontrar referencias sobre el UML. En vez de eso, se mostrarán algunas características de la parte del UML que se relaciona con el modelado de datos mediante ejemplos.

La Figura 7.26 muestra varios constructores de diagramas E-R y sus constructores equivalentes de diagramas de clases de UML. Más adelante se describen estos constructores. UML modela objetos, mientras que E-R modela entidades. Los objetos son como entidades y tienen atributos, pero también proporcionan un conjunto de funciones (denominadas métodos) que se pueden invocar para calcular valores con base en los atributos de los objetos, o para actualizar el propio objeto. Los diagramas de clases pueden describir métodos, además de atributos. Los objetos se tratan en el Capítulo 22. UML no admite atributos compuestos ni multivalorados, y los atributos derivados son equivalentes a los métodos sin parámetros. Como las clases admiten encapsulación, UML permite que los atributos y los métodos tengan el prefijo «+», «−» o «#», que significan acceso público, privado o protegido, respectivamente. Los atributos privados solo los pueden usar los métodos de la clase, mientras que los atributos protegidos solo los pueden usar los métodos de la clase y sus subclases, lo que debería resultar familiar a cualquiera que conozca Java, C++ o C#.

En terminología de UML, los conjuntos de relaciones se conocen como **asociaciones;** nos referiremos a ellas como conjuntos de relaciones por consistencia con la terminología E-R. Los conjuntos de relaciones binarias se representan en UML dibujando simplemente una línea que conecte los conjuntos de entidades. El nombre del conjunto de relaciones se escribe junto a la línea. También se puede especificar el rol que desempeña cada conjunto de entidades en un conjunto de relaciones escribiendo el nombre del rol sobre la línea, junto al conjunto de entidades. De manera alternativa, se puede escribir el nombre del conjunto de relaciones en un recuadro, junto con los atributos del conjunto de relaciones, y conectar el recuadro con una línea discontinua a la línea que describe el conjunto de relaciones. Este recuadro se puede tratar entonces como un conjunto de entidades, de la misma forma que la agregación en los diagramas E-R, y puede participar en relaciones con otros conjuntos de entidades.

Desde la versión 1.3 de UML, UML soporta las relaciones no binarias usando la misma notación de rombos utilizada en los diagramas E-R. En versiones anteriores de UML las relaciones no binarias no se podían representar directamente, había que convertirlas en relaciones binarias usando la técnica que se vio en la Sección 7.7.3. UML soporta que se use la notación rombo incluso para relaciones binarias, pero la mayoría de los diseñadores utilizan líneas.

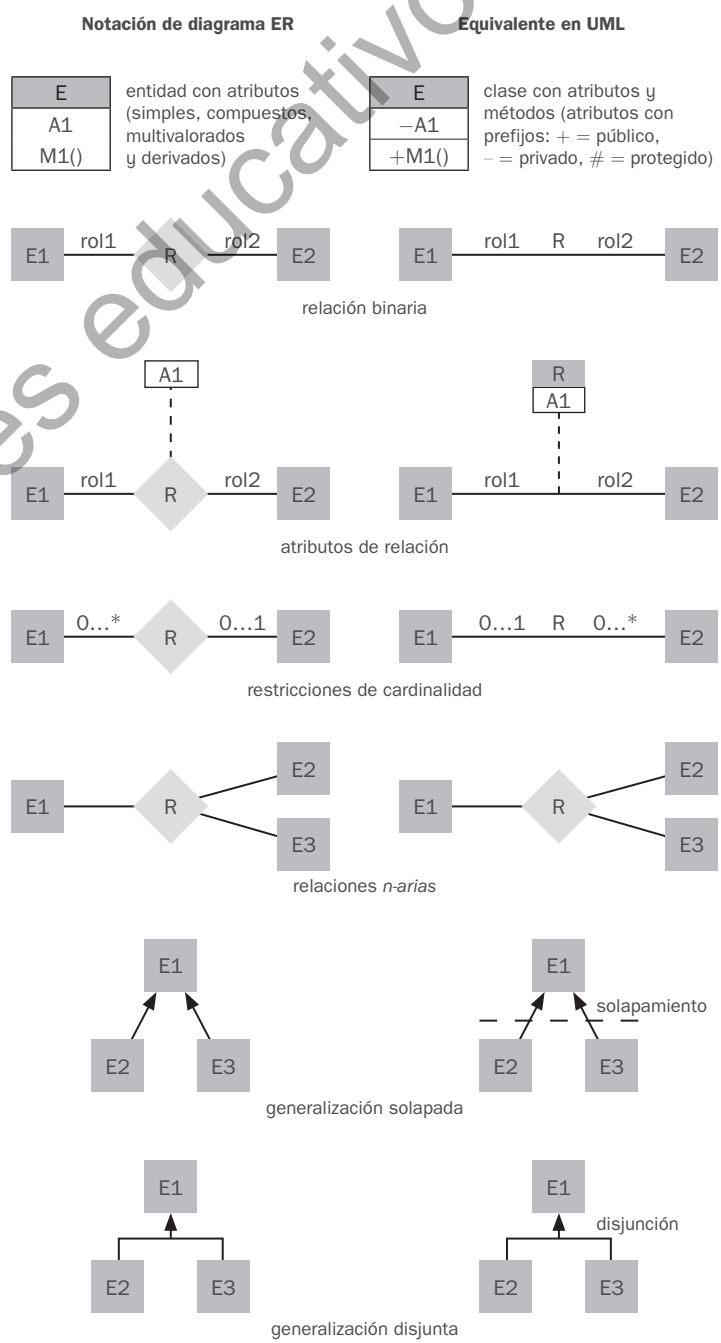



**Figura 7.26.** Símbolos usados en la notación de diagramas de clase UML.

Las restricciones de cardinalidad se especifican en UML de la misma forma que en los diagramas E-R, de la forma $l…h$, donde $l$ denota el número mínimo y $h$ el máximo de relaciones en que puede participar cada entidad. Sin embargo, hay que ser consciente de que la ubicación de las restricciones es exactamente la contraria que en los diagramas E-R, como se muestra en la Figura 7.26. La restricción $0…*$ en el lado $E2$ y $0…1$ en el lado $E1$ significa que cada entidad $E2$ puede participar, a lo sumo, en una relación, mientras que cada entidad $E1$ puede participar en varias relaciones; en otras palabras, la relación es varios a uno de $E2$ a $E1$.

Los valores aislados, como 1 o $*$, se pueden escribir en los arcos; el valor 1 sobre un arco se trata como equivalente de $1…1$, mientras que $*$ es equivalente a $0…*$. UML soporta la generalización; la notación es básicamente la misma que en la notación E-R, incluyendo la representación de generalizaciones disjuntas y solapadas.

Los diagramas de clases de UML incluyen otras notaciones que no se corresponden con las notaciones E-R vistas hasta ahora. Por ejemplo, una línea entre dos conjuntos de entidades con un rombo en un extremo especifica que la entidad en el extremo del rombo contiene a la otra (la inclusión se denomina «agregación» en la terminología de UML; no confundir este uso de agregación con el sentido en que se usa en el modelo E-R). Por ejemplo, una entidad vehículo puede contener una entidad motor.

Los diagramas de clases de UML también ofrecen notaciones para representar características de los lenguajes orientados a objetos, como las interfaces. Véanse las referencias en las notas bibliográficas para obtener más información sobre los diagramas de clases de UML.

## 7.10. Otros aspectos del diseño de bases de datos

La explicación sobre el diseño de esquemas dada en este capítulo puede crear la falsa impresión de que el diseño de esquemas es el único componente del diseño de bases de datos. En realidad, hay otras consideraciones que se tratarán con más profundidad en capítulos posteriores y que se describirán brevemente a continuación.

### 7.10.1. Restricciones de datos y diseño de bases de datos relacionales

Se ha visto gran variedad de restricciones de datos que pueden expresarse mediante SQL, como las restricciones de clave primaria, las de clave externa, las restricciones **check,** los asertos y los disparadores. Las restricciones tienen varios propósitos. El más evidente es la automatización de la conservación de la consistencia. Al expresar las restricciones en el lenguaje de definición de datos de SQL, el diseñador puede garantizar que el propio sistema de bases de datos haga que se cumplan las restricciones. Esto es más digno de confianza que dejar que cada programa haga cumplir las restricciones por su cuenta. También ofrece una ubicación central para la actualización de las restricciones y la adición de otras nuevas.

Otra ventaja de definir explícitamente las restricciones es que algunas resultan especialmente útiles en el diseño de esquemas de bases de datos relacionales. Si se sabe, por ejemplo, que el número de DNI identifica de manera unívoca a cada persona, se puede usar el número de DNI de una persona para vincular los datos relacionados con esa persona aunque aparezcan en varias relaciones. Compare esta posibilidad, por ejemplo, con el color de ojos, que no es un identificador unívoco. El color de ojos no se puede usar para vincular los datos correspondientes a una persona concreta en varias relaciones, ya que los datos de esa persona no se podrían distinguir de los datos de otras personas con el mismo color de ojos.

En la Sección 7.6 se generó un conjunto de esquemas de relación para un diseño E-R dado mediante las restricciones especificadas en el diseño. En el Capítulo 8 se formaliza esta idea y otras relacionadas y se muestra la manera en que puede ayudar al diseño de esquemas de bases de datos relacionales. El enfoque formal del diseño de bases de datos relacionales permite definir de manera precisa la bondad de cada diseño y mejorar los diseños malos. Se verá que el proceso de comenzar con un diseño entidad-relación y generar mediante algoritmos los esquemas de relación a partir de ese diseño es una buena manera de comenzar el proceso de diseño.

Las restricciones de datos también resultan útiles para determinar la estructura física de los datos. Puede resultar útil almacenar físicamente próximos en el disco los datos que están estrechamente relacionados entre sí, de modo que se mejore la eficiencia del acceso al disco. Algunas estructuras de índices funcionan mejor cuando el índice se crea sobre una clave primaria.

La aplicación de las restricciones se lleva a cabo a un precio potencialmente alto en rendimiento cada vez que se actualiza la base de datos. En cada actualización el sistema debe comprobar todas las restricciones y rechazar las actualizaciones que no las cumplen o ejecutar los disparadores correspondientes. La importancia de la penalización en rendimiento no solo depende de la frecuencia de actualización, sino también del modo en que se haya diseñado la base de datos. En realidad, la eficiencia de la comprobación de determinados tipos de restricciones es un aspecto importante de la discusión del diseño de esquemas para bases de datos relacionales que se verá en el Capítulo 8.

### 7.10.2. Requisitos de uso: consultas y rendimiento

El rendimiento de los sistemas de bases de datos es un aspecto crítico de la mayor parte de los sistemas informáticos empresariales. El rendimiento no solo tiene que ver con el uso eficiente del hardware de cálculo y de almacenamiento que se usa, sino también con la eficiencia de las personas que interactúan con el sistema y de los procesos que dependen de los datos de las bases de datos.

Existen dos métricas principales para el rendimiento:

- **Productividad:** el número de consultas o actualizaciones (a menudo denominadas *transacciones*) que pueden procesarse en promedio por unidad de tiempo.
- **Tiempo de respuesta:** el tiempo que tarda *una sola* transacción desde el comienzo hasta el final en promedio o en el peor de los casos.

Los sistemas que procesan gran número de transacciones agrupadas por lotes se centran en tener una productividad elevada. Los sistemas que interactúan con personas y los sistemas de tiempo crítico suelen centrarse en el tiempo de respuesta. Estas dos métricas no son equivalentes. La productividad elevada se consigue mediante un elevado uso de los componentes del sistema. Ello puede dar lugar a que algunas transacciones se pospongan hasta el momento en que puedan ejecutarse con mayor eficiencia. Las transacciones pospuestas sufren un bajo tiempo de respuesta.

Históricamente, la mayor parte de los sistemas de bases de datos comerciales se han centrado en la productividad; no obstante, gran variedad de aplicaciones, incluidas las aplicaciones basadas en la web y los sistemas informáticos para telecomunicaciones necesitan un buen tiempo de respuesta promedio y una cota razonable para el peor tiempo de respuesta que pueden ofrecer.

La comprensión de los tipos de consultas que se espera que sean más frecuentes ayuda al proceso de diseño. Las consultas que implican reuniones necesitan evaluar más recursos que las que no las implican. A veces, cuando se necesita una reunión, puede que el administrador de la base de datos decida crear un índice que facilite la evaluación de la reunión. Para las consultas —tanto si está implicada una reunión como si no— se pueden crear índices para acelerar la evaluación de los predicados de selección (la cláusula **where** de SQL) que sea posible que aparezcan.

Otro aspecto de las consultas que afecta a la elección de índices es la proporción relativa de operaciones de actualización y de lectura. Aunque los índices pueden acelerar las consultas, también ralentizan las actualizaciones, que se ven obligadas a realizar un trabajo adicional para mantener la exactitud de los índices.

### 7.10.3. Requisitos de autorización

Las restricciones de autorización también afectan al diseño de las bases de datos, ya que SQL permite que se autorice el acceso a los usuarios en función de los componentes del diseño lógico de la base de datos. Puede que haga falta descomponer un esquema de relación en dos o más esquemas para facilitar la concesión de derechos de acceso en SQL. Por ejemplo, un registro de empleados puede contener datos relativos a nóminas, funciones de los puestos y prestaciones sanitarias. Como diferentes unidades administrativas de la empresa pueden manejar cada uno de los diversos tipos de datos, algunos usuarios necesitarán acceso a los datos de las nóminas, mientras se les deniega el acceso a los datos de las funciones de los puestos de trabajo, a los de las prestaciones sanitarias, etc. Si todos esos datos se hallan en una relación, la deseada división del acceso, aunque todavía posible mediante el uso de vistas, resulta más complicada. La división de los datos, de este modo, pasa a ser todavía más crítica cuando estos se distribuyen en varios sistemas de una red informática, un aspecto que se considera en el Capítulo 19.

### 7.10.4. Flujos de datos y flujos de trabajo

Las aplicaciones de bases de datos suelen formar parte de una aplicación empresarial de mayor tamaño que no solo interactúa con el sistema de bases de datos, sino también con diferentes aplicaciones especializadas. Por ejemplo, en una compañía manufacturera, puede que un sistema de diseño asistido por computadora (computer-aided design, CAD) ayude al diseño de nuevos productos. Puede que el sistema CAD extraiga datos de la base de datos mediante instrucciones de SQL, procese internamente los datos, quizás interactuando con un diseñador de productos, y luego actualice la base de datos. Durante este proceso, el control de los datos puede pasar a manos de varios diseñadores de productos y de otras personas. Como ejemplo adicional, considere un informe de gastos de viaje. Lo crea un empleado que vuelve de un viaje de negocios (posiblemente mediante un paquete de software especial) y luego se envía al jefe de ese empleado, quizás a otros jefes de niveles superiores y, finalmente, al departamento de contabilidad para su pago (momento en el que interactúa con los sistemas informáticos de contabilidad de la empresa).

El término *flujo de trabajo* hace referencia a la combinación de datos y de tareas implicados en procesos como los de los ejemplos anteriores. Los flujos de trabajo interactúan con el sistema de bases de datos cuando se mueven entre los usuarios y estos llevan a cabo sus tareas en el flujo de trabajo. Además de los datos sobre los que opera el flujo de trabajo, puede que la base de datos almacene datos sobre el propio flujo de trabajo, incluidas las tareas que lo conforman y la manera en que se han de hacer llegar a los usuarios. Por tanto, los flujos de trabajo especifican una serie de consultas y de actualizaciones de la base de datos que pueden tenerse en cuenta como parte del proceso de diseño de la base de datos. En otras palabras, el modelado de la empresa no solo exige la comprensión de la semántica de los datos, sino también la de los procesos comerciales que los usan.

### 7.10.5. Otros elementos del diseño de bases de datos

El diseño de bases de datos no suele ser una actividad que se pueda dar por acabada en una sola vez. Las necesidades de las organizaciones evolucionan continuamente, y los datos que necesitan almacenar también evolucionan en consonancia. Durante las fases iniciales del diseño de la base de datos, o durante el desarrollo de las aplicaciones, puede que el diseñador de la base de datos se dé cuenta de que hacen falta cambios en el nivel del esquema conceptual, lógico o físico. Los cambios del esquema pueden afectar a todos los aspectos de la aplicación de bases de datos. Un buen diseño de bases de datos se anticipa a las necesidades futuras de la organización y el diseño se lleva a cabo de manera que se necesiten modificaciones mínimas a medida que evolucionen las necesidades de la organización.

Es importante distinguir entre las restricciones fundamentales que se espera sean permanentes y las que se anticipa que puedan cambiar. Por ejemplo, la restricción de que cada identificador de profesor identifique a un solo profesor es fundamental. Por otro lado, la universidad puede tener la norma de que cada profesor solo pueda estar asociado a un departamento, lo que puede cambiar más adelante si se permiten otras asociaciones. Un diseño de la base de datos que solo permita un departamento por profesor necesitaría cambios importantes si se pudiesen realizar otras asociaciones. Estas asociaciones se pueden representar añadiendo una relación extra, sin modificar la relación *profesor*, siempre que los profesores tengan solo asociación a un departamento principal; el cambio de la norma que permite más de una asociación principal puede requerir un cambio mayor en el diseño de la base de datos. Un buen diseño debería tener en cuenta no solo las normas actuales, sino que debería evitar o minimizar cambios porque se han anticipado dichos cambios, o existe una cierta probabilidad de que estos se produzcan.

Además, es probable que la empresa a la que sirve la base de datos interactúe con otras empresas y, por tanto, puede que tengan que interactuar varias bases de datos. La conversión de los datos entre esquemas diferentes es un problema importante en las aplicaciones del mundo real. Se han propuesto diferentes soluciones para este problema. El modelo de datos XML, que se estudia en el Capítulo 23, se usa mucho para representar los datos cuando estos se intercambian entre diferentes aplicaciones.

Finalmente, merece la pena destacar que el diseño de bases de datos es una actividad orientada a los seres humanos en dos sentidos: los usuarios finales son personas (aunque se sitúe alguna aplicación entre la base de datos y los usuarios finales) y el diseñador de la base de datos debe interactuar intensamente con los expertos en el dominio de la aplicación para comprender los requisitos de datos de la aplicación. Todas las personas involucradas con los datos tienen necesidades y preferencias que se deben tener en cuenta para que el diseño y la implantación de la base de datos tengan éxito en la empresa.

## 7.11. Resumen

- El diseño de bases de datos supone principalmente el diseño del esquema de la base de datos. El modelo de datos **entidad-relación (E-R)** es un modelo de datos muy usado para el diseño de bases de datos. Ofrece una representación gráfica adecuada para ver los datos, las relaciones y las restricciones.
- El modelo está pensado principalmente para el proceso de diseño de la base de datos. Se desarrolló para facilitar el diseño de bases de datos al permitir la especificación de un **esquema de la empresa.** Este esquema representa la estructura lógica general de la base de datos. Esta estructura general se puede expresar gráficamente mediante un **diagrama E-R.**
- Una **entidad** es un objeto que existe en el mundo real y es distinguible de otros objetos. Esa distinción se expresa asociando a cada entidad un conjunto de atributos que describen el objeto.
- Una **relación** es una asociación entre diferentes entidades. Un **conjunto de relaciones** es una colección de relaciones del mismo tipo, y un **conjunto de entidades** es una colección de entidades del mismo tipo.
- Los términos **superclave, clave candidata** y **clave primaria** se aplican a conjuntos de entidades y de relaciones al igual que a los esquemas de relación. La identificación de la clave primaria de un conjunto de relaciones requiere un cierto cuidado, ya que se compone de atributos de uno o más de los conjuntos de entidades relacionados.
- La **correspondencia de cardinalidades** expresa el número de entidades con las que otra entidad se puede asociar mediante un conjunto de relaciones.
- Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina **conjunto de entidades débiles.** Un conjunto de entidades que tiene una clave primaria se denomina **conjunto de entidades fuertes.**
- Las diferentes características del modelo E-R ofrecen al diseñador de bases de datos numerosas opciones a la hora de representar lo mejor posible la empresa que se modela. Los conceptos y los objetos pueden, en ciertos casos, representarse mediante entidades, relaciones o atributos. Ciertos aspectos de la estructura global de la empresa se pueden describir mejor usando los conjuntos de entidades débiles, la generalización, la especialización o la agregación. A menudo, el diseñador debe sopesar las ventajas de un modelo simple y compacto frente a las de otro más preciso pero más complejo.
- El diseño de una base de datos especificado en un diagrama E-R se puede representar mediante un conjunto de esquemas de relación. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay un solo esquema de relación al que se le asigna el nombre del conjunto de entidades o de relaciones correspondiente. Esto forma la base para la obtención del diseño de la base de datos relacional a partir del E-R.
- La **especialización** y la **generalización** definen una relación de inclusión entre un conjunto de entidades de nivel superior y uno o más conjuntos de entidades de nivel inferior. La especialización es el resultado de tomar un subconjunto de un conjunto de entidades de nivel superior para formar un conjunto de entidades de nivel inferior. La generalización es el resultado de tomar la unión de dos o más conjuntos disjuntos de entidades (de nivel inferior) para producir un conjunto de entidades de nivel superior. Los atributos de los conjuntos de entidades de nivel superior los heredan los conjuntos de entidades de nivel inferior.
- La **agregación** es una abstracción en la que los conjuntos de relaciones (junto con sus conjuntos de entidades asociados) se tratan como conjuntos de entidades de nivel superior y pueden participar en las relaciones.
- El **lenguaje de modelado unificado (UML)** es un lenguaje de modelado muy usado. Los diagramas de clases de UML son muy utilizados para el modelado de clases, así como para otros objetivos de modelado de datos.

## Términos de repaso

- Modelo de datos entidad-relación.
- Entidad y conjunto de entidades.
  - Atributos.
  - Dominio.
  - Atributos simples y compuestos.
  - Atributos de un solo valor y multivalorados.
  - Valor nulo (*null*).
  - Atributo derivado.
  - Superclave, clave candidata y clave primaria.
- Relaciones y conjuntos de relaciones.
  - Conjunto de relaciones binarias.
  - Grado del conjunto de relaciones.
  - Atributos descriptivos.
  - Superclave, clave candidata y clave primaria.
  - Rol.
  - Conjunto de relaciones recursivo.
- Diagrama E-R.
- Correspondencia de cardinalidades.
  - Relación uno a uno.
  - Relación uno a varios.
  - Relación varios a uno.
  - Relación varios a varios.
- Participación.
  - Participación total.
  - Participación parcial.
- Conjuntos de entidades débiles y fuertes.
  - Atributo discriminador.
  - Relación identificadora.
- Especialización y generalización.
  - Superclase y subclase.
  - Herencia de atributos.
  - Herencia simple y múltiple.
  - Pertenencia definida por condición y definida por el usuario.
  - Generalización disjunta y solapada.
  - Generalización total y parcial.
- Agregación.
- UML.
- Diagramas de clase en UML.

## Ejercicios prácticos

**7.1.** Construya un diagrama E-R para una compañía de seguros de coches cuyos clientes poseen uno o más coches cada uno. Cada coche tiene asociado un valor de cero al número de accidentes registrados. Cada póliza de seguros cubre uno o más coches, y tiene uno o más pagos «premium» asociados. Cada pago es para un periodo de tiempo, y tiene asociado un plazo y la fecha en la que el pago se ha efectuado.

**7.2.** Considere una base de datos usada para registrar las notas que obtienen los estudiantes en diferentes exámenes de distintas asignaturas ofertadas.

- **a.** Construya un diagrama E-R que modele los exámenes como entidades y utilice una relación ternaria para la base de datos.
- **b.** Construya un diagrama E-R alternativo que solo utilice una relación binaria entre *estudiantes* y *asignaturas*. Hay que asegurarse de que solo existe una relación entre cada pareja formada por un estudiante y una asignatura, pero se pueden representar las notas que obtiene cada estudiante en diferentes exámenes de una asignatura.

**7.3.** Diseñe un diagrama E-R para almacenar los logros de su equipo deportivo favorito. Se deben almacenar los partidos jugados, el resultado de cada partido, los jugadores de cada partido y las estadísticas de cada jugador en cada partido. Las estadísticas resumidas se deben representar como atributos derivados.

**7.4.** Suponga un diagrama E-R en el que el mismo conjunto de entidades aparezca varias veces. ¿Por qué permitir esa redundancia es una mala práctica que se debe evitar siempre que sea posible?

**7.5.** Los diagramas E-R se pueden ver como grafos. ¿Qué significa lo siguiente en términos de la estructura del esquema de una empresa?

- **a.** El grafo es inconexo.
- **b.** El grafo tiene un ciclo.

**7.6.** Considere la representación de una relación ternaria mediante relaciones binarias como se describió en la Sección 7.7.3 que se muestra en la Figura 7.27b (no se muestran los atributos).

- **a.** Muestre un ejemplar simple de $E$, $A$, $B$, $C$, $R_A$, $R_B$ y $R_C$ que no pueda corresponder a ningún ejemplar de $A$, $B$, $C$ y $R$.
- **b.** Modifíquese el diagrama E-R de la Figura 7.27b para introducir restricciones que garanticen que cualquier ejemplar $E$, $A$, $B$, $C$, $R_A$, $R_B$ y $R_C$ que satisfaga las restricciones corresponda a algún ejemplar de $A$, $B$, $C$ y $R$.
- **c.** Modifique la traducción anterior para manejar las restricciones de participación total sobre las relaciones ternarias.
- **d.** La representación anterior exige que se cree un atributo de clave primaria para $E$. Muéstrese la manera de tratar $E$ como un conjunto de entidades débiles de forma que no haga falta ningún atributo de clave primaria.

**7.7.** Los conjuntos de entidades débiles siempre se pueden convertir en conjuntos de entidades fuertes simplemente añadiendo a sus atributos los atributos de clave primaria del conjunto de entidades identificadoras. Describa el tipo de redundancia que se produce al hacerlo.

**7.8.** Considere una relación como *secc_asignatura*, generada desde una relación varios a uno *secc_asignatura*. ¿Las restricciones de clave primaria y externa creadas en la relación duerzan a la restricción de cardinalidad varios a uno? Explique por qué.

**7.9.** Suponga que la relación *tutor* es de uno a uno. ¿Qué restricciones extra son necesarias en la relación *tutor* para asegurar que se cumple la restricción de cardinalidad uno a uno?

**7.10.** Considere una relación varios a uno $R$ entre los conjuntos de entidades $A$ y $B$. Suponga que la relación creada de $R$ se combina con una relación creada de $A$. En SQL, los atributos participantes en la restricción de clave externa puede ser *null*. Explique cómo se puede cumplir la restricción de participación total de $A$ en $R$ utilizando restricciones **not null** en SQL.

**7.11.** En SQL, las restricciones de clave externa solo pueden referenciar los atributos de clave primaria de la relación referenciada, u otros atributos declarados como superclave usando la restricción **unique**. En consecuecia, las restricciones de participación total en una relación de varios a varios (o en el lado «uno» de una relación de uno a varios) no se puede forzar a cumplir en las relaciones creadas a partir de una relación, mediante el uso de restricciones de clave primaria, de clave externa o *not null*.

- **a.** Explique por qué.
- **b.** Explique cómo forzar las restricciones de participación total usando restricciones check complejas o aserciones (véase la Sección 4.4.7). (Desafortunadamente estas características no están disponibles en ninguna de las bases de datos más utilizadas actualmente).

**7.12.** La Figura 7.28 muestra una estructura reticular de generalización y especialización (no se muestran los atributos). Para los conjuntos de entidades $A$, $B$ y $C$ explique cómo se heredan los atributos desde los conjuntos de entidades de nivel superior $X$ e $Y$. Explique la manera de manejar el caso de que un atributo de $X$ tenga el mismo nombre que algún atributo de $Y$.

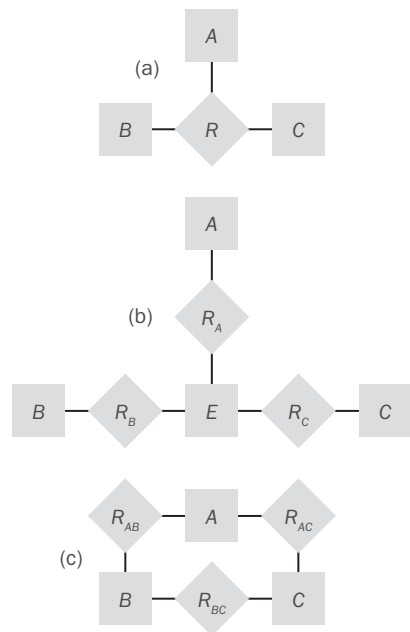



**Figura 7.27.** Diagrama E-R para los Ejercicios prácticos 7.6 y 7.24.

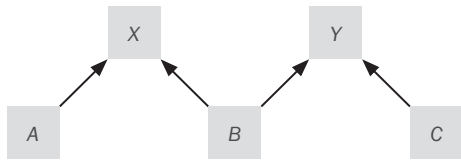



**Figura 7.28.** Diagrama E-R para el Ejercicio práctico 7.12.

**7.13. Cambios temporales:** un diagrama E-R normalmente modela el estado de una empresa en un momento del tiempo. Suponga que se desea hacer un seguimiento de *cambios temporales*, es decir, los cambios de los datos con el tiempo. Por ejemplo, Zhang puede que haya sido estudiante entre el 1 de septiembre de 2005 y el 31 de mayo de 2009, mientras que Shankar puede haber tenido como tutor al profesor Einstein desde el 31 de mayo de 2008 al 5 de diciembre de 2008 y de nuevo desde el 1 de junio de 2009 al 5 de enero de 2010. De forma similar, los valores de un atributo de una entidad o de una relación, como el *nombre* de la asignatura o los *créditos* de la misma, el *sueldo*, el *nombre* de un *profesor* y el *tot_créd* de un *estudiante*, pueden cambiar con el tiempo.

Una forma de modelar los cambios temporales es la siguiente: se define un nuevo tipo de dato que se denomine **tiempo válido,** que es un intervalo de tiempo o un conjunto de intervalos de tiempo. Entonces, se asocia un atributo *tiempo_válido* a cada una de las entidades y relaciones, registrando los periodos de tiempo durante los cuales la entidad o la relación es válida. La fecha final de un intervalo puede ser infinita; por ejemplo, si Shankar se convierte en estudiante el 2 de septiembre de 2008, y aún sigue siendo estudiante, la ficha final se puede representar como infinito para la entidad Shankar. De forma similar, se modelan los atributos que van a cambiar con el tiempo como un conjunto de valores, cada uno con su propio *tiempo_válido*.

**a.** Dibuje un diagrama E-R con las entidades *estudiante* y *profesor*, y la relación *tutor*, con las extensiones anteriores para hacer un seguimiento de los cambios temporales.

**b.** Convierta el diagrama E-R anterior en un conjunto de relaciones.

Debería quedar claro que estos conjuntos de relaciones que se han generado resultan bastante complejos, lo que conduce a dificultades en las tareas al escribir las consultas en SQL. Otra forma de hacerlo que se utiliza más es ignorar los cambios temporales cuando se diseña en modelo E-R (en particular los cambios en el tiempo de los valores de los atributos), y modificar las relaciones generadas a partir del modelo E-R para hacer el seguimiento de los cambios temporales, como se verá más adelante en la Sección 8.9.

## Ejercicios

**7.14.** Explique las diferencias entre los términos clave primaria, clave candidata y superclave.

**7.15.** Construya un diagrama E-R para un hospital con un conjunto de pacientes y un conjunto de médicos. Asóciese con cada paciente un registro de las diferentes pruebas y exámenes realizados.

**7.16.** Construya los esquemas de relación adecuados para cada uno de los diagramas E-R de los Ejercicios prácticos 7.1 a 7.3.

**7.17.** Extienda el diagrama E-R del Ejercicio práctico 7.3 para que almacene la misma información para todos los equipos de una liga.

**7.18.** Explique las diferencias entre los conjuntos de entidades débiles y los conjuntos de entidades fuertes.

**7.19.** Se puede convertir cualquier conjunto de entidades débiles en un conjunto de entidades fuertes simplemente añadiendo los atributos apropiados. ¿Por qué, entonces, se tienen conjuntos de entidades débiles?

**7.20.** Considere el diagrama E-R de la Figura 7.29, que modela una librería en línea.

**a.** Liste los conjuntos de entidades y sus claves primarias.

**b.** Supóngase que la librería añade discos Blu-Ray y vídeos descargables a su colección. El mismo elemento puede estar presente en uno o en ambos formatos, con diferentes precios. Extienda el diagrama E-R para modelar esta adición, ignorando el efecto sobre las cestas de la compra.

**c.** Extienda ahora el diagrama E-R mediante la generalización para modelar el caso en que una cesta de la compra pueda contener cualquier combinación de libros, discos Blu-Ray o vídeos descargables.

**7.21.** Diseñe una base de datos para una compañía de automóviles para proporcionar a los concesionarios asistencia en el mantenimiento de la información de los registros de clientes y el inventario del concesionario, y ayudar a los vendedores a realizar las peticiones de coches.

Los vehículos están identificados por un número de identificación de vehículo (NIV). Cada vehículo pertenece a un modelo concreto de una determinada marca de coche que vende la compañía (por ejemplo, el XF es un modelo de coche de la marca Jaguar de Tata Motors). Cada modelo se puede ofrecer con distintas opciones, pero un coche dado solo puede tener algunas (o ninguna) de las opciones disponibles. La base de datos debe guardar información de los modelos, marcas y opciones, así como sobre cada uno de los concesionarios, clientes y coches.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

**7.22.** Diseñe una base de datos para una compañía de logística internacional de paquetería (como DHL o FedEx). La base de datos debe poder realizar un seguimiento de los clientes (quién envía los paquetes) y los clientes (quien recibe los paquetes); algunos clientes pueden estar en las dos partes.

Se debe poder identificar y trazar cada uno de los paquetes, por lo que la base de datos debe poder guardar la ubicación de un paquete y su histórico de ubicaciones. Las ubicaciones pueden incluir camiones, aviones, aeropuertos y almacenes.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

**7.23.** Diseñe la base de datos de una línea aérea. La base de datos debe guardar información de los clientes y sus reservas, los vuelos y su estado, la asignación de asientos en cada uno de los vuelos, así como el plan y ruta de los futuros vuelos.

El diseño debería incluir un diagrama E-R, un conjunto de esquemas relacionales y una lista de restricciones, incluyendo restricciones de clave primaria y externa.

**7.24.** En la Sección 7.7.3 se representó una relación ternaria (que se repite en la Figura 7.27a) mediante relaciones binarias, como se muestra en la Figura 7.27b. Considere la alternativa mostrada en la Figura 7.27c. Explique las ventajas relativas de estas dos representaciones alternativas de una relación ternaria mediante relaciones binarias.

**7.25.** Considere los esquemas de relación mostrados en la Sección 7.6, que se generaron a partir del diagrama E-R de la Figura 7.15. Para cada esquema, especifique las restricciones de clave externa que hay que crear, si es que hay que crear alguna.

**7.26.** Diseñe una jerarquía de especialización–generalización para una compañía de venta de vehículos de motor. La compañía vende motocicletas, coches, furgonetas y autobuses. Justifique la ubicación de los atributos en cada nivel de la jerarquía.

Explique el motivo por el que no se deben ubicar en un nivel superior o inferior.

**7.27.** Explique la diferencia entre las restricciones definidas por condición y las definidas por el usuario. ¿Cuáles de estas restricciones pueden comprobar el sistema automáticamente?

**7.28.** Explique la diferencia entre las restricciones disjuntas y las solapadas.

**7.29.** Explique la diferencia entre las restricciones totales y las parciales.

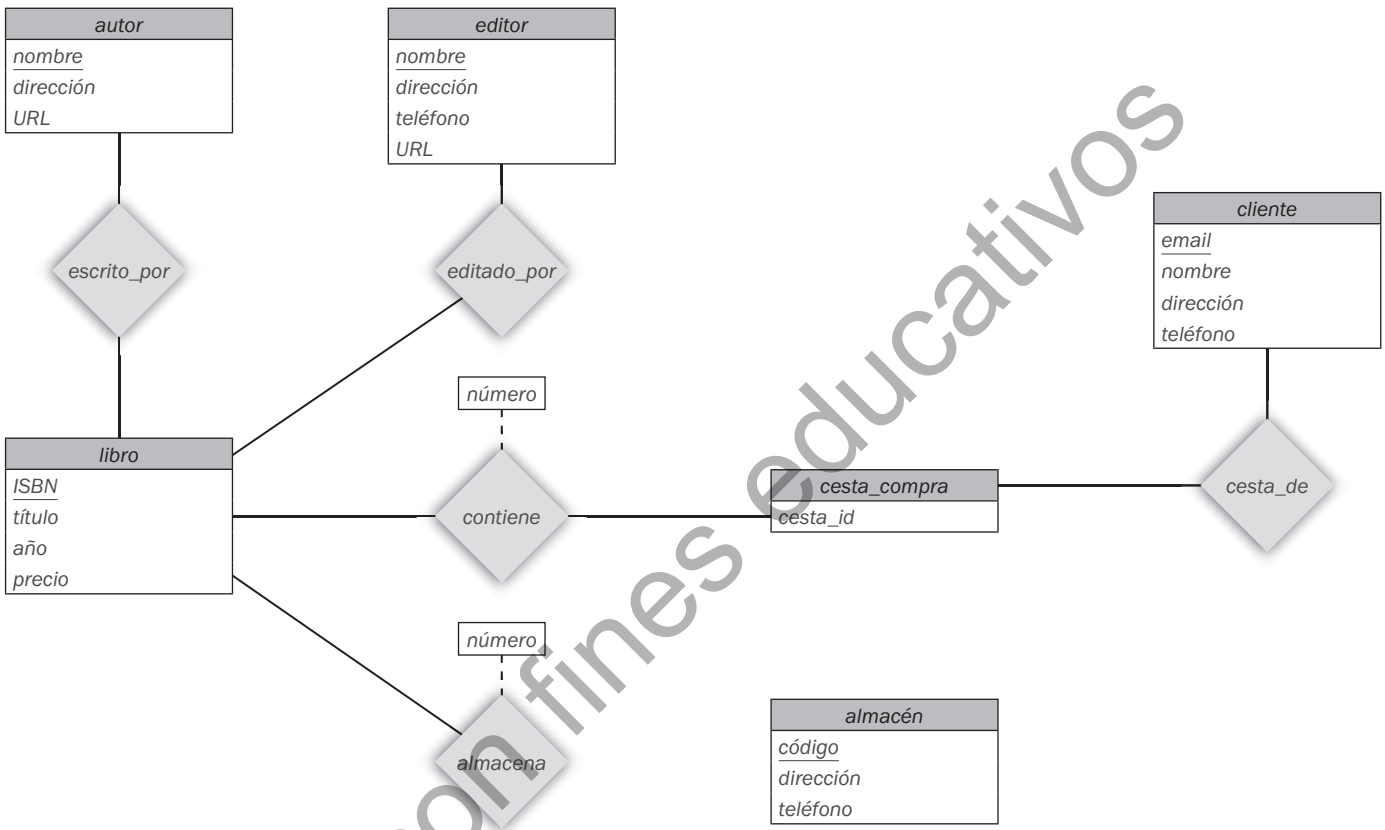



**Figura 7.29.** Diagrama E-R para el Ejercicio 7.20.



## Herramientas

Muchos sistemas de bases de datos ofrecen herramientas para el diseño de bases de datos que soportan los diagramas E-R. Estas herramientas ayudan al diseñador a crear diagramas E-R y pueden crear automáticamente las tablas correspondientes de la base de datos. Véanse las notas bibliográficas del Capítulo 1 para consultar referencias de sitios web de fabricantes de bases de datos.

También hay distintas herramientas de modelado de datos independientes de las bases de datos que soportan los diagramas E-R y los diagramas de clases de UML. La herramienta de dibujo Dia, que está disponible como freeware, dispone de diagramas E-R y de diagramas de clase de UML. Entre las herramientas comerciales están Rational Rose (www.ibm.com/software/rational), Microsoft Visio (véase www.microsoft.com/office/visio), CA's ERwin (www.ca.com/us/data-modeling.aspx), Poseidon for UML (www.gentleware.com), y SmartDraw (www.smartdraw.com).

## Notas bibliográficas

El modelo de datos E-R fue introducido por Chen [1976]. Teorey et ál. [1986] presentaron una metodología de diseño lógico para las bases de datos relacionales que usa el modelo E-R extendido. La norma de definición de la integración para el modelado de información (IDEF1X, Integration Definition for Information Modeling) IDEF1X [1993], publicado por el Instituto Nacional de Estados Unidos para Normas y Tecnología (United States National Institute of Standards and Technology, NIST) definió las normas para los diagramas E-R. No obstante, hoy en día se usa una gran variedad de notaciones E-R.

Thalheim [2000] proporciona un tratamiento detallado, pero propio de un libro de texto, de la investigación en el modelado E-R. En los libros de texto, Batini et ál. [1992] y Elmasri y Navathe [2003] ofrecen explicaciones básicas. Davis et ál. [1983] proporcionan una colección de artículos sobre el modelo E-R.

En 2009 la versión más reciente de UML era la 2.2, con la versión 2.3 cerca de la adopción final. Véase www.uml.org para obtener más información sobre las normas y las herramientas de UML.

# Diseño de bases de datos y el modelo E-R

# 08

En este capítulo se considera el problema de diseñar el esquema de una base de datos relacional. Muchos de los problemas que conlleva son parecidos a los de diseño que se han considerado en el Capítulo 7 en relación con el modelo E-R.

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

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

## 8.1. Características de los buenos diseños relacionales

El estudio del diseño entidad-relación llevado a cabo en el Capítulo 7 ofrece un excelente punto de partida para el diseño de bases de datos relacionales. Ya se vio en la Sección 7.6 que es posible generar directamente un conjunto de esquemas de relación a partir del diseño E-R. Evidentemente, la adecuación (o no) del conjunto de esquemas resultante depende, en primer lugar, de la calidad del diseño E-R. Más adelante en este capítulo se estudiarán maneras precisas de evaluar la adecuación de los conjuntos de esquemas de relación. No obstante, es posible llegar a un buen diseño empleando conceptos que ya se han **estudiado.**

Para facilitar las referencias, en la Figura 8.1 se repiten los esquemas de la base de datos de la universidad.

*aula* (*edificio*, *número_aula*, *capacidad*)
*departamento* (*nombre_dept*, *edificio*, *presupuesto*)
*asignatura* (*asignatura_id*, *nombre_asig*, *nombre_dept*, *créditos*)
*profesor* (*ID*, *nombre*, *nombre_dept*, *sueldo*)
*sección* (*asignatura_id*, *secc_id*, *semestre*, *año*, *edificio*, *número_aula*, *franja_horaria_id*)
*enseña* (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*)
*estudiante* (*ID*, *nombre*, *nombre_dept*, *tot_cred*)
*matricula* (*ID*, *asignatura_id*, *secc_id*, *semestre*, *año*, *nota*)
*tutor* (*e_ID*, *p_ID*)
*franja_horaria* (*franja_horaria_id*, *día*, *hora_inicio*, *hora_fin*)
*prerreq* (*asignatura_id*, *prerreq_id*)

**Figura 8.1.** Esquema de base de datos de la universidad.

### 8.1.1. Alternativa de diseño: esquemas grandes

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

*profesor_dept (ID, nombre, sueldo, nombre_dept, edificio, presupuesto)*

Esto representa el resultado de la reunión natural de las relaciones correspondientes a *profesor* y *departamento*. Parece una buena idea, ya que es posible expresar algunas consultas empleando menos reuniones, hasta que se considera detenidamente la realidad de la universidad que condujo a nuestro diseño E-R.

Considérese el ejemplar de la relación *profesor_dept* que se ve en la Figura 8.2. Téngase en cuenta que hay que repetir la información de departamento («edificio» y «presupuesto») para cada profesor del departamento. Por ejemplo, la información sobre el departamento Informática (Taylor, 100000) se incluye en las tuplas de los profesores Katz, Srinivasan y Brandt.

| *ID* | *nombre* | *sueldo* | *nombre_dept* | *edificio* | *presupuesto* |
|---|---|---|---|---|---|
| 22222 | Einstein | 95000 | Física | Watson | 70000 |
| 12121 | Wu | 90000 | Finanzas | Painter | 120000 |
| 32343 | El Said | 60000 | Historia | Painter | 50000 |
| 45565 | Katz | 75000 | Informática | Taylor | 100000 |
| 98345 | Kim | 80000 | Electrónica | Taylor | 85000 |
| 76766 | Crick | 72000 | Biología | Watson | 90000 |
| 10101 | Srinivasan | 65000 | Informática | Taylor | 100000 |
| 58583 | Califieri | 62000 | Historia | Painter | 50000 |
| 83821 | Brandt | 92000 | Informática | Taylor | 100000 |
| 15151 | Mozart | 40000 | Música | Packard | 80000 |
| 33456 | Gold | 87000 | Física | Watson | 70000 |
| 76543 | Singh | 80000 | Finanzas | Painter | 120000 |

**Figura 8.2.** Tabla *profesor_dept*.

Es importante que todas estas tuplas coincidan en la cuantía del presupuesto ya que si no la base de datos se volvería inconsistente. En el diseño original que utilizaba *profesor* y *departamento*, se almacenaba la cuantía de cada presupuesto una única vez. Ello sugiere que el uso de *profesor_dept* es una mala idea, ya que el presupuesto se almacena de forma redundante y se corre el riesgo de que algún usuario actualice el presupuesto en una de las tuplas pero no en todas y por tanto se cree una inconsistencia.

Aunque se decidiese mantener el problema de la redundancia, existe otro problema con el esquema *profesor_dept*. Supóngase que se crea un nuevo departamento en la universidad. En el diseño alternativo anterior, no se puede representar directamente la información de un departamento *(nombre_dept, edificio, presupuesto)* a no ser que ese departamento tenga al menos un profesor de la universidad. Es así

porque las tuplas de la tabla *profesor_dept* requieren valores para *ID*, *nombre* y *sueldo*. Esto significa que no se puede registrar información sobre el nuevo departamento creado hasta que se contrate al primer profesor para dicho departamento. En el esquema anterior, el esquema *departamento* podía tratar esta situación, pero en el diseño revisado se debería crear una tupla con un valor *null* (nulo) en *edificio* y en *presupuesto*. En algunos casos los valores *null* generan problemas, como se verá al tratar SQL. Sin embargo, si se decide que no supone un problema en este caso, vamos a intentar utilizar el nuevo esquema.

### 8.1.2. Alternativa de diseño: esquemas pequeños

Supóngase nuevamente que, de algún modo, se ha comenzado a trabajar con el esquema *profesor_dept*. ¿Cómo reconocer la necesidad de repetición de la información y de dividirla en dos esquemas, *profesor* y *departamento*?

Al observar el contenido de la relación en el esquema *profesor_dept*, se puede observar la repetición de la información que surge de realizar el listado del edificio y el presupuesto una vez para cada profesor de un departamento. Sin embargo, se trata de un proceso en el que no se puede confiar. Las bases de datos reales tienen gran número de esquemas y un número todavía mayor de atributos. El número de tuplas puede ser del orden de millones o superior. Descubrir la repetición puede resultar costoso. Hay un problema más fundamental todavía en este enfoque. No permite determinar si la carencia de repeticiones es meramente un caso especial «afortunado» o la manifestación de una regla general. En nuestro ejemplo, ¿cómo podríamos *saber* que en nuestra universidad, todos los departamentos (identificados por su nombre de departamento) *tienen que* encontrarse en un único edificio y deben tener un único presupuesto? ¿El hecho de que el presupuesto del departamento Informática aparezca tres veces con el mismo presupuesto es solo una coincidencia? Estas preguntas no se pueden responder sin volver a la propia universidad y comprender sus reglas de funcionamiento. En concreto, hay que averiguar si la universidad exige que todos los departamentos (identificados por su nombre de departamento) *deben* tener un único edificio y un único valor de presupuesto.

En el caso de *profesor_dept*, el proceso de creación del diseño E-R logró evitar la creación de ese esquema. Sin embargo, esta situación fortuita no se produce siempre.

Por tanto, hay que permitir que el diseñador de la base de datos especifique normas como «cada valor de *nombre_dept* corresponde, como máximo, a un *presupuesto*», incluso en los casos en los que *nombre_dept* no sea la clave primaria del esquema en cuestión. En otras palabras, hay que escribir una norma que diga «si hay un esquema (*nombre_dept*, *presupuesto*), entonces *nombre_dept* puede hacer de clave primaria». Esta regla se especifica como la **dependencia funcional.**

$$nombre\_dept \rightarrow presupuesto$$

Dada esa regla, ya se tiene suficiente información para reconocer el problema del esquema *profesor_dept*. Como *nombre_dept* no puede ser la clave primaria de *profesor_dept* (porque puede que cada departamento necesite varias tuplas de la relación del esquema *profesor_dept*), tal vez haya que repetir el importe del presupuesto.

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

No todas las descomposiciones de los esquemas resultan útiles. Considérese el caso extremo en que todo lo que tengamos sean esquemas con un solo atributo. No se puede expresar ninguna relación interesante de ningún tipo. Considérese ahora un caso menos extremo en el que se decida descomponer el esquema *empleado* (véase Sección 7.8):

*empleado* (*ID, nombre, calle, ciudad, sueldo*)

en los siguientes dos esquemas:

*empleado1* (*ID, nombre*)
*empleado2* (*nombre, calle, ciudad, sueldo*)

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

(57766, Kim, Main, Perryridge, 75000)
(98776, Kim, North, Hampton, 67000)

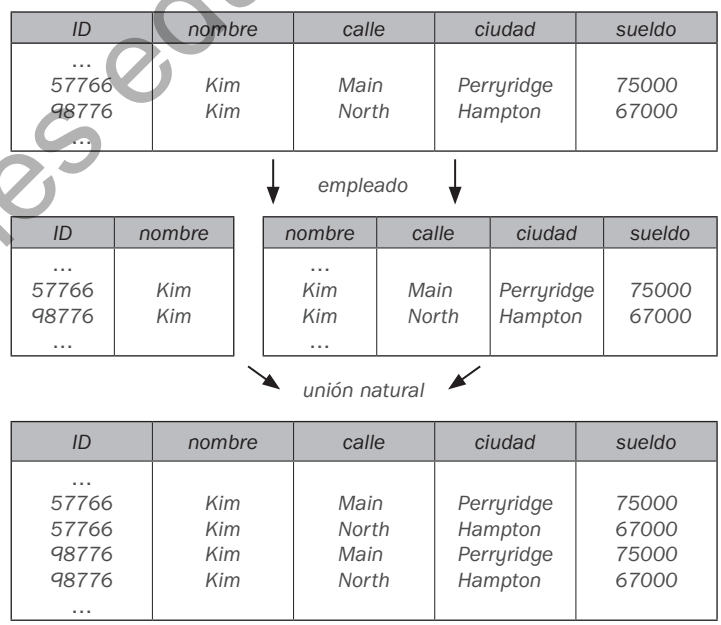



**Figura 8.3.** Pérdida de información debida a una mala descomposición.

La Figura 8.3 muestra estas tuplas, las tuplas resultantes al utilizar los esquemas procedentes de la descomposición y el resultado que se obtendría si se intentara volver a generar las tuplas originales mediante una reunión natural. Como se ve en la figura, las dos tuplas originales aparecen en el resultado junto con dos tuplas nuevas que mezclan de manera incorrecta valores de fechas correspondientes a los dos empleados llamados Kim. Aunque se dispone de más tuplas, en realidad se tiene menos información en el sentido siguiente: se puede indicar que un determinado número de teléfono y una fecha de contratación dada corresponden a alguien llamado Kim, pero no se puede distinguir a cuál de ellos. Por tanto, la descomposición propuesta es incapaz de representar algunos datos importantes de los empleados de la universidad. Evidentemente, es preferible evitar este tipo de descomposiciones. Estas descomposiciones se denominan **descomposiciones con pérdidas,** y, a la inversa, aquellas que no tienen pérdidas se denominan **descomposiciones sin pérdidas.**

## 8.2. Dominios atómicos y la primera forma normal

El modelo E-R permite que los conjuntos de entidades y los de relaciones tengan atributos con algún tipo de subestructura. Concretamente, permite atributos como *número_teléfono* de la Figura 7.11 y atributos compuestos (como el atributo *dirección* con los atributos *calle, ciudad, estado,* y *CP*). Cuando se crean tablas a partir de los diseños E-R que contienen ese tipo de atributos, se elimina esa subestructura. Para los atributos compuestos, se deja que cada atributo componente sea un atributo de pleno derecho. Para los atributos multivalorados, se crea una tupla por cada elemento del conjunto multivalorado.

En el modelo relacional se formaliza esta idea de que los atributos no tienen subestructuras. Un dominio es **atómico** si se considera que los elementos de ese dominio son unidades indivisibles. Se dice que el esquema de la relación $R$ está en la **primera forma normal** (1FN) si los dominios de todos los atributos de $R$ son atómicos.

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

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

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

Como ilustración práctica del punto anterior, considérese una organización que asigna a los empleados números de identificación de la manera siguiente: las dos primeras letras especifican el departamento y las cuatro cifras restantes son un número único para cada empleado dentro de ese departamento. Ejemplos de estos números pueden ser «CS001» y «EE1127». Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de la relación tuviera un atributo cuyo dominio consistiera en números de identificación así codificados, el esquema no se hallaría en la primera forma normal.

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

De lo dicho hasta este momento, puede parecer que nuestro uso de los identificadores de asignatura como «CS-101», donde «CS» indica departamento de Informática, significa que el dominio de identificadores de asignatura no es atómico. Este dominio no es atómico en cuanto a las personas que utilizan este sistema se refiere. Sin embargo, la aplicación de la base de datos trata al dominio como atómico, siempre que no intente dividir el identificador e interpretar las partes del mismo como una abreviatura del departamento. El esquema de la asignatura guarda el nombre del departamento como un atributo distinto, y la aplicación de la base de datos puede utilizar el valor de este atributo para encontrar el departamento de la asignatura, en lugar de interpretar los caracteres del propio identificador de asignatura. Por tanto, se puede considerar que el esquema de nuestra universidad se encuentra en primera forma normal.

El empleo de atributos de conjunto puede dar lugar a diseños con acúmulo redundante de datos, lo que a su vez genera incosistencias. Por ejemplo, en lugar de representar la relación entre los profesores y las secciones como la relación independiente *enseña*, puede que el diseñador de bases de datos esté tentado a almacenar un conjunto de identificadores de sección de asignaturas con cada profesor y un conjunto de identificadores de profesor con cada sección. (Las claves primarias de *sección* y *profesor* se utilizan como identificadores). Siempre que se modifiquen los datos del profesor que enseña en una sección, habrá que llevar a cabo la actualización en dos lugares: en el conjunto de profesores de la sección y en el conjunto de secciones del profesor. No llevar a cabo esas dos actualizaciones puede dejar la base de datos en un estado inconsistente. Guardar solo uno de esos conjuntos evitaría información repetida, pero complicaría algunas consultas, y además, no está claro a cuál de los dos habría que mantener.

Algunos tipos de valores no atómicos pueden resultar útiles, aunque deben utilizarse con cuidado. Por ejemplo, los atributos con valores compuestos suelen resultar útiles, y los atributos con el valor determinado por un conjunto también resultan útiles en muchos casos, que es el motivo por el que el modelo E-R los soporta. En muchos dominios en los que las entidades tienen una estructura compleja, la imposición de la representación en la primera forma normal supone una carga innecesaria para el programador de las aplicaciones, que tiene que escribir código para convertir los datos a su forma atómica. También hay sobrecarga en tiempo de ejecución por la conversión de los datos de su forma habitual a su forma atómica. Por tanto, el soporte de los valores no atómicos puede resultar muy útil en ese tipo de dominios. De hecho, los sistemas modernos de bases de datos soportan muchos tipos de valores no atómicos, como se verá en el Capítulo 22. Sin embargo, en este capítulo nos limitamos a las relaciones en la primera forma normal y, por tanto, todos los dominios son atómicos.

## 8.3. Descomposición mediante dependencias funcionales

En la Sección 8.1 se indicó que hay una metodología formal para evaluar si un esquema relacional debe descomponerse. Esta metodología se basa en los conceptos de clave y de dependencia funcional.

Para ver los algoritmos de diseño de bases de datos relacionales, necesitamos hablar de relaciones arbitrarias y sus esquemas, en lugar de tratar solamente sobre un ejemplo. Recordando la introducción al modelo relacional del Capítulo 2, vamos a resumir la notación empleada.

- En general, se utilizan letras griegas para los conjuntos de atributos (por ejemplo, $\alpha$). Se utilizan letras latinas minúsculas seguidas de una mayúscula entre paréntesis para referirse a un esquema de relación [por ejemplo, $r(R)$]. Se usa la notación $r(R)$ para *mostrar que el esquema es para la relación r,* con $R$ indicando el conjunto de atributos; pero algunas veces simplifica la notación para utilizar solo $R$ cuando el nombre de la relación no es importante.

  Por supuesto que un esquema de relación es un conjunto de atributos, pero no todos los conjuntos de atributos son esquemas. Cuando se utiliza una letra griega minúscula se refiere a un conjunto de atributos que pueden ser o no un esquema. Se utiliza una letra latina cuando se quiere indicar que el conjunto de atributos es definitivamente un esquema.

- Cuando un conjunto de atributos es una superclave, se indica con *K*. Una superclave pertenece a un esquema de relación específico, por lo que se utiliza la terminología «*K* es una superclave de *r* (*R*)».
- Se utiliza un nombre en minúsculas para las relaciones. En nuestro ejemplo, se intenta que los nombres sean realistas (por ejemplo, *profesor*), mientras que en nuestras definiciones y algoritmos se sugiere utilizar solo letras como *r*.
- Una relación, por supuesto, tiene un valor concreto en un momento dado, al que nos referimos como ejemplar y se usa el término «ejemplar de *r*». Cuando queda claro que se refiere a un ejemplar, se utiliza simplemente el nombre de la relación (por ejemplo, *r*).

### 8.3.1. Claves y dependencias funcionales

Una base de datos modela un conjunto de entidades y relaciones del mundo real. Normalmente existen diversas restricciones (reglas) que se aplican sobre los datos del mundo real. Por ejemplo, algunas de las restricciones que se espera que existan en la base de datos de una universidad son:

1. Los estudiantes y los profesores se identifican de forma única por su ID.
2. Los estudiantes y los profesores tienen un único nombre.
3. Los profesores y los estudiantes están (en general) asociados a un único departamento.[1]
4. Todos los departamentos tienen un único valor de presupuesto, y solo un edificio asociado.

Un ejemplar de relación que satisfaga todas las relaciones del mundo real de este tipo se llama ejemplar legal de relación; un **ejemplar legal** de una base de datos es aquel en el que todos los ejemplares de relación son ejemplares legales.

Algunas de las restricciones más habituales del mundo real se pueden representar formalmente como claves (superclaves, claves candidatas y claves primarias), o como dependencias funcionales, que se definen a continuación.

En la Sección 2.3, se definió la noción de *superclave* como conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una tupla de la relación. Reescribimos esta definición de la siguiente forma: sea $r(R)$ un esquema de relación. Un subconjunto $K$ de $R$ es una **superclave** de $r(R)$ si en cualquier ejemplar legal de $r(R)$, para todos los pares $t_1$ y $t_2$ de tuplas en el ejemplar de $r$, si $t_1 \neq t_2$, entonces $t_1[K] \neq t_2[K]$. Es decir, no puede haber dos tuplas en una instancia legal de una relación $r(R)$ que tengan los mismos valores en el conjunto de atributos $K$. Claramente, si no existen dos tuplas en $r$ que tengan los mismos valores en $K$, entonces un valor $K$ identifica unívocamente una tupla en $r$.

Siempre que una superclave sea un conjunto de atributos que identifique unívocamente una tupla, una dependencia funcional nos permite expresar restricciones que identifiquen unívocamente los valores de ciertos atributos. Supóngase un esquema de relación $r(R)$, y sea $\alpha \subseteq R$ y $\beta \subseteq R$.

- Dado un ejemplar de $r(R)$, se dice que el ejemplar **satisface la dependencia funcional** $\alpha \rightarrow \beta$ si para todos los pares de tuplas $t_1$ y $t_2$ en los ejemplares tales que $t_1[\alpha] = t_2[\alpha]$, también se cumple que $t_1[\beta] = t_2[\beta]$.
- Se dice que una dependencia funcional $\alpha \rightarrow \beta$ **cumple** con el esquema $r(R)$ si para cada ejemplar legal de $r(R)$ satisface la dependencia funcional.

Si utilizamos la notación de dependencia funcional, se dice que $K$ es una *superclave de* $r(R)$ si la dependencia funcional $K \rightarrow R$ cumple con $r(R)$. En otras palabras, $K$ es una superclave si para todos los ejemplares de $r(R)$ y para cada par de tuplas $t_1$ y $t_2$ del ejemplar, siempre que $t_1[K] = t_2[K]$, también se cumple que $t_1[R] = t_2[R]$ (es decir, $t_1 = t_2$).[2]

Las dependencias funcionales permiten expresar las restricciones que no se pueden expresar con superclaves. En la Sección 8.1.2 se consideró el esquema:

*profesor_dept* (*ID, nombre, sueldo, nombre_dept, edificio, presupuesto*)

en el que se cumple la dependencia funcional *nombre_dept* → *presupuesto*, ya que por cada departamento (identificado por *nombre_dept*) existe un único presupuesto.

El hecho de que el par de atributos (*ID, nombre_dept*) forme una superclave para *profesor_dept* se denota escribiendo:

*ID, nombre_dept* → *nombre, sueldo, edificio, presupuesto*

Las dependencias funcionales se emplean de dos maneras:

1. Para probar las relaciones y ver si satisfacen un conjunto dado $F$ de dependencias funcionales.
2. Para especificar las restricciones del conjunto de relaciones legales. Así, *solo* habrá que preocuparse de aquellas relaciones que satisfagan un conjunto dado de dependencias funcionales. Si se desea restringirse a las relaciones del esquema $r(R)$ que satisfacen un conjunto $F$ de dependencias funcionales, se dice que $F$ se **cumple** en $r(R)$.

Considérese la relación $r$ de la Figura 8.4, para ver las dependencias funcionales que se satisfacen. Obsérvese que se satisface A → C. Hay dos tuplas que tienen un valor para $A$ de $a_1$. Estas tuplas tienen el mismo valor de $C$ — por ejemplo, $c_1$. De manera parecida, las dos tuplas con un valor de $a_2$ para $A$ tienen el mismo valor de $C$, $c_2$. No hay más pares de tuplas diferentes que tengan el mismo valor de $A$. Sin embargo, la dependencia funcional $C \rightarrow A$ no se satisface. Para ver esto, considérense las tuplas $t_1 = (a_2, b_3, c_2, d_3)$ y $t_2 = (a_3, b_3, c_2, d_4)$. Estas dos tuplas tienen el mismo valor de $C$, $c_2$, pero tienen valores diferentes para $A$, $a_2$ y $a_3$, respectivamente. Por tanto, se ha hallado un par de tuplas $t_1$ y $t_2$ tal que $t_1[C] = t_2[C]$, pero $t_1[A] \neq t_2[A]$.

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

**Figura 8.4.** Ejemplar de muestra de la relación *r*.

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

1 Un profesor o un estudiante pueden estar asociados a más de un departamento, por ejemplo, una facultad asociada o un departamento menor. Nuestro esquema de universidad simplificada modela solamente el departamento principal al que está asociado un profesor o un estudiante. En una universidad real el esquema debería capturar las asociaciones secundarias en otras relaciones.

2 Tenga en cuenta que se supone que las relaciones son conjuntos. SQL puede tratar con multiconjuntos, y las declaraciones de claves primarias en SQL para un conjunto de atributos $K$ requiere no solo que $t_1 = t_2$ si $t_1[K] = t_2[K]$, sino también que no haya tuplas duplicadas. SQL también requiere que a los atributos en el conjunto $K$ no se les pueda asignar el valor *null*.

Es importante darse cuenta de que una relación dada puede, en cualquier momento, satisfacer algunas dependencias funcionales cuyo cumplimiento no sea necesario en el esquema de la relación. En la relación *aula* de la Figura 8.5 puede verse que se satisface *número_aula* → *capacidad*. Sin embargo, se sabe que en el mundo real dos aulas en diferentes edificios pueden tener el mismo número pero diferentes capacidades. Por tanto, es posible, en un momento dado, tener un ejemplar de la relación *aula* en el que no se satisfaga *número_aula* → *capacidad*. Por consiguiente, no se incluirá *número_aula* → *capacidad* en el conjunto de dependencias funcionales que se cumplen en la relación *aula*. Sin embargo, se espera que la dependencia funcional *edificio, número_aula* → *capacidad* se cumpla en el esquema *aula*.

Dado un conjunto de dependencias funcionales $F$ que se cumple en una relación $r$ ($R$), es posible inferir que también se deban cumplir en esa misma relación otras dependencias funcionales. Por ejemplo, dado el esquema $r$ ($A$, $B$, $C$), si se cumplen en $r$ las dependencias funcionales $A \rightarrow B$ y $B \rightarrow C$, se puede inferir que también $A \rightarrow C$ se debe cumplir en $r$. Esto es porque, dado cualquier valor de $A$, solo puede haber un valor correspondiente de $B$, y para ese valor de $B$, solo puede haber un valor correspondiente de $C$. Más adelante, en la Sección 8.4.1, se estudia la manera de realizar esas inferencias.

| *edificio* | *número_aula* | *capacidad* |
|---|---|---|
| Packard | 101 | 500 |
| Painter | 514 | 10 |
| Taylor | 3128 | 70 |
| Watson | 100 | 30 |
| Watson | 120 | 50 |

**Figura 8.5.** Ejemplar de la relación *aula*.

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

### 8.3.2. Forma normal de Boyce–Codd

Una de las formas normales más deseables que se pueden obtener es la **forma normal de Boyce-Codd (FNBC)**. Elimina todas las redundancias que se pueden descubrir a partir de las dependencias funcionales aunque, como se verá en la Sección 8.6, puede que queden otros tipos de redundancia. Un esquema de relación $R$ está en la FNBC respecto al conjunto $F$ de dependencias funcionales si, para todas las dependencias funcionales de $F^+$ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple, al menos, una de las siguientes condiciones:

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

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

Ya hemos visto en la Sección 8.1 un ejemplo de esquema relacional que no está en FNBC:

*profesor_dept* (*ID, nombre, sueldo, nombre_dept, edificio, presupuesto*)

La dependencia funcional *nombre_dept* → *presupuesto* se cumple en *profesor_dept*, pero *nombre_dept* no es una superclave, ya que cada departamento puede tener diferente número de profesores. En la Sección 8.1.2, se vio que la descomposición de *profesor_dept* en *profesor* y *departamento* conduce a un mejor diseño. El esquema *profesor* está en FNBC. Todas las dependencias funcionales no triviales, como:

*ID* → *nombre, nombre_dept, sueldo*

incluyen *ID* en la parte izquierda de la flecha, e *ID* es una superclave (realmente, en este caso, la clave primaria) para *profesor*. En otras palabras, existe una dependencia funcional no trivial con cualquier combinación de *nombre*, *nombre_dept* y *sueldo*, sin *ID*, en un lado. Por tanto, *profesor* está en FNBC.

De forma similar, el esquema *departamento* está en FNBC ya que todas las dependencias funcionales no triviales que se cumplen, como por ejemplo:

*nombre_dept* → *edificio, presupuesto*

incluyen *nombre_dept* en la parte izquierda de la flecha, y *nombre_dept* es una superclave (y clave primaria) de *departamento*. Por tanto, departamento está en FNBC.

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

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

En el caso anterior de *profesor_dept*, $\alpha$ = *nombre_dept*, $\beta$ = {*edificio, presupuesto*}, y *profesor_dept*

se sustituye por:

- $(\alpha \cup \beta)$ = (*nombre_dept, edificio, presupuesto*)
- $(R - (\beta - \alpha))$ = (*ID, nombre, nombre_dept, sueldo*)

En este caso resulta que $\beta - \alpha = \beta$. Hay que definir la regla tal y como se ha hecho para que trate correctamente las dependencias funcionales que tienen atributos que aparecen a los dos lados de la flecha. Las razones técnicas para esto se tratarán más adelante, en la Sección 8.5.1.

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

### 8.3.3. FNBC y la conservación de las dependencias

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

Para ilustrar esto, supóngase que se lleva a cabo una modificación aparentemente pequeña en la organización de la universidad. En el diseño de la Figura 7.15 un estudiante solo puede tener un tutor. Esto se deduce del conjunto de relaciones *tutor* que es varios a uno de estudiante a *tutor*. El «pequeño» cambio que se va a hacer es que un profesor pueda estar asociado a un único departamento y un estudiante pueda tener más de un tutor, pero como mucho uno de un departamento dado.[3]

Una forma de implementar este cambio en el diseño E-R es sustituyendo la relación *tutor* por una relación ternaria, *tutor_dept*, que incluya tres conjuntos de entidades *profesor*, *estudiante* y *departamento* que es de varios a uno del par {*estudiante, profesor*} a *departamento*, como se muestra en la Figura 8.6. El diagrama E-R especifica la restricción de que «un estudiante puede tener más de un tutor, pero como mucho uno de un departamento dado».

3 Este tipo de relación tiene sentido en el caso de estudiantes que estudien dos carreras.

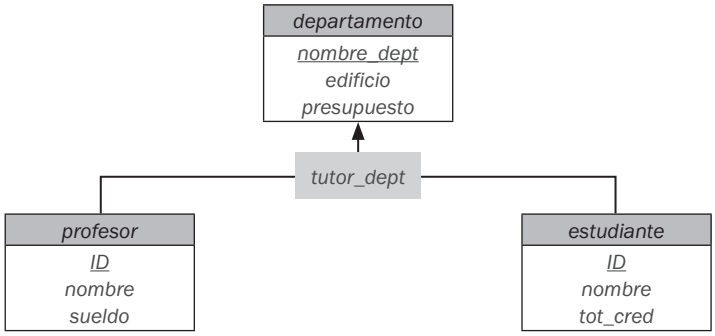



**Figura 8.6.** La relación *tutor_dept*.

Con este nuevo diagrama E-R, los esquemas de *profesor*, *departamento* y *estudiante* no cambian. Sin embargo, el esquema derivado de *tutor_dept* es ahora:

*tutor_dept* (*e_ID, p_ID, nombre_dept*)

Aunque no se especifique en el diagrama E-R, suponga que tenemos la restricción adicional de que «un profesor puede ser tutor solo en un único departamento».

Entonces, la siguiente dependencia funcional se cumple en *tutor_dept*:

$$p\_ID \rightarrow nombre\_dept$$
$$e\_ID, nombre\_dept \rightarrow p\_ID$$

La primera dependencia funcional se sigue de nuestro requisito de que «un profesor puede ser tutor solo en un departamento». La segunda dependencia funcional se sigue de nuestro requisito de que «un estudiante puede tener como mucho un tutor de un departamento dado».

Tenga en cuenta que con este diseño, estamos obligados a repetir el nombre del departamento cada vez que un profesor participa en una relación *tutor_dept*. Se puede observar que *tutor_dept* no se encuentra en FNBC ya que *p_ID* no es una superclave. Siguiendo nuestra regla para la descomposición FNBC, se obtiene:

(*e_ID, p_ID*)
(*p_ID, nombre_dept*)

Ambos esquemas están en FNBC. De hecho, se puede verificar que cualquier esquema con solo dos atributos está en FNBC por definición. Tenga en cuenta, sin embargo, que en nuestro diseño FNBC no existe ningún esquema que incluya todos los atributos que aparecen en la dependencia funcional *e_ID, nombre_dept* → *p_ID*.

Como nuestro diseño hace que sea muy difícil computacionalmente forzar esta dependencia funcional, se dice que nuestro diseño no **preserva la dependencia**.[4] Como la preservación de la dependencia se suele considerar deseable, consideraremos otra forma normal, más débil que la FNBC, que nos permita preservar dependencias. A esta forma normal se le llama tercera forma normal.[5]

## 8.3.4. Tercera forma normal

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

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

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

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

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

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

La definición de la 3FN permite ciertas dependencias funcionales que no se permiten en la FNBC. Las dependencias $\alpha \rightarrow \beta$ que solo satisfacen la tercera alternativa de la definición de la 3FN no se permiten en la FNBC, pero sí en la 3FN.[6]

Consideremos de nuevo la relación *tutor_dept*, que alberga las siguientes dependencias funcionales:

$$p\_ID \rightarrow nombre\_dept$$
$$e\_ID, nombre\_dept \rightarrow p\_ID$$

En la Sección 8.3.3 se trató que la dependencia funcional «*p_ID* → *nombre_dept*» hacía que el esquema *tutor_dept* no estuviese en FNBC. Tenga en cuenta que aquí $\alpha = p\_ID$, $\beta = nombre\_dept$, y $\beta - \alpha = nombre\_dept$. Como la dependencia funcional *e_ID, nombre_dept* →*p_ID* se cumple en *tutor_dept*, el atributo *nombre_dept* forma parte de una clave candidata y, por tanto, *tutor_dept* está en 3NF.

Se han abordado las contrapartidas que se pueden lograr entre FNBC y 3FN cuando el diseño FNBC no preserva dependencias. Estas contrapartidas se describen con más detalle en la Sección 8.5.4.

## 8.3.5. Formas normales superiores

Puede que en algunos casos el empleo de dependencias funcionales para la descomposición de los esquemas no sea suficiente para evitar la repetición innecesaria de información. Considérese una ligera variación de la definición del conjunto de entidades *profesor* en la que se registra con cada profesor un conjunto de nombres de niños y varios números de teléfono. Los números de teléfono pueden ser compartidos por varias personas. Entonces, *número_teléfono* y *nombre_niño* serán atributos multivalorados y, de acuerdo con las reglas para la generación de esquemas a partir de los diseños E-R, habrá dos esquemas, uno por cada uno de los atributos multivalorados *número_teléfono* y *nombre_niño*:

(*ID, nombre_niño*)
(*ID, número_teléfono*)

4 Técnicamente, es posible que una dependencia cuyos atributos no aparecen en ningún esquema se fuerce de forma implícita debido a la existencia de otras dependencias que la generan de forma lógica. Se tratará este caso más adelante en la Sección 8.4.5.

5 Se habrá dado cuenta de que nos hemos saltado la segunda forma normal. Solo tiene sentido desde un punto de vista histórico y en la práctica no se usa.

6 Estas dependencias son un ejemplo de **dependencias transitivas** (véase el Ejercicio práctico 8.16). La definición original de la 3FN se realizaba en términos de dependencias transitivas. La definición que utilizamos es equivalente pero más sencilla de entender.

Si se combinan estos dos esquemas para obtener:

(*ID, nombre_niño, número_teléfono*)

se descubre que el resultado se halla en la FNBC, ya que solo se cumplen dependencias funcionales no triviales. En consecuencia, se puede pensar que ese tipo de combinación es una buena idea. Sin embargo, se trata de una mala idea, como puede verse si se considera el ejemplo de un profesor con dos niños y dos números de teléfono. Por ejemplo, sea el profesor con ID 99999 que tiene dos niños llamados «David» y «William» y dos números de teléfono, 512-555-123 y 512-555-432. En el esquema combinado hay que repetir los números de teléfono una vez por cada dependiente:

(99999, David, 512-555-1234)
(99999, David, 512-555-4321)
(99999, William, 512-555-1234)
(99999, William, 512-555-4321)

Si no se repitieran los números de teléfono y solo se almacenaran la primera y la última tupla, se habrían guardado los nombres de los niños y los números de teléfono, pero las tuplas resultantes implicarían que David correspondería al 512-555-123 y que William correspondería al 512-555-432. Como sabemos, esto es incorrecto.

Debido a que las formas normales basadas en las dependencias funcionales no son suficientes para tratar con situaciones como esta, se han definido otras dependencias y formas normales. Se estudian en las Secciones 8.6 y 8.7.

## 8.4. Teoría de las dependencias funcionales

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

### 8.4.1. Cierre de los conjuntos de dependencias funcionales

Se verá que, dado un conjunto $F$ de dependencias funcionales, se puede probar que también se cumple alguna otra dependencia funcional. Se dice que $F$ «implica lógicamente» esas dependencias funcionales. Cuando se verifican las formas normales, no es suficiente considerar el conjunto dado de dependencias funcionales. También hay que considerar *todas* las dependencias funcionales que se cumplen en el esquema.

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

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

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

La dependencia funcional:

$$A \rightarrow H$$

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

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

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

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

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

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

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

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

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

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

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

Los axiomas de Armstrong son **correctos,** ya que no generan dependencias funcionales incorrectas. Son **completos**, ya que, para un conjunto de dependencias funcionales $F$ dado, permiten generar todo $F^+$. Las notas bibliográficas proporcionan referencias de las pruebas de su corrección y de su completitud.

Aunque los axiomas de Armstrong son completos, resulta pesado utilizarlos directamente para el cálculo de $F^+$. Para simplificar más las cosas se dan unas reglas adicionales. Se pueden utilizar los axiomas de Armstrong para probar que son correctas (véanse los Ejercicios prácticos 8.4, 8.5 y 8.26).

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

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

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

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

La Figura 8.7 muestra un procedimiento que demuestra formalmente el modo de utilizar los axiomas de Armstrong para calcular $F^+$. En este procedimiento puede que la dependencia funcional ya esté presente en $F^+$ cuando se le añade; en ese caso, no hay ninguna modificación en $F^+$. En la Sección 8.4.2 se verá una manera alternativa de calcular $F^+$.

Los términos a la derecha y a la izquierda de las dependencias funcionales son subconjuntos de $R$. Dado que un conjunto de tamaño $n$ tiene $2^n$ subconjuntos, hay un total de $2^n$ 3 $2^n = 2^{2n}$ dependencias funcionales posibles, donde $n$ es el número de atributos de $R$. Cada iteración del bucle del procedimiento repetido, salvo la última, añade como mínimo una dependencia funcional a $F^+$. Por tanto, está garantizado que el procedimiento termine.

## 8.4.2. Cierre del conjunto de atributos

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

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

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

**Figura 8.7.** Procedimiento para calcular $F^+$.

```
resultado := α;
repeat
   for each dependencia funcional β → γ in F do
      begin
         if β ⊆ resultado then resultado:= resultado ∪ γ;
      end
until (ya no cambie resultado)
```

**Figura 8.8.** Algoritmo para el cálculo de $\alpha^+$, el cierre de $\alpha$ bajo $F$.

Sea $\alpha$ un conjunto de atributos. Al conjunto de todos los atributos determinados funcionalmente por $\alpha$ bajo un conjunto $F$ de dependencias funcionales se le denomina **cierre** de $\alpha$ bajo $F$; se denota mediante $\alpha^+$. La Figura 8.8 muestra un algoritmo, escrito en pseudocódigo, para calcular $\alpha^+$. La entrada es un conjunto $F$ de dependencias funcionales y el conjunto $\alpha$ de atributos. La salida se almacena en la variable *resultado*.

Para ilustrar el modo en que trabaja el algoritmo, se utilizará para calcular $(AG)^+$ con las dependencias funcionales definidas en la Sección 8.4.1. Se comienza con *resultado* = $AG$. La primera vez que se ejecuta el bucle **repeat** para comprobar cada dependencia funcional se halla que:

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

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

Veamos ahora el motivo de que el algoritmo de la Figura 8.8 sea correcto. El primer paso es correcto, ya que $\alpha \rightarrow \alpha$ se cumple siempre (por la regla de reflexividad). Se afirma que, para cualquier subconjunto $\beta$ de *resultado*, $\alpha \rightarrow \beta$. Dado que el bucle **repeat** se inicia con $\alpha \rightarrow$ *resultado* como cierto, solo se puede añadir $\gamma$ a *resultado* si $\beta \subseteq$ *resultado* y $\beta \rightarrow \gamma$. Pero, entonces, *resultado* $\rightarrow \beta$ por la regla de reflexividad, por lo que $\alpha \rightarrow \beta$ por transitividad. Otra aplicación de la transitividad demuestra que $\alpha \rightarrow \gamma$ (empleando $\alpha \rightarrow \beta$ y $\beta \rightarrow \gamma$). La regla de la unión implica que $\alpha \rightarrow$ *resultado* $\cup \gamma$, por lo que $\alpha$ determina funcionalmente cualquier resultado nuevo generado en el bucle **repeat.** Por tanto, cualquier atributo devuelto por el algoritmo se halla en $\alpha^+$.

Resulta sencillo ver que el algoritmo halla todo $\alpha^+$. Si hay un atributo de $\alpha^+$ que no se halle todavía en *resultado* en cualquier momento de la ejecución, debe haber una dependencia funcional $\beta \rightarrow \gamma$ para la que $\beta \subseteq$ resultado y, como mínimo, un atributo de $\gamma$ no se halla en resultado. Cuando el algoritmo termina, se han procesado todas las dependencias funcionales, y se han añadido a *resultado* los atributos de $\gamma$; entonces podemos estar seguros de que todos los atributos de $\alpha^+$ se encuentran en *resultado*.

En el peor de los casos, es posible que este algoritmo tarde un tiempo proporcional al cuadrado del tamaño de $F$. Hay un algoritmo más rápido (aunque ligeramente más complejo) que se ejecuta en un tiempo proporcional al tamaño de $F$; ese algoritmo se presenta como parte del Ejercicio práctico 8.8.

Existen varias aplicaciones del algoritmo de cierre de atributos:

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

## 8.4.3. Recubrimiento canónico

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

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

Se puede reducir el esfuerzo dedicado a la comprobación de las violaciones comprobando un conjunto simplificado de dependencias funcionales que tenga el mismo cierre que el conjunto dado. Cualquier base de datos que satisfaga el conjunto simplificado de dependencias funcionales satisfará también el conjunto original y viceversa, ya que los dos conjuntos tienen el mismo cierre. Sin embargo, el conjunto simplificado resulta más sencillo de comprobar.

En breve se verá el modo en que se puede crear ese conjunto simplificado. Antes, hacen falta algunas definiciones.

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

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

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

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

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

**Figura 8.9.** Cálculo del recubrimiento canónico.

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

- Si $A \in \beta$, para comprobar si $A$ es raro hay que considerar el conjunto:

  $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$$

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

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

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

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

El recubrimiento canónico del conjunto de dependencias funcionales $F$ puede calcularse como se muestra en la Figura 8.9. Es importante destacar que, cuando se comprueba si un atributo es raro, la comprobación utiliza las dependencias del valor actual de $F_c$, y **no** las dependencias de $F$. Si una dependencia funcional solo contiene un atributo en su lado derecho, por ejemplo, $A \rightarrow C$, y se descubre que ese atributo es raro, se obtiene una dependencia funcional con el lado derecho vacío. Hay que eliminar esas dependencias funcionales.

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

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

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

Calcúlese el recubrimiento canónico de $F$.

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

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

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

  Por tanto, su recubrimiento canónico es:

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

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

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

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

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

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

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

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

### 8.4.4. Descomposición sin pérdidas

Sea $r(R)$ un esquema de relación y $F$ un conjunto de dependencias funcionales de $r(R)$. Supóngase que $R_1$ y $R_2$ forman una descomposición de $R$. Se dice que la descomposición es una **descomposición sin pérdidas** si no hay pérdida de información al sustituir $r(R)$ por dos esquemas de relación $r_1(R_1)$ y $r_2(R_2)$. De forma más precisa, se dice que la descomposición es sin pérdidas si para todos los ejemplares legales de la base de datos (es decir, los ejemplares de la base de datos que satisfacen las dependencias funcionales especificadas y otras restricciones), la relación $r$ contiene el mismo conjunto de tuplas como resultado de la siguiente consulta SQL:

**select** *
**from** (**select** $R_1$ **from** $r$)
  **natural join**
  (**select** $R_2$ **from** $r$)

Lo que se puede expresar de forma más sucinta en álgebra relacional como:

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

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

A modo de ejemplo de una descomposición con pérdidas, recuerde la descomposición del esquema *empleado* en:

*empleado1* (*ID, nombre*)
*empleado2* (*nombre, calle, ciudad, sueldo*)

que ya se vio en la Sección 8.1.2. Como se ha visto en la Figura 8.3, el resultado de *empleado1* ⋈ *empleado2* es un superconjunto de la relación original *empleado*, pero la descomposición es sin pérdidas, pues el resultado de la unión ha perdido información sobre qué identificador de empleado corresponde con qué direcciones y sueldos, en el caso de que dos o más empleados tengan el mismo nombre.

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

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

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

Para ilustrar esto, considérese el esquema

*profesor_dept* (*ID, nombre, sueldo, nombre_dept, edificio, presupuesto*)

cuya descomposición se hizo en la Sección 8.1.2 en los esquemas *profesor* y *departamento*:

*profesor* (*ID, nombre, nombre_dept, sueldo*)
*departamento* (*nombre_dept, edificio, presupuesto*)

Considérese la intersección de estos dos esquemas, que es *nombre_dept*. Lo vemos porque *nombre_dept* → *nombre_dept, edificio, presupuesto*, satisface la regla de descomposición sin pérdidas.

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

Mientras que la comprobación de la descomposición binaria es, claramente, una condición suficiente para la descomposición sin pérdidas, solo se trata de una condición necesaria si todas las restricciones son dependencias funcionales. Más adelante se verán otros tipos de restricciones (especialmente, un tipo de restricción denominada dependencia multivalorada, que se estudia en la Sección 8.6.1), que pueden garantizar que una descomposición sea sin pérdidas aunque no esté presente ninguna dependencia funcional.

### 8.4.5. Conservación de las dependencias

Resulta más sencillo caracterizar la conservación de las dependencias empleando la teoría de las dependencias funcionales que mediante el enfoque ad hoc que se utilizó en la Sección 8.3.3.

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

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

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

La Figura 8.10 muestra un algoritmo para la comprobación de la conservación de las dependencias. La entrada es el conjunto $D = \{R_1, R_2, \ldots, R_n\}$ de esquemas de relaciones descompuestas y el conjunto $F$ de dependencias funcionales. Este algoritmo resulta costoso, ya que exige el cálculo de $F^+$. En lugar de aplicar el algoritmo de la Figura 8.10, se consideran dos alternativas.

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

**Figura 8.10.** Comprobación de la conservación de las dependencias.

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

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

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

$resultado = \alpha$
**repeat**
  **for each** $R_i$ de la descomposición
    $t = (resultado \cap R_i)^+ \cap R_i$
    $resultado = resultado \cup t$
**until** (no haya cambios en *resultado*)

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

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

- La primera idea es comprobar cada dependencia funcional $\alpha \rightarrow \beta$ de $F$ para ver si se conserva en $F'$ (donde $F'$ es tal y como se define en la Figura 8.10). Para ello, se calcula el cierre de $\alpha$ bajo $F'$; la dependencia se conserva exactamente cuando el cierre incluye a $\beta$. La descomposición conserva la dependencia si, y solo si, se comprueba que se conservan todas las dependencias de $F$.
- La segunda idea es emplear una forma modificada del algoritmo de cierre de atributos para calcular el cierre bajo $F'$, sin llegar a calcular antes $F'$. Se desea evitar el cálculo de $F'$, ya que resulta bastante costoso. Téngase en cuenta que $F'$ es la unión de las $F_i$, donde $F_i$ es la restricción de $F$ sobre $R_i$. El algoritmo calcula el cierre de los atributos de $(resultado \cap R_i)$ con respecto a $F$, intersecta el cierre con $R_i$ y añade el conjunto de atributos resultante a *resultado*; esta secuencia de pasos es equivalente al cálculo del cierre de resultado bajo $F_i$. La repetición de este paso para cada $i$ del interior del bucle genera el cierre de resultado bajo $F'$.

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

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

## 8.5. Algoritmos de descomposición

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

### 8.5.1. Descomposición en la FNBC

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

#### 8.5.1.1. Comprobación de la FNBC

La comprobación de un esquema de relación $R$ para ver si satisface la FNBC se puede simplificar en algunos casos:

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

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

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

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

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

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

**Figura 8.11.** Algoritmo de descomposición en la FNBC.

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

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

La dependencia anterior muestra que $R_i$ viola la FNBC.

#### 8.5.1.2. Algoritmo de descomposición de la FNBC

Ahora se puede exponer un método general para descomponer los esquemas de relación de manera que satisfagan la FNBC. La Figura 8.11 muestra un algoritmo para esta tarea. Si $R$ no está en la FNBC, se puede descomponer $R$ en un conjunto de esquemas en la FNBC, $R_1, R_2 \ldots, R_n$ utilizando este algoritmo. El algoritmo utiliza las dependencias que demuestran la violación de la FNBC para llevar a cabo la descomposición.

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

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

Es fácil ver que la descomposición de *profesor_dept* de la Sección 8.3.2 se obtiene de la aplicación del algoritmo. La dependencia funcional *nombre_dept* → *edificio*, *presupuesto* satisface la condición $\alpha \cap \beta = \varnothing$ y, por tanto, se elige para descomponer el esquema.

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

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

*clase* (*asignatura_id, nombre_asig, nombre_dept, créditos, secc_id, semestre, año, edificio, número_aula, capacidad, franja_horaria_id*)

El conjunto de dependencias funcionales a exigir que se cumplan en *clase* es:

*asignatura_id* → *nombre_asig, nombre_dept, créditos*
*edificio, número_aula* → *capacidad asignatura_id, secc_id,*
*semestre, año* → *edificio, número_aula, franja_horaria_id*

Una clave candidata para este esquema es {*asignatura_id, secc_id, semestre, año*}.

Se puede aplicar el algoritmo de la Figura 8.11 al ejemplo *clase* de la manera siguiente:

- La dependencia funcional:

  *asignatura_id* → *nombre_asig, nombre_dept, créditos*

  se cumple, pero *asignatura_id* no es superclave. Por tanto, *clase* no se halla en FNBC. Se sustituye *clase* por:

  *asignatura* (*asignatura_id, nombre_asig, nombre_dept, créditos*)
  *clase-1* (*asignatura_id, secc_id, semestre, año, edificio,*
  *número_aula capacidad, franja_horaria_id*)

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

- Una clave candidata para *clase-1* es {*asignatura_id, secc_id, semestre, año*}. La dependencia funcional:

  *edificio, número_aula* → *capacidad*

  se cumple en *clase-1*, pero {*edificio, número_aula*} no es superclave de *clase-1*. Se sustituye *clase-1* por:

  *aula* (*edificio, número_aula, capacidad*)
  *sección* (*asignatura_id, secc_id, semestre, año,*
  *edificio, número_aula, franja_horaria_id*)

*aula* y *sección* se hallan en FNBC.

Por tanto, la descomposición de *clase* da lugar a los tres esquemas de relación *asignatura*, *aula* y *sección*, cada uno de los cuales se halla en la FNBC. Se corresponden con esquemas ya usados en este y en capítulos anteriores. Se puede comprobar que la descomposición es sin pérdidas y conserva las dependencias.

```
sea F_c un recubrimiento canónico de F;
i := 0;
for each dependencia funcional α → β de F_c
   i := i + 1;
   R_i := α β;
if ninguno de los esquemas R_j, j = 1, 2... , i contiene una clave candidata de R
   then
      i := i + 1;
      R_i := cualquier clave candidata de R;
/* Opcionalmente, eliminar las relaciones redundantes */
repeat
   if algún esquema R_j se encuentra contenido en otro esquema R_k
      then
         /* Eliminar R_j */
         R_j := R_i;
         i: = i − 1;
until no se puedan eliminar más R_j
return (R_1, R_2... , R_i)
```

**Figura 8.12.** Comprobación de la conservación de las dependencias.

### 8.5.2. Descomposición en la 3FN

La Figura 8.12 muestra un algoritmo para la búsqueda de descomposiciones en la 3FN sin pérdidas y que conserven las dependencias. El conjunto de dependencias $F_c$ utilizado en el algoritmo es un recubrimiento canónico de $F$. Obsérvese que el algoritmo considera el conjunto de esquemas $R_j, j = 1, 2 \ldots, i$; inicialmente $i = 0$ y, en ese caso, el conjunto está vacío.

Se aplicará este algoritmo al ejemplo de la Sección 8.3.4 en el que se demostró que:

*tutor_dept* (*e_ID, p_ID, nombre_dept*)

se halla en la 3FN, aunque no se halle en la FNBC. El algoritmo utiliza las dependencias funcionales de $F$:

$f_1$: *p_ID* → *nombre_dept*
$f_2$: *e_ID, nombre_dept* → *p_ID*

No existen atributos extraños en ninguna de las dependencias funcionales de $F$, por lo que $F_c$ contiene $f_1$ y $f_2$. El algoritmo genera como $R_1$ el esquema (*p_ID nombre_dept*), y como $R_2$ el esquema (*e_ID, nombre_dept, p_ID*). El algoritmo, entonces, descubre que $R_2$ contiene una clave candidata, por lo que no se crean más esquemas de relación.

El conjunto de esquemas que resulta puede contener esquemas redundantes, sin un esquema $R_k$ que contenga todos los atributos de otro esquema $R_j$. Por ejemplo, el esquema $R_2$ anterior contiene todos los atributos de $R_1$. El algoritmo elimina todos estos esquemas que ya están contenidos en otro esquema. Cualquier dependencia que se pueda comprobar en $R_j$ y sea eliminada también se puede comprobar en la correspondiente relación $R_k$, y la descomposición es sin pérdidas aunque se elimine $R_j$.

Ahora, consideremos de nuevo el esquema *clase* de la Sección 8.5.1.2 y apliquemos el algoritmo de descomposición de 3FN. El conjunto de dependencias funcionales que se listan resultan ser un recubrimiento canónico. Como resultado, el algoritmo nos da los mismos tres esquemas *asignatura*, *aula* y *sección*.

El ejemplo anterior muestra una propiedad interesante del algoritmo de 3FN. A veces, el resultado no solo está en 3FN, sino también en FNBC. Ello sugiere un método alternativo de generar un diseño en FNBC. En primer lugar se utiliza el algoritmo de 3FN. Después, para los esquemas del diseño en 3FN que no estén en FNBC, se descomponen utilizando el algoritmo de FNBC. Si el resultado no preserva las dependencias, se vuelve al diseño en 3FN.

### 8.5.3. Corrección del algoritmo de 3FN

El algoritmo garantiza la conservación de las dependencias mediante la creación explícita de un esquema para cada dependencia del recubrimiento canónico. Asegura que la descomposición sea sin pérdidas al garantizar que, como mínimo, un esquema contenga una clave candidata del esquema que se está descomponiendo. El Ejercicio práctico 8.15 proporciona algunos indicios de la prueba de que esto basta para garantizar una descomposición sin pérdidas.

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

Si una relación $R_i$ está en la descomposición generada por el algoritmo de síntesis, entonces $R_i$ está en la 3FN. Recuérdese que, cuando se busca la 3FN, basta con considerar las dependencias funcionales cuyo lado derecho sea un solo atributo. Por tanto, para ver si $R_i$ está en la 3FN, hay que convencerse de que cualquier dependencia funcional $\gamma \rightarrow B$ que se cumpla en $R_i$ satisface la definición de la 3FN. Supóngase que la dependencia que generó $R_i$ en el algoritmo de síntesis es $\alpha \rightarrow \beta$. Ahora bien, $B$ debe estar en $\alpha$ o en $\beta$, ya que $B$ está en $R_i$ y $\alpha \rightarrow \beta$ ha generado $R_i$. Considérense los tres casos posibles:

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

  Como $\alpha$ es clave candidata, se satisface la tercera alternativa de la definición de la 3FN.

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

### 8.5.4. Comparación entre la FNBC y la 3FN

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

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

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

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

Merece la pena destacar que el SQL no ofrece una manera de especificar las dependencias funcionales, salvo para el caso especial de la declaración de las superclaves mediante las restricciones **primary key** o **unique.** Es posible, aunque un poco complicado, escribir asertos que hagan que se cumpla una dependencia funcional determinada (véase el Ejercicio práctico 8.9); por desgracia, no existe actualmente ningún sistema de base de datos que disponga de los complejos asertos que serían necesarios para forzar las dependencias funcionales, y la comprobación de los asertos resultaría muy costosa. Por tanto, aunque se tenga una descomposición que conserve las dependencias, si se utiliza el SQL estándar, solo se podrán comprobar de manera eficiente las dependencias funcionales cuyo lado izquierdo sea una clave.

Aunque puede que la comprobación de las dependencias funcionales implique una operación reunión si la descomposición no conserva las dependencias, se puede reducir su coste empleando vistas materializadas, utilizables en la mayor parte de los sistemas de bases de datos. Dada una descomposición en la FNBC que no conserve las dependencias, se considera cada dependencia en un recubrimiento canónico $F_c$ que no se conserve en la descomposición. Para cada una de esas dependencias $\alpha \rightarrow \beta$, se define una vista materializada que calcule una reunión de todas las relaciones de la descomposición y proyecte el resultado sobre $\alpha\beta$. La dependencia funcional puede comprobarse fácilmente en la vista materializada mediante una restricción **unique** $(\alpha)$ o **primary key** $(\alpha)$.

La parte negativa es que hay una sobrecarga espacial y temporal debida a la vista materializada, pero la positiva es que el programador de la aplicación no tiene que preocuparse de escribir código para mantener los datos redundantes consistentes en las actualizaciones; es labor del sistema de bases de datos conservar la vista materializada, es decir, mantenerla actualizada cuando se actualice la base de datos. (Más adelante, en la Sección 13.5, se describe el modo en que el sistema de bases de datos puede llevar a cabo de manera eficiente el mantenimiento de las vistas materializadas.)

Por desgracia, la mayoría de los sistemas de bases de datos no disponen de restricciones en vistas materializadas. Aunque la base de datos Oracle dispone de restricciones en vistas materializadas, por defecto realiza un mantenimiento de vistas cuando se accede a la vista, no cuando se actualiza la relación subyacente;[7] el resultado es que una violación de restricción se detecta después de realizarse la actualización, lo que hace inútil su detección.

Por tanto, en caso de que no se pueda obtener una descomposición en la FNBC que conserve las dependencias, suele resultar preferible optar por la FNBC, ya que la comprobación de dependencias funcionales distintas a las de restricciones de clave primaria es difícil en SQL.

7 Al menos para la versión de Oracle de 10g.

## 8.6. Descomposición mediante dependencias multivaloradas

No parece que algunos esquemas de relación, aunque se hallen en la FNBC, estén suficientemente normalizados, en el sentido de que siguen sufriendo el problema de la repetición de información. Considérese nuevamente el ejemplo de la universidad en el que un profesor pueda estar asociado a varios departamentos:

*prof* (*ID, nombre_dept, nombre, calle, ciudad*)

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

$$ID \rightarrow nombre, calle, ciudad$$

y a que *ID* no es clave de *prof*.

No obstante, supóngase que un profesor pueda tener varios domicilios (por ejemplo, una residencia de invierno y otra de verano). Entonces, ya no se desea que se cumpla la dependencia funcional «*ID* → *calle, ciudad*», aunque, por supuesto, se sigue deseando que se cumpla «*ID* → *nombre*» (es decir, la universidad no trata con profesores que operan con varios alias). A partir del algoritmo de descomposición en la FNBC se obtienen dos esquemas:

$r_1$ (*ID, nombre*)
$r_2$ (*ID, nombre_dept, calle, ciudad*)

Los dos se encuentran en la FNBC (recuérdese que cada profesor puede estar asociado a varios departamentos y un departamento tener varios profesores y, por tanto, no se cumple ni «*ID* → *nombre_dept*» ni «*nombre_dept* → *ID*»).

Pese a que $r_2$ se halla en la FNBC, hay redundancia. Se repite la dirección de cada residencia para cada profesor una vez por cada departamento que tenga asociado a ese profesor. Este problema se puede resolver descomponiendo más aún $r_2$ en:

$r_{21}$ (*nombre_dept, ID*)
$r_{22}$ (*ID, calle, ciudad*)

pero no hay ninguna restricción que nos lleve a hacerlo.

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

### 8.6.1. Dependencias multivaloradas

Las dependencias funcionales impiden que ciertas tuplas estén en una relación dada. Si $A \rightarrow B$, entonces no puede haber dos tuplas con el mismo valor de *A* y diferentes valores de *B*. Las dependencias multivaloradas, por otro lado, no impiden la existencia de esas tuplas. Por el contrario, *exigen* que otras tuplas estén presentes en la relación de una forma determinada. Por este motivo, las dependencias funcionales a veces se denominan **dependencias que generan igualdades;** y las dependencias multivaloradas, **dependencias que generan tuplas.**

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

$$\alpha \twoheadrightarrow \beta$$

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

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

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

**Figura 8.13.** Representación tabular de $\alpha \twoheadrightarrow \beta$.

Esta definición es menos complicada de lo que parece. La Figura 8.13 muestra una representación tabular de $t_1$, $t_2$, $t_3$ y $t_4$. De manera intuitiva, la dependencia multivalorada $\alpha \twoheadrightarrow \beta$ indica que la relación entre $\alpha$ y $\beta$ es independiente de la relación entre $\alpha$ y $R - \beta$. Si todas las relaciones del esquema $R$ satisfacen la dependencia multivalorada $\alpha \twoheadrightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$ es una dependencia multivalorada *trivial* del esquema $R$. Por tanto, $\alpha \twoheadrightarrow \beta$ es trivial si $\beta \subseteq \alpha$ o $\beta \cup \alpha = R$.

Para ilustrar la diferencia entre las dependencias funcionales y las multivaloradas, considérense de nuevo el esquema $r_2$ y la relación de ejemplo de ese esquema mostrada en la Figura 8.14. Hay que repetir el nombre de departamento una vez por cada dirección que tenga el profesor, y hay que repetir la dirección de cada departamento al que esté asociado un profesor. Esta repetición es innecesaria, ya que la relación entre cada profesor y su dirección es independiente de la relación entre ese profesor y el departamento. Si un profesor con identificador *ID* 99123 está asociado al departamento de Física, se desea que ese departamento esté asociado con todas las direcciones de ese profesor. Por tanto, la relación de la Figura 8.15 es ilegal. Para hacer que esa relación sea legal hay que añadir las tuplas (Física, 22222, Main, Manchester) y (Matemáticas, 22222, North, Rye) a la relación de la Figura 8.15.

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

$$ID \twoheadrightarrow calle, ciudad$$

También se cumplirá la dependencia multivalorada *ID* →→ *nombre_dept*. Pronto se verá que son equivalentes.

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

1. Para verificar las relaciones y determinar si son legales bajo un conjunto dado de dependencias funcionales y multivaloradas.

| *ID* | *nombre_dept* | *calle* | *ciudad* |
|---|---|---|---|
| 22222 | Física | North | Rye |
| 22222 | Física | Main | Manchester |
| 12121 | Finanzas | Lake | Horseneck |

**Figura 8.14.** Ejemplo de redundancia en una relación de un esquema en FNBC.

| *ID* | *nombre_dept* | *calle* | *ciudad* |
|---|---|---|---|
| 22222 | Física | North | Rye |
| 22222 | Matemáticas | Main | Manchester |

**Figura 8.15.** Relación $r_2$ ilegal.

2. Para especificar restricciones del conjunto de relaciones legales; de este modo, *solo* habrá que preocuparse de las relaciones que satisfagan un conjunto dado de dependencias funcionales y multivaloradas.

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

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

A partir de la definición de dependencia multivalorada se pueden obtener las reglas siguientes para $\alpha, \beta \subseteq R$:

- Si $\alpha \rightarrow \beta$, entonces $\alpha \twoheadrightarrow \beta$. En otras palabras, toda dependencia funcional es también una dependencia multivalorada.
- Si $\alpha \twoheadrightarrow \beta$, entonces $\alpha \twoheadrightarrow R - \alpha - \beta$

En el Apéndice B.1.1 se describe un sistema de reglas de inferencia para dependencias multivaloradas.

### 8.6.2. Cuarta forma normal

Considérese nuevamente el ejemplo del esquema en la FNBC:

$$r_2\ (ID,\ nombre\_dept,\ calle,\ ciudad)$$

en el que se cumple la dependencia multivalorada «$ID \twoheadrightarrow calle, ciudad$». Se vio en los primeros párrafos de la Sección 8.6 que, aunque este esquema se halla en la FNBC, el diseño no es el ideal, ya que hay que repetir la información sobre la dirección del profesor para cada departamento. Se verá que se puede utilizar esta dependencia multivalorada para mejorar el diseño de la base de datos, realizando la descomposición del esquema en la **cuarta forma normal.**

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

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

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

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

Sea $r\ (R)$ un esquema de relación y sean $r_1\ (R_1), r_2\ (R_2) \ldots, r_n(R_n)$ una descomposición de $r\ (R)$. Para comprobar si cada esquema de relación $r_i$ de la descomposición se halla en la 4FN hay que averiguar las dependencias multivaloradas que se cumplen en cada $r_i$. Recuérdese que, para un conjunto $F$ de dependencias funcionales, la restricción $F_i$ de $F$ a $R_i$ son todas las dependencias funcionales de $F^+$ que *solo* incluyen los atributos de $R_i$. Considérese ahora un conjunto $D$ de dependencias funcionales y multivaloradas. La **restricción** de $D$ a $R_i$ es el conjunto $D_i$ consistente en:

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

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

donde $\alpha \subseteq R_i$ y $\alpha \twoheadrightarrow \beta$ están en $D^+$.

### 8.6.3. Descomposición en la 4NF

La analogía entre la 4FN y la FNBC es aplicable al algoritmo para descomponer esquemas en la 4FN. La Figura 8.16 muestra el algoritmo de descomposición en la 4FN. Es idéntico al algoritmo de descomposición en la FNBC de la Figura 8.11, salvo porque emplea dependencias multivaloradas en lugar de funcionales y utiliza la restricción de $D^+$ a $R_i$.

Si se aplica el algoritmo de la Figura 8.16 a (*ID, nombre_dept, calle, ciudad*), se descubre que $ID \twoheadrightarrow nombre\_dept$ es una dependencia multivalorada no trivial, y que *ID* no es superclave del esquema. De acuerdo con el algoritmo, se sustituye el esquema original por estos dos esquemas:

$$r_{21}\ (ID,\ nombre\_dept)$$
$$r_{22}\ (ID,\ calle,\ ciudad)$$

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

Como ocurría cuando se trataba solamente con las dependencias funcionales, resultan interesantes las descomposiciones que carecen de pérdidas y que conservan las dependencias. La siguiente circunstancia relativa a las dependencias multivaloradas y a la ausencia de pérdidas muestra que el algoritmo de la Figura 8.16 solo genera descomposiciones sin pérdidas:

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

**Figura 8.16.** Algoritmo de descomposición en 4FN.

- Sea $r\ (R)$ un esquema de relación, y $D$ un conjunto de dependencias funcionales y multivaloradas en $R$. Sean $r_1\ (R_1)$ y $r_2\ (R_2)$ una descomposición de $R$. Esta descomposición es sin pérdidas en $R$, si y solo si al menos una de las dependencias multivaloradas está en $D^+$:

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

Recuerde que ya se indicó en la Sección 8.4.4 que si $R_1 \cap R_2 \rightarrow R_1$ o $R_1 \cap R_2 \rightarrow R_2$, entonces $r_1\ (R_1)$ y $r_2\ (R_2)$ son descomposiciones sin pérdidas $r\ (R)$. Este hecho sobre dependencias multivaloradas es una afirmación más general sobre la característica sin pérdidas. Indica que para *todas* las descomposiciones sin pérdidas de $r\ (R)$ en dos esquemas $r_1\ (R_1)$ y $r_2\ (R_2)$, debe cumplirse una de las dos dependencias $R_1 \cap R_2 \twoheadrightarrow R_1$ o $R_1 \cap R_2 \twoheadrightarrow R_2$.

El tema de la conservación de la dependencia cuando se descompone un esquema de relación se vuelve más complicado en presencia de dependencias multivaloradas. En el Apéndice B.1.2 se trata este tema.

## 8.7. Más formas normales

La cuarta forma normal no es, de ningún modo, la forma normal «definitiva». Como ya se ha visto, las dependencias multivaloradas ayudan a comprender y a abordar algunas formas de repetición de la información que no pueden comprenderse en términos de las dependencias funcionales. Hay restricciones denominadas **depen-**

**dencias de reunión** que generalizan las dependencias multivaloradas y llevan a otra forma normal denominada **forma normal de reunión por proyección (FNRP)** (la FNRP se denomina en algunos libros **quinta forma normal**). Hay una clase de restricciones todavía más generales, que lleva a una forma normal denominada **forma normal de dominios y claves (FNDC).**

Un problema práctico del empleo de estas restricciones generalizadas es que no solo es difícil razonar con ellas, sino que tampoco hay un conjunto de reglas de inferencia seguras y completas para razonar sobre las restricciones. Por tanto, FNRP y FNDC se utilizan muy rara vez. El Apéndice B (B.2) ofrece más detalles sobre estas formas normales.

Destaca por su ausencia en este estudio de las formas normales la **segunda forma normal** (2FN). No se ha estudiado porque solo es de interés histórico. Simplemente se definirá, y se permitirá al lector experimentar con ella, en el Ejercicio práctico 8.17.

## 8.8. Proceso de diseño de la base de datos

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

Anteriormente, en este capítulo, a partir de la Sección 8.3, se ha dado por supuesto que se tenía un esquema de relación $r(R)$ y que se procedía a normalizarlo. Hay varios modos de obtener ese esquema $r(R)$:

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

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

### 8.8.1. El modelo E-R y la normalización

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

La mayor parte de los ejemplos de este tipo de dependencias surge de un mal diseño del diagrama E-R. En el ejemplo anterior, si se hubiera diseñado correctamente el diagrama E-R, se habría creado una entidad *departamento* con el atributo *dirección_departamento* y una relación entre *profesor* y *departamento*. De manera parecida, puede que una relación que implique a más de dos entidades no se halle en la forma normal deseable. Como la mayor parte de las relaciones son binarias, estos casos resultan relativamente raros (de hecho, algunas variantes de los diagramas E-R hacen realmente difícil o imposible especificar relaciones no binarias).

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

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

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

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

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

### 8.8.2. Denominación de los atributos y las relaciones

Una característica deseable del diseño de bases de datos es la **asunción de un rol único,** lo que significa que cada nombre de atributo tiene un significado único en toda la base de datos. Esto evita que se utilice el mismo atributo para indicar cosas diferentes en esquemas diferentes. Por ejemplo, puede que, de otra manera, se considerara el uso del atributo *número* para el número de teléfono en el esquema *profesor* y para el número de aula en el esquema *aula*. La reunión de una relación del esquema *profesor* con otra de *aula* carecería de significado. Aunque los usuarios y los desarrolladores de aplicaciones pueden trabajar con esmero para garantizar el uso apropiado de *número* en cada circunstancia, tener nombres de atributo diferentes para el número de teléfono y para el de aula sirve para reducir los errores de los usuarios.

Es buena idea hacer que los nombres de los atributos incompatibles sean diferentes, pero si los atributos de relaciones diferentes tienen el mismo significado, puede ser conveniente emplear el mismo nombre de atributo. Por ejemplo, se ha utilizado el nombre de atributo «*nombre*» para los conjuntos de entidades *profesor* y *estudiante*. Si no fuese así, es decir, si hubiésemos convenido una denominación diferente para los nombres de profesor y estudiante, entonces cuando deseáramos generalizar esos conjuntos de entidades mediante la creación del conjunto de entidades *persona*, habría que renombrar el atributo. Por tanto, aunque no se tenga actualmente una generalización de *estudiante* y de *profesor*, si se prevé esa posibilidad es mejor emplear el mismo nombre en los dos conjuntos de relaciones (y en sus relaciones).

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

En los esquemas de bases de datos de gran tamaño, los conjuntos de relaciones (y los esquemas derivados) se suelen denominar mediante la concatenación de los nombres de los conjuntos de entidades a los que hacen referencia, quizás con guiones o caracteres de subrayado intercalados. En este libro se han utilizado algunos nombres de este tipo, por ejemplo, *prof_secc* y *estudiante_secc*. Se han utilizado los nombres *enseña* o *matricula* en lugar de otros concatenados de mayor longitud y resulta aceptable, ya que no es difícil recordar las entidades asociadas a unas pocas relaciones. Sin embargo, no siempre se pueden crear nombres de relaciones mediante la mera concatenación; por ejemplo, la relación jefe o trabaja-para entre empleados no tendría mucho sentido si se llamara *empleado_empleado*. De manera parecida, si hay varios conjuntos de relaciones posibles entre un par de conjuntos de entidades, los nombres de las relaciones deben incluir componentes adicionales para identificar cada relación.

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

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

### 8.8.3. Desnormalización para el rendimiento

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

Por ejemplo, supóngase que hay que mostrar todos los prerrequisitos de las asignaturas junto con la información de la misma, cada vez que se accede a una asignatura. En el esquema normalizado esto exige una reunión de *asignatura* con *prerreq*.

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

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

### 8.8.4. Otros problemas de diseño

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

Considérese una base de datos de una universidad, en la que se desea almacenar el número total de profesores de cada departamento a lo largo de varios años. Se puede utilizar la relación *total_prof* (*nombre_dept, año, total*) para almacenar la información deseada. La única dependencia funcional de esta relación es *nombre_dept, año* → *total*, y se halla en la FNBC.

Un diseño alternativo es el empleo de varias relaciones, cada una de las cuales almacena la información del total para cada año. Supóngase que los años que nos interesan son 2007, 2008 y 2009; se tendrán, entonces, relaciones de la forma *total_prof_2007, total_prof_2008* y *total_prof_2009*, todas las cuales se hallan en el esquema (*nombre_dept, total*). En este caso, la única dependencia funcional de cada relación será *nombre_dept* → *total*, por lo que esas relaciones también se hallan en la FNBC.

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

Otra manera de representar esos mismos datos es tener una sola relación *dept_año* (*nombre_dept, total_prof_2007, total_prof_2008, total_prof_2009*). En este caso, las únicas dependencias funcionales van de *nombre_dept* hacia los demás atributos y, una vez más, la relación se halla en la FNBC. Este diseño también es una mala idea, ya que plantea problemas parecidos a los del diseño anterior; es decir, habría que modificar el esquema de la relación y escribir consultas nuevas cada año. Las consultas también serían más complicadas, ya que puede que tuvieran que hacer referencia a muchos atributos.

Las representaciones como las de la relación *dept_año*, con una columna para cada valor de cada atributo, se denominan de **referencias cruzadas;** se emplean mucho en las hojas de cálculo, en los informes y en las herramientas de análisis de datos. Aunque esas representaciones resultan útiles para mostrárselas a los usuarios, por las razones que se acaban de dar, no resultan deseables en el diseño de bases de datos. Se han propuesto extensiones del SQL para pasar los datos de la representación relacional normal a la de referencias cruzadas, para su visualización, como se ha tratado en la Sección 5.6.2.

## 8.9. Modelado de datos temporales

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

En general, los **datos temporales** son datos que tienen asociado un intervalo de tiempo durante el cual son **válidos.**[8] Se utiliza el término **instantánea** de los datos para indicar su valor en un

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

momento determinado. Por tanto, una instantánea de los datos de *asignatura* muestra el valor de todos los atributos, como su nombre y departamento, en un momento concreto.

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

Cuando se realiza un seguimiento del valor de los datos a lo largo del tiempo, puede que dejen de cumplirse dependencias funcionales que se suponían cumplidas, por ejemplo:

$$ID \rightarrow calle, ciudad$$

puede que no se cumpla. En su lugar, se cumpliría la restricción siguiente (expresada en castellano): «Un profesor *ID* solo tiene un valor de *calle* y de *ciudad* para cada momento *t* dado».

Las dependencias funcionales que se cumplen en un momento concreto se denominan dependencias funcionales temporales. Formalmente, la **dependencia funcional temporal** $X \xrightarrow{T} Y$ se cumple en un esquema de relación $r(R)$ si, para todos los ejemplares legales de $r(R)$, todas las instantáneas de $r$ satisfacen la dependencia funcional $X \rightarrow Y$.

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

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

El paso siguiente es añadir información sobre los periodos de validez a cada una de esas relaciones, añadiendo como atributos el momento de inicio y el de finalización. Por ejemplo, supóngase que se tiene la relación *asignatura*. El nombre de la asignatura puede cambiar con el tiempo, que se puede tratar añadiendo un periodo de tiempo, el esquema resultante sería

$$asignatura\ (asignatura\_id, nombre\_asig, nombre\_dept, inicio, fin)$$

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

Si otra relación tuviera una clave externa que hiciera referencia a una relación temporal, el diseñador de la base de datos tendría que decidir si la referencia se hace a la versión actual de los datos o a los datos de un momento concreto. Por ejemplo, se puede extender la relación *departamento* para registrar los cambios en el edificio o el presupuesto de un departamento en el tiempo, pero una referencia de la relación *profesor* o *estudiante* puede no tener nada que ver con la historia del edificio o el presupuesto, pero se puede referir de forma implícita al registro temporal actual para el correspondiente *nombre_dept*. Por otro lado, los registros del expediente de cada estudiante deben hacer referencia a la denominación de la asignatura en el momento en que la cursó ese estudiante. En este último caso, la relación que hace la referencia también debe almacenar la información temporal, para identificar cada registro concreto de la relación *asignatura*. En nuestro ejemplo, el *año* y *semestre* en que la asignatura se impartió pueden aludir a un valor de hora/fecha como desde medianoche de la fecha de inicio del semestre; el valor resultante de hora/fecha se utiliza para identificar un registro en particular a partir de la versión temporal de la relación *asignatura*, desde la que se obtiene el *nombre*.

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

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

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

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

Aunque no se ha ofrecido ningún método formal para tratar los datos temporales, los problemas estudiados y los ejemplos ofrecidos deben ayudar al lector a diseñar bases de datos que registren datos temporales. Más adelante, en la Sección 25.2, se tratan otros aspectos del manejo de datos temporales, incluidas las consultas temporales.

## 8.10. Resumen

- Se han mostrado algunas dificultades del diseño de bases de datos y el modo de diseñar de manera sistemática esquemas de bases de datos que eviten esas dificultades. Entre esas dificultades están la información repetida y la imposibilidad de representar cierta información.
- Se ha mostrado el desarrollo del diseño de bases de datos relacionales a partir de los diseños E-R, cuándo los esquemas se pueden combinar con seguridad y cuándo se deben descomponer. Todas las descomposiciones válidas deben ser sin pérdidas.
- Se han descrito las suposiciones de dominios atómicos y de primera forma normal.
- Se ha introducido el concepto de las dependencias funcionales y se ha utilizado para presentar dos formas normales, la forma normal de Boyce–Codd (FNBC) y la tercera forma normal (3FN).
- Si la descomposición conserva las dependencias, dada una actualización de la base de datos, todas las dependencias funcionales pueden verificarse a partir de las diferentes relaciones, sin necesidad de calcular la reunión de las relaciones de la descomposición.
- Se ha mostrado la manera de razonar con las dependencias funcionales. Se ha puesto un énfasis especial en señalar las dependencias que están implicadas lógicamente por conjuntos de dependencias. También se ha definido el concepto de recubrimiento canónico, que es un conjunto mínimo de dependencias funcionales equivalente para un conjunto dado de dependencias funcionales.
- Se ha descrito un algoritmo para la descomposición de las relaciones en la FNBC. Hay relaciones para las cuales no hay ninguna descomposición en la FNBC que conserve las dependencias.
- Se han utilizado los recubrimientos canónicos para descomponer las relaciones en la 3FN, que es una pequeña relajación de las condiciones de la FNBC. Las relaciones en la 3FN pueden tener alguna redundancia, pero siempre hay una descomposición en la 3FN que conserva las dependencias.
- Se ha presentado el concepto de dependencias multivaloradas, que especifican las restricciones que no pueden especificarse únicamente con las dependencias funcionales. Se ha definido la cuarta forma normal (4FN) con las dependencias multivaloradas. El Apéndice B.1.1 ofrece detalles del razonamiento sobre las dependencias multivaloradas.
- Otras formas normales, como la FNRP y la FNDC, eliminan formas más sutiles de redundancia. Sin embargo, es difícil trabajar con ellas y se emplean rara vez. El Apéndice B.2 ofrece detalles sobre estas formas normales.
- Al revisar los temas de este capítulo hay que tener en cuenta que el motivo de que se hayan podido definir enfoques rigurosos del diseño de bases de datos relacionales es que el modelo relacional de datos descansa sobre una base matemática sólida. Esa es una de las principales ventajas del modelo relacional en comparación con los otros modelos de datos que se han estudiado.

## Términos de repaso

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

## Ejercicios prácticos

**8.1.** Supóngase que se descompone el esquema $r(A, B, C, D, E)$ en:

$$r_1(A, B, C)$$
$$r_2(A, D, E)$$

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

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

**8.2.** Indíquense todas las dependencias funcionales que satisface la relación de la Figura 8.17.

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_2$ | $b_1$ | $c_1$ |
| $a_2$ | $b_1$ | $c_3$ |

**Figura 8.17.** Relación para el Ejercicio práctico 8.2.

**8.3.** Explíquese el modo en que se pueden utilizar las dependencias funcionales para indicar que:

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

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

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

**8.6.** Calcúlese el cierre del siguiente conjunto $F$ de relaciones funcionales para el esquema de relación $r(A, B, C, D, E)$.

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

Indíquense las claves candidatas de $R$.

**8.7.** Utilizando las dependencias funcionales del Ejercicio práctico 8.6, calcúlese el recubrimiento canónico $F_c$.

**8.8.** Considérese el algoritmo de la Figura 8.18 para calcular $\alpha^+$. Demuéstrese que este algoritmo resulta más eficiente que el presentado en la Figura 8.8 (Sección 8.4.2) y que calcula $\alpha^+$ de manera correcta.

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

**8.10.** Lo tratado sobre la descomposición de reunión sin pérdidas supone implícitamente que los atributos de la parte izquierda de una dependencia funcional no pueden tomar el valor nulo. ¿Qué fallaría en la descomposición si no se cumple esta propiedad?

**8.11.** En el algoritmo de descomposición en FNBC, suponga que se utiliza la dependencia funcional $\alpha \rightarrow \beta$ para descomponer un esquema de relación $r(\alpha, \beta, \gamma)$ en $r_1(\alpha, \beta)$ y $r_2(\alpha, \gamma)$.

**a.** ¿Cuáles son las restricciones de claves primaria y externa que se deberían cumplir en las relaciones descompuestas?

**b.** Indique un ejemplo de inconsistencia, que se puede producir a causa de una actualización errónea, si la restricción de clave externa no se fuerza en las relaciones descompuestas anteriores.

**c.** Cuando una relación se descompone en 3FN utilizando el algoritmo de la Sección 8.5.2, ¿qué dependencias de claves primaria y externa se debería esperar que se cumpliesen en el esquema descompuesto?

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

$$u \subset r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

**8.13.** Demuéstrese que la descomposición del Ejercicio práctico 8.1 no es una descomposición que conserve las dependencias.

```
resultado: = ∅;
/* fdcount es una matriz (array) cuyo elemento i-ésimo contiene
el número de atributos del lado izquierdo de la i-ésima
DF que todavía no se sabe que estén en α+ */
for i: = 1 para |F| do
  begin
    Supóngase que β → γ denota la i-ésima DF;
    fdcount [i]: = |β|;
  end
/* appears es una matriz con una entrada por cada atributo. La entrada
del atributo A es una lista de enteros. Cada entero i de la lista indica que
A aparece en el lado izquierdo de la i-ésima DF */
for each atributo A do
  begin
    appears [A]: = NIL;
    for i: = 1 para |F| do
      begin
        Supóngase que β → γ denota la i-ésima DF;
        if A ∈ β then añadir i a appears [A];
      end
  end
addin (α);
return (resultado);
procedure addin (α);
for each atributo A en α do
  begin
    if A ∈ resultado then
      begin
        resultado: = resultado ∪ {A};
        for each elemento i de appears [A] do
          begin
            fdcount [i]: = fdcount [i] – 1;
            if fdcount [i]: = 0 then
              begin
                Supóngase que β → γ denota la i-ésima DF;
                addin (γ);
              end
          end
      end
  end
```

**Figura 8.18.** Algoritmo para el cálculo de $\alpha^+$.

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

**8.15.** Indique un ejemplo de esquema de relación *R*' y un conjunto *F*' de dependencias funcionales (DF) tales que haya, al menos, tres descomposiciones sin pérdidas distintas de *R*' en FNBC.

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

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

- Aparece en una clave candidata
- No es parcialmente dependiente de una clave candidata.

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

**8.18.** Dé un ejemplo de esquema de relación $R$ y un conjunto de dependencias tales que $R$ se halle en la FNBC, pero no en la 4FN.

## Ejercicios

**8.19.** Indique una descomposición de reunión sin pérdidas en FNBC del esquema $R$ del Ejercicio práctico 8.1.

**8.20.** Indique una descomposición de reunión sin perdidas, que conserva las dependencias en 3FN del esquema $R$ del Ejercicio práctico 8.1.

**8.21.** Normalice el siguiente esquema, con las restricciones indicadas, a 4FN.

*libros* (*acceso_num, isbn, título, autor, editorial*)
*usuarios* (*usuario_id, nombre, dept_id, dept_nombre*)
*acceso_num* $\rightarrow$ *isbn*
*isbn* $\rightarrow$ *título*
*isbn* $\rightarrow$ *editorial*
*isbn* $\rightarrow\!\!\rightarrow$ *autor*
*userid* $\rightarrow$ *nombre*
*userid* $\rightarrow$ *dept_id*
*deptid* $\rightarrow$ *dept_nombre*

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

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

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

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

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

**8.27.** Utilizando las dependencias funcionales del Ejercicio práctico 8.6, calcúlese $B^+$.

**8.28.** Demuéstrese que la siguiente descomposición del esquema $R$ del Ejercicio práctico 8.1 no es una descomposición sin pérdidas:

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

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

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

**8.29.** Considérese el siguiente conjunto $F$ de dependencias funcionales en el esquema de relación $r$ $(A, B, C, D, E, F)$:

$$A \rightarrow BCD$$
$$BC \rightarrow DE$$
$$B \rightarrow D$$
$$D \rightarrow A$$

**a.** Calcule $B^+$.

**b.** Demuestre (usando los axiomas de Armstrong) que $AF$ es una superclave.

**c.** Calcule un recubrimiento canónico para el conjunto de dependencias funcionales $F$ anterior; indique los pasos de la derivación con las correspondientes explicaciones.

**d.** Indique una descomposición en 3FN de $r$ basada en el recubrimiento canónico.

**e.** Indique una descomposición en FNBC de $r$ utilizando el conjunto original de dependencias funcionales.

**f.** ¿Se podría obtener la misma descomposición en FNBC de $r$ anterior, utilizando el recubrimiento canónico?

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

**8.31.** En el diseño de una base de datos relacional, ¿por qué se podría llegar a elegir un diseño que no estuviese en FNBC?

**8.32.** Dados los tres objetivos de diseño de las bases de datos relacionales, ¿hay alguna razón para diseñar un esquema en la 2FN, pero que no esté en ninguna forma normal superior? (Consulte el Ejercicio práctico 8.17 para ver la definición de 2FN).

**8.33.** Dado el esquema relacional $r$ $(A, B, C, D)$, ¿implica lógicamente $A \rightarrow\!\!\rightarrow BC$ a $A \rightarrow\!\!\rightarrow B$ y a $A \rightarrow\!\!\rightarrow C$? En caso positivo, pruébese; en caso contrario, ofrézcase un contraejemplo.

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

## Notas bibliográficas

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

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

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

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

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

# Diseño y desarrollo de aplicaciones

09

Casi todo el uso de las bases de datos se produce desde los programas de aplicación. A su vez, casi toda la interacción de los usuarios con las bases de datos es indirecta, mediante los programas de aplicación. No resulta sorprendente, por tanto, que los sistemas de bases de datos lleven mucho tiempo soportando herramientas como los generadores de formularios y de interfaces gráficas de usuario, que ayudan a lograr el desarrollo rápido de aplicaciones que actúan de interfaz con los usuarios. En los últimos años, la red web se ha transformado en la interfaz de usuario con las bases de datos más utilizada.

En este capítulo se estudian las herramientas y las tecnologías necesarias para crear aplicaciones de bases de datos. En concreto, se centrará la atención en las herramientas interactivas que utilizan bases de datos para guardar datos.

Tras una introducción a los programas de aplicación y a las interfaces de usuario, en la Sección 9.1 se tratará el desarrollo de aplicaciones con interfaces basadas en web. Se comienza con una descripción general de las tecnologías web, en la Sección 9.2, y en la Sección 9.3 se tratará la tecnología Java Servlets, que se usa extensamente para la construcción de aplicaciones web. En la Sección 9.4 se presenta una breve introducción a las arquitecturas de aplicaciones web. En la Sección 9.5 se tratan las herramientas para el desarrollo rápido de aplicaciones, y en la Sección 9.6 se ven los temas de rendimiento en la construcción de grandes aplicaciones web. En la Sección 9.7 se tratan los temas de seguridad de las aplicaciones. Se termina el capítulo con la Sección 9.8, en la que se verán los temas de cifrado y su uso en las aplicaciones.

## 9.1. Interfaces de usuario y programas de aplicación

Aunque mucha gente interactúa con las bases de datos, pocas personas usan un lenguaje de consultas para interactuar directamente con los sistemas de bases de datos. La mayor parte de las personas interactúan con los sistemas de bases de datos usando un programa de aplicación que proporciona una interfaz de usuario, e interacciona con la base de datos por detrás. Estas aplicaciones recogen la entrada de los usuarios, normalmente mediante una interfaz de formularios, e insertan datos en una base de datos o extraen información de la misma de acuerdo con la interacción del usuario, generando una salida que se muestra en la pantalla.

Como ejemplo de aplicación, considere un sistema de matrícula de una universidad. Al igual que otras aplicaciones, el sistema de registro solicitará, en primer lugar, su identidad para realizar la autenticación, normalmente mediante un nombre de usuario y una contraseña. La aplicación utiliza esta información de identificación para extraer la información, como el nombre y asignaturas en las que se esté matriculado, de la base de datos, y mostrará dicha información. La aplicación proporciona un cierto número de interfaces para permitir matricularse en asignaturas y solicitar distintos tipos de información sobre asignaturas y profesores. Las organizaciones utilizan este tipo de aplicaciones para automatizar muchas tareas, como ventas, compras, contabilidad y nóminas, gestión de recursos humanos, gestión de inventario y otras.

Se pueden utilizar programas de aplicación cuando incluso no resulte evidente que se pueden usar. Por ejemplo, un sitio de noticias puede proporcionar una página que se configura de forma transparente para cada usuario, incluso sin que el usuario rellene ningún formulario cuando interacciona con el sitio. Para ello, realmente se ejecuta un programa de aplicación que genera la página personalizada para cada usuario; la personalización puede, por ejemplo, basarse en el histórico de artículos visitados.

Un programa de aplicación típico incluye un componente de «front-end», que trata con la interfaz de usuario, y un componente de «back-end», que se comunica con la base de datos, así como una capa intermedia que contiene la «lógica de negocio», es decir, el código que ejecuta las peticiones concretas de información o actualizaciones; regula las normas del negocio, como las acciones que hay que llevar a cabo para ejecutar una determinada tarea, o quién puede realizar una tarea.

Las arquitecturas de las aplicaciones han evolucionado con el tiempo, como se muestra en la Figura 9.1. Las aflicciones, como las de reserva de vuelos, llevan en funcionamiento desde los años sesenta. En aquellos tiempos, los programas de aplicación se ejecutaban en grandes «mainframe» y los usuarios interactuaban con las aplicaciones mediante terminales, algunos de los cuales disponían de formularios.

Con el amplio uso de los ordenadores personales, muchas organizaciones usaron una arquitectura diferente para las aplicaciones internas, con aplicaciones que se ejecutaban en las computadoras de los usuarios y accedían a una base de datos central. Esta arquitectura, a menudo denominada «cliente-servidor», permitió la creación de interfaces gráficas de usuario muy potentes, que las aplicaciones con terminales anteriores no podían usar. Sin embargo, había que instalar el software en cada una de las máquinas donde se ejecutaba la aplicación, lo que hacía que las actualizaciones fuesen muy costosas. Incluso en la época de las computadoras personales, cuando se hicieron populares las arquitecturas cliente-servidor, las arquitecturas de mainframes continuaron siendo la elección para aplicaciones como las de reservas de vuelos, que se usan desde muchos puntos geográficamente distribuidos.

En los últimos quince años, los navegadores web se han convertido en el *front-end universal* para las aplicaciones de base de datos, conectando con el back-end mediante Internet. Los navegadores utilizan una sintaxis normalizada, el **lenguaje de marcas de hipertexto (HTML),** que admite mostrar información con formato, como la creación de interfaces con formularios. La norma HTML es independiente del sistema operativo o del navegador, y todas las computadoras actuales tienen instalado un navegador. Por tanto, se puede acceder a una aplicación basada en web desde cualquier computadora conectada a Internet.

Al contrario que las arquitecturas cliente-servidor, no es necesario instalar ninguna aplicación específica en la máquina cliente para poder utilizar las aplicaciones basadas en web. Sin embargo, se usan ampliamente interfaces de usuario sofisticadas, con características que van más allá de lo que es posible utilizando únicamente HTML, gracias al lenguaje JavaScript, que se puede utilizar con la mayor parte de los navegadores web. Los programas con JavaScript, al contrario que los escritos en C, se pueden ejecutar en modo seguro, garantizando que no generan problemas de seguridad. Los programas en JavaScript se descargan de forma transparente con el navegador y no necesitan un proceso de instalación de software en la computadora.

Si los navegadores web proporcionan el *front-end* para la interacción con los usuarios, los programas de aplicación constituyen el *back-end*. Normalmente, las peticiones desde un navegador se envían a un servidor web, donde se ejecutan los programas de aplicación que procesan la consulta. Existen diversas tecnologías para la creación de los programas de aplicación que se ejecutan en el back-end, entre otras Java Servlets, Java Server Pages (JSP), Active Server Pages (ASP); o lenguajes de guiones, como PHP, Perl o Python.

En el resto del capítulo se describe cómo se construyen estas aplicaciones, empezando con las tecnologías web y las herramientas para construir interfaces web, así como las tecnologías para construir programas de aplicación, y después se tratarán los temas sobre arquitecturas, rendimiento y seguridad en la creación de aplicaciones.

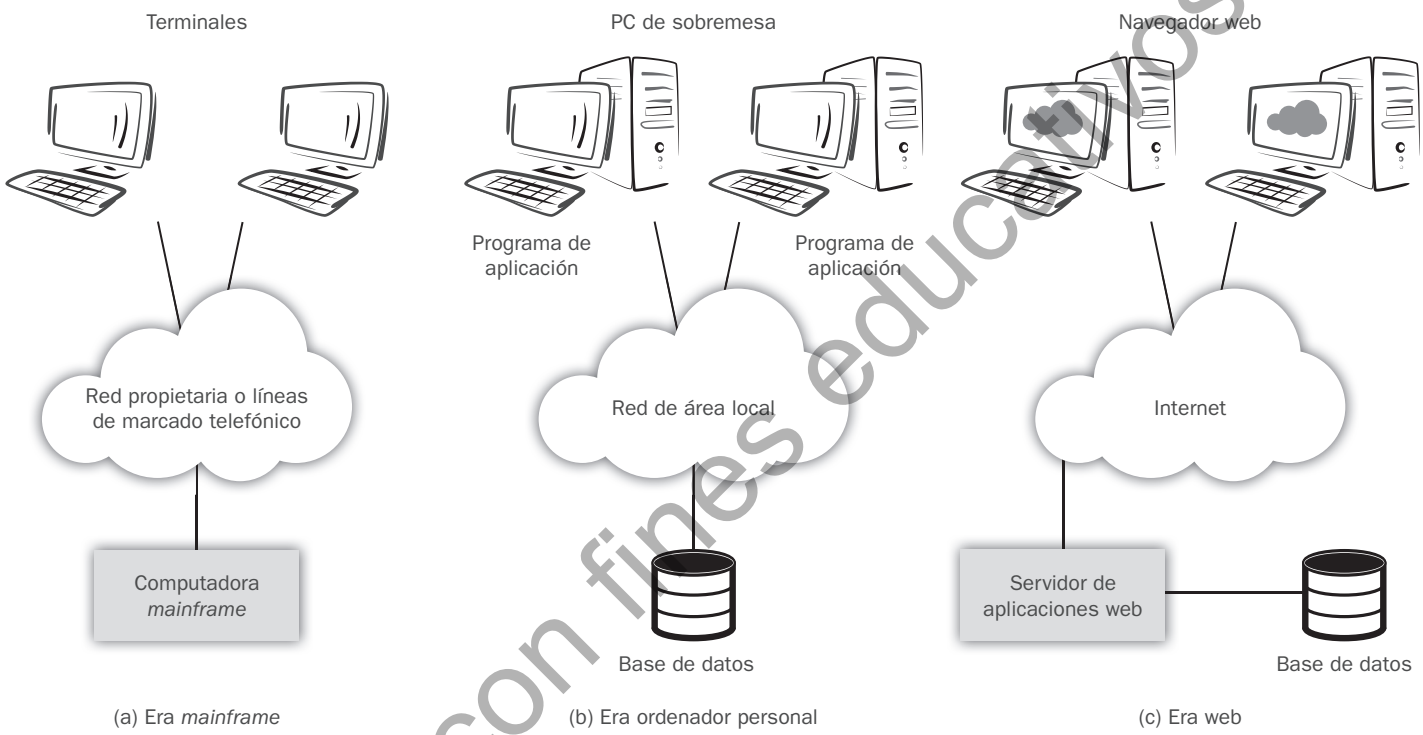



**Figura 9.1.** Arquitecturas de aplicación en diferente eras.

## 9.2. Fundamentos de la web

En esta sección se va a repasar parte de la tecnología básica que subyace en la World Wide Web, para los lectores que no estén familiarizados con ella.

### 9.2.1. Los localizadores uniformes de recursos

Un **localizador uniforme de recursos (URL: *Uniform Resource Locator*)** es un nombre globalmente único para cada documento al que se puede tener acceso en web. Un ejemplo de URL es:

http://www.acm.org/sigmod

La primera parte de la URL[1] indica el modo en que se puede tener acceso al documento: «http» indica que se puede tener acceso al documento mediante el **protocolo de transferencia de hipertexto** (*HyperText Transfer Protocol*: HTTP), que es un protocolo para transferir documentos HTML. La segunda parte indica el nombre único de una máquina con el servidor web. El resto de la URL es el nombre del camino hasta el archivo en la máquina, u otro identificador único del documento dentro de la máquina.

1 A este «localizador», en cuanto «dirección», se le asigna género femenino en el lenguaje coloquial entre usuarios. (*N. del E.*)

Una URL puede contener el identificador de un programa ubicado en la máquina servidora web, así como los argumentos que hay que dar al programa. Un ejemplo de URL de este tipo es:

http://www.google.com/search?q=silberschatz

que dice que el programa search del servidor www.google.com se debe ejecutar con el argumento q=silberschatz. El programa se ejecuta usando el argumento dado, y devuelve un documento HTML que se envía a la interfaz.

```
<html>
<body>
<table border>
<tr> <th>ID</th>       <th>Nombre</th> <th>Departamento</th> </tr>
<tr> <td>00128</td>  <td>Zhang</td>   <td>Informática</td> </tr>
<tr> <td>12345</td>  <td>Shankar</td> <td>Informática</td> </tr>
<tr> <td>19991</td>  <td>Brandt</td>   <td>Historia</td> </tr>
</table>
</body>
</html>
```

**Figura 9.2.** Datos tabulares en formato HTML.

### 9.2.2. El lenguaje de marcas de hipertexto

La Figura 9.2 es un ejemplo fuente de una tabla representado en un documento HTML, mientras la Figura 9.3 muestra la imagen que genera de este documento un navegador a partir de la representación HTML de la tabla. El código fuente en HTML muestra unas pocas etiquetas de HTML. Cada página HTML debe encerrarse con la etiqueta html, y el cuerpo de la página se encierra en una etiqueta body. Una tabla se especifica con la etiqueta table, que contiene las filas que indican las etiquetas tr. La fila de cabecera de la tabla tiene celdas de tabla especificadas con la etiqueta th, mientras que las filas normales se indican con la etiqueta td. No vamos a entrar más en detalle sobre las etiquetas. Véanse las notas bibliográficas para más información sobre la descripción de HTML.

| ID | Nombre | Departamento |
|---|---|---|
| 00128 | Zhang | Informática |
| 12345 | Shankar | Informática |
| 19991 | Brandt | Historia |

**Figura 9.3.** Vista del código HTML de la Figura 9.2.

```
<html>
<body>
<form action="PersonQuery" method=get>
Buscar:
<select name="persontype">
        <option value="student" selected>Estudiante</option>
        <option value="instructor">Profesor</option>
</select> <br>
Nombre: <input type=text size=20 name="name">
<input type=submit value="Enviar">
</form>
</body>
</html>
```

**Figura 9.4.** Formulario en HTML.

En la Figura 9.4 se muestra cómo especificar un formulario de HTML que permite a los usuarios seleccionar el tipo de persona (estudiante o profesor) en un menú e introducir un número en un cuadro de texto. La Figura 9.5 muestra cómo se muestra el formulario anterior en un navegador web. En el formulario de ejemplo se muestran dos métodos para aceptar entradas del usuario, pero HTML dispone de muchas otras. El atributo action de la etiqueta form especifica que cuando se envíe el formulario, haciendo clic en el botón enviar, los datos del formulario se deben enviar a la URL PersonQuery (la URL es relativa a la de la página). El servidor web está configurado para que cuando se accede a esa URL, se invoca al programa de aplicación correspondiente, con los valores que proporciona el usuario para los argumentos persontype y name (especificados en los campos select e input). El programa de aplicación genera un documento en HTML que se envía de vuelta y se muestra al usuario; más adelante en este capítulo se verá cómo construir estos programas.

HTTP define dos formas para que los valores que introduzca el usuario en el navegador web se envíen al servidor web. El método get codifica los valores formando parte de la URL. Por ejemplo, si la búsqueda de Google usa un formulario con un parámetro de entrada de nombre q con el método get, y el usuario escribe en él la cadena de texto «silberschatz» y envía el formulario, el navegador solicitará la siguiente URL al servidor web:

http://www.google.com/search?q=silberschatz

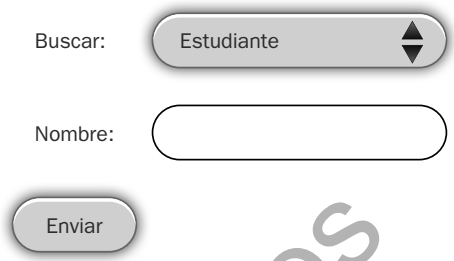



**Figura 9.5.** Vista del código HTML de la Figura 9.4.

El método post, por el contrario, envía una petición de la URL http://www.google.com, y manda el valor de los parámetros como parte del intercambio del protocolo HTTP entre el servidor web y el navegador. El formulario de la Figura 9.4 especifica que el formulario utiliza el método get.

Aunque el código HTML se puede crear usando un editor de texto sencillo, hay varios editores que permiten la creación directa de texto HTML usando una interfaz gráfica. Estos editores permiten insertar construcciones como los formularios, los menús y las tablas en el documento HTML a partir de un menú de opciones, en lugar de escribir manualmente el código para generar esas construcciones.

HTML soporta *hojas de estilo*, que pueden modificar las definiciones predeterminadas del modo en que se muestran las estructuras de formato HTML, así como otros atributos de visualización como el color de fondo de la página. La norma *hojas de estilo en cascada* (*cascading stylesheet*, CSS) permite usar varias veces la misma hoja de estilo para varios documentos HTML, dando un aspecto uniforme, aunque distintivo, a todas las páginas del sitio web.

### 9.2.3. Los servidores web y las sesiones

Los **servidores web** son programas que se ejecutan en la máquina servidora, que aceptan las solicitudes de los navegadores web y devuelven los resultados en forma de documentos HTML. El navegador y el servidor web se comunican mediante un protocolo HTTP. Los servidores web proporcionan características potentes, aparte de la mera transferencia de documentos. La característica más importante es la posibilidad de ejecutar programas, con los argumentos proporcionados por el usuario, y devolver los resultados como documentos HTML.

En consecuencia, los servidores web pueden actuar con facilidad como intermediarios para ofrecer acceso a gran variedad de servicios de información. Se pueden crear servicios nuevos mediante la creación e instalación de programas de aplicaciones que los ofrezcan. La norma **interfaz de pasarela común** (*Common Gateway Interface*: CGI) define el modo en que el servidor web se comunica con los programas de las aplicaciones. Los programas de las aplicaciones suelen comunicarse con servidores de bases de datos mediante ODBC, JDBC u otros protocolos, con objeto de obtener o guardar datos.

La Figura 9.6 muestra un servicio web que usa una arquitectura de tres capas, con un servidor web, un servidor de aplicaciones y un servidor de bases de datos. El uso de varios niveles de servidores aumenta la sobrecarga del sistema; la interfaz CGI inicia un nuevo proceso para atender cada solicitud, lo cual supone una sobrecarga aún mayor.

La mayor parte de los servicios web de hoy en día usan una arquitectura web de dos capas, en la que los programas de las aplicaciones se ejecutan en el servidor web, como en la Figura 9.7. En las secciones siguientes se estudiarán con más detalle los sistemas basados en la arquitectura de dos capas.

No existe ninguna conexión continua entre los clientes y los servidores web; cuando un servidor web recibe una solicitud, se crea temporalmente una conexión para enviar la solicitud y recibir la respuesta del servidor web. Pero se cierra la conexión, y la siguiente solicitud llega mediante otra nueva. A diferencia de esto, cuando un usuario inicia una sesión en una computadora, o se conecta a una base de datos mediante ODBC o JDBC, se crea una sesión y en el servidor y en el cliente se conserva la información de la sesión hasta que esta concluye; información como el identificador de usuario y las opciones de sesión que este haya definido. Una razón de peso para que HTTP sea **sin conexión** es que la mayor parte de las computadoras presentan un número limitado de conexiones simultáneas que pueden aceptar, por lo que se podría superar ese límite y denegar el servicio a otros usuarios.

Con un servicio sin conexión, esta se interrumpe en cuanto se satisface una solicitud, lo que deja las conexiones disponibles para otras solicitudes.[2]

La mayor parte de las aplicaciones web necesitan información de sesión para permitir una interacción significativa con el usuario. Por ejemplo, los servicios suelen restringir el acceso a la información y, por tanto, necesitan autenticar a los usuarios. La autenticación debe hacerse una vez por sesión, y las interacciones posteriores de la sesión no deben exigir que se repita la autenticación.

Para implementar sesiones a pesar del cierre de las conexiones, hay que almacenar información adicional en el cliente y devolverla con cada solicitud de la sesión; el servidor usa esta información para identificar que la solicitud forma parte de la sesión del usuario. En el servidor también hay que guardar información adicional sobre la sesión.

Esta información adicional se suele conservar en el cliente en forma de **cookie;** una cookie no es más que un pequeño fragmento de texto, con un nombre asociado, que contiene información de identificación. Por ejemplo, google.com puede definir una cookie con el nombre prefs que codifique las preferencias definidas por el usuario, como el idioma elegido y el número de respuestas mostradas por página. En cada solicitud de búsqueda google.com puede recuperar la cookie denominada prefs del navegador del usuario y mostrar el resultado de acuerdo con las preferencias especificadas. Solo se permite a cada dominio (sitio web) que recupere las cookies que ha definido, no las definidas por otros dominios, por lo que los nombres se pueden reutilizar en otros dominios.

Con objeto de realizar un seguimiento de las sesiones de usuario, una aplicación puede generar un identificador de sesión (generalmente un número aleatorio que no se esté usando como identificador de sesión) y enviar una cookie denominada, por ejemplo, idsesión que contenga ese identificador de sesión. El identificador de sesión también se almacena localmente en el servidor. Cuando llega una solicitud, el servidor de aplicaciones solicita al cliente la cookie denominada idsesión. Si el cliente no posee la cookie almacenada, o devuelve un valor que no se halla registrado como identificador de sesión válido en el servidor, la aplicación concluye que la solicitud no forma parte de una sesión actual. Si el valor de la cookie coincide con el de un identificador de sesión almacenado, la solicitud se identifica como parte de una sesión abierta.

2 Por razones de rendimiento, las conexiones se pueden mantener abiertas durante un momento para permitir más peticiones o reusar la conexión. Sin embargo, no se garantiza que la conexión se mantenga abierta, y las aplicaciones deben diseñarse suponiendo que las conexiones se cierran en cuanto se sirve la petición.

Si una aplicación necesita identificar de manera segura a los usuarios, solo puede definir la cookie después de autentificar al usuario; por ejemplo, se puede autentificar al usuario solo cuando se hayan enviado un nombre de usuario y una contraseña válidos.[3]

Para las aplicaciones que no exigen un nivel elevado de seguridad, como los sitios de noticias abiertos al público, las cookies se pueden almacenar de manera permanente en el navegador y en el servidor; identifican al usuario en visitas posteriores al mismo sitio, sin necesidad de que escriba ninguna información de identificación. Para las aplicaciones que exijan un nivel de seguridad más elevado, el servidor puede invalidar (eliminar) la sesión tras un periodo de inactividad o cuando el usuario la cierre (generalmente los usuarios cierran la sesión pulsando un botón de cierre de sesión, que remite un formulario de cierre de sesión, que invalida la sesión actual). La invalidación de la sesión consiste simplemente en eliminar el identificador de la sesión de la lista de sesiones activas del servidor de aplicaciones.

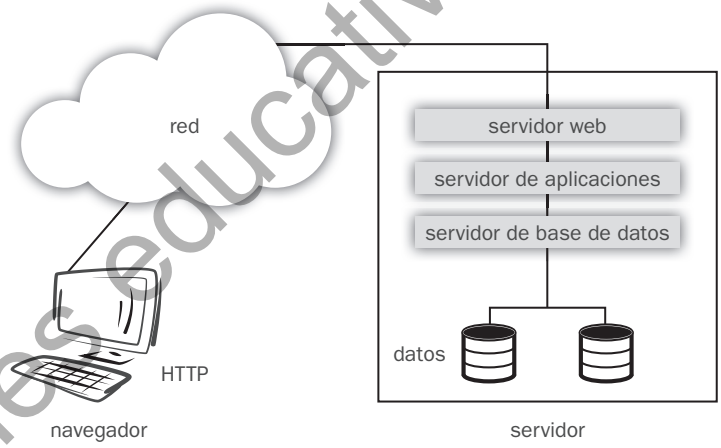



**Figura 9.6.** Arquitectura de aplicación web de tres capas.

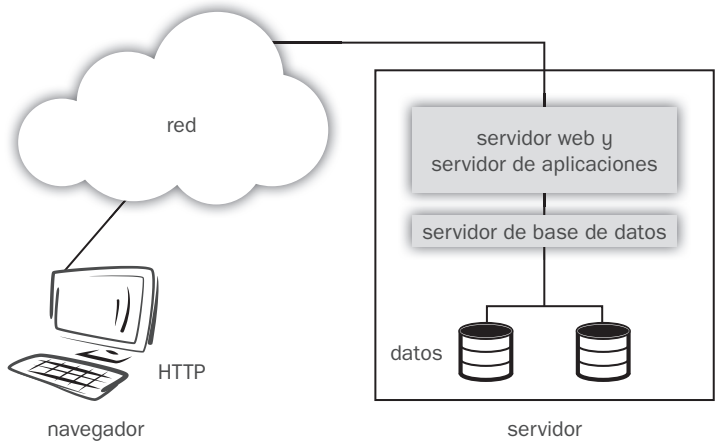



**Figura 9.7.** Arquitectura de aplicación web en dos capas.

3 El identificador de usuario puede guardarse en la parte del cliente, en una cookie denominada, por ejemplo, idusuario. Estas cookies se pueden usar para aplicaciones con un nivel de seguridad bajo, como los sitios web gratuitos que identifican a sus usuarios. No obstante, para las aplicaciones que exigen un nivel de seguridad más elevado, este mecanismo genera un riesgo: los usuarios malintencionados pueden modificar el valor de las cookies en el navegador y luego suplantar a otros usuarios. La definición de una cookie (denominada idsesión, por ejemplo) con un identificador de sesión generado aleatoriamente (a partir de un espacio de números de gran tamaño) hace muy improbable que ningún usuario pueda suplantar a un usuario diferente (es decir, que pretenda ser otro usuario). Los identificadores de sesión generados de manera secuencial, por otro lado, sí son susceptibles de suplantación.

## 9.3. Servlets y JSP

En la arquitectura web de dos capas, las aplicaciones se ejecutan como parte del propio servidor web. Un modo de implementar esta arquitectura es cargar los programas Java en el servidor web. La especificación **servlet** de Java define una interfaz de programación de aplicaciones para la comunicación entre el servidor web y los programas de aplicaciones. La clase HttpServlet de Java implementa la especificación de la API servlet; las clases servlet usadas para implementar funciones concretas se definen como subclases de esta clase.[4] A menudo se usa la palabra *servlet* para hacer referencia a un programa (y a una clase) de Java que implementa la interfaz servlet. La Figura 9.8 muestra un ejemplo de código servlet; en breve se explicará con detalle.

El código del servlet se carga en el servidor web cuando se inicia el servidor, o cuando el servidor recibe una solicitud HTTP remota para ejecutar un servlet concreto. La tarea del servlet es procesar esa solicitud, lo cual puede suponer acceder a una base de datos para recuperar la información necesaria, y generar dinámicamente una página HTML para devolvérsela al navegador del cliente.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersonQueryServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HEAD><TITLE>Resultado de la consulta</TITLE>
                   </HEAD>");
      out.println("<BODY>");

      String persontype = request.getParameter("persontype");
      String number = request.getParameter("name");
      if(persontype.equals("Estudiante")) {
         ... código para encontrar estudiantes con el nombre dado ...
         ... usando JDBC para comunicarse con la base de datos ...
         out.println("<table BORDER COLS=3>");
         out.println("<tr> <td>ID</td> <td>Nombre:</td>" +
                     "<td>Departamento</td> </tr>");
         for(... cada resultado ...) {
            ... obtener ID, nombre y nombre_dept
            ... en variables ID, nombre y nombredept
            out.println("<tr> <td>" + ID + "</td>" + "<td>" + nombre +
                        "</td>" + "<td>" + nombredept + "</td> </tr>");
         }
         out.println("</table>");
      } else {
         ... como antes, pero para profesores ...
      }
      out.println("</BODY>");
      out.close();
   }
}
```

**Figura 9.8.** Ejemplo de código de un servlet.

4 La interfaz servlet también puede admitir peticiones que no sean HTTP, aunque en el ejemplo solo se usa HTTP.

### 9.3.1. Un ejemplo de servlet

Los servlets se emplean a menudo para generar de manera dinámica respuestas a las solicitudes de HTTP. Pueden tener acceso a datos proporcionados mediante formularios HTML, aplicar la «lógica de negocio» para decidir la respuesta que deben dar y generar el resultado HTML que se devolverá al navegador.

La Figura 9.8 muestra un ejemplo de código de servlet para implementar el formulario de la Figura 9.4. El servlet se denomina PersonQueryServlet, mientras que el formulario especifica que «action="PersonQuery"». Se debe indicar al servidor web que este servlet se va a utilizar para tratar las solicitudes de PersonQuery. El formulario especifica que se usa el mecanismo get de HTTP para transmitir los parámetros. Por tanto, se invoca el método doGet() del servlet, que se define en el código.

Cada solicitud da lugar a una nueva hebra en la que se ejecuta la llamada, por lo que se pueden tratar en paralelo varias solicitudes. Los valores de los menús del formulario y de los campos de entrada de la página web, así como las cookies, se comunican a un objeto de la clase HttpServletRequest que se crea para la solicitud, y la respuesta a la solicitud se comunica a un objeto de la clase HttpServletResponse.

El método doGet() del ejemplo extrae los tipos de parámetros y el número de parámetros mediante request.getParameter(), y los utiliza para realizar una consulta a la base de datos. El código para tener acceso a la base de datos y obtener los valores de los atributos no se muestra; consulte la Sección 5.1.1.4 para conseguir detalles del modo de uso de JDBC para acceder a una base de datos. El código del servlet devuelve el resultado de la consulta al solicitante imprimiéndolos en formato HTML en la respuesta del objeto HttpServletResponse. La salida de los resultados se implementa creando un objeto out de la clase PrintWriter a partir de response y, después, escribiendo el resultado en formato HTML utilizando out.

### 9.3.2. Sesiones de los servlets

Recuerde que la interacción entre el navegador y el servidor web carece de estado. Es decir, cada vez que el navegador formula una solicitud al servidor, necesita conectarse al servidor, solicitar alguna información y desconectarse del servidor. Se pueden emplear cookies para reconocer que una solicitud dada procede de la misma sesión de navegador que otra anterior. Sin embargo, las cookies constituyen un mecanismo de bajo nivel y los programadores necesitan una abstracción mejor para tratar con las sesiones.

La API de servlet ofrece un método para realizar el seguimiento de las sesiones y almacenar la información relacionada con ellas. La invocación del método getSession(false) de la clase HttpServletRequest recupera el objeto HttpSession correspondiente al navegador que envió la solicitud. Si el valor del argumento se pone a true significa que hay que crear un nuevo objeto sesión si la solicitud fuera nueva.

Cuando se invoca el método getSession(), el servidor pide primero al cliente que devuelva una cookie con un nombre concreto. Si el cliente no tiene ninguna cookie con ese nombre, o devuelve un valor que no corresponde al de ninguna sesión abierta, la solicitud no forma parte de ninguna sesión abierta. En ese caso, devolverá un valor null y el servlet puede dirigir al usuario a una página de identificación.

La página de identificación permite que el usuario introduzca su nombre de usuario y su contraseña. El servlet correspondiente a la página de identificación puede comprobar que la contraseña coincide con la del usuario (por ejemplo, buscando la información de autenticación en la base de datos). Si el usuario se identifica correctamente, el servlet de identificación ejecuta getSession(true), que devuelve un nuevo objeto sesión. Para crear una nueva sesión,

el servidor web ejecuta internamente las siguientes tareas: definir una cookie (denominada, por ejemplo, idsesión) con un identificador de sesión como valor asociado en el navegador cliente, crear un nuevo objeto sesión y asociar el valor del identificador de sesión con el objeto sesión.

El código del servlet también puede almacenar y buscar pares (nombre-atributo, valor) en el objeto HttpSession, para mantener el estado entre varias solicitudes de una misma sesión. Por ejemplo, el servlet del inicio de sesión puede almacenar el identificador de usuario como parámetro de la sesión ejecutando el método

```
sesion.setAttribute("idusuario", idusuario)
```

sobre el objeto sesión que devuelve getSession(); se supone que la variable de Java idusuario contiene el identificador del usuario.

Si la solicitud formaba parte de una sesión abierta, el navegador habría devuelto el valor de la cookie y el servidor web habría devuelto el objeto sesión correspondiente. El servlet puede recuperar entonces los parámetros de la sesión, como el identificador de usuario, a partir del objeto sesión ejecutando el método:

```
sesion.getAttribute("idusuario")
```

con el objeto sesión obtenido anteriormente. Si el atributo idusuario no está definido, la función devuelve el valor nulo, lo que indica que el usuario del cliente no se ha autenticado.

### 9.3.3. Ciclo de vida de los servlets

El ciclo de vida de cada servlet está controlado por el servidor web en el que se ha desplegado. Cuando hay una solicitud de un cliente para un servlet concreto, el servidor comprueba primero si existe algún ejemplar de ese servlet o no. Si no existe, el servidor web carga la clase del servlet en la máquina virtual de Java (Java virtual machine, JVM) y crea un ejemplar de la clase del servlet. Además, el servidor llama al método init() para inicializar el ejemplar del servlet. Tenga en cuenta que cada ejemplar del servlet solo se inicializa una vez, cuando se carga.

Tras asegurarse de que existe el ejemplar del servlet, el servidor invoca el método service() del servlet, con un objeto request y un objeto response como parámetros. De manera predeterminada, el servidor crea una hebra nueva para ejecutar el método service(); por tanto, se pueden ejecutar en paralelo varias solicitudes al servlet, sin tener que esperar que solicitudes anteriores completen su ejecución. El método service() llama a doGet() o a doPost(), según corresponda.

Cuando ya no sea necesario, se puede cerrar el servlet llamando al método destroy(). El servidor se puede configurar para que cierre de manera automática el servlet si no se han hecho solicitudes en un periodo de tiempo determinado; este periodo es un parámetro del servidor que se puede definir según se considere adecuado para la aplicación.

### 9.3.4. Soporte de los servlets

Muchos servidores de aplicaciones ofrecen soporte predeterminado para servlets. Entre los más utilizados está Tomcat Server del Apache Jakarta Project. Otros servidores de aplicaciones que admiten el uso de servlet son Glassfish, JBoss, **BEA** Weblogic Application Server, Oracle Application Server y WebSphere Application Server de IBM.

La forma más apropiada de desarrollar aplicaciones con servlet es utilizar un **IDE** como Eclipse o NetBeans, que vienen con servidores Glassfish o Tomcat incorporados.

Estos servidores de aplicaciones ofrecen gran variedad de servicios adicionales, aparte del soporte básico para servlets. Permiten desplegar aplicaciones y pararlas, así como proporcionar funciones para monitorizar el estado del servidor de aplicaciones, incluyendo estadísticas de rendimiento. Si se modifica el código de un servlet, pueden detectarlo y volver a compilar y a cargar el servlet de manera transparente. Muchos servidores de aplicaciones permiten que el servidor se ejecute en varias máquinas en paralelo para mejorar el rendimiento y encaminar las solicitudes a la copia adecuada. Muchos servidores de aplicaciones soportan también la plataforma Java 2 Enterprise Edition (J2EE), que ofrece soporte y API para gran variedad de tareas, como el manejo de objetos, el procesamiento en paralelo en varios servidores de aplicaciones y el manejo de datos en XML (XML se describe más adelante, en el Capítulo 23).

### 9.3.5. Secuencias de comandos en el lado del servidor

La escritura, incluso de una mera aplicación web, en un lenguaje de programación como Java o C supone una tarea que consume bastante tiempo y necesita muchas líneas de código y programadores familiarizados con las complejidades del lenguaje. Un enfoque alternativo, el de las **secuencias de comandos en el lado del servidor,** ofrece un método mucho más sencillo para la creación de múltiples aplicaciones. Los lenguajes de guiones ofrecen estructuras que pueden incluirse en los documentos HTML. En los guiones en el lado del servidor, antes de entregar una página web, el servidor ejecuta las secuencias incluidas en el contenido de la página en HTML. Cada secuencia, al ejecutarse, puede generar texto que se añade a la página (o que incluso puede eliminar contenido de la página). El código fuente de los guiones se elimina de la página, de modo que puede que el cliente ni siquiera se dé cuenta de que la página contenía originalmente algún código. Puede que el guion ejecutado contenga un código SQL que se ejecute usando una base de datos.

Entre los lenguajes de guiones más utilizados se encuentran: Java Server Pages (JSP) de Sun, Active Server Pages (ASP) y su sucesor ASP.NET de Microsoft, PHP (PHP Hypertext Preprocessor, PHP), el lenguaje de marcas de ColdFusion (ColdFusion Markup Language, CFML) y Ruby on Rails. Muchos lenguajes de guiones también permiten incluir un código en lenguajes como Java, C#, VBScript, Perl y Python, de forma que se incluya en el HTML o se invoque desde páginas HTML. Por ejemplo, JSP permite incluir código en Java en las páginas en HTML, mientras que ASP.NET de Microsoft y ASP permiten incluir C# y VBScript. Muchos de estos lenguajes disponen de bibliotecas y herramientas que constituyen un marco («framework») para el desarrollo de aplicaciones web.

```
<html>
<head> <title>Hola</title> </head>
<body>
   <% if (request.getParameter("nombre") == null)
      {out.println("Hola mundo"); }
      else {out.println("Hola, " + request.getParameter("nombre")); }
   %>
</body>
</html>
```

**Figura 9.9.** Página en JSP con código Java incrustado.

A continuación se describe brevemente **Java Server Pages (JSP),** un lenguaje de guiones que permite que los programadores de HTML mezclen HTML estático con el generado de manera dinámica. El motivo es que, en muchas páginas web dinámicas, la mayor parte del contenido sigue siendo estático (es decir, se halla presente el mismo contenido siempre que se genera la página). El contenido dinámico de la página web (que se genera, por ejemplo, en función de los parámetros de los formularios) suele ser una pequeña parte de la página. La creación de estas páginas mediante la escritura de código servlet da lugar a grandes cantidades de HTML que se codifican como cadenas de Java. Por el contrario, JSP permite que el código Java se incorpore al HTML estático; el código Java incorporado genera la parte dinámica de la página. Los guiones de JSP realmente se traducen en código de servlet que luego se compila, pero se le ahorra al programador de la aplicación el problema de tener que escribir gran parte del código para crear el servlet.

La Figura 9.9 muestra el texto fuente de una página JSP que incluye un código Java incrustado. El código Java se distingue del de HTML circundante por estar encerrado entre <% … %>. El código usa request.getParameter() para obtener el valor del atributo nombre.

Cuando el navegador solicita una página JSP, el servidor de aplicaciones genera la salida en HTML de la página, que se envía de vuelta al navegador. La parte en HTML de la página en JSP se genera tal cual.[5] Siempre que haya código Java incluido entre <% … %>, el código se sustituye en la salida en HTML por el texto que escribe el objeto out. En el código JSP de la figura anterior; si no se introduce ningún valor en el parámetro nombre del formulario, el guion escribe: «Hola mundo»; si se introduce un valor, el guion escribe «Hola» seguido del nombre escrito.

Un ejemplo más realista puede llevar a cabo acciones más complejas, como buscar valores en una base de datos usando JDBC.

JSP también dispone del concepto de *biblioteca de etiquetas* (*tag library*), que permite el uso de etiquetas que se parecen mucho a las etiquetas HTML, pero se interceptan en el servidor y se sustituyen por el código HTML correspondiente. JSP ofrece un conjunto estándar de etiquetas que definen variables y controlan el flujo (iteradores, si-entonces-sino), junto con un lenguaje de expresiones basado en Javascript (pero interpretado en el servidor). El conjunto de etiquetas es extensible, y ya se han implementado varias bibliotecas de etiquetas. Por ejemplo, existe una biblioteca de etiquetas que soporta la visualización paginada de conjuntos de datos de gran tamaño y una biblioteca que simplifica la visualización y el análisis de fechas y de horas. Véanse las notas bibliográficas para conocer más referencias de información sobre las bibliotecas de etiquetas de JSP.

### 9.3.6. Secuencias de comandos en el lado del cliente

La inclusión de códigos de programas en los documentos permite que las páginas web sean **activas,** logrando actividades como animaciones ejecutando programas en el lado local, en lugar de presentar únicamente texto y gráficos pasivos. El uso principal de estos programas es una interacción flexible con el usuario, más allá de la limitada interacción de HTML y sus formularios. Más aún, la ejecución de los programas en el lado del cliente acelera la interacción en gran medida, comparado con la interacción de enviar los elementos al lado del servidor para su procesamiento.

5 JSP permite incluir código de forma más compleja, donde el código HTML se encuentra en una sentencia if-else (si-sino), y genera la salida de forma condicional dependiendo de si la condición se evalúa a cierto o falso.

**PHP**

PHP es un lenguaje de guiones en el lado del servidor muy utilizado. El código en PHP se puede mezclar con el código en HTML, de forma similar a JSP. Los caracteres «<?php» indican el comienzo de código en PHP, y los caracteres «?>» indican el fin del código en PHP. El siguiente ejemplo realiza las mismas acciones que el código JSP de la Figura 9.9.

```
<html>
<head> <title> Hola </title> </head>
<body>
  <?php if (!isset($ REQUEST['nombre']))
   { echo 'Hola mundo; }
   else { echo 'Hola, ' . $ REQUEST['nombre']; }
  ?>
</body>
</html>
```

El array $_REQUEST contiene los parámetros de la petición. Fíjese que el array usa como índice el parámetro nombre; en PHP los arrays se pueden indicar con cadenas arbitrarias y no solo con números. La función isset comprueba si el elemento del array está inicializado. La función echo escribe sus argumentos en la salida de HTML. El operador «.» entre las dos cadenas de caracteres concatena las cadenas de texto.

Un servidor web configurado apropiadamente interpreta cualquier archivo cuyo nombre termine en «.php» como un archivo PHP. Si se solicita este archivo, el servidor web lo procesa de forma similar a como se procesan los archivos JSP, y devuelve el código HTML generado al navegador.

Existen distintas bibliotecas disponibles para el lenguaje PHP, incluyendo bibliotecas para el acceso a bases de datos usando ODBC (similar a JDBC en Java).

Un peligro de este tipo de programas es que si el diseño del sistema no se ha realizado cuidadosamente, el código incluido en la página web (o equivalente en un mensaje de email), puede realizar acciones maliciosas en la computadora del usuario. Estas acciones maliciosas pueden ir desde leer información privada o borrar o modificar información de la computadora, hasta tomar el control de la computadora y propagar el código a otras computadoras (a través de email, por ejemplo). Varios virus de correo se han extendido ampliamente en los últimos años de esta forma.

Una de las razones de que el lenguaje Java se haya popularizado es que proporciona un modo seguro para la ejecución de programas en la computadora del usuario. El código en Java se puede compilar en una plataforma de «byte-code» independiente que se puede ejecutar en cualquier navegador que admita Java. Al contrario que otros programas, los programas Java (applets) se descargan como parte de la página web y no tienen capacidad de realizar ninguna acción que pueda ser destructiva. Se les permite mostrar datos en la pantalla o realizar una conexión de red al servidor desde donde se descargó la página web para obtener más información. Sin embargo, no se les permite el acceso a archivos locales, ni ejecutar programas del sistema, ni realizar conexiones de red a otras computadoras.

Mientras que Java es un lenguaje de programación completo, los hay más sencillos, llamados **lenguajes de guiones,** que pueden enriquecer la interacción del usuario y proporcionar, a la vez, la misma protección que Java. Estos lenguajes ofrecen construcciones que se pueden incluir en un documento HTML. Los **lenguajes de secuencias de comandos en el lado del cliente** son lenguajes diseñados para su ejecución en el navegador web del cliente.

De ellos, el lenguaje *JavaScript* es con mucho el más utilizado. La generación actual de interfaces web usa el lenguaje JavaScript intensivamente para construir interfaces de usuario sofisticadas. JavaScript se usa para múltiples tareas. Por ejemplo, se pueden usar funciones para comprobar errores (validación) de las entradas del usuario, como si una fecha tiene el formato apropiado, o si un determinado valor (como la edad) tiene un valor adecuado. Estas comprobaciones se realizan antes de enviar los datos al servidor web.

La Figura 9.10 muestra un ejemplo de función JavaScript utilizada para validar una entrada de formulario. La función se declara en la sección head del documento HTML. La función comprueba que los créditos introducidos para una asignatura es un número mayor que 0, y menor que 16. La etiqueta form especifica la función de validación que hay que invocar cuando el usuario envíe el formulario. Si la validación falla, se muestra un cuadro de alerta al usuario, y si es correcta el formulario se envía al servidor.

Se puede utilizar JavaScript para modificar dinámicamente el código HTML que se muestra. El navegador analiza el código HTML en la memoria en una estructura de árbol definida en una norma que se denomina **Modelo de Objetos de Documento** (DOM: *Document Object Model*). El código JavaScript puede modificar la estructura en árbol para llevar a cabo ciertas operaciones. Por ejemplo, suponga que un usuario necesita introducir un cierto número de datos, por ejemplo, varios elementos en una factura. Se puede utilizar para recoger la información una tabla que contenga cuadros de texto u otros métodos de entrada. La tabla puede tener un tamaño predeterminado, pero si se necesitan más filas, el usuario puede pulsar en un botón con la etiqueta, por ejemplo, «Añadir elemento». Este botón puede estar configurado para invocar a una función en JavaScript que modifica el árbol DOM para añadir una fila extra en la tabla.

Aunque el lenguaje JavaScript está normalizado, existen diferencias entre distintos navegadores, en particular en los detalles del modelo DOM. El resultado es que el código JavaScript que funciona en un navegador puede no funcionar en otro. Para evitar estos problemas, es mejor utilizar bibliotecas de JavaScript, como la biblioteca YUI de Yahoo, que permite escribir el código de forma independiente del navegador. Internamente, las funciones de la biblioteca pueden descubrir qué navegador se está utilizando y utilizar el código apropiado en el navegador. Véase la sección Herramientas al final del capítulo para más información sobre YUI y otras bibliotecas.

En la actualidad, JavaScript se utiliza ampliamente para crear páginas web dinámicas, junto con varias tecnologías que colectivamente se denominan **Ajax.** Los programas escritos en JavaScript se comunican con el servidor web de forma asíncrona, es decir, en segundo plano, sin bloquear la interacción del usuario con el navegador web, y pueden obtener información y mostrarla.

Como ejemplo de uso de Ajax, considere un sitio web con un formulario que permita seleccionar un país y, una vez seleccionado, se pueda elegir la provincia o estado de una lista de provincias o estados del país. Hasta que se seleccione el país, el desplegable de provincias está vacío. Ajax permite que la lista de provincias se descargue del servidor web en segundo plano, mientras se selecciona el país, y en cuanto la lista se ha conseguido, se añade a la lista desplegable y le permite al usuario seleccionar la provincia.

También existen lenguajes de secuencias de comandos de propósito especial para tareas especializadas, como las animaciones (por ejemplo, Flash y Shockwave) y el modelado en tres dimensiones (Lenguaje de Marcado de Realidad Virtual, VRML, Virtual Reality Markup Language). Flash se usa extensamente en la actualidad no solo para las animaciones, sino también para vídeo en streaming.

```
<html>
<head>
<script type="text/javascript">
   function validate() {
      var credits=document.getElementById("credits").value;
      if (isNaN(credits)|| credits<=0 || credits>=16) {
         alert("Los créditos debe ser un valor mayor que 0
         y menor que 16");
         return false
      }
   }
</script>
</head>

<body>
<form action="createCourse" onsubmit="return validate()">
   Título: <input type="text" id="title" size="20"><br />
   Créditos: <input type="text" id="credits" size="2"><br />
   <input type="submit" value="Enviar">
</form>
</body>
</html>
```

**Figura 9.10.** Ejemplo de JavaScript usado para validar un formulario de entrada de datos.

## 9.4. Arquitecturas de aplicación

Para gestionar la complejidad, las grandes aplicaciones se suelen dividir en varias capas:

- La capa de *presentación* o *interfaz de usuario*, que trata de la interacción con el usuario. Una misma aplicación puede tener distintas versiones de esta capa, que se corresponden con los distintos tipos de interfaces, como navegadores web e interfaces de usuario en teléfonos móviles, que tienen pantallas más pequeñas.

  En muchas implementaciones, la capa de presentación/interfaz de usuario se divide conceptualmente en capas, de acuerdo con la arquitectura **modelo-vista-controlador** (MVC). El **modelo** se corresponde con la capa de lógica de negocio, que se describe a continuación. La **vista** se corresponde con la presentación de los datos, un mismo modelo subyacente puede tener distintas vistas dependiendo del software o dispositivo que se use para acceder a la aplicación. El **controlador** recibe eventos (acciones del usuario), ejecuta acciones del modelo y devuelve una vista al usuario. La arquitectura MVC se usa en gran número de «frameworks» de aplicaciones web, que se tratan más adelante en la Sección 9.5.2.
- La capa de la **lógica de negocio** proporciona una vista de alto nivel y acciones con los datos. Se trata esta capa con más detalle en la Sección 9.4.1.
- La capa de **acceso a los datos** proporciona la interfaz entre la capa de la lógica de negocio y la base de datos subyacente. Muchas aplicaciones usan un lenguaje orientado a objetos para codificar la capa de la lógica de negocio, y usan un modelo de datos orientado a objetos, siendo la base de datos subyacente una base de datos relacional. En estos casos, la capa de acceso a los datos también proporciona una correspondencia entre el modelo de datos orientado a objetos que usa la lógica de negocio y el modelo relacional que usa la base de datos. Se trata esta correspondencia con más detalle en la Sección 9.4.2.

La Figura 9.11 muestra las capas citadas, junto con una secuencia de pasos del proceso de una solicitud desde el navegador web. Las etiquetas en las flechas de la figura indican el orden de los pasos. Cuando se recibe una solicitud del servidor de aplicaciones, el controlador envía una petición al modelo. El modelo procesa la solicitud, usando la lógica de negocio, que puede implicar la actualización de los objetos que forman parte del modelo, seguido de la creación de un objeto resultado.

El modelo después usa la capa de acceso a los datos para actualizar u obtener información de la base de datos. El objeto resultado que crea el modelo se envía al módulo de vista, que crea la vista en HTML del resultado, para que se muestre en el navegador web. La vista se puede adaptar de acuerdo con las características del dispositivo que se utilice para mostrar el resultado, por ejemplo, si se utiliza un monitor de computadora con una gran pantalla o una pantalla pequeña en un teléfono.

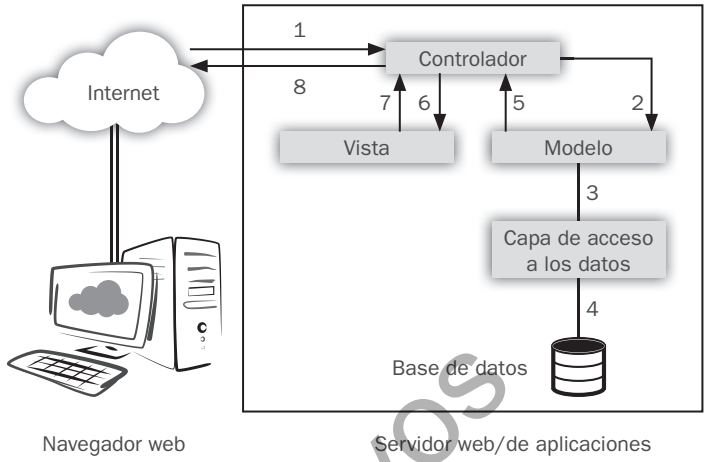



**Figura 9.11.** Arquitectura de aplicación web.

### 9.4.1. Capa de la lógica de negocio

La capa de la lógica de negocio de una aplicación para la gestión de una universidad puede proporcionar abstracciones de entidades como estudiantes, profesores, asignaturas, secciones, etc., y acciones como admitir a un estudiante en la universidad, matricular a un estudiante en una asignatura, etc. El código para implementar estas acciones asegura que las **reglas del negocio** se satisfacen; por ejemplo, el código debería asegurar que un estudiante se puede matricular de una asignatura solo si cumple todos los prerrequisitos y ha pagado sus tasas universitarias.

Además, la lógica de negocio incluye **flujos de trabajo** que describen cómo se gestiona una tarea que implica a varios partícipes. Por ejemplo, si un candidato opta a la universidad, existe un flujo de trabajo que define quién debería ver y aprobar la opción y si se aprueba en este primer paso, quién debería ver la opción a continuación, y así sucesivamente hasta que se le ofrece la aceptación al estudiante o se le envía una nota de rechazo. La gestión del flujo de trabajo necesita tratar con situaciones de error; por ejemplo, si no se cumplen los plazos para una aceptación/rechazo, puede que haya que informar a un supervisor para que intervenga y asegure que se procesa. Los flujos de trabajo se tratan con más detalle en la Sección 26.2.

### 9.4.2. La capa de acceso a los datos y la correspondencia entre objetos y relaciones

En el escenario más simple, en el que la capa de la lógica de negocio usa el mismo modelo de datos que la base de datos, la capa de acceso a los datos simplemente oculta los detalles de la interfaz con la base de datos. Sin embargo, cuando la capa de la lógica de negocios se escribe utilizando un lenguaje orientado a objetos, resulta natural modelar los datos como objetos, con métodos que se invocan sobre los objetos.

En las primeras implementaciones, los programadores tenían que escribir un código para crear los objetos recuperando los datos de la base de datos, así como para guardar las actualizaciones en los objetos de vuelta a la base de datos. Sin embargo, estas conversiones manuales entre los modelos de datos son muy engorrosas y propensas a errores. Una forma de manejar este problema fue desarrollar un sistema de base de datos que guardaba de forma nativa objetos y relaciones entre los objetos. Estas bases de datos, denominadas **bases de datos orientadas a objetos,** se tratan con más detalle en el Capítulo 22. Sin embargo, estas bases de datos no alcanzaron un gran éxito comercial por varias razones técnicas y comerciales.

Una alternativa es usar bases de datos relacionales tradicionales para guardar los datos, pero para automatizar la correspondencia entre los datos de las relaciones con los objetos en memoria, que se crean bajo demanda (y ya que la memoria normalmente no es suficiente para almacenar todos los datos de la base de datos), así como la correspondencia inversa para guardar los objetos actualizados de nuevo como relaciones en la base de datos.

Se han desarrollado varios sistemas para implementar esta **correspondencia entre objetos y relaciones.** El sistema **Hibernate** se usa extensamente para la correspondencia entre objetos de Java y relaciones. En Hibernate, la correspondencia entre una clase de Java y una o más relaciones se especifica en un archivo de correspondencias. Este archivo puede indicar, por ejemplo, que la clase de Java de nombre *Estudiante* se hace corresponder con la relación *estudiante*, donde el atributo *ID* de Java se hace corresponder al atributo *estudiante.ID*, y así sucesivamente. La información sobre la base de datos, como la máquina en la que se ejecuta, y el nombre de usuario y contraseña para la conexión a la base de datos, etc., se especifican en un archivo de *propiedades*. El programa tiene que abrir una *sesión*, que establece la conexión con la base de datos. Cuando se establece la conexión, un objeto *est* de la clase *Estudiante* creado en Java se puede guardar en la base de datos invocando a `session.save(est)`. El código Hibernate genera los comandos SQL necesarios para guardar los datos correspondientes a *estudiante* en la relación.

Se puede obtener una lista de los objetos de la base de datos ejecutando una consulta escrita en el lenguaje de consultas Hibernate; es similar a ejecutar una consulta usando JDBC, que devuelve un `ResultSet` con un conjunto de tuplas. De forma alternativa, se puede obtener un único objeto proporcionando su clave primaria. Los objetos obtenidos se pueden modificar en memoria; cuando la transacción de la sesión de Hibernate sigue activa y se confirma, Hibernate guarda automáticamente los objetos actualizados realizando las correspondientes actualizaciones en las relaciones de la base de datos.

Mientras que las entidades del modelo E-R se corresponden con objetos en un lenguaje orientado a objetos como es Java, las relaciones no. Hibernate tiene la capacidad de hacer corresponder tales relaciones como conjuntos asociados con objetos. Por ejemplo, la relación *matricula* entre *estudiante* y *sección* se puede modelar asociando un conjunto de *secciones* con cada *estudiante* y un conjunto de *estudiantes* con cada *sección*. Una vez especificada la correspondencia, Hibernate puebla estos conjuntos de forma automática a partir de la relación *matricula* de la base de datos, y las actualizaciones en los conjuntos se refleja de vuelta en la base de datos cuando se realiza una confirmación (*commit*).

La característica anterior ayuda a proporcionar al programador un modelo de alto nivel de los datos sin preocuparse por los detalles del almacenamiento relacional. Sin embargo, Hibernate como otros sistemas de correspondencia entre objetos y relacional, también permite que los programadores puedan acceder directamente al SQL de las relaciones subyacentes. Este acceso directo resulta particularmente importante para escribir consultas complejas necesarias en la generación de informes.

Microsoft ha desarrollado un modelo de datos, llamado **Modelo Dato-Entidad,** que se puede ver como una variante del modelo entidad-relación y un «framework» asociado, llamado ADO.NET Entity Framework, que puede hacer la correspondencia entre datos del Modelo Dato-Entidad y la base de datos relacional. El «framework» también proporciona un lenguaje tipo-SQL llamado **Entity SQL,** que funciona directamente sobre el Modelo Datos-Entidad.

**EJEMPLO DE HIBERNATE**

Como ejemplo del uso de Hibernate, se crea una clase Java de la relación *estudiante* de la siguiente forma:

```
public class Estudiante {
    String ID;
    String nombre;
    String departamento;
    int tot_cred;
    Estudiante(String id, String nombre, String dept, int totcreds);
    // constructor
}
```

Para ser precisos, los atributos de la clase se deberían declarar privados y proporcionar métodos get/set para acceder a los atributos, pero se omiten estos detalles.

La correspondencia de los atributos de la clase Estudiante en atributos de la relación *estudiante* se especifica en el archivo de correspondencias, escrito en XML. De nuevo se omiten los detalles.

El siguiente fragmento de código crea un objeto Estudiante y lo guarda en la base de datos.

```
Session sesion = getSessionFactory().openSession();
Transaction txn = sesion.beginTransaction();
Student est = new Estudiante("12328", "John Smith",
                             "Informática", 0);
sesion.save(est);
txn.commit();
sesion.close();
```

Hibernate genera automáticamente las sentencias **insert** de SQL necesarias para crear una tupla *estudiante* en la base de datos.

Para obtener los estudiantes, se puede usar el siguiente fragmento de código:

```
Session sesion = getSessionFactory().openSession();
Transaction txn = sesion.beginTransaction();
List estudiantes =
 sesion.find("from Estudiante as e order by e.ID asc");
for ( Iterator iter = estudiantes.iterator(); iter.hasNext(); ) {
 Estudiante est = (Estudiante) iter.next();
 … escribir la información del Estudiante …
}
txn.commit();
sesion.close();
```

El fragmento de código anterior usa una consulta en el lenguaje de consulta HQL de Hibernate. La consulta en HQL se traduce automáticamente a SQL y se ejecuta, y el resultado se convierte en una lista de objetos *Estudiante*. El bucle for itera sobre los objetos de esta lista y los escribe por pantalla.

### 9.4.3. Servicios web

En el pasado, la mayoría de las aplicaciones web solo usaban datos disponibles en el servidor de aplicaciones y su base de datos asociada. Desde hace unos años existe una amplia variedad de datos disponibles en la web que se pretende se procesen por programas, en lugar de mostrarse directamente al usuario; el programa puede estar dándose como parte de una aplicación de back-end, o puede ser un guion ejecutándose en el navegador. A estos datos se accede normalmente utilizando una interfaz de programación de aplicaciones web, es decir, se envía una solicitud de llamada a una función utilizando el protocolo HTTP; se ejecuta en un servidor de aplicaciones y los resultados se envían de vuelta al programa que hizo la llamada. Un sistema con este tipo de interfaz se llama **servicio web.**

Se utilizan dos formas para implementar servicios web. En la forma más sencilla, llamada **transferencia del estado de representación** (REST: *Representation State Transfer*), las llamadas a funciones del servicio web se ejecutan usando una solicitud estándar de HTTP a una URL de un servidor de aplicaciones con parámetros enviados como parámetros estándar de solicitudes de HTTP. El servidor de aplicaciones ejecuta la solicitud, lo que puede implicar actualizar la base de datos en el servidor, genera y codifica el resultado y devuelve este como resultado a una solicitud de HTTP. El servidor puede usar cualquier codificación para una determinada URL solicitada; se usan ampliamente XML y una codificación de objetos de JavaScript denominada **notación JavaScript de objetos** (JSON: *JavaScript Object Notation*). El solicitante analiza la página de respuesta para acceder a los datos devueltos.

En muchas aplicaciones de servicios web RESTful, es decir, servicios web que usan REST, el solicitante es un código en JavaScript ejecutado en un navegador web; el código actualiza la pantalla del navegador usando el resultado de la llamada a la función. Por ejemplo, cuando desea desplazarse por la interfaz de un mapa en la web, la parte del mapa que es necesario mostrar como nueva se puede solicitar desde código en JavaScript usando una interfaz RESTful y después mostrarla en la pantalla.

Una forma más compleja, a veces denominada «Grandes Servicios Web» («Big Web Services»), usa una codificación XML de los parámetros y de los resultados, tiene definiciones formales de la API de web usando un lenguaje especial, y usa una capa de protocolo construida encima del protocolo HTTP. Esta forma se describe con más detalle en la Sección 23.7.3.

### 9.4.4. Operación desconectado

Muchas aplicaciones necesitan seguir funcionando incluso aunque el cliente esté desconectado del servidor de aplicaciones. Por ejemplo, un estudiante que desea rellenar un formulario aunque su portátil esté desconectado de la red, y poder guardarlo para cuando se vuelva a conectar. Otro ejemplo, si un cliente de correo electrónico se ha construido como aplicación web, el usuario puede desear escribir un correo incluso aunque su portátil este desconectado de la red, y que se envíe cuando vuelva a conectarse. La construcción de este tipo de aplicaciones requiere un almacenamiento local, preferiblemente en forma de base de datos. El software **Gears,** desarrollado inicialmente por Google, es una extensión del navegador que proporciona una base de datos, un servidor web local, y permite la ejecución paralela de JavaScript en el cliente. El software funciona de forma idéntica en distintas plataformas y sistemas operativos, permitiendo a las aplicaciones disponer de una rica funcionalidad sin necesidad de instalar ningún software adicional, distinto del propio Gears. El software AIR de Adobe también proporciona una funcionalidad similar para la construcción de aplicaciones de Internet que se pueden ejecutar fuera de un navegador web.

## 9.5. Desarrollo rápido de aplicaciones

Si las aplicaciones web se construyen sin utilizar herramientas ni bibliotecas para la generación de interfaces de usuario, el esfuerzo de programación necesario para crear estas interfaces de usuario puede ser significativamente mayor que el que se necesita para la lógica de negocio y el acceso a la base de datos. Se han desarrollado varias estrategias para reducir el esfuerzo requerido para la construcción de aplicaciones:

- Proporcionar una biblioteca de funciones para generar los elementos de la interfaz de usuario con una programación mínima.
- Proporcionar funciones de arrastrar en un entorno integrado de desarrollo que permita arrastrar los elementos de la interfaz de usuario desde un menú a una vista de diseño. El entorno integrado de desarrollo genera el código que crea la interfaz de usuario invocando funciones de biblioteca.
- Generar automáticamente el código de la interfaz de usuario a partir de una especificación declarativa.

Todas estas formas se han utilizado para la creación de interfaces gráficas de usuario, antes de que existiese la web, formando parte de las herramientas de **desarrollo rápido de aplicaciones** (RAD: *Rapid Application Development*) y ahora también se usan ampliamente para la creación de aplicaciones web.

Entre los ejemplos de aplicaciones para el desarrollo rápido de interfaces para aplicaciones de bases de datos se encuentran Oracle Forms, Sybase PowerBuilder y Oracle Application Express (APEX). Además, aplicaciones diseñadas para el desarrollo de aplicaciones web, como Visual Studio y Netbeans VisualWeb, disponen de características para el desarrollo rápido de interfaces web para las aplicaciones con bases de datos.

En la Sección 9.5.1 se tratan las herramientas para la construcción de interfaces de usuario y en la Sección 9.5.2 se ven los «frameworks» que permiten la generación automática del código a partir de un modelo.

### 9.5.1. Herramientas para la construcción de interfaces de usuario

Muchas construcciones de HTML se generan mejor si se utilizan funciones definidas apropiadamente, en lugar de escribirlas como parte del código de las páginas web. Por ejemplo, los formularios de direcciones suelen requerir un menú con los países y sus provincias o estados. En lugar de escribir cada vez el código HTML necesario para crear este menú, es preferible definir una función que genere el menú y llamarla siempre que se necesite.

Los menús se suelen generar preferiblemente a partir de datos que se encuentran en una base de datos, como una tabla con los nombres de los países y de las provincias o estados. La función que genera el menú ejecuta una consulta a la base de datos y rellena el menú usando el resultado de la consulta. Añadir un país o una provincia o estado solo requiere actualizar la base de datos, y no el código de la aplicación. Tiene la desventaja de requerir una mayor interacción con la base de datos, pero esta sobrecarga se puede minimizar haciendo caché de los resultados de la consulta en el servidor de aplicaciones.

De forma similar, para los formularios de introducción de fechas y horas o para las entradas que requieren validación de datos es preferible generarlos llamando a funciones apropiadas. Estas funciones pueden generar códigos en JavaScript que realizan la validación en el navegador.

Una tarea habitual de muchas aplicaciones de base de datos es la presentación de un conjunto de resultados. Es posible construir una función genérica que a partir de una consulta SQL (o un ResultSet), como argumento, muestre las tuplas del resultado de la consulta (o del ResultSet), en forma de tabla. Se pueden utilizar llamadas a metadatos de JDBC para disponer de información como el número de columnas y el nombre y tipos de las columnas del resultado de la consulta; esta información se utiliza para mostrar apropiadamente el resultado.

Para tratar las situaciones en las que el resultado de una consulta es muy grande, estas funciones para mostrar el resultado pueden realizar también la *paginación* del mismo. La función puede mostrar solo un cierto número de registros en una página y proporcionar controles para pasar a la página siguiente o anterior, o ir a una determinada página de resultados.

Por desgracia, no existe una API estándar de Java, que se use extensamente, para funciones del tipo indicado anteriormente. La construcción de este tipo de bibliotecas puede ser un proyecto interesante de programación.

Sin embargo, existen herramientas como el framework JavaServer Faces (JSF), que dispone de las características mencionadas anteriormente. El framework JSF incluye una biblioteca de etiquetas JSP que implementa estas características. El IDE Netbeans tiene un componente llamado VisualWeb que permite construir con JSF, mediante un entorno visual de desarrollo en el que los componentes de la interfaz de usuario se pueden arrastrar en la página y personalizar sus propiedades.

Por ejemplo, JSF tiene componentes para crear menús desplegables, o mostrar una tabla que se puede configurar para obtener los datos de una consulta a una base de datos.

JSF también permite especificar la validación para componentes, por ejemplo, para que una selección o entrada sea obligatoria o para restringir los valores de una fecha dentro de un determinado intervalo.

Active Server Pages (ASP) de Microsoft, y su versión más reciente, Active Server Pages.NET(ASP.NET), es una alternativa a JSP/Java muy utilizada. ASP.NET es similar a JSP en que el código en un lenguaje como Visual Basic o C# se puede incorporar al código HTML. Además, ASP.NET proporciona muchos controles, que se interpretan en el servidor, y generan HTML que se envía al cliente. Estos controles pueden simplificar significativamente la construcción de interfaces web. Posteriormente se dará una descripción general de las ventajas de estos controles.

Por ejemplo, los controles como los menús desplegables y las listas se pueden asociar a un objeto DataSet. Los objetos DataSet son similares a los objetos ResultSet de JDBC, y se suelen crear para ejecutar una consulta a una base de datos. El contenido del menú en HTML se genera con el contenido del objeto DataSet; por ejemplo, una consulta puede obtener el nombre de todos los departamentos de una organización en un DataSet y el menú asociado debería contener esos nombres. Por tanto, el menú que dependa del contenido de la base de datos se puede crear de esta manera de forma muy conveniente con muy poca programación.

Se pueden añadir controles de validación a los campos de entrada del formulario; mediante una especificación declarativa se pueden restringir intervalos de valores, o si la entrada es obligatoria y el usuario tiene que introducir un valor. El servidor crea el código HTML apropiado combinado con JavaScript para realizar la validación en el navegador del usuario. Los mensajes de error que se muestran para cada entrada no válida se asocian en cada control de validación.

Se puede especificar que las acciones de usuario generen una acción asociada en el servidor. Por ejemplo, se puede especificar que cuando se seleccione una opción de menú genere una acción asociada en el lado del servidor (se genera código JavaScript para

detectar el evento de selección e iniciar la acción en el lado del servidor). El código en VisualBasic/C# que muestra los datos pertenecientes al valor seleccionado se pueden asociar con la acción en el lado del servidor. Por tanto, seleccionar una opción de un menú puede hacer que se actualicen los datos que se muestran en la página sin necesidad de que el usuario haga clic en un botón de enviar.

El control DataGrid proporciona un mecanismo muy apropiado para mostrar los resultados de una consulta. Un DataGrid se asocia con un objeto DataSet, que suele ser el resultado de una consulta. El servidor genera el código HTML que muestra el resultado de la consulta en forma de tabla. Las cabeceras de las columnas se generan automáticamente a partir de los metadatos del resultado. Además, los DataGrid proporcionan funciones como la paginación y permiten al usuario ordenar los resultados por columnas. Todo el código HTML, así como la funcionalidad en el lado del servidor para implementar estas funciones, se genera automáticamente en el servidor. El DataGrid también permite que el usuario pueda editar los datos y enviar los cambios de vuelta al servidor. El desarrollador puede especificar una función que se ejecuta cuando se edita una fila, lo que permite actualizar la base de datos.

Microsoft Visual Studio proporciona una interfaz gráfica de usuario para la creación de estas funciones, reduciendo en gran medida el esfuerzo de programación.

Véanse las notas bibliográficas para más información sobre ASP.NET.

### 9.5.2. Frameworks para aplicaciones web

Existen varios frameworks de desarrollo de aplicaciones web que proporcionan funciones muy utilizadas como:

- Un modelo orientado a objetos con una correspondencia entre objetos y relaciones para almacenar los datos en una base de datos relacional, como ya se vio en la Sección 9.4.2.
- Un mecanismo, relativamente declarativo, para especificar un formulario con restricciones de validación para las entradas del usuario, a partir del cual el sistema genera código HTML y JavaScript/Ajax para implementar el formulario.
- Un sistema de plantillas de guiones (similar a JSP).
- Un controlador que hace corresponder los eventos de interacción de usuario, como los envíos de formularios, con las funciones apropiadas para manejar los eventos. El controlador también maneja la autenticación y las sesiones. Algunos frameworks también proporcionan herramientas para la gestión de autorizaciones.

Estos frameworks proporcionan diversas características necesarias para la construcción de aplicaciones web de forma integrada. Al generar formularios a partir de especificaciones declarativas, y manejar el acceso a los datos de forma transparente, el framework minimiza la cantidad de código que el programador tiene que escribir para realizar la aplicación web.

Existe un gran número de este tipo de frameworks, para distintos lenguajes. Algunos de los más utilizados incluyen Ruby on Rails, basado en el lenguaje de programación Ruby, JBoss Seam, Apache Struts, Swing, Tapestry y WebObjects, todos ellos basados en Java/JSP. Algunos de ellos, como Ruby on Rails y JBoss Seam proporcionan una herramienta para crear automáticamente interfaces de usuario **CRUD** simples; es decir, interfaces que permiten crear, leer, actualizar y borrar objetos/tuplas, generando código a partir de un modelo de objetos o de una base de datos. Estas herramientas son particularmente útiles para empezar a generar aplicaciones simples de forma muy rápida, y el código generado se puede editar para construir interfaces web más sofisticadas.

### 9.5.3. Generadores de informes

Los generadores de informes son herramientas para generar informes legibles a partir de las bases de datos. Integran la consulta a la base de datos con la creación de texto con formato y gráficos-resumen (como los gráficos de barras o de tarta). Por ejemplo, un informe puede mostrar las ventas totales de los últimos dos meses para cada región de ventas.

El desarrollador de aplicaciones puede especificar el formato de los informes mediante los servicios de formato del generador de informes. Se pueden usar variables para almacenar parámetros como el mes y el año, así como para definir los campos del informe. Las tablas, gráficos de líneas, gráficos de barras u otros gráficos se pueden definir mediante consultas a la base de datos. Las definiciones de las consultas pueden hacer uso de los valores de los parámetros almacenados en las variables.

Una vez definida la estructura de los informes en un servicio generador de informes, se puede almacenar y ejecutar en cualquier momento para generar informes. Los sistemas generadores de informes ofrecen gran variedad de servicios para estructurar el resultado tabular, como la definición de encabezados de tablas y de columnas, la visualización de los subtotales correspondientes a cada grupo de la tabla, la división automática de las tablas de gran tamaño en varias páginas y la visualización de los subtotales al final de cada página.

La Figura 9.12 es un ejemplo de informe con formato. Los datos del informe se generan mediante la agregación de la información sobre los pedidos.

Gran variedad de marcas, como Crystal Reports y Microsoft (SQL Server Reporting Services), ofrecen herramientas para la generación de informes. Varias familias de aplicaciones, como Microsoft Office, incluyen procedimientos para incrustar directamente en los documentos los resultados con formato de las consultas a la base de datos. Los servicios de generación de gráficos proporcionados por Crystal Reports, o por hojas de cálculo como Excel se pueden usar para tener acceso a las bases de datos y generar descripciones tabulares o gráficas de los datos. En los documentos de texto creados con Microsoft Word, por ejemplo, se puede incrustar uno o más de estos gráficos. Los gráficos se crean inicialmente a partir de datos generados ejecutando consultas a la base de datos; las consultas se pueden volver a ejecutar y los gráficos se pueden regenerar cuando sea necesario, para crear una versión actual del informe global.

Además de generar los informes estáticos, estas herramientas permiten que sean interactivos. Así, un usuario puede «profundizar» en áreas de su interés, por ejemplo, pasar de una vista agregada que muestre las ventas totales por año completo a las cifras de ventas mensuales en un año concreto. El análisis interactivo de datos ya se trató anteriormente en la Sección 5.6.

**Acme Supply Company, Inc.**
**Informe de ventas trimestrales**

Periodo: 1 de enero a 31 de marzo, 2012

| Región | Categoría | Ventas | Subtotal |
|---|---|---|---|
| Norte | Hardware de computadora | 1.000.000 | |
| | Software de computadora | 500.000 | |
| | Todas las categorías | | 1.500.000 |
| Sur | Hardware de computadora | 200.000 | |
| | Software de computadora | 400.000 | |
| | Todas las categorías | | 600.000 |
| | | **Ventas totales** | 2.100.000 |

**Figura 9.12.** Un informe con formato.

## 9.6. Rendimiento de la aplicación

Puede que millones de personas de todo el mundo accedan a un determinado sitio web, a tasas de miles de peticiones por segundo o incluso mayores, en los sitios más populares. Asegurar que las peticiones se sirven con un tiempo de respuesta bajo es un gran desafío para los desarrolladores web. Para ello, los desarrolladores intentan acelerar el procesamiento de cada petición usando técnicas como el caché, el procesamiento paralelo o utilizando múltiples servidores web. A continuación se describen brevemente estas técnicas. El ajuste de las aplicaciones de bases de datos se describe con más detalle más adelante en el Capítulo 24 (Sección 24.1).

### 9.6.1. Reducir la sobrecarga mediante cachés

Se utilizan distintos tipos de técnicas de caché para aprovechar las partes comunes de diversas transacciones. Por ejemplo, suponga el código de la aplicación para servir las necesidades de cada petición de conectarse a la base de datos con JDBC. La creación de una nueva conexión JDBC puede tardar varios milisegundos, por lo que abrir una nueva conexión para cada petición de un usuario no es una buena idea si las tasas de transacciones son muy altas.

El método de **bancos de conexiones** se usa para reducir esta sobrecarga; funciona de la siguiente forma: el gestor del banco de conexiones, una parte del servidor de aplicaciones, crea un banco, es decir, un conjunto de conexiones ODBC/JDBC abiertas. En lugar de abrir una nueva conexión a la base de datos, el código que da el servicio a una petición de usuario, normalmente un servlet, solicita una conexión del banco de conexiones y devuelve la conexión al banco cuando el código, del servlet, termina su procesamiento. Si el banco no dispone de conexiones sin usar cuando se solicita, se abre una nueva conexión a la base de datos, con cuidado de no exceder el número máximo de conexiones que el sistema de la base de datos admite concurrentemente. Si durante un periodo de tiempo hay muchas conexiones abiertas sin usarse, el gestor del banco de conexiones puede cerrar algunas de ellas. Muchos servidores de aplicaciones y los nuevos controladores de ODBC/JDBC proporcionan un gestor de bancos de conexiones.

Un error común que cometen muchos programadores cuando crean aplicaciones web es olvidarse de cerrar una conexión JDBC abierta, o de forma equivalente cuando se usa un banco de conexiones, olvidarse de devolver la conexión al banco. Cada petición abre una nueva conexión a la base de datos, y la base de datos llega al límite del número de conexiones que puede tener abiertas a la vez. Estos problemas no aparecen cuando se prueba a pequeña escala, ya que las bases de datos suelen permitir cientos de conexiones abiertas, pues aparecen solo durante un uso intensivo. Algunos programadores asumen que las conexiones, como la memoria que se pide en los programas Java, se recolectan automáticamente. Desafortunadamente, no es así, y los programadores son los responsables de cerrar las conexiones que han abierto.

Ciertas peticiones pueden crear una consulta que ya se había enviado a la base de datos. El coste de comunicarse con la base de datos se puede reducir de forma importante haciendo un caché de los resultados de peticiones anteriores y reutilizándolos, mientras los resultados de la consulta no cambien en la base de datos. Algunos servidores web admiten este tipo de caché de resultados, sino se puede realizar explícitamente en el código de la aplicación.

El coste se puede reducir aún más haciendo caché de la página web final que se envía como respuesta a una petición. Si una nueva petición viene con los mismos parámetros que una previa, la solicitud no realiza ninguna actualización y la página web resultante está en la caché, por lo que se puede reutilizar y evitar el coste de volver a recalcular la página. El caché se puede utilizar en el nivel de fragmentos de páginas web, que se ensamblan para crear páginas web completas.

El caché de resultados de consultas y el de páginas web son formas de vistas materializadas. Si los datos de la base de datos subyacente cambian, los resultados en caché deben ser descartados o recalculados, o actualizados incrementalmente como en el mantenimiento de vistas materializadas (se describe más adelante en la Sección 13.5). Algunos sistemas de bases de datos, como Microsoft SQL Server, proporcionan una forma para que el servidor de aplicaciones registre una consulta en la base de datos y obtenga una **notificación** de la base de datos cuando el resultado de la consulta cambie. Este mecanismo de notificaciones se puede usar para garantizar que los resultados de una consulta en caché en el servidor de aplicaciones siempre están actualizados.

### 9.6.2. Procesamiento paralelo

Una forma habitual de manejar una gran carga es utilizar un gran número de servidores de aplicación ejecutándose en paralelo, cada uno de ellos manejando una fracción de las peticiones. Se puede usar un servidor web o un router de red para encaminar las peticiones de los clientes a los distintos servidores de aplicaciones. Todas las peticiones de una sesión de un determinado cliente deben ir al mismo servidor de aplicaciones, ya que el servidor mantiene el estado de una sesión del cliente. Esta propiedad se puede asegurar, por ejemplo, encaminando todas las peticiones de una determinada dirección de IP al mismo servidor de aplicaciones, de forma que los usuarios vean una vista consistente de la base de datos.

Con la arquitectura anterior, la base de datos se podría convertir en un cuello de botella, al ser compartida. Los diseñadores de aplicaciones ponen una atención especial en minimizar el número de peticiones a la base de datos mediante el caché de consultas en el servidor de aplicaciones, como ya se ha tratado. Los sistemas de bases de datos paralelos se describen en el Capítulo 18.

## 9.7. Seguridad de las aplicaciones

La seguridad de las aplicaciones tiene que tratar con distintos frentes de seguridad y temas que van más allá de los que manejan las autorizaciones de SQL.

El primer punto en el que reforzar la seguridad es en la aplicación. Para ello, las aplicaciones deben autenticar a los usuarios y asegurar que solo se les permite realizar las tareas autorizadas.

Hay muchas formas en que se puede ver comprometida la seguridad de una aplicación, incluso aunque el sistema de base de datos sea seguro, debido a una aplicación mal escrita. En esta sección, en primer lugar se describen algunos agujeros de seguridad que pueden permitir que los hackers lleven a cabo acciones saltándose las comprobaciones de autenticación y autorización de la aplicación, y se explica cómo evitar estos agujeros de seguridad. Más adelante en esta sección se describen técnicas para la autenticación segura y para las autorizaciones de grano fino. Posteriormente se describen las trazas de auditoría que pueden ayudar a recuperarse de un acceso no autorizado y de actualizaciones erróneas. La sección concluye describiendo algunos elementos de privacidad de los datos.

### 9.7.1. Inyección SQL

En los ataques de **inyección SQL,** el atacante consigue que una aplicación ejecute una consulta SQL creada por el atacante. En la Sección 5.1.1.4 se vio un ejemplo de vulnerabilidad de inyección SQL si las entradas del usuario se concatenaban directamente con una consulta SQL y se enviaban a la base de datos. Como ejemplo

adicional de inyección SQL, considere el formulario que se muestra en la Figura 9.4. Suponga que el servlet que se muestra en la Figura 9.8 crea una cadena de consulta en SQL que usa la siguiente expresión en Java:

```
String consulta = «select * from estudiante where nombre like '%"
                  + nombre + "%'»
```

donde nombre es una variable que contiene la entrada del usuario, y se ejecuta la consulta en la base de datos. Un atacante malicioso podría usar el formulario web y escribir una cadena de texto como «';<una sentencia de SQL >; −− », donde <una sentencia de SQL> indica cualquier sentencia de SQL que el atacante desee, en lugar de un nombre válido de estudiante. El servlet ejecutaría entonces la siguiente consulta:

```
select * from estudiante where nombre like ' ';
                  <una sentencia de SQL >;−−'
```

La comilla insertada por el atacante cierra la cadena, el siguiente punto y coma termina la consulta y el texto que hay a continuación, que inserta el atacante, se interpreta como una segunda consulta de SQL, y la comilla de cierre se ha comentado. Por tanto, el malicioso usuario ha conseguido insertar una sentencia de SQL arbitraria que ejecuta la aplicación. La sentencia puede causar un daño importante, ya que puede realizar cualquier acción en la base de datos, saltándose todas las medidas de seguridad implementadas en el código.

Como ya se trató en la Sección 5.1.1.4, para evitar estos ataques es mejor utilizar sentencias ya preparadas para ejecutar las consultas de SQL. Cuando se configura un parámetro en una consulta preparada, JDBC añade automáticamente caracteres de escape de forma que las comillas que incluya el usuario ya no permiten terminar una cadena. De forma similar, se podría aplicar una función que añada estos caracteres de escape a las cadenas de entrada antes de que se concatenen con las consultas SQL, en lugar de usar sentencias preparadas.

Otra fuente de riesgo de inyección SQL proviene de las aplicaciones que crean las consultas de forma dinámica a partir de condiciones de selección y ordenación de los atributos especificados en un formulario. Por ejemplo, una aplicación permite al usuario especificar qué atributos se deberían utilizar para ordenar los resultados de una consulta. Se construye la consulta apropiada en SQL de acuerdo con los atributos especificados. Suponga que la aplicación recoge el nombre del atributo del formulario, en la variable ordenAtributo y crea una cadena de consulta de la forma:

```
String consulta = «select * from takes order by» + ordenAtributo;
```

Un usuario malicioso puede enviar una cadena arbitraria en lugar de un valor de ordenAtributo con sentido, incluso aunque se use el formulario HTML para obtener la entrada y restringir los valores que proporcione el menú. Para evitar este tipo de inyección SQL, la aplicación debería asegurarse de que el valor de la variable ordenAtributo es uno de los valores permitidos (en el ejemplo, los nombres de los atributos) antes de concatenarlo.

### 9.7.2. Secuencias de comandos en sitios cruzados y falsificación de peticiones

Un sitio web que permite a los usuarios introducir texto, como un comentario o un nombre, y después lo almacena para mostrarlo más tarde a otros usuarios es potencialmente vulnerable a un tipo de ataque llamado **secuencias de comandos en sitios cruzados** (XXS: *cross-site scripting*). En estos ataques, un usuario malicioso introduce código escrito en el lenguaje de secuencias de comandos como JavaScript o Flash, en lugar de un nombre válido o un comentario. Cuando un usuario diferente ve el texto introducido, el navegador puede ejecutar una secuencia de comandos y llevar a cabo acciones como el envío de información de cookies privadas de vuelta al usuario malicioso o, incluso, ejecutar una acción en un servidor web diferente a aquel en el que el usuario se había registrado.

Por ejemplo, suponga que el usuario consigue registrarse en su cuenta del banco cuando se ejecuta la secuencia de comandos. Podría enviar la información de la cookie relacionada con el registro en la cuenta del banco de vuelta al usuario malicioso, quien podría usar la información para conectarse al servidor web del banco, engañándolo y haciéndole creer que es una conexión del usuario original. O, el guion podría acceder a las páginas apropiadas del sitio web del banco, con el conjunto de parámetros apropiados, y ejecutar transferencias de dinero. De hecho este problema en particular puede ocurrir incluso sin secuencias de comandos con una simple línea de código como:

```
<img src=
  "http://mybank.com/transfermoney?amount=
   1000&toaccount=14523">
```

suponiendo que la URL mybank.com/transfermoney acepta los parámetros especificados y lleva a cabo la transferencia de dinero. Esta última vulnerabilidad también se llama **falsificación de petición en sitios cruzados** o **XSRF** (también denominada **CSRF**).

XSS se puede realizar de otras formas como tentando al usuario a visitar un sitio web con secuencias de comandos maliciosas en sus páginas. Existen otros tipos de ataques XSS y XSRF más complejos, que no se van a tratar. Para protegerse contra estos ataques se deben tener en cuenta dos recomendaciones:

- **Evitar que un sitio web se use para lanzar ataques XSS o XSRF.** La forma más sencilla es impedir que los usuarios puedan escribir cualquier tipo de etiqueta HTML. Existen funciones para detectar o eliminar tales etiquetas. Estas funciones se pueden utilizar para evitar etiquetas HTML y por tanto impedir que cualquier secuencia de comandos se muestre al usuario. En algunos casos el formato de HTML es útil y en tal caso las funciones que analizan el texto permiten un número limitado de etiquetas impidiendo otras que pueden ser peligrosas; hay que diseñarlo cuidadosamente, ya que algo tan inocente como un programa para mostrar imágenes puede explotarse maliciosamente.
- **Proteger el sitio web de ataques XSS o XSRF que se lanzan desde otros sitios.** Si el usuario se ha registrado en un sitio web y visita otro diferente vulnerable a XSS, el código malicioso que se ejecuta en el navegador del usuario podría ejecutar acciones en tu sitio web, o pasar información de sesión relacionada con tu sitio web de vuelva al usuario malicioso que tratará de explotarla. Esto no se puede prevenir, pero se pueden tomar algunas medidas para minimizar los riesgos.
  - El protocolo HTTP permite que un servidor compruebe el **referente** de un acceso de una página, es decir, la URL de la página que tiene el enlace en el que el usuario ha hecho clic para iniciar el acceso a la página. Comprobando que el referente es válido, por ejemplo, que tiene una URL del mismo sitio web, se pueden prevenir los ataques XSS originados desde una página web diferente a la que accede el usuario.
  - En lugar de utilizar solo las cookies para identificar una sesión, también se puede restringir la dirección IP desde la que se realizó la autenticación originalmente. Con ello, aunque un usuario malicioso consiga una cookie, no podría utilizarla para acceder desde una computadora diferente.
  - No utilice nunca el método GET para realizar las actualizaciones. Esto evita los ataques que utilizan <img src…>, como el que se vio anteriormente. De hecho, la norma HTTP recomienda que los métodos GET nunca realicen actualizaciones, por otras razones como que el refresco de una página repite una acción que solo debería producirse una vez.

### 9.7.3. Fuga de contraseñas

Otro problema que deben tratar los desarrolladores de aplicaciones es el almacenamiento de las contraseñas en texto claro en el código de la aplicación. Por ejemplo, programas de secuencia de guiones como JSP suelen incluir contraseñas en texto claro. Si estos programas se guardan en un directorio accesible al servidor web, un usuario externo puede ser capaz de acceder al código fuente del programa y conseguir acceso a la contraseña de la cuenta de la base de datos que usa la aplicación. Para evitar estos problemas, muchos servidores de aplicación proporcionan mecanismos para guardar las contraseñas de forma cifrada, que el servidor descifra antes de pasarlas a la base de datos. Esta característica elimina la necesidad de almacenar las contraseñas en texto claro en los programas de aplicación. Sin embargo, si la clave de cifrado sigue siendo vulnerable, este mecanismo puede no ser muy efectivo.

Otra medida para no comprometer la contraseña de la base de datos es que muchos sistemas de base de datos permiten que el acceso a esta se restrinja a un intervalo de direcciones IP, normalmente las máquinas en las que se ejecutan los servidores de aplicaciones. El intento de conexión a la base de datos desde otras direcciones de Internet se rechaza. Por tanto, a no ser que el usuario malicioso sea capaz de registrarse en el servidor de aplicaciones, no podrá hacer daño, incluso aunque consiga acceso a la contraseña de la base de datos.

### 9.7.4. Autenticación de las aplicaciones

La autenticación se refiere a la tarea de verificar la identidad de una persona/software que se conecta a una aplicación. La forma más simple de autenticación consiste en el uso de una contraseña secreta que hay que presentar cuando un usuario se conecta a la aplicación. Desafortunadamente las contraseñas se comprometen fácilmente, por ejemplo, adivinándolas o leyendo los paquetes de la red si estas no se envían cifradas. Para aplicaciones críticas, como los entornos de banca, se necesitan esquemas más robustos. El cifrado es la base de los esquemas de autenticación robustos. La autenticación mediante cifrado se trata en la Sección 9.8.3.

Muchas aplicaciones utilizan la **autenticación en dos pasos,** en la que existen dos *pasos* distintos, es decir, piezas de información o procesos que se usan para identificar al usuario. Estos dos factores no deberían compartir una vulnerabilidad común; por ejemplo, si un sistema requiere dos contraseñas, ambas podrían comprometerse en una fuga de contraseñas de la misma forma; mediante análisis del tráfico de red o mediante un virus en la computadora del usuario. Aunque se pueden usar medidas biométricas como la huella dactilar o los escáneres de iris, no son habituales en la red.

Las contraseñas se usan como primer paso en la mayoría de los esquemas de autenticación en dos pasos. Como segundo paso ampliamente usado, se suelen usar tarjetas inteligentes u otros dispositivos cifrados conectados mediante una interfaz USB (véase la Sección 9.8.3).

Los dispositivos de contraseñas de un solo uso, que generan un nuevo número pseudoaleatorio, por ejemplo cada minuto, también son muy utilizados como segundo paso. Cada usuario dispone de uno de estos dispositivos y para identificarse debe introducir el número que muestra el dispositivo cuando realiza la autenticación, junto con la contraseña. Cada dispositivo genera una secuencia diferente de números pseudoaleatorios. El servidor de aplicaciones puede generar la misma secuencia de números pseudoaleatorios que el dispositivo del usuario, parando cuando llegue al número que se muestra durante el proceso de autenticación y verificando que los números coinciden. Este esquema requiere que el reloj del dispositivo y el del servidor estén razonablemente sincronizados.

Otro segundo paso es el envío de un SMS con un número aleatoriamente generado como contraseña de un solo uso al teléfono del usuario, cuyo número se registró previamente, siempre que el usuario desee registrarse en la aplicación. El usuario tiene que tener un teléfono con dicho número en el que recibir el SMS y, después, introducir la contraseña de un solo uso junto con su contraseña habitual, para autenticarse.

Hay que tener en cuenta que incluso en la autenticación en dos pasos, los usuarios siguen siendo vulnerables a un ataque de **hombre en el medio.** En estos ataques, un usuario que intenta conectarse a la aplicación es desviado a un sitio web falso, que acepta la contraseña, incluyendo la contraseña del segundo paso, y la usa inmediatamente para autenticarse en la aplicación original. El protocolo HTTPS, que se describe más adelante en la Sección 9.8.3.2, se usa para autenticar a los usuarios de un sitio web de forma que el usuario no se conecte a un sitio falso creyendo que era el sitio que pretendía. El protocolo HTTPS cifra los datos y, por tanto, evita el ataque de hombre en el medio.

Cuando los usuarios acceden a varios sitios web, suele resultar tedioso que el usuario tenga que autenticarse en cada uno por separado, normalmente con contraseñas diferentes en cada uno de ellos. Existen sistemas que permiten que el usuario se autentique en un servicio central de autenticación; la misma contraseña sirve para acceder a múltiples sitios web. El protocolo LDAP se usa extensamente para implementar tal punto central de autenticación; las organizaciones implantan un servidor de LDAP con la información de los nombres y contraseñas de los usuarios y las aplicaciones usan el servidor de LDAP para la autenticación de los usuarios.

Además de para la autenticación de usuarios, el servicio central de autenticación se puede utilizar, por ejemplo, para proporcionar información centralizada sobre el usuario, como su nombre, correo electrónico, dirección, etc. a la aplicación. Esto evita introducir esta información en cada aplicación. LDAP se puede usar para esta función, como se describe más adelante en la Sección 19.10.2. Otros sistemas de directorio, como los directorios activos de Microsoft, también proporcionan mecanismos para la autenticación de usuarios, así como para facilitar información de los mismos.

Un sistema de **acceso con firma única** (*single sign-on*) permite que los usuarios se autentiquen una sola vez y distintas aplicaciones puedan verificar la identidad del usuario mediante un servicio de autenticación sin requerir que se vuelvan a autenticar. En otras palabras, una vez que el usuario se ha registrado en un sitio, no tiene que volver a introducir su nombre de usuario y contraseña en el resto de sitios que utilizan el mismo servicio de acceso con firma única. Estos mecanismos de acceso con firma única se han usado en protocolos de autenticación de redes como Kerberos, y existen implementaciones disponibles para su uso en aplicaciones web.

El **lenguaje de marcado para confirmaciones de seguridad** (SAML: *Security Assertion Markup Language*) es una norma para el intercambio de información de autenticación y autorización entre distintos dominios de seguridad, que proporciona acceso de firma única entre organizaciones. Por ejemplo, suponga que una aplicación necesita proporcionar acceso a todos los estudiantes desde una determinada universidad, digamos Yale. La universidad puede configurar un servicio basado en web que lleve a cabo la autenticación. Suponga que un usuario se conecta a la aplicación con un nombre de usuario como «joe@yale.edu». La aplicación, en lugar de autenticar al usuario, desvía a este al servicio de autenticación de la Universidad de Yale, que es quien autentica al mismo y, después, dice a la aplicación quién es el usuario y puede indicarle información adicional como la categoría del mismo (estudiante o profesor) u otra información relevante. La contraseña del usuario y otros pasos de autenticación nunca se revelan a la aplicación, y el usuario no necesita registrarse de forma explícita con la misma. Sin embargo, la aplicación debe confiar en el servicio de autenticación de la universidad cuando autentica a un usuario.

La norma **OpenID** es una forma alternativa de acceso con firma única entre organizaciones y ha incrementado su aceptación en los últimos años. Un gran número de sitios web populares como Google, Microsoft, Yahoo! y otros, actúan como proveedores de autenticación OpenID. Cualquier aplicación que actúe como cliente de OpenID puede utilizar estos proveedores para autenticar a un usuario; por ejemplo, un usuario que tiene una cuanta en Yahoo! puede elegir Yahoo! como proveedor de autenticación. El usuario se ve desviado a Yahoo! para la autenticación y si se efectúa correctamente se le redirige de forma transparente de nuevo a la aplicación y puede continuar utilizándola.

### 9.7.5. Autorización al nivel de aplicación

Aunque la norma de SQL admite un sistema bastante flexible de autorización basado en roles, descritos en la Sección 4.6, el modelo de autorización de SQL desempeña un papel muy limitado en la gestión de las autorizaciones de usuario en una aplicación típica. Por ejemplo, suponga que desea que todos los estudiantes vean sus notas, pero no las notas de los otros. Esta autorización no se puede especificar en SQL por las siguientes dos razones:

1. **Falta de información sobre el usuario final.** Con el crecimiento de la web, el acceso a las bases de datos proviene principalmente de servidores de aplicaciones web. Los usuarios finales normalmente no tienen identificadores de usuario individuales en la propia base de datos y, por tanto, puede que solo haya un único identificador de usuario en la base de datos para todos los usuarios de un servidor de aplicaciones. Por tanto, en el escenario anterior no se puede usar la especificación de autorización en SQL.

   Es posible que un servidor de aplicaciones autentique usuarios finales y, entonces, traslade la información de autenticación a la base de datos. En esta sección se supondrá que la función *syscontext.user id*() devuelve el identificador del usuario de la aplicación en cuyo nombre se ejecuta la consulta.[6]
2. **Falta de autorización de grano fino.** La autorización debe realizarse en el nivel de las tuplas individuales, si se desea autorizar a los estudiantes a que vean solo sus propias notas. Esta autorización no es posible en norma SQL actual, que solo permite la autorización sobre una relación completa o una vista o sobre determinados atributos de relaciones o vistas.

   Se puede intentar solventar esta limitación creando para cada estudiante una vista de la relación *matricula* que únicamente muestre las notas de dicho estudiante. Aunque en principio podría funcionar, sería muy costoso, ya que se debería crear una vista para cada uno de los estudiantes matriculados en la universidad, lo que resulta poco práctico.[7]

   Una alternativa es crear una vista de la forma:

   **create view** *estudianteMatricula* **as**

   **select** *

   **from** *matricula*

   **where** *matricula.ID* = *syscontext.user_id*()

   Los usuarios obtienen autorización para esta vista, en lugar de para toda la relación *matricula* subyacente. Sin embargo, las consultas que se ejecutan en lugar del estudiante deben escribirse en la vista *estudianteMatricula*, en lugar de en la relación *matricula* original; mientras que las consultas ejecutadas en lugar de los profesores puede que necesiten una vista diferente. El resultado final es que la tarea de desarrollar la aplicación se vuelve más compleja.

La tarea de autenticación normalmente se lleva a cabo enteramente en la aplicación, sin las facilidades de autorización de SQL. En el nivel de aplicación, los usuarios consiguen la autorización para determinadas interfaces de acceso, y se les puede restringir la vista o actualización de ciertos datos.

Al realizar la autorización en la aplicación se consigue un alto grado de flexibilidad para los desarrolladores de aplicación, pero presenta ciertos problemas:

- El código de comprobación de la autorización se mezcla con el resto del código de la aplicación.
- La implementación de la autorización en el código de la aplicación, en lugar de especificarlo de forma declarativa en SQL, hace que sea más difícil asegurar la ausencia de agujeros de seguridad. Como resultado, puede que alguno de los programas de aplicación no compruebe la autorización y permita que usuarios no autorizados tengan acceso a datos confidenciales.

Verificar que todos los programas de aplicación realizan las comprobaciones de autorización necesarias implica revisar todo el código del servidor de aplicaciones, una tarea formidable para un gran sistema. En otras palabras, las aplicaciones tienen una gran superficie expuesta, lo que hace que la tarea de proteger a la aplicación sea mucho más difícil. De hecho, se han encontrado agujeros de seguridad en muchas aplicaciones reales.

En contraste, si una base de datos dispone directamente de autorizaciones de grano fino, las políticas de autorización se pueden especificar y forzar en el nivel de SQL, que tiene una superficie expuesta mucho más pequeña. Incluso, aunque algunas interfaces de la aplicación omitan inadvertidamente las comprobaciones de autorización necesarias, el nivel de autorización de SQL evitaría que se ejecutasen acciones no autorizadas.

Algunos sistemas de base de datos proporcionan mecanismos de autorización de grano fino. Por ejemplo, **la base de datos privada virtual** (VPD: *Virtual Private Database*) de Oracle permite que un administrador del sistema asocie una función con una relación; esta función devuelve un predicado que se añade a todas las consultas que utilicen la relación (se pueden añadir diferentes funciones para las actualizaciones). Por ejemplo, usando nuestra sintaxis para obtener los identificadores de usuario de la aplicación, la función de la relación *matricula* puede devolver un predicado como:

$$ID = sys\ context.user\_id()$$

Este predicado se añade a la cláusula **where** de todas las consultas que usen la relación *matricula*. El resultado, suponiendo que el programa de aplicación configura el valor de *user_id* al *ID* del estudiante, es que cada estudiante solo verá las tuplas correspondientes a los cursos en que se haya matriculado.

Por tanto, VPD proporciona autorización en el nivel de determinadas tuplas o filas de una relación, y se puede decir, en este sentido, que es un mecanismo de *autorización en el nivel de fila*. Un problema potencial de añadir un predicado como el descrito es que puede cambiar de forma importante el significado de una consulta. Por ejemplo, si un usuario escribe una consulta para encontrar la nota media de todas las asignaturas, debería obtener el promedio de *sus* notas, no de todas las notas. Aunque el sistema le diese la respuesta «correcta» para la consulta reescrita, la respuesta no se correspondería con la consulta que el usuario puede que desease realizar al enviarla.

Véanse las notas bibliográficas para más información sobre la VPD de Oracle.

6 En Oracle, una conexión de JDBC usando el controlador de JDBC de Oracle puede establecer el identificador de usuario final usando el método `OracleConnection.setClientIdentifier(userId)`, y la consulta de SQL puede usar la función `sys_context('USERENV', 'CLIENT_IDENTIFIER')` para obtener el identificador de usuario.

7 Los sistemas de base de datos se diseñan para gestionar grandes relaciones, pero gestionan la información de esquema, como las vistas, de una forma que supone volúmenes de datos menores para mejorar el rendimiento total.

### 9.7.6. Trazas de auditoría

Una **traza de auditoría** es un registro de todos los cambios, inserciones, borrados y actualizaciones en los datos de la aplicación, junto con la información sobre el usuario que realizó los cambios y cuándo se hicieron. Si se rompe la seguridad de la aplicación o, aunque no se rompa la seguridad, se realiza alguna actualización de forma errónea, la traza de auditoría puede (a) ayudar a descubrir qué ha pasado y quién llevó a cabo las acciones, y (b) ayudar a reparar el daño realizado tras la brecha de seguridad o la actualización errónea.

Por ejemplo, si se descubre que una nota de un estudiante es incorrecta, se puede examinar el registro de auditoría para descubrir cuándo y cómo se modificó la nota, así como para descubrir quién realizó el cambio. La universidad también puede usar la traza de auditoría para trazar todas las actualizaciones realizadas por un usuario y descubrir otras actualizaciones incorrectas o fraudulentas y corregirlas.

Las trazas de auditoría también se pueden utilizar para detectar brechas de seguridad cuando una cuenta de usuario se ha visto comprometida y ha accedido con ella un intruso. Por ejemplo, cada vez que se registra un usuario se le puede informar de todas las actualizaciones en la traza de auditoría que se realizaron en el pasado reciente; si el usuario observa una actualización que no hizo, es probable que la cuenta esté comprometida.

Es posible crear trazas de auditoría definiendo los disparadores adecuados sobre las actualizaciones de las relaciones (usando variables definidas por el sistema que identifican el nombre de usuario y la hora). Sin embargo, muchos sistemas de bases de datos proporcionan mecanismos predeterminados para crear trazas de auditoría que son más cómodos de usar. Los detalles de la creación de las trazas de auditoría varían de unos sistemas de bases de datos a otros, y se deben consultar los manuales de los diferentes sistemas para conocer los detalles.

Las trazas de auditoría del nivel de base de datos suelen ser insuficientes para las aplicaciones, ya que no suelen ser capaces de trazar quién es el usuario final de la aplicación. Más aún, las actualizaciones se registran en un nivel bajo, en términos de la actualización de las tuplas de una relación, en lugar de en un nivel alto, en términos de la lógica de negocio. Por tanto, las aplicaciones normalmente crean una traza de auditoría de alto nivel, registrando, por ejemplo, qué acción se ha realizado, por quién, cuándo y desde qué dirección IP se originó la petición.

Un tema relacionado es el de la protección de la propia traza de auditoría de forma que no sea modificada o borrada por los usuarios que rompen una brecha de seguridad de la aplicación. Una solución consiste en realizar una copia de la traza de auditoría en una máquina diferente, a la que el intruso no pueda tener acceso, y que cada registro de la traza de auditoría se copie en cuanto se genere.

### 9.7.7. Privacidad

En un mundo en el que ha crecido la cantidad de datos personales accesibles online, las personas están cada vez más preocupadas por la privacidad de sus datos. Por ejemplo, la mayoría de las personas querrán que sus datos médicos se mantengan privados y no puedan hacerse públicos. Sin embargo, los datos médicos deben estar disponibles para los médicos y técnicos de emergencias que tratan al paciente. La mayoría de los países tienen leyes sobre la privacidad de estos datos que definen cuándo y quién puede acceder a ellos. El incumplimiento de las leyes sobre privacidad puede conllevar penas criminales en algunos países. Hay que construir con sumo cuidado las aplicaciones que acceden a estos datos privados, teniendo en cuenta las leyes sobre privacidad.

Por otra parte, los datos privados agregados desempeñan un importante papel en muchas tareas como la detección de efectos secundarios de los medicamentos o la detección de la propagación de una epidemia. Cómo hacer disponibles estos datos para los investigadores sin comprometer la privacidad de las personas es un problema real. Como ejemplo, suponga que un hospital oculta los nombres de los pacientes pero proporciona al investigador las fechas de nacimiento y los códigos postales (ambos útiles para el investigador). En muchos casos, con estos dos elementos de información se puede identificar de forma única a un paciente (con información de bases de datos externas), comprometiendo su privacidad. En esta situación en particular, una solución podría ser proporcionar el año de nacimiento pero no la fecha, junto con el código postal, lo que puede ser suficiente para el investigador. De esta forma no se proporciona información suficiente como para identificar de forma unívoca a la mayoría de las personas.[8]

Otro ejemplo: los sitios web suelen recopilar información de datos personales, como dirección, teléfono, dirección de email y tarjeta de crédito. Esta información puede ser necesaria para realizar una transacción de compra de un artículo en una tienda. Sin embargo, el cliente puede querer que esta información no esté disponible para otras organizaciones, o puede querer que cierta información, como el número de la tarjeta de crédito, se borre tras cierto periodo de tiempo como forma de evitar que caiga en manos no autorizadas en caso de que se produzca una brecha de seguridad. Muchos sitios web permiten a los clientes que indiquen sus preferencias de privacidad y deben asegurar que estas preferencias se respetan.

## 9.8. Cifrado y sus aplicaciones

El cifrado se refiere al proceso de transformar los datos de forma que no sean legibles, hasta que se aplica el proceso inverso de descifrado. Los algoritmos de cifrado usan una clave de cifrado para realizar el cifrado y necesitan una clave de descifrado (que puede ser la misma que la de cifrado o no, dependiendo del algoritmo utilizado), para realizar el descifrado.

El uso más antiguo del cifrado fue la transmisión de mensajes, cifrados usando una clave secreta que solo conocían el remitente y el receptor. Aunque el mensaje cayese en manos enemigas, el enemigo sin la clave no podía descifrar el mensaje ni entenderlo. El cifrado se usa extensamente en la actualidad para proteger los datos en tránsito en muy diversas aplicaciones, como la transferencia de datos en Internet y en las redes telefónicas móviles celulares. El cifrado también se usa para realizar otras tareas como la autenticación, como se ha visto en la Sección 9.8.3.

En el contexto de las bases de datos, el cifrado se usa para guardar los datos de forma segura, de modo que aunque se consigan por un usuario no autorizado, por ejemplo por el robo de un portátil que contiene los datos, no se pueda acceder a ellos sin la clave de descifrado.

Muchas bases de datos actuales guardan información sensible de clientes, como números de tarjetas de crédito, nombre, huellas dactilares, firmas y números de identificación; por ejemplo, en los Estados Unidos, el número de la seguridad social. Un criminal que acceda a estos datos puede usarlos de muchas formas en actividades ilegales, como la compra de bienes usando los números de las tarjetas de crédito o incluso solicitando una tarjeta de crédito en nombre de otra persona. Las organizaciones, como las compañías de tarjetas de crédito, utilizan el conocimiento de la información personal para identificar quién está solicitando un bien o un

8 En el caso de personas muy ancianas, que son relativamente infrecuentes, el año de nacimiento junto con el código postal pueden ser suficientes para identificarlas unívocamente, por lo que para las personas mayores de 80 años puede proporcionarse un intervalo de valores entre 80 años o más, en lugar de la edad real.

servicio. La filtración de esta información personal permite a un criminal suplantar a otra persona y acceder a bienes y servicios; esta suplantación se denomina **robo de identidad.** Por tanto, las aplicaciones que guardan este tipo de información sensible deben tener sumo cuidado de protegerla del robo.

Para reducir la posibilidad de que información sensible llegue a manos criminales, los países establecen por ley que cualquier base de datos que guarde información sensible debe guardarla de forma cifrada. Un negocio que no protege sus datos puede ser declarado culpable criminalmente en caso de robo. Por tanto, el cifrado es un componente crítico en cualquier aplicación que guarde información sensible.

### 9.8.1. Técnicas de cifrado

Existe un enorme número de técnicas para el cifrado de los datos. Puede que las técnicas de cifrado sencillas no proporcionen la seguridad adecuada, dado que para los usuarios no autorizados puede ser sencillo romper el código. Como ejemplo de técnica de cifrado débil considérese la sustitución de cada carácter por el siguiente en el alfabeto. Por tanto,

Perryridge

se transforma en:

Qfsszsjehf

Si un usuario no autorizado solo lee «Qfsszsjehf», probablemente no tenga la información suficiente para romper el código. Sin embargo, si el intruso ve gran número de nombres de sucursales cifrados, puede usar los datos estadísticos referentes a la frecuencia relativa de los caracteres para averiguar el tipo de sustitución realizado (por ejemplo, en español la *E* es la letra más frecuente, seguida por *A, O, L, S, M*, etc.).

Una buena técnica de cifrado posee las siguientes propiedades:

- Resulta relativamente sencillo para los usuarios autorizados cifrar y descifrar los datos.
- No depende del secreto del algoritmo, sino de un parámetro del algoritmo denominado *clave de cifrado*, que se usa para cifrar los datos. En una técnica de cifrado de **clave simétrica** la clave de cifrado también se usa para el descifrado. Al contrario, en una técnica de cifrado de **clave pública** (también conocida como **clave asimétrica**) existen dos claves diferentes, la clave pública y la clave privada, usadas para cifrar y descifrar los datos.
- Es extremadamente difícil para un intruso determinar la clave de cifrado, incluso aunque el intruso tenga acceso a datos cifrados. En el caso del cifrado con clave asimétrica, es extremadamente difícil inferir la calve privada, incluso aunque se disponga de la clave pública.

La **norma de cifrado avanzado** (AES: *Advanced Encryption Standard*) es un algoritmo de cifrado de clave simétrica que fue adoptado como norma de cifrado por el Gobierno de los EE.UU. en el año 2000, y ahora se usa extensamente. La norma se basa en el **algoritmo Rijndael** (por sus inventores V. Rijmen y J. Daemen). El algoritmo utiliza bloques de 128-bits de datos cada vez, mientras que la clave puede ser de 128, 192 o 256 bits de tamaño.

El algoritmo ejecuta un conjunto de pasos para desordenar los bits del bloque de datos, de forma que se pueda invertir el proceso durante el descifrado, y realiza una operación XOR con una «subclave» de 128 bits que se deriva de la clave de cifrado. Para cada uno de los bloques de datos que se cifra se genera una nueva subclave a partir de la clave de cifrado. Durante el descifrado, las subclaves se vuelven a generar a partir de la clave de cifrado y el proceso de cifrado se invierte para recuperar los datos originales. Anteriormente se usaba extensamente una norma llamada *norma de cifrado de datos* (DES: *Data Encryption Standard*), adoptada en 1977.

Para que cualquier esquema de cifrado de clave simétrica funcione, los usuarios deben obtener la clave de cifrado mediante un mecanismo seguro. Este requisito supone la mayor debilidad, ya que el esquema no es más seguro que la seguridad que ofrezca el mecanismo para transmitir la clave de cifrado.

El **cifrado de clave pública** es un esquema alternativo que evita parte de los problemas que se afrontan con las técnicas de cifrado de clave simétrica. Se basa en dos claves; una *clave pública* y otra *clave privada*. Cada usuario $U_i$ tiene una clave pública $E_i$ y una clave privada $D_i$. Todas las claves públicas están publicadas: cualquiera puede verlas. Cada clave privada solo la conoce el usuario al que pertenece. Si el usuario $U_1$ desea guardar datos cifrados, los cifra usando la clave pública $E_1$. Descifrarlos exige la clave privada $D_1$.

Como la clave de cifrado de cada usuario es pública, se puede intercambiar información de manera segura usando este esquema. Si el usuario $U_1$ desea compartir los datos con $U_2$, los codifica usando $E_2$, la clave pública de $U_2$. Dado que solo el usuario $U_2$ conoce la manera de descifrar los datos, la información se transmite de manera segura.

Para que el cifrado de clave pública funcione, debe haber un esquema de cifrado que haga extremadamente difícil de deducir la clave privada, dada la clave pública. Existe un esquema de ese tipo y se basa en estas condiciones:

- Que haya un algoritmo eficiente para comprobar si un número es primo o no.
- Que no se conozca ningún algoritmo eficiente para encontrar los factores primos de un número.

Para los fines de este esquema, los datos se tratan como conjuntos de enteros. Se crea la clave pública calculando el producto de dos números primos grandes: $P_1$ y $P_2$. La clave privada consiste en el par ($P_1$, $P_2$). El algoritmo de descifrado no se puede usar con éxito si solo se conoce el producto $P_1P_2$; necesita el valor de $P_1$ y de $P_2$. Dado que todo lo que se publica es el producto $P_1P_2$, los usuarios no autorizados necesitan poder factorizar $P_1P_2$ para robar los datos. Eligiendo $P_1$ y $P_2$ suficientemente grandes (por encima de 100 cifras) se puede hacer el coste de la factorización de $P_1P_2$ prohibitivamente elevado (del orden de años de tiempo de cálculo, incluso para las computadoras más rápidas).

En las notas bibliográficas se hace referencia a los detalles del cifrado de clave pública y a la justificación matemática de las propiedades de esta técnica.

Pese a que el cifrado de clave pública que usa el esquema descrito es seguro, también es costoso en cuanto a cálculo. Un esquema híbrido usado para proteger las comunicaciones es el siguiente: se genera de forma aleatoria una clave de cifrado simétrica (basada, por ejemplo, en AES) y se intercambia de forma segura usando un esquema de cifrado de clave pública, entonces se usa el cifrado de clave simétrica usando dicha clave para los datos que se transmitan después.

El cifrado de valores pequeños, como identificadores o nombres, se complica por la posibilidad de un **ataque de diccionario,** en particular si la clave de cifrado es pública. Por ejemplo, si se cifran los campos de fecha de nacimiento, un atacante intentando descifrar un determinado valor *e* puede intentar cifrar todos los posibles valores de fecha de nacimiento hasta que encuentre uno cuyo valor cifrado coincida con *e*. Incluso aunque la clave de cifrado no esté disponible públicamente, la información estadística sobre la distribución de los datos se puede usar para adivinar qué representa el valor cifrado en algunos casos, como la edad o el código postal. Por ejemplo, si la edad de 18 años es la edad más común en la base de datos, se podrá inferir que el valor cifrado de la edad que se produzca con más frecuencia representa 18.

Los ataques de diccionario se pueden combatir añadiendo bits extra aleatorios al final de cada uno de los valores antes de su cifrado, y eliminándolos al descifrar. Estos bits extra, llamados **vector de inicialización** en AES, o bits de *sal* en otros contextos, proporcionan un nivel de protección extra contra ataques de diccionario.

### 9.8.2. Soporte del cifrado en las bases de datos

Muchos sistemas de archivos y de bases de datos de hoy en día disponen del cifrado de los datos. Ese cifrado protege los datos de quien pueda tener acceso a ellos pero no sea capaz de tener acceso a la clave de descifrado. En el caso del cifrado del sistema de archivos, los datos que se cifran suelen ser archivos de gran tamaño y directorios que contienen información sobre los archivos.

En el contexto de las bases de datos, el cifrado se puede llevar a cabo en diferentes niveles. En el nivel inferior, se pueden cifrar los bloques de disco que contienen datos de la base de datos usando una clave disponible para el software del sistema de bases de datos. Cuando se recupera un bloque del disco, primero se descifra y luego se usa de la manera habitual. Este cifrado en el nivel de los bloques del disco protege contra los atacantes que puedan tener acceso al contenido del disco pero no dispongan de la clave de cifrado.

En el nivel siguiente, se pueden guardar cifrados ciertos atributos de una relación, o todos. En este caso, cada atributo de una relación podría tener una clave de cifrado diferente. Muchas bases de datos actuales permiten el cifrado en el nivel de atributos, así como en el nivel de una relación completa o de todas las relaciones de la base de datos. El cifrado de solo algunos atributos minimiza la sobrecarga del cifrado, permitiendo que las aplicaciones solo cifren los atributos con valores sensibles, como los números de las tarjetas de crédito. Sin embargo, cuando se cifran determinados atributos o relaciones, no se suele permitir que se cifren las claves primarias ni las claves externas, y no se permite el indizado de atributos cifrados. En el cifrado se necesita usar bits adicionales aleatorios para evitar un ataque de diccionario, como se ha descrito anteriormente.

Obviamente se necesita una clave de descifrado para acceder a los datos cifrados. Se puede usar una clave maestra única para todos los datos cifrados; con el cifrado de atributos, se pueden usar claves diferentes para atributos distintos. En este caso las distintas claves de cifrado se pueden guardar en un archivo o relación, normalmente llamado «wallet» (cartera), que a su vez se encuentra cifrado utilizando la clave maestra.

Cuando se realiza una conexión a la base de datos que necesita acceder a atributos cifrados, debe proporcionar la clave maestra; si no se proporciona la conexión no podrá acceder a los datos cifrados. La clave maestra se guarda en el programa de aplicación, normalmente en una computadora distinta, o memorizada en la base de datos de usuario, y se proporciona cuando este se conecta a la base de datos.

El cifrado en el nivel de la base de datos tiene la ventaja de requerir relativamente poca sobrecarga de tiempo y espacio, y no necesita la modificación de las aplicaciones. Por ejemplo, si se precisa proteger los datos de una base de datos en un portátil, se puede usar este tipo de cifrado. De forma similar, quien tenga acceso a las copias de seguridad de una base de datos no debería ser capaz de acceder a los datos de la copia de seguridad sin conocer la clave de descifrado.

Una alternativa a realizar el cifrado de la base de datos es realizarlo *antes* de que los datos se guarden en la base de datos. La aplicación realiza el cifrado de los datos antes de enviarlos a la base de datos y los descifra cuando los obtiene. Esta forma de cifrado requiere una modificación significativa de la aplicación, frente a la que realiza el sistema de base de datos.

### 9.8.3. Cifrado y autenticación

La autenticación basada en contraseñas la usan mucho los sistemas operativos y las bases de datos. Sin embargo, el uso de contraseñas tiene algunos inconvenientes, especialmente en las redes. Si un espía es capaz de «esnifar» los datos que se envían por la red, puede ser capaz de averiguar la contraseña cuando se envíe por la red. Una vez que el husmeador obtiene un usuario y una contraseña, se puede conectar a la base de datos fingiendo que es el usuario legítimo.

Un esquema más seguro es el sistema de **respuesta por desafío.** El sistema de bases de datos envía una cadena de desafío al usuario. El usuario cifra la cadena de desafío usando una contraseña secreta como clave de cifrado y devuelve el resultado. El sistema de bases de datos puede verificar la autenticidad del usuario descifrando la cadena con la misma contraseña secreta, y comparando el resultado con la cadena de desafío original. Este esquema garantiza que las contraseñas no viajan por la red.

Los sistemas de clave pública se pueden usar para el cifrado en los sistemas de respuesta por desafío. El sistema de bases de datos cifra la cadena de desafío usando la clave pública del usuario y se la envía al usuario. Este descifra la cadena con su clave privada y devuelve el resultado al sistema de bases de datos. El sistema de bases de datos comprueba entonces la respuesta. Este esquema tiene la ventaja añadida de no almacenar la contraseña secreta en la base de datos, donde podrían verla los administradores del sistema.

Guardar la clave privada del usuario en una computadora, aunque sea una computadora personal, tiene el riesgo de que, si se pone en peligro la seguridad de la computadora, se puede revelar la clave al atacante, que podrá suplantar al usuario. Las **tarjetas inteligentes** proporcionan una solución a este problema. En las tarjetas inteligentes la clave se puede guardar en un chip incorporado; el sistema operativo de la tarjeta inteligente garantiza que no se pueda leer nunca la clave, pero permite que se envíen datos a la tarjeta para cifrarlos o descifrarlos usando la clave privada.[9]

#### 9.8.3.1. Firma digital

Otra aplicación interesante del cifrado de clave pública está en la **firma digital** para comprobar la autenticidad de los datos; la firma digital desempeña el rol electrónico de las firmas físicas en los documentos. La clave privada se usa para «firmar»; es decir, cifrar los datos, y los datos firmados se pueden hacer públicos. Cualquiera puede comprobar la firma descifrando los datos con la clave pública, pero nadie puede haber generado los datos firmados sin tener la clave privada. Fíjese en la inversión de los roles de las claves pública y privada en este esquema. Por tanto, se pueden **autenticar** los datos; es decir, se puede comprobar que fueron creados realmente por la persona que afirma haberlos creado.

Además, las firmas digitales también sirven para garantizar el **no repudio.** Es decir, en el caso de que la persona que creó los datos afirmara posteriormente que no los creó, el equivalente electrónico de afirmar que no se ha firmado un cheque, se puede probar que esa persona tiene que haber creado los datos, a menos que su clave privada haya caído en manos de otros.

#### 9.8.3.2. Certificados digitales

En general, la autenticación es un proceso de doble sentido en el que cada miembro del par de entidades que interactúa se autentica a sí mismo frente al otro. Esta autenticación por parejas es necesaria aunque un cliente entre en contacto con un sitio web, para

9 Las tarjetas inteligentes también proporcionan otra funcionalidad, como la capacidad de almacenar dinero digital y realizar pagos, lo que no es relevante en este contexto.

evitar que los sitios malévolos se hagan pasar por sitios web legales. Esa suplantación se puede llevar a cabo, por ejemplo, si se comprometen los encaminadores de la red y los datos se desvían al sitio malicioso.

Para que el usuario pueda estar seguro de que está interactuando con el sitio web auténtico debe tener la clave pública de ese sitio. Esto plantea el problema de hacer que el usuario consiga la clave pública, si se almacena en el sitio web, pudiendo el sitio malicioso proporcionar una clave diferente, con lo que el usuario no tendría manera de comprobar si la clave pública proporcionada es auténtica. La autenticación se puede conseguir mediante un sistema de **certificados digitales** en el que las claves públicas están firmadas por una agencia de certificación, cuya clave pública es bien conocida. Por ejemplo, las claves públicas de las autoridades de certificación raíz se almacenan en los navegadores web estándar. Los certificados emitidos por ellas se pueden comprobar mediante las claves públicas almacenadas.

Un sistema de dos niveles supondría una carga excesiva de creación de certificados para las autoridades de certificación raíz, por lo que se usa en su lugar un sistema de varios niveles, con una o más autoridades de certificación raíz y un árbol de autoridades de certificación por debajo de cada raíz. Cada autoridad, distinta de la autoridad raíz, tiene un certificado digital emitido por su autoridad padre.

El certificado digital emitido por la autoridad de certificación $A$ consiste en una clave pública $K_A$ y un texto cifrado $E$ que se puede descifrar mediante la clave pública $K_A$. El texto cifrado contiene el nombre de la parte a la que se le ha emitido el certificado y su clave pública $Kc$. En el caso de que la autoridad de certificación $A$ no sea autoridad de certificación raíz, el texto cifrado también contiene el certificado digital emitido a $A$ por su autoridad de certificación padre; este certificado autentica la propia clave $K_A$ (ese certificado puede, a su vez, contener un certificado de otra autoridad de certificación padre, y así sucesivamente).

Para comprobar un certificado, se descifra el texto cifrado $E$ mediante la clave pública $K_A$ para obtener el nombre de la parte, es decir, el nombre de la organización propietaria del sitio web; adicionalmente, si $A$ no es autoridad raíz de quien se conozca la clave pública para el verificador, la clave pública $K_A$ se comprueba de manera recursiva usando el certificado digital contenido en $E$; la recursividad termina cuando se llega a un certificado emitido por la autoridad raíz. La comprobación de los certificados establece la cadena mediante la cual se autentica cada sitio concreto y proporciona el nombre y la clave pública autenticada de ese sitio.

Los certificados digitales se usan mucho para autentificar los sitios web ante los usuarios, para evitar que sitios maliciosos suplanten a otros sitios web. En el protocolo HTTPS, la versión segura del protocolo HTTP, el sitio proporciona su certificado digital al navegador, que lo muestra entonces al usuario. Si el usuario acepta el certificado, el navegador usa la clave pública proporcionada para cifrar los datos. Los sitios maliciosos pueden tener acceso al certificado, pero no a la clave privada y, por tanto, no podrán descifrar los datos enviados por el navegador. Solo el sitio auténtico, que tiene la clave privada correspondiente, puede descifrar los datos enviados por el navegador. Hay que tener en cuenta que los costes del cifrado y descifrado con la clave pública y la clave privada son mucho más elevados que los de cifrado y descifrado mediante claves privadas simétricas. Para reducir los costes de cifrado, HTTPS crea realmente una clave simétrica de un solo uso tras la autenticación y la usa para cifrar los datos durante el resto de la sesión.

Los certificados digitales también se pueden usar para autenticar a los usuarios. El usuario debe remitir al sitio un certificado digital que contenga su clave pública; el sitio comprueba que el certificado haya sido firmado por una autoridad de confianza. Entonces se puede usar la clave pública del usuario en un sistema de respuesta por desafío para garantizar que ese usuario posee la clave privada correspondiente, lo que autentifica al usuario.



## 9.9. Resumen

- Los programas de aplicación usan las bases de datos como sistemas de soporte y su interacción con los usuarios ha existido desde los años sesenta. Las arquitecturas de aplicación han evolucionado en todo este tiempo. En la actualidad la mayoría de las aplicaciones usan navegadores web como *front-end*, y las bases de datos como *back-end*, con un servidor de aplicaciones entre medias.
- HTML tiene la capacidad de definir interfaces que combinan hipervínculos con formularios. Los navegadores web se comunican con los servidores web mediante el protocolo HTTP. Los servidores web pueden pasar solicitudes a los programas de aplicación y devolver el resultado al navegador.
- Los servidores web ejecutan programas de aplicación para implementar la funcionalidad deseada. Las servlets son un mecanismo muy usado para escribir programas de aplicación que se ejecutan como parte del proceso del servidor web, para reducir las sobrecargas. También hay muchos lenguajes de secuencias de comandos del lado del servidor que interpreta el servidor web y proporcionan la funcionalidad de los programas de aplicación como parte del servidor web.
- Hay varios lenguajes de secuencias de comandos del lado del cliente —JavaScript es el más usado— que proporcionan una mayor interacción con el usuario en el extremo del navegador.
- Las aplicaciones complejas suelen tener una arquitectura multicapa, incluyendo un modelo que implementa la lógica del negocio, un controlador y un mecanismo de vista para mostrar los resultados. También pueden incluir una capa de acceso a los datos que implementa la correspondencia objeto-relación. Muchas aplicaciones implementan y usan servicios web, que permiten invocar las funciones usando HTTP.
- Se han desarrollado herramientas para el desarrollo rápido de aplicaciones, y en particular para reducir el esfuerzo que requiere construir las interfaces gráficas de usuario.
- Para mejorar el rendimiento de las aplicaciones se utilizan técnicas de caché en varias formas, incluyendo el caché de resultados, los bancos de conexiones y el procesamiento paralelo.
- Los desarrolladores de aplicaciones deben prestar mucha atención a la seguridad, para evitar ataques como la inyección de SQL y otros ataques de secuencias de comandos de sitios cruzados.
- Los mecanismos de autorización de SQL son de grano grueso y de un valor limitado para las aplicaciones que tratan con un gran número de usuarios. En la actualidad, los programas de aplicaciones implementan la autorización de grano fino, en el nivel de tupla, tratando con un gran número de usuarios, completamente fuera del sistema de bases de datos. Se han desarrollado extensiones para proporcionar nivel de acceso al nivel de tupla y para tratar con un gran número de aplicaciones, pero aún no están estandarizados.

- La protección de la privacidad de los datos es una importante tarea de las aplicaciones de bases de datos. Muchos países tienen establecidos requisitos legales sobre la protección de ciertos tipos de datos, como la información de tarjetas de crédito o de datos médicos.
- El cifrado desempeña un papel fundamental en la protección de la información y en la autenticación de los usuarios y de los sitios web. El cifrado de clave simétrica y el cifrado de clave pública son técnicas de cifrado contrastadas y muy extendidas. El cifrado de ciertos datos sensibles almacenados en las bases de datos es un requisito legal en muchos países.
- El cifrado también desempeña un papel fundamental en la autenticación de usuarios a las aplicaciones, de sitios web a los usuarios y para la firma digital.

## Términos de repaso

- Programas de aplicación.
- Interfaces web con bases de datos.
- Hiperenlaces.
- Localizador uniforme de recursos (URL).
- Formularios.
- Protocolo de transferencia de hipertexto (HTTP).
- Interfaz de pasarela común (CGI).
- Protocolos sin conexión.
- Cookie.
- Sesión.
- Servlets y sesiones de Servlet.
- Secuencia de comandos del lado del servidor.
- JSP.
- PHP.
- ASP.NET.
- Secuencia de comandos del lado del cliente.
- JavaScript.
- Modelo de objetos de documentos (DOM).
- Applets.
- Arquitectura de aplicación.
- Capa de presentación.
- Arquitectura modelo-vista-controlador (MVC).
- Capa de la lógica de negocio.
- Capa de acceso a los datos.
- Correspondencia objeto-relación.
- Hibernate.
- Lenguaje de marcado de hipertexto (HTML).
- Servicios web.
- Servicios RESTful.
- Desarrollo rápido de aplicaciones.
- Framework de aplicaciones web.
- Generadores de informes.
- Banco de conexiones.
- Caché de resultados de consultas.
- Seguridad de las aplicaciones.
- Inyección SQL.
- Secuencias de comandos de sitios cruzados (XSS).
- Falsificación de petición en sitios cruzados (XSRF).
- Autenticación.
- Autenticación en dos pasos.
- Ataque de hombre en el medio.
- Autenticación central.
- Registro de firma única.
- OpenID.
- Base de datos privada virtual (VPD).
- Trazas de auditoría.
- Cifrado.
- Cifrado de clave simétrica.
- Cifrado de clave pública.
- Ataque de diccionario.
- Desafío-respuesta.
- Firma digital.
- Certificado digital.



## Ejercicios prácticos

**9.1.** ¿Cuál es la razón principal de que los servlets den mejor rendimiento que los programas que usan la interfaz de pasarela común (common gateway interface, CGI), pese a que los programas de Java suelen ejecutarse más lentamente que los programas de C o de C++?

**9.2.** Indique algunas de las ventajas y de los inconvenientes de los protocolos sin conexión frente a los protocolos que mantienen las conexiones.

**9.3.** Considere una aplicación web, escrita sin mucho cuidado, para un sitio de compra online que almacena el precio de cada artículo en una variable oculta de un formulario en la página web que envía al cliente; cuando el cliente envía el formulario, se usa la información de la variable oculta del formulario para calcular la factura del cliente. ¿Qué agujero tiene este esquema? Existió un ejemplo real en el que el agujero de seguridad fue explotado por algunos usuarios de sitio de compra online antes de que el problema se detectase y reparase.

**9.4.** Considere otra aplicación web escrita de manera poco cuidadosa, que usa un servlet que comprueba si existe una sesión activa, pero no comprueba si el usuario está autorizado para acceder a la página, sino que solo depende de que el enlace se muestra exclusivamente a usuarios autorizados. ¿Cuál es el riesgo de este esquema? (Existió un ejemplo real en el que los que deseaban la admisión en una universidad podían, tras registrarse en el sitio web, explotar este agujero de seguridad para ver información que no estaban autorizados a ver; el acceso no autorizado se detectó y a aquellos que accedieron a la información se les penalizó no admitiéndoles en la universidad.)

**9.5.** Indíquense tres maneras en que se puede usar el almacenamiento en caché para acelerar el rendimiento de los servidores web.

**9.6.** El comando `netstat` (disponible tanto en Linux como en Windows) muestra las conexiones de red activas en una computadora. Explique cómo se puede usar este comando para

descubrir si una página web en particular no está cerrando las conexiones que abre, o si se usa un banco de conexiones y no devuelve las conexiones al banco. Debería tener en cuenta que con el banco de conexiones las conexiones no se cierran inmediatamente.

**9.7.** Comprobando una vulnerabilidad de inyección SQL:

**a.** Sugiera una forma de comprobar en una aplicación si es vulnerable a ataques de inyección SQL en la entrada de texto.

**b.** ¿Puede producirse una inyección SQL con otras formas de entrada? Si es así, cómo se podría comprobar esta vulnerabilidad?

**9.8.** Suponga una relación de una base de datos en la que los valores de ciertos atributos se han cifrado por seguridad. ¿Por qué los sistemas de bases de datos no permiten el indizado sobre atributos cifrados? Use su respuesta a la pregunta anterior para explicar por qué los sistemas de bases de datos no permiten el cifrado de atributos de clave primaria.

**9.9.** En el Ejercicio 9.8 se trata el problema del cifrado de ciertos atributos. Sin embargo, algunos sistemas de bases de datos permiten el cifrado de toda la base de datos. Explique por qué se evitan los problemas planteados en el Ejercicio 9.8 cuando se cifra toda la base de datos.

**9.10.** Suponga que alguien suplanta a una empresa y obtiene un certificado de una autoridad emisora de certificados. ¿Cuál es el efecto sobre las cosas (como los pedidos de compras o los programas) certificadas por la empresa suplantada y sobre las cosas certificadas por otras empresas?

**9.11.** Quizás los elementos de datos más importantes de cualquier sistema de bases de datos sean las contraseñas que controlan el acceso a la base de datos. Proponga un esquema para guardar de manera segura las contraseñas. Es necesario asegurar que el esquema permite al sistema comprobar las contraseñas proporcionadas por los usuarios que intenten iniciar una sesión en el mismo.

## Ejercicios

**9.12.** Escriba un servlet y el código HTML asociado para la siguiente aplicación, muy sencilla: se permite que los usuarios remitan un formulario que contiene un valor, por ejemplo, *n*, y deben obtener una respuesta que contenga *n* símbolos "*".

**9.13.** Escriba un servlet y el código HTML asociado para la siguiente aplicación sencilla: se permite que los usuarios remitan un formulario que contiene un número, por ejemplo *n*, y deben obtener una respuesta que indique el número de veces que se ha remitido anteriormente el valor *n*. El número de veces que se ha remitido anteriormente el valor se debe almacenar en una base de datos.

**9.14.** Escriba un servlet que autentique a los usuarios (de acuerdo con los nombres de usuario y las contraseñas almacenadas en una relación de una base de datos) y defina una variable de sesión denominada *idusuario* tras la autenticación.

**9.15.** ¿Qué son los ataques de inyección de SQL? Explique cómo funcionan y las precauciones que se deben adoptar para evitarlos.

**9.16.** Escriba un pseudocódigo para gestionar un banco de conexiones. El pseudocódigo debe incluir una función que cree el banco (proporcionando una cadena de caracteres de conexión a la base de datos, el nombre de usuario de la base de datos y la contraseña como parámetros), una función que solicite una conexión al banco, una función que libere una conexión al banco y una función que cierre el banco de conexiones.

**9.17.** Explique los términos CRUD y REST.

**9.18.** Muchos sitios web actuales proporcionan ricas interfaces de usuario usando Ajax. Indique dos características que pueden revelar si un sitio usa Ajax, sin tener que ver el código fuente. Usando las características anteriores, descubra tres sitios web que usan Ajax; puede ver el código HTML de la página para comprobar si el sitio usa realmente Ajax.

**9.19.** Sobre los ataques XSS:

**a.** ¿Qué es un ataque XSS?

**b.** ¿Cómo se puede usar el campo «referer» para detectar algunos ataques XSS?

**9.20.** ¿Qué es la autenticación multipaso? ¿Cómo ayuda a protegerse contra el robo de contraseñas?

**9.21.** Considere la característica de **base de datos privada virtual** (**VPD**) de Oracle que se describe en la Sección 9.7.5 y una aplicación basada en nuestro esquema de universidad.

**a.** ¿Qué predicado (usando una subconsulta) se debería generar para permitir que los miembros de una facultad vean solo las tuplas de *matricula* que corresponden a secciones de asignaturas que ellos han enseñado?

**b.** Escriba una consulta de SQL tal que la consulta con el predicado añadido genere un resultado que es un subconjunto de la consulta resultado original sin añadir el predicado.

**c.** Escriba una consulta de SQL tal que la consulta con el predicado añadido genere un resultado que contiene una tupla que no está en el resultado original de la consulta sin añadir el predicado.

**9.22.** Indique dos ventajas de cifrar los datos guardados en una base de datos.

**9.23.** Suponga que se desea crear una traza de auditoría de las modificaciones de la relación *matricula*.

**a.** Defina los disparadores para crear una traza de auditoría y registrar la información en una relación denominada, por ejemplo, *traza_matricula*. La información registrada debe incluir el identificador de usuario (suponga que la función *id_usuario*() proporciona esa información) y una marca de tiempo, además de los valores antiguos y nuevos. También hay que proporcionar el esquema de la relación *traza_matricula*.

**b.** ¿Puede la implementación anterior garantizar que las actualizaciones realizadas por administradores maliciosos de la base de datos (o por alguien que consiga la contraseña del administrador) se hallen en la traza de auditoría? Explique su respuesta.

**9.24.** Los hackers pueden lograr hacer creer al usuario que su sitio web es realmente un sitio web en que el usuario confía (como el de un banco o el de una tarjeta de crédito). Esto puede lograrse mediante mensajes engañosos de correo electrónico o, incluso, mediante la reconfiguración de la infraestructura de la red y el redireccionamiento del tráfico de red destinado a, por ejemplo, mibanco.com, hacia el sitio de los hackers. Si se introduce el nombre de usuario y la contraseña en el sitio de los hackers, ese sitio puede registrarlo y usarlo posteriormente para entrar en la cuenta en el sitio verdadero. Cuando se usa una URL como https://mibanco.com, se usa el protocolo HTTPS para evitar esos ataques. Explique la manera en que el protocolo puede usar los certificados digitales para comprobar la autenticidad del sitio.

**9.25.** Explique qué es el sistema de autenticación de respuesta por desafío. ¿Por qué es más seguro que los sistemas tradicionales basados en las contraseñas?

## Sugerencias de proyectos

Cada uno de los proyectos que aparecen a continuación es un proyecto de grandes dimensiones. La dificultad de cada proyecto puede ajustarse fácilmente añadiendo o eliminando características.

**Proyecto 9.1** Elija su sitio web interactivo favorito, como Bebo, Blogger, Facebook, Flickr, Last.FM, Twitter, Wikipedia; son solo algunos ejemplos, hay muchos más. La mayoría de estos sitios gestionan una gran cantidad de datos y usan bases de datos para almacenarlos y procesarlos. Implemente un subconjunto de la funcionalidad del sitio web que haya elegido. Claramente, implementar solamente un subconjunto significativo de las características de un sitio queda más allá del proyecto de un curso, pero es posible definir un conjunto de características que resulte interesante implementar, pero suficientemente pequeño para un proyecto de curso.

La mayoría de los sitios web más populares usan extensamente JavaScript para crear interfaces. No sea muy ambicioso inicialmente en su proyecto, al menos inicialmente, ya que requiere mucho tiempo construir esas interfaces y después añadir las característica a las mismas. Utilice un framework de desarrollo de aplicaciones web, o bibliotecas de JavaScript disponibles en la web, como la biblioteca Yahoo User Interface, para acelerar el desarrollo.

**Proyecto 9.2** Cree un «mashup» que use servicios como las API de mapas de Google o de Yahoo para crear un sitio web interactivo. Por ejemplo, la API de mapas proporciona una forma de mostrar un mapa en una página web, en la que se puede superponer otra información. Podría implementar un sistema de recomendación de restaurantes en el que los usuarios contribuyen con información sobre los restaurantes, como su ubicación, tipo de cocina, intervalo de precios y valoración. Los resultados de búsquedas de los usuarios se podrían mostrar en el mapa. Podría añadir características del tipo de Wikipedia, como que los usuarios puedan añadir información y editar la que han añadido otros usuarios, junto con moderadores que puedan realizar revisiones frente a actualizaciones malintencionadas. También podría implementar características sociales, como dar más importancia a las valoraciones que hacen los amigos.

**Proyecto 9.3** Su universidad probablemente use un sistema de gestión de cursos como Moodle, Blackboard o WebCT. Implemente un subconjunto de la funcionalidad de un sistema de gestión de cursos. Por ejemplo, podría proporcionar el envío de tareas y su corrección, incluyendo mecanismos para que los estudiantes y los profesores hablen sobre las notas de una determinada tarea. También podría implementar un mecanismo para realizar encuestas para obtener realimentación.

**Proyecto 9.4** Considere el esquema E-R del Ejercicio práctico 7.3 (Capítulo 7), que representa la información de los equipos de una liga. Diseñe e implemente un sistema basado en web para introducir, actualizar y examinar los datos.

**Proyecto 9.5** Diseñe e implemente un sistema de carro de la compra que permita a los compradores reunir artículos en un carro (decida la información que se proporciona de cada artículo) y comprarlos todos juntos. Puede extender y usar el esquema E-R del Ejercicio 7.20 del Capítulo 7. Conviene comprobar la disponibilidad del artículo y tratar los artículos no disponibles del modo que se considere adecuado.

**Proyecto 9.6** Diseñe e implemente un sistema basado en web para registrar la información sobre las matrículas y las notas de los estudiantes para los cursos de una universidad.

**Proyecto 9.7** Diseñe e implemente un sistema que permita el registro de la información de rendimiento de las asignaturas; concretamente, las notas otorgadas a cada estudiante en cada trabajo o examen de cada asignatura, y el cálculo de una suma (ponderada) de las notas para obtener las notas totales de la asignatura. El número de trabajos o exámenes no debe estar predefinido; es decir, se pueden añadir más trabajos o exámenes en cualquier momento. El sistema también debe soportar la clasificación, permitiendo que se especifiquen separadores para varias notas.

Puede que también se desee integrarlo en el sistema de matrícula de los estudiantes del Proyecto 9.6 (que quizás implemente otro equipo de proyecto).

**Proyecto 9.8** Diseñe e implemente un sistema basado en web para la reserva de aulas de una universidad. Se debe soportar la reserva periódica (días y horas fijas cada semana para un semestre completo). También debe soportar la cancelación de aulas concretas de una reserva periódica.

Puede que también se desee integrarlo con el sistema de matrícula de estudiantes del Proyecto 9.6 (que quizás implemente otro equipo de proyecto) de modo que las aulas puedan reservarse para asignaturas y las cancelaciones de una clase o las reservas de clases adicionales puedan anotarse en una sola interfaz, se reflejen en la reserva de aulas y se comuniquen a los estudiantes por correo electrónico.

**Proyecto 9.9** Diseñe e implemente un sistema para gestionar exámenes de tipo test (con varias respuestas posibles) en línea. Se debe soportar el aporte distribuido de preguntas (por los profesores ayudantes, por ejemplo), la edición de las preguntas por quien esté a cargo de la asignatura y la creación de exámenes a partir del conjunto de preguntas disponible. También se deben poder suministrar los exámenes en línea, bien a una hora fija para todos los estudiantes o a cualquier hora pero con un límite de tiempo desde el comienzo hasta el final (se pueden soportar las dos opciones o solo una de ellas) y dar a los estudiantes información sobre su puntuación al final del tiempo concedido.

**Proyecto 9.10** Diseñe e implemente un sistema para gestionar el servicio de correo electrónico de los clientes. El correo entrante va a un buzón común. Hay una serie de agentes de atención al cliente que responden el correo electrónico. Si el mensaje es parte de una serie de respuestas (que se siguen mediante el campo *responder a* del correo electrónico) es preferible que responda el mensaje el mismo agente que lo haya respondido anteriormente. El sistema debe realizar un seguimiento de todo el correo entrante y de las respuestas, de modo que los agentes puedan ver el historial de preguntas de cada cliente antes de responder a los mensajes.

**Proyecto 9.11** Diseñe e implemente un mercado electrónico sencillo en el que los artículos puedan clasificarse para su compra o venta en varias categorías (que deben formar una jerarquía). Puede que también se deseen soportar los servicios de alerta, en los que los usuarios pueden registrar su interés por los artículos de una categoría concreta, quizás con otras restricciones añadidas, sin anunciar su interés de manera pública; después, cuando alguno de esos artículos se ofrece a la venta, se les notifica.

**Proyecto 9.12** Diseñe e implemente un sistema de grupos de noticias basado en web. Los usuarios deben poder suscribirse a los grupos de noticias y leer los artículos de esos grupos. El sistema hace un seguimiento de los artículos que el usuario ha leído, de modo que no vuelvan a mostrarse. También hay que permitir la búsqueda de artículos antiguos. Es posible que también se desee ofrecer un servicio de valoración de los artículos, de modo que los artículos con puntuación elevada se destaquen, lo que permite al lector ocupado saltarse los artículos con baja puntuación.

**Proyecto 9.13** Diseñe e implemente un sistema basado en web para gestionar una clasificación deportiva. Se registra mucha gente y se les da alguna clasificación inicial (quizá basada en resultados históricos). Cualquier usuario puede retar a cualquier otro y las clasificaciones se ajustan según los resultados. Un sistema sencillo para ajustar las clasificaciones se limita a pasar al ganador por delante del perdedor en la clasificación, en caso de que el ganador estuviera por detrás. Se puede intentar inventar sistemas de ajustes de la clasificación más complicados.

**Proyecto 9.14** Diseñe e implemente un servicio de listados de publicaciones. El servicio debe permitir la introducción de información sobre las publicaciones, como pueden ser título, autores, año en que apareció la publicación, páginas, etc. Los autores deben ser una entidad separada con atributos como nombre, institución, departamento, correo electrónico, dirección y página web.

La aplicación debe soportar varias vistas de los mismos datos. Por ejemplo, se deben facilitar todas las publicaciones de un autor dado (ordenadas por año, por ejemplo), o todas las publicaciones de los autores de una institución o de un departamento dado. También se debe soportar la búsqueda por palabras clave, tanto en la base de datos global como en cada una de las vistas.

**Proyecto 9.15** Una tarea frecuente en cualquier organización es la recogida de información estructurada de un grupo de personas. Por ejemplo, puede que un director necesite pedir a los empleados que introduzcan sus planes de vacaciones, un profesor puede que desee obtener la respuesta de los estudiantes a un tema concreto, el estudiante que organiza un evento puede que desee permitir que otros estudiantes se inscriban en él o alguien puede desear organizar una votación en línea sobre un tema concreto.

Cree un sistema que permita a los usuarios crear con facilidad eventos de reunión de información. Al crear un evento, su creador debe definir quién tiene derecho a participar; para ello, el sistema debe conservar la información de los usuarios y permitir predicados que definan subconjuntos de usuarios. El creador del evento debe poder especificar conjuntos de datos (con tipos, valores predeterminados y controles de validación) que tendrán que proporcionar los usuarios. El evento debe tener una fecha límite asociada y la posibilidad de enviar recordatorios a los usuarios que todavía no hayan remitido su información. Se puede dar al creador del evento la opción de hacer que se cumpla la fecha límite de manera automática con base en una fecha u hora específicas, o que decida iniciar una sesión y declarar que se ha superado la fecha límite. Se deben generar estadísticas sobre los datos enviados, para ello se permitirá al creador del evento que cree resúmenes sencillos de la información aportada. El creador del evento puede elegir hacer públicos parte de los resúmenes, visibles a todos los usuarios, de manera continua (por ejemplo, el número de personas que ha respondido) o una vez superada la fecha límite (por ejemplo, la puntuación promedio obtenida).

**Proyecto 9.16** Cree una biblioteca de funciones para simplificar la creación de interfaces web. Hay que implementar, como mínimo, las siguientes funciones: una función que muestre el conjunto de resultados JDBC (en formato tabular), funciones que creen diferentes tipos de datos de texto y numéricos (con criterios de validación como el tipo de entrada y un intervalo opcional, aplicado en el cliente mediante el código Javascript correspondiente), funciones que introduzcan los valores de la fecha y de la hora (con valores predeterminados) y funciones que creen elementos de menú de acuerdo con un conjunto de resultados. Para obtener más nota se debe permitir al usuario configurar parámetros de estilo, como los colores y las fuentes, y proporcionar soporte a la paginación en las tablas (se pueden usar parámetros ocultos para los formularios para especificar la página que se debe mostrar). Cree una aplicación de bases de datos de ejemplo para ilustrar el uso de estas funciones.

**Proyecto 9.17** Diseñe e implemente un sistema de agenda multiusuario basado en web. El sistema debe realizar un seguimiento de las citas de cada persona, con eventos de ocurrencia múltiple, como reuniones semanales, eventos compartidos (en los que las actualizaciones realizadas por el creador del evento se reflejan en las agendas de todos los que comparten ese evento). Proporciónese la notificación de los eventos por correo electrónico. Para obtener más nota se debe implementar un servicio web que pueda usar un programa de recordatorios que se ejecute en la máquina cliente.

## Herramientas

El desarrollo de aplicaciones web necesita distintas herramientas de software como servidores de aplicaciones, compiladores y editores de lenguajes de programación como Java o C#, y otras herramientas opcionales como los servidores web. Existen distintos entornos de desarrollo integrado que proporcionan soporte para el desarrollo de aplicaciones web. Los dos IDE de código abierto más populares son Eclipse, desarrollado por IBM, y Netbeans, desarrollado por Sun Microsystems. Visual Studio de Microsoft es el IDE más usado en mundo Windows.

Apache Tomcat (jakarta.apache.org), Glassfish (glassfish.dev.java.net), JBoss (jboss.org) y Caucho's Resin (www.caucho.com), son servidores de aplicaciones que soportan servlets y JSP. El servidor web Apache (apache.org) es el servidor web más usado en la actualidad. IIS (Internet Information Services) de Microsoft es un servidor web y de aplicaciones ampliamente usado en plataformas de Microsoft Windows que soporta ASP.NET (msdn.microsoft.com/asp.net/) de Microsoft.

El software WebSphere (www.software.ibm.com) de IBM proporciona un conjunto de herramientas para el desarrollo y despliegue de aplicaciones web, incluyendo un servidor de aplicaciones, un IDE, integración de aplicaciones middleware, software de gestión de procesos de negocio y herramientas de administración de sistemas.

Algunas de las herramientas anteriores son de código abierto que se pueden usar de forma gratuita, algunas son gratuitas para uso no comercial o uso personal, mientras que otras son de pago. Véanse los sitios web respectivos para más información.

La biblioteca de JavaScript Yahoo! User Interface (YUI) (developer.yahoo.com/yui) se usa extensamente para la creación de programas en JavaScript que se ejecutan en distintos navegadores.

## Notas bibliográficas

La información sobre los servlets, incluidos los tutoriales, las especificaciones de las normas y el software están disponibles en java.sun.com/products/servlet. La información sobre JSP está disponible en java.sun.com/products/jsp. La información sobre las bibliotecas de etiquetas de JSP también se puede encontrar en esa URL. La información sobre la estructura .NET y sobre el desarrollo de aplicaciones web mediante ASP.NET se puede hallar en msdn.microsoft.com.

Atreya et ál. [2002] proporcionan un tratamiento del nivel de los libros de texto de las firmas digitales, incluidos los certificados digitales X.509 y de la infraestructura de clave pública.