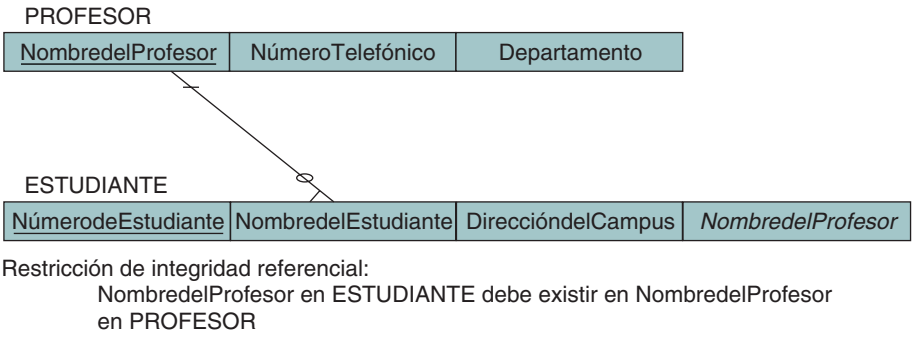
➤ FIGURA 6-9

Representación relacional de las entidades PROFESOR y ESTUDIANTE en la figura 6-8(a)



En la figura 6-9, el tenedor en el extremo de ESTUDIANTE en la línea de relación significa que puede haber muchos renglones de ESTUDIANTE por cada línea de PROFESOR. El que no aparezca un tenedor en el otro extremo significa que cada ESTUDIANTE puede ser asesorado, cuando mucho, por un PROFESOR. Al igual que con los diagramas E-R, las líneas intercaladas se usan para denotar las relaciones obligatorias y los óvalos indican las opcionales.

Observe que con NombredelProfesor almacenado como una llave externa en ESTUDIANTE, podemos procesar la relación en ambas direcciones. Con un NúmerodeEstudiante determinado, podemos buscar el renglón adecuado en ESTUDIANTE y obtener el nombre de su asesor en los datos del renglón. Para obtener los demás datos de PROFESOR usamos el nombre que obtuvimos en ESTUDIANTE con el fin de buscar el renglón adecuado en PROFESOR. Para determinar todos los estudiantes que asesora un miembro de la facultad en particular, buscamos todos los renglones de ESTUDIANTE que tengan el nombre del profesor como un valor para NombredelProfesor. Después los datos de los estudiantes se toman de esos renglones.

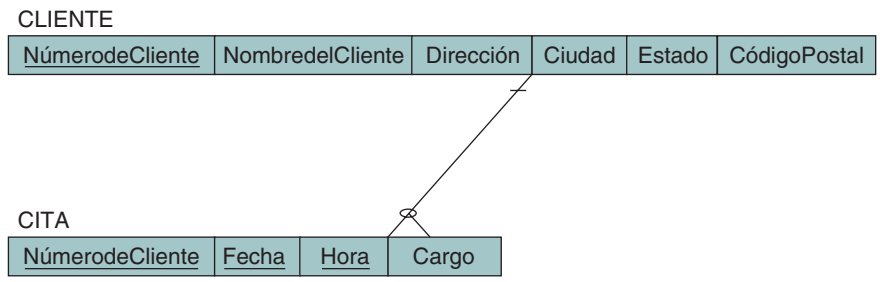
Compare esta situación con una representación de relaciones 1:1. En ambos casos almacenamos la llave de una relación como una llave externa en la segunda relación. Sin embargo, en una relación 1:1 no importa qué llave se mueva a la segunda relación. Pero sí tiene importancia en una relación 1:N. *La llave de la relación padre se debe colocar en la relación hijo*.

Para comprender mejor esto, observe qué pasaría si intentáramos colocar la llave del hijo en la relación padre (colocando NúmerodeEstudiante en PROFESOR). Debido a que los atributos en una relación sólo pueden tener un valor, en cada registro de PROFESOR sólo hay espacio para un estudiante. Por lo tanto, esta estructura no se puede usar para representar el lado "muchos" de la relación 1:N. De ahí que, para representar una relación 1:N debemos colocar la llave de la relación padre en la relación hijo.

La figura 6-10 muestra la representación de las entidades CLIENTE y CITA. Cada entidad se representa con una relación. CITA es una entidad débil dependiente de un

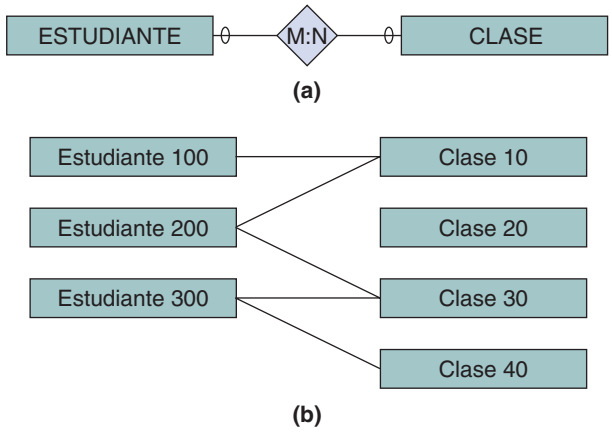
➤ FIGURA 6-10

Representación relacional de la entidad débil de la figura 6-8(c)



➤ FIGURA 6-11

Ejemplo de una relación M:N: (a) diagrama E-R de la relación ESTUDIANTE con CLASE, y (b) datos de muestra para la relación ESTUDIANTE con CLASE



identificador, por ende tiene una llave compuesta que consiste en la llave de la entidad de la cual depende cuando menos un atributo. Aquí la llave es (NúmerodeCliente, Fecha, Hora). Para representar la relación 1:N, normalmente agregaríamos la llave del padre al hijo. Sin embargo, en este caso la llave del padre (NúmerodeCliente) ya es parte del hijo, por lo que no es necesario agregarla.

REPRESENTACIÓN DE RELACIONES MUCHOS A MUCHOS. El tercero y último tipo de relación binaria es de muchos a muchos (M:N), en la cual una entidad de un tipo corresponde a muchas entidades del segundo tipo, y una entidad del segundo tipo corresponde a muchas entidades del primer tipo.

La figura 6-11(a) presenta un diagrama E-R de relación muchos a muchos entre estudiantes y clases. Una entidad ESTUDIANTE puede corresponder a muchas entidades CLASE, y la entidad CLASE puede corresponder a muchas entidades ESTUDIANTE. Observe que ambos participantes en la relación son opcionales: un estudiante no necesita estar inscrito en una clase, y la clase no necesita tener estudiantes. La figura 6-11(b) proporciona una muestra de datos.

Las relaciones muchos a muchos no pueden representarse directamente mediante relaciones en la misma forma que las relaciones uno a uno y uno a muchos. Para comprender por qué, intente usar la misma estrategia que aplicamos con las relaciones 1:1 y 1:N, colocando la llave de una relación como una llave externa en la otra relación. Primero, defina una relación para cada una de las entidades; nómbrelas ESTUDIANTE y CLASE. Ahora intente poner la llave de ESTUDIANTE (es decir, NúmerodeEstudiante) en CLASE. Debido a que no se permiten valores múltiples en las celdas de una relación, sólo tenemos espacio para un NúmerodeEstudiante, así no colocamos el registro de NúmerodeEstudiante del segundo estudiante y de los demás.

Tendremos el mismo problema si intentamos colocar la llave de CLASE (digamos NúmerodeClase) en ESTUDIANTE. Podemos almacenar con facilidad el identificador de la primera clase en la cual está inscrito un estudiante; pero no tenemos espacio para almacenar el identificador de las clases adicionales.

La figura 6-12 muestra otra estrategia (*aunque es incorrecta*). En este caso, hemos almacenado un renglón en la relación CLASE para cada ESTUDIANTE inscrito en una clase, así que hay dos registros para Clase 10 y dos para Clase 30. El problema con este esquema es que duplicamos los datos de la clase y, por lo tanto, creamos anomalías de modificación. Necesitamos cambiar muchos renglones si, por ejemplo, se modifica el programa de la Clase 10. También considere las anomalías de inserción y de eliminación: ¿cómo podemos programar una clase nueva hasta que se haya inscrito un estudiante? ¿Qué pasaría si el Estudiante 300 abandonara la Clase 40? Obviamente esta estrategia no funcionaría.

➤ FIGURA 6-12

Representación incorrecta de una relación M:N

EID	Otros datos de ESTUDIANTE
100	. . .
200	. . .
300	. . .

ESTUDIANTE

NúmerodeClase	HorariodeClase	Otros datos de clase	EID
10	10:00 MWF	. . .	100
10	10:00 MWF	. . .	200
30	3:00 TH	. . .	200
30	3:00 TH	. . .	300
40	8:00 MWF	. . .	300

CLASE

La solución para este problema es crear una tercera relación que represente la propia relación. La relación ESTUDIANTE-CLASE se define en la figura 6-13(a). En la figura 6-13(b) se muestra un ejemplo. Estas relaciones se llaman **relaciones de intersección** porque cada renglón documenta la intersección de un estudiante particular con determinada clase. Observe que en la figura 6-13(b) hay un renglón en la relación de intersección para cada línea entre ESTUDIANTE y CLASE de la figura 6-11(b).

Los diagramas de la estructura de datos para la relación ESTUDIANTE-CLASE aparecen en la figura 6-14. La relación de CLASE con ESTUDIANTE-CLASE es 1:N, y la relación de ESTUDIANTE con ESTUDIANTE-CLASE también es 1:N. En esencia, hemos dividido la relación M:N en dos relaciones 1:N. La llave de ESTUDIANTE-CLASE es la combinación de las llaves de sus padres (EID, NúmerodeClase). La llave de una relación de intersección siempre es la combinación de las llaves de sus padres. También, observe que se requieren las dos relaciones de sus padres. Ahora debe existir un padre para cada valor de la llave en la relación de intersección.

➤ FIGURA 6-13

Representación de una relación M:N: (a) relaciones necesarias para representar la relación ESTUDIANTE con CLASE, y (b) datos de ejemplo para la relación ESTUDIANTE con CLASE

ESTUDIANTE (NúmerodeEstudiante, NombredelEstudiante)

CLASE (NúmerodeClase, NombredelaClase)

ESTUDIANTE-CLASE (*NúmerodeEstudiante*, *NúmerodeClase*)

Restricción de integridad referencial:
NúmerodeClase en ESTUDIANTE-CLASE debe existir en NúmerodeClase en CLASE
NúmerodeEstudiante en ESTUDIANTE-CLASE debe existir en NúmerodeEstudiante en ESTUDIANTE

(a)

100	Jones, Mary
200	Parker, Fred
300	Wu, Jason

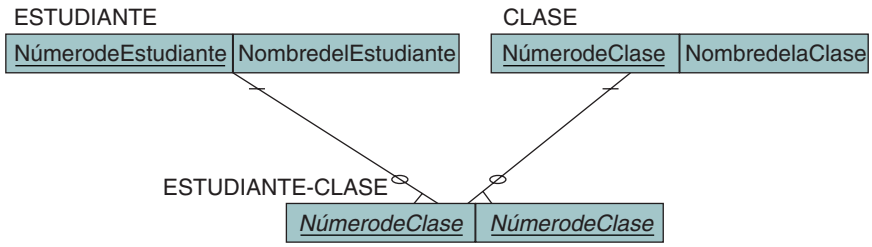
100	10
200	10
200	30
300	30
300	40

10	Contabilidad
20	Finanzas
30	Mercadotecnia
40	Base de Datos

(b)

➤ FIGURA 6-14

Diagrama de estructura de datos para la relación ESTUDIANTE con CLASE



Representación de relaciones recursivas

Una **relación recursiva** es aquella entre entidades de la misma clase. Las relaciones recursivas no son esencialmente diferentes de otras relaciones TIENE-UN, y se pueden representar usando las mismas técnicas. Al igual que con una relación recursiva TIENE-UN, hay tres tipos de relaciones recursivas: 1:1, 1:N y N:M; la figura 6-15 muestra un ejemplo de cada una.

Considere primero la relación PATROCINADOR en la figura 6-15(a). Al igual que con una relación 1:1, una persona puede patrocinar a otra, y cada una es patrocinada como máximo por otra. La figura 6-16(a) contiene datos muestra de esta relación.

Para representar una relación recursiva 1:1 adoptamos un enfoque casi idéntico al de las relaciones regulares 1:1: podemos colocar la llave de la persona patrocinada en el renglón del patrocinador, o podemos colocar la llave del patrocinador en el renglón de la persona que es patrocinada. La figura 6-16(b) muestra la primera alternativa y la figura 6-16(c), la segunda. Ambas funcionan, y así la elección depende de aspectos del desempeño.

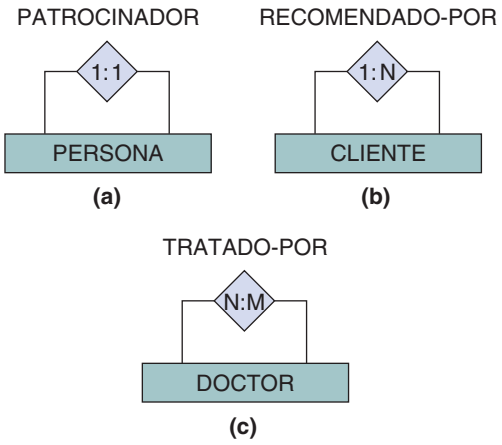
Esta técnica es idéntica a la de las relaciones no recursivas 1:1, excepto que los renglones de hijo y padre se ubican en la misma relación. Puede pensar el proceso de la forma siguiente: imagine que la relación es entre dos relaciones diferentes. Determine dónde van las llaves y después combine ambas relaciones en una sola.

Con el fin de ilustrar lo anterior, considere la relación RECOMENDADO-POR en la figura 6-15(b). Ésta es una relación 1:N, como se muestra en los datos de ejemplo en la figura 6-17(a). Cuando se colocan estos datos en una relación, un renglón representa al que recomienda, y el otro a quienes han sido recomendados. El renglón de quien recomienda adopta el papel de padre y los renglones de los recomendados, el papel de hijo. Al igual que con todas las relaciones 1:N, colocamos la llave del padre en el hijo. En la figura 6-17(b) se coloca el número del que recomienda en todos los renglones de los que han sido recomendados.

Ahora considere las relaciones recursivas M:N. La relación TRATADO-POR en la figura 6-15(c) representa la situación en la que los doctores se prescriben un tratamiento uno a otro. Los datos de ejemplo se muestran en la figura 6-18(a). Al igual que en las

➤ FIGURA 6-15

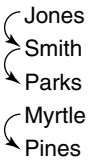
Ejemplo de relaciones recursivas: (a) relación recursiva 1:1, (b) relación recursiva 1:N y (c) relación recursiva N:M



➤ FIGURA 6-16

Ejemplo de relación recursiva 1:1: (a) datos de muestra para relación recursiva 1:1, (b) primera alternativa para representar una relación 1:1, y (c) segunda alternativa para representar una relación recursiva 1:1

Persona



(a)

Relación PERSONA1

Persona	PersonaPatrocinada
Jones	Smith
Smith	Parks
Parks	nulo
Myrtle	Pines
Pines	nulo

Restricción de integridad referencial:
PersonaPatrocinada en PERSONA1
debe existir en Persona, en PERSONA1

(b)

Relación PERSONA2

Persona	PersonaPatrocinadaPor
Jones	nulo
Smith	Jones
Parks	Smith
Myrtle	nulo
Pines	Myrtle

Restricción de integridad referencial:
PersonaPatrocinadaPor en PERSONA2
debe existir en Persona, en PERSONA2

(c)

➤ FIGURA 6-17

Ejemplo de una relación recursiva 1:N: (a) datos de muestra para la relación RECOMENDADO-POR, y (b) representación de una relación recursiva 1:N mediante una relación

Número de Cliente	Recomendación de estos clientes
100	200, 400
300	500
400	600, 700

(a)

Relación CLIENTE

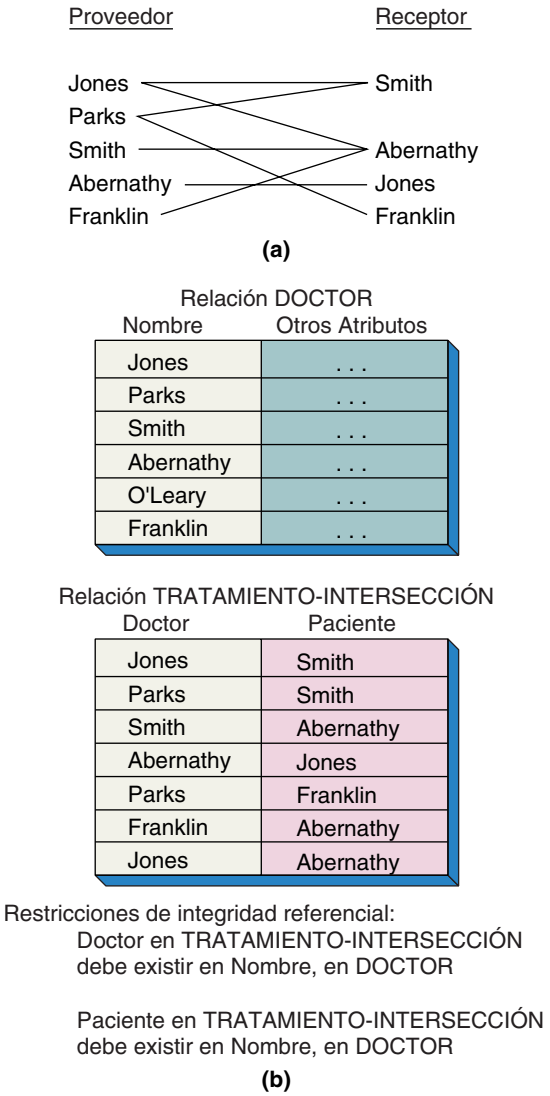
NúmerodeCliente	DatosdelCliente	RecomendadoPor
100	. . .	nulo
200	. . .	100
300	. . .	nulo
400	. . .	100
500	. . .	300
600	. . .	400
700	. . .	400

Restricción de integridad referencial:
RecomendadoPor en CLIENTE debe existir
en NúmerodeCliente, en CLIENTE

(b)

➤ FIGURA 6-18

Ejemplo de una relación recursiva M:N: (a) datos de muestra para la relación TRATADO-POR, y (b) representación de una relación recursiva M:N mediante relaciones



otras relaciones M:N, debemos crear una tabla de intersección que muestre los pares de renglones relacionados. El nombre del doctor en la primera columna es el que prescribió el tratamiento, y el del doctor en la segunda columna es aquel que lo recibió. Esta estructura se muestra en la figura 6-18(b).

Las relaciones recursivas son representadas de la misma manera que las otras relaciones. Sin embargo, los renglones de las tablas pueden representar dos funciones diferentes. Algunos son los renglones padre y los otros, los renglones hijo. Si se supone que una llave es la del padre y si el renglón no tiene padre, su valor será nulo. Si se supone que una llave es la llave hijo y el renglón no tiene hijo, su valor será nulo.

Representación de relaciones ternarias y de orden superior

Las relaciones ternarias se representan usando las técnicas que se acaban de describir, pero con frecuencia hay una consideración especial que necesita estar documentada como una regla del negocio. Considere, por ejemplo, las entidades PEDIDO, CLIENTE y VENDEDORes. En la mayoría de los casos podemos tratar esta relación ternaria como dos relaciones binarias por separado.

Por ejemplo, suponga que un PEDIDO tiene un solo CLIENTE, pero que un CLIENTE puede tener muchos PEDIDOs. Por lo tanto, esta relación es binaria N:1. De manera si-

milar, suponga que PEDIDO sólo tiene un VENDEDOR y un VENDEDOR tiene muchos PEDIDOs. Esta relación también es binaria N:1.

Ambas relaciones se pueden representar usando las técnicas antes descritas. Representamos lo primero colocando la llave de CLIENTE en PEDIDO y lo segundo, colocando la llave de VENDEDOR en PEDIDO. De esta manera, hemos tratado la relación ternaria entre PEDIDO:CLIENTE:VENDEDOR como dos relaciones binarias por separado.

Sin embargo, suponga que el negocio tiene una regla que establece que cada CLIENTE puede realizar pedidos sólo con un VENDEDOR en particular. En este caso, la relación ternaria PEDIDO:CLIENTE:VENDEDOR está limitada por una relación binaria adicional N:1 entre CLIENTE y VENDEDOR. Para representar la restricción necesitamos agregar las llaves de VENDEDOR a CLIENTE. Las tres relaciones serán las siguientes:

PEDIDO (NúmerodePedido, atributos sin datos llave, *NúmerodeCliente, NúmerodeVendedor)*
CLIENTE (NúmerodeCliente, atributos sin datos llave, *NúmerodeVendedor)*
VENDEDOR (NúmerodeVendedor, atributos sin datos llave)

La restricción de que sólo un VENDEDOR llame a un CLIENTE en particular significa que únicamente ciertos valores de NúmerodeCliente y NúmerodeVendedor pueden existir juntos en PEDIDO. Por desgracia, no hay forma de expresar esta restricción usando el modelo relacional. Debe estar documentado en el diseño; sin embargo, se debe cumplir ya sea por procedimientos almacenados (stored procedures) o mediante programas de aplicación. Véase la figura 6-19(a).

Otros tipos de restricciones binarias son: NO DEBE y DEBE CUBRIR. En una restricción NO DEBE, la relación binaria indica aquellas combinaciones que no se permite que ocurran en la relación ternaria. Por ejemplo, la relación ternaria RECETA:MEDICAMENTO:CLIENTE puede estar restringida por una relación binaria en la tabla

➤ FIGURA 6-19

Ejemplos de restricciones binarias en las relaciones ternarias: (a) ejemplo de una restricción binaria DEBE CUBRIR, sobre una relación ternaria

Tabla VENDEDOR

NúmerodeVendedor	Otros datos sin llave
10	
20	
30	

Tabla CLIENTE

NúmerodeCliente	Otros datos sin llave	*NúmerodeVendedor*
1000		10
2000		20
3000		30

Restricción binaria DEBE

Tabla PEDIDO

NúmerodePedido	Otros datos sin llave	*NúmerodeVendedor*	*NúmerodeCliente*
100		10	1000
200		20	2000
300		10	1000
400		30	3000
500			2000

Sólo se permiten 20 aquí

(a)

➤ FIGURA 6-19

(Continuación)

((b) Ejemplo de una restricción binaria NO DEBE sobre una relación ternaria

Tabla MEDICAMENTO

NúmerodeMedicamento	Otros datos sin llave
10	
20	
30	
45	
70	
90	

Tabla ALERGIA

NúmerodeCliente	NúmerodeMedicamento	Otros datos sin llave
1000	10	
1000	20	
2000	20	
2000	45	
3000	30	
3000	45	
3000	70	

Restricción binaria NO DEBE

Tabla RECETA

NúmerodeReceta	Otros datos sin llave	*NúmerodeMedicamento*	*NúmerodeCliente*
100		45	1000
200		10	2000
300		70	1000
400		20	3000
500			2000

No pueden aparecer aquí ni 20 ni 45

(b)

ALERGIA, lo cual indica los medicamentos que no debe tomar un paciente. Véase la figura 6-19(b).

En una restricción DEBE CUBRIR, la relación binaria indica todas las combinaciones que tienen que aparecer en la relación ternaria. Por ejemplo, considere la relación AUTO:REPARACIÓN:TAREA. Suponga que una reparación consta de un número de TAREAs, las cuales deben ser llevadas a cabo para que tal REPARACIÓN se concrete. En este caso, en la relación AUTO-REPARACIÓN, cuando a un automóvil se le ha asignado una REPARACIÓN todas las TAREAs para que sea reparado deben aparecer como renglones en esta relación. Véase la figura 6-19(c).

Ninguno de los tres tipos de restricciones binarias analizadas se pueden representar en el diseño relacional. En lugar de esto, todas las relaciones deben ser tratadas como una combinación de relaciones binarias. Sin embargo, las restricciones se deben documentar como parte del diseño.

Representación de relaciones ES-UN (subtipos)

La estrategia para representar subtipos, o relaciones ES-UN, difiere de la estrategia que se usa en una relación TIENE-UN. Considere el ejemplo de CLIENTE con los atributos NúmerodeCliente, NombredelCliente y CantidadAdeudada. Suponga que hay tres subtipos de CLIENTE, a saber, CLIENTE-PERSONA, CLIENTE-SOCIEDAD y CLIENTE-CORPORACIÓN, con los siguientes atributos:

➤ FIGURA 6-19

(Continuación)

(c) Ejemplo de una restricción binaria DEBE CUBRIR, con respecto a una relación ternaria

Tabla REPARACIÓN

NúmerodeReparación	Otros datos sin llave
10	
20	
30	
40	

Tabla TAREA

NúmerodeTarea	Otros datos sin llave	*NúmerodeReparación*
1001		10
1002		10
1003		10
2001		20
2002		20
3001		30
4001		40

Restricción binaria DEBE CUBRIR

Tabla AUTO-REPARACIÓN

NúmerodeFactura	NúmerodeReparación	NúmerodeTarea	Otros datos sin llave
100	10	1001	
200	10	1002	
300	10	1003	
400	20	2001	
500	20		

aquí debe aparecer el 2002

(c)

CLIENTE-PERSONA: Dirección, NúmerodeSeguroSocial

CLIENTE-SOCIEDAD: NombredelSocioAdministrador, Dirección, NúmerodeIdentificaciónFiscal

CLIENTE-CORPORACIÓN: PersonadeContacto, NúmeroTelefónico, NúmerodeIdentificaciónFiscal

Para representar esta estructura mediante relaciones, definimos una para el subtipo (CLIENTE) y otra para cada subtipo. Después, colocamos cada atributo del subtipo en la relación respectiva y cada uno de los atributos de los subtipos en las relaciones que los representan. En este punto, las relaciones de subtipo no tienen una llave. Para crear una, agregamos la llave del subtipo, o NúmerodeCliente, para cada uno de los subtipos. La lista final de relaciones es:

CLIENTE (NúmerodeCliente, NombredelCliente, CantidadAdeudada)

CLIENTE-PERSONA (NúmerodeCliente, Dirección, NúmerodeSeguroSocial)

CLIENTE-SOCIEDAD (NúmerodeCliente, NombredelSocioAdministrador, Dirección, NúmerodeIdentificaciónFiscal)

CLIENTE-CORPORACIÓN (NúmerodeCliente, PersonadeContacto, NúmeroTelefónico, NúmerodeIdentificaciónFiscal)

Observe que con esta estructura la relación entre un renglón en CLIENTE y un renglón en uno de los subtipos es 1:1. El cliente no tiene más de un renglón en una relación de subtipo, y cada subtipo corresponde únicamente a un renglón del subtipo. Dependiendo de las restricciones de la aplicación, es posible que un renglón en CLIENTE corresponda a múltiples renglones, cada uno de un subtipo diferente. Pero ningún renglón de CLIENTE puede corresponder a más de un renglón en la *misma* relación de subtipo.

Es posible que uno o más de los subtipos tengan una llave propia. Por ejemplo, la aplicación puede buscar o llamar a un NúmerodeClienteCorporación que sea distinto al NúmerodeCliente. En este caso, la llave de CLIENTE-CORPORACIÓN es NúmerodeClienteCorporación. Debido a que la relación entre CLIENTE y CLIENTE-CORPORACIÓN es 1:1, se puede establecer colocando la llave de una en la otra. Con frecuencia se considera un mejor diseño colocar la llave de la relación supertipo en la llave de la relación subtipo. Para este caso, la estructura de CLIENTE-CORPORACIÓN es:

CLIENTE-CORPORACIÓN (NúmerodeClienteCorporación, NúmerodeCliente, PersonadeContacto, NúmeroTelefónico, NúmerodeIdentificaciónFiscal).

➤ EJEMPLO DE DISEÑO

La figura 6-20(a) es una copia del diagrama E-R que se presentó en la figura 3-9 del capítulo 3. Contiene todos los elementos básicos usados en los diagramas E-R. Para representar este diagrama por medio de relaciones, comenzamos por establecer una relación para cada entidad. Supongamos que las llaves son las siguientes:

RELACIÓN	LLAVE
EMPLEADO	NúmerodeEmpleado
INGENIERO	NúmerodeEmpleado
AUTOBÚS	NúmerodeLicencia
SERVICIO	NúmerodeFactura
CLIENTE	NúmerodeCliente
CLIENTE-SERVICIO	(NúmerodeFactura, NúmerodeCliente)
CERTIFICACIÓN-INGENIERO	(NúmerodeEmpleado, NombredeCertificación)
CERTIFICACIÓN	NombredeCertificación

El siguiente paso es confrontar cada una de estas relaciones con el criterio de normalización. El ejemplo no indica qué atributos deben ser representados, así que no podemos determinar las restricciones. Supondremos que estas relaciones están en DK/NF, aunque en la práctica necesitaríamos verificar esa suposición con la lista de atributos y restricciones. Por ahora, nos concentraremos en la representación de las relaciones. Las relaciones y sus atributos llaves (incluyendo llaves externas) están listados en la figura 6-20(b).

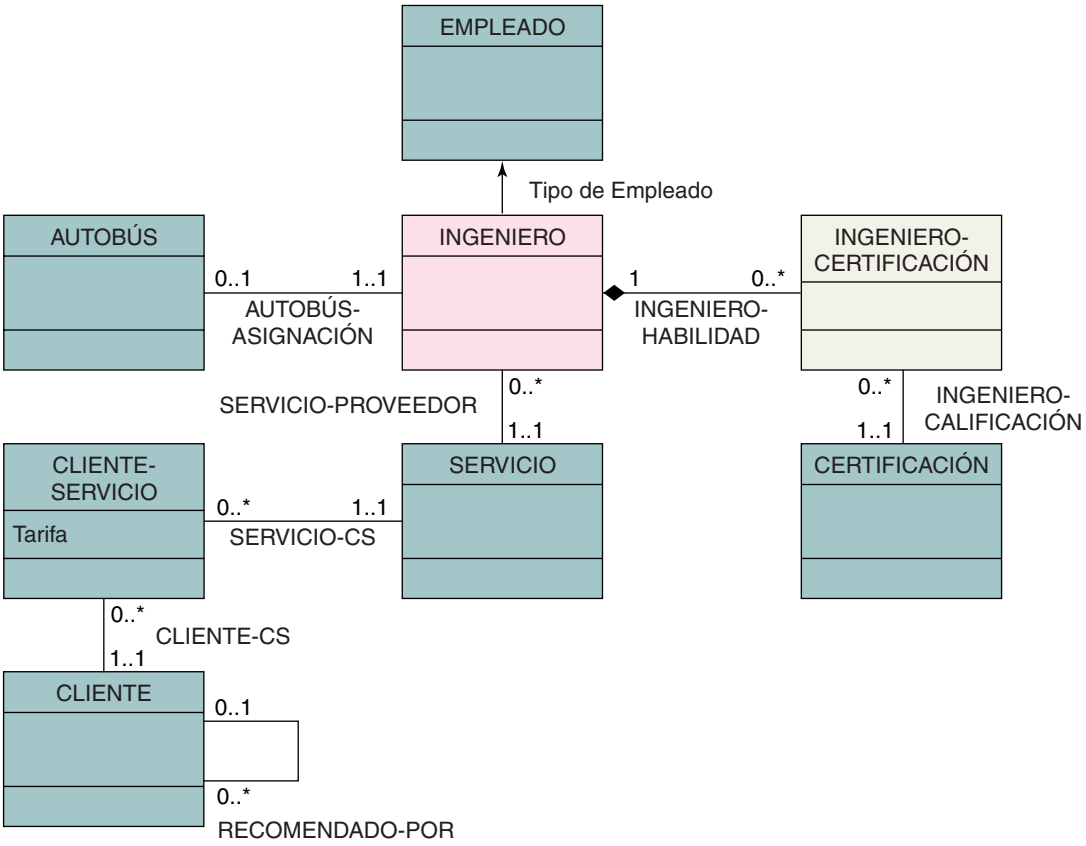
La relación entre EMPLEADO e INGENIERO ya está representada porque las relaciones tienen la misma llave, NúmerodeEmpleado. INGENIERO y AUTOBÚS tienen una relación 1:1 y así pueden estar relacionados por la colocación de la llave de una en la otra. Debido a que un autobús debe ser asignado a un empleado, no habrá valores nulos si se coloca NúmerodeEmpleado en AUTOBÚS, y así lo haremos.

Para la relación 1:N entre INGENIERO y SERVICIO colocamos la llave de INGENIERO (el padre) en SERVICIO (el hijo). La relación entre SERVICIO y CLIENTE es M:N, así que debemos crear una relación de intersección. Debido a que esta relación tiene un atributo, Tarifa, éste se agrega a la relación de intersección, CLIENTE-SERVICIO. Para la relación recursiva 1:N, RECOMENDADO-POR, agregamos el atributo RecomendadoPor a CLIENTE. El nombre *RecomendadoPor* implica, correctamente, que la llave del padre —el cliente que recomienda— se colocó en la relación.

Debido a que INGENIERO-CERTIFICACIÓN es dependiente de un identificador de INGENIERO, sabemos que NúmerodeEmpleado debe ser parte de su llave; así, la llave es compuesta (NúmerodeEmpleado, NombredeCertificación). La relación de dependencia es 1:N y se realizará por NúmerodeEmpleado. Por último, la relación entre CERTIFICACIÓN e INGENIERO-CERTIFICACIÓN es 1:N, así que normalmente agregaría-

➤ FIGURA 6-20

Representación relacional de un ejemplo de diagrama E-R: (a) diagrama E-R del capítulo 3, y (b) relaciones necesarias para representar este diagrama E-R



(a)

EMPLEADO (NúmerodeEmpleado, otros atributos sin llave de EMPLEADO . . .)

INGENIERO (NúmerodeEmpleado, otros atributos sin llave de INGENIERO . . .)

AUTOBÚS (NúmerodeLicencia, otros atributos sin llave de AUTOBÚS, *NúmerodeEmpleado*)

SERVICIO (NúmerodeFactura, otros atributos sin llave de SERVICIO, *NúmerodeEmpleado*)

CLIENTE (NúmerodeCliente, otros atributos sin llave de CLIENTE, *RecomendadoPor*)

CLIENTE-SERVICIO (NúmerodeFactura, NúmerodeCliente, Tarifa)

INGENIERO-CERTIFICACIÓN (NúmerodeEmpleado, NombredeCertificación, otros atributos sin llave de INGENIERO-CERTIFICACIÓN)

CERTIFICACIÓN (NombredeCertificación, otros atributos sin llave de CERTIFICACIÓN)

(b)

mos la llave de CERTIFICACIÓN (el padre) a INGENIERO-CERTIFICACIÓN. Pero ésta ya es parte de la relación, así que no es necesario agregarla.

Estudie este ejemplo para asegurarse de que comprende los diferentes tipos de relaciones y cómo se expresan en términos de las relaciones. Todos los elementos del

modelo E-R se representan en la figura 6-20. Véase la pregunta 6.40 referente a las restricciones de integridad referencial.

➤ ÁRBOLES, REDES Y LISTAS DE MATERIALES

Aunque ni el modelo E-R ni el de objeto semántico hacen cualquier suposición acerca de los patrones de relaciones entre entidades, algunos se repiten muy a menudo, así que les han asignado nombres especiales. Estos patrones son árboles, redes simples, redes complejas y listas de materiales. Introducimos el concepto de estos patrones aquí, en el contexto del modelo E-R.

ÁRBOLES

Un **árbol,** o **jerarquía,** como a veces se le llama, es una estructura de datos en la cual sus elementos sólo tienen relaciones de uno a muchos con otro. Cada uno de los elementos tiene cuando mucho un padre. La figura 6-21 es un ejemplo de un árbol. De acuerdo con la terminología estándar, cada elemento se llama **nodo,** y las relaciones entre los elementos, **ramas.** El nodo en la parte superior del árbol se llama **raíz** (¡qué metáfora, en la naturaleza la raíz de los árboles está en la parte inferior!). En la figura 6-21 el nodo 1 es la raíz del árbol.

Cada nodo de un árbol, excepto la raíz, tiene un **padre,** el nodo inmediato superior. Así, el nodo 2 es el padre del nodo 5; el nodo 4 es el padre del nodo 8, y así sucesivamente. Como ya se mencionó, los árboles se distinguen de otras estructuras de datos en que cada nodo tiene cuando mucho un padre. Decimos que máximo un padre porque el nodo raíz no tiene padre.

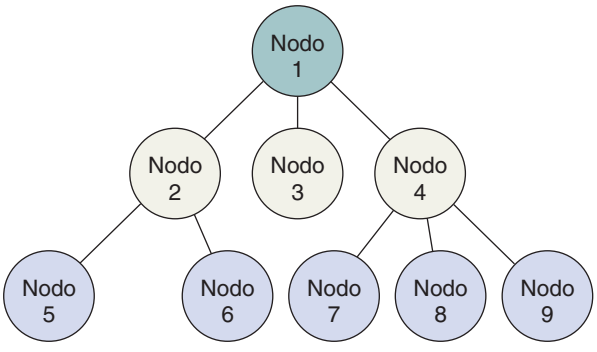
Los descendientes de un nodo se llaman **hijos.** En general, no hay un límite en el número de hijos que puede tener un nodo. El nodo 2 tiene dos hijos, los nodos 5 y 6; el nodo 3 no tiene hijo; y el nodo 4 tiene tres hijos, los nodos 7, 8 y 9. Los nodos que tienen el mismo padre se llaman **gemelos,** o **hermanos.** Por ejemplo, los nodos 5 y 6 son gemelos o hermanos.

La figura 6-22(a) ilustra un árbol de entidades en el que se pueden apreciar varias relaciones uno a muchos entre entidades en un sistema universitario. Las universidades constan de muchos departamentos, los que a su vez tienen muchos profesores y muchos empleados administrativos. Por último, los profesores asesoran a muchos estudiantes que han recibido muchas calificaciones. Hay seis tipos diferentes de entidades en esta estructura, pero todas las relaciones son 1:N.

Para representar un árbol de entidades usando el modelo relacional, simplemente aplicamos los conceptos descritos en las secciones anteriores de este capítulo. Primero transformamos cada entidad en una relación. Después examinamos las relaciones generadas en función del criterio de normalización y si es necesario se subdividen. Repre-

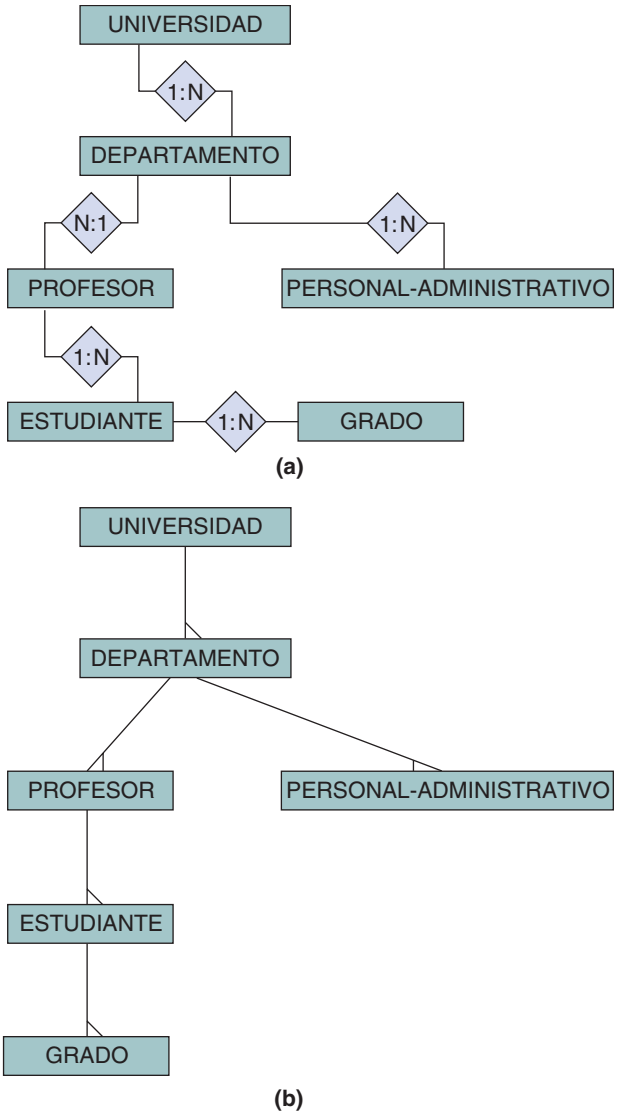
➤ FIGURA 6-21

Ejemplo de un árbol



➤ FIGURA 6-22

Representación de un árbol por medio de relaciones: (a) árbol compuesto por entidades y (b) representación de este árbol mediante relaciones



sentamos las relaciones 1:N almacenando la llave del padre en el hijo. La figura 6-22(b) es un diagrama de estructura de datos correspondiente al árbol de la figura 6-22(a).

En resumen, una jerarquía, o árbol, es un conjunto de registros organizados de tal forma que las relaciones son 1:N. Todos los registros tienen exactamente un padre, excepto la raíz. Una jerarquía se puede representar mediante un conjunto de relaciones usando los métodos antes descritos. Las jerarquías son comunes en los negocios, especialmente en aplicaciones de manufactura.

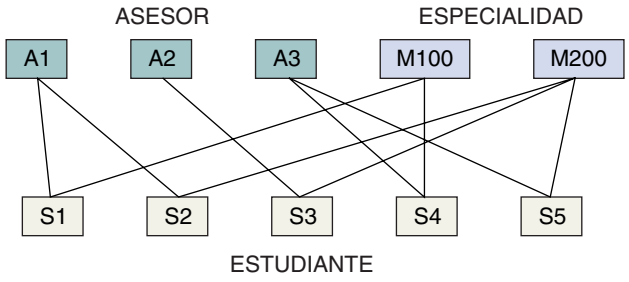
Redes simples

Una **red simple** es también una estructura de datos de elementos que sólo tienen relaciones de uno a muchos. Sin embargo, en una red simple los elementos pueden tener más de un padre, siempre y cuando sean de tipos diferentes. Por ejemplo, en la red simple que se muestra en la figura 6-23, cada entidad ESTUDIANTE tiene dos padres, una entidad ASESOR y una entidad ESPECIALIDAD. La estructura de datos que se muestra en la figura 6-23 no es un árbol porque las entidades ESTUDIANTE tienen más de un padre.

La figura 6-24(a) muestra la estructura general de esta red simple. Observe que las relaciones son de uno a muchos, pero que ESTUDIANTE tiene dos padres. En esta figu-

FIGURA 6-23

Ejemplo de una red simple



ra, los registros padre están en la parte superior y los registros hijo, debajo de ellos. Este arreglo es conveniente pero no esencial. Puede ver una red simple representada con padres al lado o debajo del hijo. Puede identificar redes simples en estos arreglos por el hecho de que un registro de tipo simple participa como un hijo en dos (o más) relaciones uno a muchos.

Para representar una red simple de entidades con el modelo relacional seguimos los procedimientos descritos anteriormente. Primero, transformamos cada entidad en una relación y si es necesario normalizamos las relaciones. Después representamos cada relación 1:N almacenando la llave de la relación padre en la relación hijo. El resultado de este proceso para la red en la figura 6-24(a) se muestra en la figura 6-24(b).

Redes complejas

Una **red compleja** es una estructura de datos de elementos en la cual cuando menos una de las relaciones es de muchos a muchos. La red compleja en la figura 6-25(a) ilustra la relación entre facturas, artículos de línea, partes y distribuidores. Dos de las tres relaciones son 1:N y la tercera es M:N. Debido a que hay cuando menos una relación de muchos a muchos, a esta estructura se le llama red compleja.

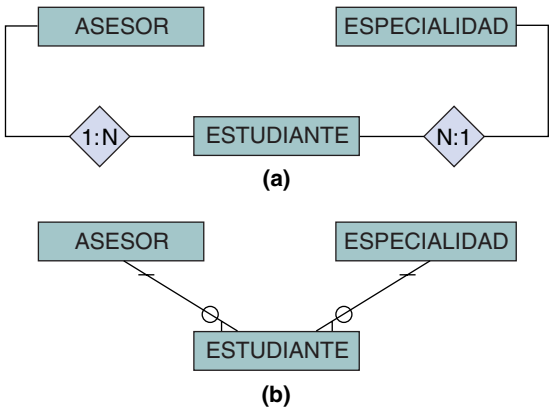
Como analizamos, las relaciones M:N no tienen representación directa en el modelo relacional. En consecuencia, antes de que se pueda almacenar esta estructura en forma relacional debemos definir una relación de intersección. En la figura 6-25(b) la relación de intersección es Proveedor-Parte.

Lista de materiales

Una **lista de materiales** es una estructura de datos especial que ocurre con frecuencia en aplicaciones de manufactura. De hecho, estas estructuras dieron mayor impulso al desarrollo de la tecnología de bases de datos en la década de 1960.

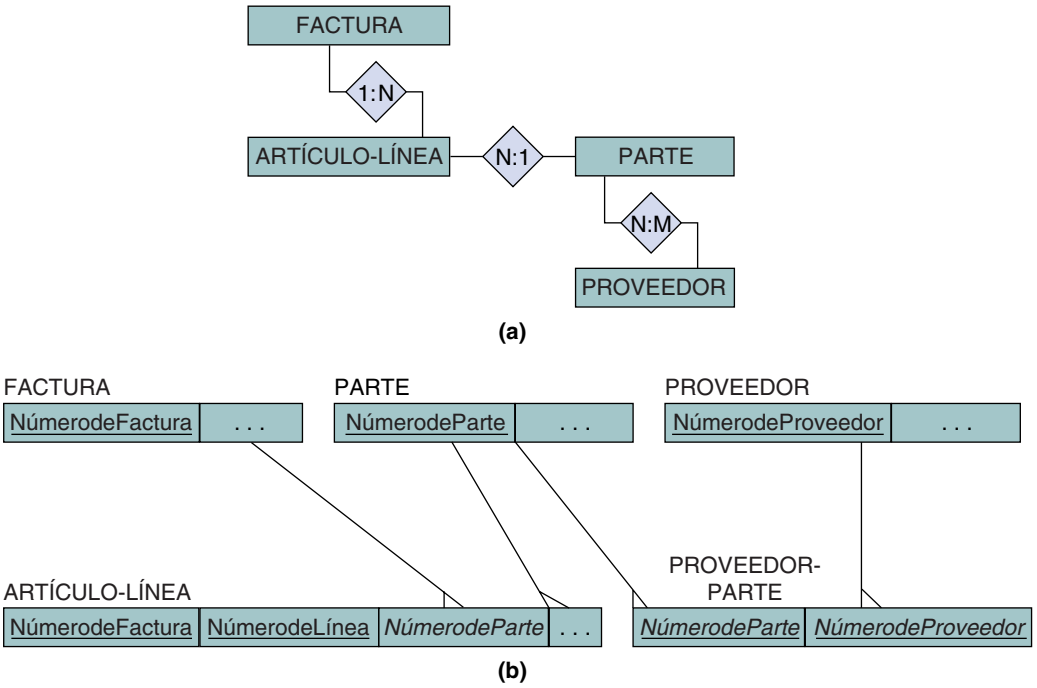
FIGURA 6-24

Representación de una red simple por medio de relaciones: (a) red simple compuesta de entidades, y (b) su representación mediante relaciones



➤ FIGURA 6-25

Representación de una red compleja por medio de relaciones: (a) red compleja compuesta de entidades, y (b) su representación mediante relaciones

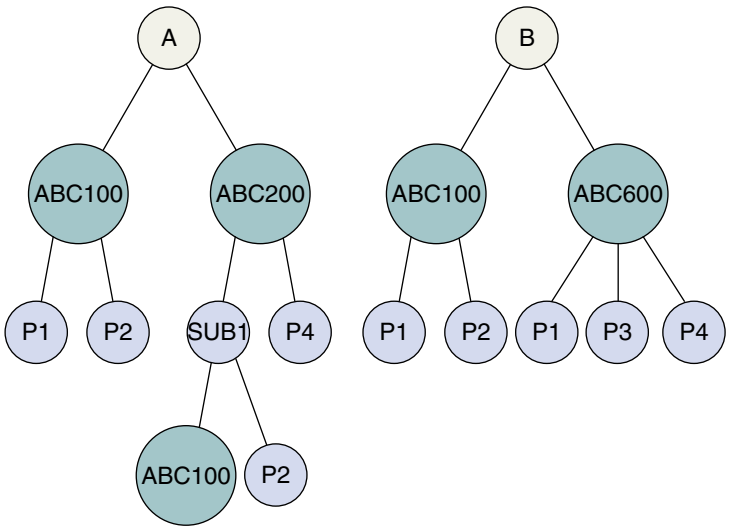


La figura 6-26 es un ejemplo de una lista de materiales, la cual muestra las partes que constituyen los productos. Cuando se observa desde el punto de vista de un producto específico, por ejemplo el producto A, esta estructura de datos es una jerarquía. Debido a que se puede usar una parte en más de un producto, esta estructura en realidad es una red. Por ejemplo, la refacción o parte ABC100 tiene dos padres: el producto A y el producto B.

Una lista de materiales se puede representar de varias formas por medio de relaciones. La más común es considerarla como una relación recursiva M:N. Una parte (producto, ensamble, subensamble, etc.) contiene muchos elementos. Al mismo tiempo, puede

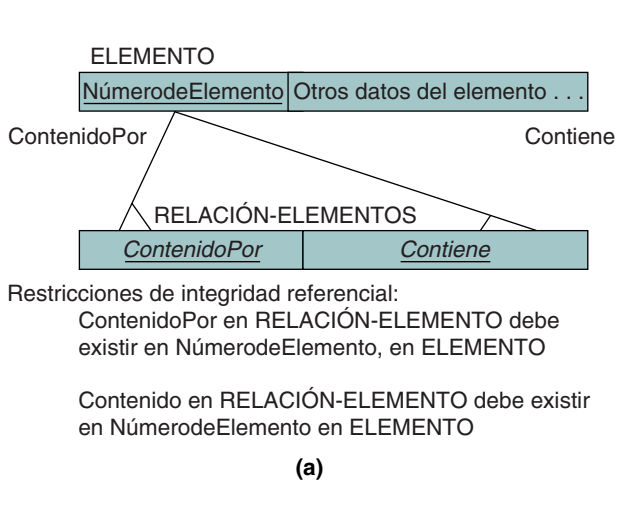
➤ FIGURA 6-26

Ejemplo de una lista de materiales



➤ FIGURA 6-27

Representación de una lista de materiales con relaciones: (a) relaciones que representan una lista de materiales, y (b) datos para la relación de intersección RELACIÓN-ELEMENTOS



Contenido Por	Contiene
A	ABC100
A	ABC200
B	ABC100
B	ABC600
ABC100	P1
ABC100	P2
ABC200	SUB1
ABC200	P4
ABC600	P1
ABC600	P3
ABC600	P4
SUB1	ABC100
SUB1	P2

Observe que el elemento A contiene a una ABC100 y el elemento ABC100 está contenido en una A

(b)

ser que muchos elementos la contengan. La figura 6-27(a) muestra la estructura de datos general de la relación recursiva M:N, y la figura 6-27(b) proporciona un ejemplo de la relación de intersección creada para representar esta lista de materiales.

Antes de concluir el presente capítulo necesitamos desarrollar dos temas importantes: llaves sustitutas y valores nulos.

Llaves sustitutas

Una **llave sustituta** es un identificador único proporcionado por el sistema, que se usa como la llave primaria de una relación. Los valores de una llave sustituta no tienen significado para los usuarios y normalmente están ocultos en las formas y los reportes. El DBMS no permitirá que se cambie el valor de una llave sustituta.

Hay dos razones para el uso de las llaves sustitutas: una pragmática y otra filosófica. Consideremos la primera.

LA RAZÓN PRAGMÁTICA. Suponga que hay una relación M:N entre las dos tablas siguientes:

MARISCO (NombredelMarisco, Tamaño, Color, Descripción)

y

PLAYA (NombredelaPlaya, Tipo, OrientacióndelaOla, TemperaturadelaCorrientedeAgua)

La relación entre estas dos tablas es de M:N porque un marisco puede ser encontrado en muchas playas, y una playa puede tener muchos tipos de mariscos. Suponga que los nombres de los mariscos son un dato de texto como *costata Cirtopleura* y están modelados con un dominio físico de texto 50. Además suponga que los nombres de la playa también están en un texto, como *Isla Perla, Playa Embrujada, Extremo Este,* y están modelados con un dominio físico de texto 75. Obviamente los índices que necesitaremos crear para reforzar la particularidad de estas columnas serán muy extensos. (Véase el apéndice A para mayor información acerca del uso de índices.)

Sin embargo, el mayor problema está en la tabla de intersección que representaría la relación M:N. Las dos columnas de esta tabla son (NombredelMarisco, NombredelaPlaya), las cuales se representan como texto 50 y texto 75, respectivamente. Por ejemplo, los datos e índices de esta tabla de intersección serán enormes. Si la base de datos

de los mariscos contiene 1000 variedades, que aparecen en un promedio de 100 playas, la tabla de intersección tendrá 100000 renglones de 125 caracteres de texto. Para hacer más complicado el problema, todos estos datos duplican NombredelMarisco, en MARISCO, y NombredelaPlaya, en PLAYA.

El almacenaje y procesamiento de estos índices se puede reducir considerablemente mediante la definición de una llave sustituta en las tablas MARISCO y PLAYA. Para lo anterior se necesita definir nuevas columnas en MARISCO y PLAYA, por ejemplo MariscoID y PlayaID, respectivamente. Estas columnas se definen en el DBMS como tipo AutoNumérico (o algo similar, dependiendo de la marca del DBMS). Cuando se define una columna de esta manera, cada vez que el DBMS crea un nuevo renglón se origina un nuevo y único valor para esa columna del nuevo renglón.

Así, un diseño corregido para estas tablas usando una llave sustituta sería:

MARISCO (MariscoID, NombredelMarisco, Tamaño, Color, Descripción)
PLAYA (PlayaID, NombredelaPlaya, Tipo, OrientacióndelaOla, TemperaturadelaCorrientedeAgua)

y

MARISCO_PLAYA_INT (MariscoID, PlayaID)

Los valores de la llave sustituta que son creados, por lo general, son enteros de 32 bites, o algo similar. Tales valores son compactos y fáciles de indizar y darán como resultado no sólo una drástica reducción del espacio en archivo, sino también un mejor desempeño.

Así, por razones pragmáticas, cada vez que una tabla tenga una llave primaria con texto extenso, o una llave compuesta de elementos grandes, considere el uso de una llave sustituta.

LA RAZÓN FILOSÓFICA. La razón filosófica para el uso de una llave sustituta es que sirve para mantener la identidad de la entidad. Cuando el usuario almacena un renglón en una tabla representa algo, que debe existir hasta que él lo elimine. La existencia de ese algo no debería depender de la presencia o la ausencia de ciertos valores de datos. Puesto que el usuario no puede cambiar las llaves sustitutas, y que garantizan su particularidad, representan la identidad de un renglón (o entidad).

Para aclarar lo anterior, considere las dos tablas en la figura 6-28(a), las cuales usan las llaves de datos más que las llaves sustitutas. Hay una relación de 1:N de ASESOR a ESTUDIANTE. La relación se realiza mediante la llave externa NombredelAsesor en ESTUDIANTE. Ahora, supongamos que Sesiones del Profesor quiere cambiar el nombre por el de Johnson. Suponga que se admite el cambio y que nuestra aplicación es demasiado fina como para diseminar ese cambio a través de una llave externa; así todos los valores de Sesión cambian a Johnson en NombredelAsesor en ESTUDIANTE. Los datos aparecerán como se muestra en la figura 6-28(b).

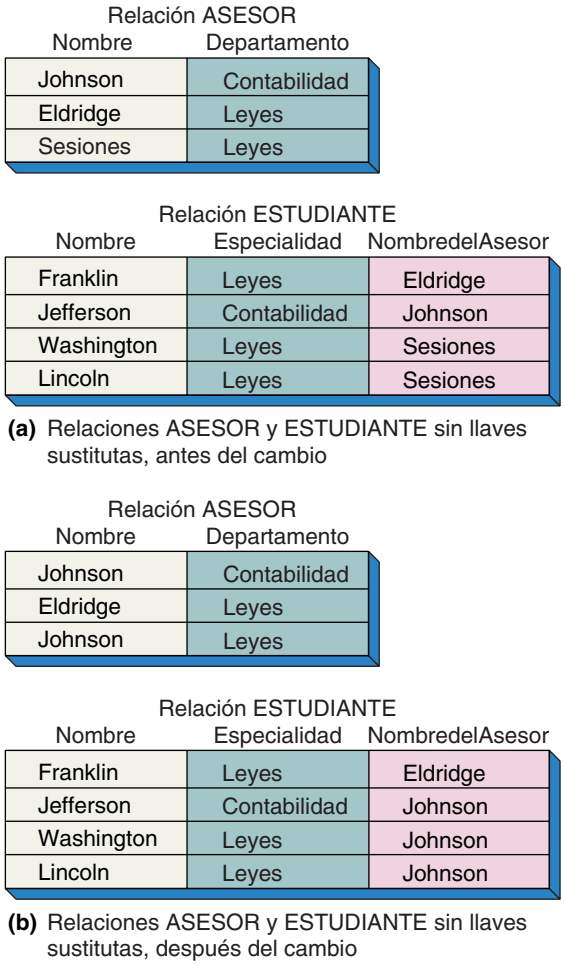
Tenemos un desorden. Hemos sobrecargado el nombre Johnson de tal forma que es imposible determinar a cuáles estudiantes asesora en Contabilidad, y a cuáles en Leyes. Lo que es peor, esta confusión es resultado de nuestro propio diseño, no se debe a ningún problema atribuible a los usuarios.

Quizás usted piense que el problema radica en que NombredelAsesor se debería definir como único, lo cual de hecho evitaría el problema; pero surge la pregunta: ¿quiénes somos, como creadores de bases de datos, para decir que los asesores no pueden tener nombres iguales? Si la semántica del negocio permite que los asesores tengan el mismo nombre, entonces, nuestro deber no es construir una base de datos que prohíba los nombres duplicados. Deformar el comportamiento de los usuarios en beneficio de la base de datos significa un mal diseño.

Considere este mismo escenario con llaves sustitutas. La figura 6-29(a) muestra los datos antes del cambio de nombre, y la figura 6-29(b) los muestra después del cambio. No hay confusión con las llaves porque las llaves sustitutas son usadas para representar la relación.

➤ FIGURA 6-28

Ejemplo de llave confusa



(a) Relaciones ASESOR y ESTUDIANTE sin llaves sustitutas, antes del cambio

(b) Relaciones ASESOR y ESTUDIANTE sin llaves sustitutas, después del cambio

Hay muchas maneras de expresar la noción de que las llaves sustitutas protegen la identidad. El mundo de la programación orientada a objetos también crea este mismo argumento en diferentes formas para la programación orientada a objetos. Todos los argumentos provienen del hecho de que en tanto un renglón tenga un identificador sin cambios que exista mientras el renglón se conserve, entonces la identidad de éste nunca se perderá.

Sin embargo, este beneficio tiene un costo. En el diseño de la figura 6-28(a) se puede observar un renglón ESTUDIANTE y saber inmediatamente quién es el asesor. Con datos clave como éstos no hay que hacer una búsqueda en la tabla ASESOR para determinar el nombre de éste, como sucede con los datos de la figura 6-29(b). También, si una base de datos intercambia datos con otras que empleen los datos clave, entonces el uso de llaves sustitutas puede crear problemas.

¿LLAVES SUSTITUTAS O NO? ¿Cuál es la respuesta? Los expertos difieren. Mientras que ninguno argumenta razones pragmáticas para usarlas, algunos dicen que su uso debería limitarse a casos en los que se deben usar, como en el ejemplo de los mariscos. Otros dicen que las llaves sustitutas se deberían usar todas al mismo tiempo, y que los datos llave no deberían emplearse nunca.

Mi propia experiencia es usar las llaves sustitutas casi todo el tiempo. Titubeo en cuanto a su uso en tablas con datos llave que son fáciles de indexar en forma natural, como por ejemplo en una tabla de PRODUCTO que tiene una columna NúmerodeProducto con valores únicos enteros. También, algunas veces no defino las llaves sustitutas en tablas que normalmente se utilizan para intercambiar datos con otras bases de datos.

Esta política significa que algunas bases de datos que he diseñado tienen una mezcla de datos y llaves sustitutas. Esto no me gusta y puede resultar confuso. En cierta

➤ FIGURA 6-29

Ejemplo de una llave no confusa con llaves sustitutas

Relación ASESOR

AsesorID	Nombre	Departamento
1	Johnson	Contabilidad
2	Eldridge	Leyes
3	Sesiones	Leyes

Relación ESTUDIANTE

EstudianteID	Nombre	Especialidad	AsesorFK
20	Franklin	Leyes	2
21	Jefferson	Contabilidad	1
22	Washington	Leyes	3
23	Lincoln	Leyes	3

(a) Las relaciones ASESOR y ESTUDIANTE con llaves sustitutas, antes del cambio

Relación ASESOR

AsesorID	Nombre	Departamento
1	Johnson	Contabilidad
2	Eldridge	Leyes
3	Johnson	Leyes

Relación ESTUDIANTE

EstudianteID	Nombre	Especialidad	AsesorFK
20	Franklin	Leyes	2
21	Jefferson	Contabilidad	1
22	Washington	Leyes	3
23	Lincoln	Leyes	3

(b) Las relaciones ASESOR y ESTUDIANTE con llaves sustitutas, después del cambio

medida, esta confusión se puede evitar nombrando siempre las columnas de las llaves sustitutas (surrogate key). Normalmente uso un esquema titulado *TableName*ID o *TableName*_SK para las columnas de llaves sustitutas.

Analice este aspecto con su profesor, el cual sin duda tendrá otras ideas y opiniones. Es una decisión artística el hecho de afirmar que, en mi opinión, es necesario hacer una tabla a la vez.

Valores nulos

Un **valor nulo** es un valor de atributo que nunca se ha asignado. El problema de los valores nulos es que son ambiguos. Un valor nulo puede significar: (a) que el valor es desconocido, (b) que no es apropiado, o (c) que el valor se acepta en blanco. Por ejemplo, considere el atributo FechadeDeceso en una relación CLIENTE. ¿Qué significa un valor nulo para FechadeDeceso? Podría significar que los usuarios no saben si el cliente está vivo o no; o también que el cliente es una corporación y que la FechadeDeceso no es apropiada; o podría significar ambas cosas: que el cliente es una persona y está viva.

Existen varias formas de eliminar estas ambigüedades. La primera es no permitirlas. Definir el atributo como se necesita, lo cual está bien siempre y cuando en la mente de los usuarios el atributo verdadero se requiera. Sin embargo, los usuarios se molestarán si se ven obligados a dar un valor de ColorPreferidodelCliente, si no es esencial para el funcionamiento de su negocio.

En el capítulo 3 vimos algunas formas de eliminar los valores nulos inapropiados, y definir los subtipos. Definir PACIENTE-MASCULINO y PACIENTE-FEMENINO como subtipos de PACIENTE, por ejemplo, evitaría que los pacientes masculinos tuvieran que proporcionar el número de embarazos, y que a las mujeres se les preguntara sobre la condición de su próstata. Sin embargo, esta solución es costosa, ya que obliga a la definición de dos nuevas tablas y se requiere unirlas para contar con todos los datos de PACIENTE.

Una tercera solución sería definir cada atributo con un valor inicial que sea reconocido en blanco. Un atributo de texto, por ejemplo, podría dar el valor inicial (*desconocido*). Los usuarios podrían subsecuentemente dar el valor (*no apropiado*) para no apropiado, cuando el valor sea inapropiado. Lo anterior sería más eficaz si estas elecciones aparecieran en cajas de texto desplegables (véase el capítulo 10). Mientras que esta solución funciona para atributos de texto, el problema persiste para atributos numéricos, fechas, booleanos y otros atributos que no son de texto. Por supuesto, una solución es modelarlos como datos de texto, así se puede introducir el valor (*desconocido*) y (*no apropiado*). Sin embargo, en ese caso tendrá que escribir su propio código editado para asegurarse de que se introducen números validos, o datos. También tendrá que proyectar los valores en el código del programa antes de efectuar operaciones numéricas o de datos.

Algunas veces la mejor solución es no hacer nada con respecto a los valores nulos. Si los usuarios pueden enfrentar la ambigüedad, o si la solución es muy costosa, sólo documente el hecho de que existe el problema y trabájelo. También, consulte el capítulo 9 para obtener más información acerca de las consecuencias de todos los valores nulos que se unen en las operaciones.

➤ RESUMEN

Para transformar los modelos de datos entidad-relación, cada entidad se representa mediante una relación. Los atributos de la entidad se convierten en los de la relación. Una vez que se ha creado la relación, se le debe examinar en función del criterio de normalización y, si es necesario, dividirla en dos o más relaciones.

Existen tres tipos de relaciones binarias TIENE-UN en el modelo E-R: 1:1, 1:N, y N:M. Para representar una relación 1:1, colocamos la llave de una relación en la otra. Las relaciones uno a uno a veces indican que dos relaciones han sido definidas en la misma entidad y se deben combinar en una sola.

Para representar una relación 1:N colocamos la llave del padre en el hijo. Por último, para representar una relación M:N, creamos una relación de intersección que contiene las llaves de las otras dos.

Las relaciones recursivas son aquellas en las que los participantes surgen de la misma clase de entidad. Hay tres tipos: 1:1, 1:N, y N:M, los cuales se representan de la misma manera que las relaciones no recursivas. Para las relaciones 1:1 y 1:N agregamos una llave sustituta que simboliza la entidad. Para una recursión N:M creamos una tabla de intersección que representa la relación M:N.

Las relaciones ternarias y de orden superior se pueden tratar como combinaciones de las relaciones binarias. Sin embargo, si se hace así, cualquiera de las restricciones binarias en las relaciones ternarias también se deben representar en el diseño. Debido a que no es posible imponer la restricción para el diseño relacional, se debe documentar como una regla del negocio. Ocurren tres tipos de estas restricciones: DEBE, NO DEBE y DEBE CUBRIR.

Las entidades de subtipo y supertipo (relaciones ES-UN) también se representan mediante relaciones. Una de ellas se define a través de la entidad supertipo, y otras por cada subtipo. Las llaves de las relaciones normalmente son las mismas y las relaciones entre los renglones se definen a través de estas llaves. Si no son las mismas, la llave de la relación subtipo se puede colocar en la relación supertipo, o la llave de la relación supertipo se coloca en la relación subtipo. Con frecuencia, la llave de la relación supertipo se coloca en la relación subtipo.

Las relaciones binarias se pueden combinar para formar tres clases de estructuras más complejas. Un árbol es un conjunto de tipos de registros en el que cada uno tiene exactamente un padre, con excepción de la raíz. En una red simple, los registros pueden tener padres múltiples, pero éstos deben ser de diferentes tipos. En una red compleja los registros tienen padres múltiples del mismo tipo. Otra forma de expresar lo anterior es que en una red compleja cuando menos una de las relaciones binarias es M:N.

Una lista de materiales es una estructura de datos que con frecuencia se ve en aplicaciones de manufactura. Estas estructuras se pueden representar mediante relaciones recursivas M:N.

Las llaves sustitutas son identificadores únicos que proporciona el sistema, las cuales se usan como las llaves primarias de una relación. Se usan por razones pragmáticas para reducir el tamaño de las llaves y de las llaves externas, así como para mejorar el desempeño. También se usan por razones filosóficas que sirven para mantener la identidad de las entidades. En general, se recomiendan estas llaves.

Un valor nulo es un valor que no ha sido asignado a un atributo. Dichos valores son ambiguos. Pueden significar que el valor es desconocido, que no es apropiado o que se acepta en blanco. Se pueden evitar mediante el requerimiento de los valores de los atributos, por el uso de subtipos, y mediante la distribución de valores sustitutos iniciales. También se pueden ignorar si la ambigüedad presente no es problema para los usuarios.

➤ PREGUNTAS DEL GRUPO I

6.1 Explique cómo se transforman en relaciones las entidades E-R.

6.2 ¿Por qué es necesario examinar las relaciones transformadas a partir de entidades, en función del criterio de normalización? ¿Bajo qué condiciones se deben alterar las relaciones si no son DK/NF? ¿En qué casos no se debe hacer?

6.3 Explique en qué difiere la representación de las entidades débiles de las entidades fuertes.

6.4 Enumere los tres tipos de relaciones binarias y dé un ejemplo sobre cada uno. No use los ejemplos de este texto.

6.5 Defina *llave externa* y dé un ejemplo.

6.6 Muestre dos formas diferentes para representar la relación 1:1 en su respuesta a la pregunta 6.4. Use diagramas de estructura de datos.

6.7 A partir de sus respuestas a la pregunta 6.6, describa un método para obtener datos acerca de las entidades, dando la llave de otra. Describa un método para obtener datos acerca de la segunda entidad, proporcionando la llave de la primera. Describa las respuestas para las alternativas que dio con respecto a la pregunta 6.6.

6.8 ¿Por qué algunas relaciones 1:1 se consideran cuestionables? ¿Bajo qué condiciones las relaciones 1:1 deben combinarse en una sola?

6.9 Defina los términos *padre* e *hijo* y dé un ejemplo sobre cada uno.

6.10 Muestre cómo se representa la relación 1:N en su respuesta a la pregunta 6.4. Use un diagrama de estructura de datos.

6.11 En su respuesta a la pregunta 6.10, describa un método con el que se obtengan datos para todos los hijos, considerando la llave del padre. Describa un método que proporcione datos para el padre, considerando la llave del hijo.

6.12 Para una relación 1:N explique por qué se debe colocar la llave del padre en el hijo, en lugar de colocar la del hijo en el padre.

6.13 Dé ejemplos sobre relaciones binarias 1:N, distintos a los de este texto, para:

a. Una relación opcional a opcional
b. Una relación opcional a obligatoria
c. Una relación obligatoria a opcional
d. Una relación obligatoria a obligatoria

Ejemplifique sus respuestas usando diagramas de estructura de datos.

6.14 Muestre cómo se representa la relación N:M en su respuesta a la pregunta 6.4. Use un diagrama de estructura de datos.

6.15 Para su respuesta a la pregunta 6.14 describa un método con el que se obtengan los hijos para una entidad, considerando la llave de la otra. También describa un método para obtener el hijo de la segunda entidad, a partir de la llave de la primera.

6.16 ¿Por qué no es posible representar relaciones N:M con la misma estrategia que se usa para representar las relaciones 1:N?

6.17 Explique el significado del término *relación de intersección*.

6.18 Defina tres tipos de relaciones binarias recursivas y proporcione un ejemplo de cada una.

6.19 Muestre cómo representar la relación recursiva 1:1 en su respuesta a la pregunta 6.18. ¿En qué difiere de la representación de una relación no recursiva 1:1?

6.20 Muestre cómo representar la relación recursiva 1:N en su respuesta a la pregunta 6.18. ¿En qué difiere de la representación de las relaciones no recursivas 1:N?

6.21 Muestre cómo representar la relación recursiva M:N en su respuesta a la pregunta 6-18. ¿En qué difiere esto de la representación de las relaciones no recursivas M:N?

6.22 Explique cómo usar las relaciones binarias para representar una relación ternaria. El ejemplo que proporcione deberá ser diferente a los expuestos.

6.23 En su respuesta a la pregunta 6.22, defina una restricción binaria para la relación ternaria y explique cómo representarla. Puesto que la restricción no puede ser obligatoria en el modelo relacional, ¿qué se debe hacer?

6.24 Proporcione ejemplos de restricciones binarias NO DEBE y DEBE CUBRIR, diferentes a los de este texto.

6.25 Mencione un ejemplo de un supertipo y dos o más subtipos, y muestre cómo se representa usando relaciones.

6.26 Defina *árbol*, *red simple* y *red compleja*.

6.27 Mencione un ejemplo de una estructura de árbol, diferente a los de este texto, y muestre cómo se representa por medio de relaciones.

6.28 Dé un ejemplo sobre red simple, diferente a los de este texto, y muestre cómo se representa por medio de relaciones.

6.29 Proporcione un ejemplo de una red compleja, diferente a los de este texto, y muestre cómo se representa por medio de relaciones.

6.30 ¿Qué es una lista de materiales? Dé un ejemplo diferente a los de este texto, y muestre cómo representar su ejemplo mediante relaciones.

6.31 Defina *llave sustituta* y describa dos razones para usarla.

6.32 Describa una situación diferente a la de este texto, sobre la cual existan buenas razones pragmáticas para usar llaves sustitutas.

6.33 Explique el enunciado "La llave sustituta sirve para mantener la identidad de la entidad". Explique por qué es importante.

6.34 ¿Cuáles son las tres posibles interpretaciones de los valores nulos?

6.35 Describa tres formas diferentes de evitar los valores nulos.

6.36 ¿Cuándo no representan un problema los valores nulos?

➤ PREGUNTAS DEL GRUPO II

6.37 Transforme el diagrama E-R para el Club de baile Jefferson (figura 3-19) en relaciones. Exprese su respuesta con un diagrama de estructura de datos y muestre las restricciones de integridad referencial.

6.38 Transforme el diagrama E-R para Paseos en barco San Juan (figura 3-21) en relaciones. Exprese su respuesta con un diagrama de estructura de datos y muestre las restricciones de integridad referencial.

6.39 Algunas de las relaciones en la figura 6-19 no son DK/NF. Identifíquelas y explique por qué no. ¿Qué forma normal tienen? ¿Cómo se puede justificar este diseño? ¿Cómo podría imponer la base de datos las restricciones binarias?

6.40 Establezca todas las restricciones de integridad referencial para las relaciones de la figura 6-20(b).

➤ PROYECTOS

A. Termine el proyecto A que está al final del capítulo 3, si aún no lo ha hecho. Transforme su diagrama E-R en un conjunto de relaciones. Si cualquiera de sus relaciones no está en DK/NF, justifique su decisión de crear relaciones sin normalizar.

B. Termine el proyecto B del final del capítulo 3, si no lo ha concluido. Transforme su diagrama E-R en un conjunto de relaciones. Si sus relaciones no están en DK/NF, justifique su decisión de haberlas creado sin normalizar.

C. Termine el proyecto C al final del capítulo 3, si todavía no lo ha concluido. Transforme su diagrama E-R en un conjunto de relaciones. Si no están en DK/NF, justifique su decisión de crear relaciones sin normalizar.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Si aún no lo ha hecho, cree diagramas E-R para las preguntas A y C en el proyecto FiredUp al final del capítulo 3.

A. Transforme el diagrama entidad-relación de la pregunta A al final del capítulo 3 en un conjunto de relaciones en forma normal dominio/llave. Para cada relación, especifique la llave primaria, las llaves candidatas, si las hay, y las llaves externas. Especifique todas las restricciones de integridad referencial. Si es necesario, establezca y justifique las suposiciones con respecto a las semánticas subrayadas de la aplicación.

B. Ajuste su respuesta a la pregunta A para permitir relaciones sin normalizar si las considera apropiadas. Justifique cualquiera de las relaciones sin normalizar que tenga. Si es necesario, establezca y justifique las suposiciones registrando las semánticas subrayadas de la aplicación.

C. Transforme el diagrama entidad-relación de la pregunta C al final del capítulo 3 en un conjunto de relaciones, preferentemente en forma normal dominio-llave. Si cualquiera de sus relaciones no están en dicha forma, explique por qué. Para cada relación

especifique la llave primaria, las llaves candidatas, si las hay, y la llave externa. Detalle todas las restricciones de la interrelación.

D. Ajuste su respuesta a la pregunta anterior para suponer que la casa, el fax y el teléfono celular representan por separado atributos de valor simple. ¿Es un mejor diseño que el de la respuesta a la pregunta C? Explique por qué.

CAPÍTULO 7

Diseño de bases de datos con modelos de objeto semántico

Este capítulo analiza la transformación de modelos de objeto semántico en diseños de bases de datos relacionales. Primero, describimos la transformación de cada uno de los siete tipos comunes de objetos semánticos. Después, ilustramos esos conceptos mostrando la modelación de objeto semántico y la presentación relacional de varios objetos del mundo real. Puesto que la mejor forma de aprender sobre este tema es trabajar con ejemplos propios, le sugerimos que realice los proyectos que se encuentran al final de este capítulo.

➤ TRANSFORMACIÓN DE OBJETOS SEMÁNTICOS EN DISEÑOS DE BASES DE DATOS RELACIONALES

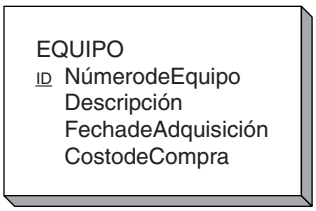
En el capítulo 4 abordamos el modelo de datos de objeto semántico y definimos siete tipos de objetos semánticos. En esta sección presentamos métodos para transformar cada uno de éstos en relaciones. Cuando se trabaja con objetos semánticos, los problemas de normalización son menos comunes que con los modelos E-R, porque la definición de los objetos semánticos normalmente separa los temas semánticos en grupos de atributos o en objetos. De esta manera, cuando se transforma un objeto en relaciones, éstas por lo general están en DK/NF o muy cerca de la forma normal dominio/llave.

OBJETOS SIMPLES

La figura 7-1 ilustra la transformación de un objeto simple en una relación. Recuerde que un objeto simple no tiene atributos multivaluados ni atributos de objeto. Por lo tanto, los objetos simples se pueden representar en la base de datos mediante una sola relación.

FIGURA 7-1

Representación relacional del ejemplo de objeto simple: (a) diagrama del objeto EQUIPO, y (b) relación que representa a EQUIPO



(a)

EQUIPO (NúmerodeEquipo, Descripción, FechadeAdquisición, CostodeCompra)

(b)

La figura 7-1(a) ejemplifica un objeto simple, EQUIPO, que se puede representar mediante una sola relación, como se muestra en la figura 7-1(b). Cada atributo del objeto se define como un atributo de la relación, y el que lo identifica, NúmerodeEquipo, se convierte en el atributo llave de la relación, que se denota subrayando NúmerodeEquipo en la figura 7-1(b).

La transformación general de objetos simples se ilustra en la figura 7-2. El objeto OBJETO1 es transformado en la relación R1. El atributo que identifica las instancias de OBJETO1 es O1; se convierte en la llave de la relación R1. Los datos que no son llave se representan mediante elipsis (. . .) en ésta y en las siguientes figuras.

Debido a que una llave es un atributo que identifica de forma única un renglón de una tabla, sólo pueden transformarse en llave los identificadores únicos —aquellos con ID subrayada—. Si no hay un identificador único en el objeto, entonces se debe crear uno, mediante la combinación de los atributos existentes para formar un identificador único, o mediante la definición de una llave sustituta.

Objetos compuestos

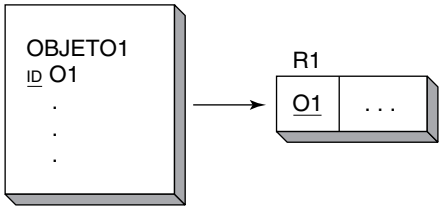
Un objeto compuesto es aquel que tiene uno o más atributos simples multivaluados, o grupo de atributos, pero que no tiene atributos de objeto. La figura 7-3(a) muestra un ejemplo de objeto compuesto: CUENTA-HOTEL. Para representarlo se crea una relación para el objeto base, CUENTA-HOTEL, y otra para la repetición del atributo en grupo, CargoDiario. Este diseño relacional se muestra en la figura 7-3(b).

En la llave de CARGO-DIARIO, NúmerodeFactura está subrayado porque es parte de esa llave, y está en cursivas porque también es una llave externa (es una llave de CARGO-DIARIO). FechadeCargo está subrayada porque es parte de la llave de CARGO-DIARIO, pero no está en cursivas porque no es una llave externa.

En general, los objetos compuestos se transforman mediante la definición de una relación para el objeto mismo y otra relación por cada atributo multivaluado. En la figura 7-4(a), el objeto OBJETO1 contiene dos grupos de atributos multivaluados, cada uno de los cuales está representado por una relación en el diseño de la base de datos. La llave de cada una de estas tablas es el compuesto del identificador del objeto, más el

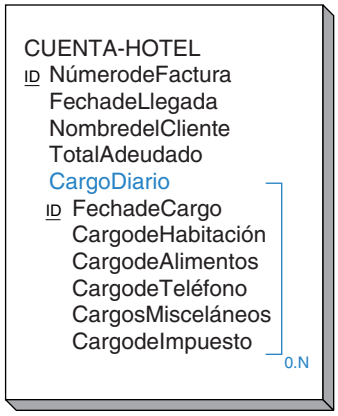
FIGURA 7-2

Transformación general de objeto simple en una relación



➤ FIGURA 7-3

Representación relacional de ejemplo de objeto compuesto: (a) diagrama del objeto CUENTA-HOTEL, y (b) representación de relaciones CUENTA-HOTEL



(a)

CUENTA-HOTEL (NúmerodeFactura, FechadeLlegada, NombredelCliente, TotalAdeudado)

CARGO-DIARIO (*NúmerodeFactura*, FechadeCargo, CargodeHabitación, CargodeAlimentos, CargodeTeléfono, CargosMisceláneos, CargodeImpuesto)

Restricción de integridad referencial:

NúmerodeFactura en CARGO-DIARIO debe existir en
NúmerodeFactura en CUENTA-HOTEL

(b)

identificador del grupo. Así, la representación de OBJETO1 es una relación R1 con la llave O1, una relación R2 con llave (O1, G1), y una relación R3 con llave (O1, G2).

La cardinalidad mínima del objeto con el grupo se especifica mediante la cardinalidad mínima del atributo de grupo. En la figura 7-4(a), la mínima del Grupo1 es 1 y para el Grupo2 es 0. Estas cardinalidades se muestran mediante una marca horizontal (en R2) y una elipse (en R3) en el diagrama de estructura de datos. La cardinalidad mínima del grupo de objetos siempre está predeterminada como 1, porque un grupo no puede existir sin el objeto que lo contiene. Estas cardinalidades mínimas se muestran mediante marcas horizontales en las líneas de relación en R1.

Como se observó en el capítulo 4, los grupos pueden estar anidados. La figura 7-4(b) muestra un objeto en el que el Grupo2 está anidado dentro del Grupo1. Cuando esto ocurre, la relación que representa al grupo anidado se subordina a la relación que representa al grupo que la contiene. En la figura 7-4(b) la relación R3 se subordina a la relación R2. La llave de R3 es la llave de R2, la cual es (O1, G1), más el identificador del Grupo2, que es G2; por lo tanto, la llave de R3 es (O1, G1, G2).

Asegúrese de que ha comprendido por qué las llaves en la figura 7-4(b) están estructuradas de esa manera. También observe que unos atributos están subrayados y en cursivas, y otros sólo subrayados, debido a que algunos atributos son locales y llaves externas, y otros sólo son llaves locales.

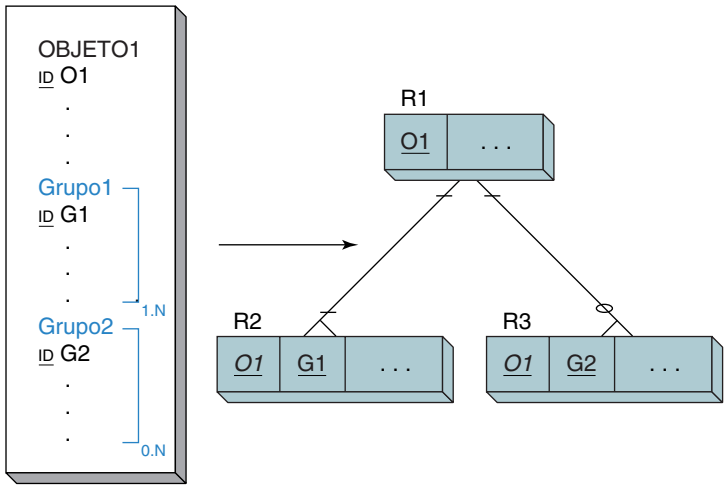
Objetos combinados

La representación relacional de objetos combinados es similar a la de entidades. De hecho, los objetos combinados y las entidades son en muchas formas completamente similares.

Como mencionamos en el capítulo 4, un objeto —OBJETO1— puede contener una o muchas instancias de un segundo objeto —OBJETO2— y OBJETO2 puede contener

➤ FIGURA 7-4

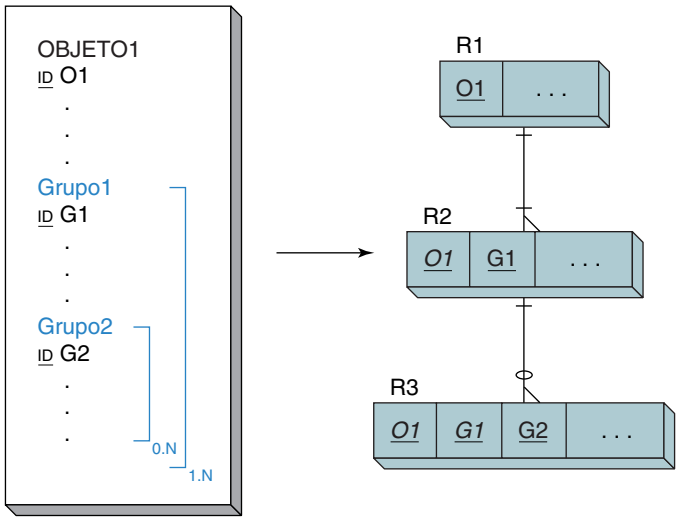
Transformación general de objetos compuestos: (a) objeto compuesto con grupos separados, y (b) objeto compuesto con grupos anidados



Restricciones de integridad referencial:

O1 en R2 debe existir en O1 en R1
O1 en R3 debe existir en O1 en R1

(a)



Restricciones de integridad referencial:

O1 en R2 debe existir en O1 en R1
(O1, G1) en R3 debe existir en (O1, G1) en R2

(b)

una o muchas instancias del primer objeto, OBJETO1. Esto conduce a los tipos de objeto que se muestran en la figura 7-5.

Todas estas relaciones implican algunas variaciones de relaciones uno a uno, uno a muchos, o muchos a muchos. Específicamente, la relación de OBJETO1 con OBJETO2

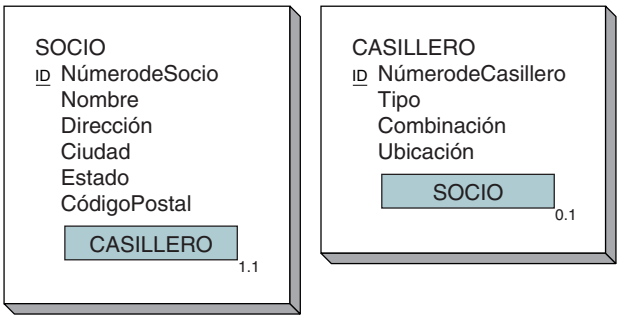
➤ FIGURA 7-5

Cuatro tipos de objetos compuestos

		Objeto1 Puede contener	
Objeto2 Puede Contener		Uno	Muchos
	Uno	**1:1**	**1:N**
	Muchos	**M:1**	**M:N**

➤ FIGURA 7-6

Ejemplo de representación relacional de objetos combinados 1:1: (a) ejemplo de objetos combinados 1:1 y (b) su representación



(a)

SOCIO (NúmerodeSocio, Nombre, Dirección, Ciudad, Estado, CódigoPostal, *NúmerodeCasillero*)

CASILLERO (NúmerodeCasillero, Tipo, Combinación, Ubicación)

Restricción de integridad referencial:

NúmerodeCasillero en SOCIO debe existir en
NúmerodeCasillero en CASILLERO

(b)

puede ser 1:1, 1:N, o N:M, siempre y cuando la relación de OBJETO2 con OBJETO1 sea 1:1, 1:M, o M:N. Para representar cualquiera de éstos, sólo necesitamos direccionar estos tres tipos de relaciones.

Representación uno a uno de objetos combinados

Considere la asignación de un CASILLERO a un SOCIO de un club deportivo. Un CASILLERO se asigna a un SOCIO y cada SOCIO tiene sólo un CASILLERO. La figura 7-6(a) muestra los diagramas de objeto. Para representar estos objetos con relaciones, definimos una relación por cada objeto, y, con relaciones de entidades de 1:1, colocamos la llave de cualquier relación en la otra relación. Así que podemos colocar la llave de SOCIO en CASILLERO o la llave de CASILLERO en SOCIO. La figura 7-6(b) muestra la ubicación de la llave de CASILLERO en SOCIO. Observe que NúmerodeCasillero está subrayado en CASILLERO porque es la llave, y está en cursivas en SOCIO porque es una llave externa en SOCIO.

En general, para una relación 1:1 entre OBJETO1 y OBJETO2 definimos una relación por cada objeto, R1 y R2. Después, colocamos la llave de cualquiera de las dos relaciones (O1 o O2) como una llave externa en la otra relación, como se ilustra en la figura 7-7.

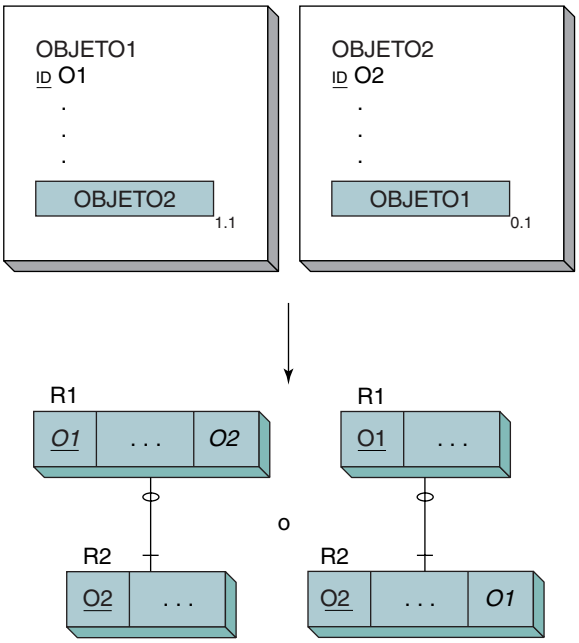
Representación de relaciones uno a muchos y muchos a uno

Ahora considere relaciones 1:N y N:1. La figura 7-8(a) muestra un ejemplo de un objeto de relación 1:N entre EQUIPO y REPARACIÓN. Un artículo de EQUIPO puede tener muchas REPARACIONes pero una REPARACIÓN se puede relacionar sólo con un artículo de EQUIPO.

Los objetos en la figura 7-8(a) se representan mediante las relaciones en la figura 7-8(b). Observe que la llave del padre (el objeto a un lado de la relación) se coloca en el hijo (el objeto en los diferentes lados de la relación).

➤ FIGURA 7-7

Transformación general de objetos combinados 1:1



Restricción de integridad referencial:

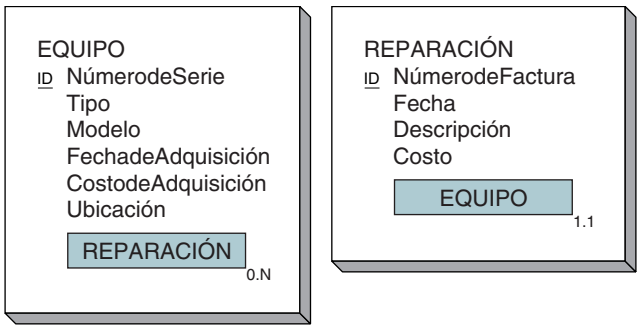
O2 en R1 debe existir en O2 en R2

Restricción de integridad referencial:

O1 en R2 debe existir en O1 en R1

➤ FIGURA 7-8

Ejemplo de representación relacional de objetos combinados de 1:N: (a) ejemplo de objetos combinados 1:N, y (b) su representación



(a)

EQUIPO (NúmerodeSerie, Tipo, Modelo, FechadeAdquisición, CostodeAdquisición, Ubicación)

REPARACIÓN (NúmerodeFactura, Fecha, Descripción, Costo, *NúmerodeSerie*)

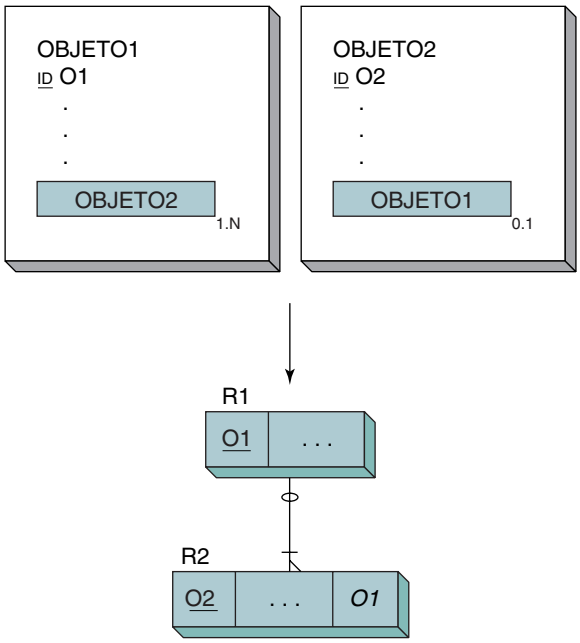
Restricción de integridad referencial:

NúmerodeSerie en REPARACIÓN debe existir en NúmerodeSerie en EQUIPO

(b)

➤ FIGURA 7-9

Transformación general de objetos combinados 1:N



La figura 7-9 muestra la transformación general de objetos combinados 1:N. El objeto OBJETO1 contiene muchos objetos OBJETO2, y el objeto OBJETO2 contiene sólo un OBJETO1. Para representar esta estructura por medio de relaciones, se representa cada objeto con una relación y se coloca la llave del padre en el hijo. De esta forma, en la figura 7-9 el atributo O1 se coloca en R2.

Si OBJETO2 contuviera muchos OBJETOs1 y OBJETO1 sólo un OBJETO2, usaríamos la misma estrategia, pero se invertirían las funciones de R1 y R2. Así que sería mejor colocar O2 en R1.

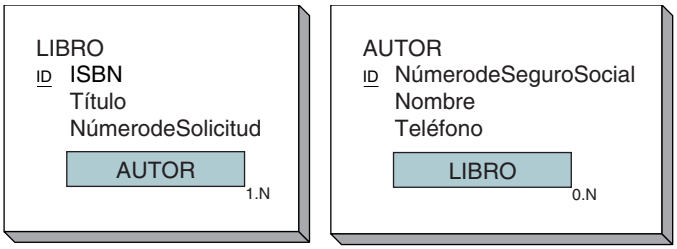
Las cardinalidades mínimas en cualquier caso están determinadas por las cardinalidades mínimas de los atributos del objeto. En la figura 7-9 el OBJETO1 requiere cuando menos un OBJETO2, pero OBJETO2 no necesariamente requiere un OBJETO1. Estas cardinalidades se muestran en el diagrama de estructura de datos con un óvalo al lado de la relación R1, y con una marca horizontal al lado de la relación R2. Estos valores de cardinalidades mínimas son simplemente ejemplos; cualquier objeto, o ambos, podría tener una cardinalidad de 0, 1, o algún otro número.

Representación de relaciones muchos a muchos

Finalmente, considere las relaciones M:N. Igual que con las relaciones de entidad M:N, definimos tres relaciones: una para cada objeto, y una tercera relación de intersección. Esta última representa la relación de los dos objetos y consta de las llaves de ambos padres. La figura 7-10(a) muestra la relación M:N entre LIBRO y AUTOR. La figura 7-10(b) describe las tres relaciones que representan estos objetos: la relación de intersección LIBRO, AUTOR y LIBRO-AUTOR-INT. Observe que LIBRO-AUTOR-INT no tiene datos llave. Ambos atributos, ISBN y NúmerodeSeguroSocial, están subrayados y en cursivas porque son llaves locales y externas.

➤ FIGURA 7-10

Representación relacional del ejemplo de objeto combinado N:M: (a) objetos LIBRO y AUTOR, y (b) su representación relacional



(a)

LIBRO (ISBN, Título, NúmerodeSolicitud)

AUTOR (NúmerodeSeguroSocial, Nombre, Teléfono)

LIBRO-AUTOR-INT (ISBN, *NúmerodeSeguroSocial*)

Restricciones de integridad referencial:

ISBN en LIBRO-AUTOR-INT debe existir en
ISBN en LIBRO

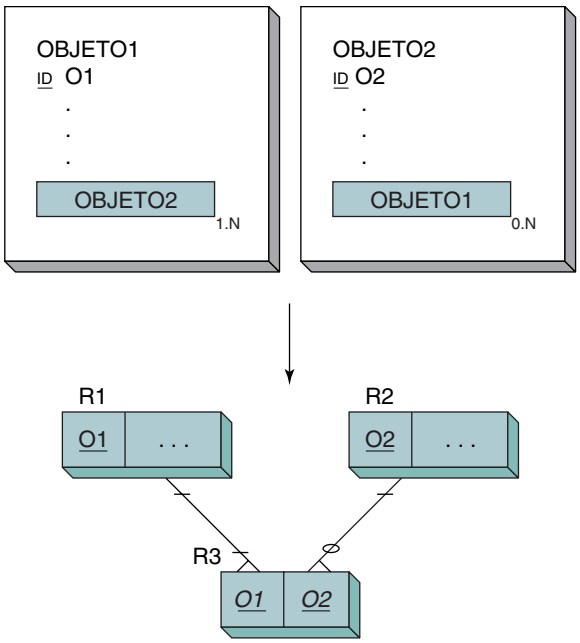
NúmerodeSeguroSocial en LIBRO-AUTOR-INT debe existir en
NúmerodeSeguroSocial en AUTOR

(b)

En general, para los dos objetos que tienen una relación M:N, definimos una relación R1 para OBJETO1, una R2 para OBJETO2 y una R3 para la relación de intersección. El esquema general se muestra en la figura 7-11. Observe que los atributos de R3 son sólo O1 y O2. Para los objetos compuestos M:N, R3 nunca tendrá datos sin llave. La importancia de este enunciado será obvia cuando comparemos relaciones combinadas M:N con relaciones de asociación.

➤ FIGURA 7-11

Transformación general de objetos combinados M:N en relaciones



Restricciones de integridad referencial:

O1 en R3 debe existir en O1 en R1
O2 en R3 debe existir en O2 en R2

Cuando se considera la cardinalidad mínima siempre se requieren los padres de la relación de intersección. Las cardinalidades mínimas de las relaciones en la relación de intersección se determinan mediante las cardinalidades mínimas de los vínculos de los objetos. En la figura 7-11, por ejemplo, un renglón en R1 requiere un renglón en R3 porque la cardinalidad mínima de OBJETO2 en OBJETO1 es 1. De manera similar, un renglón en R2 no requiere un renglón en R3 porque la cardinalidad mínima de OBJETO1 en OBJETO2 es 0.

Objetos híbridos

Los objetos híbridos pueden transformarse en diseños relacionales usando una combinación de las técnicas para los objetos compuestos y combinados. La figura 7-12(a) muestra PEDIDO-VENTAS, un objeto híbrido y objetos relacionados. Para representar este objeto por medio de relaciones establecemos una para el mismo objeto y otra para cada uno de los objetos contenidos en CLIENTE y VENDEDOR. Entonces, al igual que con un objeto compuesto, establecemos una relación para el grupo de valor múltiple, que es ArtículodeLínea. Puesto que este grupo contiene otro objeto, ARTÍCULO, también establecemos una relación para ARTÍCULO. Todas las relaciones uno a muchos se representan mediante la colocación de la llave de la relación padre en la relación hijo, como se muestra en la figura 7-12(b).

El ejemplo en la figura 7-12 es engañosamente simple. Como mencionamos en el capítulo 4, en realidad hay cuatro instancias de objetos híbridos, los cuales se resumen en la figura 7-13.

Los casos 3 y 4 son más comunes que los 1 y 2, así que los consideramos primero. En la figura 7-14 OBJETO1 muestra dos grupos: el Grupo1 ilustra el caso 3 y el Grupo2, el caso 4.

Grupo1 tiene una cardinalidad máxima de N, lo que significa que puede haber muchas instancias del Grupo1 en el OBJETO1. Además, puesto que OBJETO2 está marcado como ID único, quiere decir que un OBJETO2 en particular puede aparecer sólo en una de las instancias del GRUPO1 dentro de un OBJETO1. Así, OBJETO2 actúa como un identificador para GRUPO1 dentro de OBJETO1.

(Cabe aclarar que PEDIDO-VENTAS en la figura 7-12 ilustra este caso. ARTÍCULO es un identificador de ArtículodeLínea, así que un ARTÍCULO determinado puede aparecer en ArtículodeLínea sólo en un PEDIDO; pero un ARTÍCULO puede aparecer en muchos PEDIDOs.)

Considere la representación relacional de Grupo1 en la figura 7-14. Se crea una relación, R1, para OBJETO1, y una relación, R2, para OBJETO2. Además, se crea una tercera relación, R-G1, para Grupo1. La relación entre R1 y R-G1 es 1:N, así que colocamos la llave de R1 (que es O1) en R-G1; la relación entre R2 y R-G1 también es 1:N, así que colocamos la llave de R2 (la cual es O2) en R-G1. Puesto que un OBJETO2 sólo puede aparecer una vez con un valor particular de OBJETO1, el compuesto (O1, O2) es único para R-G1 y puede convertirse en la llave de esa relación.

Ahora consideremos el Grupo2. OBJETO3 no identifica a Grupo2, por lo que OBJETO3 puede aparecer en la mayoría de las instancias de Grupo2 en el mismo OBJETO1. (En la figura 7-12 PEDIDO-VENTAS estaría en igual circunstancia si ARTÍCULO no fuera ID único en ArtículodeLínea. Esto significa que un ARTÍCULO podría aparecer muchas veces en el mismo PEDIDO.) Debido a que OBJETO3 no es el identificador del Grupo2, suponemos que algún otro atributo, G2, es el identificador.

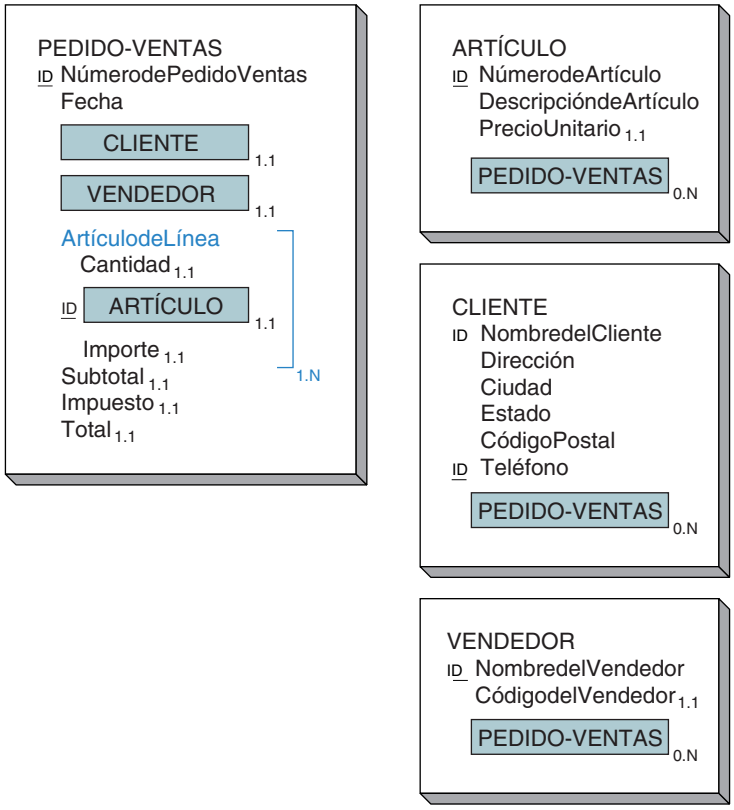
En la figura 7-14 creamos una relación R3 para OBJETO3 y otra, R-G2, para Grupo2. La relación entre R1 y R-G2 es 1:N, así que se coloca la llave de R1 (que es O1) en R-G2. La relación entre R3 y R-G2 es también 1:N, por lo cual se coloca la llave de R3 (que es O3) en R-G2.

Sin embargo, a diferencia de Grupo1, ahora(O1, O3) no puede ser la llave de R-G2 porque un O3 puede estar pareado muchas veces con un O1 determinado. Así, el compuesto (O1, O3) no es único para R-G2, por lo que la llave de R-G2 debe ser (O1-G2).

El caso 1 es similar al 3, excepto en la restricción de que un OBJETO2 se puede relacionar únicamente con un OBJETO1. Las relaciones en la figura 7-14 todavía fun-

➤ FIGURA 7-12

Representación relacional del ejemplo de objeto híbrido: (a) ejemplo de objeto híbrido, y (b) representación relacional de PEDIDO-VENTAS y objetos relacionados



(a)

PEDIDO-VENTAS (NúmerodePedidoVentas, Fecha, Subtotal, Impuesto, Total, *Teléfono, NombredelVendedor*)

CLIENTE (NombredelCliente, Dirección, Ciudad, Estado, CódigoPostal, Teléfono)

VENDEDOR (NombredelVendedor, CódigodelVendedor)

ARTÍCULO-LÍNEA (*NúmerodePedidodeVentas*, *NúmerodeArtículo*, Cantidad, Importe)

ARTÍCULO (NúmerodeArtículo, DescripciónArtículo, PrecioUnitario).

Restricciones de integridad referencial:

NombredelVendedor en PEDIDO-VENTAS debe existir en NombredelVendedor en VENDEDOR

Teléfono en PEDIDO-VENTAS debe existir en Teléfono en CLIENTE

NúmerodePedidoVentas en ARTÍCULO-LÍNEA debe existir en NúmerodePedidoVentas en PEDIDO-VENTAS

NúmerodeArtículo en ARTÍCULO-LÍNEA debe existir en NúmerodeArtículo en ARTÍCULO

(b)

cionarán, pero debemos agregar la llave de R1 (que es O1) a R2 y establecer la restricción de que (O1, O2) de R-G1 debe ser igual a (O1, O2) de R2.

El caso 2 es similar al caso 4, excepto por la restricción de que un OBJETO 3 puede estar relacionado sólo con un OBJETO1. Nuevamente, las relaciones en la figura 7-14 fun-

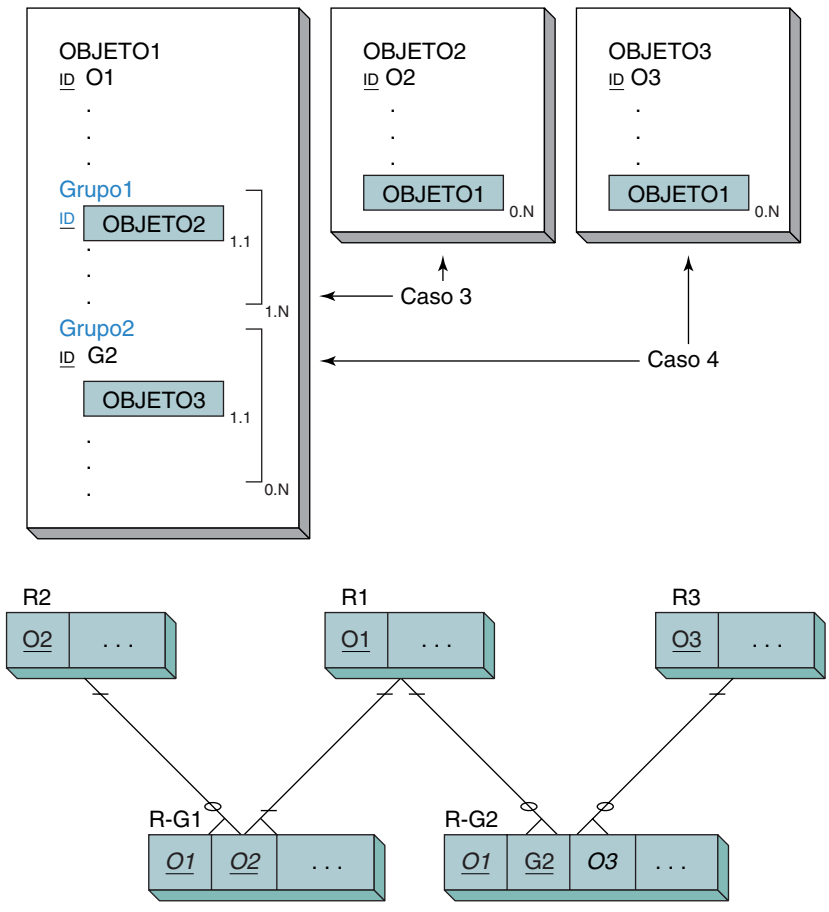
➤ FIGURA 7-13

Cuatro casos de cardinalidad de objetos híbridos

Caso	Descripción	Ejemplo
1	OBJETO2 se relaciona con una instancia de OBJETO1 y sólo aparece en una instancia de grupo dentro de ese objeto.	ARTÍCULO se relaciona con un PEDIDO y sólo puede aparecer en un ArtículodeLínea de ese PEDIDO.
2	OBJETO2 se relaciona con una instancia de OBJETO1 y es posible que aparezca en muchas instancias de grupo dentro de ese objeto.	ARTÍCULO se relaciona con un PEDIDO y puede aparecer en muchos ArtículosdeLínea de ese PEDIDO.
3	Es posible que OBJETO2 se relacione con muchas instancias de OBJETO1 y sólo aparezca en una instancia de grupo dentro de cada objeto.	ARTÍCULO se relaciona con muchos PEDIDOs y sólo puede aparecer en un ArtículodeLínea de ese PEDIDO.
4	Es posible que OBJETO2 se relacione con muchas instancias de OBJETO1 y quizá aparezca en muchas instancias dentro de esos objetos.	ARTÍCULO se relaciona con muchos PEDIDOs y puede aparecer en muchos ArtículosdeLínea de ese PEDIDO.

➤ FIGURA 7-14

Transformación general de objetos híbridos en relaciones



Restricciones de integridad referencial:

O1 en R-G1 debe existir en O1 en R1
O2 en R-G1 debe existir en O2 en R2
O1 en R-G2 debe existir en O1 en R1
O3 en R-G2 debe existir en O3 en R3

cionarán, pero debemos agregar la llave de R1 (que es O1) a R3, y establecer la restricción (O1, O3) de que R-G2 es un subconjunto de (O1, O3) en R3 (véanse las preguntas 7.7 y 7.8).

Objetos de asociación

Un objeto de asociación es aquel que une dos objetos. Es un caso especial de objetos combinados que ocurre con mayor frecuencia en las situaciones de asignación. La figura 7-15(a) muestra un objeto VUELO que liga a un AVIÓN con un PILOTO.

Para representar los objetos de asociación definimos una relación para cada uno de los tres objetos, y después representamos las relaciones entre los objetos empleando una de las estrategias que se usaron con los objetos combinados. Por ejemplo, en la figura 7-15(b) se define una relación para AVIÓN, una para PILOTO y otra para VUELO. Las relaciones entre VUELO y AVIÓN, y entre VUELO y PILOTO son 1:N, así que ponemos las llaves del padre en los hijos. En este caso, colocamos la llave de AVIÓN y la de PILOTO en VUELO.

VUELO contiene una llave propia. Aunque tiene llaves externas, sólo son atributos y no parte de la llave de VUELO. Pero éste no siempre es el caso. Si VUELO no tuviera una llave propia, ésta sería la combinación de las llaves externas de los objetos asociados. En ese caso la combinación sería {NúmerodeAvión, NúmerodePiloto, Fecha}.

En general, cuando se transforman en relaciones las estructuras del objeto de asociación, definimos una relación por cada uno de los objetos que participan en ésta. En

➤ FIGURA 7-15

Representación relacional de ejemplo de objeto de asociación: (a) objeto de asociación VUELO y objetos relacionados, y (b) representación relacional de los objetos AVIÓN, PILOTO y VUELO

(a)

AVIÓN (NúmerodeAvión, Tipo, Horas)

PILOTO (NúmerodePiloto, Nombre, Teléfono)

VUELO (NúmerodeVuelo, Fecha, CiudaddeOrigen, CiudaddeDestino,
NúmerodeAvión, NúmerodePiloto)

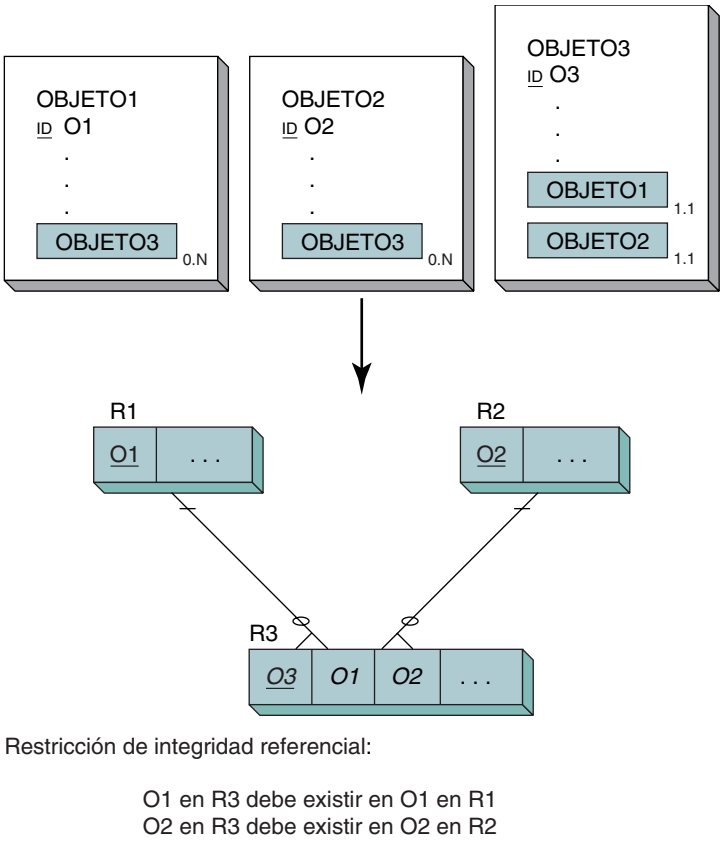
Restricción de integridad referencial:

NúmerodeAvión en VUELO debe existir en NúmerodeAvión en AVIÓN
NúmerodePiloto en VUELO debe existir en NúmerodePiloto en PILOTO

(b)

➤ FIGURA 7-16

Transformación general de los objetos de asociación dentro de las relaciones



la figura 7-16, OBJETO3 asocia a OBJETO1 y OBJETO2. En este caso, definiremos R1, R2 y R3 como se muestra. Las llaves de cada una de las relaciones padre, O1 y O2, aparecen como atributos de la llave externa en R3, la relación que representa el objeto de asociación. Si el objeto de asociación no tiene un atributo único que lo identifique, la combinación de los atributos de R1 y R2 será usado para crear un identificador único.

Observe la diferencia entre la relación de asociación de la figura 7-16 y la de intersección en la figura 7-11. La diferencia principal es que la tabla de asociación lleva datos que representan algún aspecto de la combinación de los objetos. La relación de intersección no incluye datos, sólo existe para especificar qué objetos tienen una relación entre sí.

Objetos padre y objetos subtipo

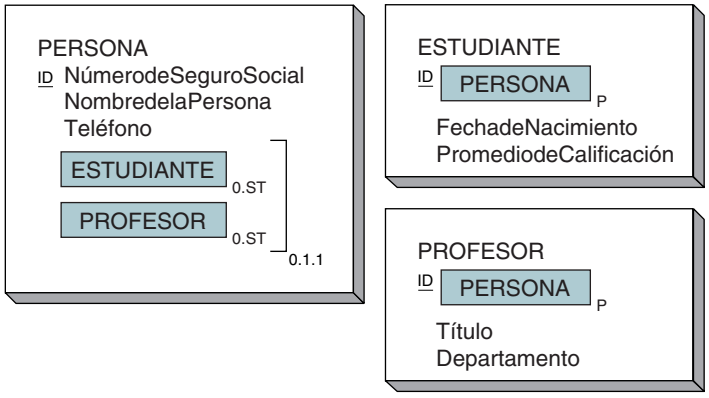
Los objetos padre (también llamados *supertipo*) y los objetos subtipo se representan de manera similar a las entidades padre y subtipo. Definimos una relación por el objeto padre y uno por cada objeto de subtipo. La llave de cada una de estas relaciones es la llave del padre.

La figura 7-17(a) muestra un objeto padre, PERSONA, que incluye dos subtipos mutuamente excluyentes, ESTUDIANTE y PROFESOR. La figura 7-17(b) muestra una representación relacional de estos tres objetos. Cada uno está representado mediante una tabla, y la llave de todas las tablas es la misma.

Sin embargo, las relaciones en la figura 7-17(b) tienen un problema: el programa de aplicación todavía necesita buscar en ambas tablas, ESTUDIANTE y PROFESOR, para determinar el tipo de PERSONA. Si se encuentra una entrada en ESTUDIANTE la persona es un estudiante; pero si se encuentra en PROFESOR, es un profesor. Éste es un

➤ FIGURA 7-17

Representación del ejemplo padre y subtipos: (a) padre PERSONA y subtipos ESTUDIANTE y PROFESOR, (b) representación relacional de padre y subtipos, y (c) representaciones alternativas de la relación padre



(a)

PERSONA (NúmerodeSeguroSocial, NombredelaPersona, Teléfono)

ESTUDIANTE (NúmerodeSeguroSocial, FechadeNacimiento, PromediodeCalificación)

PROFESOR (NúmerodeSeguroSocial, Título, Departamento)

Restricciones de integridad referencial:

NúmerodeSeguroSocial en ESTUDIANTE debe existir en NúmerodeSeguroSocial en PERSONA

NúmerodeSeguroSocial en PROFESOR debe existir en NúmerodeSeguroSocial en PERSONA

(b)

PERSONA1 (NúmerodeSeguroSocial, NombredelaPersona, Teléfono, TipodePersona)

PERSONA2 (NúmerodeSeguroSocial, NombredelaPersona, Teléfono, TipodeEstudiante, TipodeProfesor)

(c)

modo indirecto y posiblemente lento para determinar el tipo de persona y si, como puede suceder, PERSONA no es de ninguno de los tipos, ambas tablas habrán sido consultadas sin razón alguna. Debido a este problema, a veces se coloca un tipo de atributo indicador en la tabla padre.

La figura 7-17(c) muestra dos variaciones de un indicador de tipo. En la primera variación, la relación PERSONA1, el tipo de objeto se almacena en el atributo TipodePersona. Los valores posibles de este atributo son "Ninguno", "ESTUDIANTE", o "PROFESOR". La aplicación se obtendría del valor de este atributo y, por lo tanto, determinaría si un subtipo existe o no, y si existe, de qué tipo es.

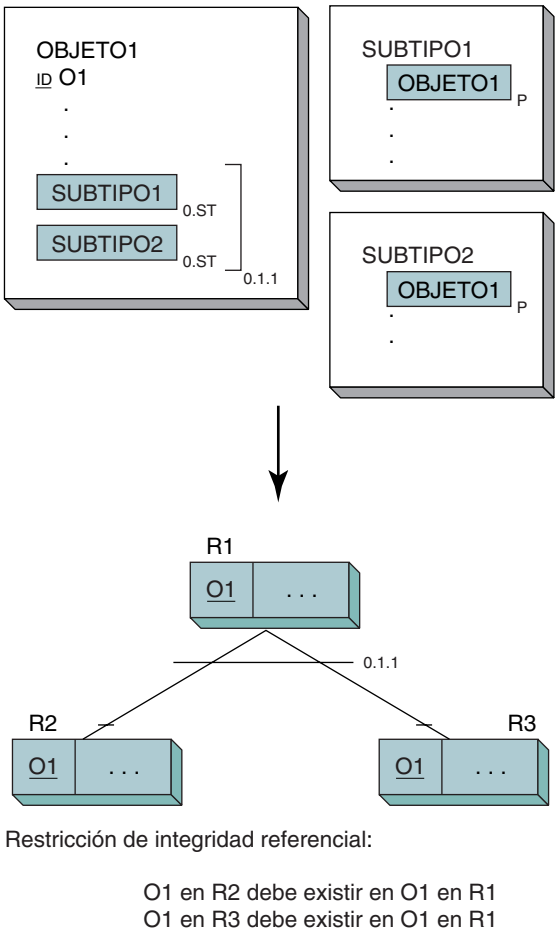
Una segunda posibilidad se aprecia en la relación PERSONA2, a la cual se han agregado dos atributos: uno para TipodeEstudiante y otro para TipodePersona. Cada uno de los atributos es una variable booleana; los valores permitidos son verdaderos o falsos. Observe que, como sucede en este caso, si una persona sólo puede ser de un tipo, y si entonces uno de estos valores es verdadero, el otro debe ser falso.

En general, los diseños del tipo PERSONA1 son mejores cuando los subtipos son mutuamente excluyentes. Los diseños del tipo PERSONA2 son mejores cuando los subtipos no son excluyentes.

Un esquema general de la representación de los subtipos se muestra en la figura 7-18. Se crea una relación para padre y otra para cada uno de los subtipos. La llave de todas las relaciones es el identificador del padre. Todas las relaciones entre el padre y el subtipo son 1:1. Observe la barra horizontal en las líneas de la relación y la presencia

➤ FIGURA 7-18

Transformación general de los objetos padre-subtipo en relaciones



de la cardinalidad del grupo de subtipo. El valor mostrado, 0.1.1, significa que no se requirió subtipo, pero si se presenta, cuando mucho se permite uno de los subtipos.

(Recuerde que en general el formato de la cardinalidad del grupo es **r.m.n.,** donde ***r*** es un valor booleano verdadero o falso, dependiendo de si se requiere o no el grupo subtipo; ***m*** es el número mínimo de subtipos que debe tener un valor dentro del grupo, y ***n*** es el número máximo de subtipos que puede tener un valor dentro del grupo. Por lo tanto, en un grupo de cinco subtipos, la cardinalidad de 1.2.4 indica que se requiere el grupo de subtipos, que cuando menos dos subtipos deben tener un valor y que un máximo de cuatro subtipos pueden tener un valor.)

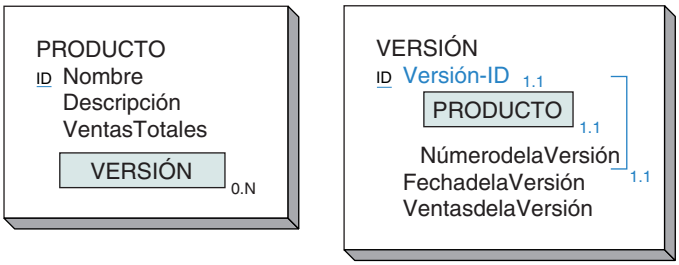
Objetos arquetipo-versión

Los objetos arquetipo-versión son objetos combinados que modelan varias repeticiones, versiones, o instancias de un objeto básico. Los objetos en la figura 7-19(a) modelan productos de software para los cuales hay varias versiones. Ejemplos de estos productos son Microsoft Internet Explorer o Netscape Navigator. Ejemplos de versiones son Access 2000 y Access 2002.

La representación relacional de PRODUCTO y VERSIÓN se muestra en la figura 7-19(b). Se crea una relación para PRODUCTO, y otra para VERSIÓN. La llave de VERSIÓN es la combinación de la llave de PRODUCTO y la llave local (NúmerodelaVersión) de VERSIÓN.

FIGURA 7-19

Representación relacional del ejemplo de objetos arquetipo-versión: (a) objeto arquetipo PRODUCTO y objeto versión VERSIÓN, y (b) representación relacional de PRODUCTO y VERSIÓN



(a)

PRODUCTO (Nombre, Descripción, VentasTotales)

VERSIÓN (*Nombre*, NúmerodelaVersión, FechadelaVersión, VentasdelaVersión)

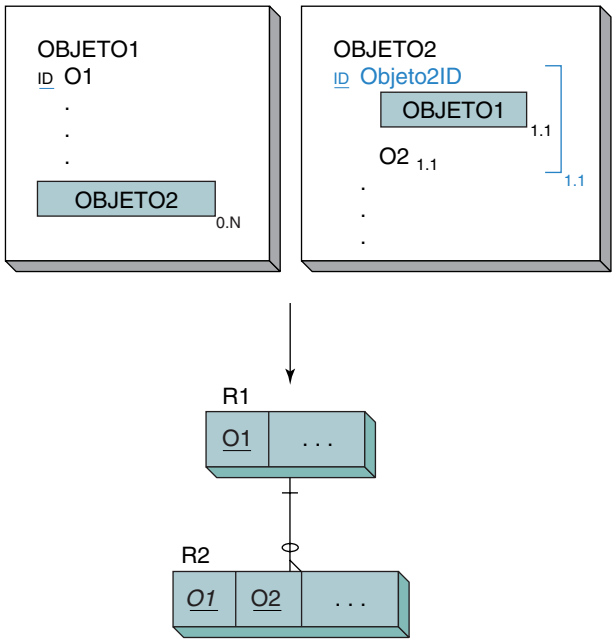
Restricción de integridad referencial:

Nombre en VERSIÓN debe existir en Nombre en PRODUCTO

(b)

FIGURA 7-20

Transformación general de los objetos de arquetipo-versión y versión VERSIÓN



Restricción de integridad referencial:

O1 en R2 debe existir en O1 en R2

En la figura 7-20 se muestra la transformación general de los objetos arquetipo-versión. El atributo O1 de R2 es tanto una llave local como una externa, pero O2 sólo es una llave local.

OBJETOS DE MUESTRA

Para reforzar los conceptos que hemos presentado, ahora consideraremos varios ejemplos de objetos tomados de negocios reales, y los presentaremos por orden de complejidad. Modelamos el objeto implícito y lo representamos en relaciones usando los métodos descritos en este capítulo. La especificación de restricciones de integridad relacional se deja para las preguntas 7.20, 7.21 y 7.22.

Forma de suscripción

La figura 7-21(a) muestra un formato de suscripción a una revista. Cuando menos dos estructuras de objeto podrían representarla. Si los editores de *Fine Woodworking* consideran a un suscriptor como un atributo de una suscripción, ésta podría ser un objeto simple representado como una relación simple, como se muestra en la figura 7-21(b).

Si esta empresa sólo cuenta con una publicación y no tiene planes para editar otras, funcionará el diseño de la figura 7-21(b); pero si tiene varias publicaciones y un cliente puede suscribirse a más de una, este diseño duplicará los datos del cliente para cada publicación. Esto no sólo sería un desperdicio de espacio en el archivo del editor, sino que lo exasperaría porque, por ejemplo, tendría que informar todos los cambios de dirección para cada publicación.

Si la editorial tiene varias publicaciones, o planea tenerlas, un diseño más apropiado sería modelar al suscriptor como un objeto por separado, como se muestra en la figura 7-21(c). CLIENTE es un objeto combinado 1:N y se representa mediante las relaciones en esta figura.

Descripción del producto

La figura 7-22(a) muestra la descripción de un producto popular de artículos empacados. Mientras que la figura 7-21(a) ilustra una forma genérica sin datos, la 7-22(a) ofrece un ejemplo de un reporte específico con datos acerca de un producto de cereales. Los reportes para todos los cereales de Kellogg's usan este formato.

La figura 7-22(b) muestra un objeto compuesto que podría estar implícito en este reporte. Decimos *podría* porque hay muchas formas diferentes en las que este objeto puede ser representado. También, una investigación adicional revelaría otros objetos que no son evidentes en este reporte. Por ejemplo, la ración diaria de consumo que recomienda el Ministerio de Agricultura de Estados Unidos puede ser un objeto semántico por derecho propio.

Con el fin de ilustrar lo anterior, hacemos diferentes suposiciones acerca de los grupos Nutriente y RaciónDiariaRecomendada. El objeto PRODUCTO-CEREAL supone que se requiere cada elemento del grupo de nutriente —calorías, proteínas, carbohidratos, grasa, colesterol, sodio, y potasio— en cada instancia de este objeto. No hacemos esa suposición para el grupo RaciónDiariaRecomendada porque sólo debe existir una instancia de este grupo.

El reporte de la figura 7-22(a) tiene muchas interpretaciones y se podría modelar de diferentes maneras. En un proyecto de desarrollo real sería importante obtener tantos reportes como sea posible acerca de los cereales, los ingredientes y la información nutricional. Posiblemente los otros documentos darían la estructura adicional para este objeto semántico.

La figura 7-22(c) muestra la representación relacional para el objeto PRODUCTO-CEREAL. La cardinalidad mínima de 7 se muestra colocando el número 7 junto a la marca horizontal en la línea de relación. Las llaves externas se colocaron como se describió anteriormente en el caso de los objetos compuestos.

Citatorio de amonestación vial

La figura 7-23(a) muestra un ejemplo de una forma de citatorio de amonestación vial que se usa en el estado de Washington. El diseñador de esta forma nos ha dado pistas importantes de los objetos implícitos. Observe que partes de la forma se distinguen porque tienen las esquinas redondeadas, lo cual indica las diferentes secciones que pertenecen a objetos diferentes. También, algunos grupos de atributos tienen nombres, lo que representa la necesidad de grupos de atributos.

La figura 7-23(b) es una manera de ilustrar los objetos implícitos del citatorio de amonestación vial. Aunque no podemos estar seguros sólo a partir de una forma, hay ciertos indicios que nos llevan a creer que el conductor, el vehículo y el oficial son ob-

➤ FIGURA 7-21

Representación alternativa de suscripción: (a) forma de pedido de suscripción, (b) suscripción modelada como un objeto, y (c) suscripción modelada como dos objetos

Fine Wood Working

Suscripción

☐ 1 año (6 ejemplares) por sólo $18; 20% menos que en el puesto de periódicos ($21 por año en el extranjero)

☐ 2 años (12 ejemplares) por sólo $34; ahorre el 24% ($40 por dos años en el extranjero)

Nombre _____

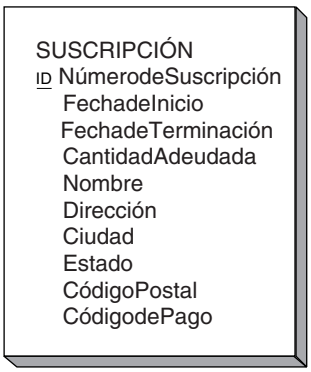
Dirección _____

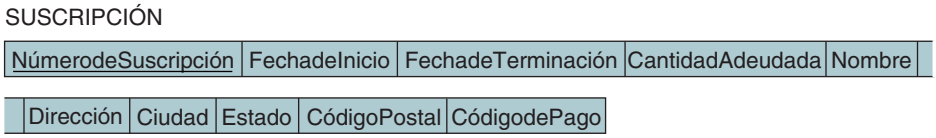
Ciudad _____ Estado _____ Código postal _____

☐ Mi pago está incluido. ☐ Pagaré al recibirla.

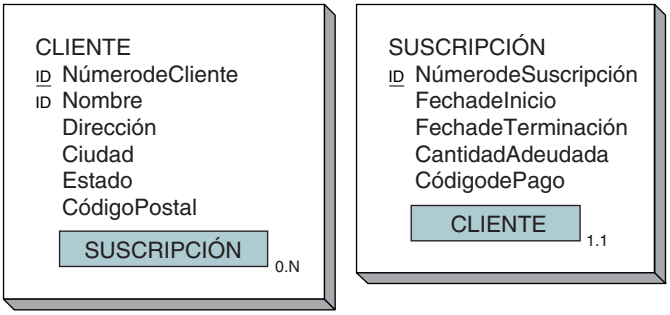
Iniciar mi suscripción con ☐ *número actual* ☐ *número siguiente*

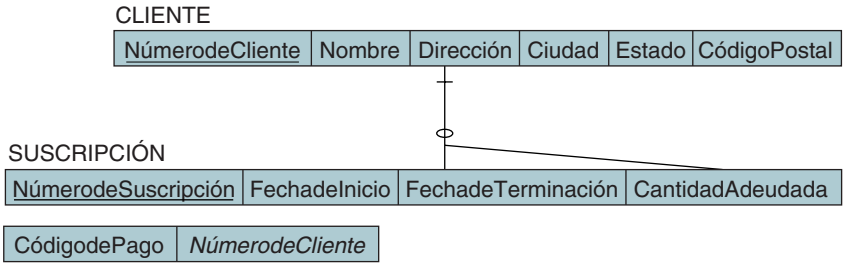
(a)





(b)





(c)

➤ FIGURA 7-22

Representación de producto cereal: (a) reporte de producto cereal, (b) diagrama de objetos de PRODUCTO-CEREAL, y (c) representación relacional de PRODUCTO-CEREAL

Kellogg's®
RICE KRISPIES®

INFORMACIÓN NUTRICIONAL
TAMAÑO DE RACIÓN: 1 ONZA (28.4 g, APROXIMADAMENTE 1 TAZA)
RACIONES POR PAQUETE: 13

	CEREAL	CON 1/2 TAZA VITAMINAS A Y D LECHE DESCREMADA
CALORÍAS	110	150 *
PROTEÍNAS	2 g	6 g
CARBOHIDRATOS	25 g	31 g
GRASAS	0 g	0 g *
COLESTEROL	0 mg	0 mg *
SODIO	290 mg	350 mg
POTASIO	35 mg	240 mg

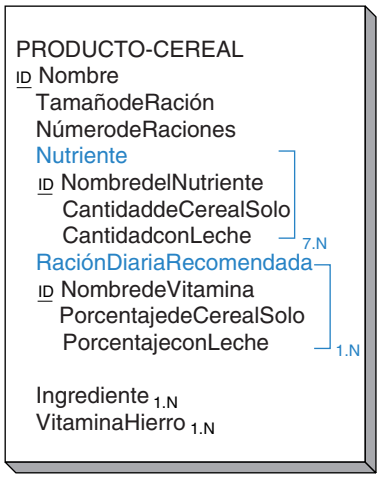
PORCENTAJE DE CANTIDADES DIARIAS QUE RECOMIENDA EL MINISTERIO DE AGRICULTURA DE ESTADOS UNIDOS DE AMÉRICA

PROTEÍNAS	2	10
VITAMINA A	25	30
VITAMINA C	25	25
TIAMINA	35	40
RIVOFLANIVA	35	45
NIACINA	35	35
CALCIO	**	15
HIERRO	10	10
VITAMINA D	10	25
VITAMINA B_6	35	35
ÁCIDO FÓLICO	35	35
FÓSFORO	4	15
MAGNESIO	2	6
ZINC	2	6
COBRE	2	4

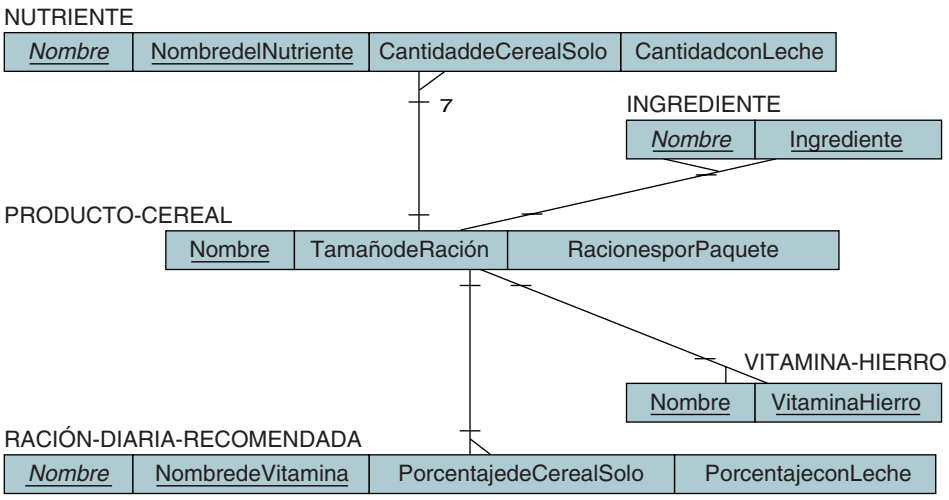
* LA LECHE ENTERA PROPORCIONA 30 CALORÍAS MÁS, 4 g DE GRASA Y 15 mg DE COLESTEROL.
** CONTIENE MENOS DEL 2% DE LA RACIÓN DIARIA RECOMENDADA.

INGREDIENTES: ARROZ, AZÚCAR, SAL, JARABE DE MAÍZ, SABORIZANTE DE MALTA.
VITAMINAS E HIERRO: VITAMINA C (ASCORBATO DE SODIO Y ÁCIDO ASCÓRBICO), NIACINAMIDA, HIERRO, VITAMINA B_6 (HIDROCLORHIDRATO DE PIRIDOXINA), VITAMINA A (PALMITATO), VITAMINA B_2 (RIVOFLAVINA), VITAMINA B_1 (HIDROCLORHIDRATO DE TIAMINA), ÁCIDO FÓLICO Y VITAMINA D. PARA MANTENER FRESCO ESTE CEREAL SE LE AGREGÓ BHT AL EMPAQUE.

(a)



(b)



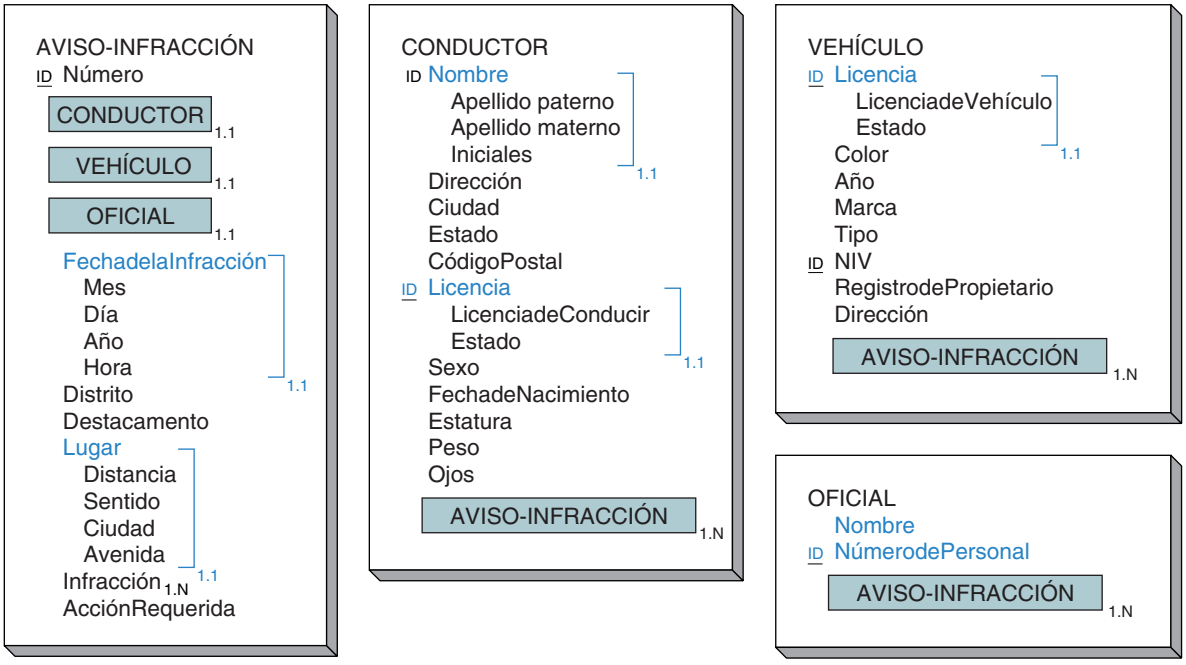
(c)

jetos independientes. Primero, los datos de cada uno están por separado en una sección del formato. Pero lo más importante: cada sección tiene campos que sin duda identifican atributos de algo aparte de AVISO-INFRACCIÓN. Por ejemplo {LicenciadeConducir, Estado} únicamente identifica al conductor; LicenciadeVehículo, Estado y NIV (Número de Identificación del Vehículo) identifican los vehículos registrados, y el

➤ FIGURA 7-23

Representación de un aviso de infracción: (a) forma de ejemplo, (b) diagrama de objeto de AVISO-INFRACCIÓN, y (c) Representación relacional de AVISO-INFRACCIÓN

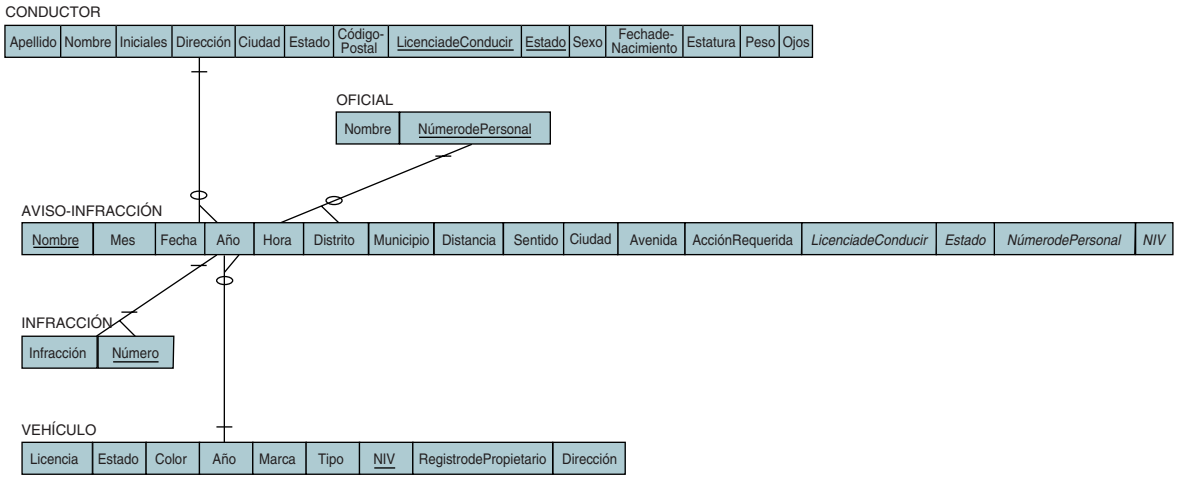
(a)



(b)

FIGURA 7-23

(Continuación)



(c)

NúmerodePersonal identifica a un oficial. Obviamente estos campos llave son determinantes, por lo tanto se definieron objetos para cada uno. En la figura 7-23(c) se ilustra la relación de estos diagramas.

RESUMEN

La transformación de objetos semánticos en relaciones depende del tipo de objeto. Los simples se representan mediante una relación individual. Los atributos que no son objetos se manejan como atributos de la relación.

Los objetos compuestos requieren dos o más relaciones para su presentación. Una relación contiene atributos de valor único del objeto. Se construye otra relación para cada atributo simple multivaluado o atributo de grupo. La llave de las relaciones que representan los atributos multivaluados siempre es una llave compuesta que contiene la llave del objeto, más un identificador del grupo compuesto dentro de ese objeto.

Se requieren cuando menos dos relaciones para representar un objeto combinado. Cada relación tiene su propia llave distinta. Hay cuatro tipos diferentes de objetos combinados: uno a uno, uno a muchos, muchos a uno y muchos a muchos, que se representan insertando llaves externas. Para las relaciones uno a uno, la llave de cada tabla se coloca en la otra tabla, y para las relaciones uno a muchos y muchos a uno la llave del padre se coloca en la relación hijo. Por último, para las relaciones muchos a muchos se crea una tabla de intersección que contenga las llaves de ambas relaciones.

Los objetos híbridos se representan creando una tabla para el atributo del grupo multivaluado del objeto compuesto, y colocando dentro la llave de la relación que representa al objeto no compuesto. Los cuatro casos de híbridos se enumeran en la figura 7-13.

Los objetos de asociación requieren cuando menos tres relaciones para su representación, una para cada objeto involucrado. Cada relación tiene su propia llave y la relación que representa a los objetos de asociación contiene, como llaves externas, las llaves de los otros dos objetos.

Los objetos padre y subtipo se representan creando una relación para el padre y una para cada subtipo. La llave de todas las relaciones normalmente es la misma. Algunas veces se coloca un atributo identificador en el padre para indicar el tipo del objeto.

Para los objetos arquetipo-versión se crea una relación para el objeto arquetipo y una segunda para la versión. La llave de la relación de la versión siempre contiene la llave del arquetipo.

➤ PREGUNTAS DEL GRUPO I

7.1 Proporcione un ejemplo de un objeto simple, diferente al que se mencionó en este texto. Muestre cómo se representa ese objeto por medio de una relación.

7.2 Exponga un ejemplo de objeto compuesto, diferente al de este texto, y diga cómo se representa por medio de relaciones.

7.3 Mencione un ejemplo de un objeto combinado 1:1, diferente al de este texto. Muestre dos formas de representarlo por medio de relaciones.

7.4 Señale un ejemplo de un objeto combinado 1:N, diferente al de este texto, y muestre cómo representarlo a través de relaciones.

7.5 Dé un ejemplo referente a objeto combinado de M:1, diferente al que se mencionó, y muestre cómo representarlo por medio de relaciones.

7.6 Provea un ejemplo de un objeto combinado M:N, diferente al de este texto. Muestre cómo representarlo por medio de relaciones.

7.7 Proporcione un ejemplo del caso 1 (véase la figura 7-13) de objeto híbrido. Muestre cómo representarlo por medio de relaciones.

7.8 Mencione un ejemplo del caso 2 (véase la figura 7-13) sobre objeto híbrido. Muestre cómo representarlo por medio de relaciones.

7.9 Dé un ejemplo sobre objetos de asociación y objetos relacionados, diferente a los de este texto. Muestre cómo representarlo por medio de relaciones. Suponga que el objeto de asociación tiene su propio identificador.

7.10 Haga lo mismo que en la pregunta 7.9, pero suponga que el objeto de asociación no tiene un identificador propio.

7.11 Proporcione un ejemplo sobre un objeto padre, con dos subtipos excluyentes cuando menos, y muestre cómo representarlos mediante relaciones. Use un atributo indicador de tipo.

7.12 Mencione un ejemplo de objeto padre con dos subtipos no excluyentes, cuando menos. Muestre cómo representar ambos objetos por medio de relaciones. Use un atributo indicador de tipo.

7.13 Encuentre un ejemplo de un formato que se emplee en la universidad a la que usted asiste, el cual esté modelado apropiadamente con un objeto simple. Muestre cómo representarlo por medio de una relación.

7.14 Localice un ejemplo sobre un formato de la universidad que esté modelado apropiadamente con un objeto compuesto. Muestre cómo representarlo por medio de relaciones.

7.15 Ubique un ejemplo sobre un formato de la universidad que esté modelado apropiadamente con uno de los tipos de un objeto combinado. Muestre cómo representarlos por medio de relaciones.

7.16 Descubra un ejemplo de un formato de la universidad que esté modelado apropiadamente con un objeto híbrido. Clasifique el objeto de acuerdo con la figura 7-13 y muestre cómo se representan estos objetos por medio de relaciones.

7.17 Localice un ejemplo de un formato de la universidad que esté modelado apropiadamente con objetos de asociación y relacionados. Muestre cómo representar dichos objetos por medio de relaciones.

7.18 Encuentre un ejemplo de un formato de la universidad que esté modelado apropiadamente con objetos padre-subtipo. Muestre cómo representarlos mediante relaciones.

7.19 Ubique un ejemplo de un formato de la universidad que esté modelado adecuadamente con objetos arquetipo-versión. Muestre cómo representar estos objetos por medio de relaciones.

7.20 ¿Qué restricciones de integridad referencial, si las hay, deberían especificarse para los diseños en la figura 7-21(b) y (c)?

7.21 ¿Qué restricciones de integridad referencial, si las hay, deberían especificarse para el diseño de la figura 7-22(c)?

7.22 ¿Qué restricciones de integridad referencial, si las hay, deberían especificarse para el diseño de la figura 7-23(c)?

7.23 Suponga que el objeto O1 tiene una relación 1:N con el objeto O2, y que el O1 tiene una segunda relación 1:N con el objeto O3. Suponga además que se requiere O2 en O1, pero O3 es opcional en O1. ¿Existe alguna diferencia entre la restricción de integridad referencial para la relación de O1 y O2 y la restricción de integridad referencial para la relación de O1 y O3? Si es así, ¿cuál es?

➤ PREGUNTAS DEL GRUPO II

7.24 Con base en la figura 7-13, proporcione un ejemplo diferente para cada uno de los cuatro casos en la columna de la derecha. Muestre cómo cada uno estaría representado con relaciones.

7.25 Modifique la figura 7-22(b) y (c) para agregar los reportes que se muestran en la figura 7-24.

➤ FIGURA 7-24

Reportes para la pregunta 7.25

REPORTE DEL MINISTERIO DE AGRICULTURA #6272
Fecha: 06/30/2001
Emisor: Compañía Kellogg´s
Título del reporte: resumen de productos por ingrediente

Maíz	Corn Flakes
	Krispix
	Nutrigrain (maíz)
Jarabe de maíz	Rice Krispies
	Frosted Flakes
	Sugar Pops
Malta	Rice Krispies
	Sugar Smacks
Trigo	Sugar Smacks
	Nutrigrain (trigo)

(a)

LISTA DE PROVEEDORES
Fecha: 06/30/2001

Ingrediente	Proveedor	Precio
Maíz	Wilson	2.80
	J. Perkins	2.72
	Pollack	2.83
	McKay	2.80
Trigo	Adams	1.19
	Kroner	1.19
	Schmidt	1.22
Cebada	Wilson	0.85
	Pollack	0.84

(b)

➤ FIGURA 7-25

Reporte para la pregunta 7.26

Amor sin barreras
Basada en una idea de Jerome Robbins

Libreto de ARTHUR LAURENTS
Música de LEONARD BERNSTEIN
Canciones de STEPHEN SONDHEIM

Dirección y coreografía de la producción original por JEROME ROBBINS

Producida originalmente en Broadway por Robert E. Griffith y Harold S. Prince, con arreglos de Roger L. Stevens
Orquestación de Leonard Bernstein, con Sid Ramin e Irwin Kostal

PUNTOS CULMINANTES DE TODA LA GRABACIÓN

María . . . . . .KIRI TE KANAWA
Tony . . . . . JOSÉ CARRERAS
Anita . . . . . TATIANA TROYANOS
Riff . . . . . KURT OLLMAN
y MARILYN HORNE cantando "Somewhere"

Rosalía . . . . . Louise Edeiken
Consuela . . . . . Stella Zambalis
Fancisca . . . . . Angelina Reaux
Acción . . . . . David Livingston
Diesel . . . . . Marty Nelson
Bebé John. . . . . Stephen Bogardus
A-rab . . . . . Peter Thom
Niño de nieve . . . . . .Todd Lester
Bernardo. . . .Richard Harrell

1 **Jet Song** [3'13]
(Riff, Acción, Bebé John, A-rab, Coro)
2 **Something's Coming** [2'33]
(Tony)
3 **María** [2'56]
(Tony)
4 **Tonight** [5'27]
(María, Tony)
5 **America** [4'47]
(Anita, Rosalía, Coro)
6 **Cool** [4'37]
(Riff, Coro)
7 **One Hand, One Heart** [5'38]
(Tony, María)
8 **Tonight** (Conjunto) [3'40]
(Todo el elenco)
9 **I Feel Pretty** [3'22]
(María, Coro)
10 **Somewhere** [2'34]
(Una chica)
11 **Gee Oficer Krupke** [4'18]
(Acción, Niño de nieve, Diesel, A-rab, Bebé John, Coro)
12 **A Boy Like That** [2'05]
(Anita, María)
13 **I Have a Love** [3'30]
(María, Anita)
14 **Taunting Scene** [1'21]
(Orquesta)
15 **Finale** [2'40]
(María, Tony)

7.26 Usando como guía la cubierta del disco compacto que se muestra en la figura 7-25, realice las siguientes tareas:

a. Dibuje los diagramas de objetos para los objetos implícitos ARTISTA, PAPEL y CANCIÓN

b. Identifique las relaciones entre esos objetos. ¿Qué tipos de objetos son? (Simple, compuesto, etcétera.)

c. Indique si cada participante en una relación es opcional u obligatorio

d. Transforme los diagramas de objetos en diagramas de relaciones

¿Cuál es la llave para cada relación? ¿Qué llave externa aparece en cada relación?

➤ PROYECTOS

A. Termine el proyecto A que está al final del capítulo 4, si aún no lo ha hecho. Transforme su modelo de objeto semántico en un conjunto de relaciones. Si cualesquiera de sus relaciones no está en DK/NF, justifique su decisión de crearlas sin normalizar.

B. Concluya el proyecto B que está al final del capítulo 4, en caso de que no lo haya hecho. Transforme su modelo de objeto semántico en un conjunto de relaciones. Si cualesquiera de sus relaciones no está en DK/NF justifique su decisión de crearlas sin normalizar.

C. Termine el proyecto C que está al final del capítulo 4, si es que no lo ha hecho. Transforme su modelo de objeto semántico en un conjunto de relaciones. Si cualesquiera de sus decisiones no está en DK/NF, justifique su decisión de crearlas sin normalizar.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Si aún no lo ha hecho, cree objetos semánticos para las preguntas A y C en el proyecto FiredUp que se encuentra al final del capítulo 4.

A. Transforme el diseño de objeto semántico de la pregunta A al final del capítulo 4 en un conjunto de relaciones en forma normal dominio-llave. Para cada relación, especifique la llave primaria, llaves candidatas, si las hay, y llaves externas. Especifique todas las restricciones de integridad referencial. Si es necesario, desarrolle y fundamente sus supuestos considerando los implícitos semánticos de la aplicación.

B. Ajuste su respuesta a la pregunta A para permitir las relaciones sin normalizar, si considera que sólo las relaciones son apropiadas. Justifique cualquiera de las relaciones sin normalizar que tenga. Si es necesario, desarrolle y justifique los supuestos considerando los implícitos semánticos de la aplicación.

C. Transforme los diseños de los objetos semánticos de la pregunta C al final del capítulo 4 en un conjunto de relaciones, preferiblemente en forma normal dominio-llave. Si cualquiera de sus relaciones no está en forma normal dominio-llave, explique el porqué. Para cada relación especifique la llave primaria, las llaves candidatas, si las hay, y las llaves externas. Especifique todas las restricciones de integridad referencial.

D. Ajuste su respuesta a la pregunta C, y suponga que casa, fax y teléfono celular están representados por separado, con atributos de valor individual. ¿Éste es un mejor diseño que el de su respuesta a la pregunta C? Explique por qué.

PARTE IV

IMPLEMENTACIÓN DE BASES DE DATOS CON EL MODELO RELACIONAL

La Parte IV aborda la implementación de bases de datos usando el modelo relacional. El capítulo 8 inicia con un análisis sobre el manejo de datos relacionales. Primero nos enfocamos en los tipos de lenguajes de manejo de datos relacionales y después explicamos los operadores básicos del álgebra relacional, e ilustramos su uso.

El capítulo 9 describe el lenguaje de consulta estructurado, o SQL. Dicho lenguaje tiene el respaldo del American National Standards Institute (ANSI), como estándar para el manejo de bases de datos relacionales, y también es el lenguaje más importante de manejo de datos para los productos DBMS relacionales de comercio. El capítulo 10 concluye esta parte con un análisis sobre el diseño de aplicaciones de bases de datos.

CAPÍTULO 8

Fundamentos de la implementación relacional

Este capítulo aborda la implementación de las bases de datos relacionales. Comenzaremos por definir los datos relacionales, revisando la terminología correspondiente y explicando cómo se define un diseño con el DBMS. Después regresaremos a la asignación de espacio y creación de datos de la base de datos. El resto del capítulo versará sobre el manejo de datos relacionales: primero se realiza un estudio de los cuatro tipos de lenguajes para el manejo de los datos relacionales (DML, por sus siglas en inglés); enseguida se abordan los tres modos comunes de interfaz DML con el DBMS y, por último, las operaciones básicas de álgebra relacional y consultas mediante ejemplo expresadas en términos de álgebra relacional.

➤ DEFINICIÓN DE DATOS RELACIONALES

Cuando se implementa una base de datos relacional hay que desarrollar varias tareas. Primero, se debe definir la estructura de una base de datos con el DBMS. Para hacerlo, los programadores usan un lenguaje de definición de datos (DDL, por sus siglas en inglés) o algún otro medio equivalente (como por ejemplo una pantalla) para mostrar la estructura. A continuación le asignan a la base de datos un medio de almacenamiento físico y lo llenan con datos. En esta sección analizaremos cada una de estas tareas, pero primero revisaremos la terminología relacional.

Revisión de terminología

Como se estableció en el capítulo 5, una **relación** es una tabla que tiene varias propiedades:

1. Las entradas en la relación tienen sólo un valor; no se permiten valores múltiples. Por lo tanto, la intersección de un renglón y una columna sólo contienen un valor.
2. Todas las entradas en cualquier columna son de la misma clase. Por ejemplo, una columna puede contener nombres de clientes y otra, de cumpleaños. Cada una tie-

➤ FIGURA 8-1

Ejemplo de una estructura de la relación PACIENTE

	Col 1 (o atributo 1)	Col 2	Col 3	Col 4	Col 5
	Nombre	Fecha de nacimiento	Género	Número de cuenta	Médico
Renglón 1 (o tuple 1)	Riley	01/19/1946	F	147	Lee
Renglón 2	Murphy	12/28/1981	M	289	Singh
Renglón 3	Krajewski	10/21/1973	F	533	Levy
Renglón 4	Ting	05/23/1938	F	681	Spock
Renglón 5	Dixon	04/15/1987	M	704	Levy
Renglón 6	Abel	06/19/1957	M	193	Singh

ne un nombre único y el orden de las columnas no es importante en la relación. Las columnas de la relación se llaman **atributos.** Cada atributo tiene un **dominio,** que es una descripción física y lógica de los valores permitidos.

3. En la relación no existen dos renglones que sean idénticos, y el orden de los renglones no es importante (véase la figura 8-1). Cada renglón de la relación se conoce como tuple.

La figura 8-1 es un ejemplo, o prueba. El formato generalizado PACIENTE (Nombre, FechadeNacimiento, Género, NúmerodeCuenta, Médico) corresponde a la estructura de la relación y es a lo que la mayoría de las personas se refiere cuando usa el término *relación*. (Recuerde que, como establecimos en el capítulo 5, un atributo subrayado es una llave.) Si agregamos restricciones en los valores de datos permitidos a la estructura de la relación, entonces tenemos un **esquema relacional.** Estos términos se resumen en la figura 8-2.

CONFUSIÓN CON RESPECTO AL TÉRMINO *LLAVE*. El término **llave** es una fuente común de confusión porque tiene diferentes significados en las etapas de diseño e implementación. Durante el diseño, el término *llave* se refiere a una o más columnas que únicamente identifican un renglón en una relación. Como explicamos en el capítulo 5, sabemos que cada relación tiene una llave porque cada renglón es único; en la restricción, la composición de cada columna en la relación es una llave. Sin embargo, con frecuencia la llave se compone de una o dos columnas.

Durante la implementación, el término *llave* se usa en forma diferente. Para la mayoría de los productos relacionales, una llave es una columna en la cual el DBMS construye un índice u otra estructura de datos. Esto se hace para tener acceso rápido a los renglones por medio de ese valor de columna. Así, las llaves no necesariamente son únicas y, de hecho, con mucha frecuencia no lo son. Éstas se construyen sólo para mejorar el desempeño (véase el apéndice A para información con respecto a estas estructuras de datos).

Por ejemplo, considere la relación PEDIDO (NúmerodePedido, FechadePedido, NúmerodeCliente, Cantidad). Desde el punto de vista del *diseño* relacional, la llave de esta relación es NúmerodePedido, ya que el subrayado significa que NúmerodePedido identifica en forma única los renglones de la relación. Sin embargo, desde el punto de vista de la *implementación* relacional, cualesquiera de las cuatro columnas pueden ser una llave. Por ejemplo, FechadePedido se puede definir como una llave. De ser así, el DBMS creará una estructura de datos de tal forma que se pueda acceder en forma rápida a los renglones de PEDIDO mediante el valor de FechadePedido. Probablemente habrá muchos renglones para un valor determinado de FechadePedido. Definirlo como llave no dice nada con respecto a su unicidad.

Algunas veces los términos **llave lógica** y **llave física** se usan para distinguir entre ambos significados de llave. Una llave lógica es un identificador único, mientras que una física es una columna que tiene un índice u otra estructura de datos definida, con el fin de mejorar el rendimiento.

➤ FIGURA 8-2

Resumen de terminología relacional

Término	*Significado*
Relación (o tabla) (o archivo)	Tabla de dos dimensiones.
Atributo (o columna) (o campo) (o elemento de datos)	Columna de una relación.
Tuple (o renglón) (o registro)	Renglón en una relación.
Dominio	Descripción física o lógica de valores permitidos.
Estructura de la relación	Formato de la relación.
Ocurrencia	Estructura de la relación con datos.
Esquema relacional	Estructura de la relación más restricciones.
Llave	Grupo de uno o más atributos que en forma única identifican un tuple en una relación.
Llave lógica	Igual que llave.
Llave física (o índice)	Un grupo de uno o más atributos soportado por una estructura de datos que facilita la rápida recuperación, o un rápido acceso secuencial.

ÍNDICE. Puesto que una llave física por lo general es un índice, algunas personas reservan el término *llave* para una llave lógica y usan *índice* para una llave física. En este texto haremos exactamente eso: usaremos dicho término para mencionar una llave lógica, y el término *índice* para representar una llave física.

Hay tres razones para la definición de índices. Una es permitir el acceso en forma rápida a los renglones por medio del valor del atributo indizado. Otra es facilitar el ordenamiento de renglones de acuerdo con ese atributo. Por ejemplo, en PEDIDO, FechadePedido puede estar definida como llave, de tal forma que pueda generarse rápidamente un reporte que muestre los pedidos por fecha.

Una tercera razón para la construcción de un índice es imponer la unicidad. A pesar de que los índices no tienen que ser únicos, cuando el programador quiere que una columna lo sea, se crea un índice por medio del DBMS. Con la mayoría de los productos DBMS relacionales se puede imponer que una columna, o grupo de columnas, tenga carácter de única usando la palabra llave UNIQUE cuando se defina la aparición de una columna en una tabla.

Implementación de una base de datos relacional

En este texto usamos el modelo relacional para expresar el diseño de bases de datos. Puesto que así lo hemos hecho, podemos proceder directamente a partir del diseño de la base de datos hasta implementarla. No es necesario transformar el diseño durante la etapa de implementación, simplemente definimos el diseño relacional que existe en el DBMS.

La situación es diferente cuando implementamos bases de datos usando productos DBMS basados en modelos de datos distintos al modelo relacional. Por ejemplo, cuando se implementa una base de datos para DL/I, debemos convertir el diseño relacional en uno jerárquico, y después definir el diseño convertido al producto DBMS.

DEFINICIÓN DE LA ESTRUCTURA DE BASE DE DATOS AL DBMS. Existen diferentes medios para describir la estructura de bases de datos en el DBMS, dependiendo del producto de DBMS que se use. Con algunos productos se construye un archivo de texto que describa la estructura de la base de datos. El lenguaje que se usa para describirla a veces se conoce como **lenguaje de definición de datos,** o DDL (Data Definition Language, por sus siglas en inglés). El archivo de texto DDL da nombre a las tablas en la base de datos, nombra y describe las columnas de esas tablas, define los índices y describe otras estructuras tales como restricciones y restricciones de seguridad. La figura 8-3 muestra el lenguaje de definición de datos típico que se usa para definir una base de datos relacional simple en el caso de un DBMS hipotético. En los capítulos 12 y 13 se muestran ejemplos más reales usando una norma llamada SQL.

Algunos productos DBMS no requieren que un DDL defina la base de datos en el formato de archivo de texto. Una alternativa común es proporcionar un medio gráfico para definir la estructura de la base de datos. Por ejemplo, con Access 2002 se le muestra al programador una estructura gráfica de lista y se le pide que llene la tabla y los nombres de las columnas en los lugares apropiados. En el capítulo 2 vimos un ejemplo de esto (figura 2-4).

En general, las facilidades de definición gráfica son comunes para los productos DBMS en las computadoras personales. Tanto los DDL textuales como gráficos son usuales en productos DBMS, en servidores y macrocomputadoras. Por ejemplo, se emplean tanto servidores de ORACLE como de SQL. La figura 8-4 resume los procesos de definición de bases de datos.

Independientemente de los medios por los cuales se defina la estructura de la base de datos, el programador debe nombrar cada tabla, definiendo las columnas en ésta, así como el formato físico (por ejemplo TEXTO 10) de cada una. Así mismo, dependiendo de los medios del DBMS, el programador puede especificar las restricciones que se imponen al DBMS. Por ejemplo, se pueden establecer los valores de columna como NO NULO (NOT NULL o UNIQUE). Algunos productos también permiten definir el rango y las restricciones de valor (partes menores que 10000, o color igual a uno de [‘Rojo’, ‘Verde’, ‘Azul’]). Por último, se pueden establecer restricciones de interrelación. Un ejemplo es que NúmerodeDepartamento en EMPLEADO debe coincidir con un valor de NúmerodeDepartamento en DEPARTAMENTO.

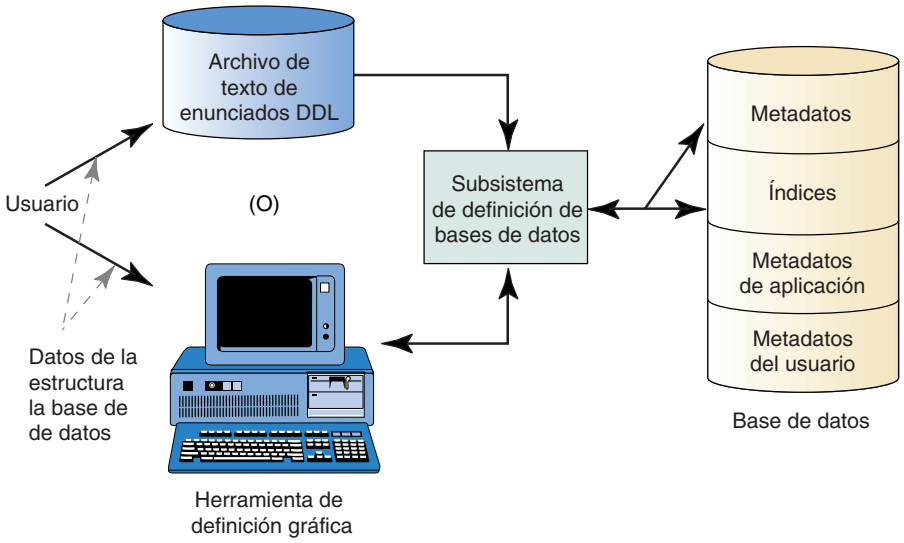
➤ FIGURA 8-3

Ejemplo de un archivo de texto DDL para la definición de una base de datos

```
CREATE SCHEMA MÉDICO

CREATE TABLE PACIENTE

(Nombre                 CHARACTER VARYING (35) NOT NULL,
 Fecha de nacimiento    DATE/TIME,
 Género                 CHARACTER VARYING (10),
 NúmerodeCuenta         INTEGER NOT NULL,
 NombredelMédico_FK1    CHARACTER VARYING (35) NOT NULL,

 PRIMARY KEY (NúmerodeCuenta)
 FOREIGN KEY (NombredelMédico_FK1)
      REFERENCES MÉDICO
)

CREATE TABLE MÉDICO

 (NombredelMédico       CHARACTER VARYING (35) NOT NULL,
  CódigodeÁrea          CHARACTER VARYING (3),
  NúmeroLocal           CHARACTER VARYING (8) NOT NULL,

  PRIMARY KEY (NombredelMédico)
 )
```

➤ FIGURA 8-4

Proceso para la definición de la base de datos



Con muchos productos, el programador también puede definir contraseñas (*passwords*) y otros medios de control y seguridad. Como se muestra en el capítulo 11, se pueden usar diferentes estrategias. Algunas colocan controles en las construcciones de datos (por ejemplo, contraseñas en tablas) y otras los colocan en las personas (el usuario de la contraseña X puede leer y actualizar las tablas T1 y T2).

ASIGNACIÓN DE ESPACIO. Además de definir la estructura de una base de datos, el programador debe asignar estructuras de base de datos a medios físicos. Otra vez, las tareas específicas dependerán del producto particular DBMS que se use. Para una base de datos personal todo lo que se necesita hacer es asignar la base de datos a un directorio y darle a ésta un nombre. Entonces el DBMS asigna espacio de almacenamiento en forma automática.

Otros productos DBMS, especialmente los que se emplean en servidores y macrocomputadoras, requieren más trabajo. Para mejorar el rendimiento y el control se debe planear con cuidado la distribución de los datos en la base, tanto en el disco como en los canales. Por ejemplo, dependiendo de la naturaleza del procesamiento de la aplicación, puede ser recomendable ubicar ciertas tablas en el mismo disco, o puede ser importante asegurarse de que éstas no sean colocadas en el mismo disco.

Considere, por ejemplo, un objeto de pedido que está conformado por los datos de las tablas PEDIDO, ARTÍCULO-LÍNEA y ARTÍCULO. Suponga que cuando se procesa un PEDIDO la aplicación recupera un renglón desde PEDIDO, varios de ARTÍCULO-LÍNEA y un renglón de ARTÍCULO por cada uno de ARTÍCULO-LÍNEA. Además, los renglones ARTÍCULO-LÍNEA de un pedido específico tienden a estar agrupados, pero los de ARTÍCULO no estarán todos agrupados. La figura 8-5 ilustra lo anterior.

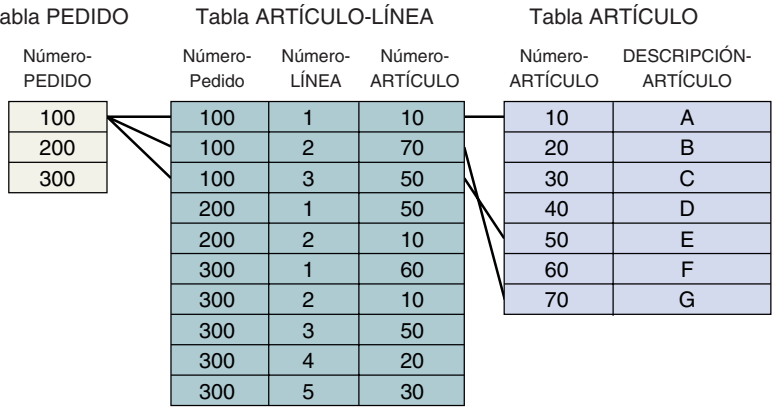
Ahora suponga que una organización procesa en conjunto muchos pedidos y tiene un disco grande y rápido y otro pequeño y lento. El programador debe determinar cuál es el mejor lugar para colocar los datos. Una posibilidad es que el desempeño mejore si la tabla ARTÍCULO se almacena en el disco más grande y más rápido, y los datos de PEDIDO y ARTÍCULO-LÍNEA se ubiquen en el más pequeño y lento. Quizás el desempeño mejorará si los datos de PEDIDO y ARTÍCULO-LÍNEA de los pedidos de meses anteriores se colocan en el disco lento y todos los datos de los pedidos de este mes en el disco rápido.

No podemos responder aquí esta pregunta, ya que la respuesta depende de la cantidad de datos, de las características de procesamiento del DBMS y del sistema operativo, del tamaño y la velocidad de los discos y canales, y de los requisitos para el procesamiento de todas las aplicaciones que usa la base de datos. Lo importante es que se deben considerar estos factores cuando se asigne el espacio a los medios de la base de datos.

Además de especificar la ubicación y cantidad de espacio para los datos de usuario, el programador también puede necesitar declarar si el espacio de archivo se incrementa-

➤ FIGURA 8-5

Datos de ejemplo de tres tablas que representan un PEDIDO



Nota: Para un pedido específico, los renglones ARTÍCULO-LÍNEA están agrupados, pero los renglones ARTÍCULO no.

rá o no cuando sea necesario y, si es así, cuánto aumentará. Por lo general, esa cantidad de espacio adicional se expresa como una cantidad especifica, o como un porcentaje del espacio inicial.

En la creación de la base de datos, el programador también necesitará ubicar el espacio de archivo para los registros de la base de datos. Usted aprenderá acerca de los registros en los capítulos 11 al 13; por ahora sólo tiene que darse cuenta de que el DBMS registrará cambios de datos que se pueden usar más tarde para recuperar la base de datos en caso de que sea necesario. El espacio para estos registros se define cuando se crea la base de datos.

CREACIÓN DE UN PLAN DE MANTENIMIENTO PARA LA BASE DE DATOS. Un plan de mantenimiento es una planificación de actividades que se llevarán a cabo de manera repetitiva. Estas tareas incluyen: respaldo de la base de datos, vaciado del contenido de los registros de la base en archivos de respaldo, verificación de violaciones de integridad referencial, optimización del espacio en disco para los datos del usuario e índices, etc. Este tema lo abordaremos en el capítulo 11, por lo pronto estemos conscientes de que se debe desarrollar un plan de mantenimiento cuando se crea la base de datos, o un poco después.

CREACIÓN DE LA BASE DE DATOS. Una vez que se ha definido la base de datos y ha sido ubicada en un almacenamiento físico, se puede alimentar con datos. Los medios a través de los cuales se realiza esto dependen de los requisitos de la aplicación y de las características de los productos DBMS. En el mejor de los casos, todos los datos ya están en un formato entendible para la computadora y el DBMS tiene características y herramientas que facilitan la importación de los datos desde medios magnéticos. En el peor de los casos, todos los datos se deberán introducir manualmente mediante el uso de programas de aplicación que desarrollen los programadores. La mayoría de las conversiones de datos están entre estos dos extremos.

Una vez que se introducen los datos, se deben revisar con esmero. La verificación es una tarea intensiva y tediosa, pero muy importante. Con frecuencia, especialmente en el caso de grandes bases de datos, es bueno valorar el tiempo y el gasto del equipo de desarrollo, así como escribir programas de verificación. Dichos programas cuentan el número de registros de varias categorías, calculan los totales de control, realizan comprobaciones razonables con respecto a los valores de datos de los artículos y proporcionan otros tipos de verificación.

➤ MANEJO DE DATOS RELACIONALES

Hasta aquí hemos analizado el diseño de bases de datos relacionales y los medios a través de los cuales se definen en el DBMS. Siempre que nos hemos referido al procesamiento de relaciones lo hemos hecho de una manera general e intuitiva. Aunque está

bien en el caso de los diseños analizados, para implementar aplicaciones necesitamos lenguajes claros y no ambiguos con el fin de expresar la lógica del procesamiento. Estos lenguajes se llaman **lenguajes de manipulación de datos (DML).**

Categorías de lenguajes de manipulación de datos relacionales

A la fecha se han propuesto cuatro estrategias diferentes de manejo de datos relacionales: el **álgebra relacional,** que es la primera de estas estrategias, define operadores que trabajan en relaciones (de manera similar a los operadores +, –, etc., del álgebra de secundaria). Las relaciones se pueden manipular usando estos operadores para lograr el resultado deseado. Pero el álgebra relacional es difícil de usar, en parte porque está basada en procedimientos; esto es, cuando se usa álgebra relacional debemos saber no sólo *qué* queremos, sino *cómo* obtenerlo.

En el proceso de bases de datos comerciales no se usa el álgebra relacional. A pesar de que los productos DBMS no comerciales exitosos proporcionan facilidades de álgebra relacional, aquí analizaremos ésta porque ayuda a aclarar el manejo relacional y a establecer las bases para aprender SQL.

El **cálculo relacional** es un segundo tipo de manejo de datos relacionales. No se basa en procedimientos; es un lenguaje para decir lo que queremos sin expresar cómo obtenerlo. Recordemos la variable de integración en cálculo, cuyo rango se extiende sobre el intervalo de integración. El cálculo relacional tiene una variable similar. Para el cálculo relacional de tuples la variable se extiende sobre los tuples de la relación, y para el cálculo relacional del dominio la variable se extiende sobre los valores de un dominio. El cálculo relacional se deriva de una rama de las matemáticas llamada cálculo de predicados.

A menos que usted se vaya a convertir en un teórico de la tecnología relacional, probablemente no necesitará aprender este tipo de cálculo. Nunca se usa en el procesamiento comercial de bases de datos, y para nuestros fines no es necesario aprenderlo, así que no lo analizaremos.

Aunque el cálculo relacional es difícil de entender y usar, su característica respecto a que no está basado en procedimientos es muy conveniente. Por lo tanto, los diseñadores DBMS buscaron otras técnicas no basadas en procedimientos, lo cual condujo a la tercera y cuarta categorías de DML relacionales.

Los **lenguajes orientados a la transformación** son una clase de lenguajes no basados en procedimientos que transforman datos de entrada expresados como relaciones en resultados que se expresan como una relación independiente. Estos lenguajes proporcionan estructuras fáciles de usar para manifestar lo que es conveniente, considerando los datos proporcionados. SQUARE, SEQUEL y SQL son lenguajes orientados a la transformación. Estudiaremos SQL con detenimiento en los capítulos 9, 12 y 13.

La cuarta categoría de DML relacional es gráfica. **Query-by-Example** (consulta mediante ejemplo) y **Query-by-Form** (consulta por forma) entran en esta categoría. Los productos que se basan en esta categoría son Approach (de Lotus) y Access. Mediante una interfaz gráfica, se ofrece al usuario la materialización de una o más relaciones, la cual podría ser una forma de ingreso de datos, una hoja de cálculo, o alguna otra estructura. Los DBMS asocian la materialización a la relación implícita y construyen consultas (probablemente en SQL) en beneficio del usuario. Los usuarios entonces hacen que las instrucciones DML se ejecuten, pero no tienen conocimiento de este hecho. Las cuatro categorías de los DML relacionales se listan en la figura 8-6.

➤ FIGURA 8-6

Cuatro categorías de DML relacional

- Álgebra relacional
- Cálculo relacional
- Lenguajes orientados a la transformación (como SQL)
- Query-by-example/Query-by-form

➤ FIGURA 8-7

Ejemplo de una forma de pantalla tabular predeterminada

Nombre	FechadeNacimiento	Género	NúmerodeCuenta	Medio
Riley	1/19/46	F	147	Lee
Abel	6/19/57	M	193	Singh
Murphy	12/28/81	M	289	Singh
Krajewski	10/21/73	F	533	Levy
Ting	5/23/38	F	661	Spock
Dixon	4/15/87	M	704	Levy

Interfaces DML con el DBMS

En esta sección consideramos cuatro interfaces diferentes para manejar datos de la base.

MANIPULACIÓN DE DATOS MEDIANTE FORMAS. La mayoría de los productos DBMS relacionales incluyen herramientas para construir formas. Algunas se crean automáticamente cuando se define una tabla, pero otras las debe crear el programador, quizás con ayuda inteligente como la que proporcionan los Wizards, de Access. Una forma puede ser tabular, por ejemplo una hoja de cálculo, en cuyo caso mostrará múltiples renglones a la vez, o puede mostrar cada renglón como una entidad independiente. Las figuras 8-7 y 8-8 muestran un ejemplo de cada una para la tabla PACIENTE (PATIENT) en la figura 8-1. Con la mayoría de los productos, se proporciona alguna flexibilidad en el procesamiento de formas y reportes. Por ejemplo, se pueden seleccionar renglones para el procesamiento basado en valores de columnas y también se pueden ordenar. La tabla de la figura 8-7 está ordenada por NúmerodeCuenta.

Muchas de las formas predeterminadas presentan datos de una sola relación a la vez. Si se requieren datos de dos o más relaciones, entonces por lo general se fabrican formas que se deben crear usando herramientas DBMS. Es posible crear ambas formas multitabla y multirrenglón usando estas herramientas. Sin embargo, el uso de dichas herramientas depende del producto, así que por lo tanto no las analizaremos.

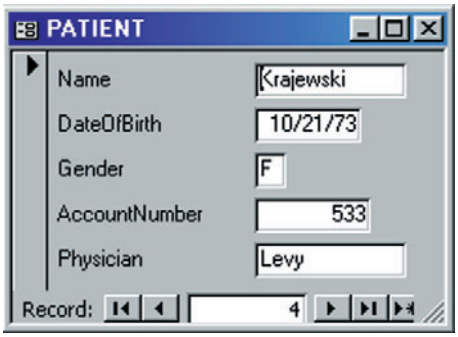
INTERFAZ DE LENGUAJE CONSULTA-ACTUALIZACIÓN. El segundo tipo de interfaz para una base de datos es a través de un **lenguaje de consulta-actualización,** o simplemente **lenguaje de consulta** (aunque la mayoría de estos lenguajes realizan tanto la consulta como la actualización, por lo general se les conoce como lenguajes de consulta). Con este tipo de lenguaje el usuario introduce órdenes de consulta que especifican acciones en la base de datos. Los DBMS decodifican las órdenes y llevan a cabo las acciones apropiadas. La figura 8-9 muestra los programas involucrados en el procesamiento de consultas.

El lenguaje de consulta más importante es SQL. Para dar una idea sobre los lenguajes de consulta, considere la siguiente instrucción de SQL que procesa la relación PACIENTE, la cual se muestra en la figura 8-1:

```
SELECT    Nombre, FechadeNacimiento
FROM      PACIENTE
WHERE     Médico = 'Levy'
```

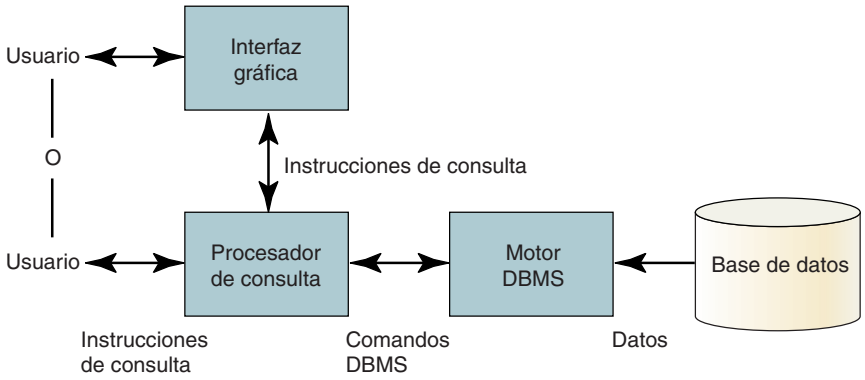
➤ FIGURA 8-8

Ejemplo de una forma de pantalla de un renglón individual predeterminado



➤ FIGURA 8-9

Programas involucrados en el procesamiento de consultas



Esta instrucción de consulta SQL extrae todos los renglones de la relación PACIENTE en la que el médico es 'Levy'. En una segunda tabla despliega Nombre y FechadeNacimiento para los renglones que cumplen esta condición.

PROCEDIMIENTOS ALMACENADOS (*STORED PROCEDURES*). Con el tiempo, los usuarios y los programadores de bases de datos encuentran que ciertas secuencias de órdenes SQL necesitan realizarse periódicamente. El único cambio de un periodo a otro es el uso de diferentes valores en las cláusulas WHERE. Por ejemplo, factura de fin de mes implica procesar la misma instrucción SQL, pero con una fecha de cierre diferente. Para atender esta necesidad, los proveedores de DBMS desarrollaron **procedimientos almacenados,** los cuales son conjuntos de instrucciones SQL que se guardan como un archivo al que se puede llamar con una sola orden. Cuando se invoca los parámetros pueden pasar por ese procedimiento para ser llenados en las cláusulas WHERE, etc. Un ejemplo de su uso es:

```
DO BILLING STORED_PROCEDURE FOR BILLDATE = "9/1/2000"
(HACER PROCEDIMIENTO DE FACTURACIÓN ALMACENADA PARA FECHA DE FACTURA = "19/1/2000")
```

A medida que los programadores obtienen experiencia el problema se aclara. SQL fue creado como un sublenguaje de datos y no como un conjunto de características de programación. Sin embargo, algunos de los aspectos faltantes eran necesarios para almacenar procedimientos, por lo cual los fabricantes de DBMS ampliaron a SQL para agregarlos. Uno de tales lenguajes, PL/SQL, lo desarrolló Oracle; otro, llamado TRANSACT-SQL, lo creó SQL Server. En los capítulos 12 y 13 aprenderá sobre estos lenguajes.

Un tipo especial de procedimientos almacenados, llamado **disparador** (trigger), lo invoca el DBMS cuando tiene lugar una condición específica. Por ejemplo, en una aplicación de entrada de órdenes, un programador puede crear un disparador que se active siempre que CantidadDisponible de un artículo en un inventario esté por debajo de su asociado CantidadReOrdenada. Aprenderá más acerca de procedimientos almacenados y disparadores en los capítulos 12 y 13.

INTERFAZ DE PROGRAMAS DE APLICACIÓN. El cuarto tipo de interfaz para el ingreso de datos se encuentra en los programas de aplicación escritos en lenguajes de programación tales como COBOL, BASIC, Perl, Pascal y C++. Además, algunos programas de aplicación están escritos en lenguajes que proporcionan los fabricantes de DBMS, entre los cuales el lenguaje de programación de dBASE es el más conocido.

Existen dos estilos de interfaz de programas de aplicación para los DBMS. En uno, el programa de aplicación invoca rutinas de una biblioteca de funciones que forma parte del DBMS. Por ejemplo, para leer determinado renglón de una tabla, dicho programa llama a una función de lectura del DBMS y transmite parámetros que indican la tabla a la que se debe accesar, los datos que hay que recuperar, el criterio de selección de renglones, etcétera.

En algunos casos la sintaxis de la orientación de objetos es más usada que las llamadas de función. En el siguiente código Access, el objeto de referencia *db* se coloca

en la actual base de datos abierta y un segundo objeto de referencia *rs* apunta a los renglones de la tabla PATIENT (PACIENTE).

```
set db = currentdb( )
set rs = db.OpenRecordset("PATIENT")
```

Entonces se puede tener acceso a las propiedades del *record set* abierto y se pueden ejecutar métodos usando la variable de referencia. Por ejemplo, la propiedad *rs.AllowDeletions* se puede referenciar para decidir cuáles registros en PACIENTE se pueden eliminar. El método *rs.MoveFirst*, se puede usar para posicionar un cursor en el primer renglón.

El segundo y más antiguo estilo de interfaz a veces se usa en macrocomputadoras y productos de servidores DBMS. En este caso, el vendedor del DMBS define un conjunto de comandos de acceso de datos de alto nivel. Estos comandos, que son particulares del procesamiento de bases de datos y no son parte de cualquier lenguaje estándar, están implantados dentro del código de los programas de aplicación.

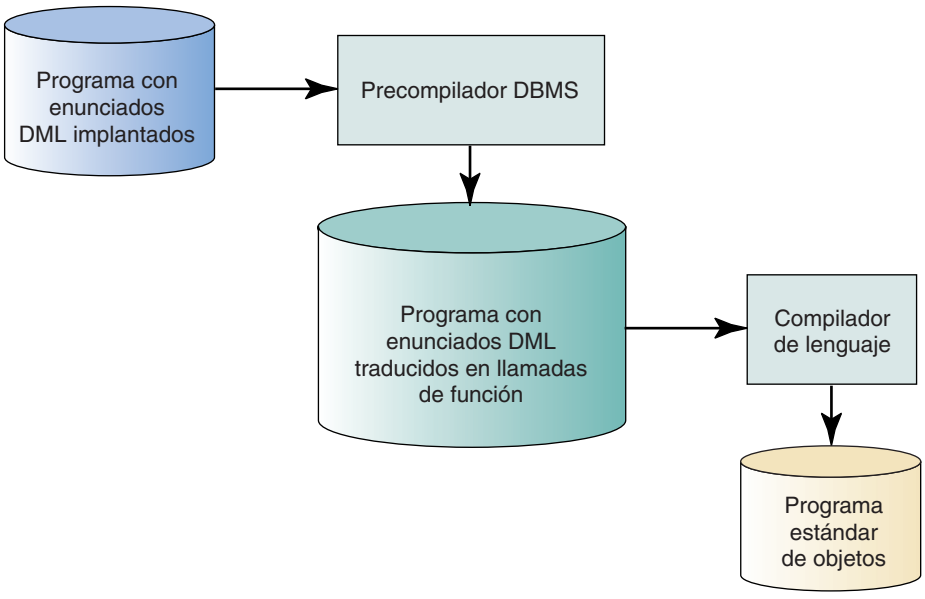
El programa de aplicación, con órdenes implantadas, es sometido entonces a un precompilador que proporciona el vendedor de DBMS. Este precompilador traduce las instrucciones de acceso de datos en llamadas de funciones válidas y definiciones de estructura de datos. En este proceso, el precompilador establece secuencias de parámetros para las llamadas y define áreas de datos que compartirán los programas de aplicación y el DBMS. El precompilador también inserta programas lógicos para mantener las áreas de datos. Así, la rutina del precompilador se somete al compilador de lenguaje. En la figura 8-10 se muestran las relaciones de los programas que involucra este proceso.

Además de su papel en el procesamiento de consultas, SQL también se usa como un lenguaje en programas de aplicación. Así, las instrucciones de SQL se implantan en los programas y se traducen en funciones de llamadas mediante un precompilador. Los costos de entrenamiento y el tiempo de aprendizaje se reducen debido a que se puede usar el mismo lenguaje para acceder tanto a la consulta como a los programas de aplicación. Sin embargo, hay un problema: SQL es un lenguaje orientado a la transformación que acepta relaciones, las maneja y arroja una relación resultante. Por lo tanto, aborda una relación a la vez. Casi todos los programas de aplicación están orientados a los renglones (registros); esto es, leen un renglón, lo procesan, leen el siguiente renglón, y así sucesivamente. Estos programas trabajan con un renglón a la vez.

Por ende, existe un desajuste entre la orientación básica de SQL y los lenguajes de programas de aplicación. Para corregir dicho desajuste, se supone que en el programa

FIGURA 8-10

Programa de procesamiento con enunciados DML implantados



de aplicación los resultados de las instrucciones de SQL se consideran como archivos. Para ilustrar esto suponga que los siguientes enunciados SQL (los mismos de antes) están insertados en el programa de aplicación:

```
SELECT    Nombre, FechadeNacimiento
FROM      PACIENTE
WHERE     Médico = 'Levy'
```

El resultado de estos enunciados es una tabla que tiene dos columnas y *N* renglones. Con el fin de aceptar los resultados de esta consulta, el programa de aplicación está escrito para suponer que esos enunciados han producido un archivo con *N* registros. La aplicación abre la consulta, procesa el primer renglón, procesa el siguiente renglón, y así sucesivamente, hasta que el último ha sido procesado. Esta lógica es la misma que para el procesamiento de un archivo secuencial. Verá ejemplos de estos programas de aplicación en los capítulos 12, 13, 15 y 16. Por ahora, sólo tenga en mente que existe un desajuste en la orientación básica de SQL (orientado a relación) y los lenguajes de programación (orientados a renglones, o registros), y que ese desajuste se debe corregir cuando los programas acceden a una base de datos relacional a través de SQL.

➤ ÁLGEBRA RELACIONAL

El álgebra relacional es similar al álgebra que aprendió en la preparatoria, pero con una diferencia importante: en el álgebra de la preparatoria las variables representan números, y los operadores como +, −, ×, y / operan sobre cantidades numéricas. Sin embargo, en el álgebra relacional las variables son relaciones y los operadores las manejan para formar nuevas relaciones. Por ejemplo, la operación de unión combina los tuples de una relación con los de otra, con lo cual producen una tercera relación. De hecho, el álgebra relacional es *cerrada*, lo cual significa que el resultado de una o más operaciones relacionales son *siempre* una relación.

Las relaciones son conjuntos. Los tuples de una relación se pueden considerar elementos de un conjunto y, por lo tanto, las operaciones que se pueden realizar en conjuntos también se pueden desarrollar en relaciones. Primero mostraremos cuatro de estos operadores de conjuntos y después analizaremos otros que son particulares del álgebra relacional. Sin embargo, antes de proceder considere las siguientes relaciones muestra que usaremos en este capítulo y en el siguiente.

OPERADORES RELACIONALES

En la figura 8-11 se muestran seis relaciones, sus atributos y definiciones de dominio. Observe que el atributo Nombre se usa en varias relaciones. Cuando nos referimos a un atributo específico, lo calificamos con el nombre de la relación. De acuerdo con lo anterior, Nombre en CLASE a veces se denota como CLASE.Nombre.

En el siguiente análisis los valores de caracteres se muestran entre comillas y los caracteres no entrecomillados representan nombres. Así "HABITACIÓN" difiere de Habitación porque "HABITACIÓN" es un valor, en tanto que Habitación es, por así decirlo, un nombre de dominio. Con respecto a datos numéricos, los números no entrecomillados se refieren a cantidades numéricas, y los entrecomillados, a cadenas de caracteres. Por lo que 123 es un número y "123" es una cadena de los caracteres "1", "2" y "3".

UNIÓN. La **unión** de dos relaciones está formada por la adición de tuples de una relación con los de una segunda relación que produce una tercera. El orden en el que aparecen los tuples en la tercera relación no es importante, pero se deben eliminar los que estén duplicados. La unión de relaciones A y B se denota por A + B.

Para que esta operación tenga sentido, las relaciones deben ser **compatibles en la unión;** esto es, cada relación debe tener el mismo número de atributos, y los atributos en las columnas correspondientes deben provenir del mismo dominio. Si, por

➤ FIGURA 8-11

Ejemplos de relaciones y dominios: (a) definiciones de relaciones, (b) atributos y dominios, y (c) definiciones de dominios

1. JUNIOR (Enum, Nombre, Especialidad)
2. ESTUDIANTE-HONOR (Número, Nombre, Interés)
3. ESTUDIANTE (EID, Nombre, Especialidad, Grado)
4. CLASE(Nombre, Horario, Aula)
5. INSCRIPCIÓN (NúmerodeEstudiante, NombredeClase, NúmerodePosición)
6. FACULTAD (FID, Nombre, Departamento)

(a)

	Atributo	*Dominio*
1.	Enum	IdentificadordePersona
	JUNIOR.Nombre	NombresdePersonas
	Especialidad	NombresdeMaterias
2.	Número	IdentificadordePersonas
	ESTUDIANTE-HONOR.Nombre	NombresdePersonas
	Interés	NombresdeMaterias
3.	EID	IdentificadordePersonas
	ESTUDIANTE.Nombre	NombresdePersonas
	Especialidad	NombresdeTemas
	Grado	Clases
4.	CLASE.Nombre	NombresdeClases
	Horario	HorariodeClases
	Aula	Aulas
5.	NúmerodeEstudiante	IdentificadoresdePersonas
	NombredeClase	NombresdeClases
	NúmerodePosición	TamañodeClases
6.	FID	IdentificadoresdePersonas
	FACULTAD.Nombre	NombresdePersonas
	Departamento	NombresdeTemas

(b)

Nombre del dominio	*Formato*
IdentificadoresdePersonas	Decimal (3)
NombresdePersonas	Carac (8) (no real, pero manejable en el caso de estos ejemplos)
NombresdeTemas	Carac (10)
Clases	Uno de [FR, SO, JR, SN, GR]
NombresdeClases	Carac (10)
HorariosdeClases	Carac (5) formato: DDDHH, donde D es uno de [M,T, W, R, F, o blanco], y HH es decimal entre 1 y 12
Aulas	Carac (5) formato: BBRRR, donde BB es un código construido y RRR es un número de aula
TamañodeClases	Decimal desde 0 hasta 100

(c)

➤ FIGURA 8-12

Relaciones y uniones de JUNIOR y ESTUDIANTE-HONOR: (a) ejemplo de Relación JUNIOR, (b) ejemplo de relación ESTUDIANTE-HONOR, y (c) relación de unión de JUNIOR y ESTUDIANTE-HONOR

Enum	Nombre	Especialidad
123	JONES	HISTORIA
158	PARKS	MATEMÁTICAS
271	SMITH	HISTORIA

(a)

Número	Nombre	Interés
105	ANDERSON	ADMINISTRACIÓN
123	JONES	HISTORIA

(b)

Enum o Número	Nombre	Especialidad o interés
123	JONES	HISTORIA
158	PARKS	MATEMÁTICAS
271	SMITH	HISTORIA
105	ANDERSON	ADMINISTRACIÓN

(c)

ejemplo, el tercer atributo de una relación proviene del dominio Aulas, el tercer atributo de la segunda relación también debe provenir del dominio Aulas.

En la figura 8-11 las relaciones JUNIOR y ESTUDIANTE-HONOR son compatibles en la unión porque ambas tienen tres atributos, los cuales provienen del mismo dominio. JUNIOR.Enum y ESTUDIANTE-HONOR.Nombre tienen el dominio IdentificadoresdePersona; JUNIOR.Nombre y ESTUDIANTE-HONOR.Nombre tienen el dominio NombresdePersonas; y JUNIOR.Especialidad y ESTUDIANTE-HONOR.Interés tienen el dominio NombredeMateria. Las relaciones JUNIOR y CLASE poseen tres atributos cada una, pero tienen **incompatibilidad de la unión** porque los tres atributos no tienen el mismo dominio.

En la figura 8-12 se muestra la unión de las relaciones JUNIOR y ESTUDIANTE-HONOR. Observe que el tuple [123, JONES, HISTORIA], el cual ocurre en ambas relaciones, no está duplicado en la unión.

DIFERENCIA. La **diferencia** de dos relaciones es una tercera relación que contiene tuples que están presentes en la primera relación, pero no en la segunda. Las relaciones deben ser compatibles en la unión. La diferencia de JUNIOR y ESTUDIANTE-HONOR se muestra en la figura 8-13. Al igual que en aritmética, el orden de la sustracción es importante y por lo tanto A – B *no es* lo mismo que B – A.

INTERSECCIÓN. La **intersección** de dos relaciones es una tercera relación que contiene los tuples que aparecen tanto en la primera como en la segunda relación. Una vez más, las relaciones deben ser compatibles en la unión. En la figura 8-14 la intersec-

➤ FIGURA 8-13

Relación JUNIOR menos ESTUDIANTE-HONOR

Enum	Nombre	Especialidad
158	PARKS	MATEMÁTICAS
271	SMITH	HISTORIA

➤ FIGURA 8-14

Relaciones de intersección de JUNIOR y ESTUDIANTE-HONOR

Enum o Número	Nombre	Especialidad o Interés
123	JONES	HISTORIA

ción de JUNIOR y ESTUDIANTE-HONOR es un tuple independiente [123, JONES, HISTORIA], el cual es el único que aparece en JUNIOR y en ESTUDIANTE-HONOR.

PRODUCTO. El **producto** de dos relaciones (a veces llamado **producto cartesiano**), es la concatenación de cada tuple de una relación con cada tuple de una segunda relación. El producto de relación A (con *m* tuples) y relación B (con *n* tuples) tiene *m* veces *n* tuples. El producto se denota como A × B, o A VECES B. En la figura 8-15 la relación ESTUDIANTE tiene cuatro tuples y la relación INSCRIPCIÓN tiene tres. Por lo tanto, ESTUDIANTE VECES INSCRITO tiene doce tuples que se muestran en la figura 8-16. (La relación resultante en la figura 8-16 contiene algunos tuples sin importancia. Necesitarían llevarse a cabo otras operaciones que mostraremos más adelante para extraer cualquier información importante de esta relación. Esto es simplemente una ilustración del operador producto.)

PROYECCIÓN. **Proyección** es un operador que selecciona atributos específicos de una relación. El resultado de la proyección es una nueva relación con los atributos seleccionados; en otras palabras, una proyección escoge columnas de una relación. Por ejemplo, considere los datos de la relación ESTUDIANTE en la figura 8-15(a), de los cuales la proyección de ESTUDIANTE en los atributos Nombre y Especialidad, denotados con corchetes como ESTUDIANTE [Nombre, Especialidad], se muestran en la figura 8-17(a). La proyección de ESTUDIANTE en Especialidad y Grado, que se denota como ESTUDIANTE [Especialidad, Grado], aparece en la figura 8-17(b).

Observe que aunque ESTUDIANTE tiene cuatro tuples para empezar, la proyección ESTUDIANTE [Especialidad, Grado] tiene sólo tres. Se eliminó un tuple porque después que la proyección fue terminada, el tuple [HISTORIA, JUNIOR] aparece dos veces. Debido a que el resultado de proyección es una relación, y a que las relaciones no pueden contener tuples duplicados, se elimina el tuple redundante.

La proyección también se puede usar para cambiar el orden de los atributos en una relación. Por ejemplo, la proyección ESTUDIANTE [Grado, Especialidad, Nombre, EID] invierte el orden de atributos de ESTUDIANTE (véase la figura 8-11 para conocer el orden original). Esta característica se puede usar algunas veces para hacer dos relaciones compatibles en la unión.

➤ FIGURA 8-15

Ejemplo de relaciones: (a) ESTUDIANTE, y (b) INSCRIPCIÓN

EID	Nombre	Especialidad	Grado
123	JONES	HISTORIA	JR
158	PARKS	HISTORIA	GR
105	ANDERSON	ADMINISTRACIÓN	SN
271	SMITH	HISTORIA	JR

(a)

Númerode-Estudiante	Nombrede-Clase	Númerode-Posición
123	H350	1
105	BA490	3
123	BA490	7

(b)

➤ FIGURA 8-16

Producto de las relaciones ESTUDIANTE e INSCRIPCIÓN en la figura 8-15

EID	Nombre	Especialidad	Grado	Númerode-Estudiante	Nombre-deClase	Número-dePosición
123	JONES	HISTORIA	JR	123	H350	1
123	JONES	HISTORIA	JR	105	BA490	3
123	JONES	HISTORIA	JR	123	BA490	7
158	PARKS	MATEMÁTICAS	GR	123	H350	1
158	PARKS	MATEMÁTICAS	GR	105	BA490	3
158	PARKS	MATEMÁTICAS	GR	123	BA490	7
105	ANDERSON	ADMINISTRACIÓN	SN	123	H350	1
105	ANDERSON	ADMINISTRACIÓN	SN	105	BA490	3
105	ANDERSON	ADMINISTRACIÓN	SN	123	BA490	7
271	SMITH	HISTORIA	JR	123	H350	1
271	SMITH	HISTORIA	JR	105	BA490	3
271	SMITH	HISTORIA	JR	123	BA490	7

SELECCIÓN. Así como el operador de proyección toma un subconjunto vertical (columnas) de una relación, el operador **selección** toma un subconjunto horizontal (renglón). Proyección identifica los *atributos* que serán incluidos en la nueva relación, y selección identifica los *tuples* que serán incluidos en la nueva relación. Selección se denota especificando el nombre de la relación, seguido por la palabra llave WHERE, y después por una condición que involucra atributos. La figura 8-18(a) muestra la selección de la relación ESTUDIANTE WHERE Especialidad = 'Matemáticas', y la figura 8-18(b) muestra la selección de ESTUDIANTE WHERE Grado = 'JR'.

JOIN. La operación **join** es una combinación del producto, selección y (posiblemente) operaciones de proyección. La asociación de dos relaciones, es decir, A y B, opera de la siguiente manera: primero forma el producto de A veces B. Después lleva a cabo una selección para eliminar algunos tuples (los criterios de selección se especifican como parte de la unión). Después (optativamente), se eliminan algunos atributos por medio de proyección.

Considere las relaciones ESTUDIANTE e INSCRIPCIÓN que se muestran en la figura 8-15. Suponga que queremos conocer Nombre y Número de Posición de cada estudiante. Para encontrarlos, necesitamos juntar los tuples de ESTUDIANTE acoplando con los de INSCRIPCIÓN basados en el EID. Denotamos a este join como ESTUDIANTE JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN. El significado de esta expresión es "Juntar UN tuple de ESTUDIANTE con un tuple de INSCRIPCIÓN si el EID de ESTUDIANTE es igual al NúmerodeEstudiante de INSCRIPCIÓN".

Para formar este join, primero encontramos el producto de ESTUDIANTE e INSCRIPCIÓN, una operación que se ha mostrado en la figura 8-16. A continuación SELECCIONamos los tuples a partir del producto donde EID de ESTUDIANTE es igual a NúmerodeEstudiante de INSCRIPCIÓN (sólo hay tres). Esta operación conduce a la re-

➤ FIGURA 8-17

Proyección de las relaciones ESTUDIANTE: (a) ESTUDIANTE [Nombre, Especialidad], y (b) ESTUDIANTE [Especialidad, Grado]

Nombre	Especialidad
JONES	HISTORIA
PARKS	MATEMÁTICAS
ANDERSON	ADMINISTRACIÓN
SMITH	HISTORIA

(a)

Especialidad	Grado
HISTORIA	JR
MATEMÁTICAS	GR
ADMINISTRACIÓN	SN

(b)

➤ FIGURA 8-18

Ejemplos de selección relacional: (a) ESTUDIANTE WHERE Especialidad = 'MATEMATICAS' y (b) ESTUDIANTE WHERE Grado = 'JR'

EID	Nombre	Especialidad	Grado
158	PARKS	MATEMÁTICAS	GR

(a)

EID	Nombre	Especialidad	Grado
123	JONES	HISTORIA	JR
271	SMITH	HISTORIA	JR

(b)

lación de la figura 8-19(a). Observe que dos atributos son idénticos: EID y NúmerodeEstudiante. Uno es redundante, así que lo eliminamos (en este caso escogemos NúmerodeEstudiante). El resultado es el join de la figura 8-19(b). El join en la figura 8-19(a) se llama el **equijoin de igualdad,** y el de la figura 8-19(b), **join natural.** A menos que se especifique lo contrario, cuando alguien dice join se refiere al join natural.

Debido a que formar el producto de dos grandes relaciones lleva tiempo, el algoritmo que usa un DBMS para join de dos relaciones es diferente al que se ha descrito aquí. Sin embargo, el resultado será idéntico.

Hacer un join en condiciones distintas a la igualdad también es posible. Por ejemplo, ESTUDIANTE JOIN (EID not = NúmerodeEstudiante) INSCRIPCIÓN, o ESTUDIANTE JOIN (EID < FID) FACULTAD. El último join dará como resultado tuples en los que los números de estudiante son menores que los números de facultad. Este join tendría significado si los IdentificadoresdePersonas estuvieran asignados en orden cronológico. Este join representaría pares de estudiantes y maestros, de tal forma que los estudiantes parecerían tener más tiempo en la institución que el maestro.

Existe una restricción importante en las condiciones de un join: los atributos en la condición deben provenir de un dominio común, de tal forma que ESTUDIANTE JOIN

➤ FIGURA 8-19

Ejemplos de relaciones de un join entre ESTUDIANTE e INSCRIPCIÓN: (a) equijoin, (b) join natural, y (c) left outer join

EID	Nombre	Especialidad	Grado	Número-de-Estudiante	Nombre-de-Clase	Número-de-Posición
123	JONES	HISTORIA	JR	123	H350	1
123	JONES	HISTORIA	JR	123	BA490	7
105	ANDERSON	ADMINISTRACIÓN	SN	105	BA490	3

(a)

EID	Nombre	Especialidad	Grado	Nombre-de-Clase	Número-de-Posición
123	JONES	HISTORIA	JR	H350	1
123	JONES	HISTORIA	JR	BA490	7
105	ANDERSON	ADMINISTRACIÓN	SN	BA490	3

(b)

EID	Nombre	Especialidad	Grado	Número-de-Estudiante	Nombre-de-Clase	Número-de-Posición
123	JONES	HISTORIA	JR	123	H350	1
123	JONES	HISTORIA	JR	123	BA490	7
158	PARKS	MATEMÁTICAS	GR	nulo	nulo	nulo
105	ANDERSON	ADMINISTRACIÓN	SN	105	BA490	3
271	SMITH	HISTORIA	JR	nulo	nulo	nulo

(c)

(Especialidad = CLASE.Nombre) CLASE es ilógico. Aun cuando el valor de Especialidad y CLASE.Nombre son Carac (10), no se desprenden del mismo dominio. Semánticamente, este tipo de join no tiene sentido (por desgracia muchos productos relacionales DBMS permiten este join).

OUTER JOIN. La operación join producirá una relación de los estudiantes y las clases que están tomando. Sin embargo, los estudiantes que no toman ninguna clase serán omitidos del resultado. Si queremos incluir a todos podemos usar un **outer join.** Así, ESTUDIANTE LEFT OUTER JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN incluirá cada renglón de estudiante. El resultado se muestra en la figura 8-19(c). El estudiante Smith quedó incluido aunque no está inscrito en ninguna clase. La palabra clave IZQUIERDA (LEFT) especifica que todos los renglones en la tabla del lado izquierdo de la expresión (ESTUDIANTE) aparecerán en el resultado. ESTUDIANTE RIGHT OUTER JOIN (EID = NúmerodeEstudiante), INSCRIPCIÓN especifica que todos los renglones en INSCRIPCIÓN están incluidos en el resultado. Los outer joins son útiles cuando funcionan en una relación en la que la mínima cardinalidad es cero en uno o ambos lados. Cuando puede surgir ambigüedad entre los dos tipos de join, a veces se usa el término INNER JOIN en lugar de JOIN.

Expresión de consultas en álgebra relacional

La figura 8-20 resume las operaciones básicas relacionales recién analizadas. El conjunto de operaciones estándar incluye +, −, intersección y producto. Selección escoge tuples específicos (renglones) de una relación, de acuerdo con las condiciones de valores de atributo. Proyección escoge atributos específicos (columnas) de una relación, mediante el significado del nombre de atributo. Por último, join concatena los tuples de dos relaciones de acuerdo con una condición sobre los valores de los atributos.

Ahora veamos cómo se pueden usar los operadores relacionales para expresar consultas usando las relaciones ESTUDIANTE, CLASE e INSCRIPCIÓN de la figura 8-11; la figura 8-21 incluye datos muestra. Nuestro propósito es demostrar el uso de relaciones. Aunque probablemente nunca usará el álgebra relacional en un ambiente comercial, estos ejemplos le ayudarán a entender cómo se pueden manejar las relaciones.

➤ FIGURA 8-20

Resumen de operaciones de álgebra relacional

Tipo	*Formato*	*Ejemplo*
Conjunto de operaciones	+, –, intersección, producto	ESTUDIANTE [Nombre] – JUNIOR [Nombre]
Selección	Relación WHERE condición	CLASE WHERE Nombre = 'A'
Proyección	relación [lista de atributos]	ESTUDIANTE [Nombre, Especialidad]
Join	relación 1 JOIN (condición) relación 2	ESTUDIANTE JOIN (EID = Número-de-Estudiante) INSCRIPCIÓN
Inner Join	Sinónimo de join	
Outer Join	relación 1 LEFT OUTER JOIN (condición) relación 2	ESTUDIANTE LEFT OUTER JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN
	o	
	relación 1 RIGHT OUTER JOIN (condición) relación 2	ESTUDIANTE RIGHT OUTER JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN

➤ FIGURA 8-21

Datos de ejemplo para las relaciones definidas en la figura 8-11: (a) relación ESTUDIANTE, (b) relación INSCRIPCIÓN, y (c) relación CLASE

EID	Nombre	Especialidad	Grado
100	JONES	HISTORIA	GR
150	PARKS	CONTABILIDAD	SO
200	BAKER	MATEMÁTICAS	GR
250	GLASS	HISTORIA	SN
300	BAKER	CONTABILIDAD	SN
350	RUSSELL	MATEMÁTICAS	JR
400	RYE	CONTABILIDAD	FR
450	JONES	HISTORIA	SN

(a)

Númerode-Estudiante	Nombre-deClase	Númerode-Posición
100	BD445	1
150	BA200	1
200	BD445	2
200	CS250	1
300	CS150	1
400	BA200	2
400	BF410	1
400	CS250	2
450	BA200	3

(b)

Nombre	Tiempo	Aula
BA200	M-F9	SC110
BD445	MWF3	SC213
BF410	MWF8	SC213
CS150	MWF3	EA304
CS250	MWF12	EB210

(c)

1. ¿Cuáles son los nombres de todos los estudiantes?

   ESTUDIANTE [Nombre]

   Ésta es sólo la proyección del atributo Nombre de la relación ESTUDIANTE y el resultado es

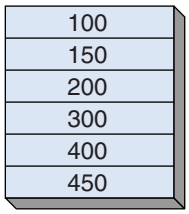
JONES
PARKS
BAKER
GLASS
RUSSELL
RYE

   Los nombres duplicados han sido omitidos. Aunque los nombres JONES y BAKER realmente aparecen dos veces en la relación ESTUDIANTE, se omitieron las repeticiones porque el resultado de una proyección es una relación, y las relaciones no pueden tener tuples duplicados.

2. ¿Cuáles son los números de todos los estudiantes inscritos en una clase?

   INSCRIPCIÓN [NúmerodeEstudiante]

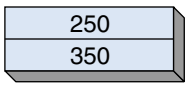
   Ésta es similar a la primera consulta, pero la proyección ocurre en la relación INSCRIPCIÓN. El resultado es

   Otra vez se omitieron los tuples duplicados.
3. ¿Cuáles son los números de estudiante de todos los que no se han inscrito en una clase?

   ESTUDIANTE[EID] - INSCRIPCIÓN [NúmerodeEstudiante]

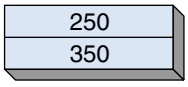
   Esta expresión encuentra la diferencia en la proyección de las dos relaciones: ESTUDIANTE [EID] tiene los números de todos los estudiantes, e INSCRIPCIÓN [NúmerodeEstudiante] tiene los números de todos que están inscritos en una clase. La diferencia es el número de los que no están inscritos. El resultado es

4. ¿Cuáles son los números de estudiantes inscritos en la clase 'BD445'?

   INSCRIPCIÓN WHERE NombredeClase = 'BD445' [NúmerodeEstudiante]

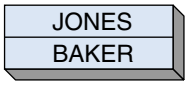
   Esta expresión selecciona los tuples apropiados y luego los proyecta en el atributo NúmerodeEstudiante. El resultado es

5. ¿Cuáles son los nombres de los estudiantes inscritos en la clase 'BD445'?

   ESTUDIANTE JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN WHERE NombredeClase = 'BD445' [ESTUDIANTE.Nombre]

   Para responder a esta consulta, se necesitan tanto los datos de ESTUDIANTE como los de INSCRIPCIÓN. Específicamente, los nombres de estudiantes deben provenir de ESTUDIANTE, mientras que la condición "inscripción en BD445" se debe verificar en INSCRIPCIÓN. Debido a que ambas relaciones son necesarias se deben juntar. Después de haber juntado ESTUDIANTE e INSCRIPCIÓN, se aplica la selección y posteriormente una proyección de los nombres de estudiante. El resultado es

   Como ya lo establecimos, cuando se consideran dos o más relaciones se pueden duplicar los nombres de los atributos. Por lo tanto, para aclarar, el nombre de la relación se puede anteponer al nombre del atributo. Así, en nuestro ejemplo, la proyección es sobre [ESTUDIANTE.Nombre]. En este ejemplo se añadió este prefijo só-

lo por claridad, ya que los nombres de los atributos son diferentes; pero cuando son idénticos (un join que involucra ESTUDIANTE y CLASE produce dos atributos, ambos llamados Nombre) se requiere el prefijo. Considere la siguiente consulta:

6. ¿Cuáles son los nombres y los horarios de las clases del estudiante 'PARKS'?

Para responder esta pregunta debemos juntar los datos en las tres relaciones. Necesitamos los datos ESTUDIANTE para encontrar el número de PARKS; los datos de INSCRIPCIÓN para saber en qué clases está inscrito, y los datos de CLASE para determinar los horarios de las clases a las que asiste.

ESTUDIANTE WHERE Nombre = 'PARKS' JOIN (EID = NúmerodeEstudiante) INSCRIPCIÓN JOIN (NombredeClase = Nombre) CLASE [CLASE.Nombre, Horario]

Esta expresión primero selecciona el tuple PARKS y lo junta con los tuples correspondientes de INSCRIPCIÓN. Después el resultado se junta con los tuples correspondiente de CLASE. Por último, se toma la proyección para imprimir clases y horarios. El resultado es

BA200	M-F9

Debemos especificar CLASE.Nombre; el hecho de sólo especificar Nombre es ambiguo porque tanto ESTUDIANTE como CLASE tienen un atributo llamado Nombre.

Existen otras formas equivalentes de responder a esta consulta. Una es

JOIN ESTUDIANTE (EID = NúmerodeEstudiante) INSCRIPCIÓN JOIN (NombredeClase = Nombre) CLASE WHERE ESTUDIANTE.Nombre = 'PARKS' [CLASE.Nombre, Horario]

Esta expresión difiere de la primera porque la selección de PARKS no se efectúa hasta después que se realizaron todas las juntas. Suponga que la computadora lleva a cabo las operaciones como se establecen; esta expresión será más lenta que la primera porque se juntan muchos tuples.

Estas diferencias son la mayor desventaja del álgebra relacional. Para el usuario, dos consultas equivalentes podrían tomar el mismo tiempo (y por tanto el mismo costo). Imagine la frustración si una de las formas de consulta costara $1.17 y la otra $4356. Para un usuario incauto y nada sofisticado la diferencia en el costo parecerá un capricho.

7. ¿Cuáles son los grados y las aulas de todos los estudiantes, incluyendo a los que no están inscritos en una clase?

Debido a que todos deberán ser incluidos, esta consulta requiere el uso de un OUTER JOIN. La sintaxis es directa:

ESTUDIANTE LEFT OUTER JOIN (EID = NúmerodeEstudiante) JOIN INSCRIPCIÓN (NombredeClase = Nombre) CLASE [Grado, Aula]

El resultado incluye el Grado de Glass y de Russell, quienes no están inscritos en ninguna clase.

GR	SC213
SO	SC110
GR	EB210
SN	Nulo
SN	EA304
JR	Nulo
FR	SC110
FR	SC213
FR	EB210
SN	SC110

➤ RESUMEN

Cuando se implementa una base de datos relacional hay que llevar a cabo varias tareas. Primero, se debe definir la estructura de la base de datos en el DBMS. Después, ubicar el espacio de archivo, y por último, llenar la base con los datos.

El modelo relacional representa y procesa datos en forma de tablas llamadas relaciones. Las columnas de la tabla se llaman atributos y los renglones tuples. Los valores de los atributos surgen de dominios. Los términos *tabla*, *columna* y *renglón,* y *archivo*, *campo* y *registro* se usan como sinónimos de los términos *relación*, *atributo* y *tuple,* respectivamente.

El uso del término *llave* puede ser confuso porque se emplea de manera diferente en las etapas de diseño e implementación. Durante el diseño, el término significa una llave lógica, la cual es uno o más atributos que únicamente definen un renglón. Durante la implementación, el término significa una llave física, que es una estructura de datos usada para mejorar el desempeño. Una llave lógica puede ser o no una llave física, y una llave física puede ser o no una llave lógica. En este texto usaremos llave para significar una llave lógica, e índice para referirnos a una llave física.

Debido a que estamos usando el modelo relacional para expresar diseños de bases de datos, no es necesario transformar el diseño durante la etapa de implementación. Sólo definimos el diseño relacional en el DBMS. Dos formas de definirlo son: expresarlo en un archivo de texto DDL y usar una herramienta gráfica para la definición de datos. En cualquier caso, las tablas, columnas, índices, restricciones, contraseñas y otros controles están definidos en el DBMS.

Además de establecer la estructura de la base de datos, los programadores deben asignar espacio medio para la base de datos. Con sistemas multiusuario, esta asignación puede ser importante para el desempeño eficaz del DBMS. Por último, se llena la base con datos usando las herramientas que proporcionan los fabricantes del DBMS, los programas que éstos han desarrollado, o ambos.

Las cuatro categorías de lenguajes de manejo de datos relacionales son el álgebra relacional, el cálculo relacional, los lenguajes orientados a la transformación, y el Query-by-Example. El álgebra relacional consta de un grupo de operadores relacionales que se pueden usar para manejar relaciones y así obtener el resultado deseado. El álgebra relacional está basada en procedimientos. Los lenguajes orientados a la transformación ofrecen un medio que no está basado en procedimientos para transformar un conjunto de relaciones en un resultado deseado. SQL es el ejemplo más común.

Existen tres medios para acceder a una base de datos relacional: usar las formas y reportes que proporciona el DBMS; usar un lenguaje de consulta-actualización, del cual el SQL es el más común, y finalmente a través de programas de aplicación.

Las interfaces de los programas de aplicación pueden ser mediante llamadas de función, métodos de objeto o comandos de base de datos de propósito especial que se traducen por medio de un precompilador. La orientación al procesamiento del modelo relacional es una relación a la vez, pero la orientación de la mayoría de los lenguajes de programación es un renglón a la vez. Se deben desarrollar algunos medios para corregir este desajuste.

El álgebra relacional se usa para manejar relaciones con el fin de obtener el resultado deseado. Los operadores son la unión, la diferencia, la intersección, el producto, la proyección, la selección, el join (interno) y el join externo.

➤ PREGUNTAS DEL GRUPO I

8.1 Nombre y describa las tres tareas necesarias para implementar una base de datos relacional.

8.2 Defina *relación*, *atributo*, *tuple* y *dominio*.

8.3 Explique el uso de los términos *tabla*, *columna*, *renglón*, *archivo*, *campo* y *registro*.

8.4 Explique la diferencia entre un esquema relacional y una relación.

8.5 Defina *llave*, *índice*, *llave lógica* y *llave física*.

8.6 Mencione tres razones para usar índices.

8.7 ¿Bajo qué condiciones es necesario transformar el diseño de datos durante la etapa de implementación?

8.8 Explique el término *lenguaje de definición de datos*. ¿Para qué sirve?

8.9 ¿Cómo puede definirse una estructura de base de datos en forma distinta de la de un archivo de texto?

8.10 ¿Qué aspectos del diseño de base de datos se necesita definir en el DBMS?

8.11 Dé un ejemplo, diferente al de este texto, en el cual la asignación de la base de datos a medios físicos sea importante.

8.12 Describa los extremos mejor y peor para cargar una base de datos.

8.13 Nombre y explique brevemente las cuatro categorías de DML relacional.

8.14 Describa cómo se pueden manejar los datos relacionales por medio de formas.

8.15 Explique el papel que desempeñan los lenguajes de consulta en el manejo de los datos relacionales. ¿Cómo difieren las consultas almacenadas de los programas de aplicación? ¿Por qué se usan?

8.16 Describa los dos estilos de interfaz de programas de aplicación a la base de datos. En su respuesta explique el papel del precompilador.

8.17 Describa el desajuste entre la orientación del SQL y la orientación de la mayoría de los lenguajes de programación. ¿Cómo se corrige ese desajuste?

8.18 ¿Cuál es la diferencia entre el álgebra relacional y el álgebra de la escuela preparatoria?

8.19 ¿Por qué el álgebra relacional es *cerrada*?

8.20 Defina *compatibilidad de unión*. Dé un ejemplo sobre dos relaciones que estén en compatibilidad de unión y dos que estén en incompatibilidad de unión.

Las preguntas 8.21 a la 8.23 se refieren a las siguientes relaciones:

COMPAÑÍA (Nombre, NúmerodeEmpleados, Ventas)
FABRICANTES (Nombre, TotaldePersonas, Ingreso)

8.21 Exponga un ejemplo sobre la unión de estas dos relaciones.

8.22 Proporcione un ejemplo que muestre la diferencia de esas dos relaciones.

8.23 Mencione un ejemplo de la intersección de esas dos relaciones.

Las preguntas 8.24 a 8.28 se refieren a las tres relaciones siguientes:

VENDEDOR (Nombre, Salario)
PEDIDO (Número, NombredelCliente, NombredelVendedor, Cantidad)
CLIENTE (Nombre, Ciudad, TipodeIndustria)

Una prueba de estas relaciones se muestra en la figura 8-22. Use los datos de estas tablas para los siguientes problemas:

8.24 Mencione un ejemplo del producto de VENDEDOR y PEDIDO.

8.25 Muestre un ejemplo de:

VENDEDOR [Nombre, Salario]
VENDEDOR [Salario]

¿Bajo qué condiciones el VENDEDOR [Salario] tendrá menos renglones que VENDEDOR?

8.26 Muestre un ejemplo de una selección en VENDEDOR Nombre, VENDEDOR Salario y en ambas, VENDEDOR Nombre y Salario.

➤ FIGURA 8-22

Datos de muestra para las preguntas 8.24 a 8.28

Nombre	Salario
Abel	120,000
Baker	42,000
Jones	36,000
Murphy	50,000
Zenith	118,000
Kobad	34,000

VENDEDOR

Número	NombredelCliente	NombredelVendedor	Cantidad
100	Abernathy Construction	Zenith	560
200	Abernathy Construction	Jones	1800
300	Manchester Lumber	Abel	480
400	Amalgamated Housing	Abel	2500
500	Abernathy Construction	Murphy	6000
600	Tri-City Builders	Abel	700
700	Manchester Lumber	Jones	150

PEDIDO

Nombre	Ciudad	TipodeIndustria
Abernathy Construction	Willow	B
Manchester Lumber	Manchester	F
Tri-City Builders	Memphis	B
Amalgamated Housing	Memphis	B

CLIENTE

8.27 Muestre un ejemplo de un equijoin y un join natural de VENDEDOR y PEDIDO en el cual Nombre del VENDEDOR sea igual a NombredelVendedor en PEDIDO.

8.28 Muestre las expresiones del álgebra relacional para

a. Los nombres de todos los vendedores

b. Los nombres de todos los vendedores que tienen un renglón PEDIDO

c. Los nombres de los vendedores que no tienen un renglón PEDIDO

d. Los nombres de los vendedores que tienen un PEDIDO con Abernathy Construction

e. Los salarios de los vendedores que tienen un PEDIDO con Abernathy Construction

f. La ciudad de todos los CLIENTES que tienen un PEDIDO con el vendedor Jones

g. Los nombres de todos los vendedores con los nombres de los clientes que les han hecho pedidos. Incluya a los vendedores que no tienen pedidos

CAPÍTULO 9

Lenguaje de consulta estructurado

El lenguaje de consulta estructurado, o SQL (Structured Query Language), es el lenguaje de manejo de datos relacionales actual más importante. Ha recibido el respaldo del American National Standards Institute (ANSI) como el lenguaje seleccionado para el manejo de bases de datos relacionales, y es el lenguaje de acceso a datos que usan muchos productos DBMS comerciales, incluyendo DB2, SQL/DS, Oracle, INGRES, SYBASE, SQL Server, dBase para Windows, Paradox, Microsoft Access y muchos otros. Debido a su popularidad, SQL ha sido el lenguaje estándar para el intercambio de información entre computadoras. Puesto que hay una versión SQL que puede funcionar en casi todas las computadoras y sistemas operativos, los sistemas de cómputo pueden intercambiar datos, consultas y respuestas.

El desarrollo del SQL empezó en las instalaciones de investigación de IBM San José, a mediados de la década de 1970 con el nombre de SEQUEL. Salieron varias versiones de SEQUEL y en 1980 el producto se denominó SQL. Desde entonces, IBM se ha unido a muchos otros proveedores en el desarrollo de productos SQL. El American National Standards Institute se ha dado a la tarea de mantener SQL y periódicamente publica versiones actualizadas de la norma SQL. Este capítulo analiza el núcleo de SQL como se describe en la norma ANSI de 1992, que con frecuencia se conoce como SQL92[1]. La versión más reciente, SQL3, se refiere a las extensiones del lenguaje de programación orientada a objetos. Esa versión es analizada en el capítulo 18.

Las construcciones y expresiones en una implementación particular de SQL (por ejemplo, en Oracle o en el SQL Server) pueden diferir un poco de la norma ANSI, en parte porque muchos de los productos DBMS se desarrollaron antes de que hubiera un acuerdo sobre la norma, y también debido a que los proveedores agregaron capacidades a sus productos para que fueran más competitivos. Desde una perspectiva de mercadotecnia, a la norma ANSI a veces se le juzgó poco interesante.

Las órdenes de SQL se pueden usar interactivamente como un lenguaje de consulta, o se pueden insertar en programas de aplicación. De esta manera, SQL no es un lenguaje de programación (como COBOL), sino un *sublenguaje de datos*, o un *lenguaje de acceso de datos* que se implanta en otros lenguajes.

[1] Revista de la Organización Internacional de Normas ISO/IEC 9075: 1992, *Database Language SQL*.

En este capítulo presentamos enunciados interactivos SQL que necesitan ser ajustados y modificados cuando se implantan en programas, como se muestra en los capítulos 12 y 13. El presente capítulo se refiere sólo a los enunciados para el manejo de datos; en los capítulos 12 y 13 se analizan los enunciados de definición de datos.

SQL es un lenguaje orientado a la transformación que acepta como entrada una o más relaciones y produce una sola relación de salida. El resultado de cada consulta SQL es una relación; incluso si el resultado es un número independiente, ese número se considera como una relación con un solo renglón y una sola columna. Por lo tanto, SQL es como el álgebra relacional: *cerrado*.

➤ CONSULTA DE UNA SOLA TABLA

En esta sección consideramos las facilidades SQL para consultar una tabla independiente. Más adelante analizaremos tablas múltiples y enunciados de actualización. Por costumbre, las palabras reservadas de SQL tales como SELECT y FROM están escritas con letras mayúsculas. También, los enunciados SQL por lo general se escriben en líneas múltiples, como se muestra en este capítulo. Sin embargo, los compiladores de lenguaje SQL no requieren ni letras mayúsculas ni líneas múltiples. Estas convenciones se usan sólo para proporcionar una claridad especial a quienes leen los enunciados SQL.

Usamos el mismo conjunto de seis relaciones con las que ilustramos el álgebra relacional en el capítulo 8. La estructura de estas relaciones se ejemplifica en la figura 9-1 y los datos de muestra para tres de ellas aparecen en la figura 9-2.

➤ FIGURA 9-1

Relaciones usadas para los ejemplos SQL

1. JUNIOR (Enum, Nombre, Especialidad)
2. ESTUDIANTE-HONOR (Número, Nombre, Interés)
3. ESTUDIANTE (EID, Nombre, Especialidad, Grado)
4. CLASE (Nombre, Horario, Aula)
5. INSCRIPCIÓN (NúmerodeEstudiante, NombredeClase, NúmerodePosición)
6. FACULTAD (FID, Nombre, Departamento)

	Atributo	*Dominio*
1.	Enum	IdentificadoresdePersonas
	Nombre.JUNIOR	NombresdePersonas
	Especialidad	NombresdeTemas
2.	Número	IdentificadoresdePersonas
	Nombre.ESTUDIANTE-HONOR	NombresdePersonas
	Interés	NombresdeTemas
3.	EID	IdentificadoresdePersonas
	ESTUDIANTE.Nombre	NombresdePersonas
	Especialidad	NombresdeTemas
	Grado	Clases
4.	CLASE.Nombre	NombresdeClases
	Tiempo	HorariodeClases
	Aula	Aulas
5.	NúmerodeEstudiante	IdentificadoresdePersonas
	NombredeClase	NombresdeClases
	NúmerodePosición	TamañosdeClases
6.	FID	IdentificadoresdePersonas
	FACULTAD.Nombre	NombresdePersonas
	Departamento	NombresdeTemas

➤ FIGURA 9-2

Datos de muestra usados en los ejemplos SQL: (a) relación ESTUDIANTE, (b) relación INSCRIPCIÓN, y (c) relación CLASE

EID	Nombre	Especialidad	Grado
100	JONES	HISTORIA	GR
150	PARKS	CONTABILIDAD	SO
200	BAKER	MATEMÁTICAS	GR
250	GLASS	HISTORIA	SN
300	BAKER	CONTABILIDAD	SN
350	RUSSELL	MATEMÁTICAS	JR
400	RYE	CONTABILIDAD	FR
450	JONES	HISTORIA	SN

(a)

Númerode-Estudiante	Nombre-deClase	Número-dePosición
100	BD445	1
150	BA200	1
200	BD445	2
200	CS250	1
300	CS150	1
400	BA200	2
400	BF410	1
400	CS250	2
450	BA200	3

(b)

Nombre	Horario	Aula
BA200	M-F9	SC110
BD445	MWF3	SC213
BF410	MWF8	SC213
CS150	MWF3	EA304
CS250	MWF12	EB210

(c)

Proyecciones utilizando SQL

Para formar una proyección con SQL nombramos la relación a proyectarse y listamos las columnas que van a ser mostradas. Utilizando la sintaxis estándar SQL, la proyección ESTUDIANTE [EID, Nombre, Especialidad] se especifica como

```
SELECT    EID, Nombre, Especialidad
FROM      ESTUDIANTE
```

Las palabras reservadas SELECT y FROM siempre se requieren; las columnas a obtener se listan después de la palabra reservada SELECT, y la tabla que se va a usar se enumera después de la palabra reservada FROM. El resultado de esta proyección para los datos de la figura 9-2 es

100	JONES	HISTORIA
150	PARKS	CONTABILIDAD
200	BAKER	MATEMÁTICAS
250	GLASS	HISTORIA
300	BAKER	CONTABILIDAD
350	RUSSELL	MATEMÁTICAS
400	RYE	CONTABILIDAD
450	JONES	HISTORIA

No confunda la palabra reservada SELECT con el operador de selección del álgebra relacional. SELECT es un verbo SQL que se puede usar para realizar una proyección de álgebra relacional, seleccionar, y especificar otras acciones. Por otra parte, Selección difiere de SELECT porque es la operación de álgebra relacional para obtener un subconjunto de renglones de una tabla.

Considere otro ejemplo:

```
SELECT      Especialidad
FROM        ESTUDIANTE
```

El resultado de esta operación es el siguiente:

HISTORIA
CONTABILIDAD
MATEMÁTICAS
HISTORIA
CONTABILIDAD
MATEMÁTICAS
CONTABILIDAD
HISTORIA

Como puede ver, esta tabla contiene renglones duplicados y, en consecuencia, en un sentido estricto esta tabla no es una relación. De hecho, SQL no elimina automáticamente los duplicados porque puede tomar mucho tiempo y en muchos casos no es deseable ni necesario.

Si se deben eliminar los renglones duplicados el calificador DISTINCT debe ser especificado así:

```
SELECT      DISTINCT Especialidad
FROM        ESTUDIANTE
```

El resultado de esta operación es la relación:

HISTORIA
CONTABILIDAD
MATEMÁTICAS

Selecciones usando SQL

El operador de selección de álgebra relacional también se lleva a cabo con la orden SQL SELECT. Un ejemplo de esto es:

```
SELECT      EID, Nombre, Especialidad, Grado
FROM        ESTUDIANTE
WHERE       Especialidad = 'MATEMÁTICAS'
```

Esta expresión SELECT especifica los nombres de todas las columnas de las tablas. FROM especifica la tabla que hay que usar, y la nueva frase, WHERE, proporciona la(s) condición(es) para la selección. El formato SELECT-FROM-WHERE es la estructura fundamental de los enunciados SQL. La siguiente es una forma equivalente de la consulta anterior:

```
SELECT      *
FROM        ESTUDIANTE
WHERE       Especialidad = 'MATEMÁTICAS'
```

El asterisco (*) significa que deben obtenerse todas las columnas de la tabla. El resultado de ambas consultas es:

200	BAKER	MATEMÁTICAS	GR
350	RUSSELL	MATEMÁTICAS	JR

Podemos combinar la selección y la proyección como sigue:

```
SELECT    Nombre, Grado
FROM      ESTUDIANTE
WHERE     Especialidad = 'MATEMÁTICAS'
```

El resultado es:

BAKER	GR
RUSSELL	JR

Se pueden expresar varias condiciones en la cláusula WHERE. Por ejemplo, con la expresión:

```
SELECT    Nombre, Grado
FROM      ESTUDIANTE
WHERE     Especialidad = 'MATEMÁTICAS' y Grado = 'GR'
```

se obtiene lo siguiente:

BAKER	GR

Las condiciones en las cláusulas WHERE se pueden referir a un conjunto de valores. Para hacer esto, se pueden usar las palabras reservadas IN o NOT IN. Considere:

```
SELECT    Nombre
FROM      ESTUDIANTE
WHERE     Especialidad IN ['MATEMÁTICAS', 'CONTABILIDAD']
```

Observe que dentro de los corchetes se pueden colocar valores múltiples. Esta expresión significa: "Desplegar los nombres de los estudiantes que tienen una especialidad, ya sea en matemáticas o en contabilidad". El resultado es:

PARKS
BAKER
BAKER
RUSSELL
RYE

La expresión:

```
SELECT    Nombre
FROM      ESTUDIANTE
WHERE     Especialidad NOT IN ['MATEMÁTICAS', 'CONTABILIDAD']
```

hace que aparezcan los nombres de estudiantes que no tienen especialidad en matemáticas o contabilidad. El resultado es:

JONES
GLASS
JONES

La expresión ESPECIALIDAD IN significa que el valor de la columna Especialidad puede ser igual a *cualquiera* de las especialidades listadas. Esto es equivalente al operador lógico OR. La expresión ESPECIALIDAD NOT IN significa que el valor debe ser diferente a *todas* las especialidades listadas.

Las cláusulas WHERE también se pueden referir a rangos y valores parciales. La palabra reservada BETWEEN se usa para los rangos. Por ejemplo, con el enunciado:

```
SELECT      Nombre, Especialidad
FROM        ESTUDIANTE
WHERE       EID BETWEEN 200 AND 300
```

se obtendrá el siguiente resultado:

BAKER	MATEMÁTICAS
GLASS	HISTORIA
BAKER	CONTABILIDAD

Esta expresión equivale a:

```
SELECT      Nombre, Especialidad
FROM        ESTUDIANTE
WHERE       EID >= 200 AND EID <= 300
```

Por lo tanto, los valores finales de BETWEEN (200 y 300 en este caso) están incluidos en el rango seleccionado.

La palabra reservada LIKE se usa en expresiones SQL para seleccionar valores parciales. El símbolo_ (guión bajo) representa un carácter independiente no especificado; el símbolo % representa una serie de uno o más caracteres no especificados. Así, el resultado de la expresión:

```
SELECT      Nombre, Grado
FROM        ESTUDIANTE
WHERE       Grado LIKE '_R'
```

es una relación que tiene las columnas Nombre y Grado, donde Grado consta de dos caracteres, el segundo de los cuales es R:

JONES	GR
BAKER	GR
RUSSELL	JR
RYE	FR

De manera similar, la siguiente expresión encontrará a los estudiantes cuyos apellidos terminen con S:

```
SELECT      Nombre
FROM        ESTUDIANTE
WHERE       Nombre LIKE '%S'
```

el resultado es:

JONES
PARKS
GLASS
JONES

Cabe señalar que Microsoft Access usa un conjunto de símbolos comodines diferentes a la norma ANSI. Un "?" se usa en lugar del guión bajo, y un "*" se emplea en lugar de "%".

Por último, las palabras reservadas IS NULL se usan para buscar valores nulos (o faltantes). Con la expresión:

```
SELECT    Nombre
FROM      ESTUDIANTE
WHERE     Grado IS NULL
```

se obtendrán los nombres de los estudiantes que no tienen un valor registrado de Grado. Para los datos de la figura 9-2 todos los estudiantes tienen un Grado y esta expresión regresará una relación sin renglones.

Ordenamiento

Los renglones del resultado de la relación se pueden ordenar por los valores de una o más columnas. Considere el siguiente ejemplo:

```
SELECT    Nombre, Especialidad, Grado
FROM      ESTUDIANTE
WHERE     Especialidad = 'CONTABILIDAD'
ORDER BY  Nombre
```

Esta consulta listará las especialidades de contabilidad en secuencia ascendente por el valor del nombre. El resultado es:

BAKER	CONTABILIDAD	SN
PARKS	CONTABILIDAD	SO
RYE	CONTABILIDAD	FR

Se puede elegir más de una columna para el ordenamiento. Si es así, la primera columna listada será el campo ordenado de la especialidad; la segunda, el siguiente campo ordenado de la especialidad, y así sucesivamente. Las columnas también pueden declararse como ascendentes (ASC) o descendentes (DESC), como se muestra en el siguiente enunciado:

```
SELECT    Nombre, Especialidad, Grado
FROM      ESTUDIANTE
WHERE     Grado IN ['FR', 'SO', 'SN']
ORDER BY  Especialidad ASC, Grado DESC
```

El resultado es:

PARKS	CONTABILIDAD	SO
BAKER	CONTABILIDAD	SN
RYE	CONTABILIDAD	FR
GLASS	HISTORIA	SN
JONES	HISTORIA	SN

ORDER BY se puede combinar con cualquiera de los enunciados SELECT.

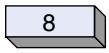
Funciones SQL interconstruidas

SQL proporciona cinco funciones interconstruidas: COUNT (contar), SUM (sumar), AVG (promediar), MAX (obtener el valor máximo) y MIN[2] (obtener el valor mínimo). Aunque COUNT y SUM suenan similares, son realmente diferentes. COUNT calcula el número de renglones en una tabla, mientras que SUM totaliza las columnas numéricas, AVG, MAX y MIN también operan en columnas numéricas: AVG calcula el valor promedio, y MAX y MIN obtienen los valores máximos y mínimos de una columna en una tabla.

La expresión de consulta:

```
SELECT    COUNT(*)
FROM      ESTUDIANTE
```

cuenta el número de renglones ESTUDIANTE y despliega ese total en una tabla con un renglón independiente y una sola columna:



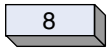
Considere las expresiones:

```
SELECT    COUNT(Especialidad)
FROM      ESTUDIANTE
```

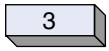
y

```
SELECT    COUNT (DISTINCT Especialidad)
FROM      ESTUDIANTE
```

La primera expresión cuenta todas las especialidades, incluyendo duplicados, y la segunda cuenta sólo las especialidades únicas. Los resultados son:



y



respectivamente.

Con excepción de GROUP BY (que analizaremos a continuación), las funciones interconstruidas no se pueden mezclar con los nombres de las columnas en el enunciado SELECT. Así,

```
SELECT    Nombre, COUNT (*)
```

no está permitido.

Las funciones interconstruidas se pueden usar para solicitar un resultado, como en los ejemplos anteriores. En la mayoría de las implementaciones de SQL y en la norma ANSI de SQL, las funciones interconstruidas *no se pueden* usar como parte de una cláusula WHERE.

Funciones interconstruidas y de agrupamiento

Para aumentar su utilidad, se pueden aplicar funciones interconstruidas a grupos de renglones dentro de una tabla. Estos grupos se forman uniendo esos renglones (en forma lógica, no física) que tengan el mismo valor de una columna específica. Por ejemplo, se

[2] A veces las funciones construidas se conocen como funciones agregadas para distinguirlas de las funciones interconstruidas de lenguajes de programación tales como SUBSTRING.

pueden agrupar los estudiantes por especialidad, lo que significa que un grupo estará formado por cada valor de ESPECIALIDAD. Para los datos de la figura 9-2, hay un grupo de estudiantes de HISTORIA, otro de CONTABILIDAD, y otro más de MATEMÁTICAS.

La palabra reservada de SQL: GROUP BY, instruye al DBMS a que agrupe esos renglones que tengan el mismo valor de una columna. Considere:

```
SELECT      Especialidad, COUNT (*)
FROM        ESTUDIANTE
GROUP BY    Especialidad
```

El resultado de esta expresión es:

HISTORIA	3
CONTABILIDAD	3
MATEMÁTICAS	2

Los renglones de la tabla ESTUDIANTE se han agrupado en forma lógica por el valor de ESPECIALIDAD, y la función COUNT suma el número de renglones en cada grupo. El resultado es una tabla con dos columnas, el nombre de la especialidad y la suma. Para los subgrupos se pueden especificar las columnas y las funciones interconstruidas en el enunciado SELECT.

En algunos casos no queremos considerar todos los grupos. Por ejemplo, podríamos formar grupos de estudiantes con la misma especialidad y considerar sólo aquellos grupos que tengan más de dos estudiantes. En este caso, usaríamos la cláusula SQL HAVING para identificar el subconjunto de grupos que deseamos considerar.

Los siguientes enunciados SQL pueden listar las especialidades que tengan más de dos estudiantes y también la cuenta de estudiantes en cada una de las especialidades.

```
SELECT      Especialidad, COUNT (*)
FROM        ESTUDIANTE
GROUP BY    Especialidad
HAVING      COUNT (*) > 2
```

Aquí se forman grupos de estudiantes con la misma especialidad y a continuación se seleccionan los grupos que tienen más de dos estudiantes (los demás grupos se ignoran). La especialidad y la cuenta de estudiantes se produce en estos grupos seleccionados. El resultado es:

HISTORIA	3
CONTABILIDAD	3

Para una generalidad aun mayor se pueden agregar las cláusulas WHERE. Sin embargo, al hacerlo se pueden generar ambigüedades. Por ejemplo,

```
SELECT      Especialidad, MAX (EID)
FROM        ESTUDIANTE
WHERE       Grado = 'SN'
GROUP BY    Especialidad
HAVING      COUNT (*) > 1
```

El resultado de esta expresión diferirá dependiendo de si la condición WHERE se aplica o no antes o después de la condición HAVING. Para eliminar esta incertidumbre, la norma SQL especifica que las cláusulas WHERE deberán ser aplicadas primero. En concordancia, en los enunciados anteriores las operaciones son: seleccionar a los estudiantes graduados; formar los grupos; seleccionar los grupos que cumplan la condición HAVING; desplegar los resultados. En este caso, el resultado es:

HISTORIA	450

(No hay que perder de vista que esta consulta no es válida para todas las implementaciones SQL. Para algunas, los únicos atributos que pueden aparecer en la frase SELECT de una consulta con GROUP BY son los atributos que pueden aparecer en la frase GROUP BY y las funciones interconstruidas de esos atributos. Así, en esta consulta sólo serían permitidas ESPECIALIDAD y las funciones interconstruidas de ESPECIALIDAD.)

➤ CONSULTAS DE TABLAS MÚLTIPLES

En esta sección ampliamos nuestro análisis de SQL para incluir operaciones en dos o más tablas. Los datos de la figura 9-2 ESTUDIANTE, CLASE, e INSCRIPCIÓN se utilizan para ilustrar estos comandos SQL.

RECUPERACIÓN USANDO UNA SUBCONSULTA

Suponga que necesitamos saber los nombres de aquellos estudiantes inscritos en la clase BD445. Si sabemos que los estudiantes con EID de 100 y 200 están inscritos en esa clase, lo siguiente producirá los nombres correctos:

```
SELECT    Nombre
FROM      ESTUDIANTE
WHERE     EID IN [100, 200]
```

En general no conocemos los EID de los estudiantes en una clase, pero tenemos los medios para encontrarlos. Examine la expresión:

```
SELECT    NúmerodeEstudiante
FROM      INSCRIPCIÓN
WHERE     NombredeClase = 'BD445'
```

El resultado de esta operación es:

100
200

Éstos son los números de estudiantes que necesitamos. Combinando las últimas dos consultas, obtenemos lo siguiente:

```
SELECT    Nombre
FROM      ESTUDIANTE
WHERE     EID IN
              (SELECT    NúmerodeEstudiante
              FROM       INSCRIPCIÓN
              WHERE      NombredeClase = 'BD445')
```

El segundo SELECT, denominado **subconsulta,** está entre paréntesis.

Puede ser más fácil comprender estos enunciados si trabaja desde la parte inferior y lee hacia arriba. Los tres últimos enunciados producen los nombres de los dos estudiantes inscritos en BD445, y los primeros tres enunciados arrojan los nombres de los dos estudiantes seleccionados. El resultado de esta consulta es:

JONES
BAKER

Para que esta operación sea semánticamente correcta, EID y NúmerodeEstudiante deben provenir del mismo dominio.

Las subconsultas pueden constar de tres o más tablas. Por ejemplo, supongamos que queremos saber los nombres de los estudiantes inscritos en la clase de los lunes, miércoles y viernes a las 3:00 p.m. (denotado como LMV3 en nuestros datos). Primero, necesitamos los nombres de las clases que cumplen este horario:

```
SELECT      CLASE.Nombre
FROM        CLASE
WHERE       Horario = 'LMV3'
```

(Puesto que estamos tratando con tres tablas diferentes, calificamos los nombres de las columnas con los nombres de las tablas para evitar confusión y ambigüedad. Así, CLASE.Nombre se refiere a la columna Nombre en la relación CLASE.)

Ahora obtenemos los números de identificación de los estudiantes en estas clases usando la siguiente expresión:

```
SELECT      INSCRIPCIÓN.NúmerodeEstudiante
FROM        INSCRIPCIÓN
WHERE       INSCRIPCIÓN.ClaseNombre IN
                (SELECT     CLASE.Nombre
                FROM        CLASE
                WHERE       HORARIO = `LMV3´)
```

Esto da como resultado:

100
200
300

los cuales son los números de los estudiantes en la clase LMV3. Para obtener los nombres de aquellos estudiantes, especificamos:

```
SELECT      ESTUDIANTE.Nombre
FROM        ESTUDIANTE
WHERE       ESTUDIANTE.EID IN
                (SELECT INSCRIPCIÓN.NúmerodeEstudiante
                FROM INSCRIPCIÓN
                WHERE INSCRIPCIÓN.NombredeClase IN
                    (SELECT CLASE.Nombre IN
                    FROM        CLASE
                    WHERE       CLASE.Horario = 'LMV3'))
```

El resultado es:

JONES
BAKER
BAKER

Esta estrategia funciona bien siempre que los atributos en la respuesta provengan de una sola tabla. Sin embargo, si el resultado proviene de dos o más tablas tendremos un problema. Por ejemplo, suponga que queremos saber los nombres de los estudiantes y los de sus clases. Digamos que necesitamos EID, NombredelEstudiante, y Nom-

bredeClase. En este caso, los resultados provienen de dos tablas diferentes (ESTUDIANTE e INSCRIPCIÓN) y así la estrategia de la subconsulta no funcionará.

JOIN CON SQL

Para producir el EID, Nombre y NombredeClase para todos los estudiantes, debemos hacer el join de la tabla ESTUDIANTE con la tabla INSCRIPCIÓN. Los siguientes enunciados harán esto:

```
SELECT    ESTUDIANTE.EID, ESTUDIANTE.Nombre, INSCRIPCIÓN.NombredeClase
FROM      ESTUDIANTE, INSCRIPCIÓN
WHERE     ESTUDIANTE.EID = INSCRIPCIÓN.NúmerodeEstudiante
```

Recuerde que un join es la combinación de la operación de un producto seguido por una selección, y (generalmente) por una proyección. Así, el enunciado FROM expresa el producto de ESTUDIANTE e INSCRIPCIÓN, y el enunciado WHERE, la selección. El significado es: "Seleccionar del producto ESTUDIANTE e INSCRIPCIÓN aquellos renglones en las que el EID de ESTUDIANTE sea igual a NúmerodeEstudiante en INSCRIPCIÓN". Por último, después de seleccionar, se toma la proyección del número de estudiante, su nombre y el de la clase. El resultado es:

100	JONES	BD445
150	PARKS	BA200
200	BAKER	BD445
200	BAKER	CS250
300	BAKER	CS125
400	RYE	BA200
400	RYE	BF410
400	RYE	CS250
450	JONES	BA200

La cláusula WHERE puede contener calificadores además de los que se necesitan para la junta. Por ejemplo,

```
SELECT    ESTUDIANTE.EID INSCRIPCIÓN.NombredeClase
FROM      ESTUDIANTE, INSCRIPCIÓN
WHERE     ESTUDIANTE.EID = INSCRIPCIÓN.NúmerodeEstudiante
AND       ESTUDIANTE.Nombre = 'RYE'
AND       INSCRIPCIÓN.NúmerodePosición = 1
```

Los calificadores adicionales aquí son ESTUDIANTE.Nombre = 'RYE' e INSCRIPCIÓN.NúmerodePosición = 1. Esta operación listará el número de estudiante y el nombre de la clase de todos los estudiantes llamados RYE que se inscribieron primero en la clase. El resultado es:

400	BF410

Cuando se necesiten datos de más de dos tablas, podemos usar una estrategia similar. En el siguiente ejemplo se juntan tres tablas:

```
SELECT    ESTUDIANTE.EID, CLASE.Nombre, CLASE.Horario,
          INSCRIPCIÓN.NúmerodePosición
FROM      ESTUDIANTE, INSCRIPCIÓN, CLASE
WHERE     ESTUDIANTE.EID = INSCRIPCIÓN.NúmerodeEstudiante
AND       INSCRIPCIÓN.NombredeClase = CLASE.Nombre
AND       ESTUDIANTE.Nombre = 'BAKER'
```

El resultado de esta operación es:

200	BD445	LMV3	2
200	CS250	LMV12	1
300	CS150	LMV3	1

Comparación del join y de la subconsulta SQL

Se puede usar un join como forma alternativa de expresión para muchas subconsultas. Por ejemplo, usamos una subconsulta para encontrar a los estudiantes inscritos en la clase BD445. También podemos usar un join para expresar esta consulta:

```
SELECT    ESTUDIANTE.Nombre
FROM      ESTUDIANTE, INSCRIPCIÓN
WHERE     ESTUDIANTE.EID = INSCRIPCIÓN.NúmerodeEstudiante
AND       INSCRIPCIÓN.NombredeClase = 'BD445'
```

De manera similar, la consulta: "¿Cuáles son los nombres de los estudiantes en las clases LMV a las 3?" se puede expresar como:

```
SELECT    ESTUDIANTE.NOMBRE
FROM      ESTUDIANTE, INSCRIPCIÓN, CLASE
WHERE     ESTUDIANTE.EID = INSCRIPCIÓN.NúmerodeEstudiante
AND       INSCRIPCIÓN.NombredeClase = CLASE.Nombre
AND       CLASE.Horario = 'LMV3'
```

Aunque las expresiones del join pueden sustituir a muchas expresiones de subconsultas, no pueden reemplazarlas a todas. Por ejemplo, las subconsultas que implican EXISTS y NOT EXISTS (las cuales analizaremos en la siguiente sección) no se pueden representar por medio de un join.

De igual forma, las subconsultas no se pueden sustituir por todos los join. Cuando usamos una unión, las columnas desplegadas pueden provenir de cualquiera de las tablas juntadas, pero cuando usamos una subconsulta las columnas desplegadas pueden provenir sólo de la tabla indicada en la expresión FROM en el primer SELECT. Por ejemplo, supongamos que queremos saber los nombres de las clases que toman los no graduados. Podemos expresar esto como una subconsulta:

```
SELECT    DISTINCT NombredeClase
FROM      INSCRIPCIÓN
WHERE     NúmerodeEstudiante IN
              (SELECT EID
              FROM ESTUDIANTE
              WHERE Grado NOT = 'GR')
```

o como un join:

```
SELECT    DISTINCT INSCRIPCIÓN.NombredeClase
FROM      INSCRIPCIÓN, ESTUDIANTE
WHERE     INSCRIPCIÓN.NúmerodeEstudiante = ESTUDIANTE.EID
AND       ESTUDIANTE.Grado NOT = 'GR'
```

Pero si queremos conocer los nombres de las clases y los niveles de grado de los estudiantes que aún no terminan su posgrado, debemos usar un join. Una subconsulta no será suficiente porque los resultados deseados provienen de dos tablas diferentes. Esto es, los nombres de las clases en INSCRIPCIÓN y los nombres de los estudiantes se almacenan en ESTUDIANTE. Lo siguiente arroja la respuesta correcta:

```
SELECT    DISTINCT INSCRIPCIÓN.NombredeClase, ESTUDIANTE.Grado
FROM      INSCRIPCIÓN, ESTUDIANTE
WHERE     INSCRIPCIÓN.NúmerodeEstudiante = ESTUDIANTE.EID
AND       ESTUDIANTE.Grado NOT = 'GR'
```

El resultado es:

BA200	SO
CS150	SN
BA200	FR
BF410	FR
CS250	FR
BA200	SN

Outer join

La norma SQL de la ANSI no avala los outer join. Sin embargo, muchos productos DBMS los manejan. Aquí ilustraremos el uso de uno de ellos.

Suponga que queremos una lista de todos los estudiantes y los nombres de las clases que toman, y que además deseamos incluir a todos los estudiantes, incluso aquellos que no están tomando clases. La siguiente expresión SQL dará este resultado utilizando Microsoft Access:

```
SELECT    Nombre, NombredeClase
FROM      ESTUDIANTE LEFT JOIN INSCRIPCIÓN
          ON EID = NúmerodeEstudiante;
```

El resultado es:

JONES	BD445
PARKS	BA200
BAKER	BD445
BAKER	CS250
GLASS	Nulo
BAKER	CS150
RUSSELL	Nulo
RYE	BA200
RYE	BF410
RYE	CS250
JONES	BA200

Observe las diferencias en Access SQL y la norma de notación ANSI. Las condiciones de la junta se especifican usando la palabra llave ON. También, todas las expresiones SQL terminan con punto y coma.

➤ EXISTS Y NOT EXISTS

EXISTS y NOT EXISTS son operadores lógicos cuyo valor puede ser verdadero o falso, dependiendo de la presencia o ausencia de renglones que cumplan las condiciones calificativas. Por ejemplo, suponga que deseamos saber los números de estudiantes inscritos en más de una clase.

```
SELECT      DISTINCT NúmerodeEstudiante
FROM        INSCRIPCIÓN A
WHERE       EXISTS
                (SELECT     *
                FROM        INSCRIPCIÓN B
                WHERE       A.NúmerodeEstudiante = B.NúmerodeEstudiante
                    AND     A.NombredeClase NOT = B.NombredeClase)
```

En este ejemplo, tanto la consulta como la subconsulta se refieren a la tabla INSCRIPCIÓN. Para evitar la ambigüedad, a estos dos usos de INSCRIPCIÓN se les ha asignado un nombre diferente. En el primer enunciado FROM, a INSCRIPCIÓN se le ha asignado, temporal y arbitrariamente, el nombre A, y en el segundo enunciado FROM, el nombre B.

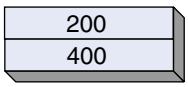
El significado de la expresión subconsulta es: Encontrar dos renglones en INSCRIPCIÓN que tengan el mismo número de estudiante, pero diferentes nombres de clases (lo cual significa que el estudiante está tomando más de una clase). Si existen dos de estos renglones, entonces el valor lógico de EXISTS es verdadero. En este caso, presentamos el número de estudiante en la respuesta. De otra manera, el valor lógico de EXISTS es falso, así que no presentamos ese EID en la respuesta.

Otra forma de ver esta consulta es imaginar dos copias separadas e idénticas de la tabla INSCRIPCIÓN. Nombre a una copia Tabla A y a la otra Tabla B. Compare cada renglón de la tabla A con cada renglón de la B. Primero busque en el primer renglón en A y en el primer renglón de B. En este caso, puesto que ambos son idénticos, tanto NúmerosdeEstudiantes como NombresdeClases son iguales, así que no desplegamos el EID.

Ahora observe el primer renglón en A y el segundo renglón en B. Si NúmerosdeEstudiantes son los mismos y NombresdeClases son diferentes, desplegamos NúmerodeEstudiante. Esencialmente, estamos comparando el primer renglón de INSCRIPCIÓN con el segundo renglón de INSCRIPCIÓN. Para los datos de la figura 9-2, ni NúmerosdeEstudiantes ni NombresdeClases son iguales.

Continuamos comparando el primer renglón de A con cada renglón de B. Si las condiciones se cumplen imprimimos el NúmerodeEstudiante. Cuando se hayan examinado todos los renglones en B, nos movemos al segundo renglón de A y lo comparamos con todos los renglones en B (realmente, si estamos considerando los *n* renglones en A, sólo se necesita considerar en B los renglones mayores que *n*).

El resultado de esta consulta es:



Para ilustrar la aplicación NOT EXISTS suponga que queremos conocer los nombres de los estudiantes que asisten a todas las clases. Otra forma de establecer lo anterior es que queremos los nombres de los estudiantes donde no haya clases que éstos no tomen. Lo siguiente expresa esto:

```
SELECT      ESTUDIANTE.Nombre
FROM        ESTUDIANTE
WHERE       NOT EXISTS
                (SELECT  *
                FROM     INSCRIPCIÓN
                WHERE    NOT EXISTS
                    (SELECT    *
                    FROM       CLASE
                    WHERE      CLASE.Nombre = INSCRIPCIÓN.NombredeClase
                    AND        INSCRIPCIÓN.NúmerodeEstudiante = ESTUDIANTE.EID))
```

Esta consulta tiene tres partes. En la inferior, se encuentran las clases que el estudiante ha tomado. La parte media determina si se encontraron algunas clases que el estudiante no haya tomado. De lo contrario, significa que el estudiante está tomando todas las clases y su nombre será desplegado.

Esta consulta puede ser difícil de comprender. Si tiene problemas use los datos de la figura 9-2 y siga las instrucciones. Para estos datos la respuesta es que ningún estudiante está tomando todas sus clases. Puede tratar de cambiar los datos, de tal forma que un estudiante tome todas las clases. Otra forma de analizar esta consulta es tratar de resolverla de otra manera usando NOT EXISTS. Los problemas que encuentre le ayudarán a comprender por qué NOT EXISTS es necesario.

➤ CAMBIO DE DATOS

SQL tiene medios para cambiar datos en tablas insertando renglones nuevos, eliminando renglones, y modificando los valores de los renglones existentes. SQL también puede cambiar la estructura de los datos, pero esto lo abordaremos hasta los capítulos 12 y 13.

Inserción de datos

En una tabla se pueden insertar renglones uno por uno o en grupos. Para insertar uno sólo establecemos:

```
INSERT      INTO INSCRIPCIÓN
            VALUES (400, 'BD445', 44)
```

Si no conocemos todos los datos —por ejemplo, si no conocemos el NúmerodePosición— podríamos decir:

```
INSERT      INTO INSCRIPCIÓN
            (NúmerodeEstudiante, NombredeClase)
            VALUES (400, 'BD445')
```

El NúmerodePosición se puede agregar después. Como podrá observar, esto ocasiona que el valor de NúmerodePosición tenga un valor nulo en el nuevo renglón.

También podemos copiar renglones en bloque de una tabla a otra. Por ejemplo, suponga que queremos llenar la tabla JUNIOR que se muestra en la figura 9-1.

```
INSERT      INTO JUNIOR
                VALUES
                (SELECT EID, Nombre, Especialidad
                FROM        ESTUDIANTE
                WHERE       Grado = 'JR')
```

Se pueden usar las expresiones contenidas en SELECT y todas las SELECT de SQL desarrolladas en las dos secciones anteriores para identificar los renglones que serán copiados. Esta característica ofrece capacidades bastante poderosas.

Eliminación de datos

Al igual que con la inserción, los renglones se pueden eliminar uno por uno o en grupos. El siguiente ejemplo suprime el renglón para Estudiante 100:

```
DELETE      FROM ESTUDIANTE
WHERE       ESTUDIANTE.EID = 100
```

Observe que si Estudiante 100 está inscrito en las clases, esta eliminación causará un problema de integridad: como los renglones de INSCRIPCIÓN tienen NúmerodeEstudiante = 100 no tendrán correspondencia con el renglón ESTUDIANTE.

Los grupos de renglones se pueden borrar como se muestra en los dos ejemplos siguientes, lo cual elimina todas las inscripciones para la especialidad de contabilidad, así como también a todos los estudiantes de esa especialidad.

```
DELETE      FROM INSCRIPCIÓN
WHERE       INSCRIPCIÓN.NúmerodeEstudiante IN
                (SELECT     ESTUDIANTE.EID
                FROM        ESTUDIANTE
                WHERE       ESTUDIANTE.Especialidad = 'Contabilidad')
DELETE      FROM ESTUDIANTE
WHERE       ESTUDIANTE.Especialidad = 'Contabilidad'
```

El orden de estas dos operaciones es importante, porque si se invirtiera ninguno de los renglones INSCRIPCIÓN se eliminarían debido a que los renglones coincidentes de ESTUDIANTE ya habrían sido eliminados.

Modificación de datos

Los renglones también se pueden modificar uno a la vez o en grupos. La palabra reservada SET se usa para cambiar el valor de una columna. Después de SET, se cambia el nombre de la columna y se especifica el nuevo valor o la forma de calcularlo. Considere dos ejemplos:

```
UPDATE      INSCRIPCIÓN
SET         NúmerodePosición = 44
WHERE       EID = 400
```

y

```
UPDATE      INSCRIPCIÓN
SET         NúmerodePosición = MAX (NúmerodePosición) + 1
WHERE       EID = 400
```

En el segundo enunciado UPDATE, el valor de la columna se calcula usando la función MAX. Sin embargo, algunas implementaciones de SQL no permiten usar la función interconstruida como argumento en el comando SET.

Para ilustrar las actualizaciones en masa, suponga que el nombre de un curso ha cambiado de BD445 a BD564. En este caso, para evitar problemas de integridad se deben cambiar las tablas INSCRIPCIÓN y CLASE.

```
UPDATE      INSCRIPCIÓN
SET         NombredeClase = 'BD564'
WHERE       NombredeClase = 'BD445'
UPDATE      CLASE
SET         NombredeClase = 'BD564'
WHERE       NombredeClase = 'BD445'
```

Recuerde que las actualizaciones en bloque pueden ser bastante peligrosas. Se le da un gran poder al usuario —poder que cuando se usa en forma correcta ayuda a realizar rápidamente la tarea, pero cuando se usa erróneamente puede ocasionar serios problemas.

➤ RESUMEN

SQL es el lenguaje de manejo de datos relacionales más importante que hay. Se ha convertido en el estándar para el intercambio de información entre computadoras, y su popularidad continúa en aumento. Los enunciados SQL que operan en una sola tabla incluyen SELECT, SELECT con WHERE, SELECT con GROUP BY y SELECT con GROUP BY y HAVING. SQL también contiene las funciones interconstruidas de COUNT, SUM, AVG, MAX y MIN.

Las operaciones en dos o más tablas se pueden hacer usando subconsultas, join, EXISTS y NOT EXISTS. Las subconsultas y los join realizan muchas de las mismas operaciones, pero no las sustituyen completamente. Las subconsultas requieren que los atributos recuperados provengan de una relación independiente, pero los join, no. Por otro lado, algunas consultas son posibles con subconsultas y EXISTS y NOT EXISTS, las cuales son imposibles con los join.

Los enunciados SQL para la modificación de datos incluyen las órdenes INSERT, DELETE y UPDATE, que se usan para agregar, remover y cambiar valores de datos.

En este capítulo presentamos los comandos básicos SQL de manera genérica, y en los capítulos 13, 14, y 16 los usaremos para procesar una base de datos empleando productos comerciales DBMS.

➤ PREGUNTAS DEL GRUPO I

Las preguntas en este grupo se refieren a las tres relaciones siguientes:

VENDEDOR (Nombre, PorcentajedeCuota, Salario)
PEDIDO (Número, NombredeCliente, NombredeVendedor, Cantidad)
CLIENTE (Nombre, Ciudad, TipodeIndustria)

Una instancia de estas relaciones se muestra en la figura 9-3. Use los datos en esas tablas y muestre los enunciados SQL para desplegar o modificar datos como se indica en las siguientes preguntas:

9.1 Muestre los salarios de todos los vendedores.

9.2 Muestre los salarios de todos los vendedores, pero omita duplicados.

9.3 Muestre los nombres de todos los vendedores que están por abajo del 30% de la cuota.

9.4 Muestre los nombres de todos los vendedores que tengan un pedido con Abernathy Construction.

9.5 Muestre los nombres de todos los vendedores que ganan más de $49999 y menos de $100000.

9.6 Muestre los nombres de todos los vendedores con un PorcentajedeCuota mayor a 49 y menor de 60. Use la palabra reservada BETWEEN.

9.7 Muestre los nombres de todos los vendedores con un PorcentajedeCuota de más de 49 y menos de 60. Use la palabra reservada LIKE.

9.8 Muestre los nombres de los clientes que se localicen en una Ciudad que termine con S.

9.9 Muestre los nombres y los salarios de todos los vendedores que no tengan un pedido con Abernathy Construction, en orden ascendente de salario.

9.10 Calcule el número de pedidos.

9.11 Calcule el número de diferentes clientes que tienen un pedido.

9.12 Calcule el porcentaje promedio de cuota para los vendedores.

9.13 Muestre el nombre del vendedor con el porcentaje de cuota más alto.

9.14 Calcule el número de pedidos de cada vendedor.

9.15 Calcule el número de pedidos de cada vendedor, considerando sólo los pedidos que excedan de 500.

9.16 Muestre los nombres y porcentajes de los vendedores que tienen un pedido con ABERNATHY CONSTRUCTION, en orden descendente de porcentaje de cuota (use una subconsulta).

9.17 Muestre los nombres y porcentajes de cuota de los vendedores que tengan un pedido con ABERNATHY CONSTRUCTION, en orden descendente de porcentaje de cuota (use un join).

9.18 Muestre los porcentajes de cuota de los vendedores que tengan un pedido con un cliente en MEMPHIS (use una subconsulta).

9.19 Muestre los porcentajes de cuota de los vendedores que tengan un pedido con un cliente en MEMPHIS (use un join).

9.20 Muestre el tipo de industria y los nombres de los vendedores de todos los pedidos para las compañías en MEMPHIS.

9.21 Muestre los nombres de los vendedores junto con los nombres de los clientes que les hayan hecho un pedido. Incluya a los vendedores que no tengan pedidos. Use la notación de Microsoft Access.

9.22 Muestre los nombres de los vendedores que tengan dos o más pedidos.

➤ FIGURA 9-3

Datos de muestra para las preguntas del grupo 1

Nombre	Porcentaje-deCuota	Salario
Abel	63	120,000
Baker	38	42,000
Jones	26	36,000
Murphy	42	50,000
Zenith	59	118,000
Kobad	27	36,000

VENDEDOR

Número	NombredelCliente	NombredelVendedor	Cantidad
100	Abernathy Construction	Zenith	560
200	Abernathy Construction	Jones	1800
300	Manchester Lumber	Abel	480
400	Amalgamated Housing	Abel	2500
500	Abernathy Construction	Murphy	6000
600	Tri-City Builders	Abel	700
700	Manchester Lumber	Jones	150

PEDIDO

Nombre	Ciudad	TipodeIndustria
Abernathy Construction	Willow	B
Manchester Lumber	Manchester	F
Tri-City Builders	Memphis	B
Amalgamated Housing	Memphis	B

CLIENTE

9.23 Muestre los nombres y porcentajes de cuota de los vendedores que tengan dos o más pedidos.

9.24 Muestre los nombres y edades de los vendedores que tengan un pedido con todos los clientes.

9.25 Muestre un enunciado SQL para insertar un nuevo renglón en CLIENTE.

9.26 Muestre un enunciado SQL para insertar un nombre nuevo y edad en VENDEDOR; suponga que el salario no está determinado.

9.27 Muestre un enunciado SQL para insertar renglones en una tabla nueva, ALTO-RENDIMIENTO (Nombre, salario), el cual incluya que un vendedor debe tener un salario de cuando menos $100000.

9.28 Muestre un enunciado SQL para borrar un cliente de ABERNATHY CONSTRUCTION.

9.29 Muestre un enunciado SQL para eliminar todos los pedidos de ABERNATHY CONSTRUCTION.

9.30 Muestre un enunciado SQL para cambiar el salario del vendedor JAIMES a $45,000.

9.31 Muestre un enunciado SQL para dar a todos los vendedores un aumento de 10 por ciento.

9.32 Suponga que el vendedor JAIMES cambia su apellido a PARKER. Muestre el enunciado SQL que crea los cambios apropiados.

➤ PREGUNTAS DEL GRUPO II

9.33 Instale Access 2002 y abra la base de datos Northwind. Use la herramienta Query-by-design/SQL View; escriba los enunciados SQL para las siguientes preguntas e imprímalas.

a. Liste todas las columnas de proveedores

b. Liste NombredelaCompañía de los proveedores con NombredeCompañía empezando con “Nuevo”

c. Liste todas las columnas de los productos que abastecen los proveedores con NombredeCompañía empezando con “Nuevo”. Muestre las respuestas usando un join y una subconsulta

d. Liste NiveldeReordenamiento y cuente todos los productos

e. Liste NiveldeReordenamiento y cuente todos los NivelesdeReordenamiento que tengan más de un elemento

f. Liste NiveldeReordenamiento y cuente todos los NivelesdeReordenamiento que tengan más de un elemento para los productos de los proveedores cuyos nombres empiecen con “Nuevo”

➤ PREGUNTAS DEL PROYECTO FIREDUP

Suponga que FiredUp ha creado una base de datos con las siguientes tablas:

CLIENTE (ClienteSK, Nombre, Teléfono, CorreoElectrónico)
ESTUFA (NúmerodeSerie, Tipo, Versión, FechadeFabricación)
REGISTRO (*ClienteSK*, *NúmerodeSerie*, Fecha)
REPARACIÓN_ESTUFA (NúmerodeFacturadeReparación, *NúmerodeSerie*, Fecha, Descripción, Costo, *ClienteSK*)

Codifique SQL para lo siguiente: suponga que todas las fechas están en el formato *mmddaaaa*.

A. Muestre todos los datos en cada una de las cuatro tablas FiredUp.

B. Liste las versiones de todas las estufas.

C. Liste las versiones de todas las estufas del tipo "FiredNow".

D. Liste NúmerodeSerie y Fecha de todos los registros en el año 2000.

E. Liste NúmerodeSerie y Fecha de todos los registros en Febrero. Use el comodín guión bajo (_).

F. Liste NúmerodeSerie y Fecha de todos los registros en Febrero. Use el comodín (%).

G. Liste los nombres y direcciones de CorreoElectrónico de los clientes que lo tengan.

H. Liste los nombres de todos los clientes que no tengan CorreoElectrónico; presente los resultados en orden descendente de Nombre.

I. Determine el costo máximo de una reparación de estufa.

J. Determine el promedio del costo de una reparación de estufa.

K. Cuente todas las estufas.

L. Cuente las estufas de cada tipo y despliegue el Tipo y el total.

M. Liste los nombres y direcciones de correo electrónico de todos los clientes que han tenido una reparación de una estufa superior a $50. Use una subconsulta.

N. Liste los nombres y las direcciones de correo electrónico de todos los clientes que tienen registrado un tipo de estufa "FiredNow". Use una subconsulta.

O. Liste los nombres y las direcciones de correo electrónico de todos los clientes que tengan una reparación que haya costado más de $50. Use un join.

P. Liste los nombres y las direcciones de correo electrónico de todos los clientes que tengan registrada un tipo de estufa "FiredNow". Use un join.

Q. Liste los nombres, direcciones de correo electrónico y fecha de registro de todos los registros de los clientes.

R. Muestre los nombres y direcciones de correo electrónico de todos los clientes que tengan registrada una estufa, pero que no hayan solicitado alguna reparación.

S. Muestre los nombres y las direcciones de correo electrónico de todos los clientes que tengan registrada una estufa, pero que no hayan tenido que repararla.

CAPÍTULO 10

Diseño de aplicaciones de bases de datos

Este capítulo introduce los conceptos fundamentales para el diseño de aplicaciones de bases de datos. Iniciamos con la enumeración y el análisis de las funciones de la aplicación de una base de datos. Cada función se describe detalladamente usando como ejemplo la aplicación de una galería de arte. Las ideas que se presentan en este capítulo son relativas a las aplicaciones de bases de datos que se desarrollan para usarlas en ambientes tradicionales tales como Windows. En los capítulos 14 al 16 se ampliarán estos conceptos para el diseño de las aplicaciones que usan la tecnología Internet.

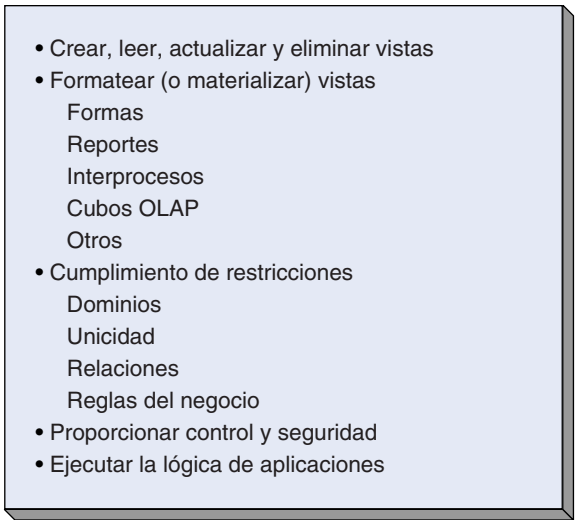
Antes de comenzar conviene una observación acerca de la terminología. En las primeras etapas del procesamiento de bases de datos era muy fácil encontrar un nexo entre el DBMS y la aplicación —las aplicaciones eran programas independientes que invocaban al DBMS—. Hoy, especialmente con los productos de escritorio del DBMS tales como Microsoft Access, esa conexión se ha vuelto confusa. Para evitar malos entendidos en este capítulo, supondremos que todas las formas, reportes, menúes y cualquier código de programa que esté contenido en cualesquiera de ellos, son parte de la aplicación de la base de datos. Además, los programas que llaman al DBMS son también parte de la aplicación. Cualquier estructura, regla o restricción que sea parte de la tabla, así como las definiciones de la relación, las maneja el DBMS y son parte de la base de datos. Así, una regla que se coloca en determinada columna de una tabla durante la definición de ésta se vuelve obligatoria mediante el DBMS. La misma regla, puesta en una forma de control, se considera obligatoria mediante la aplicación de la base de datos. A medida que avancemos podremos ver la importancia de esta diferencia.

➤ FUNCIONES DE UNA APLICACIÓN DE BASE DE DATOS

La figura 10-1 lista las funciones de una aplicación de base de datos. La primera es procesar vistas de datos. Hay cuatro funciones de procesamiento básico: crear, leer, actualizar y eliminar. En inglés a veces a estas funciones se les denomina mediante el acrónimo **CRUD.** Así, la primera tarea de una aplicación de base de datos es para vistas CRUD.

➤ FIGURA 10-1

Funciones de una aplicación de base de datos



Una vista de aplicación es más que un renglón de una tabla y más que el resultado de un enunciado SQL. Con frecuencia la construcción de una vista de aplicación requiere dos o más enunciados SQL, como se verá en la siguiente sección.

Una segunda tarea de la aplicación es **formatear** o **materializar** las vistas que están siendo procesadas. Observe que existe una diferencia entre los datos contenidos (la vista) y la apariencia de ese contenido (el formato o materialización). Una vista determinada por lo general tiene diferentes formatos. La distinción entre contenido y forma es especialmente importante en las aplicaciones de bases de datos que usan la tecnología Internet.

Hay varios tipos de materialización de vistas. Las formas y reportes son los dos más comunes; sin embargo, otros también son importantes. Una materialización del interproceso se usa para enviar una vista de un servidor a otro, o de una aplicación a otra. El formato de tales materializaciones se determina mediante una interfaz estándar como Microsoft COM, o un protocolo como XML, que presentaremos en el capítulo 14. El intercambio electrónico de datos es un buen ejemplo de su uso. Otros tipos de materialización son más especializadas. En el capítulo 17 verá el papel que desempeñan los cubos OLAP. El lenguaje natural es otro tipo de ejemplo.

El cumplimiento de restricciones es una tercera función. Éstas pueden ser estructurales, por ejemplo que los datos de valores requeridos se ajusten a las especificaciones de dominio, con lo que se asegura la unicidad y se imponen restricciones a la relación. Otras restricciones implican reglas del negocio, tales como: "Ningún vendedor puede tratar con ningún cliente cuya dirección asentada en el contrato esté fuera de la región que tiene asignada."

La cuarta función de una aplicación de bases de datos es proporcionar mecanismos de seguridad y control. La aplicación funcionará en conjunto con el sistema operativo y el DBMS para aumentar la seguridad que proporcionan los nombres de los usuarios y las contraseñas (passwords). Los menúes y estructuras similares determinan qué y cuándo pueden llevar a cabo determinadas acciones los usuarios.

La última tarea de una aplicación de base de datos es ejecutar la lógica del negocio. Por ejemplo, en una aplicación de captura de pedidos, cuando un cliente ordena cinco ejemplares de un libro, la aplicación necesita reducir automáticamente la cantidad de libros en el almacén. Si no hay ejemplares suficientes en inventario, o si la cantidad disponible es menor al punto de reorden, también se necesita llevar a cabo otra acción.

Las últimas dos funciones se abordan detalladamente en las clases de desarrollo de sistemas, por lo tanto casi no las abordaremos. Sin embargo, las primeras tres funciones son específicas de las aplicaciones de base de datos, y por ende centraremos nuestra atención en ellas.

Antes de aprender mayores detalles acerca de dichas funciones considere las necesidades para una base de datos y la aplicación con respecto a la galería View Ridge.

➤ CASO DE APLICACIÓN: GALERÍA VIEW RIDGE

La galería View Ridge es una pequeña empresa que vende arte contemporáneo, incluyendo litografías, pinturas originales y fotografías. Todas las litografías y fotografías están firmadas y numeradas y la mayoría de las piezas cuestan entre $1000 y $25000. View Ridge ha estado en el negocio durante 27 años; tiene un solo propietario que le dedica tiempo completo, así como tres vendedores y dos trabajadores que elaboran marcos, cuelgan los cuadros en la galería y preparan las obras de arte para su envío.

View Ridge realiza exposiciones y otros eventos con el fin de atraer clientes. Las obras de arte también se exhiben en compañías locales, restaurantes y otros lugares públicos. View Ridge es dueña de todo el material que vende; no tiene obras de arte a consignación.

Requerimientos para la aplicación

Los requerimientos para la aplicación View Ridge se resumen en la figura 10-2. En primer lugar, tanto el dueño como los vendedores desean dar seguimiento a sus clientes, así como a los intereses de éstos en cuanto a la compra de arte. Los vendedores necesitan saber a quién contactar cuando llega nuevo material, y tener toda esa información para crear cartas personalizadas y comunicarse verbalmente con sus clientes.

Además, la base de datos debe registrar las compras de arte de los clientes para que los vendedores dediquen más tiempo a los compradores más activos. En ocasiones también necesitan usar los registros de adquisiciones para ubicar las obras de arte, porque a veces la galería compra nuevamente algunas obras que son difíciles de encontrar y las revende. La aplicación de la base de datos debe tener un formato para agregar las nuevas obras que compra la galería.

View Ridge quiere que su aplicación de base de datos proporcione una lista de los artistas y los trabajos que se han exhibido en la galería. Al dueño también le gustaría poder determinar con qué rapidez se vende el trabajo de un artista y cuáles son los márgenes de utilidad, y la aplicación de base de datos debe mostrar un inventario actual en una página Web a la que los clientes puedan tener acceso a través de Internet. Por último, la galería View Ridge quiere que la base de datos produzca reportes que aminoren la carga de trabajo del contador de tiempo parcial que está contratado.

Diseño de la base de datos

La figura 10-3(a) muestra la estructura de los objetos que se requieren para manejar la base de datos de la galería. CLIENTE (CUSTOMER) y ARTISTA (ARTIST) son objetos compuestos y TRABAJO (WORK) es un objeto híbrido cuyo único identificador es el grupo {ARTISTA, Título y Copia} {ARTIST, Title y Copy}. El grupo multivaluado Transacción (Transaction) representa las adquisiciones y las ventas. Considerando que determinada obra puede pasar por la galería varias veces (debido a las adquisiciones y a

➤ FIGURA 10-2

Resumen de los requerimientos para las aplicaciones de la base de datos de la galería View Ridge

Dar seguimiento a los clientes y a sus intereses de compra.

Registrar las compras de material artístico de los clientes.

Registrar las compras.

Listar los artistas y las obras que se han exhibido en la galería.

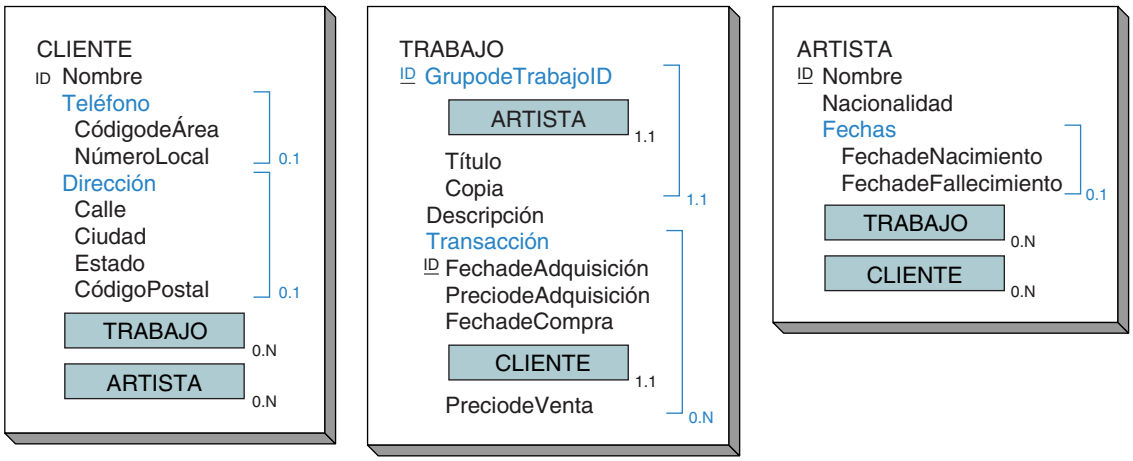
Reportar con qué rapidez se vende el trabajo de un artista y con qué margen de utilidad.

Mostrar el inventario actual en una página Web.

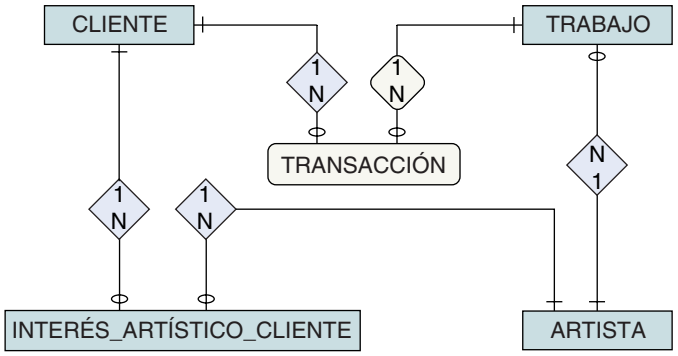
Listar reportes que empleará el contador de la galería.

➤ FIGURA 10-3

Diseño de la base de datos de la galería View Ridge: (a) objetos semánticos, (b) diagrama E-R, (c) diseño relacional, (d) diseño relacional con llaves sustitutas, (e) diagrama de relación de Access



(a)



(b)

CLIENTE (NúmerodeCliente, Nombre, CódigodeÁrea, NúmeroLocal, Calle, Ciudad, Estado, CódigoPostal)

TRABAJO (NombredelArtista, Título, Copia, Descripción)

TRANSACCIÓN (*NombredelArtista*, *Título*, *Copia*, FechadeAdquisición, PreciodeAdquisición, FechadeCompra, *NúmerodeCliente*, PreciodeVenta)

ARTISTA (NombredelArtista, Nacionalidad, FechadeNacimiento, FechadeFallecimiento)

INTERÉS_ARTÍSTICO_CLIENTE (NúmerodeCliente, NombredelArtista)

(c)

los intercambios), Transacción tiene valores múltiples. TRABAJO es un objeto híbrido porque contiene el atributo objeto CLIENTE en el grupo Transacción multivaluado (o multivalor).

View Ridge quiere registrar los intereses de sus clientes. En particular, desea conocer los artistas en los cuales un cliente en particular está interesado, y aquellos a los que les interesa un artista en especial. Estos requisitos se logran colocando los atributos multivaluados ARTISTA en CLIENTE, y CLIENTE en ARTISTA. En la figura 10-3(b) se muestra un diagrama E-R.

➤ FIGURA 10-3

(Continuación)

CLIENTE (ClienteID, Nombre, CódigodeÁrea, NúmeroLocal, Calle, Ciudad, Estado, CódigoPostal)
CUSTOMER (CustomerID, Name, AreaCode, LocalNumber, Street, City, State, Zip)

TRABAJO (TrabajoID, *ArtistaID,* Título, Copia, Descripción)
WORK (WorkID, *ArtistID*, Title, Copy, Description)

TRANSACCIÓN (TransacciónID, *TrabajoID,* FechadeAdquisición, PreciodeAdquisición, FechadeCompra, *ClienteID,* PreciodeVenta)
TRANSACTION (TransactionID, *WorkID*, DateAcquired, AcquisitionPrice, PurchaseDate, *CustomerID*, SalesPrice)

ARTISTA (ArtistaID, NombredelArtista, Nacionalidad, FechadeNacimiento, FechadeFallecimiento)
ARTIST (ArtistID, ArtistName, Nationality, Birthdate, DeceasedDate)

INTERÉS_ARTÍSTICO_CLIENTE (*ClienteID, ArtistaID*)
CUSTOMER_ARTIST_INT (*CustomerID*, *ArtistID*)

Restricciones de integridad referencial:

ArtistaID en TRABAJO debe existir en ArtistaID en ARTISTA

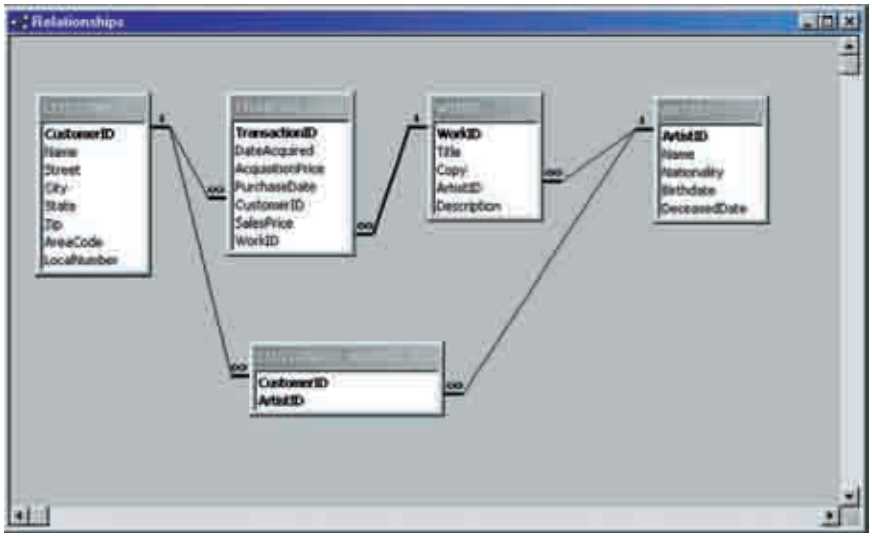
TrabajoID en TRANSACCIÓN debe existir en TrabajoID en TRABAJO

ClienteID en TRANSACCIÓN debe existir en ClienteID en CLIENTE

ClienteID en INTERÉS_ARTÍSTICO_CLIENTE debe existir en ClienteID en CLIENTE

ArtistaID en INTERÉS_ARTÍSTICO_CLIENTE debe existir en ArtistaID en ARTISTA

(d)



(e)

La representación relacional de los objetos de View Ridge se muestra en la figura 10-3(c). Puesto que el objeto CLIENTE no tiene un identificador único, se debe crear uno para usarlo como llave. Aquí hemos agregado un número de identificación: NúmerodeCliente. La cardinalidad máxima de los grupos Teléfono y Dirección es 1, así que los atributos en esos grupos se pueden colocar en la tabla CLIENTE; de hecho, no aparecen en la tabla como grupos, sino que la información del grupo se usa después para construir formas.

La llave de TRABAJO consiste en la llave de ARTISTA más Título y Copia. El único atributo sin la llave de TRABAJO es Descripción. Puesto que el atributo Transacción tie-

ne valores múltiples, debe creársele una tabla. Su llave es la del objeto en el que está contenida (NombredelArtista, Título, Copia) más la llave del grupo, la cual es FechadeAdquisición. Observe también que NúmerodeCliente se toma como llave externa en TRANSACCIÓN.

El objeto ARTISTA se representa mediante una tabla independiente, y NombredelArtista se puede usar como la llave de la tabla porque los nombres de los artistas están modelados para ser únicos. La tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE se debe crear para desarrollar la relación M:N entre CLIENTE y ARTISTA.

Debido al número de llaves de texto, y en particular a la gran llave compuesta de TRANSACCIÓN, este diseño puede mejorar sustancialmente si se reemplazan llaves de datos con llaves sustitutas. Esto se ha hecho en el diseño que se muestra en la figura 10-3(d). Los nombres de las llaves sustitutas se construyen anexando las letras ID al nombre de la tabla.

El diseño de la figura 10-3(d) es mejor porque hay menos datos duplicados. No es necesario copiar las columnas NombredelArtista, Título y Copia en la tabla Transacción. Debido a que puede haber muchas transacciones para un trabajo determinado, estos ahorros son valiosos. La figura 10-3(e) muestra un diagrama de relación para el diseño de la llave sustituta.

➤ CREAR, LEER, ACTUALIZAR Y BORRAR INSTANCIAS DE VISTAS

Una **vista** es una lista estructurada de elementos de datos de entidades u objetos semánticos definidos en el modelo. Un ejemplo de una vista es aquél que contiene datos para una entidad u objeto semántico. La figura 10-4(a) muestra un ejemplo de vista para la base de datos de la galería en la figura 10-3. Esta vista muestra datos de un cliente, sus transacciones e intereses artísticos. En esta vista existen potencialmente muchas TRANSACCIONes para cada CLIENTE, y cada TRANSACCIÓN tiene un TRABAJO. También, pueden existir muchos nombres de ARTISTA por cada CLIENTE (los artistas en los cuales el cliente está interesado).

Observe que ARTISTA.Nombre está repetido en esta vista. La primera vez es el nombre de un artista cuyo trabajo ha comprado el CLIENTE. La segunda, es el nombre del artista que le interesa al CLIENTE. Por ejemplo, en la figura 10-4(b) un cliente ha comprado trabajos de Tobey y de Miró y está interesado no sólo en el trabajo de estos dos artistas, sino también en el de Dennis Frings. En este caso, Tobey y Miró aparecen dos veces en la vista, una como un valor de TRABAJO.ARTISTA.Nombre y otra como un valor de ARTISTA.Nombre.

De nuevo, una vista es una lista estructurada de atributos. Debido a que está estructurada, los atributos pueden encontrarse más de una vez. También observe que una vista es sólo una lista de valores de datos. Puede ser formateada o materializada de muchas formas diferentes —como forma, reporte, u otro tipo de materialización.

Ahora considere las acciones CRUD que se pueden ejecutar en una vista. Comenzamos con Leer.

Lectura de ejemplos de vistas

Para leer una vista la aplicación debe ejecutar uno o más enunciados SQL para obtener valores de datos y luego poner los valores resultantes en la estructura de la vista. La vista de Cliente en la figura 10-4 contiene datos en dos caminos de acceso: uno a través de la tabla TRANSACCIÓN, y el otro mediante la tabla INTERÉS_ARTÍSTICO_CLIENTE. La estructura de los enunciados SQL es tal que sólo se puede seguir un camino a través del esquema con un enunciado SQL independiente. Así, esta vista requerirá un enunciado SQL para cada camino. Para el primero el siguiente enunciado SQL obtendrá los datos necesarios para el nombre del cliente "Jackson, Mary":

```
SELECT    CLIENTE.ClienteID, CLIENTE.Nombre,
          CLIENTE.CódigodeÁrea, CLIENTE.NúmeroLocal,
```

➤ FIGURA 10-4

Vista de Cliente: (a) lista estructurada de atributos, y (b) datos de muestra

CLIENTE.Nombre
CLIENTE.CódigodeÁrea
CLIENTE.NúmeroLocal
    TRANSACCIÓN.FechadeCompra
    TRANSACCIÓN.PreciodeVenta . . .
        TRABAJO.ARTISTA.Nombre
        TRABAJO.Título
        TRABAJO.Copia
    ARTISTA.Nombre . . .

(a) Lista estructurada de atributos

Jackson, Elizabeth		
206		
284-6783		
	7/15/94	
	4,300	
		Mark Tobey
		Lithograph One
		10/75
	2/5/99	
	2,500	
		Juan Miró
		Poster 14
		5/250
	11/22/97	
	17,850	
		Juan Miró
		Awakening Child
		1/1
	Juan Miró	
	Mark Tobey	
	Dennis Frings	

(b) Datos de muestra

```
          ARTISTA.Nombre, TRABAJO.Título, TRABAJO.Copia,
          TRANSACCIÓN.FechadeCompra, TRANSACCIÓN.PreciodeVenta
FROM      CLIENTE, TRANSACCIÓN, TRABAJO, ARTISTA
WHERE     CLIENTE.ClienteID = TRANSACCIÓN.ClienteID
  AND     TRABAJO.TrabajoID = TRANSACCIÓN.TrabajoID
  AND     ARTISTA.ArtistaID = TRABAJO.ArtistaID
  AND     CLIENTE.Nombre = 'Jackson, Mary'
```

Repasemos este enunciado SQL mientras revisamos el diagrama de relación en la figura 10-3(e). Los tres joins son necesarios para obtener datos a través de las tres relaciones en la parte superior del diagrama.

En el contexto del desarrollo de la aplicación, al resultado de un enunciado SQL a veces se le llama **recordset.** Para Microsoft, este término significa una relación con un empaquetador de programación de objetos. Un recordset tiene tanto métodos como

propiedades. *Open* es un ejemplo de método del recordset, *Cursor Type* es un ejemplo de una propiedad del recordset. Aprenderá más acerca de esto en el capítulo 14.

Para obtener los intereses de un cliente en determinado artista, se requiere un enunciado SQL para seguir el camino a través de INTERÉS_ARTÍSTICO_CLIENTE. El enunciado SQL para este camino es:

```
SELECT    CLIENTE.ClienteID, ARTISTA.Nombre
FROM      CLIENTE, INTERÉS_ARTÍSTICO_CLIENTE, ARTISTA
WHERE     CLIENTE.ClienteID = INTERÉS_ARTÍSTICO_CLIENTE.ClienteID
  AND     INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ARTISTA.ArtistaID
  AND     CLIENTE.Nombre = 'Jackson, Mary'
```

Debido a que CLIENTE.Nombre no es único, posiblemente los dos recordset de estos enunciados recuperarán datos de más de un cliente. Por lo tanto, la aplicación necesita tener lógica para examinar los valores de ClienteID en el recordset y asociar los renglones correctos.

Después de ejecutar estos dos enunciados, la aplicación tiene todos los datos necesarios para construir uno o más ejemplos de la vista que se presenta en la figura 10-3. Cómo se hace esto depende del lenguaje que se utilice. En COBOL la información se coloca en estructuras definidas en el Data Division. En Visual Basic la información se puede colocar en una estructura de datos o una serie de arreglos. En C++ y Java la información se coloca en los objetos. En realidad esos puntos no nos preocupan; lo que importa es que usted logre tener una idea completa de cómo debe ejecutar la aplicación uno o más enunciados SQL para llenar la estructura de la vista de datos. Véase el capítulo 16 para tener un ejemplo de Java.

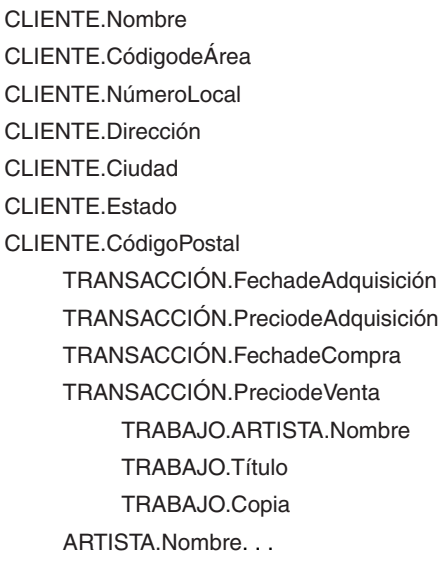
Creación de instancias de una vista

Para crear una instancia de una vista, la aplicación primero debe obtener los nuevos valores de datos y relaciones. Esto se hace mejor mediante un formato de entrada de información, pero también recibe datos de otros programas, y en otras maneras. Una vez que la aplicación tiene valores de datos, entonces ejecuta los enunciados SQL para almacenar la información en la base de datos.

Considere la vista Nuevo Cliente en la figura 10-5; se usa cuando un nuevo cliente compra una pintura. Contiene información acerca de él, de la transacción de compra y de sus múltiples intereses en obras de arte. Esta vista difiere de la de la figura 10-4 porque tiene más datos de clientes y también permite sólo una transacción. Sin embargo,

➤ FIGURA 10-5

Nueva vista de Cliente



puede haber valores múltiples ARTISTA.Nombre que registren los nuevos intereses del cliente.

Suponga que los valores de los datos para esta vista se localizan en una estructura de programa llamada NuevoCliente; además suponga que podemos accesar a los valores en la estructura anexando los caracteres NuevoCliente a los nombres en la estructura. Así, NuevoCliente.CLIENTE.Nombre tiene acceso a Nombre del CLIENTE en la estructura NuevoCliente.

Para crear esta vista en la base de datos debemos almacenar los nuevos datos del cliente en CLIENTE, los nuevos datos de transacción en TRANSACCIÓN y crear un renglón en la tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE para cada uno de los artistas que le interesan al cliente.

El siguiente enunciado SQL almacenará los nuevos datos del cliente:

```
INSERT    INTO CLIENTE
          (CLIENTE.Nombre,
          CLIENTE.CódigodeÁrea,
          CLIENTE.NúmeroLocal,
          CLIENTE.Dirección,
          CLIENTE.Ciudad,
          CLIENTE.Estado,
          CLIENTE.CódigoPostal)
VALUES    (NuevoCliente.CLIENTE.Nombre,
          NuevoCliente.CLIENTE.CódigodeÁrea,
          NuevoCliente.CLIENTE.NúmeroLocal,
          NuevoCliente.CLIENTE.Dirección,
          NuevoCliente.CLIENTE.Ciudad,
          NuevoCliente.CLIENTE.Estado,
          NuevoCliente.CLIENTE.CódigoPostal)
```

Suponga que cuando se crea el nuevo renglón el DBMS asigna el valor de la llave sustituta CLIENTE.ClienteID. Necesitaremos los valores de esta llave para terminar la creación de la nueva instancia, así que la aplicación necesitará obtenerla. Una manera de hacerlo es ejecutar el siguiente SELECT de SQL:

```
SELECT    CLIENTE.ClienteID, CLIENTE.CódigodeÁrea,
          CLIENTE.NúmeroLocal
FROM      CLIENTE
WHERE     CLIENTE.Nombre = NuevoCliente.CLIENTE.Nombre
```

Debido a que CLIENTE.Nombre no es único, puede aparecer más de un renglón en el recordset. En este caso, lo correcto sería identificarlo examinando la información de número de teléfono. Suponga que esto se ha realizado si es necesario y que el valor correcto está colocado en la estructura del programa como NuevoCliente.CLIENTE.ClienteID.

Un enunciado INSERT también será utilizado para almacenar el nuevo registro TRANSACCIÓN. Sin embargo, en este caso los valores de las llaves externas TRANSACCIÓN.TrabajoID y TRANSACCIÓN.ClienteID tendrán que ser proporcionados. Ya hemos mostrado cómo obtener el valor de ClienteID, así que lo único que falta es obtener el valor de TrabajoID. El siguiente SQL hará esto:

```
SELECT    TRABAJO.TrabajoID
FROM      TRABAJO,ARTISTA
WHERE     TRABAJO.ArtistaID = ARTISTA.ArtistaID
  AND     ARTISTA.Nombre = NuevoCliente.TRABAJO.ARTISTA.Nombre
  AND     TRABAJO.Título = NuevoCliente.TRABAJO.Título
  AND     TRABAJO.Copia = NuevoCliente.TRABAJO.Copia
```

Suponga que el valor de la llave sustituta que se ha regresado se almacena como NuevoCliente.TRABAJO.TrabajoID.

Se puede ejecutar el siguiente SQL para agregar el nuevo renglón TRANSACCIÓN:

```
INSERT      INTO TRANSACCIÓN
                (TRANSACCIÓN.TrabajoID,
                TRANSACCIÓN.FechadeAdquisición,
                TRANSACCIÓN.PreciodeAdquisición,
                TRANSACCIÓN.FechadeCompra,
                TRANSACCIÓN.ClienteID,
                TRANSACCIÓN.PreciodeVenta)
            VALUES
                (NuevoCliente.TRABAJO.TrabajoID,
                NuevoCliente.TRANSACCIÓN.FechadeAdquisición,
                NuevoCliente.TRANSACCIÓN.PreciodeAquisición,
                NuevoCliente.TRANSACCIÓN.FechadeCompra,
                NuevoCliente.CLIENTE.ClienteID,
                NuevoCliente.TRANSACCIÓN.PreciodeVenta)
```

Ahora todo lo que falta es crear renglones para la tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE. Para hacerlo, necesitamos obtener el ArtistaID de cada artista que le interesa al cliente, y luego crear un nuevo renglón en la tabla de intersección. El siguiente seudocódigo ilustra la lógica:

```
Para cada NuevoCliente.ARTISTA.Nombre:
        SELECT      ARTISTA.ArtistaID
        FROM        ARTISTA
        WHERE       ARTISTA.Nombre = NuevoCliente.ARTISTA.Nombre
        INSERT       INTO INTERÉS_ARTÍSTICO_CLIENTE
                    (ClienteID, ArtistaID)
                    VALUES (NuevoCliente.CLIENTE.ClienteID,
                    ARTISTA.ArtistaID)
El siguiente NuevoCliente.ARTISTA.Nombre
```

En este punto, la vista de Nuevo Cliente se ha almacenado en la base de datos. Por supuesto, una aplicación completa incluye la lógica para captar los errores que ha regresado el DBMS y procesarlos. Por ejemplo, la aplicación debe manejar los casos de TRABAJO que no existen en la base de datos, y los de un ARTISTA que tampoco tiene registrado.

Actualización de instancias de vistas

La tercera acción fundamental para realizar una vista es la actualización. Cuando se actualiza una vista hay tres tipos de cambio que son posibles: uno es un cambio simple de valores, como por ejemplo un cliente que cambia de número telefónico. Otra es un cambio a una relación; por ejemplo cuando un cliente ya no tiene interés en determinado artista. Un tercer tipo de actualización requeriría agregar uno o más nuevos renglones; eso ocurriría en nuestro ejemplo cuando un cliente hiciera una nueva compra.

El primer tipo de actualización se puede lograr con enunciados UPDATE SQL. Por ejemplo, suponga que un programa tiene una estructura llamada ActualizaciónCliente

(UpdateCust), y que ésta a su vez tiene ClienteID, CódigodeÁrea y NúmeroLocal. El siguiente SQL actualizará los nuevos valores:

```
UPDATE   CLIENTE
SET      CLIENTE.CódigodeÁrea = ActualizaciónCliente.CódigodeÁrea
         CLIENTE.NúmeroLocal = ActualizaciónCliente.NúmeroLocal
WHERE    CLIENTE.ClienteID = ActualizaciónCliente.ClienteID
```

Los cambios a relaciones también son directos. Si la relación es uno a muchos, entonces sólo se necesita actualizar el valor de la llave externa conforme al nuevo valor. Por ejemplo, suponga que la relación de DEPARTAMENTO con EMPLEADO es 1:N. Entonces Depto# (u otra llave) será almacenado como una llave externa en EMPLEADO. Para mover un EMPLEADO a un nuevo DEPTO, la aplicación sólo necesita cambiar el valor Depto# por uno nuevo.

Si la relación es muchos a muchos, entonces la llave externa en la tabla de intersección necesitará ser modificada. Por ejemplo, si en la galería un cliente ya no se interesa por Mark Tobey sino por Dennis Frings, entonces el renglón de intersección que representa la conexión con Mark Tobey necesita ser cambiado para que apunte a Dennis Frings.

Suponga que ActualizaciónCliente.ClienteID tiene el ID del cliente, ActualizaciónCliente.ViejoArtistaID tiene el ID del renglón de Mark Tobey en ARTISTA, y ActualizaciónCliente.NuevoArtistaID tiene el ID del renglón de Dennis Frings en ARTISTA. El siguiente SQL hará el cambio necesario:

```
UPDATE   INTERÉS_ARTÍSTICO_CLIENTE
SET      INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ActualizaciónCliente.NuevoArtistaID
WHERE    INTERÉS_ARTÍSTICO_CLIENTE.ClienteID = ActualizaciónCliente.ClienteID
  AND    INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ActualizaciónCliente.ViejoArtistaID
```

En una relación muchos a muchos también es posible eliminar una conexión sin reemplazarla, y crear una nueva sin desaparecer una vieja. Para borrar una conexión sólo eliminaríamos el renglón apropiado en la tabla de intersección. Para crear una agregaríamos un nuevo renglón en la tabla de intersección.

El tercer tipo de actualización requiere aumentar un nuevo renglón en una o más tablas. Si, por ejemplo, un cliente hace una nueva compra, entonces se necesitará crear otro renglón en la tabla de TRANSACCIÓN. Esto se puede realizar de la misma manera en que un nuevo renglón de TRANSACCIÓN se agrega cuando se crea una nueva instancia de vista.

Borrado de instancias de vista

Borrar una vista implica eliminar renglones de las tablas que la integran. La clave está en saber cuánto borrar. Por ejemplo, suponga que la galería quiere eliminar información de un cliente cuyo nombre es “Jones, Mary”. Obviamente, es necesario eliminar el renglón “Jones, Mary” en CLIENTE. Todo lo de los renglones de intersección en INTERÉS_ARTÍSTICO_CLIENTE que corresponden a esa clienta también deberá ser eliminado. ¿Pero qué hay acerca de los renglones en la tabla de TRANSACCIÓN? Esta tabla contiene ClienteID y si su renglón es eliminado todos los renglones en TRANSACCIÓN que tienen su valor de ClienteID tendrán datos inválidos.

Las respuestas a preguntas como esta surgen a partir del modelo de datos. En el caso del modelo E-R, todas las entidades débiles son eliminadas si la entidad de la que dependen también lo es. De otra manera, ningún renglón adicional de la tabla es suprimido. En el caso del modelo de objeto semántico todos los datos dentro de un objeto

son eliminados (puede haber muchas tablas en un objeto si éste tiene atributos multivaluados), pero no se suprime ningún dato en un objeto diferente. En adición a estas reglas, no se debe permitir ninguna eliminación si puede causar una violación a las cardinalidades de la relación. Tocaremos este tema ahora, y más adelante lo trataremos con más profundidad.

Considere el modelo en la figura 10-3. Cuando se borra un cliente, el renglón en CLIENTE es eliminado junto con todos sus renglones en INTERÉS_ARTÍSTICO_CLIENTE correspondientes. Los datos de TRANSACCIÓN no serán eliminados porque residen en un objeto diferente llamado TRABAJO. Sin embargo, observe que la cardinalidad mínima de CLIENTE en TRANSACCIÓN es 1. Por lo tanto, si un CLIENTE específico está ligado a una TRANSACCIÓN, entonces éste se requiere y su eliminación no debe ser permitida.

También podemos concluir de la figura 10-3 que si una vista de TRABAJO es eliminada, entonces el renglón TRABAJO y todos los renglones de TRANSACCIÓN relacionados se borrarán.

Por último, si se suprime una vista de ARTISTA, entonces los renglones ARTISTA e INTERÉS_ARTÍSTICO_CLIENTE se borrarán. Además, si hay algún objeto TRABAJO que esté ligado a ese ARTISTA, entonces la eliminación no podrá ser permitida.

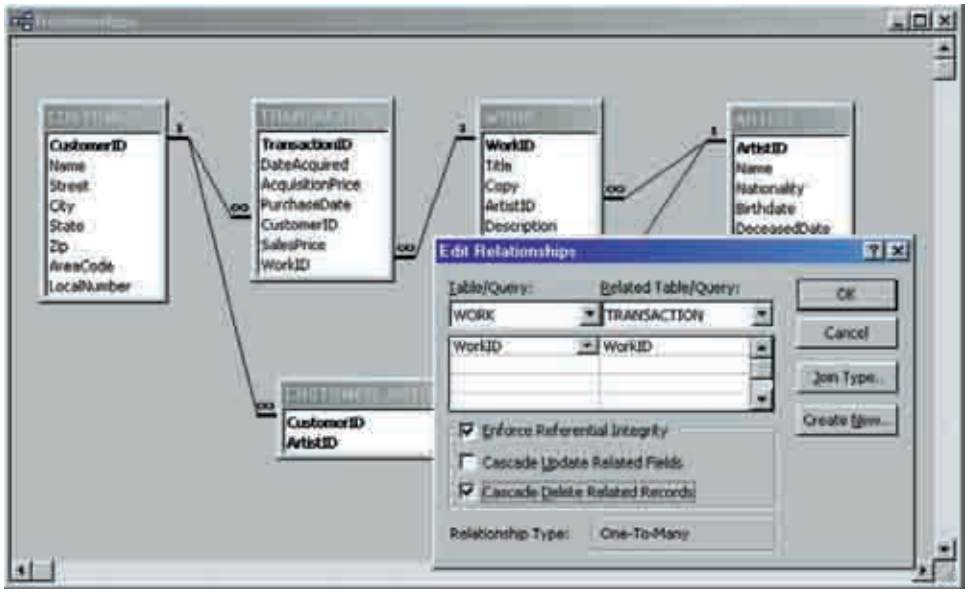
Por lo general algunos productos DBMS soportan la eliminación de los renglones dependientes bajo el término **eliminaciones en cascada.** La figura 10-6 muestra la caja de diálogo de la relación para Access. Observe que ha sido verificada la caja de registros de la relación de eliminación en cascada. Esto significa que cuando un renglón WORK (TRABAJO) es suprimido, Access automáticamente elimina cualquier renglón relacionado con TRANSACTION (TRANSACCIÓN).

Como se estableció, la cardinalidad de la relación desempeña un papel importante en la decisión de eliminar o no los renglones en la vista. Regresaremos a este tema más adelante cuando abordemos cómo forzar o hacer cumplir restricciones.

En esta sección estudiamos las acciones que se deben seguir cuando se crean, leen, actualizan y eliminan instancias de vistas. Algunas de las acciones aquí descritas se pueden realizar automáticamente mediante los productos DBMS. Por ejemplo, Access tiene asistentes (wizards) que generarán un formato con Cliente y un camino autónomo —ya sea para los intereses relacionados con Transacciones o con Artista—. Los usuarios de este formato pueden crear, leer, actualizar y eliminar instancias de un Cliente y datos en uno de los caminos. Sin embargo, los asistentes de Access no generarán un formato que apoye ambos caminos en la vista de Cliente en la figura 10-3. Para que lo anterior sea posible, un programador necesitará escribir el código de un programa.

➤ FIGURA 10-6

Especificación de eliminaciones en cascada



➤ DISEÑO DE FORMAS

Como se mostró en la figura 10-1, la segunda función principal de una aplicación de la base de datos es generar materializaciones de vistas. En este capítulo analizaremos materializaciones de formatos y reportes. En el capítulo 14 abordaremos materializaciones de interproceso usando XML, y en el capítulo 17 estudiaremos materializaciones de cubos OLAP.

Una **forma** es un desplegado en pantalla que se usa para introducir y editar datos. Las formas de sólo lectura también se pueden usar para reportar datos, pero en la mayoría de los casos, cuando los programadores hablan de formas, se refieren a las que se usan para el ingreso y la edición de datos.

Algunas formas son fáciles de usar y dan como resultado pocos errores en el ingreso de datos. Otras parecen difíciles de manejar, artificiales, y es difícil usarlas sin que se creen errores. En esta sección analizamos e ilustramos varios principios sobre el diseño correcto de formas.

La estructura de la forma debe reflejar la estructura de la vista

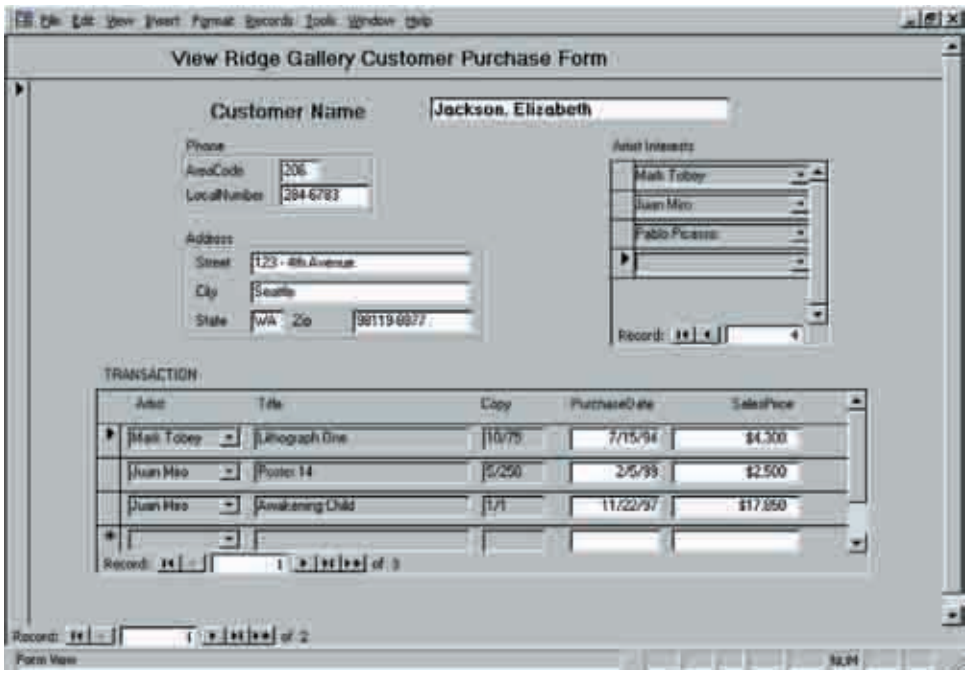
Primero, para que parezca natural y sea fácil de usar, *la estructura de una forma debe reflejar la estructura de la vista que materializa*. Observe la forma de la figura 10-7, la cual es una materialización de la vista Cliente en la figura 10-4(a).

La estructura de esta forma refleja la de la vista Cliente (Customer). Una sección de la forma tiene los datos básicos del cliente tales como Nombre, Teléfono y Dirección (Name, Phone and Address). La segunda sección muestra los intereses del cliente en Artista (Artist interests). Por último, la tercera sección lista las transacciones de compra del cliente. Los usuarios encontrarán esta forma fácil de usar debido a que los atributos están agrupados de manera tal que reflejan su comprensión sobre la estructura de los datos del cliente.

En general, cuando se diseña una forma toda la información de una relación independiente se debe colocar en una sección contigua de la forma. Suponiendo que la base de datos está en DK/NF, cada relación debe pertenecer a un tema específico, y el

➤ FIGURA 10-7

Materialización de una forma de vista de Cliente en la figura 10-4(a)



usuario esperaría encontrar en un lugar los datos de ese tema. Por lo tanto, en la forma hay clientes, interés del artista y secciones de transacción.

Hay una excepción a esta regla: los atributos en la relación base de la vista (aquí, la relación CLIENTE) a veces no están colocados en forma contigua. Suponga, por ejemplo, que CLIENTE tiene un atributo independiente llamado TotaldeCompras. Si siguiéramos esta regla pondríamos TotaldeCompras en la primera sección de la forma. Pero para los usuarios sería más claro poner ese atributo al final, después de que se hayan listado todas las compras.

La forma en la figura 10-7 no es la única aceptable de esta vista. Por ejemplo, TRANSACTION se podría colocar antes de Artist Interests. Los datos básicos de CUSTOMER podrían ser reacomodados para que tuvieran una apariencia más horizontal. CustomerName, AreaCode y LocalNumber se podrían colocar en una columna, y Street, City, State y Zip en otra. Todas estas alternativas permiten la estructura de la forma para que refleje la estructura de los objetos implícitos.

La semántica de la información debe ser gráficamente obvia

Otra característica de una forma bien diseñada es que la semántica de los datos es gráficamente obvia. Considere la sección de datos básicos del cliente de la forma CUSTOMER en la figura 10-7. Observe que existen rectángulos alrededor de AreaCode y LocalNumber y alrededor de Street, City, State y Zip.

Para entender por qué se hace esto, remitámonos nuevamente a la figura 10-3(a). El objeto semántico CUSTOMER (Cliente) tiene un atributo de grupo llamado Teléfono y otro Dirección. Puesto que ambos tienen una cardinalidad máxima de 1, en realidad no se requieren —desde el punto de vista del diseño relacional—. De hecho, no aparecen en las relaciones en la figura 10-3(c) o (d). El único propósito de estos atributos de grupo es asociar semánticamente CódigodeÁrea con NúmeroLocal, y Calle, Ciudad, Estado y CódigoPostal con algún otro.

El propósito de los rectángulos en la forma Customer Purchase Form es aclarar gráficamente estas asociaciones. La mayoría de los usuarios están de acuerdo con este arreglo. Debido a que saben que el número de teléfono consta de CódigodeÁrea y NúmeroLocal, les parece que la asociación gráfica de ambos es muy apropiada.

La estructura de la forma debe fomentar la acción apropiada

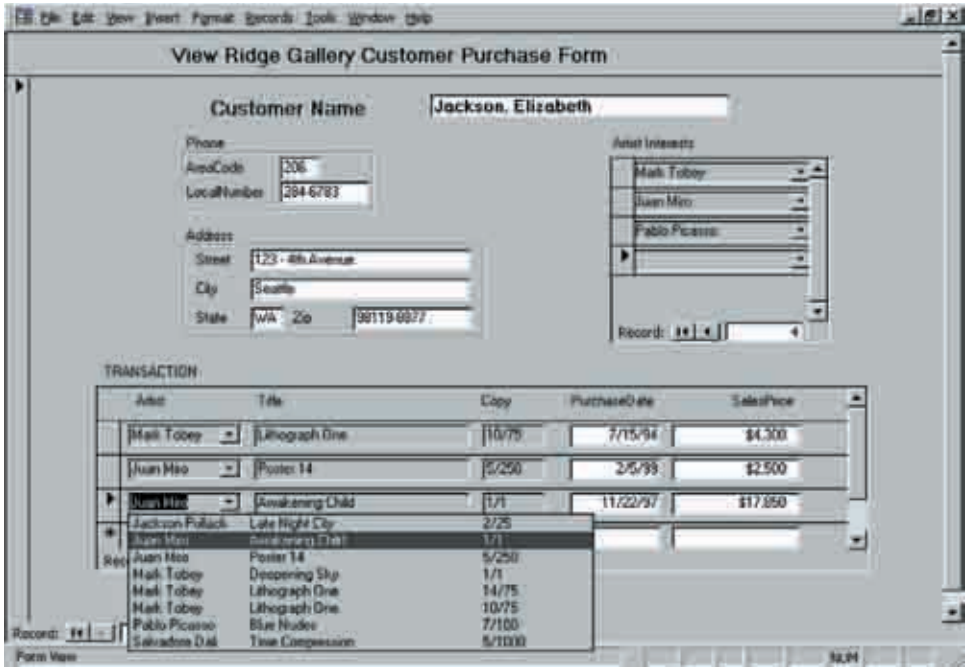
La estructura de las formas debe facilitar la ejecución de acciones apropiadas y dificultar la ejecución de acciones erróneas. Por ejemplo, el campo para ingresar State (Estado) en la forma de la figura 10-7 es pequeño. Obviamente, se supone que el usuario introduce sólo una abreviación de dos dígitos; un diseño más apropiado sólo permitiría ingresar dos dígitos; pero el mejor diseño presentaría los nombres de los estados en una lista desplegable y permitiría que el usuario seleccionara.

En la figura 10-7 algunos campos son blancos y otros grises. Las formas en esta aplicación han sido diseñadas para que el usuario sólo pueda ingresar datos en los espacios en blanco; los elementos de los datos en gris no pueden ser modificados. Así, el usuario no puede escribir en los campos Title o Copy en la sección TRANSACTION. En lugar de eso, para seleccionar o cambiar una obra, el usuario hace "clic" en la caja que está abajo de Artist. Cuando hace clic en la flecha, se despliega una lista de obras, como se muestra en la figura 10-8. Si el usuario quiere actualizar las obras o los datos de un artista, o agregar o eliminar datos de esa obra o dicho artista, debe utilizar una forma diferente.

La razón de este diseño es que cuando los usuarios están ingresando nuevas transacciones para los clientes, sólo deberían agregar los datos de la transacción; a saber, PurchaseDate (FechadeCompra) y SalesPrice (PreciodeVenta). Permitirles cambiar datos en ARTIST o WORK abriría la puerta a cambios accidentales o erróneos, que obstaculizaría la venta de una obra.

➤ FIGURA 10-8

Uso de la caja de lista desplegable



Un diseño similar se usa para Artist Interests. Los usuarios sólo pueden elegir de una lista. Sin embargo, el caso no es tan inflexible. Los vendedores podrían querer registrar que un cliente se interesa en un artista que no forma parte de la base de datos de la galería. Si es así, la forma cambiaría para permitir que los usuarios agreguen otros nombres a la lista. Como justificación a este diseño de la figura 10-7, el dueño de la galería quizá quiera registrar únicamente a los artistas de interés con los que trabaja o a los cuales representará. Probablemente quiera tener control de qué nombres pueden aparecer.

Formas en un ambiente GUI

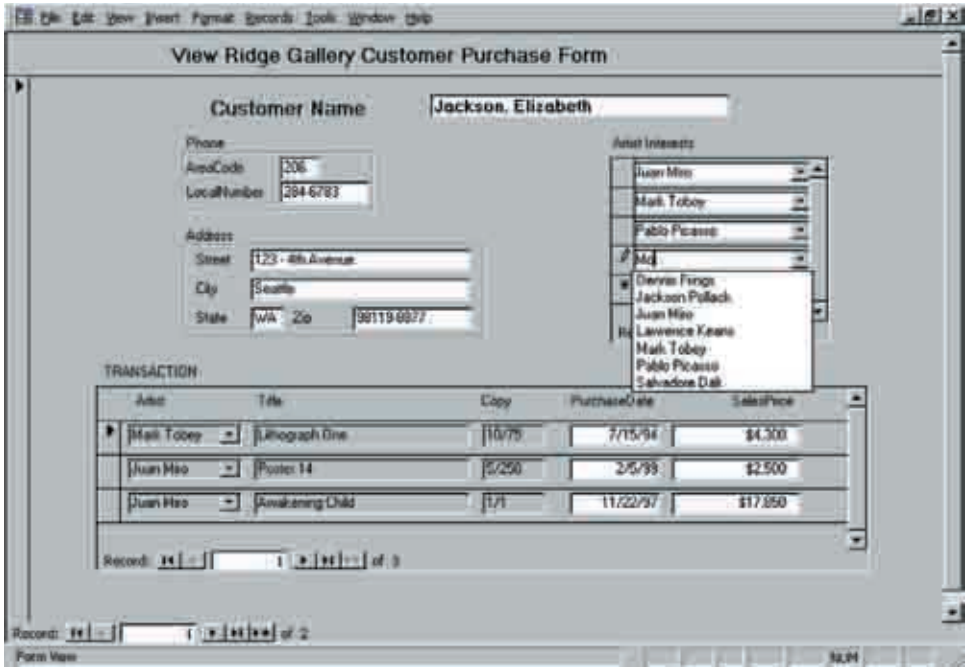
Hay varias características peculiares de los sistemas GUI que pueden facilitar enormemente el uso de las aplicaciones de la base de datos.

CAJAS DE LISTAS DESPLEGABLES. Una **caja de lista desplegable** es un control GUI que presenta una lista de elementos entre los cuales puede elegir el usuario. Una propiedad del control determina si la lista es fija o si los usuarios pueden agregarle algo. La figura 10-9 muestra una versión diferente de Customer Purchase Form (Forma de Compra del Cliente) en la cual los usuarios pueden agregar Nombres de Artistas que no formen parte de la lista. Tras bambalinas, la aplicación se almacena en un nuevo renglón de ARTIST en la base de datos. Por supuesto, para que esto funcione sólo ArtistName (NombredelArtista) puede ser un atributo requerido. Si hubiera otros campos requeridos, el DBMS rechazaría la inserción del nuevo renglón.

Las cajas de listas tienen muchas ventajas sobre las cajas de entrada de datos. Primero, para la gente es más fácil *reconocer* que *recopilar*. Por ejemplo, en la forma en la figura 10-8, es más fácil elegir de la lista el nombre de un artista, que recordar a todos los que forman parte de la base de datos. También es más fácil reconocer un nombre que escribirlo en forma correcta. Finalmente, si la caja de lista está preparada para desplegar sólo los valores presentes en la base de datos, el usuario no puede introducir errores de mecanografía. Por desgracia, el DBMS considerará como dos artistas diferentes a Juan Miró (con un espacio entre la *n* y la *M*) y a Juan Miró (con dos espacios entre la *n* y la *M*). Las cajas de listas fijas desplegables evitan estos errores.

➤ FIGURA 10-9

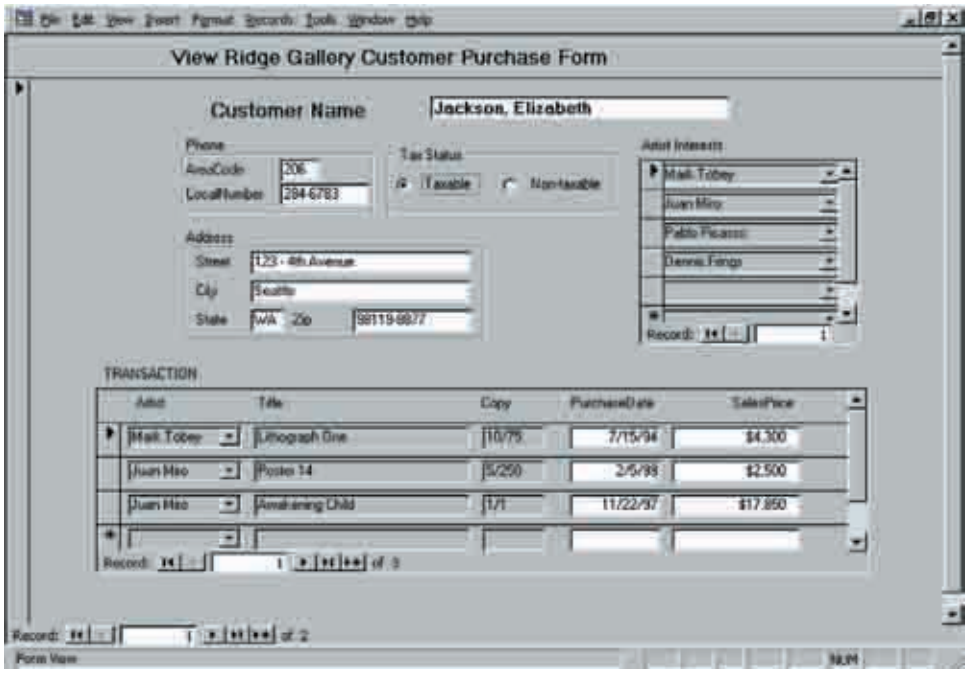
Caja de lista que permite nuevos valores



BOTONES DE OPCIONES Y GRUPOS. Un **botón de opción,** o un **botón de radio,** es un dispositivo de desplegado que permite a los usuarios seleccionar una condición alternativa o estado a partir de una lista de posibilidades. Por ejemplo, suponga que la galería decide que necesita registrar el estado fiscal de sus clientes. Suponga que éste tiene dos posibilidades:Taxable and Non-taxable (gravable y no gravable), y que ambas son mutuamente excluyentes. Se podría usar un grupo de opción como el que se muestra en la figura 10-10 para recopilar esos datos. Debido a la forma en la que el grupo de opción funciona, si el usuario selecciona Taxable, entonces Non-taxable automática-

➤ FIGURA 10-10

Uso de botones de grupo y de opción



mente deja de ser una selección. Si el usuario opta por Nogravable, entonces Gravable se elimina automáticamente.

Detrás de la forma, el programa de aplicación debe almacenar información en una columna de la tabla que representa el botón de radio seleccionado. Para este ejemplo, la columna se llama TaxStatus (EstadoFiscal). Se usa una de las dos maneras posibles para almacenar los datos del botón de opción. Una es almacenar un número entero del 1 al número de botones. En este ejemplo, uno de los valores 1 o 2 sería almacenado. La segunda opción es almacenar un valor de texto que describa la opción elegida. Las posibilidades son Sí o No.

CAJAS DE VERIFICACIÓN (CHECK BOXES). Las **cajas de verificación** son similares a los botones de opción, excepto que las alternativas en un grupo de cajas de verificación no son mutuamente excluyentes —se puede elegir más de una alternativa—. Suponga, por ejemplo, que la galería quiere registrar el tipo, o los tipos, de material artístico que les interesan a los clientes. Los posibles tipos son Lithographs, Oils, Pastels, and Photographs (Litografías, Óleos, Pasteles y Fotografías), y se muestran en una serie de cajas de verificación en la versión de la forma CUSTOMER en la figura 10-11. El usuario selecciona o marca las cajas apropiadas.

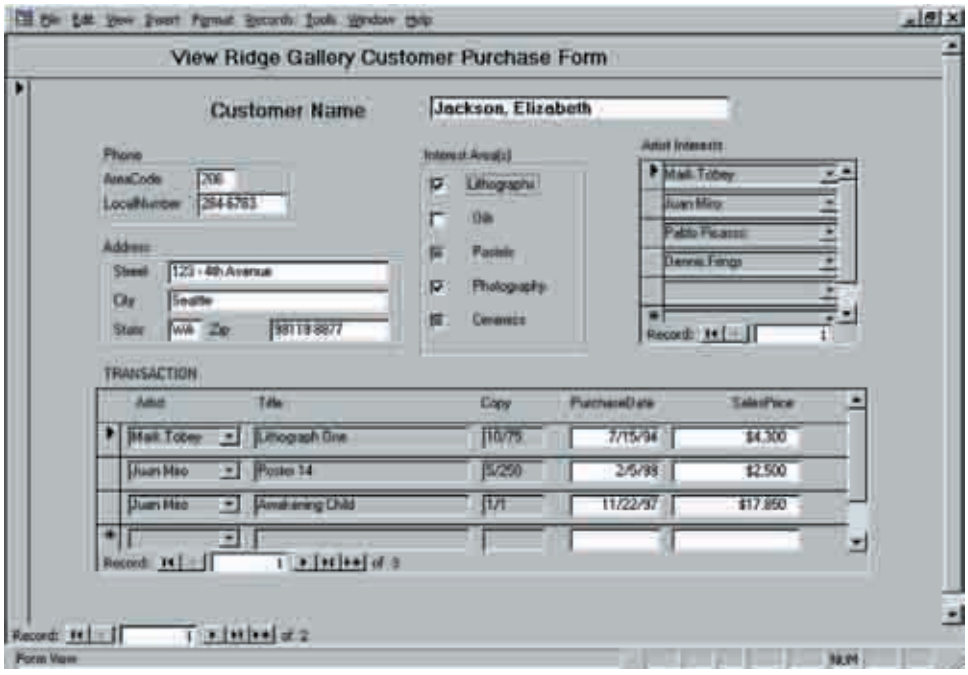
Hay varias formas de representar cajas de verificación en las relaciones. Una manera común y sencilla es definir una columna de valores booleanos para cada elemento en el grupo de cajas de verificación. El valor de cada una de estas columnas es binario; esto es, puede ser 1 o 0 para representar Sí o No. Hay otras posibilidades tales como codificar bits en un byte, aunque éstos no son importantes para nuestro análisis.

En la figura 10-11 observe los cuadros grises en Pastels y Ceramics. Ese gris significa que los valores de esas opciones son nulos. Los usuarios pueden interpretarlos como que este cliente está interesado en Lithographs y Photography y no en Oils. Su interés en Pastels y Ceramics es ambiguo (véase el análisis de nulos en el capítulo 6). Si es importante registrar el estado nulo, entonces el diseño de la base de datos debe permitir tres valores para cada caja de verificación: verificado, no verificado y nulo. Aunque el análisis también corresponde a los grupos con botones de opción.

FORMAS WEB. Las formas usadas en el explorador son GUI y todos los comentarios anteriores también se aplican. Sin embargo, las formas Web tienen una característica que las de Windows no tienen: hiperconexiones, las cuales son vínculos especial-

➤ FIGURA 10-11

Uso de cajas de verificación



mente útiles para materializar conexiones con otros objetos semánticos o entidades fuertes. Por ejemplo, en la forma de la figura 10-7, las conexiones con artistas de interés se pueden reemplazar por hiperconexiones para que cuando el usuario haga clic en éstas sea llevado a una forma ARTIST con los datos apropiados. Se puede hacer lo mismo con los enlaces a WORKs.

Movimiento del cursor y teclas de uso exclusivo

Otro aspecto sobre el diseño de formas es la acción del cursor, el cual se debe mover a través de la forma de manera fácil y natural. Esto generalmente significa que el cursor sigue el patrón de procesamiento del usuario conforme éste lee los documentos de ingreso de los datos fuente. Si se usan formas para introducir datos por teléfono, el cursor debe controlar el ritmo de la conversación. En este caso, este movimiento progresará de tal manera que el cliente lo encuentre natural y conveniente.

El movimiento del cursor es especialmente importante durante y después de una condición de excepción. Suponga que usando la forma en la figura 10-7 se comete un error —quizás al introducir un código inválido de estado—. La forma se debe procesar para que el cursor se mueva a una posición lógica. Por ejemplo, la aplicación puede desplegar una caja de lista de valores de estado disponibles y colocar al cursor en una posición lógica en la lista —quizás en el primer estado que empieza con la primera letra, la cual introdujo el usuario—. Cuando se selecciona el State (estado), el cursor debe regresar a Zip (CódigoPostal), el siguiente espacio adecuado de la forma.

Las acciones de las teclas de propósito especial, tales como ESC y las llaves de función, deberían ser *consistentes* y *exclusivas* en su uso. Si se usa ESC para salir de formas, se le debe usar únicamente para ese propósito (excepto cuando se trate de acciones que sean lógicamente equivalentes para salir de las formas). Las acciones de las teclas deben ser consistentes en toda la aplicación. Esto es, si se usa ESC para salir de una forma, también se debe usar para salir de todas las formas. Si se usa Ctrl-D para borrar datos en una forma, se debe usar para borrar datos en todas las formas. De otro modo, los hábitos que se adquieren en una parte de la aplicación se deben olvidar o aprender nuevamente en otras partes de la aplicación. Esto, además de ser un desperdicio de tiempo, es frustrante, exasperante y ocasiona errores. Aunque estos comentarios pueden parecer obvios, el cuidado con este tipo de detalles es lo que hace a una forma fácil y conveniente de usar.

➤ DISEÑO DE REPORTES

El tema del diseño de reportes, más que el diseño de formas, se ha analizado ampliamente en textos de desarrollo de aplicaciones. No duplicaremos, ni siquiera trataremos de resumir esos análisis aquí, sino más bien observaremos varios conceptos directamente relacionados con la noción de un reporte, tal como una materialización de la vista de una base de datos.

Estructura de reportes

Los principios para el diseño de un reporte eficaz son similares a los de diseño de formas. Igual que con las formas, *la estructura de un reporte debe reflejar la estructura de la vista implícita*. Esto significa que los datos de una tabla generalmente deben localizarse en un grupo contiguo al reporte. Al igual que las formas, una excepción a esta regla es que la relación básica de la vista (por ejemplo, la relación CLIENTE de la vista Cliente) puede estar por separado en el reporte. Los grupos de atributos, como teléfono, también se deben localizar juntos y distinguirse de alguna manera.

La figura 10-12 muestra un ejemplo de reporte de la galería View Ridge que lista los datos para cada obra de arte y las transacciones para cada una; calcula el margen total de utilidad por trabajo y por artista, así como el gran total.

➤ FIGURA 10-12

Reporte de lista de ventas

Lista de ventas
15-Nov-01

NombredelArtista	Título		Copia		
Dennis Frings	**South Toward Emerald Sea**		**106/195**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
4/17/1986	$750.00	5/3/1986	Heller, Max	$1,000.00	$250.00
3/15/2000	$1,200.00	5/11/2000	Jackson, Elizabeth	$1,800.00	$600.00
			MargenTotal de South Toward Emerald Sea, Copia 106/195		**$850.00**
			Margen total de utilidad en el caso de Dennis Frings		**$850.00**
Mark Tobey	**Patterns III**		**27/95**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
7/3/1971	$7,500.00	9/11/1971	Cooper, Tom	$10,000.00	$2,500.00
1/4/1986	$11,500.00	3/18/1986	Jackson, Elizabeth	$15,000.00	$3,500.00
9/11/1999	$17,000.00	10/17/1999	Cooper, Tom	$21,000.00	$4,000.00
			Margen Total para Patrones III, Copia 27/95		**$10,000.00**
Mark Tobey	**Rhythm**		**2/75**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
4/8/2001	$17,000.00	7/14/2001	Heller, Max	$27,000.00	$10,000.00
			Margen total de utilidad en el caso de Rhythm, Copia 2/75		**$10,000.00**
			Margen total de utilidad en el caso de Mark Tobey		**$20,000.00**
			Gran Total:		**$20,850.00**

La estructura del reporte en la figura 10-12 refleja la estructura del objeto TRABAJO. La sección para cada obra comienza con el nombre de ésta, el cual incluye al artista, el título y la copia. Sigue una sección de renglones de repetición que muestra las transacciones de la obra. Dentro de cada sección el nombre del cliente se ha extraído de la tabla CLIENTE.

Hay que estar conscientes de que con la mayoría de los editores de reportes es difícil construir un reporte que siga más de un camino de valores múltiples a través del esquema de la base de datos. El reporte en la figura 10-12 es una materialización de una vista de ARTISTA que sigue el camino desde ARTISTA hasta TRABAJO y luego a TRANSACCIÓN. El diagrama de relación (relationship) de la figura 10-3(e) muestra otra pista —que va de la tabla INTERÉS_ARTÍSTICO_CLIENTE (CUSTOMER_ARTIST_INT) a la de CLIENTEs (CUSTOMER) que están interesados en un artista específico—. Con la mayoría de los editores de reportes será difícil construir un reporte que muestre ambos caminos.

Los reportes con frecuencia tienen atributos de cálculo de datos que no son parte de la vista implícita ni están almacenados en la base de datos. El reporte en la figura 10-12 tiene cálculos para MargenTotaldeUtilidad, margen total de utilidad por Trabajo, margen total de utilidad por Artista y Gran Total. Todos estos valores se calculan al momento en que se produce el reporte.

Si bien estos valores calculados se podrían almacenar en la base de datos, hacerlo casi nunca es una buena idea porque los valores usados para el cálculo pueden cambiar.

Si, por ejemplo, un usuario modifica el PreciodeVenta de una transacción en particular y no calcula nuevamente el MargenTotaldeUtilidad y los totales basados en éste, los valores almacenados serán erróneos. Sin embargo, si se hacen todos los cálculos necesarios mientras se procesan las transacciones actualizadas probablemente dará como resultado un procesamiento inaceptablemente lento. Por lo tanto, los totales como este por lo general se calculan mejor sobre la marcha. Así, las fórmulas para calcularlos se consideran parte de la materialización del reporte y no parte de la vista implícita.

Objetos implícitos

Considere la solicitud: "Imprimir todos los ARTISTAs ordenados por margen total". A primera vista, parece ser una solicitud de impresión de un reporte acerca del objeto ARTISTA. Sin embargo, las palabras *ordenados por*, indican que se debe considerar más de un ARTISTA. De hecho, esta solicitud no está basada en el objeto ARTISTA, sino en el objeto CONJUNTO DE TODOS LOS ARTISTAs. El reporte en la figura 10-12 muestra los datos de múltiples ARTISTAs y de hecho está basado en el objeto CONJUNTO DE TODOS LOS ARTISTAs, y no en el objeto ARTISTA.

La mente humana cambia con tanta rapidez del objeto *OBJETO-A* al objeto *CONJUNTO DE TODOS LOS OBJETOS-A* que normalmente ni siquiera sabemos que ha ocurrido un cambio. Sin embargo, cuando se desarrollan las aplicaciones de la base de datos es importante observar este cambio, porque la aplicación necesita comportarse en forma diferente cuando esto sucede. Considere tres maneras en las que un ordenamiento puede cambiar la naturaleza del objeto base: (1) ordenar por identificador de objeto; (2) ordenar por columnas no identificadoras, y columnas no objeto; y (3) ordenar por atributos contenidos en los atributos de objeto.

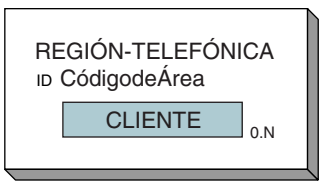
ORDENAMIENTO POR IDENTIFICADOR DE OBJETO. Si el reporte se va a ordenar por un atributo que es un identificador de objeto, el objeto verdadero es un conjunto de esos objetos. Así, un reporte de ARTISTA ordenado por NombredelArtista se refiere al objeto CONJUNTO DE TODOS LOS ARTISTAS. Para la mayoría de los productos de escritura de reportes DBMS, un reporte sobre el objeto CONJUNTO DE TODOS LOS X no es más difícil de producir que un reporte acerca del objeto X. Sin embargo, es importante saber que ha ocurrido un cambio en el tipo de objetos.

ORDENAMIENTO POR COLUMNAS NO IDENTIFICADORAS, Y COLUMNAS NO OBJETO. Cuando un usuario necesita un reporte ordenado por un atributo que es un identificador del objeto, es muy probable que en su mente el objeto verdadero sea un tipo de objeto totalmente diferente. Por ejemplo, el usuario desea producir un reporte acerca de ARTISTA ordenado por Nacionalidad. Dicho reporte en realidad es una materialización del objeto NACIONALIDAD, y no del objeto ARTISTA. De manera similar, si el usuario solicita un reporte sobre CLIENTE ordenado por CódigodeÁrea, el reporte en realidad está basado en un objeto llamado REGIÓN-TELEFÓNICA, o algún objeto similar. La figura 10-13 muestra un ejemplo del objeto REGIÓN-TELEFÓNICA.

Los objetos tales como NACIONALIDAD y REGIÓN-TELEFÓNICA son **objetos implícitos;** esto es, su existencia se puede inferir por el hecho de que el usuario solicitó tal reporte. Si tiene sentido para el usuario solicitar algo como un valor de ordenamiento, entonces ese algo debe ser un objeto en su mente, sin importar que esté modelado en la base de datos o no.

➤ FIGURA 10-13

Objeto REGIÓN TELEFÓNICA



ORDENAMIENTO POR ATRIBUTOS CONTENIDOS EN ATRIBUTOS OBJETO. La tercera manera en la cual se pueden ordenar los reportes es por atributos contenidos en los atributos del objeto. Por ejemplo, el usuario podría solicitar un reporte acerca de los TRABAJOs ordenados por FechadeNacimiento de ARTISTA, el cual es un atributo del objeto TRABAJO, y la Fecha de Nacimiento es un atributo contenido en ARTISTA. En este caso, el usuario en realidad está solicitando un reporte acerca de un objeto implícito (digamos, TIEMPO o PERIODO ARTÍSTICO) que contiene muchos ARTISTAS, cada uno de los cuales incluye muchos TRABAJOs en la galería.

Comprender este cambio en los objetos puede facilitar la tarea de desarrollar el reporte. Proceder como si TRABAJO fuera el objeto básico del reporte lo haría lógicamente artificial. Si TRABAJO se considera la base, se deben crear las materializaciones de los objetos TRABAJO que incluyen ARTISTA y FechadeNacimiento para todos los objetos TRABAJO, almacenados en el disco, y luego ordenarlos por FechadeNacimiento. Por otra parte, si este reporte se refiere a un objeto implícito que contiene ARTISTA, el cual a su vez abarca TRABAJO, entonces se pueden crear los objetos ARTISTA y ordenarlos por FechadeNacimiento, y tratar a los objetos TRABAJO como renglones de valores múltiples en cada objeto ARTISTA.

➤ CUMPLIMIENTO DE RESTRICCIONES

Como se observa en la figura 10-1, la tercera función principal de una aplicación de la base de datos es el cumplimiento de restricciones. En muchos casos, el DBMS es más capaz de imponer restricciones que la aplicación, y ésta no es estrictamente una función del programa de aplicación. Sin embargo, nuestra preocupación es describir los tipos de restricciones y cómo se pueden cumplir, sin importar el código de la aplicación o el DBMS que las haga cumplir.

En primer lugar, la razón de que el DBMS sea con frecuencia el mejor lugar para hacer cumplir restricciones es que constituye el punto central a través del cual deben pasar todos los cambios de información. Sin tomar en cuenta la fuente de un cambio de datos (forma, otro programa, importación de un bloque de datos) el DBMS tendrá la oportunidad de examinar y rechazar el cambio, si es necesario. Además, ciertas reglas (la unicidad es una de ellas) pueden requerir la revisión de todos los renglones de la tabla; el DBMS tiene más capacidad para ejecutar esta función que una aplicación. Además, si el DBMS impone una regla, entonces se necesita codificarla sólo una vez. Si las aplicaciones imponen una regla, entonces se necesitará codificar cada nueva aplicación. Esto es una pérdida de tiempo y genera la posibilidad de que los programas de aplicación impongan reglas de manera inconsistente.

Cabe aclarar que no todos los productos DBMS tienen las características y funciones necesarias para hacer cumplir restricciones. A veces es mucho más difícil escribir e instalar el código de aplicación de restricciones en productos DBMS que en un código de aplicación. Además, en algunas arquitecturas (el servidor de transacción de Microsoft, o Microsoft Transaction Server, es una de ellas) el procesamiento es más eficiente si se elimina la verificación de restricciones del servidor de datos. También, hay algunas restricciones que forman parte de la aplicación. Por ejemplo, un usuario de una forma en particular no se puede permitir ciertas acciones o introducir ciertos datos. En este caso, la aplicación tiene más capacidad para imponer la restricción.

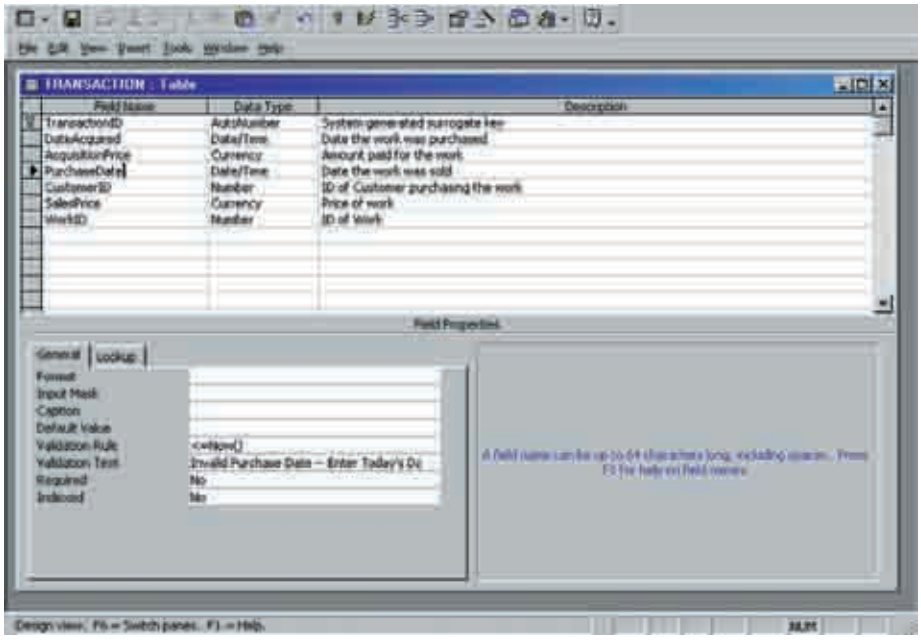
La meta que usted tiene es entender los tipos de restricciones y los medios por los cuales se aplican. Cuando sea posible, hay que imponerlas en el DBMS, cuando no, en la aplicación. Ahora consideraremos cuatro tipos de restricción: dominio, unicidad, integridad referencial y reglas del negocio.

Restricciones de dominio

Como establecimos en el el capítulo 4, un dominio es el conjunto de valores que un atributo puede tener. Las definiciones de dominio tienen un componente semántico

➤ FIGURA 10-14

Creación de una restricción de dominio con Access



(el nombre de un artista) y uno físico (texto 25, alfabético). En general, no es posible imponer el componente semántico mediante un proceso automatizado. La industria de la computación aún no está en una etapa de desarrollo tal en que los procesos automatizados puedan determinar que 3M es el nombre de una compañía y no el de un artista. Por lo tanto, actualmente estamos principalmente preocupados por la aplicación de la parte física de una definición de dominio.

Las restricciones de dominio surgen del modelo de datos, como se describió en el capítulo 4. La especificidad de las restricciones de dominio varían ampliamente. En la base de datos de la galería View Ridge, Artista.Nombre puede ser cualquier conjunto de 25 o menos caracteres de texto. Sin embargo, un dominio como el de Copia, es más específico. El formato usado por la galería es nn/mm, donde nn es un número particular de copias y mm es el número total de copias hechas. Así, 5/100 significa la quinta copia de 100. Claramente, un valor como 105/100 no tiene sentido. Por lo tanto, se debe desarrollar el código de validación para asegurar que se siga este formato.

La figura 10-14 muestra un ejemplo de restricción de dominio en Access. La restricción es que PurchaseDate (Fecha de compra) debe ser menor o igual a la fecha del reloj de la computadora en la hora y la fecha en que se introdujo la compra. Observe el renglón etiquetado Validation Rule (regla de validación). La expresión < = Now ( ) define la regla. Si el usuario intenta introducir datos que violen esta regla, se genera una caja de mensaje usando el texto que se introdujo en el siguiente renglón (texto de validación, o Validation Text). La figura 10-15 muestra qué ocurre cuando el usuario intenta ingresar la fecha 10/15/99 cuando la computadora tiene una fecha anterior. En ese caso Access rechaza los datos.

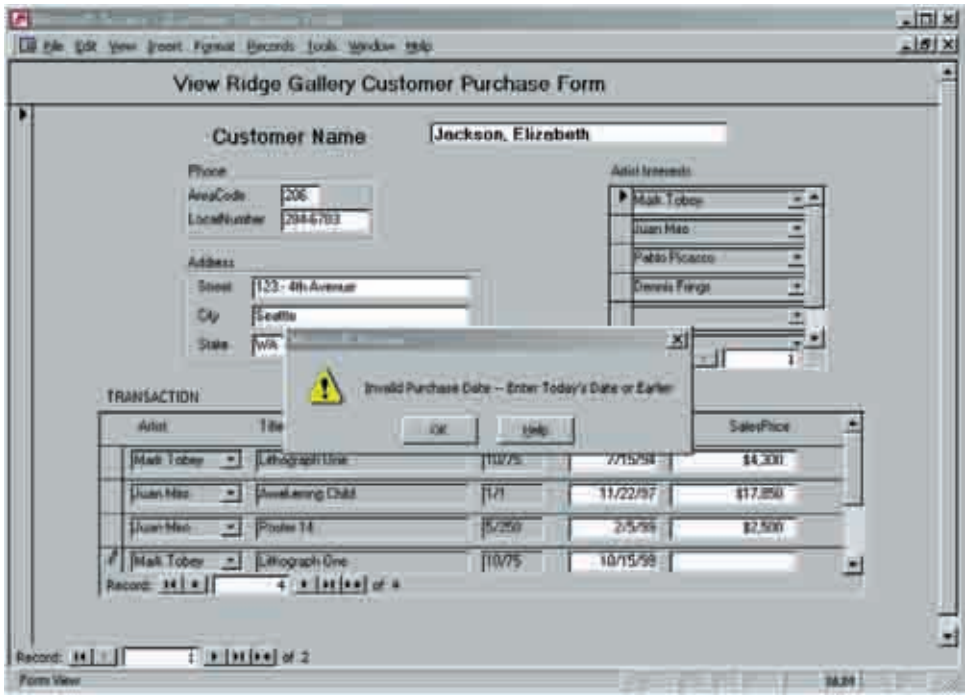
Otro tipo de restricción es si los valores son o no requeridos. Estrictamente hablando, el requerimiento de que se proporcione un valor es una restricción de la tabla, más que una restricción de dominio. Un dominio es un conjunto de valores; si un valor se requiere o no es una pregunta que surge cuando el dominio aparece en una tabla.

La modelación de datos debe indicar si se requieren los valores del atributo. En el modelo de objeto semántico si la cardinalidad mínima de un atributo es 1, entonces se requiere ese atributo. Para imponer esta restricción con Access todo lo que se necesita es establecer la propiedad Required de una columna en Yes. Se hizo lo mismo con Name en la definición de la tabla ARTIST en la figura 10-16. Como verá, se usan diferentes medios con otros productos DBMS.

Las restricciones de valor que se requieren son importantes porque eliminan la posibilidad de valores nulos. Otra forma de hacer esto es definir un valor inicial que el

➤ FIGURA 10-15

Resultado de violar una regla de validación



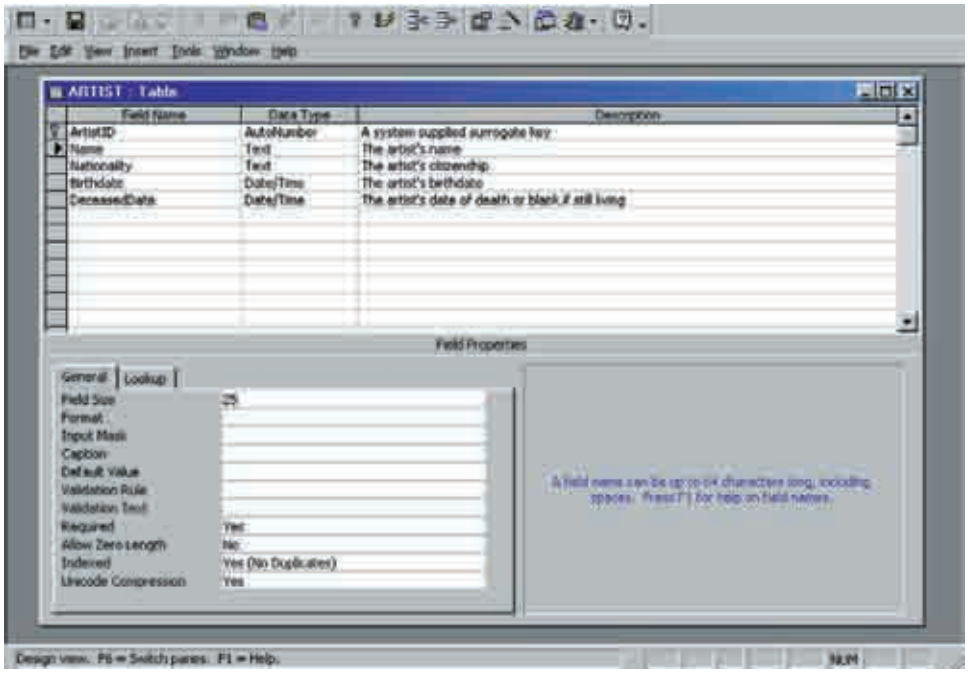
DBMS asigne a la columna cuando se cree el renglón. Con Access esto se hace colocando un valor o expresión en la propiedad Default Value (Valor Predeterminado). Véanse la figura 10-16 y el análisis de los valores nulos en el capítulo 6.

Restricciones de unicidad

La unicidad es el segundo tipo de restricción. Como se estableció, las restricciones de este tipo las aplica mejor el DBMS debido a que puede crear estructuras de datos para efectuar más rápido la verificación de la unicidad. Véase el apéndice A para obtener una descripción de la manera en que se pueden usar los índices con este propósito.

➤ FIGURA 10-16

Definición de la unicidad para Nombre en ARTISTA (Name in ARTIST)



La figura 10-16 muestra cómo Name en ARTIST puede ser definido como único en Access. Observe que Indizar se ha puesto en Yes (No Duplicates). Con esa precisión Access se asegurará de que no se dupliquen los nombres de los artistas que ingresen desde cualquier fuente.

Restricciones de relación

Hay dos tipos de restricciones de relación: **integridad referencial** y **cardinalidad de relación.** La integridad referencial consiste en las restricciones a los valores de las llaves externas. Por ejemplo, las restricciones que surgen de la normalización cuando una relación individual se divide en dos, y eso ocurre entre entidades fuertes y débiles. Las restricciones de cardinalidad de relación ocurren debido a los valores de cardinalidad máxima y mínima en las relaciones. En esta sección consideramos ambos tipos.

RESTRICCIONES DE INTEGRIDAD REFERENCIAL. Todas las restricciones de integridad referencial son restricciones a valores de llaves externas. Considere la primera restricción de integridad referencial en la figura 10-3(d): ArtistaID en TRABAJO debe existir en ArtistaID en ARTISTA. ArtistaID en TRABAJO es una llave externa y la restricción sólo significa que su valor debe concordar con un valor en ArtistaID en la tabla ARTISTA, como ya se explicó.

Para que no exista violación alguna a la composición de la llave externa es necesario que se cumplan las siguientes condiciones: que todas las aplicaciones estén codificadas en forma correcta, que todas las transacciones se procesen adecuadamente y que ningún usuario borre por accidente un artista sin haber eliminado lo que se relaciona con TRABAJO (el administrador debe borrar cualquier TRANSACCIÓN relacionada, conforme a lo que se estipula en la segunda condición y en eliminar todo CLIENTE relacionado, según lo que se establece en la tercera condición). Después de todo, la base de datos comienza sin problemas de integridad; cuando éstos ocurren es que fueron introducidos.

Lo anterior, por supuesto, es esperar demasiado. Los errores suceden; por lo tanto, los productos DBMS proporcionan la facilidad de que estas restricciones de la llave externa las pueda definir e imponer el DBMS. Esto significa que el DBMS no permitirá ninguna inserción, eliminación, o modificación de un valor de llave externa que ocasione una violación a la restricción de una llave externa. Por ejemplo, observe la figura 10-20, donde la marca frente a la opción Enforce Referential Integrity (Imponer Integridad Referencial) significa que se le ha pedido a Access que cumpla la restricción de que EmployeeName (NombredeEmpleado) en:

EMPLOYEE_PhoneNumber (EMPLEADO_NúmerodeTeléfono) exista en Name (Nombre) en EMPLOYEE (EMPLEADO)

Facilidades similares existen para todos los productos DBMS. Además, los productos del servidor DBMS tales como los ORACLE y SQL Server proporcionan utilidades que pueden estar corriendo periódicamente para asegurar que ninguna restricción de llave externa haya sido permitida en la base de datos debido a transacciones parcialmente terminadas, datos importados, u otros misterios del universo. Estas utilidades corren normalmente como parte del plan de mantenimiento de la base de datos (véase el capítulo 11).

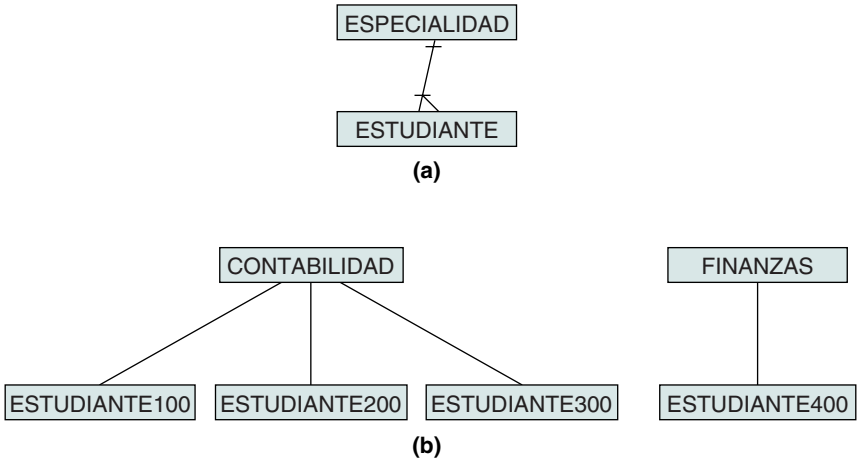
RESTRICCIONES DE CARDINALIDAD DE LA RELACIÓN. Las restricciones de cardinalidad de la relación surgen del establecimiento de la cardinalidad en los atributos de conexión del objeto. Por ejemplo, en la figura 10-3(a) la conexión CLIENTE en TRABAJO. La transacción tiene cardinalidad de 1.1; por lo tanto, una Transacción debe tener una conexión con CLIENTE.

En general, estas restricciones surgen de dos fuentes: establecimiento de no cero en la cardinalidad mínima, o asignaciones de cardinalidad máxima que no son 1 ni N. Así, las cardinalidades de 1.1, 1.N, 2.N ocasionarán que las restricciones de cardinalidad se presenten, como lo harán las cardinalidades de 0.3, 1.4, y 2.4. Con una excepción, estas restricciones se deben imponer mediante un código de aplicación.

La excepción son las cardinalidades 1.1 en el lado hijo de relaciones 1:N. En este caso, la restricción se puede imponer haciendo obligatoria la llave externa. Así, la res-

➤ FIGURA 10-17

Ejemplo de una restricción obligatoria-a-obligatoria: (a) muestra de relación obligatoria-a-obligatoria, y (b) muestra de datos para ésta



tricción 1.1 de CLIENTE en TRABAJO.Transacción se puede imponer haciendo ClienteID obligatoria en la tabla TRANSACCIÓN.

Además de esta salvedad, el programador de la base de datos debe escribir un código para imponer restricciones de cardinalidad. Dicho código se puede colocar en los procedimientos almacenados (stored procedures) para que la invoque el DBMS cuando se hagan cambios de relación, se puede poner en programas de aplicación, o llamarlo durante ciertos eventos de forma, tal como AntesdelaActualización (BeforeUpdate), que analizaremos más adelante.

Para simplificar el análisis sólo consideraremos restricciones 1.1 y 1.N. La lógica para las otras restricciones es una extensión directa de la que se presenta aquí.

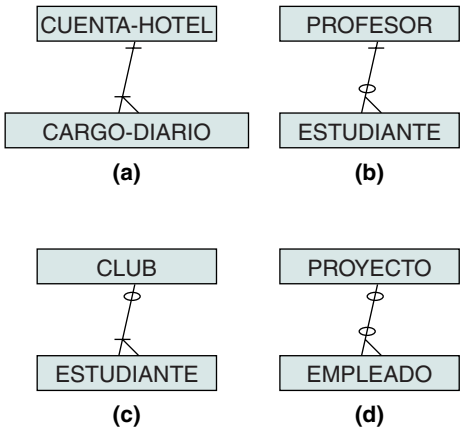
La figura 10-17 describe la relación entre las relaciones ESPECIALIDAD y ESTUDIANTE. Como se muestra en la figura 10-17(a), una ESPECIALIDAD debe tener cuando menos un ESTUDIANTE y un ESTUDIANTE debe tener exactamente una ESPECIALIDAD. Cuando los usuarios actualizan cualquiera de estas relaciones, se debe invocar el código para imponer la restricción. Por ejemplo, en la figura 10-17(b) si un usuario intenta eliminar el renglón de Estudiante 400, el código debe rechazar esa solicitud. Si ésta fuera permitida, el renglón para FINANZAS no tendría un renglón hijo, y la restricción obligatoria sería violada. De manera similar, una nueva ESPECIALIDAD, digamos BIOLOGÍA, no se puede agregar hasta que haya un estudiante que se esté especializando en esa materia.

Un renglón que existe en forma inadecuada sin un padre o hijo con frecuencia se llama **fragmento,** y a los renglones hijos que existen sin un padre obligatorio a veces se les llama **huérfanos.** Una de las funciones de un programa de aplicación es evitar la creación de fragmentos y huérfanos.

Los medios para evitar los fragmentos dependen del tipo de restricción. La figura 10-18 muestra ejemplos de las cuatro restricciones posibles en las relaciones 1:N: obli-

➤ FIGURA 10-18

Ejemplo de los cuatro tipos de restricciones: (a) restricción obligatoria a obligatoria (M-M); (b) restricción obligatoria a opcional (M-O); (c) restricción opcional a obligatoria (O-M), y (d) restricción opcional a opcional (O-O)



gatoria a obligatoria (M-M); obligatoria a opcional (M-O); opcional a obligatorio (O-M), y opcional a opcional (O-O). Estas restricciones se muestran en las relaciones uno a muchos, pero los mismos cuatro tipos se aplican también a otras.

Se pueden violar las restricciones cuando hay cambios en los atributos de la llave. Por ejemplo, en la figura 10-17(b), cambiar la especialidad de Estudiante 300 de CONTABILIDAD a FINANZAS reasigna a ese estudiante al departamento de finanzas. Aunque esto produce un cambio de padre, no viola la restricción.

Sin embargo, dicha violación existirá si la especialidad de estudiante 400 se cambia a CONTABILIDAD. Cuando se hace esto, FINANZAS ya no tiene ningún estudiante, y también se viola la restricción M-M entre ESPECIALIDAD y ESTUDIANTE.

La figura 10-19 presenta reglas para evitar fragmentos en cada uno de estos tipos de restricciones. La figura 10-19(a) muestra acciones sobre el renglón padre, y la figura 10-19(b), acciones en los renglones hijos. Como estas figuras lo indican, las posibles acciones son insertar nuevos renglones, modificar los datos llave y eliminar renglones. La

➤ FIGURA 10-19

Reglas para evitar fragmentos: (a) condiciones para permitir cambios en los registros padres, y (b) condiciones para permitir cambios en los registros hijos

Acción propuesta en padre

Tipo de relación	*Insertar*	*Modificar (llave)*	*Eliminar*
M-M	Crear cuando menos un hijo	Cambiar llaves coincidentes de todos los hijos	Eliminar todos los hijos o reasignar todos los hijos
M-O	OK	Cambiar llaves coincidentes de todos los hijos	Eliminar todos los hijos o reasignar todos los hijos
O-M	Insertar nuevo hijo o Existe hijo apropiado	Cambiar la llave de cuando menos un hijo o Existe hijo apropiado	OK
O-O	OK	OK	OK

(a)

Acción propuesta en hijo

Tipo de relación	*Insertar*	*Modificar (llave)*	*Eliminar*
M-M	Existe padre o Crear padre	Existe padre con un nuevo valor (o crear uno) y Existe hermano	Existe hermano
M-O	Existe padre o Crear padre	Existe padre con un nuevo valor o Crear padre	OK
O-M	OK	Existe hermano	Existe hermano
O-O	OK	OK	OK

(b)

figura 10-19 lista las reglas para relaciones uno a muchos; las reglas para relaciones uno a uno son similares.

RESTRICCIONES PARA ACTUALIZAR LOS RENGLONES PADRE. El primer renglón de la figura 10-19(a) se refiere a restricciones M-M. Sólo se puede insertar un nuevo renglón padre si al mismo tiempo se creó cuando menos un renglón hijo, lo cual se puede hacer insertando un nuevo renglón hijo o reasignando un hijo de un padre diferente (no obstante, esta ultima acción puede ocasionar en sí misma una violación a la restricción).

En una relación M-M sólo se permite un cambio de la llave de un padre si también se modifican los nuevos valores correspondientes a la llave externa en los renglones hijo (es posible reasignar todos los hijos a otro padre y después crear cuando menos un nuevo hijo para el padre, pero esto se hace muy rara vez). Así, se permite cambiar la factura en CUENTA-HOTEL siempre y cuando también se cambie Factura en todos los renglones CARGO-DIARIO. Observe que si se usan las llaves sustitutas esta acción nunca ocurrirá.

Por último, un padre de una relación M-M se puede eliminar siempre y cuando todos los hijos también sean eliminados o reasignados.

En lo que se refiere a las restricciones M-O, se puede agregar un nuevo padre sin ninguna restricción, debido a que no necesitan tener hijos. Para la relación en la figura 10-18(b) se puede agregar un nuevo renglón PROFESOR sin restricción. Sin embargo, sólo se permite un cambio en el valor de la llave padre si los valores correspondientes en los renglones hijos también se cambian. Si un PROFESOR en la relación de la figura 10-18(b) cambia su llave, también se debe alterar el valor del Asesor en todos los renglones de los estudiantes que asesora el profesor.

Finalmente, en una relación con una restricción M-O, el renglón padre se puede suprimir sólo si todos los hijos son eliminados o reasignados. Para la relación PROFESOR-ESTUDIANTE, probablemente todos los renglones de estudiante serían reasignados.

Para las restricciones O-M, sólo puede insertarse un padre si al mismo tiempo se agrega cuando menos un hijo, o si éste ya existe. Por ejemplo, para la relación O-M entre CLUB y ESTUDIANTE de la figura 10-18(c) se puede agregar un nuevo club sólo si se puede crear un renglón ESTUDIANTE (ya sea agregando un nuevo estudiante, o cambiando el valor de CLUB en un ESTUDIANTE). Alternativamente puede existir un renglón adecuado de estudiante.

De manera similar, la llave del padre en una relación O-M sólo se puede cambiar si se crea un hijo o si ya existe un renglón hijo adecuado. Esto es, el Club de Esquí puede cambiar su nombre por el de Buceo sólo si cuando menos un esquiador está dispuesto a unirse a Buceo, o si un estudiante se ha inscrito ya en Buceo. No hay restricciones para la eliminación de un renglón padre en una relación O-M.

En la figura 10-18(d) se muestra el último tipo de restricción de relación: O-O. No existe ninguna restricción a las actualizaciones en los renglones de una relación O-O. Los renglones PROYECTO y EMPLEADO se pueden actualizar conforme sea necesario.

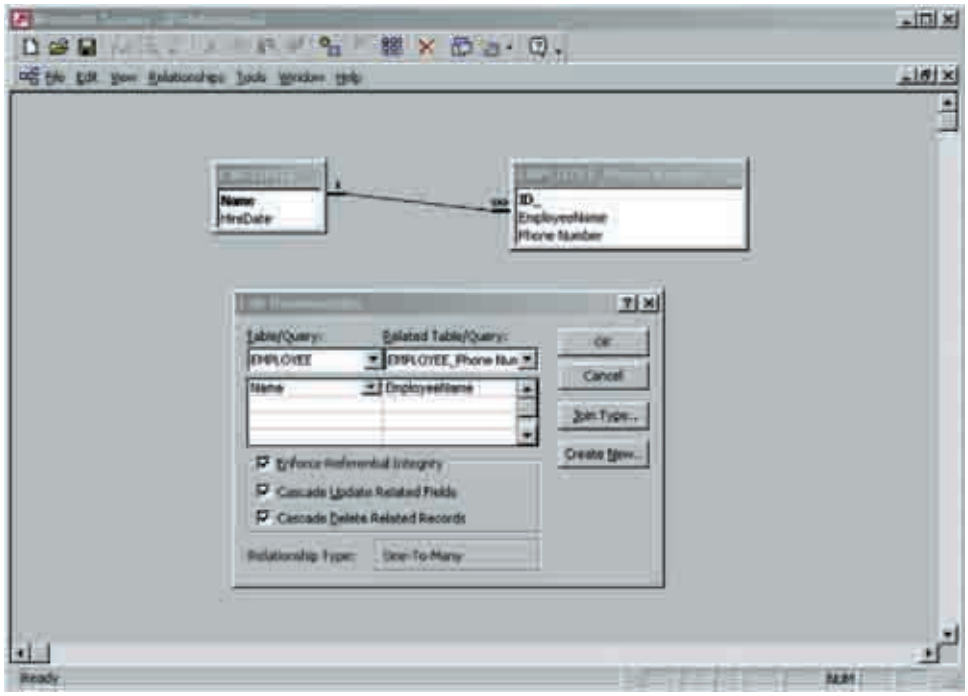
RESTRICCIONES A LA ACTUALIZACIÓN DE RENGLONES HIJO. Las reglas para evitar fragmentos cuando se actualizan renglones hijo se muestran en la figura 10-19(b) y son similares a las de la figura 10-19(a). La única diferencia notable es que, en varios casos, los renglones hijo se pueden modificar o eliminar siempre y cuando existan renglones hermano. Por ejemplo, en una restricción M-M, un renglón hijo se puede eliminar siempre y cuando existan hermanos. (¡El último hijo nunca se va de casa!) Para la restricción M-M de la figura 10-18(a), se puede eliminar un renglón particular CARGO-DIARIO siempre y cuando quede cuando menos uno de ellos.

Con excepción de las consideraciones relacionadas con los hermanos, las reglas para evitar fragmentos cuando se procesan renglones hijo son similares a las de los padres. Asegúrese de entender cada enunciado de la figura 10-19(b).

USO DEL DBMS PARA IMPONER RESTRICCIONES DE CARDINALIDAD. Considerando el análisis anterior es útil tomar en cuenta las restricciones en la figura 10-19 en el contexto de las facilidades de definición de la relación en Access. Suponga que

➤ FIGURA 10-20

Propiedades de relación en Access



una base de datos tiene un objeto EMPLEADO con múltiples números de teléfono. Para mostrar la relevancia de las columnas al centro de la figura 10-19 suponga que la llave de EMPLOYEE (EMPLEADO) no es sustituta, sino EMPLOYEE.Name (EMPLEADO.Nombre).

La figura 10-20 muestra la ventana de definición de relación para este ejemplo. La marca en la caja Enforce Referential Integrity (Imponer Integridad Referencial) indica que Access no permitirá crear un nuevo renglón EMPLOYEE_PhoneNum, a menos que el valor de EmployeeName ya esté presente en un renglón en Employee. Éste es el significado de integridad referencial, como se analizó previamente. La segunda marca, en Cascade Update Related Fields (Campos relacionados de actualización en cascada), significa que si se cambia el nombre de un empleado en la tabla Employee, entonces ese cambio se propagará a todos los renglones relacionadas con EMPLOYEE_PhoneNum. Esto impone las reglas en las columnas de enmedio de la figura 10-19. (Nuevamente, esto no sería necesario si la llave sustituta se usara en su lugar.)

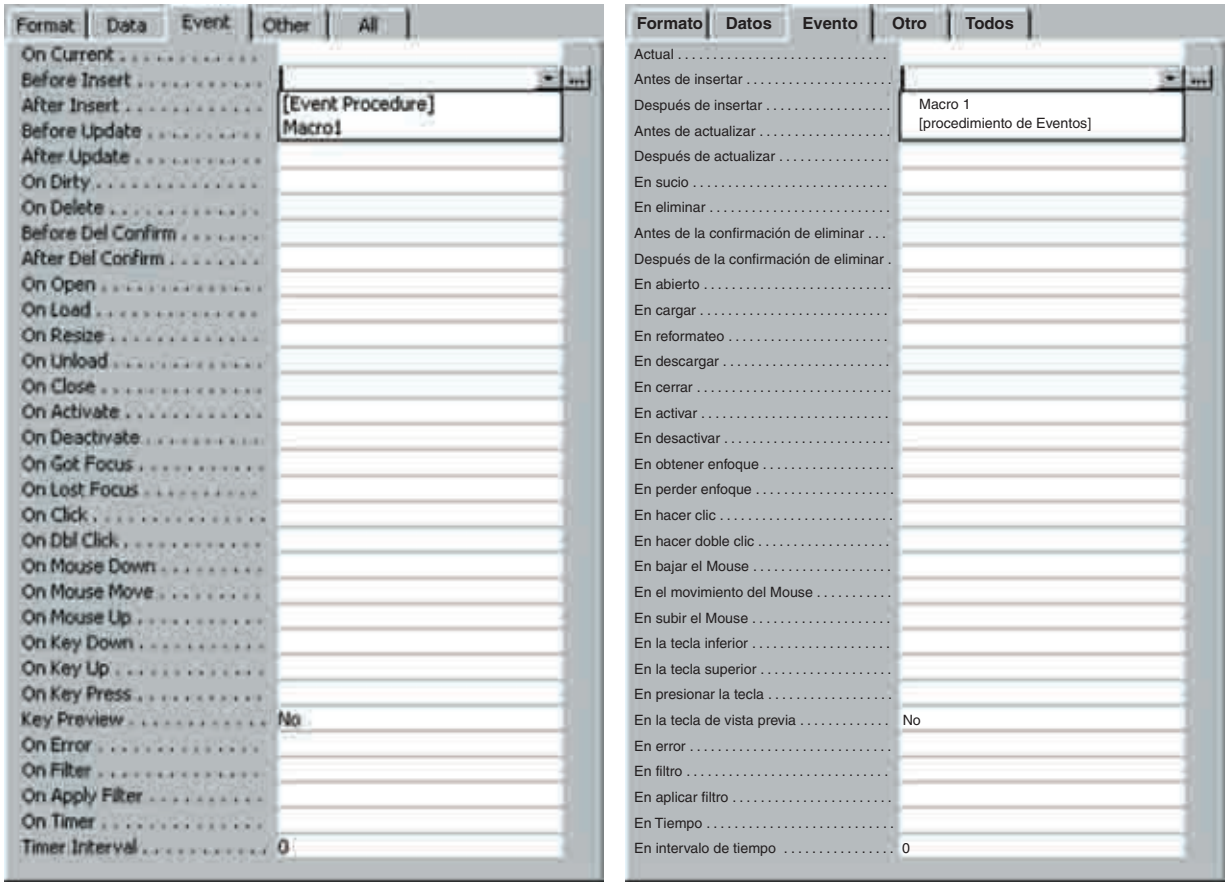
La última marca, en Cascade Delete Related Records (Registros relacionados de eliminación en cascada), significa que cuando se borra un renglón EMPLOYEE se eliminarán también todos los que están conectados con EMPLOYEE_PhoneNum. Esto es similar a lo que se analizó anteriormente en la sección de eliminación de instancias de vistas.

Esta característica de Access direcciona las reglas en la segunda y tercera columnas de la figura 10-19(a) y las de la primera y segunda columnas de la figura 10-19(b). No direcciona las reglas en la primera columna de la figura 10-19(a), tampoco en la tercera columna de la figura 10-19(b). Esas reglas necesitan ser codificadas en procedimientos almacenados (stored procedures) o en la aplicación.

Restricciones de las reglas del negocio

Las restricciones de las reglas del negocio son específicas para la lógica y los requerimientos de una aplicación determinada. Surgen de los procedimientos y las políticas que existen en la empresa los cuales usará la aplicación de la base de datos. Ejemplos de reglas del negocio en una aplicación de ventas son:

➤ FIGURA 10-21 *Eventos para formas de Access*



- Ningún cheque de comisión puede exceder el 50% del total de la "bolsa de comisión"
- No se generará ninguna cancelación de pedido si el valor total de los artículos es menor a $200
- Los costos de envío no se le cobran a los clientes distinguidos
- Los vendedores no pueden crear pedidos para ellos mismos
- Para que un empleado sea gerente de ventas primero debe ser vendedor

Debido a que las reglas del negocio son dependientes de la aplicación, no existen características genéricas de los productos DBMS para imponerlas. Más bien, los productos DBMS proporcionan medios para insertar un código antes o después de los eventos más importantes. La figura 10-21 muestra una lista de los eventos que pueden ser capturados en las formas de Access. En esta figura, el programador está en el proceso de añadir lógica al evento Antes de insertar (Before Insert). La lógica puede tener una de varias formas: un procedimiento del evento en el lenguaje de expresión de Access, o de Visual Basic (u otro lenguaje de programación), o un macro de Access. Todos los datos en la forma y en la base de datos son accesibles para el procedimiento del evento o macro. Así, se pueden imponer cualesquiera de las reglas de la lista de viñetas mediante los acontecimientos que se capturan.

Con los productos servidores DBMS tales como ORACLE y SQL Server, se usa un medio similar. La lógica puede ser codificada en disparadores (trigger), los cuales son procedimientos almacenados (stored procedures) que se invocan cuando tienen lugar eventos en la base de datos. Los eventos que se pueden capturar son similares a los que se muestran en la figura 10-21.

➤ SEGURIDAD Y CONTROL

La cuarta función principal de una aplicación de base de datos listada en la figura 10-1 es proporcionar mecanismos de seguridad y control. El objetivo es crear aplicaciones en las que sólo los usuarios autorizados puedan realizar actividades apropiadas en el tiempo correcto.

SEGURIDAD

La mayoría de los productos DBMS proporcionan nombre del usuario y contraseña de seguridad. Una vez que el usuario se registra, se puede limitar el acceso a ciertas formas, reportes, tablas e incluso a columnas de tablas. Esto es tan apropiado y útil como se quiera. Sin embargo, no ayuda a limitar los datos que los usuarios pueden ver.

Por ejemplo, en la aplicación de la base de datos de recursos humanos cada empleado sólo debería poder ver sus propios registros. Ciertos empleados de recursos humanos deberían poder consultar alguna información acerca de todos los empleados, y los gerentes de recursos humanos deberían poder ver toda la información acerca de todos los empleados.

Limitar el acceso a ciertas formas y reportes no ayuda. Cada empleado necesita ver la Forma del Empleado; la restricción necesita ser que el empleado sólo pueda ver la información de la forma que le pertenece (con las excepciones señaladas). Algunas veces usted escuchará los términos **seguridad horizontal** y **seguridad vertical.** Para entenderlos, piense en una tabla. La seguridad vertical limitaría el acceso a ciertas columnas, pero se podrían ver todos los renglones. La seguridad horizontal limitaría el acceso a ciertos renglones, pero se podrían ver todas las columnas.

Las aplicaciones que limitan a los usuarios a ciertas formas, reportes, tablas, o columnas proporcionan seguridad vertical. Las que limitan a los usuarios a ciertos datos en formas, reportes, tablas o columnas proporcionan seguridad horizontal. Los nombres de usuarios y las contraseñas se pueden usar con facilidad para proporcionar seguridad vertical. La seguridad horizontal generalmente requiere que el programador escriba un código de aplicación.

Por ejemplo, para proporcionar seguridad horizontal en la aplicación de empleado, el código de aplicación obtendría el nombre del usuario del sistema de seguridad del DBMS y limitaría el acceso a los renglones que contienen el nombre, o que están ligadas a renglones con ese nombre a través de joins. Una manera de hacer esto es anexar el nombre del usuario como una cláusula WHERE en los enunciados SQL.

Debido a que cada situación es diferente, no podemos decir más. Sólo esté consciente de que cuando los productos DBMS dicen que apoyan la seguridad, con frecuencia sólo significa que tienen seguridad vertical vía el nombre del usuario y la contraseña.

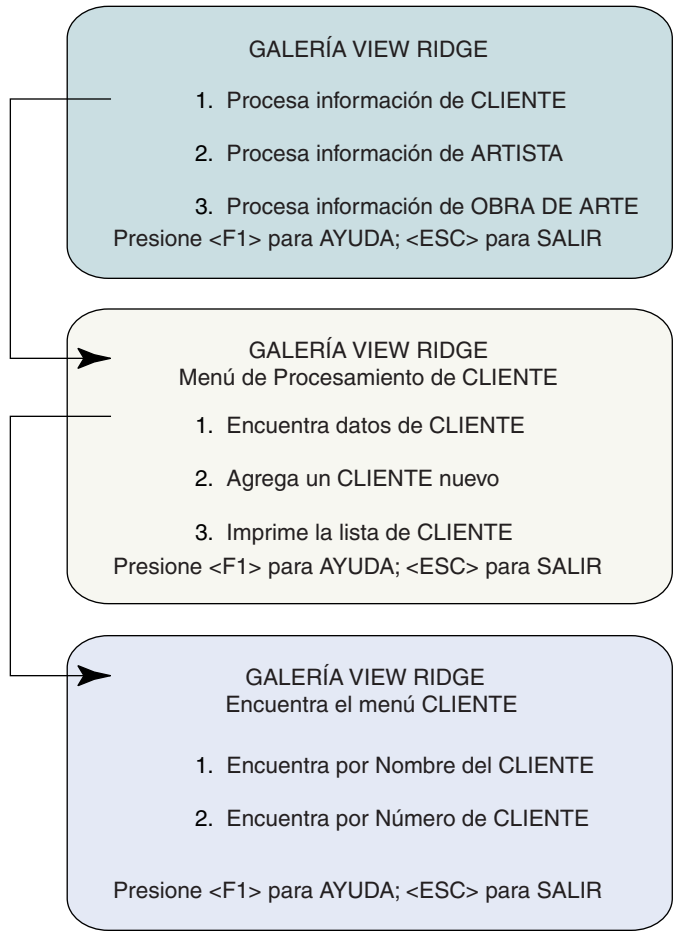
CONTROL

La mayoría de las aplicaciones de base de datos proporcionan control a través de menúes. La figura 10-22(a) muestra un sistema de menúes para una aplicación pre-GUI, y la figura 10-22(b) muestra los mismos menúes para una aplicación GUI.

Los menúes en la figura 10-22 son estáticos. El control más eficaz se puede proporcionar cambiando dinámicamente el contenido del menú conforme el usuario cambia el contexto. Puede ver menúes de ese tipo en Access cuando la barra de herramientas cambia, dependiendo de si usted está en las herramientas de definición de la tabla, de definición de la forma, o de definición del reporte. Los programadores de aplicación de la base de datos pueden usar una estrategia similar para cambiar las opciones de menú, dependiendo de la forma o reporte que un usuario esté viendo, e incluso de la acción que el usuario esté llevando a cabo en la forma. Con el uso de Access, un programador puede cambiar opciones de menú cuando captura eventos como los que se muestran en la figura 10-21, y reestructurar drásticamente las opciones del menú.

➤ FIGURA 10-22

Jerarquía de menúes de la galería View Ridge: (a) sin usar una interfaz de usuario gráfica, y (b) usando una interfaz de usuario gráfica



(a)

(b)

Un tipo diferente de control es el que se refiere a las transacciones. En el capítulo 11 aprenderá los medios de control del procesamiento multiusuarios para que las acciones de uno no tengan consecuencias inapropiadas en las acciones de otro. Una parte clave del control multiusuarios es identificar las fronteras de trabajo que se deben completar como unidad, a veces se les llama **fronteras de transacción.** Por ejemplo, la serie de enunciados SQL para crear la vista que se muestra al principio de este capítulo se necesita completar como una sola transacción.

No anticiparemos el análisis en ese capítulo, sólo diremos que especificar las fronteras de la transacción es trabajo de la aplicación. Por lo general esto se lleva a cabo mediante la ejecución de un enunciado como BEGIN TRANSACTION al comienzo de una unidad de trabajo y END TRANSACTION cuando el trabajo concluye.

Establecer fronteras es fácil si las vistas del usuario están bien diseñadas: las instrucciones de la aplicación BEGIN TRANSACTION al inicio de la vista y END TRANSACTION al final de la vista. Continuaremos con este análisis en el capítulo 12.

➤ LÓGICA DE LA APLICACIÓN

La última función de la aplicación de una base de datos que se lista en la figura 10-1 es ejecutar la lógica de la aplicación. Este tema por lo general se analiza en las clases de desarrollo de sistemas y en diversos textos, y por eso aquí diremos muy poco al respecto.

Las necesidades de la lógica de aplicación surgen de los requerimientos de los sistemas. En un sistema de registro de pedidos la lógica de la aplicación se refiere a cómo llevar inventarios de almacén, cómo encargarse de las existencias insuficientes en inventario, cómo programar un pedido cancelado y actividades similares. En una aplicación de base de datos basada en una forma como la de la galería, el código para avalar esa lógica se relaciona con eventos como los que se muestran en la figura 10-21.

Para otras aplicaciones de la base de datos, donde las formas se materializan mediante la aplicación, en lugar de que sea por medio del DBMS (común en las aplicaciones de macrocomputadoras), la aplicación procesa los datos en una forma similar a la del procesamiento de archivos. La lógica se codifica en un renglón dentro de la aplicación conforme recibe los datos y los pone en el DBMS. Algunos programas de aplicación reciben información de otros; en este caso, también la lógica de la aplicación está contenida en línea.

Así, los medios por los cuales la lógica de la aplicación se implanta en las aplicaciones de la base de datos depende tanto de la lógica como del medio ambiente. Se usan diferentes medios para las aplicaciones de escritorio, servidor del cliente, macrocomputadora, y tecnología de Internet. Usted ha visto cómo capturar hechos en las aplicaciones de escritorio. En los siguientes capítulos consideraremos otros medios.

➤ RESUMEN

Las cinco funciones principales de la aplicación de una base de datos son: (1) crear, leer, actualizar y eliminar las vistas (CRUD), (2) materializar o formatear vistas, (3) imponer restricciones, (4) proporcionar mecanismos de seguridad y control, y (5) ejecutar la lógica del negocio.

Una vista es una lista estructurada de elementos de datos de las entidades u objetos semánticos en el modelo de datos. Una instancia de vista es una que está llena de datos. Debido a que las vistas están estructuradas, un elemento de datos puede aparecer más de una vez. Para leer una vista se emiten uno o más enunciados SQL con el fin de obtener valores de datos. Se requerirá más de un enunciado SQL si la vista incluye dos o más caminos de relaciones a través del esquema. Un recordset es una relación con un empaquetador del objeto de programación.

Crear una vista requiere almacenar uno o más nuevos renglones en tablas, y posiblemente crear o cambiar valores de llaves externas, así como establecer relaciones. Hay tres tipos de actualizaciones: cambiar los datos existentes, cambiar las relaciones, y crear nuevos renglones para los atributos de multivaluados. Eliminar una vista implica borrar uno o más renglones y ajustar las llaves externas. El problema cuando se elimina es saber qué tanto se debe descartar. Hay que suprimir las entidades débiles si la entidad de la que dependen se elimina. También hay que quitar los atributos multivaluado dentro de un objeto semántico. Con algunos productos DBMS las relaciones se pueden marcar para la eliminación en cascada, de tal forma que el DBMS remueva los renglones dependientes que sea apropiado.

Una forma es una pantalla desplegada que se usa para introducir datos y editarlos. Los principios del diseño de formas son que la estructura de ésta debe reflejar la de la vista, la semántica de los datos debe ser gráficamente evidente, y la estructura de la forma debe fomentar la acción apropiada. Las cajas de lista desplegables, los botones de opción y las cajas de verificación se pueden usar para aumentar la utilidad de las formas.

Los reportes también se deben diseñar de tal forma que su estructura refleje la de la vista que materializan. El ordenamiento de los reportes con frecuencia implica la

existencia de otros objetos. Con la mayoría de los editores de reportes es difícil construir reportes que sigan más de una pista multivaluada a través del esquema. Los reportes con frecuencia calculan atributos de datos; por lo general es mejor no almacenar esos atributos en la base de datos.

Las restricciones se pueden imponer ya sea mediante el DBMS o a través de un programa de aplicación. En la mayoría de los casos es mucho mejor para el DBMS aplicarlas cuando sea posible, principalmente debido a que el DBMS es un punto central a través del cual pasan todos los cambios de datos. En algunos casos, el DBMS no tiene las características para imponer restricciones; sin embargo, se deben imponer mediante la aplicación. Las restricciones de dominio imponen la parte física de las definiciones de dominio. Los valores requeridos son otro tipo de restricción. Tener un valor requerido evita la ambigüedad de los valores nulos. Las restricciones de unicidad se aplican mejor con el DBMS; por lo general se hace construyendo índices.

Hay dos tipos de restricciones de relación: integridad referencial y cardinalidad de relación. Las restricciones de integridad referencial se aplican mejor por medio del DBMS. Las restricciones de cardinalidad de relación surgen del establecimiento de la cardinalidad en los vínculos del objeto, de cualquier establecimiento de cardinalidad mínima que no sea cero, o de una cardinalidad máxima que no sea 1 ni N. Excepto en el caso de la cardinalidad 1.1 en el lado de los hijos de las relaciones 1:N, estas restricciones se deben imponer mediante un código de aplicación. Las reglas para la aplicación de restricciones en las relaciones 1:N se resumen en la figura 10-19. Las restricciones de la regla de negocio mediante un código de aplicación que se invoca a través de eventos capturados, de disparadores (triggers), o de programas de aplicación en línea.

La mayoría de los productos DBMS proporcionan el nombre del usuario y la contraseña de seguridad. Esto se usa para proporcionar seguridad vertical; la seguridad horizontal se debe proveer mediante el código de aplicación. La mayoría de las aplicaciones ofrecen control a través de menúes. El mejor control ocurre cuando se cambian los menúes conforme cambia el contexto del usuario. Los programas de aplicación desempeñan un papel importante en la definición de las fronteras de transacción. La lógica de la aplicación se codifica y se invoca capturando eventos, así como a través de otros medios que serán explicados en los siguientes capítulos.

➤ PREGUNTAS DEL GRUPO I

10.1 Enumere las cinco funciones principales de una aplicación de la base de datos.

10.2 Explique el significado del acrónimo CRUD.

10.3 Defina el término *vista* como se utilizó en este capítulo.

10.4 ¿Qué es una instancia de vista?

10.5 Explique en qué difiere una vista de una materialización.

10.6 ¿Un atributo puede aparecer más de una vez en una vista? ¿Por qué?

10.7 ¿Bajo qué condiciones se puede leer una vista con un enunciado SQL?

10.8 ¿En qué casos leer una vista requiere más de un enunciado SQL?

10.9 Explique los dos caminos que existen en la vista de Cliente en la figura 10-4.

10.10 Defina el término *recordset*.

10.11 Describa en términos generales el trabajo que se requiere cuando se crea una instancia de vista.

10.12 ¿Cómo se crean las nuevas relaciones cuando se desarrolla una instancia de vista?

10.13 ¿Qué técnica se puede usar para obtener el valor de una llave sustituta cuando se insertan nuevos renglones en una tabla?

10.14 Liste los tres tipos de cambio que pueden ocurrir cuando se actualiza una instancia de vista.

10.15 Explique cómo cambiar las relaciones 1:N, y las relaciones N:M.

10.16 ¿Cuál es la dificultad principal cuando se escribe un código para eliminar una instancia de vista?

10.17 ¿Cómo puede ayudar un modelo E-R a determinar qué tanto se debe eliminar?

10.18 ¿Cómo puede ayudar un objeto semántico a determinar qué tanto se debe suprimir?

10.19 ¿Qué son las eliminaciones en cascada y por qué son importantes?

10.20 Explique el enunciado "la estructura de la forma debería reflejar la estructura de la vista".

10.21 ¿Cómo se pueden diseñar las formas para hacer la semántica de la información gráficamente evidente?

10.22 ¿Cómo se pueden diseñar las formas para fomentar la acción apropiada?

10.23 Explique el papel de las cajas de listas desplegables, los grupos de opción y las cajas de verificación en el diseño de formas.

10.24 ¿Qué limitación existe para la materialización de reportes de las vistas?

10.25 Explique por qué los valores calculados en reportes por lo regular no deberían almacenarse en la base de datos.

10.26 Explique cómo el requerimiento de objetos de reporte ordenados por un valor cambia el objeto implícito del reporte.

10.27 ¿Por qué las restricciones normalmente deberían ser impuestas por el DBMS y no mediante un programa particular de formas, reportes o aplicación?

10.28 ¿Por qué a veces las restricciones se imponen en programas de aplicación?

10.29 Dé un ejemplo sobre una restricción de dominio y explique cómo se puede imponer con Access.

10.30 Describa la ambigüedad que surge cuando los valores son nulos, así como dos maneras para que éstos valores puedan ser eliminados.

10.31 ¿Por qué el DBMS es el que normalmente debería imponer restricciones de unicidad?

10.32 Describa las dos fuentes de restricciones de cardinalidad.

10.33 Nombre dos tipos de restricción de relación.

10.34 ¿Cuál es la mejor manera de imponer restricciones sobre valores de llave externa?

10.35 ¿Cómo se puede imponer una restricción de cardinalidad 1.1 en el lado de los hijos de una relación 1:N?

10.36 Defina *fragmento* y *huérfano*.

10.37 Explique las entradas en la primera columna de la figura 10-19(a).

10.38 Explique por qué la columna central de la figura 10-19(a) es innecesaria cuando se usan las llaves sustitutas.

10.39 Explique las entradas en la tercera columna de la figura 10-19(a).

10.40 Explique las entradas en la primera columna de la figura 10-19(b).

10.41 Explique las entradas en la tercera columna de la figura 10-19(b).

10.42 Explique por qué la primera columna en la figura 10-19(a) y la tercera columna en la figura 10-19(b) no están impuestas mediante las propiedades de relación de Access que se muestran en la figura 10-20.

10.43 Dé un ejemplo sobre restricción de una regla del negocio que se podría aplicar al modelo de datos de la figura 10-3. Explique cómo se puede imponer esta restricción al capturar un evento.

10.44 Defina seguridad *horizontal* y *vertical*.

10.45 ¿Qué tipo de seguridad se avala con el nombre del usuario y la contraseña?

10.46 ¿Qué tipo de seguridad se debe apoyar mediante un código de aplicación?

10.47 Explique por qué los menúes dinámicos son mejores que los estáticos.

10.48 ¿Cómo es la lógica de negocios conectada a una base de datos cuando se usa Access?

➤ PREGUNTAS DEL GRUPO II

Las preguntas 10.49 a la 10.52 pertenecen a la siguiente vista de Artista, la cual está basada en el modelo de datos de la figura 10-3.

ARTISTA.Nombre
ARTISTA.Nacionalidad
TRANSACCIÓN.FechadeCompra
TRANSACCIÓN.PreciodeVenta. . .
CLIENTE.Nombre
CLIENTE.Teléfono.CódigodeÁrea
CLIENTE.Teléfono.NúmeroLocal
CLIENTE.Nombre. . .

Las elipsis (. . .) se refieren a estructuras que se pueden repetir.

10.49 Codifique los enunciados SQL para leer la instancia de "Mark Tobey" en esta vista.

10.50 Codifique los enunciados SQL para crear una nueva instancia de esta vista. Suponga que usted tiene datos para ARTISTA, una TRANSACCIÓN y muchos CLIENTE.Nombre(s) para la segunda instancia de CLIENTE.Nombre. Suponga que estos datos se localizan en una estructura llamada NuevoArtista. Use sintaxis similar a la del texto.

10.51 Codifique los enunciados SQL para actualizar esta vista como sigue:

a. Cambie la ortografía de Mark Tobey a Mark Toby

b. Cree una nueva Transacción para Mark Toby. Suponga que tiene los datos necesarios de la transacción, obra y cliente en una estructura llamada NuevaTransacción

c. Agregue nuevos clientes interesados en Mark Toby. Suponga que se encuentran almacenados en un conjunto al que usted puede tener acceso con el enunciado "Para cada NuevoCliente.Nombre"

10.52 Codifique los enunciados SQL para eliminar el renglón de Mark Toby y todos los renglones TRABAJO y TRANSACCIÓN relacionados.

➤ PROYECTOS

A. Usando Access, cree la base de datos que se muestra en la figura 10-3. Cree una forma para la vista de Artista que se muestra en la pregunta 10.49. Justifique el diseño de su forma usando los principios de este capítulo. Sugerencia: puede utilizar un asistente (wizard) para crear uno de los subformularios, pero necesitará agregar el segundo de forma manual. También, agregue las cajas de conjunto manualmente después de que haya creado las formas para la subforma.

B. Termine el Proyecto A al final de los capítulos 3 y 4, si aún no lo ha hecho.

1. Liste y describa el propósito de las tres vistas, las tres formas y los tres reportes que considere serán necesarios para esta aplicación.
2. Muestre la estructura de un menú GUI desplegable para esta aplicación. Usando su modelo, diseñe una de las formas para introducir nuevas propiedades. Expli-

que qué tipo de control (caja de texto, lista desplegable) se usa para cada campo. Justifique la estructura usando los conceptos que presentamos en este capítulo.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Lea el proyecto FiredUp al final del capítulo 9. Use las cuatro tablas que se describen ahí para resolver lo siguiente:

A. Construya las siguientes vistas. Use la figura 10-4 como ejemplo.

1. Construya una vista que comience con ESTUFA y contenga todas las tablas y los datos. Llame a la vista ESTUFA_VISTA.
2. Construya una vista que inicie en CLIENTE e incluya todas las tablas y los datos. Llame a la vista CLIENTE_VISTA.
3. Construya una vista que dé comienzo en REGISTRO y contenga todas las tablas, excepto ESTUFA_REPARACIÓN. Llame a la vista REGISTRO_VISTA.
4. Construya una vista que inicie con ESTUFA_REPARACIÓN y contenga todas las tablas y los datos. Llame a la vista ESTUFA_REPARACIÓN_VISTA.

B. Construya enunciados SQL para procesar vistas como sigue. Use el comienzo de SQL de la página 262 como ejemplo.

1. Muestre los enunciados SQL necesarios para leer ESTUFA_VISTA. Suponga que inicia con un NúmerodeSerie en particular.
2. Muestre los enunciados SQL que se necesitan para construir una nueva instancia de REGISTRO_VISTA. Suponga que los datos de estufa que se necesitan ya están en la base de datos, pero falta la información necesaria del cliente.
3. Muestre los enunciados SQL para construir una nueva instancia de ESTUFA_REPARACIÓN. Suponga que los datos de ESTUFA están en la base de datos, pero falta de información de CLIENTE. Registre la ESTUFA mientras anota la reparación.
4. Muestre los enunciados SQL para eliminar todos los registros referentes a una estufa en particular. Use la vista de ESTUFA.

PARTE

PROCESAMIENTO DE BASES DE DATOS MULTIUSUARIO

Los tres capítulos que conforman la parte V describen puntos importantes y problemas de las bases de datos multiusuario y muestran respuestas y soluciones mediante dos productos DBMS (Database management system, por sus siglas en inglés) populares. En el capítulo 11 describimos la administración de la base de datos, así como las principales tareas y técnicas para administrar una base de datos multiusuario. Los dos capítulos siguientes ilustran la implementación de estos conceptos, con el uso de Oracle 8i en el capítulo 12, y con el SQL Server 2000 en el capítulo 13.

CAPÍTULO 11

Administración de bases de datos multiusuario

Si bien es cierto que las bases de datos multiusuario son de gran valor para las organizaciones que las crean y las usan, también es verdad que les plantean problemas difíciles. Para unas, la base de datos multiusuario es complicada de diseñar y desarrollar debido a que maneja muchas consultas de usuarios al mismo tiempo. Además, los requerimientos cambian con el tiempo y dichos cambios necesitan otros cambios en la estructura de la base de datos. Estos cambios de estructura deben ser cuidadosamente planeados y controlados para que un cambio hecho para un grupo no cause problemas en otro. Además, cuando los usuarios procesan una base de datos con frecuencia se necesitan controles especiales para asegurar que el trabajo de un usuario no afecte inapropiadamente el de otro. Como verá, este tema es importante y complicado.

En organizaciones grandes se necesita definir e imponer los derechos de procesamiento y las responsabilidades. Por ejemplo, ¿qué sucede cuando un empleado deja la empresa? ¿Cuándo se pueden eliminar sus registros? Si se trata de la elaboración de la nómina, tendrá que ser después del último periodo de pago; en el caso del reporte trimestral, al final del periodo; con respecto a los impuestos, cuando termine el año, y así sucesivamente. Es claro que ningún departamento puede decidir en forma unilateral cuándo eliminar la información. Se pueden hacer comentarios similares con respecto a la inserción y al cambio de valores de los datos. Por esta y otras razones es necesario desarrollar sistemas de seguridad para permitir que sólo los usuarios autorizados lleven a cabo acciones permitidas dentro de horarios también establecidos.

Las bases de datos se han convertido en las componentes claves de operaciones organizacionales, e incluso en componentes claves de la plusvalía de la empresa. Por desgracia, las bases de datos fallan y ocurren desastres. Así, los planes de respaldo y recuperación, las técnicas y los procedimientos eficaces son esenciales.

Por último, conforme pasa el tiempo se necesitará cambiar el mismo DBMS para mejorar su rendimiento e incorporar nuevas características y versiones a medida que se realicen los cambios en el sistema operativo. Todo esto requiere de una administración cuidadosa.

Con el fin de asegurarse que estos problemas están identificados y son resueltos, la mayoría de las empresas cuentan con una oficina de administración de base de datos.

Comenzaremos por mencionar las tareas de dicha oficina, y después describiremos la combinación de software, y las prácticas y los procedimientos manuales que se usan para desempeñar estas tareas. En los dos capítulos siguientes analizaremos las características y funciones de Oracle 8i y del SQL Server 2000, respectivamente, para tratar estos puntos.

➤ ADMINISTRACIÓN DE LA BASE DE DATOS

En la industria se usan los términos **administración de datos** y **administración de la base de datos.** En algunos casos los términos se consideran sinónimos; en otros, tienen diferentes significados. En este libro usamos el término *administración de datos* para referirnos a una función que se aplica a toda la empresa. El término *administración de la base de datos* se refiere a la función que es específica para una base de datos en particular, incluyendo las aplicaciones que la procesan. Este capítulo aborda la administración de la base de datos. En el capítulo 17 se analizará la administración de datos.

Las bases de datos varían considerablemente en tamaño y alcance, desde una base de datos personal de usuario único hasta una gran base de datos interorganizacional como el sistema de reservaciones de una aerolínea. Todas esas bases de datos requieren administración, aunque las tareas a realizar varían en cuanto a su complejidad. Por ejemplo, en una base de datos personal el usuario sigue procedimientos simples para respaldar su información y conservar registros mínimos con fines documentales. En este caso, esa persona que usa la base de datos ejecuta las funciones del DBA (Database Administration, por sus siglas en inglés) aun cuando probablemente no esté consciente de que lo hace.

Para aplicaciones de bases de datos multiusuario, la administración se vuelve más importante y difícil. Por lo tanto, usualmente tiene reconocimiento formal. Para algunas aplicaciones, esta función se le asigna a una o dos personas de tiempo parcial. En el caso de grandes bases de datos de Internet o Intranet, con frecuencia las responsabilidades de la administración de la base de datos llevan mucho tiempo y varían mucho como para que sean manejadas incluso por una sola persona de tiempo completo. Mantener en funcionamiento una base de datos con docenas o centenas de usuarios requiere mucho tiempo, así como conocimientos técnicos y técnicas diplomáticas que usualmente se manejan en una oficina de administración de base de datos. Al jefe de la oficina con frecuencia se le conoce como **administrador de la base de datos,** en este caso, el acrónimo **DBA** se refiere tanto a la oficina como a la administración.

La responsabilidad del DBA es facilitar el desarrollo y el uso de la base de datos. Por lo general, esto significa balancear las metas conflictivas de protección de la base de datos y maximizar su disponibilidad y beneficio para los usuarios. La DBA es responsable del desarrollo, operación y mantenimiento de la base de datos y de sus aplicaciones. En la figura 11-1 se muestran las tareas específicas. Consideraremos cada una de éstas en las siguientes secciones.

➤ FIGURA 11-1

Resumen de las tareas de la administración de la base de datos

> Administración de la estructura de la base de datos

> Control de los procesos repetitivos

> Administración de los derechos y responsabilidades del procesamiento

> Desarrollo de la seguridad de la base de datos

> Servicios de recuperación de la base de datos

> Administración del DBMS

> Mantenimiento del repositorio de datos

Administración de la estructura de la base de datos

La administración de la estructura de la base de datos incluye la participación en el diseño inicial y su implementación, así como controlar y administrar los cambios en ésta. Lo ideal es que el DBA participe en el desarrollo de la base de datos y sus aplicaciones, colabore en el estudio de los requisitos y en la evaluación de las alternativas, incluyendo el DBMS que será usado, y ayude a diseñar la estructura de la base de datos. En el caso de aplicaciones para grandes empresas u organizaciones, el administrador de la base de datos por lo general supervisa el trabajo del personal técnico que diseña la base.

Como se describió en el capítulo 8, crear la base de datos implica varias tareas diferentes. Primero se crea la base de datos y se asigna el espacio para ésta y sus registros. Después se generan las tablas, se crean los índices y se escriben los procedimientos almacenados y los disparadores. En los dos capítulos siguientes analizaremos ejemplos de todo esto. Una vez que se crean las estructuras de la base de datos, se ingresan éstos. La mayoría de los fabricantes de DBMS proporcionan utilidades para insertar grandes cantidades de datos.

CONTROL DE CONFIGURACIÓN. Después de que se implementan la base de datos y sus aplicaciones, los cambios en los requerimientos son inevitables. A partir de éstos pueden surgir nuevas necesidades, desde cambios en el medio ambiente del negocio, hasta cambios en sus normas y políticas, y así sucesivamente. Cuando los cambios en los requerimientos afectan la estructura de la base de datos, debe tenerse mucho cuidado porque pocas veces involucran sólo una aplicación.

Por lo tanto, una administración eficaz de base de datos debe incluir procedimientos y políticas por medio de las cuales los usuarios puedan registrar sus necesidades de cambios, y absolutamente todos tengan oportunidad de analizar sus repercusiones y se pueda tomar una decisión en conjunto con respecto a implementar o no los cambios propuestos.

Debido al tamaño y complejidad de las bases de datos y sus aplicaciones, a veces los cambios tienen resultados inesperados. Así, la DBA debe estar preparada para reparar la base de datos y reunir información suficiente para diagnosticar y corregir el problema que ocasionó el daño. La base de datos es más susceptible a fallar después de un cambio en su estructura.

DOCUMENTACIÓN. La responsabilidad final de la DBA en cuanto a la administración de la estructura de la base de datos es la documentación. Es muy importante saber qué cambios se hicieron, y cómo y cuándo fueron hechos. Un cambio en la estructura de la base de datos podría ocasionar un error que no se manifieste hasta seis meses después; sin la documentación apropiada del cambio, diagnosticar el problema será casi imposible. Se puede requerir repetir la ejecución de trabajos para identificar el punto en el que ciertos síntomas aparecen primero, y por esta razón es importante conservar un registro de los procedimientos de prueba y de las pruebas que se hicieron para verificar un cambio. Si los procedimientos de prueba son estándar, y si se usan formas de prueba y métodos para mantener registros, documentar los resultados de las pruebas no llevará mucho tiempo.

Aunque conservar la documentación es tedioso y poco satisfactorio, el esfuerzo se recompensa cuando sucede un desastre y la documentación es la diferencia entre resolver o no un problema mayor (y costoso). Actualmente están surgiendo varios productos que facilitan elaborar la documentación. Por ejemplo, se pueden usar muchas herramientas CASE para documentar diseños lógicos de la base de datos. Se puede usar software de control de versiones para registrar los cambios. Los diccionarios de datos proporcionan reportes y otros productos para leer e interpretar las estructuras de la base de datos.

Otra razón para documentar con cuidado los cambios en la estructura de la base de datos es tener datos históricos apropiados. Si, por ejemplo, mercadotecnia quiere analizar las ventas de tres años que han estado en los archivos durante dos años, será necesario saber qué estructura se usaba en el momento en que se registraron los datos la última vez. Se pueden usar los registros que muestran los cambios en la estructura pa-

➤ FIGURA 11-2

Resumen de las responsabilidades del DBA para la administración de la estructura de la base de datos

Participación en la base de datos y en el desarrollo de la aplicación
- Ayudar en la etapa de requerimientos y en la evaluación de las alternativas
- Desempeñar un papel activo en el diseño de la base de datos y la creación

Facilitar cambios en la estructura de la base de datos
- Buscar soluciones para toda la comunidad
- Evaluar el impacto en todos los usuarios
- Proporcionar un foro de control de configuración
- Estar preparados para los problemas después de que se hacen los cambios
- Conservar la documentación

ra contestar esa pregunta. Surge una situación similar cuando se debe usar una copia de los datos de respaldo de hace seis años para reparar una base de datos dañada (aunque esto no debería suceder, a veces pasa). Se puede usar la copia de respaldo para reconstruir la base de datos al estado en que estaba al momento del respaldo. Después se hacen los cambios estructurales y las transacciones en orden cronológico para restablecer la base de datos a su estado actual. La figura 11-2 resume las responsabilidades del DBA en cuanto a la administración de la estructura de la base de datos.

➤ CONTROL DE CONCURRENCIA

Se toman medidas de control de concurrencia para asegurar que el trabajo de un usuario no influya negativamente en el de otro. En algunos casos, estas medidas aseguran que un usuario obtendrá el mismo resultado cuando procesa con otros usuarios, que el que obtendría si estuviera procesando solo. En otros casos, esto significa que su trabajo se ve influenciado por el de otros, pero de manera anticipada.

Por ejemplo, en un sistema de registro de pedidos, un usuario debería poder ingresar un pedido y obtener el mismo resultado sin importar que haya otro usuario, o cientos de éstos. Por otra parte, un usuario que está imprimiendo un reporte del estado actual del inventario quizás quiera obtener los cambios en el procesamiento de datos de otros usuarios, a pesar de que exista el peligro de que estos cambios sean abortados más tarde.

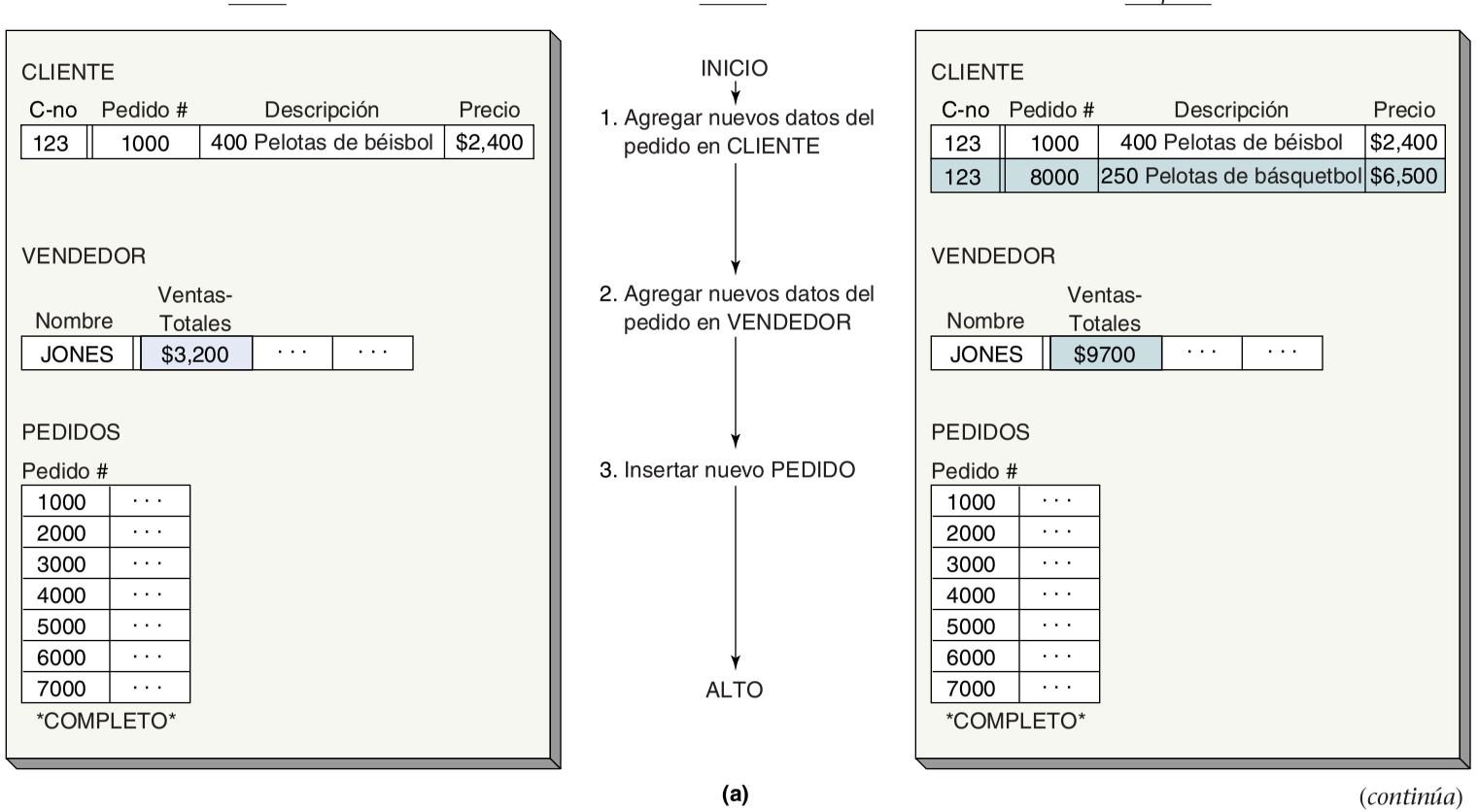
Por desgracia, ninguna técnica o mecanismo de control de concurrencias es ideal para todas las circunstancias. Todas implican decidir entre varias alternativas. Por ejemplo, un usuario puede lograr un control de concurrencia muy estricto al aplicar locks a la base de datos completa, pero mientras esté trabajando con ésta nadie más podrá hacer nada. Ésta es una protección muy estricta, pero a un precio muy alto. Como verá, hay otras medidas disponibles que son más difíciles de programar o aplicar, pero que permiten un rendimiento más eficaz. Incluso hay otras medidas que maximizan el rendimiento eficaz, pero con un nivel bajo de control de concurrencia. Cuando se diseñan aplicaciones de base de datos multiusuarios es necesario elegir una de estas alternativas.

La necesidad de transacciones atómicas

En la mayoría de las aplicaciones de la base de datos los usuarios transmiten su trabajo en forma de **transacciones,** que también se conocen como **unidades lógicas de trabajo** (LUWs, Logical Units of Work, por sus siglas en inglés). Una transacción (o LUW) es una serie de acciones que se llevarán a cabo en la base de datos, de tal manera que todas se ejecuten con éxito, o que ninguna se realice por completo; en ese caso la base de datos permanecerá sin cambios. Algunas veces esta transacción se llama **atómica,** puesto que se ejecuta como una unidad.

➤ FIGURA 11-3

Comparación de resultados de la aplicación de acciones seriales contra una transacción de pasos múltiples: (a) dos de tres actividades que concluyeron con éxito, lo que originó anomalías de la base de datos, y (b) no se realizó ningún cambio debido a que la transacción total no tuvo éxito

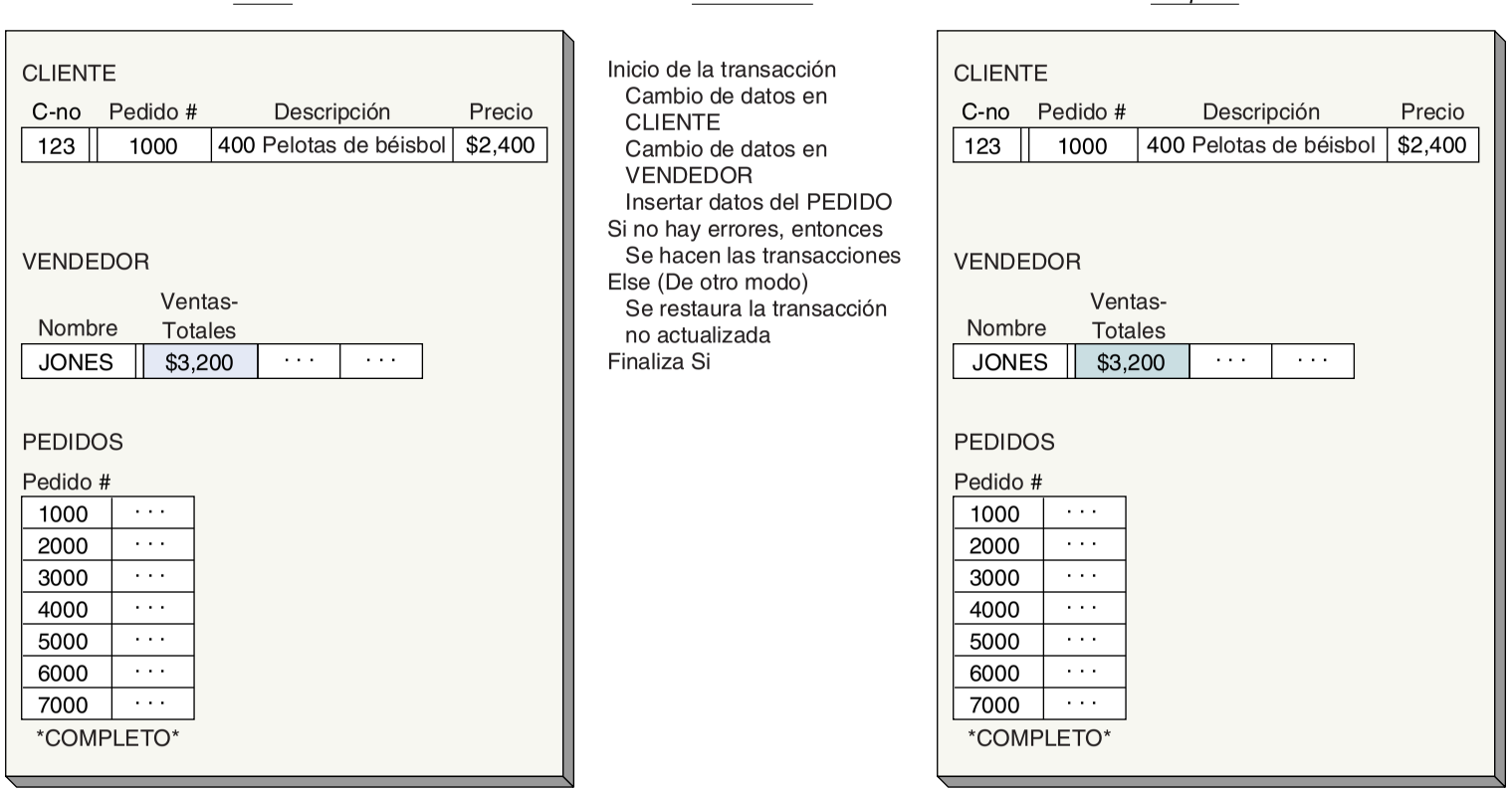


(a)

(*continúa*)

➤ FIGURA 11-3

(Continuación)



(b)

Considere la siguiente secuencia de acciones de la base de datos que puede ocurrir cuando se registra un pedido nuevo:

1. Cambia el registro del cliente y aumenta Cantidad que Adeuda.
2. Cambia el registro del vendedor y aumenta Comisión que se Debe.
3. Ingresa el nuevo registro de pedido en la base de datos.

Suponga que falla el último paso, quizás porque no hay suficiente espacio en el archivo. Imagine la confusión que habría si se hicieran los dos primeros cambios, pero no el tercero. Al cliente se le cobraría por un pedido que nunca recibió y un vendedor recibiría la comisión por un pedido que nunca fue enviado. Obviamente, se necesita considerar estas tres acciones en conjunto —ya sea que se realicen todas o que ninguna se lleve a cabo.

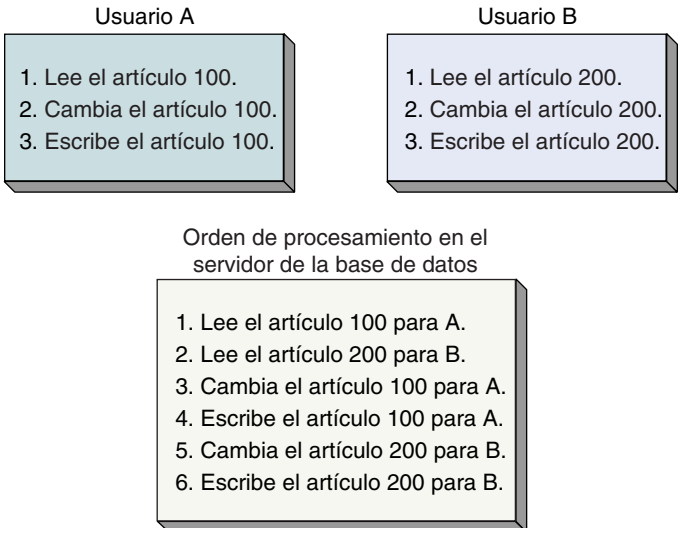
En la figura 11-3 se comparan los resultados de la realización de estas actividades como una serie de pasos independientes (figura 11-3[a]) y como una transacción atómica (figura 11-3[b]). Observe que cuando se realizan los pasos atómicamente y falla uno, no se realiza ningún cambio en la base de datos. También note que las instrucciones Iniciar transacción (Start Transaction), Confirmar transacción (Commit Transaction), o Deshacer transacción (Rollback Transaction) los debe emitir el programa de aplicación para que se marquen los límites de la lógica de transacción. La forma particular de estas instrucciones varía de un producto DBMS a otro.

PROCESAMIENTO DE TRANSACCIONES CONCURRENTES. Cuando se han procesado dos transacciones en una base de datos al mismo tiempo, se les llama *transacciones concurrentes*. Aun cuando al usuario le pueda parecer que las transacciones concurrentes se han procesado en forma simultánea, esto no puede ser verdad, ya que la CPU de la máquina que está procesando la base de datos sólo puede ejecutar una instrucción a la vez. Por lo general las transacciones están mezcladas, lo que significa que el sistema operativo alterna los servicios de la CPU entre las tareas para que alguna parte de cada una de ellas se realice en un intervalo determinado. Este cambio entre las tareas se realiza tan rápidamente que dos personas usando browsers, y sentadas lado a lado procesando la misma base de datos, quizás crean que sus transacciones fueron completadas simultáneamente, pero, en realidad, las dos están mezcladas.

La figura 11-4 muestra dos transacciones concurrentes. La del usuario A lee el artículo 100, lo cambia, y lo rescribe en la base de datos. La transacción del usuario B lleva a cabo las mismas acciones, pero en el artículo 200. La CPU procesa lo del usuario A hasta que encuentra una interrupción de I/O o alguna otra causa de retraso. El sistema operativo cambia el control al usuario B. La CPU procesa ahora lo del usuario B hasta que encuentra una interrupción; en este punto el sistema operativo pasa el control de regreso al usuario A. Para los usuarios, el procesamiento parece simultáneo, pero en realidad está mezclado, o es concurrente.

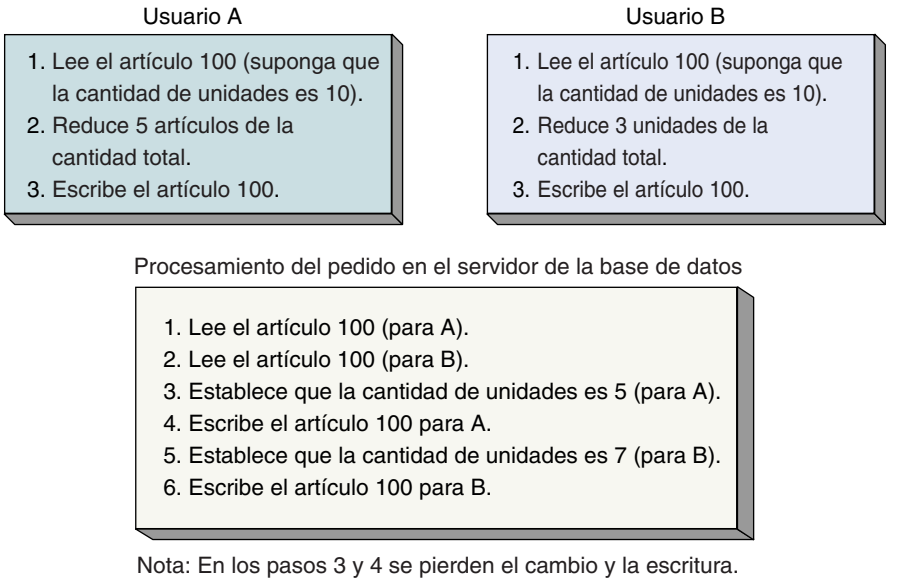
➤ FIGURA 11-4

Ejemplo de procesamiento concurrente de las tareas de dos usuarios



➤ FIGURA 11-5

Problema de pérdida de una actualización



PROBLEMA DE PÉRDIDAS EN LA ACTUALIZACIÓN. El procesamiento concurrente en la figura 11-4 no tiene problemas porque dos usuarios están procesando datos diferentes. Pero supongamos que ambos quieren procesar el artículo 100. Por ejemplo, el usuario A quiere ordenar cinco unidades del artículo 100, y el B quiere ordenar tres unidades del mismo artículo.

La figura 11-5 ilustra el problema. El usuario A lee el registro del artículo 100 en el área de trabajo de un usuario. De acuerdo con el registro, existen 10 unidades en inventario. El usuario B lee el registro del artículo 100 en otra área de trabajo. Nuevamente, de acuerdo con el registro existen 10 unidades en inventario. Ahora el usuario A toma cinco, disminuye la cantidad de elementos en su área de trabajo a cinco y rescribe el registro para el artículo 100. Luego el usuario B toma tres, disminuye la cantidad en su área de trabajo a siete, y rescribe el registro para el elemento 100. La base de datos ahora muestra, incorrectamente, que existen siete elementos en el inventario del elemento 100. Revisemos: comenzamos con 10 en inventario, el usuario A toma cinco, el usuario B toma tres y la base de datos muestra que hay siete en inventario. Obviamente, hay un problema.

Ambos usuarios tuvieron datos que eran correctos al momento de obtenerlos. Pero cuando el usuario B leyó el registro, el A ya tenía una copia que estaba por actualizarse. Esta situación se llama **problema de pérdida de la actualización** o **problema concurrente en la actualización.** Existe otro problema similar, llamado **problema de la lectura inconsistente.** En éste, el usuario A lee información que ha sido procesada por una parte de la transacción del usuario B. Como resultado, el A lee incorrectamente los datos.

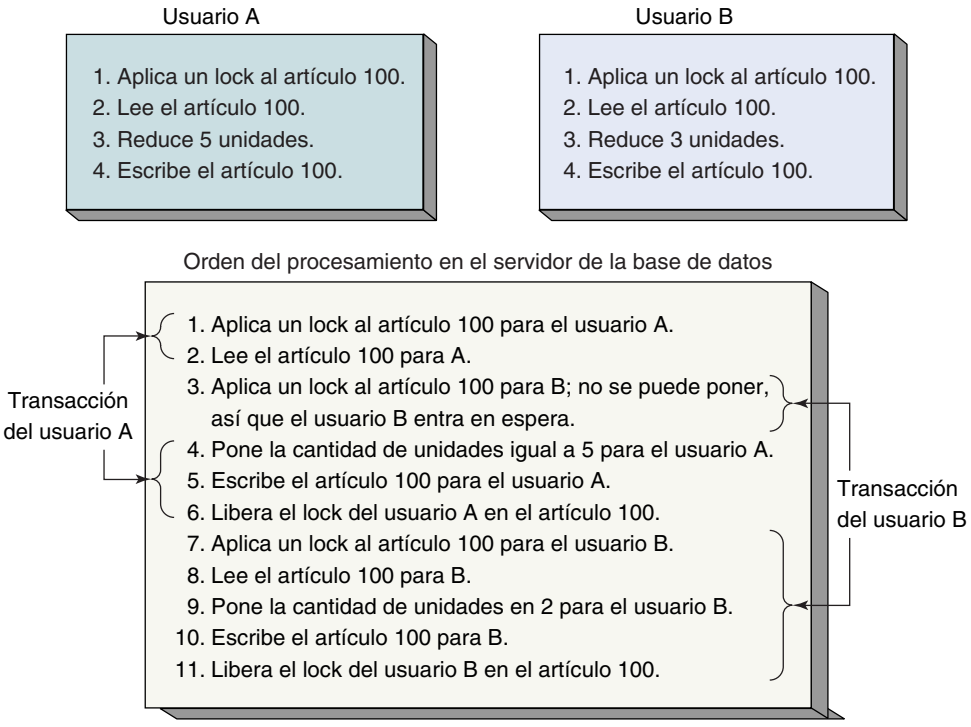
Una solución para las inconsistencias causadas por procesamiento concurrente es evitar que aplicaciones múltiples obtengan copias de un mismo registro cuando va a tener cambios. A esto se le denomina **lock (Bloqueo) de recursos** (resource locking).

Lock de recursos

Una manera de evitar problemas de procesamiento concurrente es anular cualquier posibilidad de compartir información mediante el lock de los datos que se recuperan para la actualización. La figura 11-6 muestra el orden del procesamiento usando un comando de **lock (Bloqueo).** Debido al bloqueo, la transacción del usuario B debe esperar hasta que el A haya terminado con los datos del artículo 100. Usando esta estrategia, el B puede leer el registro del elemento 100 sólo después de que el usuario A ha terminado la modificación. En este caso, la cantidad final del artículo almacenado en la base de datos es dos. (Comenzamos con 10; A tomó cinco y B tomó tres, por lo tanto quedan dos.)

➤ FIGURA 11-6

Procesamiento concurrente con locks (Bloqueos) explícitos



TERMINOLOGÍA DE LOCKS. Los locks se pueden poner ya sea en forma automática, a través del DBMS, o mediante una instrucción enviada al DBMS desde el programa de aplicación, o a solicitud del usuario. Los locks que impone el DBMS se llaman **locks implícitos;** los que se ponen mediante una instrucción se llaman **locks explícitos.**

En el ejemplo anterior los locks se aplicaron en los renglones de datos. Sin embargo, no se aplican todos los locks en este nivel. Algunos productos DBMS aplican locks por página, otros, por tabla, y otros más, a toda la base de datos. Al tamaño de un lock se le llama **granularidad del bloqueo.** Los locks con una gran granularidad son fáciles de administrar en el DBMS pero con frecuencia ocasionan conflictos. Los locks con granularidad pequeña son difíciles de administrar (hay muchos más detalles que debe atender el DBMS para conservar los registros y verificarlos), pero los conflictos son menos comunes.

Los locks también varían en su tipo. Un **lock exclusivo** impide el acceso a cualquier tipo de artículo. Ninguna otra transacción puede leer o cambiar los datos. Un **lock compartido** impide que se hagan cambios al elemento de datos, pero permite la lectura. Es decir, otras transacciones pueden leer de ese dato siempre y cuando no intenten modificarlo.

TRANSACCIONES SERIALIZABLES. Cuando dos o más transacciones se procesan al mismo tiempo, los resultados en la base de datos deben ser lógicamente consistentes con los resultados de las transacciones que hayan sido procesadas de manera serial arbitraria. A un esquema para el procesamiento concurrente de transacciones se le llama **serializable.**

La seriabilidad se puede alcanzar por diferentes medios. Una manera es procesar la transacción usando un **lock de dos fases.** Con esta estrategia, a las transacciones se les permite tener locks cuando sea necesario, pero una vez que se libera el primer lock, no se puede obtener otro. Las transacciones tienen una **fase de crecimiento,** en la que los locks se pueden obtener, y una **fase de liberación** en la que se liberan los locks.

Un caso especial de lock de dos fases se usa en varios productos DBMS. Con éste, los locks se obtienen durante la transacción, pero no se libera ninguno hasta que se emite la instrucción COMMIT o ROLLBACK. Esta estrategia es más restrictiva que los requerimientos del lock de dos fases, pero es más fácil de implementar.

En general, las fronteras de una transacción deberían corresponder a la definición de la vista de la base de datos que está en proceso. Siguiendo con la estrategia de dos fases, los renglones de cada relación en la vista se bloquean cuando es necesario. Se hacen los cambios, pero no se confirman los datos en la base hasta que se ha procesado toda la vista. En este punto, se hacen los cambios en la base de datos real y se liberan todos los locks.

Considere una transacción de registro de pedidos que implique un objeto CLIENTE-PEDIDO, el cual se construye a partir de los datos de las tablas CLIENTE, VENDEDOR y PEDIDO. Esto asegura que la base de datos no sufrirá ninguna anomalía debido a la concurrencia, la transacción de registro de pedidos aplica locks en CLIENTE, VENDEDOR y PEDIDO cuando es necesario, hace todos los cambios de la base de datos y luego libera todos sus locks.

DEADLOCK (BLOQUEO MORTAL). Aunque los locks resuelven un problema, introducen otro. Considere lo que puede pasar cuando dos usuarios quieren ordenar dos artículos del inventario. Suponga que el usuario A quiere ordenar papel y si puede obtenerlo quiere ordenar algunos lápices. Entonces suponga que el usuario B quiere ordenar algunos lápices, y si puede obtenerlos entonces quiere algo de papel. El orden del procesamiento podría ser el que se muestra en la figura 11-7.

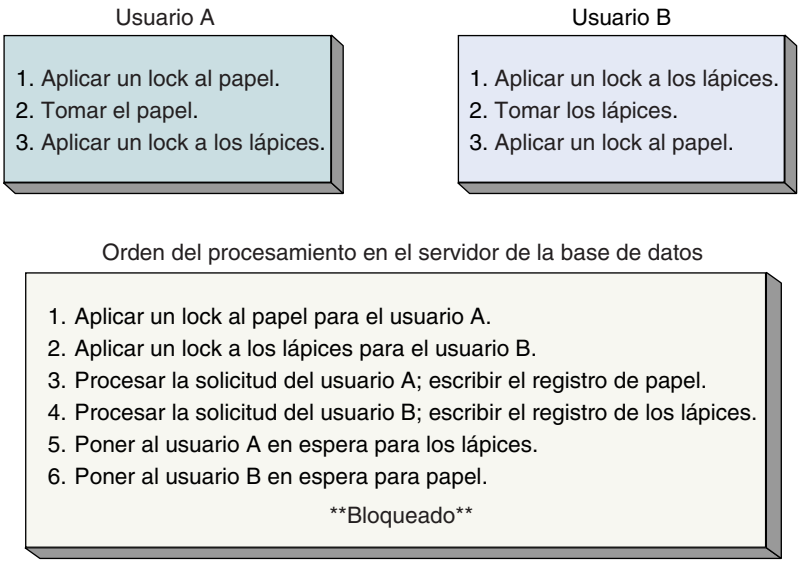
En esta figura, los usuarios A y B están atrapados en una condición llamada **deadlock** o **abrazo mortal.** Cada uno está esperando un recurso que el otro bloqueó. Hay dos maneras comunes de resolver este problema: evitar que ocurra el deadlock, o permitirlo y luego romperlo.

El deadlock se puede evitar de varias maneras. Una consiste en permitir que los usuarios emitan sólo una solicitud de lock a la vez. En esencia, los usuarios deben bloquear al instante todos los recursos que requieren. Si el usuario A en la ilustración bloquea al principio los registros de papel y lápices el deadlock nunca tendrá lugar. Una segunda manera de evitar el deadlock es requerir que todos los programas de aplicación bloqueen los recursos en el mismo orden. Incluso si no se bloquean todas las aplicaciones en ese orden, el deadlock se reducirá a los que lo hacen. Esta filosofía podría extenderse a un estándar de programación organizacional tal como: "Siempre que se procesen renglones de las tablas en una relación padre-hijo, bloquear al padre antes que a los renglones hijo". Esto cuando menos reducirá la probabilidad del deadlock y podría eliminarlo por completo.

Casi todos los DBMS tienen algoritmos para detectar el deadlock. Cuando éste ocurre la solución más común es revertir una de las transacciones para eliminar dead-

➤ FIGURA 11-7

Deadlock



lock sus cambios en la base de datos. En los dos capítulos siguientes usted verá variantes con Oracle y con el SQL Server.

Bloqueo optimista contra bloqueo pesimista

Los locks se pueden invocar en dos estilos básicos. Con el **lock optimista** se supone que no ocurrirá ningún conflicto. Los datos se leen, se procesan las transacciones y las actualizaciones y después se realiza una verificación para ver si ocurre algún conflicto; en caso contrario se termina la transacción. Si hay un conflicto ésta se repite hasta que se procese sin ningún conflicto. El **bloqueo pesimista** supone que ocurrirá el conflicto. Primero se aplican los locks, después se procesa la transacción y luego se liberan los locks.

La figura 11-8 muestra un ejemplo de cada estilo para una transacción que está restando cinco a la cantidad del renglón de lápiz en PRODUCTO. La figura 11-8(a) muestra un lock optimista. Primero se leen los datos y se almacena el valor real de Cantidad de lápices en la variable ViejaCantidad. Después la transacción se procesa y, suponiendo que todo está bien, se aplica un lock en PRODUCTO. El lock podría ser sólo para el renglón lápiz, o podría tener un nivel más grande de granularidad. En cualquiera de los casos se emite una instrucción SQL para actualizar el renglón lápiz con una condición WHERE con lo cual el valor actual de Cantidad sea igual a ViejaCantidad. Si otra transacción no ha cambiado la Cantidad del renglón lápiz, entonces este UPDATE (ACTUALIZACIÓN) tendrá éxito. Si otra transacción ha cambiado la Cantidad del renglón lápiz, El UPDATE fallará y será necesario repetir la transacción.

La figura 11-8(b) muestra la lógica para la misma transacción usando el lock pesimista. Aquí, se aplica un lock en PRODUCTO (con algún nivel de granularidad) antes de iniciar cualquier trabajo. Los valores se leen, la transacción se procesa, ocurre el UPDATE y PRODUCTO se desbloquea.

La ventaja del lock optimista es que se obtiene sólo después de que la transacción ha sido procesada. Así, el lock se mantiene durante menos tiempo que con el lock pesimista. Si la transacción es complicada, o si el cliente es lento (debido a los retrasos en la transmisión, o a que esté realizando otro trabajo, consiguiendo una taza de café, o que apague sin salirse del explorador, o browser), el lock se mantendrá durante un tiempo considerablemente menor. Esta ventaja será aún más importante si la granularidad del lock es grande; digamos, la tabla PRODUCTO.

La desventaja del lock optimista es que si existe mucha actividad en el renglón lápiz, la transacción se puede repetir muchas veces. Así, las transacciones que implican mucha actividad en un renglón determinado son muy poco adecuadas para el lock optimista.

En general, Internet es un sitio salvaje y denso, y a los usuarios les gusta tomar decisiones inesperadas, como por ejemplo abandonar las transacciones a la mitad. Así que a menos que los usuarios de Internet hayan sido precalificados (por ejemplo, al inscribirse en línea en un plan de corretaje de compra), el lock optimista es una mejor opción. Sin embargo, en intranet, la decisión es más difícil. Probablemente el lock optimista es el más indicado a menos que algunas características de la aplicación causen actividad sustancial en algunos renglones en particular, o si los requisitos de la aplicación hacen el reprocesamiento de transacciones particularmente indeseable.

Declarar las características del lock

Como usted puede ver, el control de concurrencias es un tema complicado; algunas decisiones acerca de los tipos y estrategias de los locks se han tenido que tomar a base de prueba y error. Debido a ésta y a otras razones, los programas de aplicación de base de datos, por lo general, no emiten locks explícitos, sino que marcan las fronteras de la transacción y después establecen el tipo de comportamiento de bloqueo que desean use el

➤ FIGURA 11-8

Lock optimista contra pesimista: (a) lock optimista; (b) lock pesimista

```
SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

ViejaCantidad = PRODUCTO.Cantidad

Set NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar transacción – llevar a cabo acción de excepción si NuevaCantidad < 0, etc.

Suponer que todo está BIEN:}

LOCK        PRODUCTO {con algún nivel de granularidad}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'
      AND   PRODUCTO.Cantidad = ViejaCantidad

UNLOCK      PRODUCTO

{comprobar para ver si la actualización tuvo éxito; de lo contrario, repetir la transacción}
```

(a)

```
LOCK        PRODUCTO {con algún nivel de granularidad}

SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

Set NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar transacción – llevar a cabo acción de excepción si NuevaCantidad < 0, etc.

Suponer que todo está BIEN:}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'

UNLOCK      PRODUCTO

{no se necesita comprobar si la actualización tuvo éxito}
```

(b)

DBMS. De esta manera, si se necesita cambiar el comportamiento del bloqueo, no se necesita rescribir la aplicación para colocar locks en diferentes lugares en la transacción. En lugar de eso, se cambia la declaración del lock.

La figura 11-9 muestra la transacción de lápiz con las fronteras de la transacción marcadas con INICIAR TRANSACCIÓN, COMMIT TRANSACCIÓN, ROLLBACK TRANSACCIÓN (BEGIN TRANSACTION, COMMIT TRANSACTION y ROLLBACK TRANSACTION). Estas fronteras son la información esencial que el DBMS necesita para aplicar las diferentes estrategias de bloqueo. Si el programador ahora declara (a través de un parámetro del sistema, o por medios similares) que quiere que el lock sea optimista, el

➤ FIGURA 11-9

Establecimiento de las fronteras de transacción

```
INICIAR TRANSACCIÓN:

SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

ViejaCantidad = PRODUCTO.Cantidad

SET NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar parte de la transacción – llevar a cabo la acción de excepción si NuevaCantidad < 0, etc.}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'

{continuar procesando transacción}...

Si la transacción ha terminado normalmente      THEN (ENTONCES)

      COMMIT TRANSACCIÓN

ELSE (DE OTRO MODO)

      ROLLBACK TRANSACCIÓN

END IF (TERMINA SI)

Continuar procesando otras acciones que no son parte de esta transacción...
```

DBMS establecerá implícitamente los locks en los lugares correctos para ese tipo de bloqueo. Si el programador más tarde cambia las tácticas y solicita el bloqueo pesimista, el DBMS configurará implícitamente los locks en un lugar diferente.

Transacciones consistentes

Algunas veces usted verá el acrónimo ACID aplicado a las transacciones. Una **transacción ACID** (**a**tomic, **c**onsistent, **i**nsolated and **d**urable, por sus siglas en inglés) es aquella que es atómica, consistente, aislada y durable. Atómica y durable son fáciles de definir. Como acaba de leer, una transacción atómica es aquella en la que todas las acciones de la base de datos pueden ocurrir, o también ninguna. Una transacción durable es aquella para la que todos los cambios confirmados son permanentes. El DBMS no eliminará esos cambios, incluso en el caso de fracasar. Si la transacción es durable, el DBMS proporcionará las facilidades para recuperar los cambios de todas las acciones confirmadas cuando sea necesario.

Sin embargo, los términos *consistente* y *aislada* no son tan definitivos como *atómica* y *durable*. Considere la siguiente instrucción de actualización de SQL:

```
UPDATE    CLIENTE
SET       CódigodeÁrea = '425'
WHERE     CódigoPostal = '98050'
```

Suponga que hay 500, 000 renglones en la tabla CLIENTE y que 500 tienen CódigoPostal igual a '98050'. Le tomará algún tiempo al DBMS encontrar los 500 renglones. Durante ese tiempo, ¿otras transacciones permitirán actualizar los campos de CódigodeÁrea o CódigoPostal de CLIENTE? Si la instrucción de SQL es consistente, estas actualizaciones estarán prohibidas. La actualización se aplicará para establecer que los

renglones como éstos existen en el momento en que el enunciado de SQL inició. Esta consistencia se llama **consistencia del nivel de instrucción.**

Ahora considere una transacción que contenga dos instrucciones de actualización de SQL:

```
BEGIN TRANSACTION
UPDATE    CLIENTE
SET       CódigodeÁrea = '425'
WHERE     CódigoPostal = '98050'
{otra transacción en funcionamiento}
UPDATE    CLIENTE
SET       Descuento = 0.05
WHERE     CódigodeÁrea = '425'
{otra transacción en funcionamiento}
COMMIT    TRANSACTION
```

En este contexto, ¿qué significa consistente? La consistencia del nivel de instrucción quiere decir que cada instrucción procesa independientemente renglones consistentes, pero que los cambios de otros usuarios de estos renglones se pueden permitir durante el intervalo entre las dos instrucciones SQL. El **nivel de consistencia de la transacción** significa que todos los renglones impactados por cualquiera de las instrucciones SQL son protegidos de cambios durante la transacción completa. Observe, sin embargo, que para algunas implementaciones de la consistencia del nivel de transacción, una transacción no verá sus propios cambios. En este ejemplo, la segunda instrucción SQL puede no ver los cambios en los renglones derivados de la primera instrucción SQL.

Así, cuando usted escuche el término *consistente*, ponga más atención para determinar a qué tipo de consistencia se refiere. También tenga cuidado con la trampa potencial de consistencia del nivel de transacción.

La situación es más complicada para el término *aislada*, el cual consideraremos a continuación.

Nivel de aislamiento de la transacción

Los locks evitan que los procesos concurrentes ocasionen la pérdida de actualizaciones; pero hay otro tipo de problemas que no evitan. Específicamente, ocurre una **lectura sucia** cuando una transacción lee un registro cambiado que no ha sido registrado en la base de datos. Por ejemplo, esto puede ocurrir si una transacción lee un renglón cambiado por una segunda transacción no confirmada, la cual más tarde aborta.

Las **lecturas no repetibles** ocurren cuando una transacción relee datos que fueron leídos con anterioridad y encuentra modificaciones o eliminaciones ocasionadas por una transacción confirmada. Por último, las **lecturas fantasma** tienen lugar cuando una transacción relee los datos y encuentra que se insertaron nuevos renglones como resultado de una transacción confirmada en la lectura anterior.

El estándar ANSI SQL de 1992 define cuatro **niveles de aislamiento,** que especifican cuál de estos problemas se permite que ocurra. El objetivo es que el programador de la aplicación pueda declarar el tipo de nivel de aislamiento que quiere y entonces tener la administración de los locks del DBMS, así como lograr ese nivel de aislamiento.

Como se muestra en la figura 11-10, la lectura en un nivel de aislamiento no confirmado permite que ocurran lecturas sucias, lecturas no repetibles y lecturas fantasmas. Con aislamiento de lecturas confirmadas, las lecturas sucias están prohibidas. El nivel de aislamiento de lecturas repetibles prohíbe tanto las lecturas sucias como las no repetibles. El nivel de aislamiento serializable no permitirá que ocurra ninguna de las tres.

Por lo general, el nivel más restrictivo, de menor rendimiento efectivo, depende mucho de la carga de trabajo y de cómo estén escritos los programas de aplicación. Además, no todos los productos DBMS usan todos estos niveles. Los productos también varían en la manera en la cual se manejan y en cuanto a la carga que ponen en el

➤ FIGURA 11-10

Resumen de los niveles de aislamiento

		Nivel de aislamiento			
		Lectura no confirmada	**Lectura confirmada**	**Lectura repetible**	**Serializable**
Tipo de problema	**Lectura sucia**	Posible	No posible	No posible	No posible
	Lectura no repetible	Posible	Posible	No posible	No posible
	Lectura fantasma	Posible	Posible	Posible	No posible

programador de la aplicación. En los dos siguientes capítulos aprenderá cómo Oracle y el SQL Server manejan los niveles de aislamiento.

Tipo de cursor

Un cursor es un apuntador a un conjunto de renglones. Los cursores generalmente se definen usando las instrucciones SELECT. Por ejemplo, la instrucción:

```
DECLARE CURSOR TransCursor AS
SELECT    *
FROM      TRANSACCIÓN
WHERE     PreciodeCompra > '10000'
```

define un cursor llamado TransCursor que opera sobre el conjunto de renglones que indica la instrucción SELECT. Cuando un programa de aplicación abre un cursor y lee el primer renglón, se dice que está "apuntando al primer renglón".

Una transacción abre varios cursores —ya sea en secuencia o al mismo tiempo—. Además, se pueden abrir dos o más cursores en la misma tabla; ya sea directamente en ésta o a través de la vista SQL en la tabla. Debido a que los cursores requieren mucha memoria, el hecho de tener muchos abiertos al mismo tiempo —por ejemplo, para mil transacciones concurrentes—, puede consumir mucha memoria y tiempo del CPU. Una manera de reducir los gastos del cursor es definir cursores de capacidad reducida y utilizarlos cuando no se necesite uno de capacidad completa.

La figura 11-11 lista cuatro tipos de cursores usados en el ambiente de Windows 2000 (los tipos de cursores para otros sistemas son similares). El cursor más simple es sólo hacia delante (forward only). Con éste, la aplicación sólo se puede mover hacia adelante a través del conjunto de registros. Los cambios hechos por otros cursores en esta transacción y por otras transacciones serán visibles sólo si están en los renglones arriba del cursor.

Los tres tipos siguientes de cursor se llaman **cursores deslizables** puesto que la aplicación puede deslizarse hacia adelante y hacia atrás del conjunto de registros (recordset). Un cursor estático procesa una "fotografía" de la relación que se tomó cuando se abrió el cursor. Los cambios que se hacen usando este cursor son visibles para éste; pero los cambios de cualquier otra fuente no son visibles.

Los cursores keyset combinan algunas características de los cursores estáticos con otras de los dinámicos. Cuando se abre el cursor se guarda un valor de la llave primaria para cada renglón en el conjunto de registros (recordset). Cuando la aplicación posiciona el cursor sobre un renglón, el DBMS usa el valor de la llave para leer el valor actual del renglón. Si la aplicación emite una actualización sobre un renglón que ha sido eliminado por un cursor diferente en esta transacción, o por una transacción diferente, el DBMS crea un nuevo renglón con el valor de la llave anterior y pone los valores de la actualización en el renglón nuevo (suponga que están presentes todos los campos requeridos). Las inserciones de nuevos renglones que hacen otros cursores en esta transac-

➤ FIGURA 11-11

Resumen de tipos de cursor

Tipo de cursor	Descripción	Comentarios
Sólo hacia delante (forward only)	La aplicación sólo puede moverse hacia adelante a través del conjunto de registros.	Los cambios hechos por otros cursores en esta transacción o en otra estarán visibles sólo si se localizan en líneas adelante del cursor.
Estático	La aplicación ve los datos como si éste fuera el momento en que se abrió el cursor.	Los cambios hechos por este cursor son visibles. Las cambios de otras fuentes no lo son. Puede deslizarse hacia adelante y hacia atrás.
Keyset	Cuando se abre el cursor se salva el valor de la llave primaria para cada renglón en el conjunto de registros. Cuando la aplicación accede a un renglón, la llave se usa para buscar los valores actuales para el renglón.	Las actualizaciones de cualquier fuente son visibles. Las inserciones de fuentes fuera de este cursor no son visibles (no hay una llave para estas en el keyset). Las inserciones de este cursor aparecen al final del conjunto de registros (recordset). Las eliminaciones de cualquier fuente están visibles. Los cambios en el orden de los renglones no son visibles. Si el nivel de aislamiento es de lectura sucia, entonces las actualizaciones confirmadas y las eliminaciones están visibles; de otra manera, sólo las actualizaciones confirmadas y las eliminaciones son visibles.
Dinámico	Los cambios de cualquier tipo y fuente están visibles.	Todas las inserciones, actualizaciones, eliminaciones y cambios en el orden del conjunto de registros (recordset) están visibles. Si el nivel de aislamiento es de lectura sucia, entonces los cambios no confirmados están visibles; de otra manera, sólo se ven los confirmados.

ción, u otras transacciones, no están visibles para un cursor de keyset. A menos que el nivel de aislamiento de la transacción sea una lectura sucia, sólo las actualizaciones confirmadas y las eliminaciones serán visibles para el cursor.

Un cursor dinámico es un cursor plenamente caracterizado. Todas las inserciones, actualizaciones, eliminaciones, cambios en un renglón están visibles para un cursor dinámico. Al igual que con los cursores de keyset, a menos que el nivel de aislamiento de la transacción sea una lectura sucia, sólo los cambios confirmados estarán visibles.

La cantidad de gastos y el procesamiento que se requieren para sostener un cursor es diferente para cada tipo. En general, el costo aumenta conforme nos movemos hacia abajo de los tipos de cursor de la figura 11-11. Por lo tanto, con el fin de mejorar el rendimiento del DBMS, el programador de la aplicación sólo debería crear cursores que sean lo suficientemente poderosos como para hacer el trabajo. También es muy importante entender cómo un DBMS en particular implementa cursores, y si éstos están localizados en el servidor o en el cliente. En algunos casos, sería mejor poner un cursor dinámico a disposición del cliente, que tener un cursor estático en el servidor. No se puede establecer ninguna regla general debido a que el desempeño depende de la implementación que usa el producto DBMS, así como de los requerimientos de la aplicación.

Advertencia: si no se especifica el nivel de aislamiento de una transacción, o no se especifica el tipo de cursores que abre, el DBMS usará el nivel y los tipos predeterminados. Estas predeterminaciones pueden ser perfectas para su aplicación, pero también

terribles. Así, aunque se pueden ignorar estos asuntos, no se evitarán sus consecuencias. Aprenda las capacidades de su producto DBMS y úselas con sabiduría.

➤ SEGURIDAD DE LA BASE DE DATOS

El objetivo de la seguridad de la base de datos es asegurar que sólo los usuarios autorizados puedan desempeñar actividades permitidas en momentos también establecidos. Este objetivo es difícil de alcanzar y para lograrlo el equipo programador de la base de datos debe, durante la fase de especificación de requerimientos del proyecto, determinar los derechos y responsabilidades de procesamiento de todos los usuarios. Estos requerimientos de seguridad se pueden imponer usando las características de seguridad del DBMS y las adiciones a las características que se describen en los programas de aplicación.

DERECHOS Y RESPONSABILIDADES DEL PROCESAMIENTO

Considere, por ejemplo, las necesidades de la galería View Ridge que analizamos en el capítulo 10. Existen tres tipos de usuarios: el personal de ventas, el administrativo y los administradores del sistema. Al personal de ventas se le permite ingresar un nuevo cliente y los datos de la transacción, cambiar la información del cliente y solicitar cualquier información. No se les permite meter nuevos artistas o la información del trabajo. Tampoco pueden eliminar cualquier dato.

El personal administrativo tiene acceso a todos los permisos del personal de ventas, además pueden introducir nuevos artistas y datos de las obras de arte, así como modificar los datos de las transacciones. Aunque el personal administrativo está autorizado para eliminar datos, no tienen ese permiso en esta aplicación. Esta restricción tiene como fin evitar la posibilidad de perder información accidentalmente.

Al administrador del sistema se le permite el acceso irrestricto a los datos. Puede crear, actualizar, leer y eliminar cualquier información de la base de datos. También puede otorgar derechos de procesamiento a otros usuarios y cambiar la estructura de los elementos de la base de datos tales como tablas, índices, procedimientos almacenados y cosas por el estilo. La figura 11-12 resume estos requerimientos.

➤ FIGURA 11-12

Derechos del procesamiento en la galería View Ridge

	Cliente	**Transacción**	**Trabajo**	**Artista**
Personal de ventas	Insertar, Cambiar, Consultar	Insertar, Consultar	Consultar	Consultar
Personal administrativo	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar
Administrador del sistema	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura

Los permisos en esta tabla están por tipos de usuarios, o **grupos de usuarios,** y no por personas. Esto es normal, pero no es obligatorio. Sería posible decir, por ejemplo, que el usuario identificado como "Benjamin Franklin" tiene ciertos derechos de procesamiento. También observe que cuando se usan los grupos es necesario tener un medio para asignar usuarios a los grupos. Cuando "Mary Smith" se registra en la computadora, se necesitan algunos medios para determinar a qué grupo o grupos pertenece. Analizaremos esto con mayor detalle en la siguiente sección.

En este análisis hemos usado la frase *derechos y responsabilidades de procesamiento*; como lo indica dicha frase, las responsabilidades van unidas a los derechos de procesamiento. Si, por ejemplo, el administrador del sistema elimina los datos de la transacción, tiene la responsabilidad de asegurarse que esto no afecte el funcionamiento de la galería, la contabilidad, etcétera.

Las responsabilidades de procesamiento no se pueden imponer a través del DBMS o de las aplicaciones de la base de datos. Éstas son codificadas en procedimientos manuales y se les explican a los usuarios durante la capacitación para operar el sistema. Éstos son temas de un texto de desarrollo de sistemas, por lo tanto no los abordaremos más aquí, sólo reiteramos que las *responsabilidades* van unidas a los *derechos*. Estas responsabilidades deben ser documentadas e impuestas.

De acuerdo con la figura 11-1, el DBA tiene la tarea de administrar los derechos y responsabilidades del procesamiento. Como es obvio, esos derechos y responsabilidades cambiarán con el tiempo. Conforme se use la base de datos, y se realicen cambios a las aplicaciones y la estructura del DBMS, surgirá la necesidad de derechos y responsabilidades nuevos o diferentes. El DBA es un punto focal para el análisis de estos cambios y su implementación.

Una vez que los derechos del procesamiento han sido definidos, se pueden implementar en muchos ámbitos: sistema operativo, servidor Web, el DBMS y la aplicación. En las dos secciones siguientes consideraremos al DBMS y la aplicación. Los otros están más allá del ámbito de este texto.

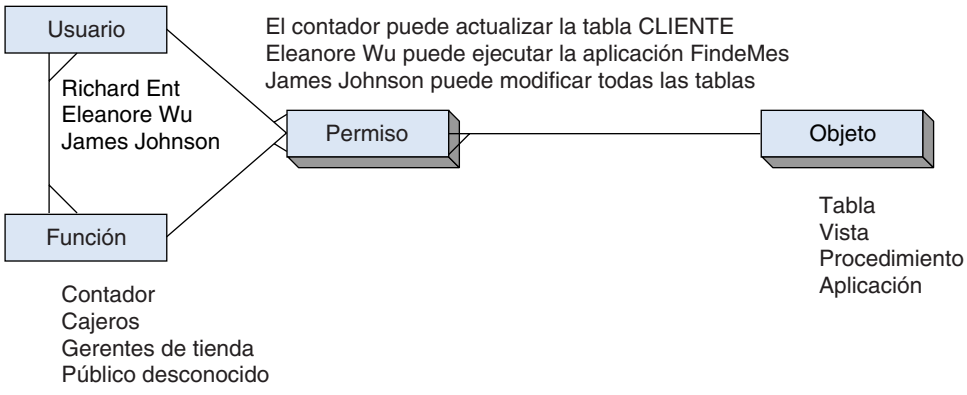
Seguridad DBMS

La terminología, las características y funciones de la seguridad del DBMS dependen del producto DBMS que se use. Básicamente, todos proporcionan dispositivos que limitan ciertas acciones u objetos a determinados usuarios. En la figura 11-13 se muestra un modelo general de seguridad DBMS. Se puede asignar a un usuario una o más funciones y una de éstas puede tener uno o más usuarios. Tanto los usuarios como las funciones tienen muchos permisos. Los objetos (usados en un sentido genérico) tienen muchos permisos asignados. Cada permiso pertenece a un usuario o función y a un objeto.

Cuando un usuario se registra en la base de datos, el DBMS limita sus acciones a los permisos que le fueron conferidos, y a los permisos para las funciones que se le asignaron. Determinar si alguien realmente es quien dice ser, en general, es una tarea difícil. Todos los productos comerciales usan alguna versión del nombre del usuario y la

➤ FIGURA 11-13

Un modelo de seguridad DBMS



contraseña, aun cuando esta seguridad se anula con facilidad si los usuarios no tienen cuidado con sus contraseñas.

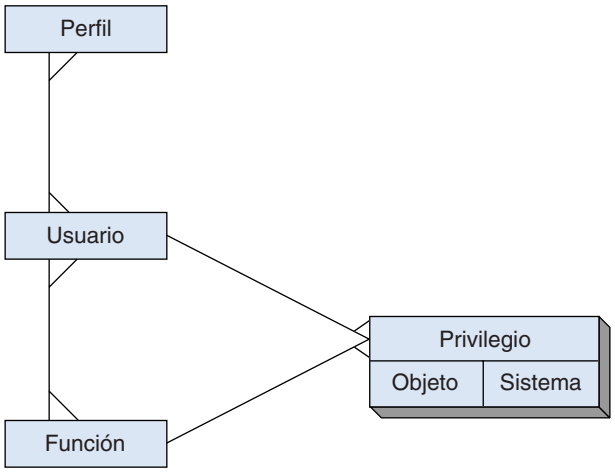
Los usuarios pueden introducir su nombre y contraseña, o, en algunas aplicaciones, el nombre y la contraseña se introducen en el nombre del usuario. Por ejemplo, el nombre y contraseña del usuario de Windows 2000 se puede pasar directamente a SQL Server. En otros casos, un programa de aplicación proporciona el nombre y la contraseña.

Las aplicaciones de Internet por lo general definen a un grupo como "Público desconocido" y le asignan usuarios anónimos cuando se inscriben. De esta manera, compañías como Dell Computer no necesitan introducir a cada usuario en su sistema de seguridad por nombre y contraseña.

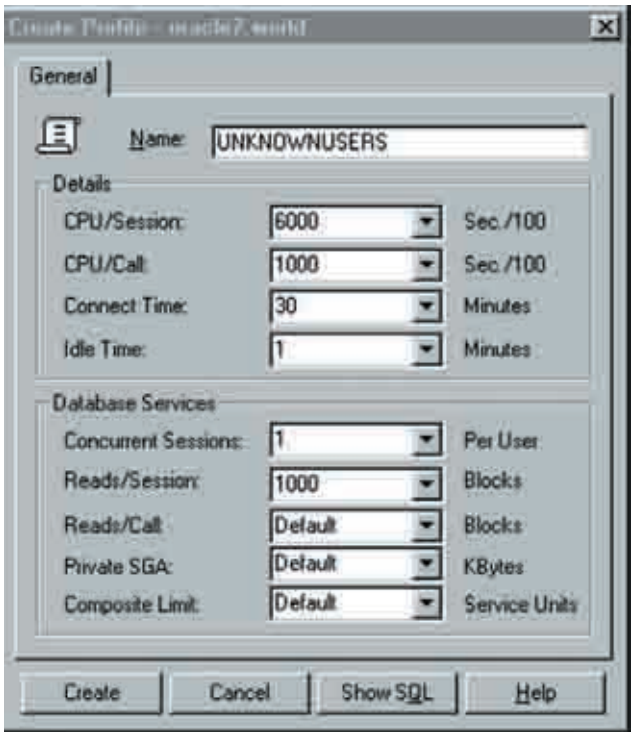
Los modelos de los sistemas de seguridad que usa Oracle y SQL Server se ilustran en las figuras 11-14 y 11-15. Como puede ver, ambos son variantes del modelo general

➤ FIGURA 11-14

Seguridad en Oracle: (a) modelo de seguridad de Oracle, (b) definición del perfil de Oracle, (c) usuario, funciones y perfiles de Oracle



(a) Modelo de seguridad de Oracle

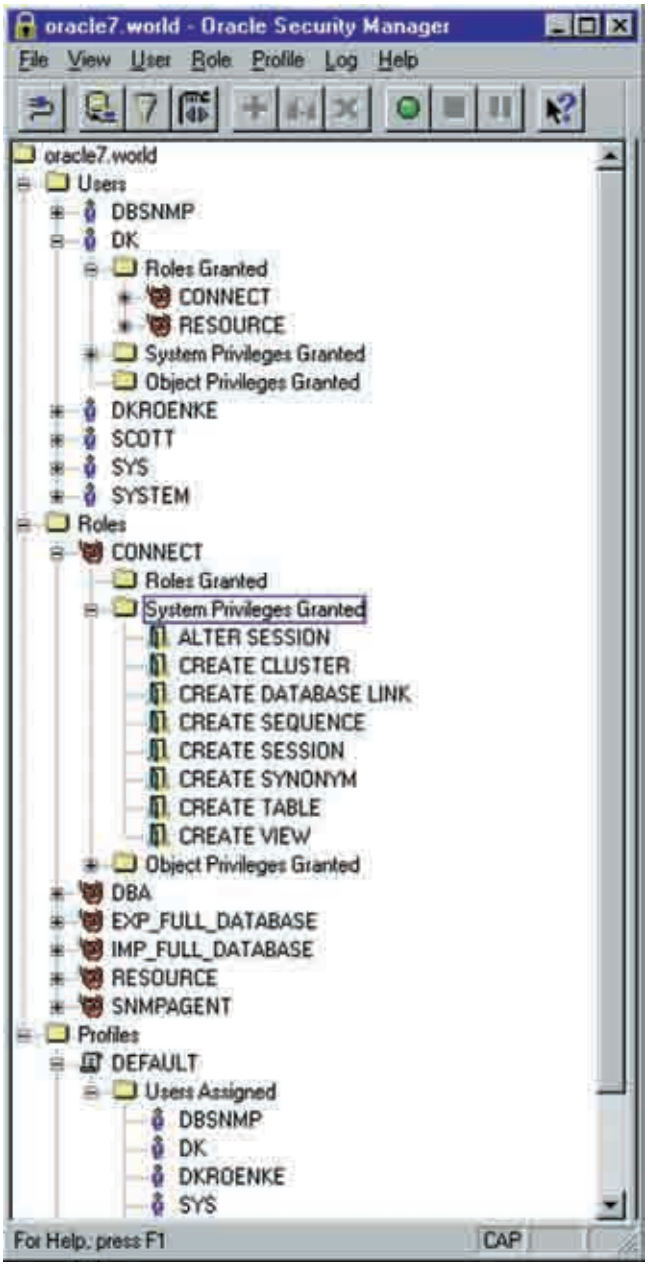


(b)

(continúa)

➤ FIGURA 11-14

(Continuación)



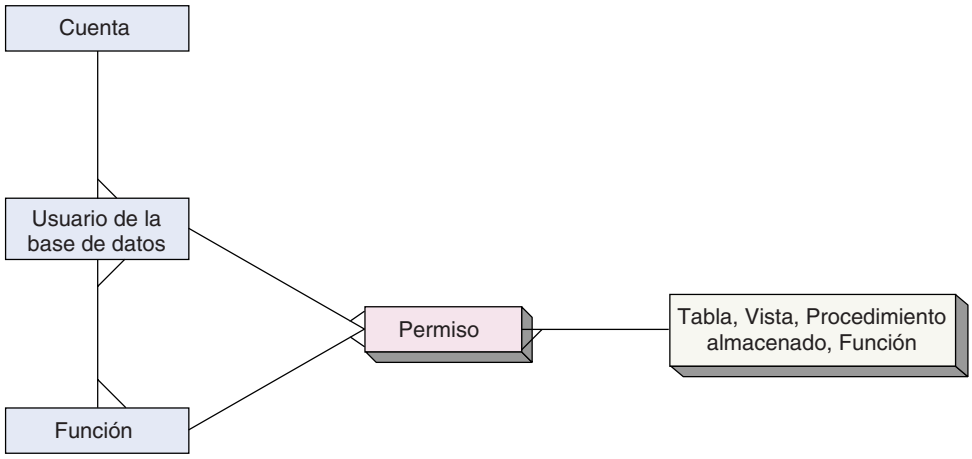
(c)

de la figura 11-13. Considerando el modelo de Oracle en la figura 11-14(a), los usuarios tienen un perfil, que es una especificación de los recursos de los sistemas que se pueden usar. En la figura 11-14(b) se muestra un ejemplo de la definición de perfil. A los usuarios se les pueden asignar muchas funciones y una función se asigna a muchos usuarios. Cada usuario y función tienen muchos privilegios, de los cuales hay dos subtipos: los del objeto que se refieren a las acciones que se pueden llevar a cabo sobre los objetos de la base de datos como tablas, vistas, e índices; y los privilegios de los sistemas que se refieren a las acciones usando los comandos de Oracle.

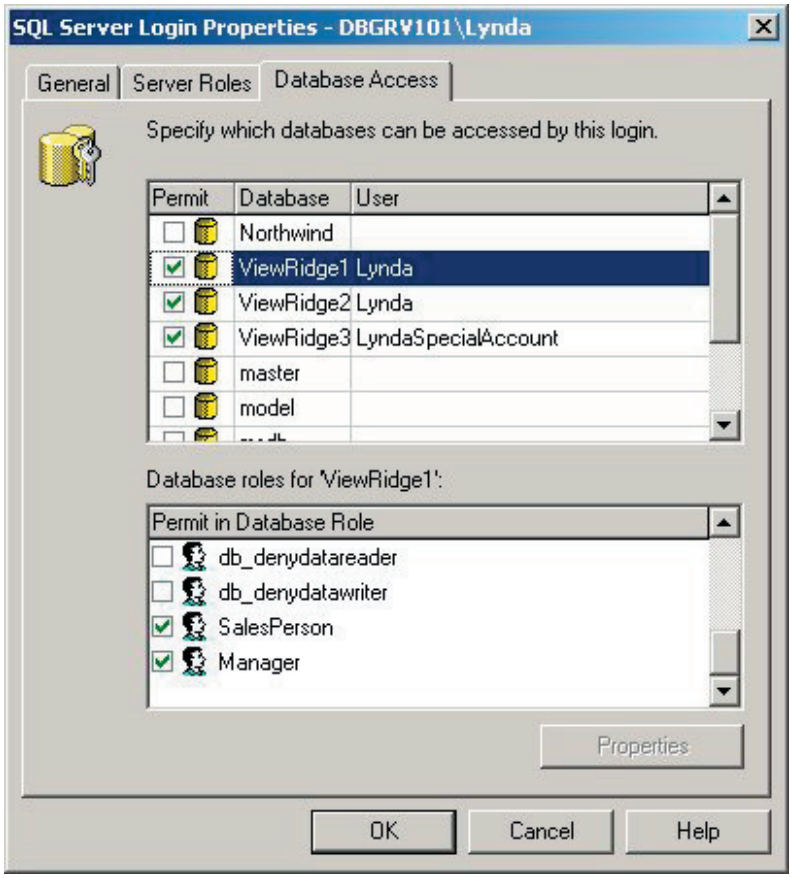
En la figura 11-14(c) se ejemplifica este sistema de seguridad. Observe que al usuario DK se le han asignado funciones para CONNECT y RESOURCE. Si vemos más abajo en la lista, a la función CONNECT se le han otorgado privilegios del sistema como

➤ FIGURA 11-15

Seguridad del SQL Server: (a) un modelo de seguridad del SQL Server, (b) ejemplo de entrada al SQL Server, (c) Ejemplo de la función del SQL Server y (d) ejemplo del usuario del SQL Server



(a)



(b)

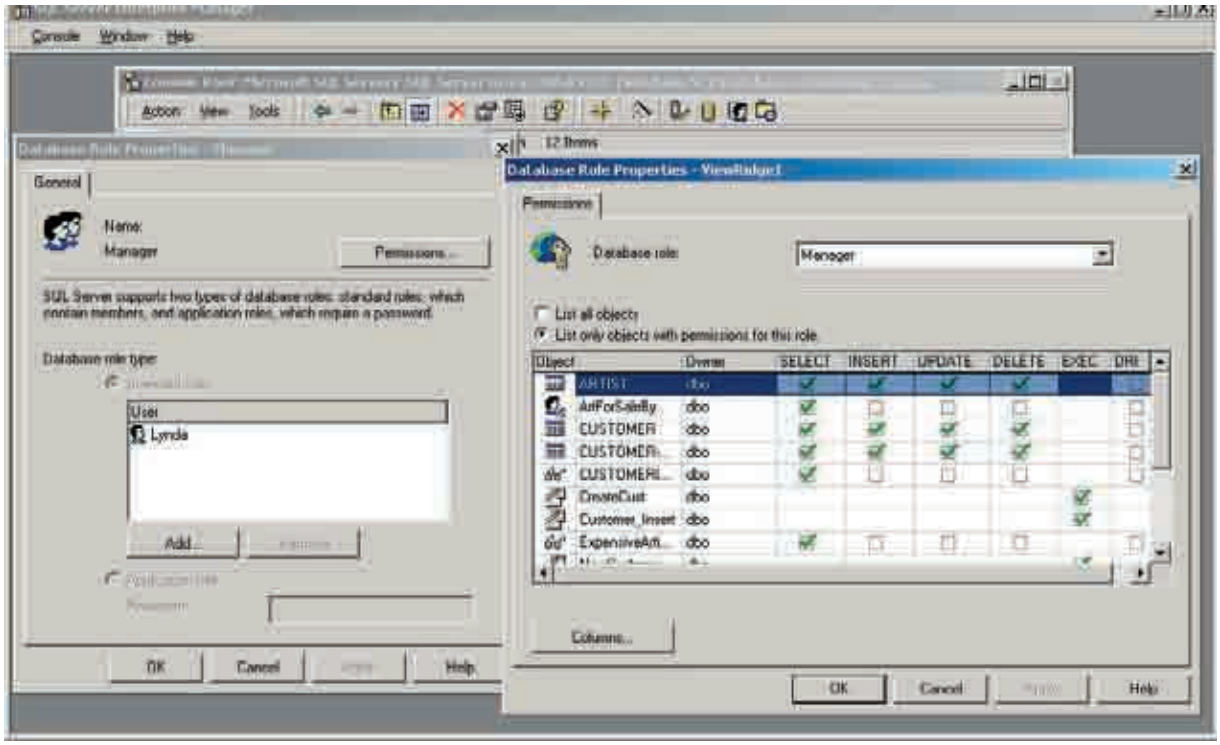
(*continúa*)

ALTER SESSION, CREATE CLUSTER y otros, incluyendo CREATE TABLE. Si vemos más abajo en la lista, el usuario DK ha sido asignado a DEFAULT.

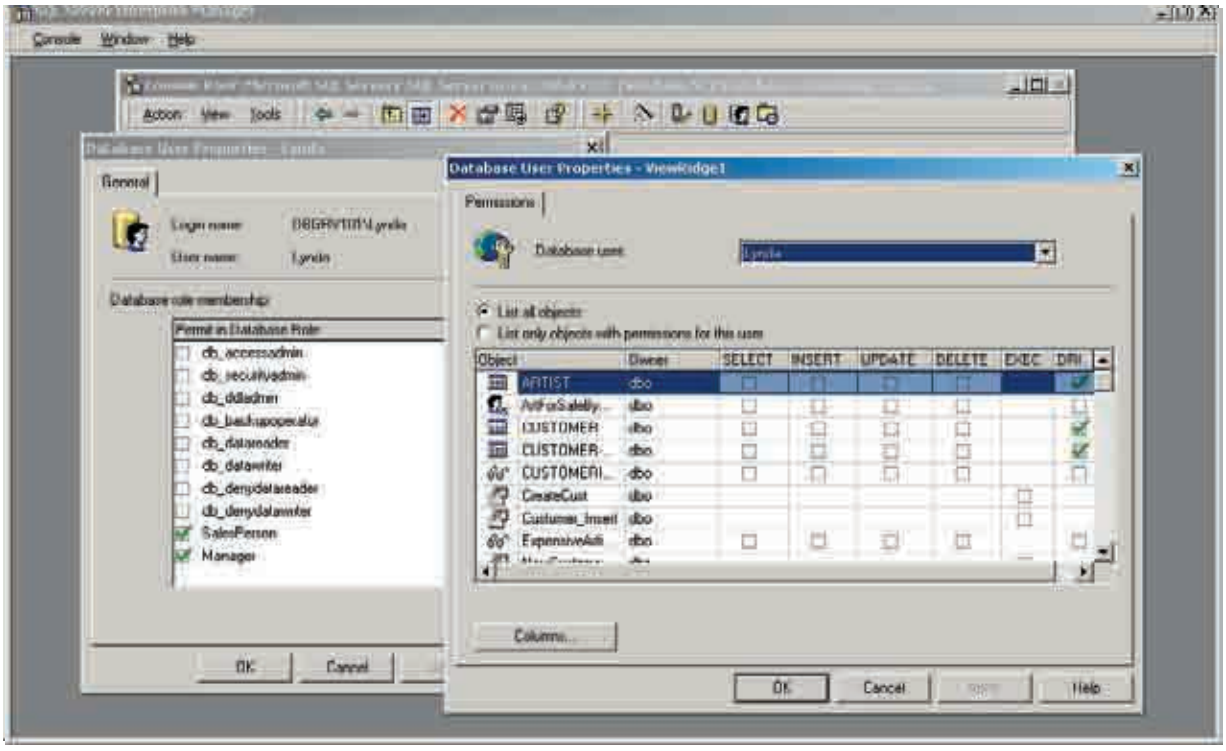
En la figura 11-15(a) se ilustra el esquema de seguridad para SQL Server. Una cuenta (el nombre y la contraseña usadas cuando se registra en SQL Server o en Windows) se puede asociar con una o más combinaciones de base de datos/usuario. En la figura 11-15(b), por ejemplo, al nombre de la cuenta de Lynda en el dominio DBGRV101 se

➤ FIGURA 11-15

(Continuación)



(c)



(d)

le ha permitido el acceso a la base de datos ViewRidge1 con el nombre de usuario Lynda, para ViewRidge2 con el nombre de usuario Lynda y a ViewRidge3 con el nombre del usuario LyndaSpecialAccount. Como se muestra en la figura 11-15(a), se puede asociar más de una entrada con una combinación determinada de base de datos/usuario.

La figura 11-15(c) muestra los permisos de una función llamada Manager (Gerente) en la base de datos ViewRidge1. Un rol puede tener uno o más usuarios, pero en este caso sólo tiene uno: Lynda. Los permisos de la función Manager se muestran con una marca verde en la caja de permisos. EXEC se refiere al permiso de ejecutar un procedimiento almacenado y DRI, al permiso para crear, cambiar o eliminar restricciones declarativas de integridad referencial, tales como CustomerID en CUSTOMER-ARTIST-INT que debe ser un subconjunto de CustomerID en CUSTOMER.EXEC; sólo es aplicable a objetos de procedimiento almacenado, y DRI sólo se aplica a tablas y vistas.

Una base de datos/usuario tiene todos los permisos de las funciones a las cuales está asignado el usuario, así como también cualquier permiso que sólo se le asigna a él. La figura 11-15(d) muestra que al usuario ViewRidge1/Lynda se le han asignado las funciones SalesPerson y Manager, y tendrá todos los permisos de éstas. Además, se le ha permitido crear, modificar y eliminar restricciones DRI en las tablas ARTIST, CUSTOMER y CUSTOMER-ARTIST-INT (el nombre de esta última tabla está incompleto en esta figura).

Seguridad de la aplicación

Si bien los productos DBMS, como Oracle y SQL Server, proporcionan capacidades de seguridad sustanciales de la base de datos, por su misma naturaleza son genéricos. Si la aplicación requiere medidas de seguridad específicas tales como: "Ningún usuario puede ver un renglón de una tabla, o un join de una tabla, que tenga el nombre de otro empleado" los dispositivos del DBMS no serán adecuados. En este caso, se debe aumentar la seguridad del sistema mediante algunas características en la aplicación de la base de datos.

Por ejemplo, como verá en el capítulo 14, la seguridad en las aplicaciones de Internet con frecuencia las proporciona la computadora del servidor Web. Ejecutar la seguridad de la aplicación en este servidor significa que la información de seguridad no necesita ser transmitida a la red.

Para entenderlo mejor, suponga que escribe una aplicación de tal manera que cuando los usuarios aprieten un botón en particular en una página del explorador, se envíe la siguiente solicitud al servidor Web y luego al DBMS:

```
SELECT    *
FROM      EMPLEADO
```

Esta instrucción, por supuesto, regresará a todos los renglones EMPLEADO. Si la seguridad de la aplicación limita a los empleados para que sólo tengan acceso a sus propios datos, entonces un servidor Web podría agregar la cláusula que se muestra abajo de esta consulta:

```
SELECT    *
FROM      EMPLEADO
WHERE EMPLEADO.Nombre = ´<%=SESIÓN("EmpleadoNombre")%>´
```

Como aprenderá en los capítulos 15 y 16, una expresión como ésta ocasionaría que el servidor Web llenara los nombres de los empleados en la cláusula WHERE. Para un usuario inscrito con el nombre 'Benjamin Franklin', la instrucción que resulta de esta expresión es:

```
SELECT    *
FROM      EMPLEADO
WHERE EMPLEADO.Nombre = "Benjamin Franklin"
```

Debido a que el nombre lo inserta un programa en el servidor Web, el usuario del explorador no sabrá qué está ocurriendo y no podrá interferir aunque lo intente.

Este procesamiento de seguridad se puede llevar a cabo en un servidor Web como se muestra aquí, pero también se puede hacer dentro de los mismos programas de aplicación, o se puede escribir como procedimientos almacenados (stored procedures) o disparadores (triggers) para que los ejecute el DBMS en los momentos apropiados.

Esta idea se puede extender mediante el almacenamiento adicional de datos en una base de datos de seguridad al que se tenga acceso a través del servidor Web, o mediante procedimientos almacenados y disparadores. Por ejemplo, esa base de datos de seguridad podría contener las identidades de los usuarios asociadas con los valores adicionales de las cláusulas WHERE. Por ejemplo, suponga que los usuarios en el departamento de personal pueden acceder sólo a sus propios datos. Los predicados para las cláusulas apropiadas WHERE se podrían almacenar en la base de datos de seguridad, que las lea el programa de aplicación, y se anexen a las instrucciones SELECT de SQL cuando sea necesario.

Existen muchas otras posibilidades para extender la seguridad DBMS con el procesamiento de la aplicación. Sin embargo, en general, es mejor que primero use usted las características de seguridad del DBMS. Sólo si son inadecuadas para los requerimientos, deberá agregarlas con el código de la aplicación. Mientras más cercana esté la aplicación de seguridad a la información, menos probabilidades existen de que ocurra alguna infiltración. Asimismo, usar las características de seguridad del DBMS es más rápido, barato y probablemente se obtienen resultados de mayor calidad que desarrollando las propias.

➤ RECUPERACIÓN DE LA BASE DE DATOS

Los sistemas de la computadora pueden fallar. Los programas tienen defectos; los procedimientos humanos conllevan errores, y las personas se equivocan. Todas estas fallas pueden ocurrir y de hecho ocurren en las aplicaciones de la base de datos. Puesto que una base de datos la comparten muchas personas, y que con frecuencia esto es un elemento clave de las operaciones de la organización, es importante recuperarla lo más rápido posible.

Hay varios problemas que conviene señalar. Primero, desde el punto de vista del negocio, las operaciones deben continuar. Por ejemplo, los pedidos del cliente, las transacciones financieras y las listas de empaque deben llenarse en forma manual. Más tarde, cuando se opera de nuevo la aplicación de la base de datos, se pueden introducir datos nuevos. Segundo, el personal de operaciones de la computadora debe restaurar el sistema para su uso tan rápido como sea posible para que funcione como antes del desperfecto. Tercero, los usuarios deben saber qué hacer cuando el sistema vuelva a estar disponible. Quizá sea necesario ingresar otra vez datos y los usuarios deben saber qué tanto necesitan regresar.

Cuando ocurren fallas es imposible arreglar el problema y reanudar el procesamiento. Incluso si no se perdieron datos durante la falla (lo que nos hace suponer que todos los tipos de memoria son no volátiles, lo que constituye una suposición irreal), el tiempo y el horario del procesamiento de la computadora son muy complejos para ser recreados en forma exacta. Se requieren enormes cantidades de datos y de procesamiento para que el sistema operativo pueda reiniciar el procesamiento exactamente en donde se interrumpió. Es imposible revertir el reloj y poner todos los electrones en la misma configuración donde estaban en el momento de la falla. Hay dos vías posibles: recuperar a través del reprocesamiento, y recuperar a través de revertir/avanzar (rollback/rollforward).

RECUPERACIÓN MEDIANTE REPROCESAMIENTO.

Puesto que el procesamiento no se puede reanudar en un punto preciso, la mejor alternativa es regresar a un punto conocido y reprocesar la carga de trabajo desde ahí. La forma más simple de este tipo de recuperación es hacer periódicamente una copia de la base de datos **(guar-**

dar la base de datos) y conservar un registro de todas las transacciones que se han procesado desde que se guardó. Así, cuando hay una falla, el personal puede restaurar la base de datos que se guardó y reprocesar todas las transacciones.

Por desgracia, esta estrategia simple no es factible. Primero, las transacciones del reprocesamiento tardan el mismo tiempo que si se procesan por primera vez. Si la computadora se programa con carga muy pesada, puede ser que el sistema nunca se ponga al corriente.

Segundo, cuando las transacciones se procesan concurrentemente, los sucesos son asincrónicos. Las ligeras variaciones en la actividad humana —por ejemplo un usuario que inserta un disco flexible lentamente, o uno que lee un mensaje de correo electrónico antes de responder a una aplicación inmediata— pueden cambiar el orden de la ejecución de las transacciones concurrentes. Por lo tanto, mientras que el cliente A viaja en el último asiento de un vuelo durante el procesamiento original, el cliente B puede ir en el último asiento durante el reprocesamiento. Por estas razones, normalmente no es el reprocesamiento una forma viable de recuperación a partir de una falla en sistemas de procesamiento concurrente.

RECUPERACIÓN MEDIANTE ROLLBACK/ROLLFORWARD. Una segunda opción es hacer una copia de la base de datos (guardar la base de datos) periódicamente y llevar un registro de los cambios que se hicieron y guardaron en la base de datos con las transacciones. Así, cuando hay una falla, se puede optar por uno de los dos métodos. Si se usa el primer método, llamado **rollforward,** la base de datos se restaura usando la información que ha sido guardada y todas las transacciones válidas se aplican nuevamente (no estamos reprocesando las transacciones, ni los programas de aplicación están implicados en el avance. En lugar de eso, los cambios procesados, como se grabaron en el registro, se aplican nuevamente).

El segundo método es **rollback,** con el cual deshacemos los cambios que se realizaron con las transacciones erróneas, o parcialmente procesadas. De esta manera, las transacciones válidas que estaban en proceso en el momento de la falla se reinician.

Ambos métodos requieren que se mantenga un **registro** de las transacciones (log). Éste contiene los cambios de datos en orden cronológico. Las transacciones se deberán escribir en el log antes de que se apliquen a la base de datos. De ese modo, si el sistema se colapsa entre el momento en que una transacción se registra y el momento en que se aplica, entonces en el peor de los casos existirá un registro de una transacción no aplicada. Si, por otro lado, las transacciones se aplicaron antes de que fueran registradas, sería posible (aunque también indeseable) cambiar la base de datos, pero sin tener el registro del cambio. Si esto sucede, un usuario no precavido podría ingresar una transacción ya terminada.

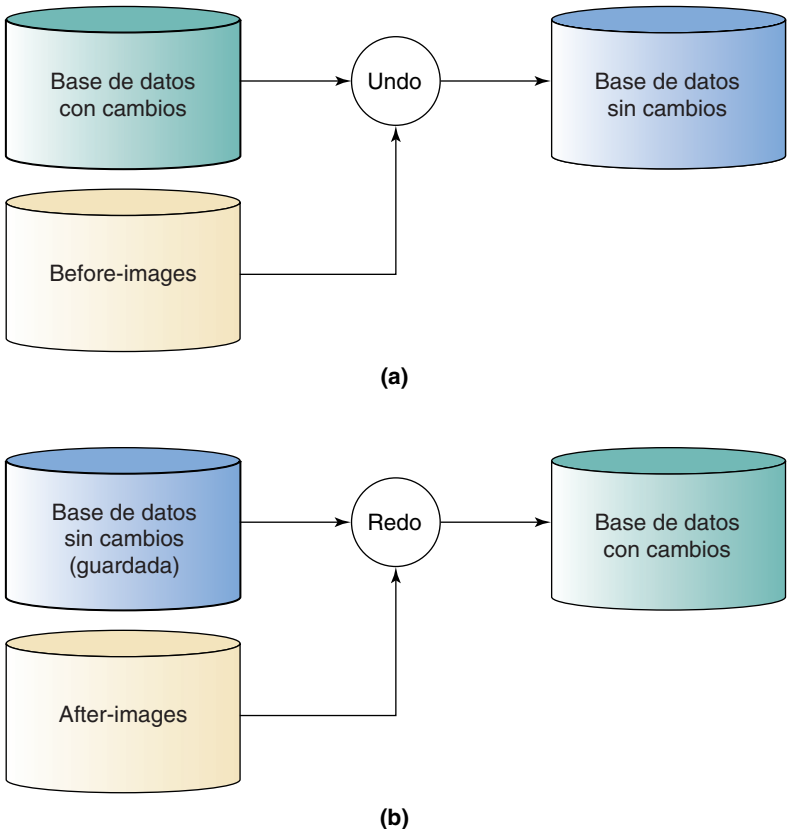
Cuando ocurre una falla, el log se usa tanto para rehacer (redo) como para deshacer (undo) las transacciones, como se muestra en la figura 11-16. Para deshacer (undo) una transacción, el log debe contener una copia de cada registro de la base de datos (o página) antes de que ésta haya cambiado. Dichos registros se llaman **before-images.** Una transacción se deshace aplicando las before-images de todos sus cambios en la base de datos.

Para rehacer (redo) una transacción el log debe contener una copia de cada registro de la base de datos (o página) después de que cambió. Estos registros se llaman **after-images.** Una transacción se rehace aplicando las after-images de todos sus cambios en la base de datos. Los posibles elementos de datos de un log de transacción se muestran en la figura 11-17(a).

Para este ejemplo de log, cada transacción tiene un nombre único con fines de identificación. Además, todas las imágenes de una transacción determinada están encadenadas mediante apuntadores. Un apuntador apunta al cambio previo hecho por la transacción (reverse pointer) y el otro apunta al siguiente cambio (forward pointer). Un cero en el campo del apuntador significa que es el final de la lista. El subsistema de recuperación DBMS usa estos apuntadores para localizar todos los registros para una transacción en particular. La figura 11-17(b) muestra un ejemplo de los encadenamientos de los registros del log.

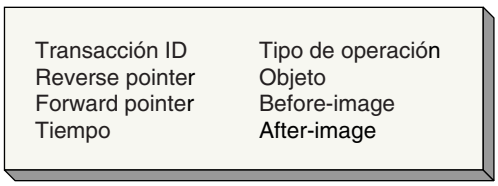
➤ FIGURA 11-16

Transacciones deshacer y rehacer: (a) se eliminan los cambios en la base de datos (rollback), y (b) se reaplican los cambios en la base de datos (rollforward)



➤ FIGURA 11-17

Log de una transacción: (a) elementos de datos en un log, y (b) ejemplo del log para tres transacciones



Número de registro relativo

1	OT1	0	2	11:42	INICIO			
2	OT1	1	4	11:43	MODIFICAR	CLIENTE 100	(valor viejo)	(valor nuevo)
3	OT2	0	8	11:46	INICIO			
4	OT1	2	5	11:47	MODIFICAR	SP AA	(valor viejo)	(valor nuevo)
5	OT1	4	7	11:47	INSERTAR	PEDIDO 11		(valor nuevo)
6	CT1	0	9	11:48	INICIO			
7	OT1	5	0	11:49	CONFIRMAR			
8	OT2	3	0	11:50	CONFIRMAR			
9	CT1	6	10	11:51	MODIFICAR	SP BB	(valor viejo)	(valor nuevo)
10	CT1	9	0	11:51	CONFIRMAR			

(b)

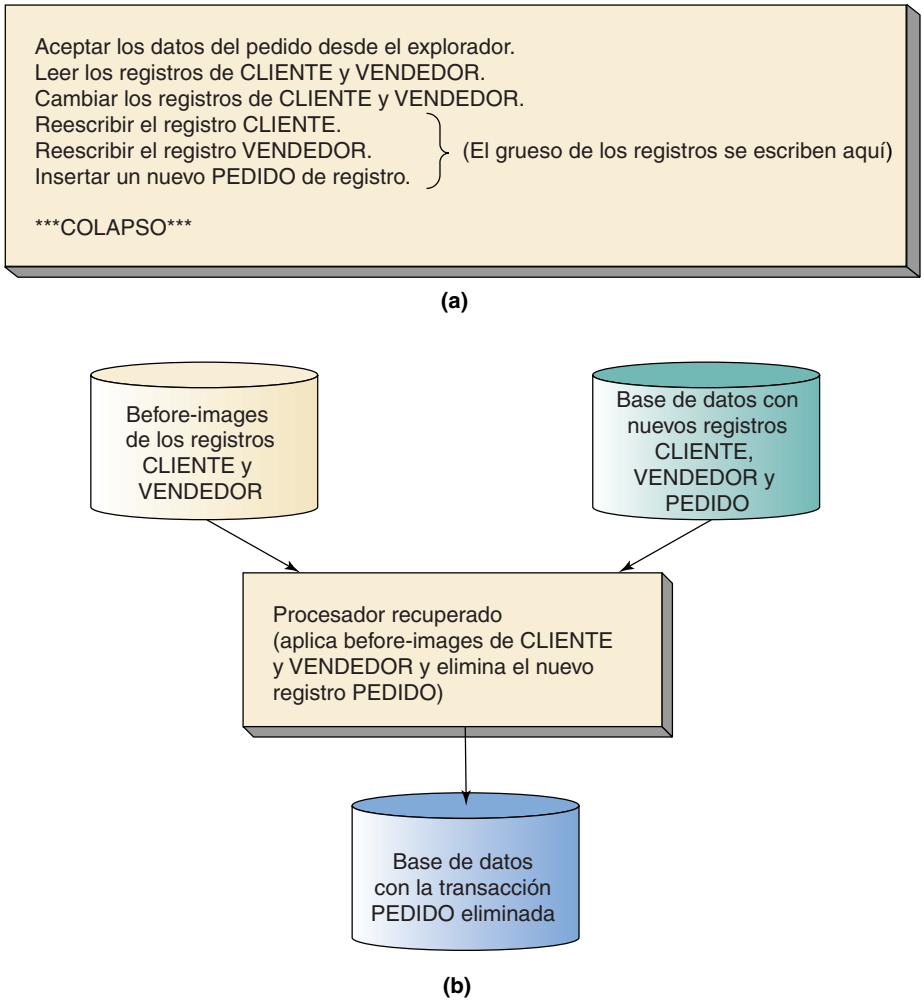
Otros elementos de datos en el log son: el tipo de operación (START señala el comienzo de una transacción y COMMIT termina una transacción, liberando así todos los locks que había); el objeto que actuó, por ejemplo el tipo e identificador de registro; y, por último, las before-images y las after-images.

Si consideramos un registro con before-images y con after-images, las acciones undo y redo son directas. Para deshacer la transacción en la figura 11-18 el procesador de recuperación simplemente remplaza cada registro cambiado con su before-image. Cuando se han restaurado todas las before-images se deshace la transacción. Para rehacer una transacción el procesador de recuperación inicia con la versión de la base de datos en el momento en que comenzó la transacción y se aplica a todas las after-images. Como se indicó, esta acción supone que hay una versión anterior de la base de datos desde una base de datos guardada.

Restaurar una base de datos a su versión más reciente y reaplicar todas las transacciones puede requerir un procesamiento considerable. Para reducir el retraso los productos DBMS algunas veces usan puntos de verificación. Un **checkpoint** es un punto de sincronización entre la base de datos y el log de transacciones. Para ejecutar un checkpoint el DBMS rechaza nuevas peticiones, finaliza las peticiones de procesamiento pendientes y escribe sus buffers en un disco. El DBMS espera hasta que el sistema operativo le notifica que todos los requerimientos escritos pendientes en la base de datos y en el log se han terminado con éxito. En este punto, el log y la base de datos están sincronizados. Así, un log del checkpoint se escribe en el log. Más tarde, la base de datos se puede recuperar a partir del checkpoint y de las after-images para las transacciones que comenzaron después de que fue necesario aplicar el checkpoint.

➤ FIGURA 11-18

Ejemplo de una estrategia de recuperación: (a) la transacción PEDIDO, y (b) procesamiento de recuperación para deshacer un registro de PEDIDO



Los checkpoints son operaciones baratas y es factible usar tres o cuatro (o más) por hora. De esta manera, no se necesitan más de 15 o 20 minutos de procesamiento para la recuperación. La mayoría de los productos DBMS se verifican a sí mismos automáticamente, lo que hace innecesaria la intervención humana.

En los dos capítulos siguientes verá ejemplos específicos de técnicas de respaldo y recuperación para Oracle y SQL Server. Por ahora, sólo necesita entender las ideas básicas y darse cuenta de que es responsabilidad del DBA asegurarse que los planes de respaldo y recuperación se lleven a cabo, que la base de datos se guarde, y que los registros se hayan generado como se requirió.

Administración del DBMS

Además de encargarse del manejo de datos y de la estructura de la base de datos, el DBA tiene que administrar el mismo DBMS. Debe recopilar y analizar las estadísticas con respecto al desempeño de los sistemas e identificar las áreas potenciales de problema. Tenga en cuenta que la base de datos está sirviendo a muchos grupos de usuarios. El DBA necesita investigar todas las quejas con respecto al tiempo de respuesta de los sistemas, la exactitud de los datos, la facilidad de su uso, etcétera. Si es necesario hacer cambios, el DBA debe planearlos e implementarlos.

El DBA debe monitorear periódicamente la actividad de los usuarios en la base de datos. Los productos DBMS incluyen características para recolectar y hacer reportes estadísticos. Por ejemplo, algunos de éstos pueden indicar qué usuarios han estado activos, qué archivos y quizás cuáles elementos de los datos se han usado, y qué métodos de acceso se han empleado. Los porcentajes de error y los tipos de fallas también se pueden capturar y reportar. El DBA analiza esta información para determinar si se necesita un cambio en el diseño de la base de datos para mejorar el desempeño, o para facilitar las tareas de los usuarios. Si es necesario el cambio, el DBA se asegurará de que se lleve a cabo.

El DBA debe analizar las estadísticas en cuanto a tiempo de funcionamiento y desempeño de la base de datos. Cuando se identifique un problema de desempeño (ya sea mediante un reporte o una queja del usuario), el DBA deberá determinar si procede una modificación de la estructura de la base de datos o del sistema. Algunos ejemplos de posibles modificaciones a la estructura son el establecimiento de nuevas llaves, la purga de datos, la eliminación de llaves y la asignación de nuevas relaciones entre los objetos.

Cuando el vendedor del DBMS comienza a anunciar las características de un nuevo producto, el DBA debe considerarlas a la luz de las necesidades de toda la comunidad de usuarios. Si decide incorporar las nuevas características, tendrá que notificarlo a los programadores y capacitarlos. Por consiguiente, el DBA debe administrar y controlar los cambios en el DBMS, así como en la estructura de la base de datos.

Otros cambios en el sistema de los cuales el DBA es responsable varían dependiendo del producto DBMS, así como del software y hardware en uso. Por ejemplo, los cambios en el software (tal como el sistema operativo o el Servidor Web) pueden significar que algunas características del DBMS, las funciones, o los parámetros se deben cambiar. Por lo tanto, el DBA también debe adaptar el producto DBMS al software en uso.

Las opciones del DBMS (tales como los niveles de aislamiento de las transacciones) se eligen inicialmente cuando se sabe muy poco acerca de cómo se desempeñará el sistema en el ambiente del usuario. Por lo tanto, la experiencia operacional y el análisis de desempeño en un periodo determinado pueden revelar si son necesarios los cambios. Aun cuando el desempeño parezca aceptable, el DBA quizá quiera modificar las opciones y observar el efecto en el desempeño. Este proceso se llama *afinación,* u *optimización,* del sistema. La figura 11-19 resume las responsabilidades del DBA en cuanto a la administración del producto DBMS.

➤ FIGURA 11-19

Resumen de las responsabilidades del DBA en la administración del DBMS

- Generar reportes sobre el rendimiento de las aplicaciones de la base de datos
- Investigar las quejas de los usuarios acerca del desempeño
- Evaluar la necesidad de cambios en la estructura de la base de datos o en el diseño de la aplicación
- Modificar la estructura de la base de datos
- Evaluar e implementar las nuevas características del DBMS
- Afinar u optimizar el DBMS

Mantenimiento del repositorio de datos

Considere una aplicación de la base de datos en Internet grande y activa, como las que usan las compañías comerciales a través del correo electrónico; por ejemplo, una aplicación que usa una compañía que vende música en Internet. Este sistema puede implicar información de varias bases de datos diferentes, docenas de diferentes páginas Web, y cientos o miles de usuarios.

Suponga que la compañía que está usando esta aplicación decide expandir su línea de productos para incluir la venta de artículos deportivos. El administrador de esta compañía podría pedir al DBA que desarrolle un estimado de tiempo y de otros recursos que se requieran para modificar la aplicación de la base de datos con el fin de agregar esta nueva línea de productos.

Para que el DBA responda a este requerimiento necesita los metadatos exactos de la base, de las aplicaciones y sus componentes, de los usuarios y sus derechos y privilegios, y con respecto a otros elementos del sistema. La base de datos contiene algo de estos metadatos en las tablas del sistema, pero no son adecuados para responder las preguntas que ha planteado el administrador general. El DBA necesita metadatos adicionales con respecto a los objetos COM y ActiveX, los procedimientos de escritura y las funciones, las páginas ASP, hojas de estilo, definición de tipos de documentos y cosas por el estilo. Además, si bien es cierto que los mecanismos de seguridad del DBMS hacen documentos de usuarios, grupos y privilegios, lo realizan de una forma sumamente estructurada y con frecuencia inconveniente.

Por todos estos motivos, muchas organizaciones desarrollan y mantienen **repositorios de datos,** los cuales son conjuntos de metadatos acerca de la base, aplicaciones de ésta, páginas Web, usuarios y otros componentes de la aplicación. El repositorio puede ser virtual porque se compone de metadatos de muchas fuentes diferentes: el DBMS, software de control de versiones, las bibliotecas de código, la generación de páginas Web, las herramientas de edición, etc. Asimismo, el repositorio de datos puede ser un producto integrado de la herramienta CASE, o de otras compañías tales como Microsoft u Oracle.

De cualquier manera, el momento en que el DBA debe pensar cómo construir un tipo de repositorio es mucho antes de que el administrador haga preguntas. De hecho, el repositorio se debe construir conforme se desarrolle el sistema y debería considerarse como parte importante de los productos a entregar del sistema. De lo contrario, el DBA estará siempre tratando de mantener las aplicaciones que están en uso, adaptándolas a las nuevas necesidades y de algún modo juntando los metadatos para formar un repositorio.

Los mejores repositorios son **activos,** son parte del proceso de desarrollo del sistema, de tal manera que los metadatos se originan automáticamente conforme se crean los componentes del sistema. Menos deseables, pero más eficaces, son los **repositorios pasivos** que se llenan sólo cuando alguien se toma el tiempo de generar los metadatos necesarios y colocarlos en el repositorio.

Internet ha creado enormes oportunidades para que los negocios expandan su base de clientes y aumenten sus ventas y ganancias. Las bases de datos y las aplicaciones

de la base que manejan estas compañías son un elemento esencial de su éxito. Por desgracia existirán organizaciones cuyo crecimiento se obstaculice por su falta de habilidad para aumentar sus aplicaciones o adaptarlas a las necesidades de cambio. Con frecuencia es más fácil construir un nuevo sistema que adaptar uno que ya existe; ciertamente construir un nuevo sistema que se integre a uno viejo mientras lo remplaza puede ser muy difícil.

➤ RESUMEN

Las bases de datos multiusuarios plantean problemas difíciles para las organizaciones que las crean y las usan, y la mayoría han creado una oficina de administración de la base de datos para asegurarse de que esos problemas se resuelvan. En este texto, el término *administrador de la base de datos* se refiere a la persona u oficina involucrada en una sola base de datos. El término *administrador de datos* se usa para describir una función similar que tiene que ver con todos los datos de los activos de una organización. En el capítulo 17 se analizará la administración de datos. Las funciones principales del administrador de la base de datos se enumeran en la figura 11-1.

El DBA se involucra en el desarrollo inicial de las estructuras de la base de datos y tiene el control de la configuración cuando se necesite hacer cambios. Documentar de manera precisa la estructura y las modificaciones que se hagan es una función importante del DBA.

La meta del control de concurrencia es asegurar que el trabajo de un usuario no interfiera con el de otro usuario. Ninguna técnica para el control de concurrencia es ideal para todas las circunstancias. Se necesita hacer acuerdos entre el nivel de protección y el rendimiento eficaz. Una transacción o unidad lógica de trabajo es una serie de acciones que se efectuaron en la base de datos, la cual tiene lugar como una unidad atómica; ya sea que todas o ninguna ocurren. La actividad de las transacciones concurrentes está soportada en el servidor de la base de datos. En algunos casos, se pueden perder las actualizaciones si no se controlan las transacciones concurrentes. Otro problema de concurrencia es el de las lecturas inconsistentes.

Para evitar problemas de concurrencia se bloquean los elementos de la base de datos. Los locks implícitos se ponen a través del DBMS; los explícitos, mediante el programa de aplicación. El tamaño del recurso bloqueado se llama granularidad del lock. Un lock excluyente prohíbe a otros usuarios leer el recurso bloqueado; un lock compartido permite a otros usuarios leer el recurso bloqueado, pero no pueden actualizarlo.

A dos transacciones que funcionan coincidentemente y generan resultados que son consistentes con los que se obtendrán por separado se les llama transacciones serializables. El lock de dos fases, en el cual los locks entran en una fase creciente y se liberan en una fase decreciente, es un esquema para la serialización. Un caso especial de lock de dos fases es adquirir locks durante la transacción, pero no liberar ninguno hasta que ésta termine.

El deadlock, o abrazo mortal, tiene lugar cuando dos transacciones están cada una esperando un recurso que la otra transacción conserva. El deadlock se puede evitar requiriendo que las transacciones tengan todos los locks al mismo tiempo; una vez que esto ocurre, la única manera para remediarlo es abortar una de las transacciones (y regresar al trabajo parcialmente terminado). El locking optimista supone que no ocurrirá ningún conflicto de transacción y enfrenta las consecuencias cuando las hay. El locking pesimista supone que el conflicto ocurrirá, así que se evita con anticipación mediante locks. En general, se prefiere el lock optimista en Internet y en muchas aplicaciones de intranet.

La mayoría de las aplicaciones no declaran explícitamente los locks. En lugar de eso, señalan los límites de la transacción con instrucciones de transacción BEGIN, COMMIT y ROLLBACK e indican el comportamiento concurrente que quieren. Entonces el DBMS bloquea la aplicación, lo que dará como resultado el comportamiento que se requiere.

Una transacción ACID es atómica, consistente, aislada y durable. Durable significa que los cambios de la base de datos son permanentes. Consistente puede significar ya sea a nivel de instrucción, o consistencia a nivel de transacción. Con la consistencia a nivel de transacción, una transacción puede no ver sus propios cambios. El estándar SQL de 1992 define cuatro niveles de aislamiento de transacción: leer sin confirmar, leer confirmando, leer repetible, y serializable. Las características de cada una están resumidas en la figura 11-10.

Un cursor es un apuntador a un conjunto de registros. Sobresalen cuatro tipos: rollforward, estático, keyset, y dinámico. Los programadores deben seleccionar los niveles de aislamiento y tipos de cursores que son apropiados para la carga de trabajo de su aplicación y para el producto DBMS que se están usando.

El objetivo de la seguridad de la base de datos es asegurar que sólo los usuarios autorizados puedan ejecutar actividades permitidas en momentos específicos. Para desarrollar una seguridad eficaz de la base de datos, se deben establecer los derechos de procesamiento y las responsabilidades de todos los usuarios.

Los productos DBMS proporcionan dispositivos de seguridad. La mayoría implican la declaración de los usuarios, objetos que deben ser protegidos y permisos o privilegios de los objetos. Casi todos los productos DBMS usan alguna forma de seguridad, tal como nombre del usuario y contraseña. La seguridad del DBMS se puede aumentar mediante la seguridad de la aplicación.

Si el sistema falla, se debe restaurar esa base de datos a un estado en que se pueda usar lo más pronto posible. Las transacciones en proceso en el momento de la falla se deben reaplicar o reiniciar. Mientras que en algunos casos la recuperación se puede llevar a cabo mediante el reprocesamiento, casi siempre se prefiere el uso de logs, rollback y rollforward. Los ckeckpoints se pueden consultar para reducir la cantidad de trabajo que se debe repetir después de una falla.

Además de estas tareas. El DBA administra el mismo producto DBMS, midiendo el desempeño de la aplicación de la base de datos y evaluando las necesidades de cambio en la estructura, o afinando el desempeño del DBMS. El DBA también se debe asegurar que los nuevos dispositivos del DBMS sean evaluados y usados correctamente. Por último, el DBA es responsable de conservar el repositorio de la base de datos.

➤ PREGUNTAS DEL GRUPO I

11.1 Describa brevemente cinco problemas difíciles para empresas que crean y usan bases de datos multiusuarios.

11.2 Explique la diferencia entre un administrador de base de datos y un administrador de datos.

11.3 Enumere siete tareas importantes de un DBA.

11.4 Resuma las responsabilidades del DBA en cuanto a la administración de la estructura de la base de datos.

11.5 ¿Qué es el control de configuración? ¿Por qué es necesario?

11.6 Explique el significado de la palabra *inapropiadamente* en la frase “que el trabajo de un usuario no afecte inapropiadamente el de otro”.

11.7 Explique las inconveniencias del control de concurrencia.

11.8 Defina una transacción atómica y explique por qué es importante la atomicidad.

11.9 Explique la diferencia entre transacciones concurrentes y simultáneas. ¿Cuántas CPU se requieren para las transacciones simultáneas?

11.10 Dé un ejemplo, diferente al de este texto, sobre el problema de actualización con pérdida.

11.11 Explique la diferencia entre un lock explícito y uno implícito.

11.12 ¿Qué es la granularidad del lock?

11.13 Explique la diferencia entre un lock excluyente y uno compartido.

11.14 Explique el lock de dos fases.

11.15 ¿De qué manera se relaciona la liberación de todos los locks al final de la transacción con el lock de dos fases?

11.16 En general, ¿cómo se deberían definir las fronteras de una transacción?

11.17 ¿Qué es el deadlock? ¿Cómo se puede evitar? ¿Cómo se puede solucionar cuando ocurre?

11.18 Explique las diferencias entre lock optimista y pesimista.

11.19 Explique los beneficios de señalar fronteras de transacción, declarar las características del lock, y dejar que el DBMS los ponga.

11.20 Explique el uso de los enunciados BEGIN, COMMIT Y ROLLBACK TRANSACTION.

11.21 Explique el significado de la expresión: transacción ACID.

11.22 Describa consistencia a nivel de instrucción.

11.23 Describa consistencia a nivel de transacción.

11.24 ¿Cuál es el propósito de los niveles de aislamiento de la transacción?

11.25 Explique el nivel de aislamiento de la lectura no confirmada. Exponga un ejemplo de su uso.

11.26 Explique el nivel de aislamiento de la lectura confirmada. Dé un ejemplo sobre su uso.

11.27 Explique el nivel de aislamiento de la lectura repetible. Proporcione un ejemplo de su uso.

11.28 Explique el nivel de aislamiento serializable. Mencione un ejemplo de su uso.

11.29 Explique el término *cursor*.

11.30 Explique por qué una transacción puede tener muchos cursores. También, ¿cómo es posible que una transacción pueda tener más de un cursor en una tabla determinada?

11.31 ¿Cuál es la ventaja de usar diferentes tipos de cursores?

11.32 Explique los cursores forward only. Dé un ejemplo de su uso.

11.33 Explique los cursores estáticos. Proporcione un ejemplo de su uso.

11.34 Explique los cursores de keyset. Mencione un ejemplo de su uso.

11.35 Explique los cursores dinámicos. Dé un ejemplo de su uso.

11.36 ¿Qué sucede si usted no establece en el DBMS el nivel de aislamiento de la transacción y el tipo de cursor? ¿Esto es bueno o malo?

11.37 Explique la necesidad de definir los derechos y las responsabilidades del procesamiento. ¿Cómo se imponen estas responsabilidades?

11.38 Explique las relaciones de USUARIOS, GRUPOS, PERMISOS, Y OBJETOS para un sistema de seguridad de una base de datos genérica.

11.39 Describa las ventajas y desventajas de la seguridad que proporciona el DBMS.

11.40 Describa las ventajas y desventajas de seguridad que proporciona la aplicación.

11.41 Explique cómo se puede recuperar una base de datos a través del reprocesamiento. ¿Por qué en general no es factible?

11.42 Defina *rollback* y *rollforward*.

11.43 ¿Por qué es importante escribir en el registro antes de cambiar los valores de la base de datos?

11.44 Describa el proceso rollback. ¿Bajo qué condiciones se debe usar?

11.45 Describa el proceso rollforward. ¿Bajo qué condiciones se debe usar?

11.46 ¿Cuál es la ventaja de establecer checkpoints de una base de datos?

11.47 Resuma las responsabilidades del DBA en la administración del DBMS.

11.48 ¿Qué es un repositorio de datos? ¿Un repositorio de datos pasivo? ¿Un repositorio de datos activo?

11.49 Explique por qué es importante un repositorio de datos. ¿Qué puede suceder si no hay uno?

➤ PREGUNTAS DEL GRUPO II

11.50 Visite www.microsoft.com y busque información acerca de los niveles de aislamiento y tipos de cursores. Por ahora, ignore la información acerca de RDS, ADO, ODBC, Y OLE/DB y concéntrese en las características y funciones del SQL Server. Compare y contraste esas capacidades con las que describimos.

11.51 Visite www.oracle.com y busque información acerca de los niveles de aislamiento y tipos de cursores. Ignore la información acerca de ODBC y concéntrese en las características y funciones de Oracle. Compare y contraste esas capacidades con las que describimos en este texto.

11.52 Describa las ventajas y desventajas de la seguridad del nombre del usuario y la contraseña. ¿De qué manera los usuarios pueden descuidar sus identidades? ¿Cómo puede comprometer la seguridad de la base de datos ese descuido? ¿Qué pasos se podrían seguir para reducir el riesgo de esos problemas?

11.53 Busque la Web de las herramientas CASE que proporcionan repositorios y productos para repositorio. Decida cuál es el mejor y enumere sus funciones y características principales. Explique cómo se podrían usar las funciones y características para ayudar al DBA de una compañía comercial que se anuncia en Internet a que agregue nuevas líneas de productos a su negocio.

➤ PROYECTO

A. Considere la vista Cliente de la figura 10-4.

1. Suponga que usted está desarrollando una aplicación para crear nuevas instancias de esta vista. ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores involucrados y recomiende un tipo de cursor para cada uno.
2. Suponga que está desarrollando una aplicación para modificar los valores (sólo valores, no relaciones) de esta vista. ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores y recomiende un tipo de cursor para cada uno.
3. Suponga que está desarrollando una aplicación para modificar tanto los valores de datos como las relaciones en esta vista. ¿Cómo cambia su respuesta a la pregunta A.2?
4. Suponga que está desarrollando una aplicación para eliminar instancias de clientes inactivos (definidos como clientes que nunca han comprado arte). ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores implicados y recomiende un tipo de cursor para cada uno.

➤ PREGUNTAS DEL PROYECTO FIREDUP

A. Suponga que la compañía FiredUp lo ha contratado como un consultor de base de datos para desarrollar su base de datos operacional, considerando las cuatro tablas descritas al final del capítulo 9. Suponga que el personal de FiredUp son los dos dueños, un administrador de oficina, un técnico en reparaciones y dos empleados de producción. El administrador de la oficina procesa todas las formas de registro. El técnico en reparaciones introduce la información de reparación, y los empleados de producción, los datos de las estufas que han producido. Prepare un memorándum de tres a cinco páginas, dirigido a la gerencia de FiredUp, con los siguientes puntos:

1. La necesidad de contar con un administrador de la base de datos en la compañía FiredUp.
2. Su recomendación sobre quién debe servir como administrador de la base de datos. Suponga que FiredUp no es lo suficientemente grande como para contratar un administrador de la base de datos de tiempo completo.
3. Usando la figura 11-1 como guía, describa la naturaleza de las actividades de la administración de la base de datos de FiredUp. Como consultor audaz, tenga en cuenta que puede recomendarse a sí mismo para desempeñar algunas de las funciones del DBA.

B. Para los empleados que se mencionan en la pregunta A, defina usuarios, grupos y permisos de información en las cuatro tablas que se describen al final del capítulo 9. Utilice el esquema de seguridad que se muestra en la figura 11-13 como ejemplo. Nuevamente, no olvide incluirse usted mismo.

C. Considere REGISTRO_VISTA y ESTUFA_REPARACIÓN_VISTA definidas al final del capítulo 10.

1. Mencione un ejemplo de una lectura sucia, una lectura no repetible y una lectura fantasma cuando se procesa REGISTRO_VISTA.
2. ¿Qué medidas de control de concurrencia cree que serían apropiadas para una transacción que actualice REGISTRO_VISTA? ¿Qué medidas de control de concurrencia piensa usted que serían apropiadas para una transacción que elimine la información de REGISTRO_VISTA? Establezca sus hipótesis.
3. Responda la pregunta 2 en el caso de CLIENTE_VISTA. Justifique cualquier diferencia de su respuesta con la pregunta 2.
4. ¿Qué nivel de aislamiento usaría para una transacción que imprime un reporte listando todas las reparaciones de estufas durante un periodo determinado? ¿Qué tipo de cursor usaría?

CAPÍTULO 12

Administración de bases de datos con Oracle

Oracle es un DBMS (Database Management System, por sus siglas en inglés) poderoso y robusto que funciona en muchos sistemas operativos diferentes, incluyendo Windows 98, Windows 2000, diversas variantes de UNIX, diferentes sistemas operativos de macrocomputadoras, y Linux. Éste es el DBMS más popular del mundo y tiene una larga historia de desarrollo y uso. Oracle muestra al programador mucha de su tecnología y consecuentemente puede afinarse y ajustarse de diversas maneras.

Sin embargo, esto quiere decir que Oracle puede ser difícil de instalar y hay mucho que aprender. Una muestra de la amplitud de Oracle es que uno de sus referentes más populares, *Oracle 8i, The Complete Reference*, escrito por Loney y Koch, tiene más de 1,300 páginas y no abarca todo. Además, puede ser necesario modificar las técnicas que funcionan con una versión de Oracle en un sistema operativo, cuando se esté trabajando con una versión en un sistema operativo diferente. Tendrá que ser paciente con Oracle y consigo mismo y estar consciente de que no dominará este tema de la noche a la mañana.

Existen muchas configuraciones de la serie Oracle. Para empezar, hay dos versiones diferentes del motor DBMS de Oracle: Oracle personal y Oracle empresarial. Además, existen Formas y Reportes y también el Diseñador Oracle, así como un sistema central de herramientas para la publicación de las bases de datos de Oracle en la Web. Agregue a lo anterior la necesidad que tienen de operar en muchos sistemas operativos diferentes, en redes mediante diversos protocolos de comunicación, y verá que aprenderlo es considerablemente complejo.

Oracle SQL Plus es un programa de utililería para el procesamiento de SQL y la creación de componentes tales como los procedimientos almacenados (stored procedures) y los disparadores (triggers). Asimismo, es un componente constante a lo largo de todas estas configuraciones de productos. Por lo anterior concentraremos toda la atención en éste. El último, el PL/SQL, es un lenguaje de programación que agrega construcciones de programación al lenguaje SQL. Emplearemos el PL/SQL para crear tablas, restricciones, relaciones, vistas, procedimientos almacenados (stored procedures) y disparadores (triggers).

Este capítulo utiliza el ejemplo View Ridge del capítulo 10, y el análisis puede ser burdamente paralelo al de la administración de la base de datos del capítulo 11.

➤ INSTALACIÓN DE ORACLE

Para aprovechar completamente este capítulo es conveniente tener instalado Oracle en su computadora. Conforme avancemos, repita las acciones que se describen. Puede tener dos versiones de la edición personal: Windows 98 o Windows 2000. Puede utilizar la versión que tenga. Será suficiente con el Oracle básico.

➤ CREACIÓN DE UNA BASE DE DATOS DE ORACLE

Existen tres formas de crear una base de datos de Oracle: mediante el asistente de configuración de la base de datos, a través de los procedimientos de creación de la base de datos proporcionada por Oracle, o por medio de la instrucción SQL CREATE DATABASE (CREAR UNA BASE DE DATOS). El asistente de configuración de la base de datos de Oracle es por mucho el más fácil, por lo tanto es el que debe usar.

Puede encontrar el asistente de configuración de la base de datos en uno de los directorios cuando se instaló Oracle. Dependiendo de su sistema operativo, lo encontrará haciendo clic en Start/Programs/Oracle-OraHome81/Database Administration, o algo similar. Probablemente sus directorios no se llamen exactamente igual; búsquelo a través de los directorios debajo de Start/Programs (Inicio/Programas) para encontrar el asistente de configuración de la base de datos.

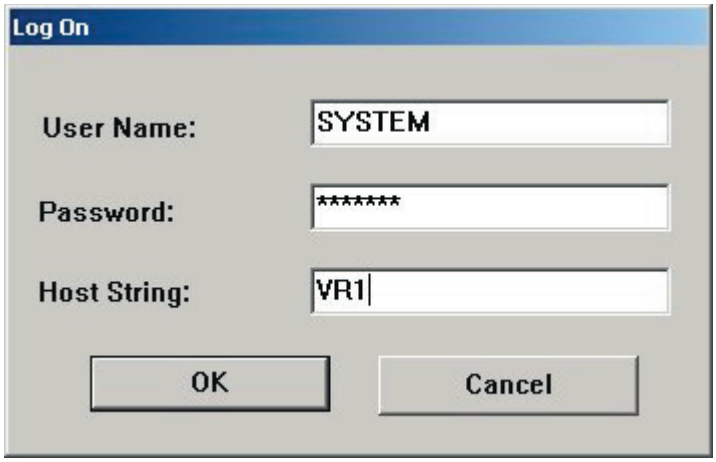
Cuando inicie el asistente de configuración verá la figura 12-1. Haga clic en *Create a database* y seleccione en la forma usual; elija copiar los archivos de la base de datos del CD; después seleccione un nombre para su base. En este capítulo se utiliza el nombre de la base de datos VR1. Haga clic en Finish (Terminar) y se creará su base de datos.

➤ FIGURA 12-1

Creación de una base de datos de Oracle

➤ FIGURA 12-2

Cuenta Oracle



Por acción predeterminada, Oracle crea tres cuentas en su nueva base de datos: *INTERNAL,* con contraseña *ORACLE; SYS,* con contraseña *CHANGE _ON_ INSTALL;* y *SYSTEM,* con contraseña *MANAGER.* (Los nombres y contraseñas de las cuentas son insensibles a mayúsculas y minúsculas.)

Oracle creará archivos predeterminados para sus datos y para logs de actividad de datos. El programador tiene gran control sobre el modo y el lugar donde se crean estos archivos, pero aquí se omitirá el análisis del tema. Usted podrá investigar más cuando haya aprendido los fundamentos de Oracle.

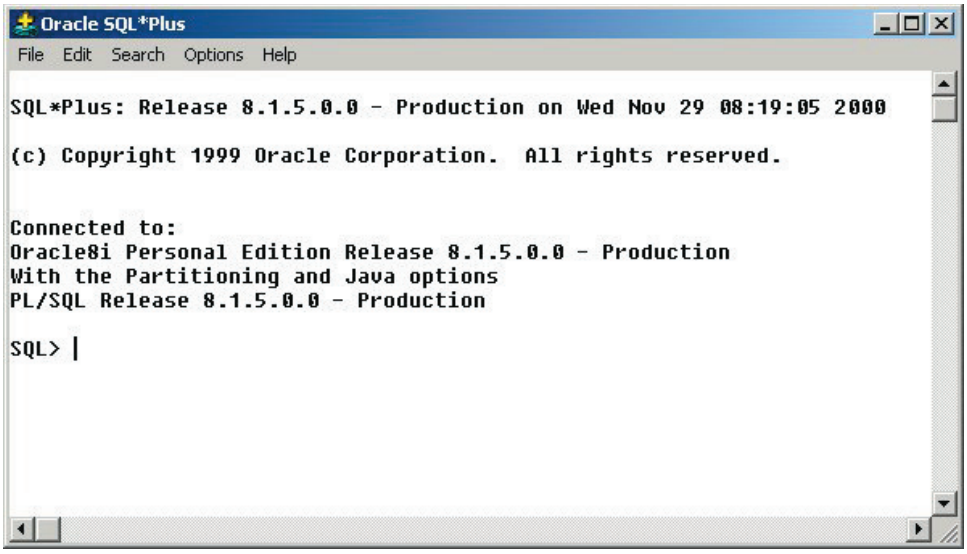
Uso de SQL Plus

Para utilizar SQL Plus encuentre el ícono correspondiente debajo de los menúes Start/Programs y haga clic. Ingrese a su base de datos mediante la cuenta SYSTEM e introduzca el nombre de su base de datos debajo del *Host String*, como se muestra en la figura 12-2. (Este procedimiento puede ser diferente si está empleando una versión de Oracle instalada por alguien más. Si es el caso, compruébelo con su administrador de base de datos.) Haga clic en OK y verá una ventana similar a la de la figura 12-3.

Entre sus muchas funciones, SQL Plus es un editor de textos. El trabajo con Oracle será más fácil si aprende un poco acerca de este editor. Primero, cuando escriba en SQL Plus lo que tecleó se colocará dentro de una memoria intermedia (buffer). Cuando pre-

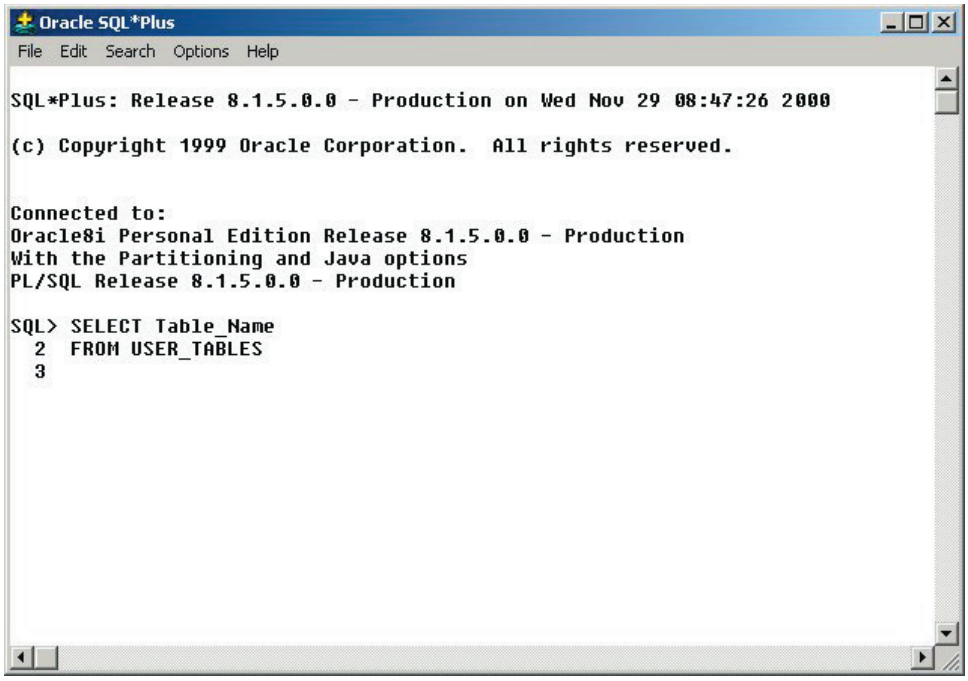
➤ FIGURA 12-3

Apuntador SQL Plus



➤ FIGURA 12-4

Memoria intermedia (buffer) de líneas múltiples de SQL Plus



sione Enter, SQL Plus guardará lo que apenas escribió dentro de un renglón en la memoria intermedia (buffer), e irá a un nuevo renglón, pero no terminará ni ejecutará la instrucción.

MEMORIA INTERMEDIA DE SQL PLUS. Por ejemplo, en la figura 12-4 el usuario introdujo dos renglones en una instrucción SQL. Si es necesario, puede agregar más renglones. Cuando escriba un punto y coma y presione {Enter}, SQL Plus finalizará la instrucción y la ejecutará. Inténtelo, pero ignore los resultados, porque posteriormente los analizaremos.

➤ FIGURA 12-5

Utilización de LIST