CAPÍTULO

6 INTEGRIDAD Y SEGURIDAD

Las restricciones de integridad proporcionan un medio de asegurar que las modificaciones hechas a la base de datos por los usuarios autorizados no provoquen la pérdida de la consistencia de los datos. Por tanto, las restricciones de integridad protegen a la base de datos contra los daños accidentales.

En el Capítulo 2 ya se ha visto una modalidad de restricciones de integridad para el modelo E-R. Estas restricciones eran de los tipos siguientes:

- **Declaración de claves** – la estipulación de que ciertos atributos pueden formar una clave para un conjunto de entidades determinado.
- **Forma de la relación** – de varios a varios, de uno a varios, de uno a uno.

En general, la restricción de integridad puede ser un predicado arbitrario referente a la base de datos. Sin embargo, los predicados arbitrarios pueden resultar complicados de verificar. En consecuencia, lo habitual es limitarse a restricciones de integridad que puedan verificarse con una sobrecarga mínima. En los apartados 6.1 y 6.2 se estudian estas formas de restricciones de integridad y una forma más compleja en el Apartado 6.3. En el Capítulo 7 se estudia otra forma de restricción de integridad, denominada «dependencia funcional», que se usa principalmente en el proceso del diseño de esquemas.

En el Apartado 6.4 se estudian los *disparadores*, que son instrucciones que el sistema ejecuta automáticamente como efecto colateral de una modificación de la base de datos. Los disparadores se usan para asegurar algunos tipos de integridad.

Además de la protección contra la introducción accidental de inconsistencia, puede ser necesario proteger los datos almacenados en la base de datos frente a accesos no autorizados y destrucción o alteración malintencionada. En los apartados 6.5 hasta el 6.7 se examinan formas en que se puede hacer un mal uso de los datos o hacerlos intencionadamente inconsistentes, y se presentan mecanismos de seguridad para protegerse contra ello.

6.1. RESTRICCIONES DE LOS DOMINIOS

Se ha visto que hay que asociar a cada atributo un dominio de valores posibles. En el Capítulo 4 se vieron varios tipos de dominios estándar, tales como los enteros, caracteres y fecha/tiempo en SQL. La declaración de que un atributo pertenezca a un determinado dominio actúa como una restricción sobre los valores que puede tomar. Las restricciones de los dominios son la forma más simple de restricción de integridad. El sistema las verifica fácilmente siempre que se introduce en la base de datos un nuevo elemento de datos.

Es posible que varios atributos tengan el mismo dominio. Por ejemplo, los atributos *nombre-cliente* y *nombre-empleado* pueden tener el mismo dominio: el conjunto de los nombres de persona. Sin embargo, los dominios de *saldo* y de *nombre de la sucursal* deben ser, ciertamente, diferentes. Quizá resulte menos evidente si *nombre-cliente* y *nombre-sucursal* deben tener el mismo dominio. En el nivel de implementación, tanto los nombres de los clientes como los de las sucursales son cadenas de caracteres. Sin embargo, normalmente no se considerará que la consulta «Hallar todos los clientes que tengan el nombre de una sucursal» tenga sentido. Por tanto, si se considera la base de datos desde el punto de vista teórico, en vez de hacerlo desde el punto de vista físico, *nombre-cliente* y *nombre-sucursal* deben tener dominios diferentes.

De la discusión anterior se puede deducir que una definición adecuada de las restricciones de los dominios no sólo permite verificar los valores introducidos en la base de datos, sino también examinar las consultas para asegurarse de que tengan sentido las comparaciones que hagan. El principio subyacente a los dominios de los atributos es parecido al de los tipos de las variables en los lenguajes de programación. Los lenguajes de programación con tipos estrictos permiten al compilador examinar el programa con mayor detalle.

La cláusula **create domain** se puede usar para definir nuevos dominios. Por ejemplo, las instrucciones:

create domain *Euros* **numeric**(12,2)
create domain *Dólares* **numeric**(12,2)

definen los dominios *Euros* y *Dólares* como números decimales con un total de 12 dígitos, dos de los cuales se sitúan después de la coma decimal. Un intento de asignar un valor de tipo *Dólares* a una variable de tipo *Euros* resultaría en un error sintáctico, aunque ambos tengan el mismo tipo numérico. Tal asignación probablemente es debida a un error del programador, en el que este olvidó las diferencias de cambio. La declaración de diferentes dominios para diferentes monedas ayuda a detectar errores.

Los valores de un dominio pueden ser *convertidos* a otro dominio. Si el atributo *A* de la relación *r* es de tipo *Euros*, se puede convertir a *Dólares* escribiendo:

cast *r.A* **as** *Dólares*

En una aplicación real se multiplicaría *r.A* por el factor de cambio antes de convertirlo a dólares. SQL también proporciona las cláusulas **drop domain** y **alter domain** para borrar o modificar dominios que se hayan declarado anteriormente.

La cláusula **check** de SQL permite restringir los dominios de maneras poderosas que no permiten la mayor parte de los sistemas de tipos de los lenguajes de programación. Concretamente, la cláusula **check** permite al diseñador del esquema especificar un predicado que debe satisfacer cualquier valor asignado a una variable cuyo tipo sea el dominio. Por ejemplo, una cláusula **check** puede asegurar que un dominio de sueldo por hora sólo permita valores mayores que un valor especificado (como puede ser el sueldo mínimo), tal y como se muestra aquí:

create domain *sueldo-por-hora* **numeric**(5,2)
constraint *comprobación-valor-sueldo*
check(**value** ≥ 4.00)

El dominio *sueldo-por-hora* tiene una restricción que asegura que el sueldo por hora sea mayor que 4,00. La orden **constraint** *comprobación-valor-sueldo* es opcional y se utiliza para dar a la restricción el nombre de *comprobación-valor-sueldo*. El nombre se utiliza para indicar la restricción violada por una actualización.

La cláusula **check** también puede utilizarse para restringir un dominio para que no contenga valores nulos, como se muestra aquí:

create domain *número-cuenta* **char**(10)
constraint *comprobación-número—cuenta-nulo*
check(**value not null**)

En este otro ejemplo el dominio se puede limitar para que contenga sólo un conjunto especificado de valores usando la cláusula **in**:

create domain *tipo-cuenta* **char**(10)
constraint *comprobación-tipo-cuenta*
check(**value in** ('Corriente', 'Ahorro'))

Las condiciones **check** anteriores se puede comprobar muy fácilmente cuando se inserta o modifica una tupla. Sin embargo, en general, las condiciones **check** pueden ser muy complejas (y difíciles de comprobar) dado que se permiten subconsultas que se refieren a otras relaciones en la condición **check**. Por ejemplo, esta restricción se podría especificar sobre la relación *préstamo*:

check (nombre-sucursal **in**
(**select** nombre-sucursal **from** sucursal))

La condición **check** verifica que *nombre-sucursal* en cada tupla en la relación *préstamo* es realmente el nombre de una sucursal de la relación *cuenta*. Así, la condición no sólo se debe comprobar cuando se inserte o modifique *préstamo*, sino también cuando cambie la relación *sucursal* (en este caso, cuando se borre o modifique *cuenta*).

La restricción anterior es realmente un ejemplo de una clase de restricciones denominadas restricciones de *integridad referencial*. En el Apartado 6.2 se estudian estas restricciones y la forma de especificarlas de forma más sencilla en SQL.

Las condiciones **check** complejas pueden ser útiles cuando se desee asegurar la integridad de los datos, pero se deben usar con cuidado, dado que pueden ser costosas de comprobar.

6.2. INTEGRIDAD REFERENCIAL

A menudo se desea asegurar que un valor que aparece en una relación para un conjunto de atributos determinado aparezca también en otra relación para un cierto conjunto de atributos. Esta condición se denomina **integridad referencial**.

6.2.1. Conceptos básicos

Considérese un par de relaciones r (R) y s (S) y la reunión natural $r \bowtie s$. Puede haber una tupla t_r de r que no se reúna con ninguna tupla de s. Es decir, no hay ningún t_s en s

tal que $t_r [R \cap S] = t_s [R \cap S]$. Estas tuplas se denominan *colgantes*. Las tuplas colgantes pueden ser aceptables en función del conjunto de entidades o de relaciones que se esté modelando. En el Apartado 3.5.2 se tomó en consideración una forma de reunión modificada —la reunión externa— para operar con las relaciones que contengan tuplas colgantes. En este caso el motivo de preocupación no son las consultas, sino el momento en que se desea permitir que haya tuplas colgantes en la base de datos.

Supóngase que hay una tupla t_1 en la relación *cuenta* con t_1[*nombre-sucursal*] = «As Pontes» pero que no hay ninguna tupla de la relación *sucursal* para la sucursal de As Pontes. Esta situación no es deseable. Se espera que la relación *sucursal* muestre una relación de todas las sucursales bancarias. Por tanto, la tupla t_1 hará referencia a una cuenta de una sucursal que no existe. Evidentemente, es preferible tener una restricción de integridad que prohíba las tuplas colgantes de este tipo.

Sin embargo, algunos casos de tuplas colgantes pueden resultar convenientes. Supóngase que hay una tupla t_2 de la relación *sucursal* con t_2[*nombre-sucursal*] = «Gijón» pero que no hay ninguna tupla de la relación *cuenta* para la sucursal de Gijón. En este caso hay una sucursal que no tiene ninguna cuenta. Aunque esta situación no es frecuente, puede producirse cuando se abre una sucursal o cuando está a punto de cerrar. Por tanto, no es deseable prohibir esta situación.

La distinción entre estos dos ejemplos se basa en dos hechos:

- El atributo *nombre-sucursal* de *Esquema-cuenta* es una clave externa que hace referencia a la clave primaria de *Esquema-sucursal*.
- El atributo *nombre-sucursal* de *Esquema-sucursal* no es una clave externa.

(Recuérdese del Apartado 3.1.3 que una clave externa es un conjunto de atributos del esquema de una relación que forma la clave primaria de otro esquema.)

En el ejemplo de As Pontes la tupla t_1 de *cuenta* tiene un valor en la clave externa *nombre-sucursal* que no aparece en *sucursal*. En el ejemplo de la sucursal de Gijón, la tupla t_2 de *sucursal* tiene un valor de *nombre-sucursal* que no aparece en *cuenta*, pero *nombre-sucursal* no es una clave externa. Por tanto, la distinción entre los dos ejemplos de tuplas colgantes es la presencia de una clave externa.

Sean $r_1(R_1)$ y $r_2(R_2)$ dos relaciones con las claves primarias K_1 y K_2, respectivamente. Se dice que un subconjunto α de R_2 es una **clave externa** que hace referencia a K_1 de la relación r_1 si se exige que para cada t_2 de r_2 haya una tupla t_1 de r_1 tal que $t_1[K_1] = t_2[\alpha]$. Las exigencias de este tipo se denominan **restricciones de integridad referencial** o **dependencia de subconjunto**. El último término se debe a que esta última restricción de integridad referencial puede escribirse como $\Pi_\alpha (r_2) \subseteq \Pi_{K_1} (r_1)$. Obsérvese que, para que una restricción de integridad referencial tenga sentido, α debe ser igual a K_1, o bien α y K_1 deben ser conjuntos compatibles de atributos.

6.2.2. Integridad referencial en el modelo E-R

Las restricciones de integridad referencial aparecen con frecuencia. Si se obtiene el esquema de la base de datos relacional creando tablas a partir de los diagramas E-R, tal y como se vio en el Capítulo 2, cada relación que proceda de un conjunto de relaciones tendrá restricciones de integridad referencial. En la Figura 6.1 se muestra un conjunto n-ario de relaciones R, que relaciona los conjuntos de entidades $E_1, E_2, \ldots, E_n$. K_i denota la clave primaria de E_i. Los atributos del esquema de la relación del conjunto de relaciones R incluyen $K_1 \cup K_2 \cup \ldots \cup K_n$. Cada K_i del esquema de R es una clave externa que lleva a una restricción de integridad referencial.

Otra fuente de restricciones de integridad referencial son los conjuntos de entidades débiles. Recuérdese del Capítulo 2 que el esquema de la relación de un conjunto de entidades débiles debe incluir la clave primaria del conjunto de entidades del que este depende. Por tanto, el esquema de la relación de cada conjunto de entidades débiles incluye una clave externa que lleva a una restricción de integridad referencial.

6.2.3. Modificación de la base de datos

La modificación de la base de datos puede ocasionar violaciones de la integridad referencial. A continuación se describe la comprobación que debe hacerse para cada tipo de modificación de la base de datos para preservar la siguiente restricción de integridad referencial:

$$\Pi_\alpha (r_2) \subseteq \Pi_K (r_1)$$

- **Insertar.** Si se inserta una tupla t_2 en r_2, el sistema debe asegurar que hay una tupla t_1 de r_1 tal que $t_1[K] = t_2[\alpha]$. Es decir,

$$t_2[\alpha] \in \Pi_K (r_1)$$

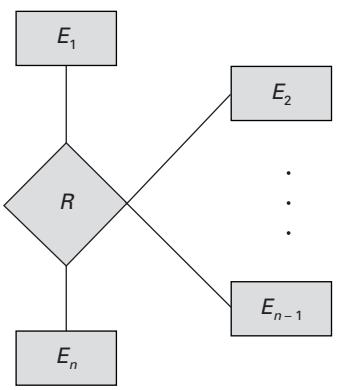

FIGURA 6.1. Un conjunto de relaciones N-ario.

- **Borrar.** Si se borra una tupla t_1 de r_1 el sistema debe calcular el conjunto de tuplas de r_2 que hacen referencia a r_1:

  $$\sigma_{\alpha = t_1[K]}(r_2)$$

  Si este conjunto no es el conjunto vacío, o bien se rechaza la orden borrar como error, o bien se deben borrar las tuplas que hacen referencia a t_1. La última solución puede llevar a borrados en cascada, dado que las tuplas pueden hacer referencia a tuplas que hagan referencia a t_1, etcétera.

- **Actualizar.** Hay que considerar dos casos: las actualizaciones de la relación que realiza la referencia (r_2) y las actualizaciones de la relación a la que se hace referencia (r_1).

  — Si se actualiza la tupla t_2 de la relación r_2 y esta actualización modifica valores de la clave externa α, se realiza una comprobación parecida a la del caso de la inserción. Si t'_2 denota el nuevo valor de la tupla t_2, el sistema debe asegurar que

  $$t'_2[\alpha] \in \Pi_K(r_1)$$

  — Si se actualiza la tupla t_1 de la relación r_1 y esta actualización modifica valores de la clave primaria (K), se realiza una comprobación parecida a la del caso del borrado. El sistema debe asegurar que

  $$\sigma_{\alpha = t_1[K]}(r_2)$$

  utilizando el valor anterior de t_1 (el valor antes de que se lleve a cabo la actualización). Si este conjunto no es el conjunto vacío, la actualización se rechaza como error o se ejecuta en cascada de manera parecida al borrado.

6.2.4. Integridad referencial en SQL

Las claves primarias pueden especificarse como parte de la instrucción **create table** de SQL usando la cláusula **foreign key**. Se ilustrarán las declaraciones de clave externa usando la definición del LDD de SQL de parte de la base de datos bancaria, mostrada en la Figura 6.2.

De manera predeterminada, una clave externa referencia los atributos que forman la clave primaria de la tabla referenciada. SQL también soporta una versión de la cláusula **references**, donde se puede especificar explícitamente una lista de atributos de la relación referenciada. Esta lista se debe declarar como clave candidata de la relación referenciada.

Se puede usar la siguiente forma abreviada como parte de la definición de un atributo para declarar que el atributo forma una clave externa:

nombre-sucursal **char**(15) **references** *sucursal*

Cuando se viola una restricción de integridad referencial, el procedimiento normal es rechazar la acción

```
create table cliente
    (nombre-cliente     char(20),
    calle-cliente       char(30),
    ciudad-cliente      char(30),
    primary key (nombre-cliente))

create table sucursal
    (nombre-sucursal    char(15),
    ciudad-sucursal     char(30),
    activo              integer,
    primary key (nombre-sucursal),
    check (activo > = 0))

create table cuenta
    (número-cuenta      char(10),
    nombre-sucursal     char(15),
    saldo               integer,
    primary key (número-cuenta),
    foreign key (nombre-sucursal) references sucursal,
    check (saldo >= 0))

create table impositor
    (nombre-cliente     char(20),
    número-cuenta       char(10),
    primary key (nombre-cliente, número-cuenta),
    foreign key (nombre-cliente) references cliente,
    foreign key (número-cuenta) references cuenta)
```

FIGURA 6.2. Definición en SQL de los datos de parte de la base de datos bancaria.

que provocó la violación. Sin embargo, la cláusula **foreign key** puede especificar que si una acción de borrado o de actualización de la relación a la que hace referencia viola la restricción, en lugar de rechazar la acción, hay que adoptar medidas para modificar la tupla de la relación que hace la referencia con objeto de restaurar la restricción. Considérese la siguiente definición de una restricción de integridad de la relación *cuenta*:

```
create table cuenta
    (…
    foreign key (nombre-sucursal) references sucursal
                                  on delete cascade
                                  on update cascade,
    …)
```

Debido a la cláusula **on delete cascade** asociada con la declaración de la clave externa, si un borrado de una tupla de *sucursal* da lugar a que se viole la restricción de integridad referencial, el sistema no rechaza el borrado. En su lugar, el borrado se realiza en «cascada» en la relación *cuenta*, borrando la tupla que hace referencia a la sucursal que se borró. De modo parecido, no se rechaza la actualización de un campo al que haga referencia la restricción si viola esta; en vez de eso, el campo *nombre-sucursal* de las tuplas que realizan la referencia de *cuenta* se actualizan también al nuevo valor. SQL también permite que la cláusula **foreign key** especifique una acción diferente a **cascade** si se viola la restricción: el campo que hace la referencia (en este caso, *nombre-*

sucursal) se puede establecer en nulo o darle un valor predeterminado para el dominio (usando **set default**).

Si hay una cadena de dependencias de claves externas entre varias relaciones, un borrado o una actualización en uno de sus extremos puede propagarse por toda la cadena. En el Ejercicio 6.4 se toma en consideración un caso interesante en el que la restricción **foreign key** de una relación hace referencia a esa misma relación. Si una actualización o borrado en cascada crea una violación de la restricción que no puede resolverse mediante una nueva operación en cascada, el sistema aborta la transacción. En consecuencia, todas las modificaciones generadas por la transacción y por sus acciones en cascada se deshacen.

En SQL la semántica de las claves se complica por el hecho de que SQL permite valores **nulos**. Los atributos de las claves externas pueden ser nulos, dado que no se han declarado como no nulos. Si todas las columnas de una clave externa contienen nulos para una tupla en concreto, se usa para esta tupla la definición usual de las restricciones de clave externa. Si alguna de las columnas contiene un valor nulo, la tupla se define automáticamente como que cumple la restricción.

Puede que esta definición no sea siempre la mejor elección, así que SQL proporciona constructores que permiten cambiar el comportamiento con los valores nulos; aquí no se tratan estos constructores. Para evitar esta complejidad, es mejor asegurarse de que todas las columnas con especificaciones **foreign key** se declaren como no nulas.

Las transacciones pueden consistir en varios pasos, y las restricciones de integridad se pueden violar temporalmente después de un paso, pero en un paso posterior la violación puede desaparecer. Por ejemplo, supóngase que se tiene la relación *personacasada* con clave primaria *nombre* y un atributo *cónyuge*, y supónganse que *cónyuge* es una clave externa de *personacasada*. Es decir, la restricción dice que el atributo *cónyuge* debe contener un nombre que aparezca en la tabla *persona*. Supóngase que se desea añadir el hecho de que Juan y María están casados insertando dos tuplas, una para Juan y otra para María en la relación anterior. La inserción de la primera tupla violaría la restricción de clave externa, independientemente de cuál de las dos tuplas se haya insertado primero. Después de que se inserte la segunda tabla, la restricción de clave externa se volvería a cumplir.

Para manejar estas situaciones las restricciones de integridad se comprueban al final de la transacción en lugar de en los pasos intermedios[1].

6.3. ASERTOS

Un **aserto** es un predicado que expresa una condición que se desea que la base de datos satisfaga siempre. Las restricciones de dominio y las de integridad referencial son formas especiales de los asertos. Se ha prestado una atención especial a estos tipos de asertos porque se pueden verificar con facilidad y se aplican a una gran variedad de aplicaciones de bases de datos. Sin embargo, hay muchas restricciones que no se pueden expresar utilizando únicamente estas formas especiales. Ejemplos de estas restricciones pueden ser

- La suma de todos los importes de los préstamos de cada sucursal debe ser menor que la suma de todos los saldos de las cuentas de esa sucursal.
- Cada préstamo tiene al menos un cliente que tiene una cuenta con un saldo mínimo de 1.000 €.

En SQL los asertos adoptan la forma

create assertion <nombre-aserto> **check** <predicado>

Las dos restricciones mencionadas pueden escribirse tal y como se muestra a continuación. Dado que SQL no proporciona ningún mecanismo «para todo X, $P(X)$» (donde P es un predicado), no queda más remedio que implementarlo utilizando su equivalente «no existe X tal que no $P(X)$», que puede escribirse en SQL.

```
create assertion restricción-suma check
    (not exists (select * from sucursal
        where (select sum(importe) from préstamo
            where préstamo.nombre-sucursal
                = sucursal.nombre-sucursal)
          >= (select sum (importe) from cuenta
            where préstamo.nombre-sucursal
                = sucursal.nombre-sucursal)))

create assertion restricción-saldo check
    (not exists (select * from préstamo
        where not exists (select *
            from prestatario, impositor, cuenta
            where préstamo.número-préstamo
                = prestatario.número-préstamo
              and prestatario.nombre-prestatario
                = impositor.nombre-cliente
              and impositor.número-cuenta
                = cuenta.número-cuenta
              and cuenta.saldo >= 1000)))
```

[1] El problema de este ejemplo se puede resolver de otra forma si el atributo *cónyuge* puede ser nulo: se ponen los atributos *cónyuge* a nulo al insertar las tuplas de Juan y María, y se actualizan más tarde. Sin embargo, esta técnica no es aconsejable y no funciona si los atributos no pueden tomar el valor nulo.

Cuando se crea un aserto el sistema comprueba su validez. Si el aserto es válido, sólo se permiten las modificaciones posteriores de la base de datos que no hagan que se viole el aserto. Esta comprobación puede introducir una sobrecarga importante si se han realizado asertos complejos. Por tanto, los asertos deben utilizarse con mucha cautela. La elevada sobrecarga debida a la comprobación y al mantenimiento de los asertos ha llevado a algunos desarrolladores de sistemas a soslayar el soporte para los asertos generales, o bien a proporcionar formas especializadas de aserto que resultan más sencillas de verificar.

6.4. DISPARADORES

Un *disparador* es una orden que el sistema ejecuta de manera automática como efecto secundario de la modificación de la base de datos. Para diseñar un mecanismo disparador hay que cumplir dos requisitos:

1. Especificar las condiciones en las que se va a ejecutar el disparador. Esto se descompone en un *evento* que causa la comprobación del disparador y una *condición* que se debe cumplir para ejecutar el disparador.
2. Especificar las *acciones* que se van a realizar cuando se ejecute el disparador.

Este modelo de disparadores se denomina modelo **evento-condición-acción**.

La base de datos almacena disparadores como si fuesen datos normales, por lo que son persistentes y accesibles para todas las operaciones de la base de datos. Una vez se almacena un disparador en la base de datos, el sistema de base de datos asume la responsabilidad de ejecutarlo cada vez que ocurra el evento especificado y se satisfaga la condición correspondiente.

6.4.1. Necesidad de los disparadores

Los disparadores son mecanismos útiles para alertar a los usuarios o para realizar de manera automática ciertas tareas cuando se cumplen determinadas condiciones. A modo de ejemplo supóngase que, en lugar de permitir saldos de cuenta negativos, el banco trata los descubiertos dejando a cero el saldo de las cuentas y creando un préstamo por el importe del descubierto. Este préstamo recibe un número de préstamo idéntico al número de cuenta que ha tenido el descubierto. En este ejemplo la condición para ejecutar el disparador es una actualización de la relación *cuenta* que dé lugar a un valor negativo de *saldo*. Supóngase que Santos retiró cierta cantidad de dinero de una cuenta que dio lugar a que el saldo de la cuenta fuera negativo. *t* denota la tupla de la cuenta con un valor negativo de *saldo*. Las acciones que hay que emprender son las siguientes:

- Insertar una nueva tupla *s* a la relación *préstamo* con

  s[*nombre-sucursal*] = t[*nombre-sucursal*]
  s[*número-préstamo*] = t[*número-cuenta*]
  s[*importe*] = – t[*saldo*]

  (Obsérvese que, dado que t[*saldo*] es negativo, hay que cambiar el signo de t[*saldo*] para obtener el importe del préstamo – un número positivo).

- Insertar una nueva tupla *u* a la relación *prestatario* con

  u[*nombre-cliente*] = «Santos»
  u[*número-préstamo*] = t[*número-cuenta*]

- Hacer que t[*saldo*] sea 0.

Como otro ejemplo del uso de disparadores, supóngase un almacén que desee mantener un inventario mínimo de cada producto; cuando el nivel de inventario de un producto cae por debajo del nivel mínimo, se debería solicitar un pedido automáticamente. Para implementar con disparadores esta regla de negocio se haría: al modificar el nivel de inventario de un producto, el disparador debería comparar el nivel con el mínimo y si el nivel es inferior al mínimo, se añadiría un nuevo pedido a la relación *pedido*.

Obsérvese que los sistemas de disparadores en general no pueden realizar actualizaciones fuera de la base de datos, y por lo tanto, en este último ejemplo, no se puede usar un disparador para realizar un pedido en el mundo externo. En su lugar se añade un pedido a la relación *pedidos* como en el ejemplo del inventario. Se debe crear un proceso del sistema separado que se esté ejecutando constantemente que explore periódicamente la relación *pedidos* y solicite los pedidos. Este proceso del sistema anotaría las tuplas de la relación *pedidos* que se han procesado y cuándo se ha procesado cada pedido. El proceso también llevaría cuenta del despacho de pedidos y alertaría a los gestores en caso de condiciones excepcionales tales como retrasos en las entregas.

6.4.2. Disparadores en SQL

Los sistemas de bases de datos SQL usan ampliamente los disparadores, aunque antes de SQL:1999 no fueron parte de la norma. Por desgracia, cada sistema de bases de datos implementó su propia sintaxis para los disparadores, conduciendo a incompatibilidades. En la Figura 6.3 se describe la sintaxis SQL:1999 para los disparadores (que es similar a la sintaxis de los sistemas de bases de datos DB2 de IBM y de Oracle).

```
create trigger descubierto after update on cuenta
referencing new row as nfila
for each row
when nfila.saldo < 0
begin atomic
    insert into prestatario
        (select nombre-cliente, número-cuenta
        from impositor
        where nfila.número-cuenta = impositor.número-cuenta);
    insert into préstamo values
      (nfila.número-cuenta, nfila.nombre-sucursal, – nfila.saldo)
    update cuenta set saldo = 0
        where cuenta.número-cuenta = nfila.número-cuenta
end
```

FIGURA 6.3. Ejemplo de la sintaxis de SQL:1999 para los disparadores.

Esta definición de disparador especifica que el disparador se inicia *después* (**after**) de cualquier actualización de la relación *cuenta*. Una instrucción de actualización SQL podría actualizar múltiples tuplas de la relación, y la cláusula **for each row** en el código del disparador se iteraría explícitamente por cada fila actualizada. La cláusula **referencing new row as** crea una variable *nfila* (denominada **variable de transición**) que almacena el valor de una fila actualizada después de la actualización.

La instrucción **when** especifica una condición, en este caso *nfila.saldo* < 0. El sistema ejecuta el resto del cuerpo del disparador sólo para las tuplas que satisfacen la condición. La cláusula **begin atomic** … **end** sirve para encuadrar varias instrucciones SQL en una única instrucción compuesta. Las dos instrucciones **insert** en la estructura **begin** … **end** realizan las tareas específicas para la creación de nuevas tuplas en las relaciones *prestatario* y *préstamo* para representar el nuevo préstamo. La instrucción **update** sirve para establecer en 0 el saldo de la cuenta.

El evento del disparador puede tener varias formas:

- El *evento* del disparador puede ser **insert** o **delete** en lugar de **update**.

  Por ejemplo, la acción sobre el borrado (**delete**) de una cuenta podría comprobar si los tenedores de la cuenta tienen otras cuentas y, si no las tienen, borrarlas de la relación *impositor*. Se deja al lector la definición de este disparador como ejercicio (Ejercicio 6.7).

  Como otro ejemplo, si se inserta un nuevo *impositor*, la acción del disparador podría ser enviar una carta de bienvenida al impositor. Obviamente, un disparador no puede causar directamente esta acción fuera de la base de datos, pero en su lugar sí puede insertar una nueva tupla a una relación que almacene las direcciones a las que se deben enviar las cartas de bienvenida. Un proceso separado examinaría esta relación e imprimiría las cartas a enviar.
- Para las actualizaciones el disparador puede especificar columnas cuya actualización cause la ejecución del disparador. Por ejemplo, si la primera línea del disparador de descubiertos se reemplazase por

  create trigger *descubierto* **after update of** *saldo* **on** *cuenta*

  entonces el disparador se ejecutaría sólo cuando se actualizase *saldo*; las actualizaciones del resto de atributos no causarían su ejecución.
- La cláusula **referencing old row as** se puede usar para crear una variable que almacene el valor anterior de una fila actualizada o borrada. La cláusula **referencing new row as** se puede usar con las inserciones además de las actualizaciones.
- Los disparadores se pueden activar antes (**before**) del evento (**insert/delete/update**) en lugar de después (**after**) del evento.

  Estos disparadores pueden servir como restricciones extras que pueden evitar actualizaciones no válidas. Por ejemplo, si no se desea permitir descubiertos, se puede crear un disparador **before** que retroceda la transacción si el nuevo saldo es negativo.

  Como otro ejemplo, supóngase que el valor del campo de número telefónico de una tupla insertada está vacío, que indica la ausencia de un número de teléfono. Se puede definir un disparador que reemplace el valor por el valor **null**. Se puede usar la instrucción **set** para realizar estas modificaciones.

  ```
  create trigger poner-nulo before update on r
  referencing new row as nfila
  for each row
  when nfila.número-teléfono = ' '
  set nfila.número-teléfono = null
  ```
- En lugar de realizar una acción por cada fila afectada, se puede realizar una acción única para la instrucción SQL completa que causó la inserción, borrado o actualización. Para hacerlo se usa la cláusula **for each statement** en lugar de **for each row**.

  Las cláusulas **referencing old table as** o **referencing new table as** se pueden usar entonces para

hacer referencia a tablas temporales (denominadas *tablas de transición*) conteniendo todas las filas afectadas. Las tablas de transición no se pueden usar con los disparadores **before**, pero sí con los **after**, independientemente de si son disparadores de instrucciones (**statement**) o de filas (**row**).

Una instrucción SQL única se puede usar entonces para realizar varias acciones en términos de las tablas de transición.

Volviendo al ejemplo de inventario del almacén, supóngase que se tienen las siguientes relaciones:

- *inventario*(*producto*, *nivel*), que denota la cantidad actual (número/peso/volumen) del producto en el almacén.
- *nivelmínimo*(*producto*, *nivel*), que denota la cantidad mínima a mantener de cada producto.
- *nuevopedido*(*producto*, *cantidad*), que denota la cantidad de producto a pedir cuando su nivel cae por debajo del mínimo.
- *pedidos*(*producto*, *cantidad*), que denota la cantidad de producto a pedir.

Se puede usar entonces el disparador mostrado en la Figura 6.4 para solicitar un nuevo pedido del producto.

Obsérvese que se ha sido cuidadoso a la hora de realizar un pedido sólo cuando su cantidad caiga desde un nivel superior al mínimo a otro inferior. Si sólo se comprobase si el nuevo valor después de la actualización está por debajo del mínimo, se podría realizar erróneamente un pedido cuando el producto ya se ha pedido.

Muchos sistemas de bases de datos proporcionan implementaciones no estándar de disparadores, o implementan sólo algunas de las características de los disparadores. Por ejemplo, muchos de los sistemas de bases de datos no implementan la cláusula **before**, y usan la palabra clave **on** en lugar de **after**. Puede que no implementen la cláusula **referencing**. En su lugar, pueden especificar tablas de transición usando las palabras clave **inserted** o **deleted**. La Figura 6.5 ilustra cómo se escribiría el disparador de los descubiertos de las cuentas en SQLServer de Microsoft. Consúltese el manual del sistema de bases de datos que se esté usando para obtener más información sobre las características de los disparadores que soporta.

6.4.3. Cuándo no deben usarse los disparadores

Hay muchos buenos usos de los disparadores, como los que se acaban de ver en el Apartado 6.4.2, pero algunos usos se manejan mejor con otras técnicas. Por ejemplo, anteriormente, los diseñadores de sistemas usaban los disparadores para realizar resúmenes de datos. Por ejemplo, usaban disparadores bajo la inserción, borrado o actualización de una relación *empleado* que contiene los atributos *sueldo* y *dept* para mantener el sueldo total de cada departamento. Sin embargo, muchos sistemas de bases de datos actuales soportan las vistas materializadas (véase el Apartado 3.5.1), que proporcionan una forma mucho más sencilla de mantener los datos de resumen. Los diseñadores también usaban ampliamente los disparadores para las réplicas de las bases de datos; usaban disparadores bajo la inserción, borrado o actualización de las relaciones para registrar los cambios en relaciones **cambio** o **delta**. Un proceso separado copiaba los cambios a la réplica (copia) de la base de datos, y el sistema ejecutaba los cambios sobre la réplica. Sin embargo, los sistemas de bases de datos modernos proporcionan características incorporadas para la réplica de bases de datos, haciendo innecesarios a los disparadores para la réplica en la mayoría de los casos.

De hecho, muchas aplicaciones de disparadores, incluyendo el ejemplo de descubiertos, se pueden sustituir por las características de «encapsulación» introducidas en SQL:1999. La encapsulación se puede usar para asegurar que las actualizaciones del atributo *saldo* de *cuenta* se hagan sólo mediante un procedimiento especial. Este procedimiento comprobaría un saldo negativo y realizaría las acciones de disparador de descubiertos. Las encapsulaciones pueden reemplazar el disparador de nuevos pedidos de forma similar.

```
create trigger nuevopedido after update of cantidad on inventario
referencing old row as ofila, new row as nfila
for each row
when nfila.nivel <= (select nivel
                     from nivelmínimo
                     where nivelmínimo.producto = ofila.producto)
and ofila.nivel <= (select nivel
                    from nivelmínimo
                    where nivelmínimo.producto = ofila.producto)
begin
    insert into pedidos
        (select producto, cantidad
         from nuevopedido
         where nuevopedido.producto= ofila.producto);
end
```

FIGURA 6.4. Ejemplo de un disparador para hacer un nuevo pedido de un producto.

```
define trigger descubierto on cuenta
for update
as
if nfila.saldo < 0
begin
      insert into prestatario
            (select nombre-cliente, número-cuenta
             from impositor, inserted
             where inserted.número-cuenta = impositor.número-cuenta)
      insert into préstamo values
            (inserted.número-cuenta, inserted.nombre-sucursal, – inserted.saldo)
      update cuenta set saldo = 0
            from cuenta, inserted
            where cuenta.número-cuenta = inserted.número-cuenta))
end
```

FIGURA 6.5. Ejemplo de disparador en la sintaxis de SQLServer de Microsoft.

Los disparadores se deberían escribir con sumo cuidado, dado que un error de un disparador detectado en tiempo de ejecución causa el fallo de la instrucción de inserción, borrado o actualización que inició el disparador. En el peor de los casos esto podría dar lugar a una cadena infinita de disparos. Por ejemplo, supóngase que un disparador de inserción sobre una relación realice otra (nueva) inserción sobre la misma relación. La acción de inserción dispara otra acción de inserción, y así hasta el infinito. Generalmente, los sistemas de bases de datos limitan la longitud de las cadenas de disparadores (por ejemplo, hasta 16 o 32), y consideran que las cadenas mayores son erróneas.

Los disparadores (desencadenadores en terminología de Microsoft) se denominan a veces *reglas* o *reglas activas*, pero no se deben confundir con las reglas Datalog (véase el Apartado 5.2), que son realmente definiciones de vistas.

La palabra clave **new** utilizada delante de *T.saldo* indica que debe utilizarse el valor de *T.saldo* posterior a la actualización; si se omite, se utilizará el valor previo a la actualización.

6.5. SEGURIDAD Y AUTORIZACIÓN

Los datos guardados en la base de datos deben estar protegidos contra los accesos no autorizados, de la destrucción o alteración malintencionadas además de la introducción accidental de inconsistencias que evitan las restricciones de integridad. En este apartado se examina el modo en que se pueden utilizar mal los datos o hacerlos inconsistentes de manera intencionada. Se presentan luego mecanismos para protegerse de dichas incidencias.

6.5.1. Violaciones de la seguridad

Entre las formas de acceso malintencionado se encuentran:

- La lectura no autorizada de los datos (robo de información)
- La modificación no autorizada de los datos
- La destrucción no autorizada de los datos

La **seguridad de las bases de datos** se refiere a la protección frente a accesos malintencionados. No es posible la protección absoluta de la base de datos contra el uso malintencionado, pero se puede elevar lo suficiente el coste para quien lo comete como para disuadir la mayor parte, si no la totalidad, de los intentos de tener acceso a la base de datos sin la autorización adecuada.

Para proteger la base de datos hay que adoptar medidas de seguridad en varios niveles:

- **Sistema de bases de datos.** Puede que algunos usuarios del sistema de bases de datos sólo estén autorizados a tener acceso a una parte limitada de la base de datos. Puede que otros usuarios estén autorizados a formular consultas pero tengan prohibido modificar los datos. Es responsabilidad del sistema de bases de datos asegurarse de que no se violen estas restricciones de autorización.
- **Sistema operativo.** Independientemente de lo seguro que pueda ser el sistema de bases de datos, la debilidad de la seguridad del sistema operativo puede servir como medio para el acceso no autorizado a la base de datos.
- **Red.** Dado que casi todos los sistemas de bases de datos permiten el acceso remoto mediante terminales o redes, la seguridad en el nivel del software de la red es tan importante como la seguridad física, tanto en Internet como en las redes privadas de las empresas.

- **Físico.** Los sitios que contienen los sistemas informáticos deben estar protegidos físicamente contra la entrada de intrusos.
- **Humano.** Los usuarios deben ser autorizados cuidadosamente para reducir la posibilidad de que alguno de ellos dé acceso a intrusos a cambio de sobornos u otros favores.

Debe conservarse la seguridad en todos estos niveles si hay que asegurar la seguridad de la base de datos. La debilidad de los niveles bajos de seguridad (físico o humano) permite burlar las medidas de seguridad estrictas de niveles superiores (base de datos).

En el resto de este apartado se aborda la seguridad en el nivel del sistema de bases de datos. La seguridad en los niveles físico y humano, aunque importante, cae fuera del alcance de este texto.

La seguridad dentro del sistema operativo se aplica en varios niveles, que van desde las contraseñas para el acceso al sistema hasta el aislamiento de los procesos concurrentes que se ejecutan en el sistema. El sistema de archivos también proporciona algún nivel de protección. Las notas bibliográficas hacen referencia al tratamiento de estos temas en los textos sobre sistemas operativos. Finalmente, la seguridad en el nivel de la red ha logrado un amplio reconocimiento a medida que Internet ha pasado de ser una plataforma para la investigación académica a convertirse en la base del comercio electrónico internacional. Las notas bibliográficas muestran el tratamiento dado por los libros de texto a los principios básicos de la seguridad de las redes. Se presenta la discusión sobre la seguridad en términos del modelo relacional de datos, aunque los conceptos de este capítulo son igualmente aplicables a todos los modelos de datos.

6.5.2. Autorizaciones

Los usuarios pueden tener varios tipos de autorización para diferentes partes de la base de datos. Entre ellas están las siguientes:

- **La autorización de lectura** permite la lectura de los datos, pero no su modificación.
- **La autorización de inserción** permite la inserción de datos nuevos, pero no la modificación de los existentes.
- **La autorización de actualización** permite la modificación de los datos, pero no su borrado.
- **La autorización de borrado** permite el borrado de los datos.

Los usuarios pueden recibir todos los tipos de autorización, ninguno de ellos o una combinación determinada de los mismos.

Además de estas formas de autorización para el acceso a los datos, los usuarios pueden recibir autorización para modificar el esquema de la base de datos:

- **La autorización de índices** permite la creación y borrado de índices.
- **La autorización de recursos** permite la creación de relaciones nuevas.
- **La autorización de alteración** permite el añadido o el borrado de atributos de las relaciones.
- **La autorización de eliminación** permite el borrado de relaciones.

Las autorizaciones de **eliminación** y de **borrado** se diferencian en que la autorización de borrado sólo permite el borrado de tuplas. Si un usuario borra todas las tuplas de una relación, la relación sigue existiendo, pero está vacía. Si se elimina una relación, deja de existir.

La capacidad de crear nuevas relaciones queda regulada mediante la autorización de **recursos**. El usuario con la autorización de **recursos** que crea una relación nueva recibe automáticamente todos los privilegios sobre la misma.

La autorización de **índices** puede parecer innecesaria, dado que la creación o borrado de un índice no afecta a los datos de las relaciones. Más bien, los índices son una estructura para las mejoras de rendimiento. Sin embargo, los índices también ocupan espacio y se exige que todas las modificaciones de las bases de datos actualicen los índices. Si se concediera a todos los usuarios la autorización de **índices**, los que llevaran a cabo actualizaciones estarían tentados de borrar los índices, mientras que los que formularan consultas estarían tentados de crear numerosos índices. Para permitir al *administrador de la base de datos* que regule el uso de los recursos del sistema es necesario tratar la creación de índices como un privilegio.

La forma superior de autoridad es la concedida al administrador de la base de datos. El administrador de la base de datos puede autorizar usuarios nuevos, reestructurar la base de datos, etcétera. Esta forma de autorización es análoga a la proporcionada al **superusuario** u operador del sistema operativo.

6.5.3. Autorizaciones y vistas

En el Capítulo 3 se introdujo el concepto de las *vistas* como medio de proporcionar a un usuario un modelo personalizado de la base de datos. Una vista puede ocultar los datos que un usuario no necesita ver. La capacidad de las vistas para ocultar datos sirve para simplificar el uso del sistema y para mejorar la seguridad. El uso del sistema se simplifica porque se permite al usuario restringir su atención a los datos de interés. Aunque puede que se niegue el acceso directo a una relación, puede que se le permita el acceso a parte de esa relación mediante una vista. Por tanto, se puede utilizar una combinación de seguridad en el nivel relacional y en el nivel de las vistas para limitar el acceso de un usuario precisamente a los datos que necesita.

En el ejemplo bancario considérese un empleado que necesita saber los nombres de todos los clientes que tienen un préstamo en cada sucursal. Este empleado no está autorizado a ver la información concerniente a los préstamos concretos que pueda tener cada cliente. Por tanto, se le debe negar el acceso directo a la relación *préstamo*. Pero si va a tener acceso a la información necesaria se le debe conceder acceso a la vista *cliente-préstamo*, que consiste sólo en los nombres de los clientes y las sucursales en los que tienen un préstamo. Esta vista se puede definir en SQL de la manera siguiente:

```
create view cliente-préstamo as
    (select nombre-sucursal, nombre-cliente
    from prestatario, préstamo
    where prestatario.número-préstamo
          = préstamo.número-préstamo)
```

Supóngase que el empleado formula la siguiente consulta SQL:

```
select *
from préstamo-cliente
```

Evidentemente, el empleado está autorizado a ver el resultado de esta consulta. Sin embargo, cuando el procesador de consultas traduce la consulta en una consulta sobre las relaciones reales de la base de datos, se obtiene una consulta sobre *prestatario* y *préstamo*. Por tanto, se debe comprobar la autorización de la consulta del empleado antes de que comience el procesamiento de la misma.

La creación de vistas no necesita la autorización de **recursos**. El usuario que crea una vista no recibe necesariamente todos los privilegios sobre la misma. Ese usuario sólo recibe los privilegios que no proporcionan autorizaciones adicionales respecto de las que ya posee. Por ejemplo, un usuario no puede recibir la autorización de **actualización** sobre una vista sin tener la autorización de **actualización** sobre las relaciones utilizadas para definir la vista. Si un usuario crea una vista sobre la que no se puede conceder ninguna autorización, se deniega la solicitud de creación de la vista. En el ejemplo de la vista *cliente-préstamo*, el creador de la vista debe tener autorización de **lectura** sobre las relaciones *prestatario* y *préstamo*.

6.5.4. Concesión de privilegios

El usuario al que se le ha concedido alguna forma de autorización puede ser autorizado a transmitir esa autorización a otros usuarios. Sin embargo, hay que tener cuidado con el modo en que se puede transmitir la autorización entre los usuarios para asegurar que la misma pueda retirarse en el futuro.

Considérese, a modo de ejemplo, la concesión de la autorización de actualización sobre la relación *préstamo* de la base de datos bancaria. Supóngase que, inicialmente, el administrador de la base de datos concede autorización de actualización sobre *préstamo* a los usuarios U_1, U_2 y U_3, que, a su vez, pueden transmitirla a otros usuarios. La transmisión de la autorización de un usuario a otro puede representarse mediante un **grafo de autorización**. Los nodos de este grafo son los usuarios. Se incluye un arco $U_i \rightarrow U_j$ en el grafo si el usuario U_i concede la autorización de actualización sobre *préstamo* a U_j. La raíz del grafo es el administrador de la base de datos. En la Figura 6.6 aparece un grafo sencillo. Obsérvese que tanto U_1 como U_2 conceden autorización al usuario U_5; sólo U_1 concede autorización a U_4.

Un usuario tiene autorización *si y sólo si* hay un camino desde la raíz del grafo de autorización (es decir, el nodo que representa al administrador de la base de datos) hasta el nodo que representa a ese usuario.

Un par de usuarios taimados podría intentar eludir las reglas de retirada de autorizaciones concediéndose autorización mutuamente, como se muestra en la Figura 6.7a. Si el administrador de la base de datos retira la autorización a U_2, U_2 la conserva mediante U_3, como se muestra en la Figura 6.7b. Si a continuación se retira la autorización a U_3, U_3 parece conservarla mediante U_2, como se muestra en la Figura 6.7.c. Sin embargo, cuando el administrador retira la autorización a U_3, los arcos de U_3 a U_2 y de U_2 a U_3 ya no forman parte de ningún camino que comience en el administrador de la base de datos. Hace falta que todos los arcos de un grafo de autorización sean parte de algún camino que comience en el administrador de la base de datos. Estos arcos se borran; el grafo de autorización resultante queda como se muestra en la Figura 6.8.

Se requiere que todos los arcos de un grafo de autorización sean parte de algún camino que se origine en el administrador de la base de datos. Los arcos entre U_2 y U_3 se borran, y el grafo de autorización es como el mostrado en la Figura 6.8.

6.5.5. El concepto de papel

Considérese un banco que tiene muchos cajeros. Cada cajero debe tener los mismos tipos de autorizaciones

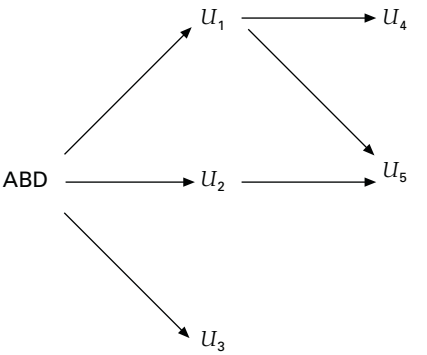

FIGURA 6.6. Grafo de concesión de autorizaciones.

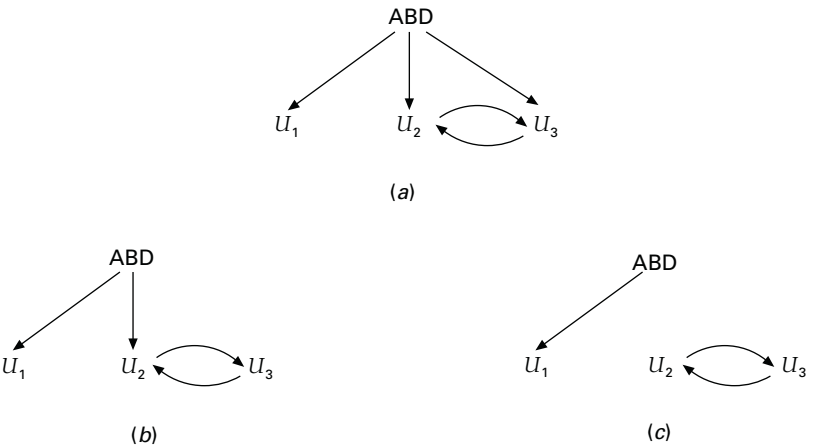

FIGURA 6.7. Intento de eludir la retirada de autorizaciones.

para el mismo conjunto de relaciones. Cada vez que se contrata un nuevo cajero hay que darle todas esas autorizaciones individualmente.

Un esquema mejor sería especificar las autorizaciones que deben tener los cajeros e identificar separadamente cuáles de los usuarios de la base de datos son cajeros. El sistema puede usar estas dos piezas de información para determinar las autorizaciones de cada persona que sea un cajero. Cuando se contrate a una nueva persona como cajero, se le debe asignar un identificador de usuario e identificarle como cajero. Ya no es necesario especificar permisos individuales para los cajeros.

Los **papeles** capturan este esquema. En la base de datos se crea un conjunto de papeles. Las autorizaciones se conceden a los papeles, de igual modo que se conceden a usuarios individuales. Se concede un conjunto de papeles a cada usuario de la base de datos (que puede ser vacío) para los que está autorizado.

En el ejemplo bancario, algunos ejemplos de papeles serían *cajero, gestor-sucursal, auditor* y *administrador-del-sistema*.

Una alternativa menos preferible sería crear un identificador de usuario *cajero* y permitir que cada cajero se conectase a la base de datos usando este identificador. El problema con este esquema es que no sería posible identificar exactamente al cajero que ha realizado una determinada transacción, conduciendo a problemas de seguridad. El uso de los papeles tiene la ventaja de requerir a los usuarios que se conecten con su propio identificador de usuario.

Cualquier autorización que se pueda conceder a un usuario se puede conceder a un papel. Los papeles se conceden a los usuarios al igual que las autorizaciones. Y, como otras autorizaciones, un usuario también puede conceder autorización para conceder un papel particular a otros. Así, los gestores de las sucursales pueden tener autorización de concesión para conceder el papel *cajero*.

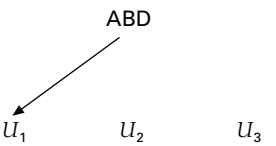

FIGURA 6.8. Grafo de autorización.

6.5.6. Trazas de auditoría

Muchas aplicaciones de bases de datos seguras requieren que se mantenga una **traza de auditoría**. Una traza de auditoría es un registro histórico de todos los cambios (inserciones, borrados o actualizaciones) de la base de datos, junto con información sobre el usuario que realizó el cambio y en qué momento.

La traza de auditoría ayuda a la seguridad de formas diferentes. Por ejemplo, si el saldo de una cuenta es incorrecto, el banco desearía revisar todas las actualizaciones realizadas sobre la cuenta para encontrar las actualizaciones incorrectas (o fraudulentas), así como las personas que realizaron los cambios. El banco podría entonces usar la traza de auditoría para revisar todas las actualizaciones realizadas por estas personas para encontrar las actualizaciones incorrectas o fraudulentas.

Es posible crear una traza de auditoría definiendo disparadores adecuados sobre las actualizaciones de las relaciones (usando variables definidas del sistema que identifican el nombre de usuario y la hora). Sin embargo, muchos sistemas de bases de datos proporcionan mecanismos incorporados para crear trazas de auditoría que son más convenientes de usar. Los detalles de la creación de las trazas de auditoría varían entre los sistemas de bases de datos y se deben consultar los manuales para los detalles.

6.6. AUTORIZACIÓN EN SQL

El lenguaje SQL ofrece un mecanismo bastante potente para la definición de autorizaciones. En este apartado se describen estos mecanismos y sus limitaciones.

6.6.1. Privilegios en SQL

La norma SQL incluye los privilegios **delete**, **insert**, **select** y **update** El privilegio **select** se corresponde con el privilegio de **lectura**. SQL también incluye un privilegio **references** que permite a un usuario o papel declarar claves externas al crear relaciones. Si la relación que se va a crear incluye una clave externa que hace referencia a atributos de otra relación, el usuario o papel debe haber recibido el privilegio **references** sobre esos atributos. El motivo de que el privilegio **references** sea una característica útil es algo sutil y se explica más adelante en este mismo apartado.

El lenguaje de definición de datos de SQL incluye órdenes para conceder y retirar privilegios. La instrucción **grant** se utiliza para conferir autorizaciones. La forma básica de esta instrucción es la siguiente:

grant <lista de privilegios> **on** <nombre de relación o de lista> **to** <lista de usuarios/papeles>

La *lista de privilegios* permite la concesión de varios privilegios con una sola orden.

La siguiente instrucción **grant** concede a los usuarios U_1, U_2 y U_3 la autorización **select** sobre la relación *sucursal*:

grant select on *sucursal* **to** U_1, U_2, U_3

La autorización **update** puede concederse sobre todos los atributos de la relación o sólo sobre algunos. Si se incluye la autorización **update** en una instrucción **grant**, la lista de atributos sobre los que se va a conceder la autorización **update** opcionalmente aparece entre paréntesis justo después de la palabra clave **update**. Si se omite la lista de atributos se concede el privilegio **update** sobre todos los atributos de la relación.

La siguiente instrucción **grant** concede a los usuarios U_1, U_2 y U_3 la autorización **update** sobre el atributo *importe* de la relación *préstamo*:

grant update (*importe*) **on** *préstamo* **to** U_1, U_2, U_3

En SQL el privilegio **insert** también puede especificar una lista de atributos; cualquier inserción en la relación debe especificar sólo esos atributos y al resto de los atributos se les conceden valores predeterminados (si hay alguno definido para ellos) o se les da el valor nulo.

El privilegio SQL **references** se concede sobre atributos concretos de manera parecida a la mostrada para el privilegio **update**. La siguiente instrucción **grant** permite al usuario U_1 crear relaciones que hagan referencia a la clave *nombre-sucursal* de la relación *sucursal* como si fuera una clave externa:

grant references (*nombre-sucursal*) **on** *sucursal* **to** U_1

En un principio puede parecer que no hay ningún motivo para evitar que los usuarios creen claves externas que hagan referencia a otra relación. Sin embargo, hay que recordar del Apartado 6.2 que las restricciones de las claves externas limitan las operaciones de borrado y de actualización sobre la operación a la que hacen referencia. En el ejemplo anterior, si U_1 crea una clave externa en una relación *r* que haga referencia al atributo *nombre-sucursal* de la relación *sucursal* y luego inserta en *r* una tupla correspondiente a la sucursal de Navacerrada, ya no es posible borrar la sucursal de Navacerrada de la relación *sucursal* sin modificar también la relación *r*. Por tanto, la definición de una clave externa por U_1 limita la actividad futura de los demás usuarios; por tanto, el privilegio **references** es necesario.

El privilegio **all privileges** puede utilizarse como una forma abreviada de todos los privilegios que se pueden conceder. De manera parecida, el nombre de usuario **public** hace referencia a todos los usuarios presentes y futuros del sistema. SQL incluye también un privilegio **usage** que autoriza a un usuario a utilizar un dominio especificado (recuérdese que el dominio se corresponde con el concepto de tipo de un lenguaje de programación y puede ser definido por el usuario).

6.6.2. Papeles

Los papeles se pueden crear en SQL:1999 como sigue

create role *cajero*

Se pueden conceder privilegios a los papeles al igual que a los usuarios, como se ilustra en la siguiente instrucción

grant select on *cuenta*
to *cajero*

Los papeles se pueden asignar a los usuarios, así como a otros papeles, como muestran estas instrucciones.

grant *cajero* **to** juan
create role *gestor*
grant *cajero* **to** *gestor*
grant *gestor* **to** maría

Los privilegios de un usuario o papel consisten en:

- Todos los privilegios concedidos directamente al usuario o papel.
- Todos los privilegios concedidos a papeles que se hayan concedido al usuario o papel.

Nótese que puede haber una cadena de papeles; por ejemplo, el papel *empleado* se puede conceder a todos los *cajeros*. A su vez, el papel *cajero* se concede a todos los *gestores*. Así, el papel *gestor* hereda todos los privilegios concedidos a los papeles *empleados* y *cajeros* además de los privilegios concedidos directamente a *gestor*.

6.6.3. El privilegio de conceder privilegios

Un usuario o papel al que se le concede un privilegio no está autorizado de manera predeterminada a concedérselo a otros usuarios o papeles. Si se desea conceder un privilegio a un usuario y permitirle que lo transmita a otros usuarios hay que añadir la cláusula **with grant option** a la orden **grant** correspondiente. Por ejemplo, si se desea conceder a U_1 el privilegio **select** sobre *sucursal* y que pueda transmitirlo a otros, hay que escribir

grant select on *sucursal* **to** U_1 **with grant option**

Para retirar una autorización se utiliza la instrucción **revoke**. Adopta una forma casi idéntica a la de **grant**:

revoke <lista de privilegios> **on** <nombre de relación o de vista>
from <lista de usuarios o papeles> [**restrict** | **cascade**]

Por tanto, para retirar los privilegios que se han concedido con anterioridad hay que escribir

revoke select on *sucursal* **from** U_1, U_2, U_3
revoke update (*importe*) **on** *préstamo* **from** U_1, U_2, U_3
revoke references (*nombre-sucursal*) **on** *sucursal* **from** U_1

Como se vio en el Apartado 6.5.4, la retirada de un privilegio a un usuario o papel puede hacer que otros usuarios o papeles también lo pierdan. Este comportamiento se denomina *retirada en cadena*. En la mayoría de los sistemas de bases de datos, la retirada en cascada es el comportamiento predeterminado; por tanto, la palabra clave **cascade** se puede omitir como se ha hecho en los ejemplos anteriores. La instrucción **revoke** también puede especificar **restrict**:

revoke select on *sucursal* **from** U_1, U_2, U_3 **restrict**

En este caso se devuelve un error si hay alguna retirada de privilegios en cadena y en este caso la acción de retirada de privilegios no se lleva a cabo. La siguiente instrucción **revoke** sólo anula la opción de concesión **grant** y no el propio privilegio **select**.

revoke grant option for select on *sucursal* **from** U_1

6.6.4. Otras características

El creador de un objeto (relación, vista o papel) obtiene todos los privilegios sobre el objeto, incluyendo el privilegio de conceder privilegios a otros.

La norma SQL especifica un mecanismo primitivo de autorización para el esquema de la base de datos: sólo el propietario del esquema puede ejecutar cualquier modificación del esquema. Por tanto, las modificaciones del esquema —como la creación o borrado de las relaciones, la adición o eliminación de atributos de las relaciones y la adición o eliminación de índices— sólo pueden ser ejecutadas por el propietario del mismo. Varias implementaciones de las bases de datos tienen mecanismos de autorización más potentes para los esquemas de las bases de datos, parecidos a los discutidos anteriormente, pero esos mecanismos no son estándar.

6.6.5. Limitaciones de la autorización SQL

Las normas SQL actuales de autorización tienen algunas deficiencias. Por ejemplo, supóngase que se desea que todos los estudiantes sean capaces de ver sus propias notas, pero no las del resto. La autorización debe estar en el nivel de las tuplas, lo cual no es posible en los estándares de autorización de SQL.

Más aún, con el crecimiento de Web, los accesos a bases de datos vienen principalmente de los servidores de aplicaciones Web. Los usuarios finales puede que no tengan identificadores de usuario individuales en la base de datos y realmente puede que haya sólo un único identificador de usuario en la base de datos que corresponda a todos los usuarios del servidor de aplicaciones.

La tarea de la autorización recae entonces sobre el servidor de aplicaciones; el esquema completo de autorización de SQL se omite. El beneficio es que la aplicación puede implementar las autorizaciones de grano fino, como las de las tuplas individuales. Estos son los problemas:

- El código para comprobar la autorización se entremezcla con el resto del código de la aplicación.
- La implementación de la autorización mediante código de aplicación, en lugar de realizarlo declarativamente con SQL, hace que sea difícil asegurar la ausencia de agujeros de seguridad. Debido a un descuido, uno de los programas de aplicación podría no comprobar la autorización, permitiendo el acceso de usuarios no autorizados a datos confidenciales. La verificación de que todos los programas de aplicación realizan todas las comprobaciones de autorización implica la lectura de todo el código del servidor de la aplicación, una tarea formidable en un gran sistema.

6.7. CIFRADO Y AUTENTICACIÓN

Puede que las diferentes provisiones para la autorización que haga un sistema de bases de datos no proporcionen suficiente protección para los datos extremadamente delicados. En tales casos se pueden **cifrar** los datos. Los datos cifrados no se pueden leer a menos que el lector sepa la manera de **descifrarlos** (*descodificarlos*). El cifrado también forma la base de los buenos esquemas para la autenticación de usuarios en una base de datos.

6.7.1. Técnicas de cifrado

Hay 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 puede ser sencillo para un usuario no autorizado romper el código. Como ejemplo de una técnica de cifrado débil considérese la sustitución de cada carácter por el siguiente en el alfabeto. Así,

Navacerrada

se transforma en

Ñbwbdfssbeb

Si un usuario no autorizado sólo lee «Ñbwbdfssbeb» probablemente no tenga la información suficiente para romper el código. Sin embargo, si el intruso ve un gran número de nombres de sucursales codificados puede utilizar los datos estadísticos referentes a la frecuencia relativa de los caracteres para averiguar el tipo de sustitución realizado (por ejemplo, en inglés la *E* es la más frecuente, seguida por *T, A, O, N, I, …*).

Una buena técnica de cifrado tiene las propiedades siguientes:

- Es relativamente sencillo para los usuarios autorizados cifrar y descifrar los datos.
- El esquema de cifrado no depende de lo poco conocido que sea el algoritmo, sino más bien de un parámetro del algoritmo denominado *clave de cifrado*.
- Es extremadamente difícil para un intruso determinar la clave de cifrado.

Un enfoque, la *norma de cifrado de datos* (*Data Encription Standard*, DES) realiza una sustitución de caracteres y una reordenación de los mismos en función de una clave de cifrado. Para que este esquema funcione los usuarios autorizados deben proveerse de la clave de cifrado mediante un mecanismo seguro. Este requisito es una debilidad importante, dado que el esquema no es más seguro que el mecanismo por el que se transmite la clave de cifrado. La norma DES se reafirmó en 1983, 1987 y de nuevo en 1993. Sin embargo, la debilidad de DES se reconoció en 1993 cuando se necesitó una nueva norma denominada *norma de cifrado avanzado* (*Advanced Encription Standard*, AES). En el año 2000, el **algoritmo Rijndael** (denominado así por sus inventores, V. Rijmen y J. Daemen) se seleccionó para ser la norma AES. Este algoritmo se eligió por su nivel significativamente más fuerte de seguridad y su facilidad relativa de implementación en los sistemas informáticos actuales, así como en dispositivos como tarjetas inteligentes. Al igual que la norma DES, el algoritmo Rijndael es un algoritmo de clave compartida (o clave simétrica) en el que los usuarios autorizados comparten una clave.

El **cifrado de clave pública** es un esquema alternativo que evita parte de los problemas que se afrontan con DES. Se basa en dos claves; una *clave pública* y una *clave privada*. Cada usuario U_i tiene una clave pública C_i y una clave privada D_i. Todas las claves públicas están publicadas: cualquiera puede verlas. Cada clave privada sólo la conoce el usuario al que pertenece. Si el usuario U_1 desea guardar datos cifrados, los cifra utilizando la clave pública C_1. Descifrarlos requiere la clave privada D_1.

Como la clave de cifrado de cada usuario es pública es posible intercambiar información de manera segura utilizando este esquema. Si el usuario U_1 desea compartir los datos con U_2 los codifica utilizando E_2, la clave pública de U_2. Dado que sólo 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 pueda hacerse público sin permitir a la gente descubrir el esquema de descifrado. En otros términos, debe ser difícil deducir la clave privada dada la clave pública. Existe un esquema de ese tipo y se basa en lo siguiente:

- Hay un algoritmo eficiente para comprobar si un número es primo.
- No se conoce ningún algoritmo eficiente para encontrar los factores primos de un número.

Para los fines de este esquema los datos se tratan como una colección de enteros. Se crea una 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) y el algoritmo de descifrado no se puede utilizar con éxito si sólo se conoce el producto P_1P_2. Dado que todo lo que se publica es el producto P_1P_2, un usuario no autorizado debería poder factorizar P_1P_2 para robar datos. Eligiendo P_1 y P_2 suficientemente grandes (por encima de 100 dígitos) se puede hacer el coste de la factorización prohibitivamente elevado (del orden de años

de tiempo de cálculo, incluso en las computadoras más rápidas).

En las notas bibliográficas se hace referencia a los detalles del cifrado de clave pública y la justificación matemática de las propiedades de esta técnica.

Pese a que el cifrado de clave pública que utiliza el esquema descrito es seguro, también es costoso en cuanto a cálculo. Un esquema híbrido utilizado para la comunicación segura es el siguiente: las claves DES se intercambian mediante un esquema de cifrado de clave pública y posteriormente se utiliza el cifrado DES con los datos transmitidos.

6.7.2. Autenticación

La autenticación se refiere a la tarea de verificar la identidad de una persona o software que se conecte a una base de datos. La forma más simple consiste en una contraseña secreta que se debe presentar cuando se abra una conexión a la base de datos.

La autenticación basada en palabras clave se usa ampliamente por los sistemas operativos y bases de datos. Sin embargo, el uso de contraseñas tiene algunos inconvenientes, especialmente en una red. Si un husmeador es capaz de «oler» los datos que circulan por la red, puede ser capaz de encontrar la contraseña que se está enviando por la red. Una vez que el husmeador tiene un usuario y contraseña, se puede conectar a la base de datos pretendiendo que es el usuario legítimo.

Un esquema más seguro es el sistema de **desafío-respuesta**. 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 mima contraseña secreta, y comparando el resultado con la cadena de desafío original. Este esquema asegura que las contraseñas no circulen por la red.

Los sistemas de clave pública se pueden usar para cifrar en un sistema de desafío-respuesta. El sistema de bases de datos cifra una cadena de desafío usando la clave pública del usuario y lo envía al usuario. Éste 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 en la base de datos, donde podría ser vista potencialmente por administradores del sistema.

Otra aplicación interesante de la criptografía está en las **firmas digitales** para verificar la autenticidad de los datos; las firmas digitales desempeñan el papel electrónico de las firmas físicas en los documentos. La clave privada se usa para firmar los datos y los datos firmados se pueden hacer públicos. Cualquiera podría verificarlos con la clave pública, pero nadie podría haber generado los datos codificados sin tener la clave privada. Por tanto, se puede comprobar que los datos fueron creados realmente por la persona que afirma haberlos creado.

Además, las firmas digitales también sirven para asegurar el **rechazo**. Es decir, en el caso de que una persona que creó los datos afirmase más tarde que no lo hizo (el equivalente electrónico de afirmar que no se ha firmado un talón) se puede probar que esa persona ha creado los datos (a menos que haya cedido su clave privada a otros).

6.8. RESUMEN

- Las restricciones de integridad aseguran que las modificaciones realizadas a la base de datos por los usuarios autorizados no den lugar a la pérdida de la consistencia de los datos.
- En capítulos anteriores se tomaron en consideración varios tipos de restricciones, incluyendo la declaración de claves y la declaración de la forma de las relaciones (de varios a varios, de varios a uno, de uno a uno). En este capítulo se han tomado en consideración otros tipos de restricciones y se han discutido mecanismos para asegurar el mantenimiento de estas restricciones.
- Las restricciones de los dominios especifican el conjunto de valores que se pueden asociar con un atributo. Estas restricciones también pueden impedir el uso de valores nulos para atributos concretos.
- Las restricciones de integridad referencial aseguran que un valor que aparece en una relación para un conjunto de atributos dado aparezca también para un conjunto de atributos concreto en otra relación.
- Las restricciones de dominio y las de integridad referencial son relativamente sencillas de verificar. El uso de restricciones más complejas puede provocar una sobrecarga considerable. Se han visto dos maneras de expresar restricciones más generales. Los asertos son expresiones declarativas que manifiestan predicados que deben ser siempre verdaderos.
- Los disparadores definen las acciones que se deben ejecutar automáticamente cuando se producen ciertos eventos. Los disparadores tienen muchos usos, tales como la implementación de reglas de negocio, registro histórico de la auditoría e incluso para realizar acciones fuera del sistema de bases de datos. Aunque los disparadores se añadieron tarde a la norma SQL como parte de SQL:1999, la mayoría de sistemas de bases de datos los han implementado desde hace tiempo.

- Los datos guardados en las bases de datos deben protegerse de los accesos no autorizados, de la destrucción malintencionada o de la alteración y de la introducción accidental de inconsistencias.
- Es más sencilla la protección contra la pérdida accidental de la consistencia de los datos que contra el acceso malintencionado a las bases de datos. La protección total de las bases de datos contra las acciones malintencionadas no es posible, pero puede lograrse que el coste para quienes las cometan sea lo bastante elevado como para disuadir la mayor parte, si no a la totalidad, de los intentos de acceso a la base de datos sin la correspondiente autorización.
- Los usuarios pueden tener varios tipos de autorización para diferentes partes de la base de datos. La autorización es un medio por el que se puede proteger al sistema de bases de datos contra los accesos malintencionados o no autorizados.
- Los usuarios a los que se les haya concedido algún tipo de autorización pueden ser autorizados a transmitir la misma a otros usuarios. Sin embargo, hay que tomar precauciones respecto a la manera en que se pueden transmitir las autorizaciones entre los usuarios si se debe asegurar que las mismas se podrán retirar en el futuro.
- Los papeles facilitan la asignación de un conjunto de privilegios a un usuario de acuerdo al papel que el usuario desempeña en la organización.
- Puede que las diferentes provisiones de autorización de los sistemas de bases de datos no proporcionen la protección suficiente para los datos más delicados. En tales casos se pueden *cifrar* los datos. Sólo quienes conozcan la manera de *descifrar* (descodificar) los datos cifrados podrán leerlos. El cifrado también forma la base de la autenticación segura de los usuarios.

TÉRMINOS DE REPASO

- Aserto
- Autenticación
- Autorización
- Cascada
- Cifrado
- Cifrado de clave pública
- Cifrado de clave secreta
- Cláusula check
- Concesión de privilegios
- Disparador
- Disparadores before y after
- Firma digital
- Grafo de autorización
- Integridad referencial
- Modelo evento-condición-acción
- Niveles de seguridad
- Papeles
- Privilegios
  - Lectura
  - Inserción
  - Actualización
  - Borrado
  - Índices
  - Recursos
  - Alteración
  - Eliminación
  - Concesión
  - Todos los privilegios
- Rechazo
- Restricción de clave externa
- Restricción de clave primaria
- Restricciones de los dominios
- Restricción de unicidad
- Seguridad de la base de datos
- Sistema de desafío-respuesta
- Variables y tablas de transición

EJERCICIOS

6.1. Complétese la definición del LDD de SQL de la base de datos bancaria de la Figura 6.2 para incluir las relaciones *préstamo* y *prestatario*.

6.2. Considérese la siguiente base de datos relacional:

empleado (*nombre-empleado*, *calle*, *ciudad*)
trabaja (*nombre-empleado*, *nombre-empresa*, *sueldo*)
empresa (*nombre-empresa*, *ciudad*)
jefe (*nombre-empleado*, *nombre-jefe*)

Dese una definición en el LDD de SQL de esta base de datos. Identifíquense las restricciones de integridad

referencial que deban cumplirse e inclúyanse en la definición del LDD.

6.3. Las restricciones de integridad referencial tal y como se han definido en este capítulo implican exactamente a dos relaciones. Considérese una base de datos que incluye las relaciones siguientes:

trabajador-fijo (*nombre*, *despacho*, *teléfono*, *sueldo*)
trabajador-tiempo-parcial (*nombre*, *sueldo-por-hora*)
dirección (*nombre*, *calle*, *ciudad*)

Supóngase que se desea exigir que cada nombre que aparece en *dirección* aparezca en *trabajador-fijo* o en *trabajador-tiempo-parcial*, pero no necesariamente en ambos.

- **a.** Propóngase una sintaxis para expresar esta restricción.
- **b.** Discútanse las acciones que debe realizar el sistema para aplicar una restricción de este tipo.

6.4. SQL permite la dependencia de una clave externa para hacer referencia a la misma relación, como en el ejemplo siguiente:

```
create table jefe
    (nombre-empleado char (20) not null
     nombre-jefechar (20) not null,
     primary key nombre-empleado,
     foreign key (nombre-jefe) references jefe
                              on delete cascade)
```

Aquí, *nombre-empleado* es una clave de la tabla *director*, lo que significa que cada empleado tiene como máximo un director. La orden de clave externa exige que cada director sea también empleado. Explíquese exactamente lo que ocurre cuando se borra una tupla de la relación *jefe*.

6.5. Supóngase que hay dos relaciones r y s tales que la clave externa B de r hace referencia a la clave primaria A de s. Descríbase la manera en que puede utilizarse el mecanismo de los disparadores para implementar la opción **on delete cascade** cuando se borra una tupla de s.

6.6. Escríbase un aserto para la base de datos bancaria que asegure que el valor del activo de la sucursal de Navacerrada sea igual a la suma de todos los importes prestados por esa sucursal.

6.7. Escríbase un disparador SQL para realizar la siguiente acción: cuando se borre una cuenta, comprobar para cada tenedor de la cuenta si tiene otras cuentas y, si no, borrarlo de la relación impositor.

6.8. Considérese la vista sucursal-cliente definida como sigue:

```
create view sucursal-cliente as
    select nombre-sucursal, nombre-cliente
    from impositor, cuenta
    where impositor.número-cuenta
          = cuenta.número-cuenta
```

Supóngase que la vista está *materializada*, es decir, la vista se calcula y se almacena. Escríbanse reglas activas para *mantener* la vista, es decir, mantenerla actualizada según las inserciones y borrados en *impositor* o *cuenta*. No hay que preocuparse de las actualizaciones.

6.9. Confecciónese una lista de los problemas de seguridad de un banco. De cada elemento de la lista hay que decir si afecta a la seguridad física, a la personal, a la del sistema operativo o a la de las bases de datos.

6.10. Utilizando las relaciones de la base de datos bancaria de ejemplo escríbase una expresión SQL para definir las vistas siguientes:

- **a.** Una vista que contenga los números de cuenta y los nombres de los clientes (pero no los saldos) de todas las cuentas de la sucursal de El Escorial.
- **b.** Una vista que contenga el nombre y la dirección de todos los clientes que tengan cuenta en el banco pero no tengan ningún préstamo.
- **c.** Una vista que contenga el nombre y el saldo medio de la cuenta de cada cliente de la sucursal de Collado Villalba.

6.11. Para cada una de las vistas definidas en el Ejercicio 6.10 explíquese la manera en que deben realizarse las actualizaciones (si es que se deben permitir). *Sugerencia:* véase la discusión de las vistas en el Capítulo 3.

6.12. En el Capítulo 3 se describió el uso de las vistas para facilitar el acceso a la base de datos de los usuarios que sólo necesiten ver una parte de la misma. En este capítulo se ha descrito el uso de las vistas como mecanismo de seguridad. ¿Pueden entrar en conflicto estas dos finalidades de las vistas? Explíquese la respuesta.

6.13. ¿Cuál es la finalidad de tener categorías diferentes para la autorización de índices y para la de recursos?

6.14. Los sistemas de bases de datos que guardan cada relación en un archivo diferente del sistema operativo pueden utilizar los esquemas de seguridad y de autorización del sistema operativo en lugar de definir un esquema especial propio. Discútanse las ventajas e inconvenientes de ese enfoque.

6.15. ¿Cuáles son las dos ventajas de cifrar los datos guardados en una base de datos?

6.16. 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. Propóngase un esquema para guardar de manera segura las contraseñas. Asegúrese de que el esquema permita al sistema comprobar las contraseñas proporcionadas por los usuarios que intenten iniciar una sesión en el mismo.

NOTAS BIBLIOGRÁFICAS

En Hammer y McLeod [1975], Stonebraker [1975], Eswaran y Chamberlin [1975], Schmid y Swenson [1975] y en Codd [1979] se ofrecen discusiones sobre las restricciones de integridad en el modelo relacional. Las propuestas originales de SQL para los asertos y los disparadores se discuten en Astrahan et al. [1976], Chamberlin et al. [1976] y en Chamberlin et al. [1981]. Véanse las notas bibliográficas del Capítulo 4 para obtener referencias de las normas SQL.

Las discusiones referentes al mantenimiento eficiente y a la comprobación de los asertos de integridad semántica se ofrecen en Hammer y Sarin [1978], Badal y Popek [1979], Bernstein et al. [1980a], Hsu e Imielinski [1985], McCune y Henschen [1989] y Chomicki [1992a]. Una alternativa al uso de comprobaciones de integridad en tiempo de ejecución es certificar la corrección de los programas que tienen acceso a la base de datos. Este enfoque se discute en Sheard y Stemple [1989].

Las **bases de datos activas** son bases de datos que soportan disparadores y otros mecanismos que permiten a la base de datos realizar acciones cuando sucedan eventos. En McCarthy y Dayal [1989] se discute la arquitectura de un sistema de bases de datos activas basado en el formalismo evento-condición-acción. Widom y Finkelstein [1990] describen la arquitectura de un sistema de reglas basado en reglas orientadas a conjuntos; la implementación de un sistema de reglas en el sistema de bases de datos extensibles Starbust se presenta en Widom et al. [1991]. Considérese un mecanismo de ejecución que permita la elección no determinista de la regla a ejecutar a continuación. Se dice que un sistema de reglas es **confluente** si, independientemente de la regla elegida, el estado final es el mismo. Los aspectos de terminación, no determinismo y confluencia de los sistemas de reglas se discuten en Aiken et al. [1995].

Los aspectos de la seguridad de los sistemas informáticos en general se discuten en Bell y LaPadula [1976] y en el Departamento de Defensa de EE.UU. [U.S. Dept. of Defense 1985]. Los aspectos de la seguridad de SQL se pueden encontrar en las normas de SQL y en los libros de texto sobre SQL referenciados en las notas bibliográficas del Capítulo 4.

Stonebraker y Wong [1974] estudian el enfoque de la seguridad de Ingres, que implica la modificación de las consultas de los usuarios para asegurar que los usuarios no tienen acceso a datos para los que no se les ha concedido autorización. Denning y Denning [1979] presentan una visión del estado del arte de la seguridad de las bases de datos.

Los sistemas de bases de datos que pueden producir respuestas erróneas cuando es necesario para el mantenimiento de la seguridad se discuten en Winslett et al. [1994] y en Tendick y Matloff [1994]. El trabajo en la seguridad de las bases de datos relacionales incluye el de Stachour y Thuraisingham [1990], Jajodia y Sandhu [1990], Kogan y Jajodia [1990] y Qian y Lunt [1996]. Los aspectos de la seguridad de los sistemas operativos se discuten en la mayor parte de los textos sobre sistemas operativos, incluyendo a Silberschatz y Galvin [1998].

Stallings [1998] proporciona una descripción en un libro de texto de la criptografía. Daemen y Rijmen [2000] presentan el algoritmo Rjndael. El Departamento de Comercio de EE. UU. [U.S. Dept. of Commerce 1977] presentó la norma para cifrado de datos. El cifrado mediante clave pública se discute en Rivest et al. [1978]. Otras discusiones sobre criptografía incluyen las de Diffie y Hellman [1979], Simmons [1979], Fernández et al. [1981] y Akl [1983].