BASES DE Datos

• Catherine M. Ricardo

Mc Graw Hill

Bases de datos

Bases de datos

Catherine M. Ricardo

Iona College

Revisión técnica

Ingeniero Antonio González y Peña

Universidad Iberoamericana, Ciudad de México

Doctor Francisco Javier Cartujano

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Ciudad de México

Ingeniera Lucila Patricia Arellano Mendoza

Universidad Nacional Autónoma de México

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA
MADRID • NUEVA YORK • SAN JUAN • SANTIAGO • SÃO PAULO
AUCKLAND • LONDRES • MILÁN • MONTREAL • NUEVA DELHI
SAN FRANCISCO • SINGAPUR • SAN LUIS • SIDNEY • TORONTO

Director Higher Education: Miguel Ángel Toledo Castellanos
Director editorial: Ricardo Alejandro del Bosque Alayón
Editor sponsor: Pablo Roig
Coordinadora editorial: Marcela I. Rocha Martínez
Editora de desarrollo: María Teresa Zapata Terrazas
Supervisor de producción: Zeferino García García
Traductores: Víctor Campos Olguín y Javier Enríquez Brito

BASES DE DATOS

A Subsidiary of ***The McGraw-Hill*** *Companies, Inc.*
Prolongación Paseo de la Reforma 1015, Torre A,
Piso 17, Col. Desarrollo Santa Fe,
Delegación Álvaro Obregón
C. P. 01376, México, D. F.
Miembro de la Cámara Nacional de la Industria Editorial Mexicana, Reg. Núm. 736

ISBN 13: 978-970-10-7275-2

Traducido de la primera edición de DATABASES ILLUMINATED.
Published by Jones and Bartlett Publishers Inc., 40 Tall Pine Drive, Sudbury, MA 01766.

ISBN: 0-7637-3314-8

0123456789 08765432109

Impreso en México *Printed in Mexico*

A mi esposo, Henry, y a Henry Jr., Marta, Cathy, Christine, Tomás y Nicholas

Contenido

Prefacio

Propósito de este libro

El estudio de los sistemas, diseño y gestión de bases de datos es una parte esencial de la educación en ciencias de la computación y de los estudiantes en ciencias de la información. Un curso de bases de datos debe proporcionar un fuerte fondo teórico, práctica en el diseño de bases de datos y la experiencia de crear y desarrollar una base de datos operativa. Al enseñar cursos de bases de datos durante más de 20 años, he utilizado muchos libros de texto diferentes y encontré que algunos se concentran en la teoría y todos ignoran la implementación, mientras que otros presentan un cúmulo de detalles acerca de sistemas de gestión de bases de datos particulares, pero quedan cortos en la teoría. Este libro está diseñado para ayudar a los estudiantes a integrar el material teórico con el conocimiento práctico, mediante un enfoque que tenga una firme base teórica aplicada a la implementación de bases de datos prácticas.

Estructura

Los fundamentos teóricos se presentan primero y los conceptos se usan de manera repetida a lo largo del libro, incluidos los capítulos que tratan la implementación. Al diseño de bases de datos lógicas se le da amplia consideración. El modelo entidad-relación se introduce en los primeros capítulos y luego se mapea al modelo relacional. La normalización relacional se estudia con detalle, y se discuten muchos ejemplos del proceso de normalización. Se presenta el modelo entidad-relación mejorado y se mapea tanto al modelo relacional como al objeto-relacional. El modelo orientado a objetos se presenta con el uso de UML como vehículo para el diseño lógico. XML y el modelo de datos semiestructurado se introducen en el capítulo 13. Un ejemplo continuo de una base de datos universitaria se incorpora a lo largo del texto, para ilustrar los conceptos y técnicas y proporcionar tanto continuidad como contraste. Otros ejemplos se presentan según se necesite. Los sistemas de bases de datos puramente relacionales, objeto-relacional y orientadas a objetos se describen y usan para la implementación de los ejemplos. Los detalles de los sistemas de gestión de bases de datos se describen de modo que los estudiantes puedan aprender las especificidades de estos sistemas en la vida real, desde el nivel de implementación física. En los ejemplos, inicialmente se usa Microsoft Access™, pero Oracle™ se introduce conforme se desarrolla el material. Sin embargo, los ejemplos son adecuados para su uso con cualquier DBMS (sistema de gestión de base de datos) relacional u objeto-relacional.

Propósito del proyecto de muestra y los proyectos estudiantiles

Varios enfoques se integran en el proyecto de muestra, lo que es una característica única de este libro. Desde el final del primer capítulo, los estudiantes ven un proyecto de muestra que se desarrolla conforme el libro avanza. Este proyecto es independiente de los ejemplos que se presentan en el texto del capítulo o en los ejercicios. El diseño comienza con una descripción de la aplicación, una base de datos que necesita una galería de arte. Al final del primer capítulo, los estudiantes ven cómo se especifican las necesidades de información. Después de que los estudiantes estudian las técnicas de planeación en el capítulo 2, se percatan cómo crear un diccionario de datos completo, orientado al usuario y cómo se podrían usar otras herramientas de planeación. Cuando aprenden acerca de los modelos lógicos, observan el desarrollo paso a paso de un diagrama E-R al final del capítulo 3. Comprenden los pasos involucrados para mapear el diagrama al modelo relacional después del capítulo 4, y la normalización del modelo después del capítulo 5. Los detalles completos para crear y manipular una base de datos Oracle™ puramente relacional con el uso de SQL se presentan para el modelo relacional después del capítulo 6. El diagrama E-R se expande a un diagrama EE-R completo y se mapea a una base de datos objeto-relacional en el capítulo 7, con las características objeto-relacional de Oracle. El diagrama EE-R se transforma luego en un diagrama UML en el capítulo 8, que también incluye el diseño y creación de una base de datos orientada a objeto con ODL. El proyecto de muestra se extiende en el capítulo 12 para mostrar un diseño distribuido que se puede usar en un ambiente cliente-servidor o un verdadero ambiente distribuido. En el capítulo 13 se proporcionan los detalles para la creación de un sitio web simple para la galería de arte. Por tanto, cada técnica importante de planeación, diseño e implementación se ilustran usando sistemas de la vida real. A la sección del proyecto de muestra siempre siguen proyectos estudiantiles continuos, que requieren que los estudiantes imiten los pasos de la muestra. A partir de mis experiencias al asignar proyectos similares mientras enseño cursos de bases de datos, encontré que los estudiantes aprenden de manera más efectiva al crear una base de datos operativa y desarrollar sus propios proyectos de continuidad para incorporar nuevos conceptos, y que se benefician al ver muestras conforme avanzan. Tales experiencias realistas proporcionan la comprensión que ninguna cantidad de lecturas puede producir. Los capítulos finales tratan con temas de seguridad de bases de datos, control de concurrencia, técnicas de recuperación, optimización de consultas, bases de datos distribuidas, conflictos sociales y éticos, y almacenes de datos. Los apéndices cubren la organización física de datos, el modelo de red y el modelo jerárquico.

Características de aprendizaje

El estilo de escritura es coloquial. Cada capítulo comienza con un enunciado de los objetivos de aprendizaje. Los ejemplos y las aplicaciones se presentan a lo largo del texto. Las ilustraciones se usan tanto para clarificar el material como para variar la presentación. El proyecto de muestra al final de cada capítulo es parte importante del texto, y proporciona una aplicación del material recién presentado. Los proyectos estudiantiles que siguen a la muestra se introducen en el primer capítulo, y se espera que los estudiantes elijan uno, o se les asigne uno, y que desarrollen dicho proyecto en forma paralela a la muestra conforme avancen en el curso. Los proyectos estudiantiles se pueden realizar individualmente o en grupo. En el texto se incluyen resúmenes de capítulo para ofrecer un rápido repaso o vista previa del material y para ayudar a los estudiantes a comprender la importancia relativa de los conceptos presentados.

Audiencia

El material es adecuado para especialidades en ciencias de la computación o en ciencias de la información con un buen antecedente técnico. Los estudiantes deben completar al menos un año de programación, incluidas estructuras de datos. El libro también se podría usar como un texto introductorio a las bases de datos para estudiantes graduados o para autoestudio.

Mapeo de lineamientos curriculares

Aunque este libro se planeó con base en las experiencias de la autora, también se ajusta a los lineamientos curriculares ACM-IEEE. CC2001 proporciona un modelo de curso de bases de datos, CS270T, para el currículum basado en temas. El curso incluye tres áreas, Interacción humano-computadora (HCI), Gestión de información (IM) y Conflictos sociales y profesionales (SP). A continuación se mencionan las unidades incluidas en CS270T, junto con los correspondientes capítulos en el libro.

IM1 Modelos y sistemas de información (3 horas centrales): capítulos 1, 2

IM2 Sistemas de bases de datos (3 horas centrales): capítulo 2

IM3 Modelado de datos (4 horas centrales): secciones 2.7, 13.4; capítulos 3, 4, 7, 8; apéndices B, C

IM4 Bases de datos relacionales (5 horas): capítulo 4

IM5 Lenguajes de consulta de bases de datos (4 horas): secciones 4.6, 7.7, 8.5; capítulos 6, 11, 13; apéndices B, C

IM6 Diseño de bases de datos relacionales (4 horas): capítulo 5

IM7 Procesamiento de transacción (3 horas): capítulo 10

IM8 Bases de datos distribuidas (3 horas): capítulos 12, 13

IM9 Diseño de bases de datos físicas (3 horas): apéndice A

HCI Fundamentos de interacción humano-computadora (2 horas centrales): capítulo 14

SP6 Propiedad intelectual (3 horas centrales): capítulo 14

SP7 Privacidad y libertades civiles (2 horas centrales): capítulos 9, 14

Temas optativos (1 hora): capítulo 15

El proyecto de muestra y los proyectos estudiantiles también proporcionan práctica en el modelo de datos, diseño de bases de datos, interacción humano-computadora, bases de datos relacionales, lenguajes de consulta de bases de datos, bases de datos distribuidas y diseño de bases de datos físicas. Algunos aspectos de privacidad y libertades civiles se discuten en el proyecto de muestra, y conflictos similares surgirán y se deberán tratar en los proyectos estudiantiles.

Reconocimientos

Muchas personas ofrecieron comentarios útiles, consejos y aliento durante la escritura de este libro. Estudiantes y colegas a lo largo del mundo que usaron mi libro anterior, *Database Systems: Principles, Design, and Implementation*, proporcionaron comentarios que me ayudaron a dar forma a este libro. Los estudiantes en las clases de no graduados y graduados en el Iona College me ayudaron al ofrecer retroalimentación durante las pruebas en clase del primer borrador del manuscrito. Ted Leibfried fue útil para ayudarme a aterrizar el proyecto y concertar la prueba en clase del primer borrador en su institución, la University of Houston en Clearlake. Estoy muy agradecida por su perspicacia y generosa ayuda. Mis colegas en Iona, en especial mi jefe de cátedra, Bob Schiaffino, me brindaron apoyo a lo largo del proceso de desarrollo de este libro. Agradezco al colegio haberme otorgado el año sabático para poder trabajar en este proyecto.

Quiero agradecer a Ayad Boudiab, Georgia Perimeter College; Jeffery Peden, Longwood University; Reggie Haseltine, University of Maryland; Jim Patus, Ivy Tech State College; y Marcus Schaefer, DePaul University, quienes revisaron el primer borrador e hicieron sugerencias útiles que mejoraron enormemente el manuscrito final. Por su aliento y apoyo para este proyecto, agradezco a Michael Stranz, ex editor en jefe en Jones & Bartlett. Quiero agradecer al equipo editorial y de producción en Jones and Bartlett, especialmente a Caroline Senay y Karen Ferreira. Estoy muy agradecida con mi editor, Stephen Solomon, por su guía y apoyo.

Finalmente, quisiera agradecer a mi familia, especialmente a mi esposo, Henry, por su amor, paciencia y comprensión, y por su ayuda activa al criticar y corregir las pruebas del manuscrito.

CAPÍTULO 1

Conceptos introductorios a las bases de datos

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Cómo se usan las bases de datos en la vida cotidiana
- Las principales funciones de un sistema de gestión de bases de datos
- Ventajas de usar un sistema de base de datos integrada
- Desventajas de las bases de datos
- Roles en el entorno de base de datos integrada
- La historia de los sistemas de información

1.1 Bases de datos en la vida cotidiana

En la actualidad, las bases de datos se usan tan ampliamente que se pueden encontrar en organizaciones de todos los tamaños, desde grandes corporaciones y agencias gubernamentales, hasta pequeños negocios e incluso en hogares. Las actividades diarias con frecuencia lo ponen en contacto con las bases de datos, ya sea directa o indirectamente.

- Cuando visita un **portal de Internet del consumidor** que permite navegar y ordenar en línea bienes como libros o ropa, accede a una base de datos. La información acerca de los productos disponibles y los datos acerca del pedido se almacenan en una base de datos. También es posible que pueda ver los datos almacenados acerca de pedidos anteriores que haya levantado. Algunos sitios Web pueden usar información acerca de sus pedidos, o incluso sus actividades de navegación, para sugerir productos o servicios que es probable que le interesen.
- Cuando visita un sitio Web interactivo de **servicio al cliente**, como la página de inicio de una compañía de servicios o una aseguradora de salud, es capaz de acceder a información acerca de sus propios registros de servicios o productos proporcionados. Es posible que sea capaz de actualizar entradas en la base de datos con información personal como su dirección o número telefónico. Algunos sitios Web de servicios al cliente le permiten hacer cambios a los servicios a los que se suscribe. Por ejemplo, su proveedor de servicios telefónicos o compañía eléctrica pueden permitirle cambiar planes en línea.
- Si usa **banca electrónica**, puede recuperar registros de base de datos acerca de depósitos, retiros, pago de facturas y otras transacciones para sus cuentas. Puede transferir fondos, ordenar cheques y realizar muchas otras funciones, todas las cuales involucran el uso de una base de datos.
- Cuando usa una **tarjeta de crédito**, el vendedor por lo general espera la aprobación por computadora de su compra antes de presentarle un recibo para que lo firme. El proceso de aprobación consulta una base de datos para verificar que su tarjeta no se perdió o la robaron y para encontrar su límite de crédito, saldo actual y cantidad de compras ya aprobadas. La base de datos se actualiza automáticamente para reflejar la nueva cantidad aprobada. Para una tarjeta de débito, se consulta la base de datos del banco para verificar su número de cuenta, su NIP, su saldo actual y su saldo ajustado previo a la aprobación de la compra. La cantidad de compra se deduce en forma automática de su cuenta mientras la transacción se completa.
- Cuando compra bienes en un **supermercado** o **tienda al menudeo**, se usan escáneres para leer códigos universales de producto u otros identificadores de mercancía. Al usar el código escaneado, el sistema de base de datos puede identificar el artículo exacto y producir un recibo con el nombre del artículo y su precio, y toma en consideración cualquier precio de venta especial. El sistema también puede proporcionar entrada para un sistema de **control de inventarios**, de modo que el registro de inventario para cada artículo se puede actualizar con el fin de reflejar la venta. Si el inventario cae por abajo de un nivel llamado punto de resurtido, la computadora automáticamente puede colocar un pedido para volver a surtir el inventario.
- Cuando hace planes para viajar, puede ingresar al **sistema de reservaciones de una aerolínea** en la que se usa una base de datos para rastrear los vuelos programados y las reservaciones de pasajeros. Dado que muchos viajeros pueden solicitar reservaciones de manera simultánea, el sistema debe ser capaz de manejar peticiones rápidamente, resolver conflictos y aceptar solicitudes hasta que se alcance el número máximo de asientos. Muchas cadenas hoteleras y compañías de renta de autos tam-

bién tienen sistemas centralizados de reservaciones para aceptar reservaciones en cualquiera de sus ubicaciones, con el uso de un sistema de base de datos integrada.

- Si visita al médico, es posible que sus **registros médicos y datos de facturación** se conserven en una base de datos. Cuando le extienden una receta, probablemente el farmacéutico usará una base de datos para registrar información acerca de la prescripción, comprobar las interacciones con los medicamentos que use en la actualidad e imprimir la etiqueta y la receta. Tanto el médico como el farmacéutico pueden usar sus bases de datos para hacer cobranzas a terceras partes, que automáticamente verifican la cobertura y extienden las cobranzas del seguro para los gastos cubiertos, mientras que usted sólo paga el deducible. A todos los proveedores de salud en Estados Unidos se les requiere proteger la privacidad durante estas transacciones, en concordancia con la legislación de privacidad de la Ley de Transportabilidad de Responsabilidad en Seguros de Salud (HIPAA, por sus siglas en inglés).
- Sus **registros laborales** se pueden mantener en una base de datos que almacena información básica como nombre, dirección, identificación de empleado, labores a desarrollar y evaluaciones de desempeño. La nómina probablemente se produce con el uso de una base de datos que almacena información acerca de cada periodo de pago y datos acerca de pago bruto anual, deducciones de impuestos e impuestos retenidos, entre otras cosas. Su recibo de pago refleja estos datos cada día de pago.
- Sus **registros escolares** tal vez se conservan en una base de datos que se actualiza cada periodo al registrar su inscripción, conclusión y calificación para cada clase.
- Para hacer investigación, puede usar una **base de datos bibliográfica** en la que ingrese palabras clave que describan el tema de interés. Puede obtener resultados que contengan hipertexto, lo que le permite recuperar resúmenes o artículos de interés completos en su área de interés.

Como demuestra este breve panorama de actividades, las bases de datos se usan para satisfacer las necesidades de información de muchas organizaciones e individuos en una variedad de áreas. Sin embargo, una base de datos deficientemente diseñada fracasa para proporcionar la información requerida u ofrece información no actualizada, falsa o contradictoria. Con la finalidad de maximizar sus beneficios potenciales, es importante comprender los fundamentos teóricos, estructura interna, diseño y gestión de las bases de datos.

1.2 Una base de datos de muestra

Considere una base de datos simple que registra información acerca de estudiantes universitarios, las clases que toman durante un semestre y los profesores que imparten las clases. La información de cada estudiante incluye identificación (ID), nombre, especialidad y número total de créditos obtenidos del estudiante. Con el uso de **Microsoft Access** para este ejemplo, se tiene una **tabla** para estos datos, como se muestra en la figura 1.1(a). La tabla `Student` (estudiante) tiene cinco columnas, llamadas `stuId` (identificación del estudiante), `lastName` (apellido), `firstName` (nombre), `major` (especialidad) y `credits` (créditos). Cada fila de la tabla muestra la identificación del estudiante, apellido, nombre, especialidad y número de créditos para un estudiante. Los valores de estos ítems para cada estudiante se colocan en las columnas con los nombres correspondientes. La tabla `Faculty` (facultad) tiene columnas llamadas `facId` (identificación del docente), `name` (nombre), `department` (departamento) y `rank` (posición), como se muestra en la figura 1.1(b). Cada fila de dicha tabla proporciona la identificación del profesor, el apellido, departamento y posición de un miembro del personal académico. La información de clase que se mantiene para cada clase impartida incluye el número de clase, la ID del profesor en la facultad, el

horario y el salón, en columnas apropiadas como se muestra en la tabla `Class` (Clase) en la figura 1.1(c). Estas tres tablas solas no permiten determinar cuáles clases toma un estudiante. Para representar el hecho de que un estudiante está inscrito en una clase particular, se necesita otra tabla, que se llama `Enroll` (inscripción), que se presenta en la figura 1.1(d). Las columnas de `Enroll` son `stuId`, `classNumber` (número de clase) y `grade` (calificación). Note que la tabla `Enroll` representa la **relación** entre `Student` y `Class`, lo que indica cuáles filas de estas tablas se relacionan (es decir, cuáles estudiantes toman cuáles clases). Por ejemplo, la primer fila, con valores S1001, ART103A, dice que el estudiante cuya ID es S1001 está inscrito en la clase cuyo número de clase es ART103A. La última columna de la fila menciona la calificación que obtuvo cada estudiante en cada clase. Dado que esto representa inscripciones actuales, se supondrá que la calificación es la calificación de medio semestre. Al final del semestre, se puede cambiar a la calificación final. Note que en esta tabla no se pusieron ni los nombres de estudiante ni las ID, porque ya están en la tabla `Student`, y se quiere evitar la redundancia y posibles inconsistencias que

FIGURA 1.1(a)
La tabla Student

Student

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	Historia	90
S1002	Chin	Ann	Mat	36
S1005	Lee	Perry	Historia	3
S1010	Burns	Edward	Arte	63
S1013	McCarthy	Owen	Mat	0
S1015	Jones	Mary	Mat	42
S1020	Rivera	Jane	CSC	15

FIGURA 1.1(b)
La tabla Faculty

Faculty

facId	name	department	rank
F101	Adams	Arte	Profesor
F105	Tanaka	CSC	Instructor
F110	Byrne	Mat	Asistente
F115	Smith	Historia	Asociado
F221	Smith	CSC	Profesor

FIGURA 1.1(c)
La tabla Class

Class

classNumber	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
CSC203A	F105	MThF12	M110
HST205A	F115	MWF11	H221
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

Enroll		
stuId	**classNumber**	**grade**
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

FIGURA 1.1(d)
La tabla Enroll

Query1	
lastName	**firstName**
Smith	Tom
Chin	Ann
Burns	Edward

FIGURA 1.2
Resultados de la consulta: "Encontrar nombres de todos los estudiantes inscritos en ART103A"

causaría el almacenar los nombres dos veces. Las tablas que se muestran se crearon con Access, y se puede usar Access para actualizarlas, para plantear preguntas (queries, consultas) acerca de los datos en ellas, para crear reportes acerca de los datos y para hacer muchas otras funciones. Como ejemplo de una consulta, suponga que se quieren los nombres de todos los estudiantes inscritos en ART103A. Primero, ¿cómo encuentra visualmente la respuesta a la pregunta? Al buscar en la base de datos, se ve que la tabla `Enroll` dice cuáles estudiantes están inscritos en ART103A. Sin embargo, da el `stuId` de cada estudiante, no el nombre. Los nombres de los estudiantes aparecen en la tabla `Student`. Un plan para responder la pregunta es buscar en la tabla `Enroll` y encontrar todas las filas donde el valor de `classNumber` sea ART103 y tomar nota de los valores `stuId` en dichas filas, a saber, S1001, S1002 y S1010. Luego se busca en la tabla `Student` y se encuentran las filas que contengan dichos valores en la columna `stuId`. La respuesta a la pregunta se encuentra al citar los valores `lastName` y `firstName` en dichas filas, lo que produce Smith Tom, Chin Ann y Burns Edward. Access proporciona una **herramienta de consulta** (query tool) que permite comprobar cuáles columnas se incluyen en una consulta y especificar condiciones para los registros en los resultados. La figura 1.2 muestra los resultados de ejecutar la consulta precedente usando esta herramienta. También se puede usar la **herramienta reporte** (reporting tool) en Access para generar una variedad de reportes. La figura 1.3 muestra un reporte común llamado Class Lists (listas de clase) que muestra cada número de clase, la ID y el nombre del miembro docente que imparte la clase, y las ID y nombres de todos los estudiantes en dicha clase.

1.3 El entorno de base de datos integrada

Un **entorno de base de datos integrada** tiene un único gran repositorio de datos, llamado base de datos, que usan de manera simultánea muchos departamentos y usuarios en una

FIGURA 1.3
Reporte Class Lists

Class Lists

classNumber	facId	name	stuId	lastName	firstName
ART103A	F101	Adams	S1001	Smith	Tom
			S1002	Chin	Ann
			S1010	Burns	Edward
CSC201A	F105	Tanaka	S1002	Chin	Ann
			S1020	Rivera	Jane
HST205A	F115	Smith	S1001	Smith	Tom
MTH101B	F110	Byrne	S1020	Rivera	Jane
MTH103C	F110	Byrne	S1002	Chin	Ann
			S1010	Burns	Edward

organización. Todos los datos que la organización necesita para un grupo específico de aplicaciones, o incluso para todas sus aplicaciones, se almacenan juntos, con tan poca repetición como sea posible. (Nota: Aunque la palabra *data* (*datos*) es plural en el inglés usual, es obligatorio usarla como singular y plural en la literatura de bases de datos, como en "datos es" y "datos son".) En la base de datos pueden aparecer diferentes tipos de registros. Las conexiones lógicas entre los ítems y registros de datos también se almacenan en la base de datos, de modo que el sistema "sabe", por ejemplo, cuál registro de docente está conectado a un registro de clase particular. La base de datos no es propiedad de un solo departamento, sino que es un recurso compartido. En una organización grande, la base de datos la gestiona un **administrador de base de datos** (ABD), quien es el responsable de crear y mantener la base de datos para satisfacer las necesidades de los usuarios. Todos los accesos a la base de datos están controlados mediante un sofisticado paquete de software llamado **sistema de gestión de base de datos** (DBMS, por sus siglas en inglés). Este paquete tiene programas que establecen las estructuras de almacenamiento originales, cargan los datos, aceptan peticiones de datos de programas y usuarios, dan formato a los datos recuperados de modo que aparezcan en la forma que el programa o el usuario esperan, ocultan datos a los que un usuario particular no debe tener acceso, aceptan y realizan actualizaciones, permiten el uso concurrente de los datos sin hacer que los usuarios interfieran unos con otros, y realizan respaldos y procedimientos de recuperación automáticamente. Éstas son sólo algunas de las muchas funciones del sistema de gestión de la base de datos.

La figura 1.4 ilustra un entorno de base de datos integrada. Aquí, todos los datos acerca de estudiantes, clases, personal docente e inscripciones se almacenan en una sola base de datos. Los datos están integrados, de modo que los ítems de datos se almacenan en formatos compatibles y las conexiones lógicas entre ellos también se almacenan. La base de datos contiene una descripción de su propia estructura, de modo que el DBMS "sabe" cuáles ítems de datos existen y cómo están estructurados o agrupados. La comparten muchos usuarios, por lo general de manera concurrente. Todo acceso a los datos es a través del DBMS. Los programas de aplicaciones, que se pueden escribir en diferentes lenguajes de programación, pasan por el DBMS, que puede presentar los datos en la forma que cada programa espera. Sólo el DBMS está al tanto de las estructuras de almacenamiento utilizadas en la base de datos. Además de proporcionar apoyo para las aplicaciones, el DBMS proporciona una interfaz de usuario para consultas interactivas. Los usuarios autorizados pueden preguntar a la base de datos directamente, con el lenguaje de consulta del DBMS particular.

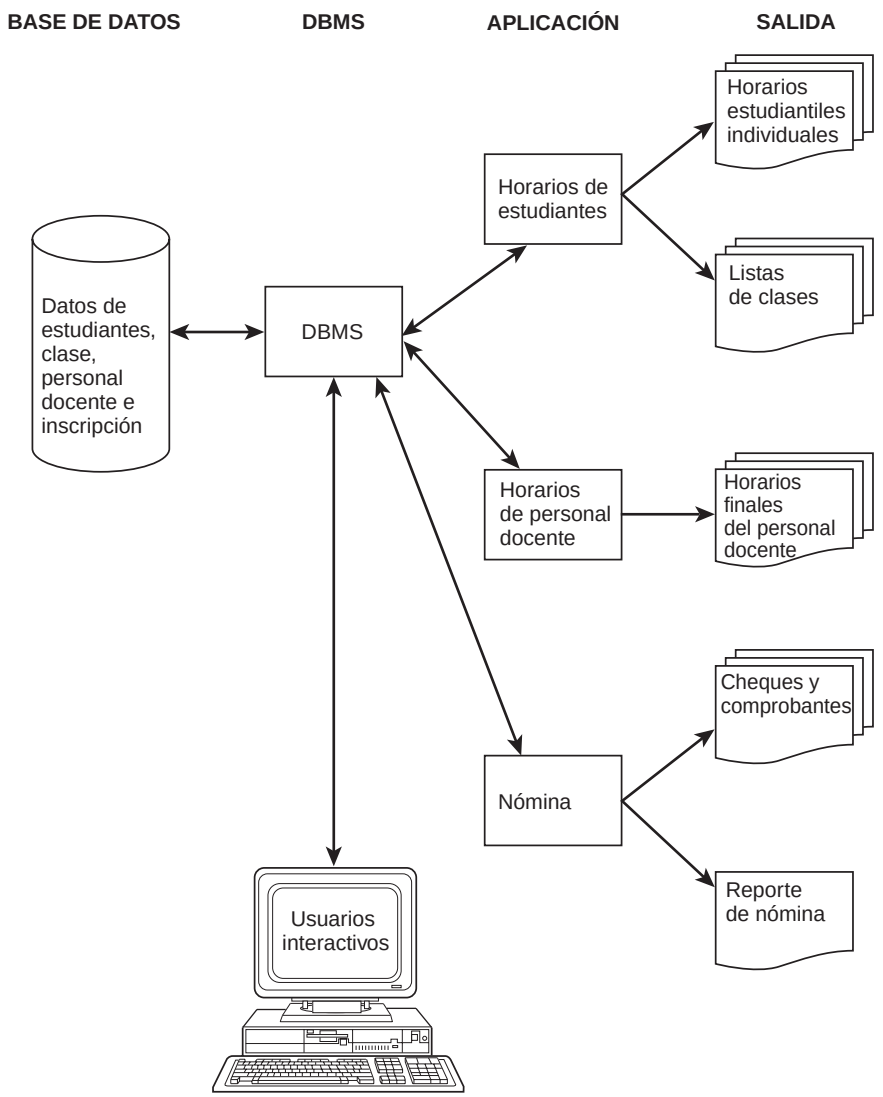


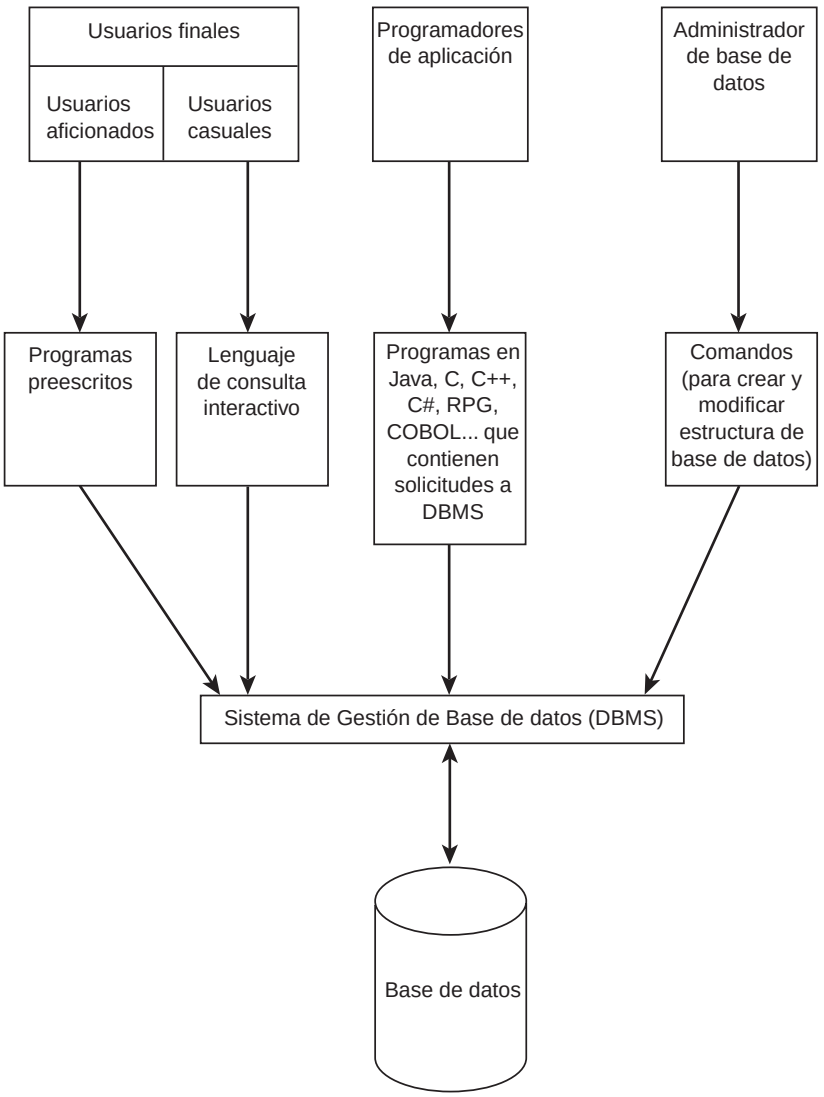
FIGURA 1.4
El entorno de base de datos integrada

1.4 Roles en el entorno de base de datos integrada

Muchos individuos o grupos están involucrados en las operaciones de un sistema de base de datos. Juegan diferentes roles, dependiendo de la forma en que interactúan con la base de datos, como se muestra en la figura 1.5.

- Usuarios finales
  La base de datos se diseña, crea y mantiene para satisfacer las necesidades de información de los **usuarios finales**, las personas que usan los datos para realizar sus labores. Sin importar la elegancia del diseño de la base de datos, o la sofisticación del hardware y el software utilizados, si la base de datos no proporciona información adecuada a los usuarios, es un fracaso. A final de cuentas, son los usuarios quienes juzgan el éxito del sistema. Los usuarios se pueden categorizar de acuerdo con la forma en que acceden a los datos. Los usuarios sofisticados (también llamados **usuarios casuales**) están capacitados en el uso del lenguaje de consulta interactivo, y acceden a los datos mediante el ingreso de consultas en estaciones de trabajo. La flexibilidad del lenguaje de consulta les permite realizar muchas operaciones diferentes

FIGURA 1.5
Roles en el entorno de base de datos



sobre la base de datos, sólo limitadas por las vistas que se les asigne y sus autorizaciones. Los usuarios casuales pueden realizar operaciones de recuperación, inserción, borrado o actualización mediante el lenguaje de consulta, siempre que tengan autorización para hacerlo. Los **usuarios aficionados** no usan el lenguaje de consulta interactivo, sino que acceden a los datos mediante programas de aplicación que se escribieron para ellos. Invocan los programas al ingresar comandos simples o elegir opciones de un menú. No necesitan conocer detalle alguno de la estructura o lenguaje del sistema de base de datos. Interactúan con el sistema en una forma menos sofisticada y restringen su acceso a operaciones realizadas por los programas. Los programas mismos pueden realizar operaciones de actualización o recuperación. Un grupo todavía más grande de **usuarios secundarios** puede usar la información en la base de datos sin interactuar directamente con ella, al recibir salida que usan en sus labores.

Por ejemplo, en una oficina de admisión en una universidad, los empleados pueden ser usuarios aficionados, mientras que el encargado de admisión puede ser un usuario casual. Los empleados realizan simples tareas repetitivas como imprimir transcripciones estudiantiles. Pueden ingresar el nombre de la transacción, TRANSCRIP-

CIÓN, o elegir una opción como IMPRIMIR TRANSCRIPCIÓN de un menú. El programa TRANSCRIBIR pediría al empleado la ID del estudiante u otra información de identificación, y completaría su tarea sin mayores instrucciones del empleado. El encargado de admisión usa el lenguaje de consulta para plantear preguntas específicas como "¿cuántos estudiantes están registrados en seis o más clases este semestre?" Si no hay un programa preescrito en el DBMS para responder esta pregunta, el encargado escribe enunciados en el lenguaje de consulta de dicha base de datos particular. Los estudiantes que reciben transcripciones impresas, y los profesores que reciben listas de nombres para la clase, son usuarios secundarios.

- Programadores de aplicaciones
  Este grupo incluye a los programadores que escriben aplicaciones batch, o interactivas, para otros usuarios. Sus programas de aplicación se pueden escribir en una variedad de lenguajes de programación huéspedes como Java, C, C++, C#, Visual BASIC, RPG o COBOL. Cada programa que accede a la base de datos contiene enunciados que solicitan al sistema de gestión de la base de datos realizar actualizaciones o recuperaciones en la base de datos. Algunos usuarios finales sofisticados que tienen tanto el conocimiento del lenguaje de programación, como el permiso para hacerlo, son capaces de escribir aplicaciones para su propio uso.
- Administrador de la base de datos
  El administrador de la base de datos es el individuo o grupo responsable del diseño, creación de la estructura y mantenimiento de la base de datos. En muchos casos, la base de datos la diseña un especialista, y el ABD toma la responsabilidad una vez que el diseño está completo. El diseñador de la base de datos comienza el proceso de diseño al entrevistar a los usuarios para determinar sus necesidades de datos. Examina el sistema actual, analiza la organización y sus necesidades de información, y desarrolla un modelo tentativo para la base de datos. El modelo se refina y mejora conforme el diseñador, al consultar a los usuarios, está más al tanto de sus necesidades de datos y aprende más acerca del funcionamiento de la organización. Cuando se desarrolla un diseño satisfactorio, el ABD lo implementa. Una vez más, se consulta a los usuarios para determinar si el sistema operativo es adecuado. El diseño, refinamiento y rediseño del sistema son esfuerzos de equipo, con el diseñador, el ABD y los usuarios trabajando en conjunto para desarrollar la mejor fuente de datos para toda la organización. El ABD interactúa con la base de datos operativa como un "superusuario", quien controla y accede a la información acerca de la estructura y el uso de la base de datos en sí, en oposición a los usuarios finales, quienes acceden a los datos dentro de la base de datos. El capítulo 2 contiene una descripción más detallada de las funciones del administrador de la base de datos.

1.5 Ventajas del enfoque de base de datos integrada

Antes de crearse las bases de datos integrada, se usaban sistemas de **procesamiento de archivos**, y los datos que usaban los programas de aplicación de una organización se almacenaban en archivos separados. Por lo general, un departamento que necesitaba un programa de aplicación trabajaba con el departamento de procesamiento de datos de la organización para crear especificaciones tanto para el programa como para los datos necesarios para él. Con frecuencia los mismos datos se recopilaban y almacenaban de manera independiente mediante varios departamentos dentro de una organización, pero no se compartían. Cada aplicación tenía sus propios archivos de datos que se creaban de manera específica para la aplicación, y éstos pertenecían al departamento para el que se escribía la aplicación. El enfoque de base de datos integrada tiene muchas ventajas:

1. Compartición de datos
   La base de datos pertenece a toda la organización. El ABD gestiona los datos, pero éstos no pertenecen a algún individuo o departamento. Por ende, la organización tiene el control sobre los datos que necesita para dirigir su negocio. Muchos usuarios pueden tener autorización para acceder al mismo trozo de información. La autorización para acceder a los datos la otorga el ABD, no otro departamento.

2. Control de redundancia
   Cuando se almacena en una base de datos, la información se integra de modo que múltiples copias de los mismos datos no se almacenan a menos que sea necesario. Se permite alguna redundancia limitada para mantener las conexiones lógicas entre los ítems de datos o para mejorar el rendimiento. Para poner un caso, en el ejemplo de universidad que se discutió en la sección 1.2, la ID del estudiante aparecía tanto en la tabla Student como en la tabla Enroll. El sistema de gestión de la base de datos "sabe acerca" de dicha repetición. Una base de datos de ordinario no tiene múltiples copias de registros enteros, a diferencia de un sistema de archivos, donde distintos departamentos podían tener duplicados de archivos enteros.

3. Consistencia de datos
   Un efecto de eliminar o controlar la redundancia es que los datos son consistentes. Si un ítem de datos aparece sólo una vez, cualquier actualización a su valor necesita realizarse sólo una vez, y todos los usuarios tendrán acceso al mismo nuevo valor. Si el sistema tiene cierta redundancia controlada, cuando recibe una actualización a un ítem que aparezca más de una vez con frecuencia puede realizar actualizaciones en cascada. Esto significa que automáticamente actualizará cada ocurrencia de dicho ítem, lo que mantiene consistente a la base de datos. Por ejemplo, si se cambia la ID de un estudiante en la tabla Student, los registros Enroll para dicho estudiante se actualizarán para mostrar la nueva ID en forma automática.

4. Estándares de datos mejorados
   El ABD, que es responsable del diseño y mantenimiento de la base de datos para satisfacer las necesidades de todos los usuarios, define y refuerza los estándares de toda la organización para la representación de datos en la base de datos. En esta categoría se incluyen reglas como el formato de todos los ítems de datos, convenciones acerca de nombres de datos, estándares de documentación, frecuencia de actualizaciones, procedimientos de actualización, frecuencia de respaldos, procedimientos de respaldos y uso permitido de la base de datos. Por ejemplo, el ABD puede elaborar una regla para que las direcciones se almacenen en un formato particular. En Estados Unidos, una convención puede ser que, para los nombres de los estados, se usen abreviaturas con dos letras. La base de datos se puede configurar de modo que cualquier otra representación se rechace. En otros países, las zonas postales pueden definirse con base en cierto número de caracteres.

5. Mejor seguridad de datos
   Los datos en la base de datos de una organización son un valioso recurso corporativo que se debe proteger de mal uso intencional o accidental. La seguridad de datos es la protección de la base de datos de acceso no autorizado por personas o programas que puedan hacer mal uso o dañar los datos. Un sistema de base de datos permite la definición y fortalecimiento de restricciones de seguridad en varios niveles. Todo acceso autorizado a la base de datos es a través del DBMS, que puede requerir que los usuarios pasen a través de procedimientos de seguridad o usar contraseñas para obtener acceso a la base de datos. Para eliminar la posibilidad de que un usuario pase por un lado del DBMS y obtenga acceso a los datos en forma ilegal, el DBMS puede encriptar los datos antes de almacenarlos. Entonces, cuando un usuario autorizado

desee recuperar datos, se desencriptarán automáticamente. Los datos recuperados en cualquiera otra forma aparecerán en su forma encriptada. Los usuarios autorizados pueden no estar al tanto de la encriptación de datos. A cada usuario se le proporciona una vista de una porción predefinida de la base de datos. Por ejemplo, en una universidad, la oficina de admisión puede tener acceso a cierta información del personal docente, como la tabla Faculty en el ejemplo anterior, pero no a ítems como el salario. En la vista se incluyen descripciones de los ítems de datos a los que se permite el acceso del usuario, y el tipo de acceso permitido, ya sea sólo recuperación, actualización o borrado de registros existentes, o inserción de nuevos registros. Si un usuario intenta acceder a un ítem que no está incluido en su vista, o intenta una operación no autorizada, el DBMS automáticamente registra la ID del usuario en una bitácora (log) de seguridad que está disponible al ABD.

6. Integridad de datos mejorada
   Algunos sistemas de gestión de base de datos permiten al ABD definir restricciones de integridad: reglas de consistencia que la base de datos debe obedecer. Estas restricciones se aplican a ítems dentro de un registro (restricciones intrarregistro) o a registros que se relacionan mutuamente (restricciones interregistro), o pueden ser restricciones generales del negocio. Por ejemplo, en los registros de clase, puede haber una regla de que el número de estudiantes inscritos en una clase nunca supere algún máximo de inscripción permitido. Otra regla puede ser que la ID del personal docente en un registro de clase deba corresponder a una ID de personal docente real en un registro de personal docente. El DBMS es responsable de nunca permitir la inserción, el borrado o la actualización de un registro que viole una restricción de integridad.
7. Equilibrio de los requisitos en conflicto
   Cada departamento o usuario individual tiene necesidades de datos que pueden estar en conflicto con los de otros usuarios. El ABD está al tanto de las necesidades de todos los usuarios y puede tomar decisiones acerca del diseño, uso y mantenimiento de la base de datos que proporcionen las mejores soluciones para la organización como un todo. Estas decisiones por lo general favorecen las aplicaciones más importantes, posiblemente a costa de las menos vitales.
8. Desarrollo más rápido de nuevas aplicaciones
   Una base de datos bien diseñada proporciona un modelo preciso de las operaciones de la organización. Cuando se propone una nueva aplicación, es probable que los datos requeridos ya estén almacenados en la base de datos. Si es así, el DBMS puede proporcionar datos en la forma requerida por el programa. El tiempo de desarrollo se reduce porque no se necesita una fase de creación de archivos para la nueva aplicación, como ocurría cuando se usaban los sistemas de procesamiento de archivos.
9. Mejor accesibilidad de datos
   Además de proporcionar datos para los programas, la mayoría de los sistemas de gestión de base de datos permiten acceso interactivo a los usuarios. Proporcionan lenguajes de consulta que permiten a los usuarios plantear preguntas *ad hoc* y obtener la información deseada.
10. Economía de escala
    Cuando todos los requisitos de datos de la organización se satisfacen mediante una base de datos en lugar de muchos archivos separados, el tamaño de la operación combinada proporciona muchas ventajas. La porción del presupuesto que de ordinario se asignaría a varios departamentos para sus costos de diseño, almacenamiento y costos de datos, se puede combinar, lo que posiblemente resulte en un costo total más bajo. Los recursos combinados se pueden usar para desarrollar un sistema más

sofisticado y poderoso que cualquier departamento podría costear en forma individual, lo que proporciona características no disponibles en un entorno de procesamiento de archivos. El tiempo de programador que ordinariamente se dedicaría al diseño de archivos y la escritura de programas para acceder a ellos se puede emplear en la mejora de la base de datos. Cualquier mejora a la base de datos beneficia a muchos usuarios.

11. Más control sobre la concurrencia

    Si a dos usuarios se les permite ingresar a datos simultáneamente, y al menos uno de ellos actualiza datos, es posible que interfieran uno con el otro. Por ejemplo, si ambos intentan realizar actualizaciones, una actualización se puede perder, porque el segundo puede sobrescribir el valor grabado por el primero. Si las actualizaciones tienen la intención de ser acumulativas, éste es un serio problema. La mayoría de los sistemas de gestión de bases de datos integradas tienen subsistemas para controlar concurrencia, de modo que las transacciones no se pierdan o desempeñen de manera incorrecta.

12. Mejores procedimientos de respaldo y recuperación

    En un entorno de base de datos, los registros de la base de datos por lo general se respaldan (copian) de manera regular, acaso por la noche. Para mantener seguro el respaldo, se usa una cinta u otro medio. Conforme se realizan transacciones, cualquier actualización se registra en una bitácora (log) de cambios. Si el sistema fracasa, cinta y log se usan para llevar la base de datos al estado en que estaba justo antes de la falla. Por tanto, el sistema se autorrecupera.

1.6 Desventajas del enfoque de base de datos integrada

También existen algunas desventajas en un entorno de base de datos integrada, en comparación con un sistema de archivos:

1. Alto costo de DBMS

   Puesto que un sistema de gestión de base de datos completo es una pieza de software muy grande y sofisticada, su compra o arrendamiento es costoso.

2. Costos de hardware más altos

   Para correr el DBMS se requieren memoria adicional y potencia de procesamiento, lo que resulta en la necesidad de actualizar el hardware.

3. Costos de programación más altos

   Puesto que un DBMS es una herramienta compleja con muchas características, los programadores de la organización necesitan un conocimiento extenso del sistema con la finalidad de usarlo con mayor ventaja. Si la organización contrata programadores de base de datos experimentados o capacita a su propio personal de programación, paga por la experiencia.

4. Altos costos de conversión

   Cuando una organización convierte a un nuevo sistema de base de datos, se tienen que remover datos de los archivos existentes y cargarlos en la base de datos. Debido a los diferentes formatos usados en los archivos, éste puede ser un proceso difícil y que consume tiempo. Además, los programas de aplicaciones, que contienen detalles acerca del almacenamiento y la estructura de los archivos antiguos, se deben modificar para trabajar con el DBMS.

5. Procesamiento más lento de algunas aplicaciones

   Aunque una base de datos integrada se diseña para proporcionar mejor información más rápidamente que un sistema tradicional que use archivos separados, algunas aplicaciones son más lentas. Por ejemplo, un archivo de nómina típico se configura en una secuencia que coincide con el programa de nómina, y contiene sólo los datos necesarios para esta aplicación. Está diseñado específicamente para hacer a dicha aplicación tan eficiente como sea posible. En la base de datos, los registros de los empleados pueden no almacenarse de manera consecutiva y la recuperación normal puede no ser en la secuencia necesaria por el programa de nómina. Por tanto, este programa tardará más tiempo en ejecutarse.

6. Vulnerabilidad aumentada

   Siempre que los recursos están centralizados, existe un aumento en el riesgo de seguridad. Dado que todas las aplicaciones dependen del sistema de base de datos, la falla de cualquier componente del sistema puede llevar las operaciones a un estancamiento. La falla de un solo programa de aplicaciones puede tener un efecto sobre otros programas que es posible usen datos incorrectos creados por el programa dañado.

7. Recuperación más difícil

   El proceso de recuperación después de una falla de la base de datos es complicado porque muchas transacciones podrían estar en progreso cuando falle el sistema. Como parte de su recuperación, el sistema debe determinar cuáles transacciones se completaron y cuáles todavía estaban en progreso al momento de la falla. Si la base de datos está dañada, se puede recuperar con el uso de la cinta de recuperación y el log. El hecho de que una base de datos permita a los usuarios realizar actualizaciones concurrentes complica aún más el proceso de recuperación.

1.7 Desarrollos históricos en los sistemas de información

La necesidad de registrar datos se remonta a la historia reconocida más antigua. Los intentos para proporcionar registros permanentes de transacciones se ven en las tablas de arcilla sumerias, en artefactos dejados por los babilonios, en los jeroglíficos del antiguo Egipto e incluso en las pinturas rupestres. Los registros en papel u otras formas escritas se han usado durante siglos para registrar información acerca de historias familiares, tratados y otros acuerdos, inventarios domésticos o empresariales, reclutamiento escolar, registros de empleados, pago de bienes o servicios, datos censales y muchas otras facetas de la vida.

El uso de **tarjetas perforadas** para almacenamiento de datos se introdujo en 1890, cuando los datos censales estadounidenses se recopilaron y almacenaron en tarjetas perforadas por primera vez. La constitución estadounidense requiere que se realice un censo completo cada 10 años. El censo de 1880 tardó siete años en completarse porque la población del país aumentó tanto que se anticipó que no habría suficiente tiempo para completar el censo antes de 1900, cuando comenzaría uno nuevo. La oficina censal patrocinó una competencia a fin de que se proporcionaran ideas acerca de las formas para hacer el censo más eficiente. Herman Hollerith, empleado de la oficina, propuso el uso de tarjetas perforadas para registrar las respuestas censales de cada hogar y facilitar el procesamiento de las respuestas. Tales tarjetas ya estaban en uso en la industria de tejido de seda en Lyon, Francia, para controlar el telar Jacquard, que tejía patrones en tela de seda. Hollerith diseñó un método con el fin de usar la misma tecnología para almacenar datos censales y examinar sus patrones. Ganó la competencia y, debido a su diseño, el censo se completó en tiempo récord, y se inventó una nueva técnica para el procesamiento de datos. Después de dicho éxito, el equipo mecánico de tarjetas perforadas se usó durante muchos años para almacenamiento, ordenación,

análisis y reporte de datos, y las tarjetas perforadas funcionaron como un medio de entrada para las computadoras, tanto para programas como para datos.

La **cinta de papel perforado** se usó para almacenar tanto programas de computadora como datos desde comienzos de la década de 1940, cuando se inventaron las primeras computadoras electromecánicas y electrónicas. Desde aproximadamente 1950, la **cinta magnética** se desarrolló y usó para entrada en las primeras computadoras, incluida la UNIVAC 1, la primera computadora comercialmente disponible. Los mazos de tarjetas perforadas, los bucles de cinta de papel perforada o los carretes de cinta magnética se usaron todos esencialmente en la misma forma, tanto para almacenar programas como para proporcionar un método de almacenamiento y entrada de datos. Los datos en estos medios sólo se podían leer en el orden en el que se almacenaban. Este tipo de **procesamiento secuencial de archivos** era en extremo eficiente, pero no muy flexible. Por lo general, la nómina era la primera aplicación que un negocio elegía automatizar, debido a los complejos cálculos y requerimientos de reporte que eran tediosos para las personas que los realizaban.

La figura 1.6 proporciona un panorama de una aplicación de nómina que usa procesamiento secuencial de archivo. El **archivo maestro** que contiene datos de nómina relativamente permanentes para cada empleado se mantiene en orden mediante un campo clave, acaso Número de Empleado. Los registros en este archivo también pueden contener ítems como el nombre del empleado, dirección, salario semanal, exenciones, deducciones de impuestos, totales de pago bruto en lo que va del año, impuestos y salario neto. Cada semana se prepararía un **archivo de transacción** que contendría nuevos datos, como el número de horas laboradas dicha semana, cualquier cambio en salario, deducciones u otros datos, y cualquiera otra nueva información necesaria para la nómina de dicha semana. Con frecuencia se usaba la cinta magnética para el archivo maestro, y las tarjetas perforadas para el archivo de transacción. Ambos archivos tienen que estar en el mismo orden, por Número de Empleado. Un programa leería un registro maestro, luego leería el registro de transacción para el mismo empleado y completaría el procesamiento de nómina para dicha persona. En el proceso, la información en el antiguo registro maestro cambiaría para reflejar nuevos datos, y un nuevo registro se escribiría a una nueva cinta maestra. Al final del programa, la nueva cinta se convertiría en la cinta maestra actual, y se usaría la siguiente semana. A esto se le conoce como un sistema **maestro antiguo/maestro nuevo** o **padre/hijo**. El tipo de procesamiento descrito aquí, donde un conjunto de registros se envían como unidad a un programa que luego opera sobre ellos sin mayor intervención humana, se conoce como **procesamiento por lote (batch)**.

El almacenamiento en **disco magnético** estuvo disponible hacia finales de la década de 1950, lo que hizo posible el **acceso directo** (acceso no secuencial) de registros. Los programas ya no requerían que el orden de acceso coincidiera con el orden físico de los registros. Las actualizaciones se podían hacer al disco, sin rescribir todo el archivo. Durante la década de 1960 se desarrollaron lenguajes de programación, incluidos COBOL y PL/1, para procesamiento de datos comercial que usaba datos almacenados tanto en cinta como en disco. Originalmente, se usaron organizaciones de archivo simples para organizar datos en estos dispositivos de almacenamiento secundario, pero conforme las aplicaciones se volvieron más complejas, se necesitaron métodos de almacenamiento y recuperación de datos más sofisticados. Dos modelos de base de datos competitivas, la red y la jerárquica, se desarrollaron en esta época. Sin embargo, continuó el uso de los sistemas de archivos para muchas aplicaciones.

El **modelo jerárquico** para bases de datos se desarrolló durante la década de 1960, como una solución *ad hoc* para las necesidades inmediatas de aplicaciones reales. El sistema de gestión de base de datos jerárquico más antiguo, el IMS de IBM, se creó con el fin de organizar y almacenar información necesaria para el programa espacial del proyecto de aluniza-

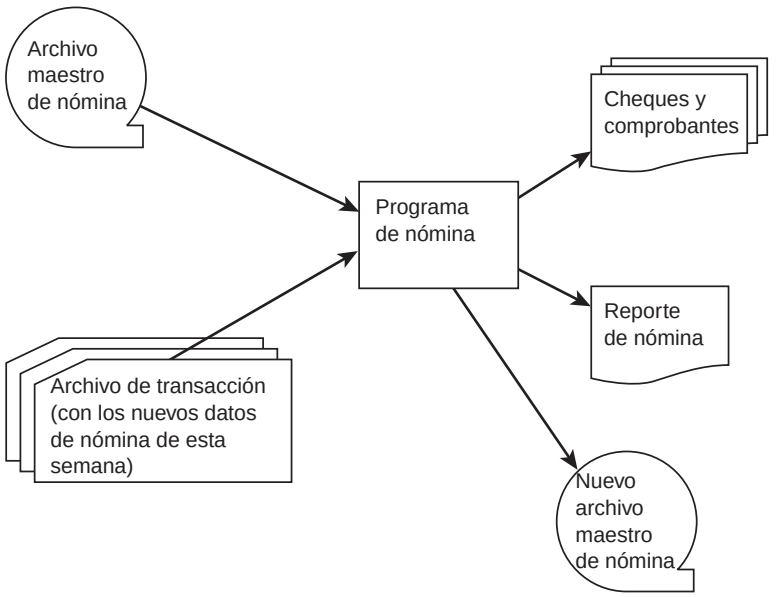


FIGURA 1.6
Roles en el entorno de base de datos

je del Apolo. North American Aviation (que se convirtió en Rockwell) e IBM trabajaron en conjunto para producir la primera versión de IMS, que se lanzó en 1968. Las primeras versiones de IMS se diseñaron para utilizarse con dispositivos de cinta magnética, pero más tarde el disco magnético se convirtió en estándar. IMS pronto se volvió el sistema de gestión de base de datos jerárquico dominante en el mercado y durante muchos años fue el más ampliamente usado de todos los DBMS, hasta que lo sustituyeron los sistemas relacionales. Después de 1968 se hicieron muchas mejoras a IMS, lo que resultó en nuevas versiones que sacaron ventaja de las mejoras en hardware y software, y proporcionó nuevas características como comunicaciones de datos y maximizar el rendimiento. El sistema de reservaciones de la aerolínea SABRE se basó en IMS. IMS se conoció como un "caballo de batalla" capaz de procesar eficientemente grandes cantidades de datos. Usaba una estructura de árbol familiar a los programadores que estaban acostumbrados a trabajar con archivos, y proporcionó rendimiento predecible.

A principios de la década de 1960, Charles Bachman, en General Electric, creó uno de los sistemas de gestión de base de datos más antiguo, Integrated Data Store (IDS, almacén datos), usando un **modelo de red**. Este sistema de gestión de base de datos influyó el desarrollo del área de base de datos durante muchos años. La Conferencia sobre Lenguajes de Sistemas de Datos (**CODASYL**, por sus siglas en inglés), organización que consiste en representantes de grandes proveedores y usuarios de hardware y software, se formó para intentar estandarizar muchos aspectos del procesamiento de datos. Tuvo estándares exitosamente escritos para el lenguaje COBOL. A finales de la década de 1960 formó un subgrupo llamado Database Task Group (**DBTG**, Grupo de tarea de base de datos) para abordar la cuestión de la estandarización para los sistemas de gestión de base de datos. Influidos por IDS, el grupo propuso un modelo basado en red y especificaciones para definición de datos y lenguajes de manipulación de datos. El reporte del anteproyecto se publicó en 1969 y resultó en muchas sugerencias para cambios por parte de sus lectores. El DBTG reconsideró su propósito y publicó su primer reporte oficial en 1971. Este documento hito se envió al American National Standards Institute (ANSI) con la esperanza de que sus especificaciones se aceptarían como un estándar para los sistemas de gestión de base de datos. Sin embargo, ANSI se rehusó a aceptar o rechazó el estándar propuesto. Al reporte de 1971 siguieron muchas versiones más nuevas, notablemente en 1973, 1978, 1981 y 1984, pero siguió siendo

el principal documento que describía un modelo basado en red al que generalmente se le conoce como el modelo CODASYL o el modelo DBTG, y muchos sistemas de gestión de base de datos populares se basaron en él. Además, proporcionó el vocabulario y marco conceptual para discutir los temas de base de datos, y estableció por primera vez la noción de una arquitectura de base de datos en capas y terminología común. El DBTG evolucionó en 1972 a un comité CODASYL permanente, el DDLC, o Data Description Language Committee (Comité de Lenguaje de Descripción de Datos), que continuó operando y publicaba sus descubrimientos periódicamente en su *Journals of Development* hasta 1984, cuando el comité de estandarización ANSI X3H2 se apoderó de su función. A pesar del hecho de que DBTG y DDLC continuaron realizando cambios al modelo CODASYL, la proposición 1971 la usaron grandes proveedores como la base para sus sistemas de gestión de bases de datos. El más ampliamente usado de estos sistemas basados en red fue el IDMS de Cullinet. Otros incluyeron PRIME DBMS de PRIME Computer, IDS II de Honeywell, DMS 170 de Control Data Corporation, DC, DMSII y DMS1100 de UNISYS, y DBMS 10 y DBMS 11 de Digital Equipment Corporation.

Aunque los modelos jerárquico y de red eran poderosos y eficientes, eran complejos, y requerían que los usuarios entendieran las estructuras de datos y las rutas de acceso a los datos. Estaban diseñados para usarse con programas en lugar de para acceso interactivo de los usuarios, de modo que no soportaban consultas *ad hoc*. No estaban basados sobre un fundamento teórico sólido, sino que eran soluciones construidas sobre sistemas de archivos existentes.

El **modelo relacional** fue propuesto por primera vez por E. F. Codd en 1970, en un artículo llamado "A Relational Model of Data for Large Shared Data Banks". Fue el primer modelo que se basó en nociones teóricas de matemáticas, que proporcionó una fuerte base teórica. La investigación sobre el modelo la realizaron Codd y otros en el IBM Research Laboratory en San José, California. **System R**, un prototipo de sistema de gestión de base de datos relacional, lo desarrollaron investigadores de IBM a finales de la década de 1970. DB2, el sistema de gestión de base de datos relacional de IBM, se basó en System R. **SQL**, un lenguaje desarrollado por System R, se convirtió en el lenguaje de datos estándar para las bases de datos relacionales, con estándares aprobados por ANSI publicados en 1986, 1989, 1992 y 1999. Otros tempranos proyectos de investigación de modelo relacional fueron el **Peterlee Relational Test Vehicle**, creado en el IBM UK Scientific Laboratory e **INGRES**, desarrollado en la Universidad de California en Berkeley. La investigación condujo a una versión "universitaria" de INGRES, así como a un producto comercial. **ORACLE** se desarrolló y comercializó usando muchos de los resultados del System R. El extenso uso de microcomputadoras que comenzó en la década de 1980 condujo a la creación de sistemas de gestión de base de datos basados en PC, que eran todos relacionales. Entre los primeros sistemas de gestión de bases de datos relacionales basados en microcomputadora estaban **dBase, R:Base, Foxpro** y **Paradox**. Access de Microsoft, que usa el modelo relacional, ahora es el sistema de gestión de base de datos basado en microcomputadora de uso más extendido. **Oracle**, **DB2, Informix, Sybase** y el **Server SQL** de Microsoft, que usan el modelo relacional, en la actualidad son los sistemas de gestión de bases de datos empresariales más importantes.

El modelo relacional usa tablas simples para organizar datos. No permite a los diseñadores de bases de datos expresar algunas distinciones importantes cuando modelan una empresa. En 1976, P. P. Chen desarrolló un nuevo tipo de modelo, el modelo **entidad-relación**. Éste es un ejemplo de un **modelo semántico** que intenta capturar el significado de los datos que representa. El modelo entidad-relación en sí mismo se ha extendido muchas veces para hacerlo semánticamente más rico. Otros modelos semánticos para bases de datos se crearon para intentar capturar más del significado en los datos.

La necesidad de almacenar y manipular datos complejos que no es fácil modelar usando las simples tablas del modelo relacional, así como la aparición de lenguajes de programación usando el paradigma orientado a objeto, condujo al desarrollo de bases de datos **orientadas a objeto** en la década de 1990. Estas bases de datos se crearon para manipular los datos requeridos para aplicaciones avanzadas como sistemas de información geográfica (gis), multimedia, diseño y manufactura asistidos por computadora (CAD/CAM), y otros entornos complejos. Los sistemas de gestión de base de datos relacional como Oracle agregan algunas capacidades orientadas a objeto a sus productos, lo que resulta en bases de datos híbridas **objeto-relacional**.

Los **almacenes de datos** se desarrollaron en la década de 1990 para proporcionar un método de captura de datos consolidado a partir de muchas bases de datos. Un almacén de datos por lo general guarda datos históricos acerca de una organización, con el propósito de **minar datos**, un proceso de análisis estadístico de datos que permite a la organización descubrir las tendencias que puedan estar presentes en sus propios registros.

El amplio uso de Internet ha tenido un tremendo impacto sobre el desarrollo de las bases de datos. Internet conecta a los usuarios a una rica red de bases de datos en constante expansión, y proporciona acceso a bibliotecas digitales, recursos multimedia, recursos educativos y mucho más. Los sitios de comercio electrónico proporcionan acceso a bases de datos de información acerca de productos y servicios a clientes a lo largo del mundo. Los dispositivos computacionales inalámbricos y clientes pequeños como los PDA son otros desarrollos que permiten a los usuarios conectarse a recursos de base de datos en formas nuevas y flexibles.

1.8 Resumen del capítulo

Las bases de datos se usan en cientos de miles de organizaciones que van desde grandes agencias gubernamentales hasta pequeños negocios. El estudio de la teoría, diseño y gestión de bases de datos permite maximizar sus beneficios potenciales.

En una base de datos típica, los datos se almacenan en un formato que hace fácil acceder, ya sea para consultas individuales o grandes reportes. En un entorno de base de datos integrada, todos los datos se mantienen en un solo repositorio llamado base de datos, y se gestiona mediante el administrador de base de datos (ABD). Todo acceso a la base de datos es a través del sistema de gestión de base de datos (DBMS), un paquete de software que configura estructuras de almacenamiento, carga datos, proporciona acceso a programas y usuarios interactivos, formatea datos recuperados, oculta ciertos datos, realiza actualizaciones, controla concurrencia y efectúa respaldos y recuperación para la base de datos. Los datos en una base de datos están integrados, son autodescriptivos y se comparten de manera concurrente mediante muchos usuarios. El DBMS proporciona una interfaz de programa y una interfaz de usuario para consultas interactivas que se expresan en el lenguaje de consulta del DBMS particular. Las personas en el entorno de la base de datos integrada incluyen usuarios finales, programadores de aplicación y el ABD, quienes interactúan con la base de datos en diferentes formas.

Algunas de las ventajas del enfoque de base de datos integrada son compartición de datos, control de redundancia, consistencia de datos, mejora en estándares de datos, mejor seguridad de datos, integridad de datos mejorada, balance de requerimientos en conflicto, economía de escala, más control sobre la concurrencia y mejores procedimientos de respaldo y recuperación. Algunas de las desventajas del enfoque de base de datos integrada son los costos más altos de DBMS, hardware, programación y conversión, y el procesamiento más lento de algunas aplicaciones, aumento en vulnerabilidad y recuperación más difícil.

El desarrollo de los sistemas de información depende de los avances tecnológicos en hardware y software. A partir de las tarjetas perforadas, la tecnología de almacenamiento se movió hacia la cinta de papel, la cinta magnética, el disco magnético y dispositivos más recientes. El procesamiento secuencial de archivos, requerido por las cintas, se sustituyó por el procesamiento directo de archivos una vez que se inventaron los dispositivos de acceso directo, como los discos. El modelo de base de datos jerárquica se desarrolló a partir de tecnología de procesamiento de archivos, y el primer sistema de gestión de base de datos jerárquica, IMS, fue creado por IBM y Norh American Aviation a fin de manipular la gran cantidad de datos necesarios para el proyecto de alunizaje del Apolo. En General Electric, Charles Bachman creó IDS, con base en el modelo de red, y fue la inspiración para el propósito de estandarización CODASYL DBTG. El modelo relacional fue propuesto por E. F. Codd, y un prototipo llamado System R, fue desarrollado, junto con SQL, como el lenguaje de datos relacional estándar. La mayoría de las bases de datos actuales, en especial los basados en PC, usan el modelo relacional. El modelo entidad-relación lo desarrolló P. P. Chen como un modelo semántico, que captura más del significado de los datos que el modelo relacional. Los modelos orientados a objeto se crearon para permitir la representación de ítems de datos más complejos necesarios para aplicaciones de bases de datos avanzadas. Los sistemas híbridos objeto-relacional añaden algunas características de objeto a las bases de datos relacionales. Los almacenes de datos permiten la colección de datos a partir de muchas bases de datos, lo que proporciona una organización con un rico recurso de datos para minado de datos. El amplio uso de Internet y el crecimiento del comercio electrónico han hecho a las bases de datos más accesibles al público.

Ejercicios

1.1 Proporcione cuatro ejemplos de sistemas de bases de datos distintos a los mencionados en la sección 1.1.

1.2 Mencione cinco tareas realizadas por el DBMS.

1.3 Mencione tres funciones que pueda realizar con una base de datos que no pueda hacer con una hoja de cálculo.

1.4 Distinga entre una base de datos y un sistema de gestión de base de datos.

1.5 Mencione cinco ventajas de un sistema de base de datos y proporcione un ejemplo de cada una.

1.6 Mencione cinco desventajas de un sistema de base de datos y explique cada una.

1.7 Mencione tres responsabilidades del ABD.

1.8 Proporcione un ejemplo de un usuario final y describa una tarea típica que un usuario pueda realizar sobre una base de datos.

1.9 Explique qué se entiende por lenguaje de programación huésped.

1.10 Proporcione un ejemplo de una aplicación además de nómina que pueda usar procesamiento de lote secuencial y dibuje un diagrama similar al de la figura 1.6.

1.11 Defina brevemente cada uno de los siguientes términos que se usan en los sistemas de base de datos:

a. integración de datos

b. concurrencia

c. lenguaje de consulta

d. consistencia de datos

e. restricción de integridad
f. encriptado de datos
g. economía de escala
h. log (bitácora) de recuperación
i. vista de usuario
j. log de seguridad

Ejercicios de laboratorio

Ejercicio de laboratorio 1.1: Exploración de la base de datos Access para el ejemplo Universidad

Este laboratorio le proporcionará práctica en:

- Copia y uso de una base de datos existente
- Examen de tablas existentes, relaciones, consultas y reportes
- Diseño y ejecución de nuevas consultas
- Diseño y uso de un nuevo reporte
- Diseño y uso de una forma
- Actualización de tablas

1. Descargue la base de datos UniversityDB del website de este libro y guárdela en su propio directorio. Ábrala y úsela para los siguientes ejercicios.

Nota: Access usa muchas de las mismas convenciones que el resto de Microsoft Office y proporciona varias formas de realizar tareas. Existen muchas variaciones de los comandos descritos en esta sección que funcionan de la misma forma. Siéntase en libertad de explorar dichas variaciones. Si cierra Access accidentalmente, sólo vuelva a abrirlo y continúe los ejercicios. Si borra parte de la base de datos, borre su copia y comience de nuevo, y haga otra copia desde el CD.

2. Asegúrese de que el objeto *Tables* (tablas) se elija en el panel izquierdo de la ventana UniversityDB. Abra la tabla `Student` con un doble clic sobre su nombre en el panel derecho y recorra los registros de dicha tabla. Luego cierre la tabla. Haga lo mismo para las tablas restantes.

3. En la barra de herramientas Access, encuentre el ícono para *Relationships* (relaciones), que tiene tres rectángulos conectados mediante líneas. Dé clic en el ícono para ver las relaciones entre las tablas que Access "conoce". Note que el programa "sabe" que el `facId` en `Class` se relaciona con el `facId` en `Faculty`, que el `classNumber` en `Enroll` coincide con el `classNumber` en `Class`, y que el `stuId` en `Enroll` coincide con el `stuId` en `Student`. Cierre la ventana *Relationships*.

4. Elija el objeto *Queries* (consultas) en el panel izquierdo. Abra *Query1* con doble clic en su nombre. Note que la consulta se ejecuta inmediatamente. Cierre la pantalla de resultados de *Query1*.

5. Con el objeto *Queries* todavía elegido, desde la barra de herramientas de la base de datos UniversityDB, elija *Design* (diseño). Se muestra el diseño de *Query1*. La ventana superior muestra las tablas que se usan en la consulta, junto con sus relaciones. La ventana inferior muestra cuáles campos se incluyen en la consulta, junto con algunas condiciones (criterios). El diseñador puede elegir *Show* (mostrar) o no, para indicar si el campo se debe desplegar en el resultado. Desde la pantalla de diseño, presione el

ícono **!** en la barra de herramientas Access para correr la consulta. Cierre la ventana de resultados.

6. Regrese a la pantalla de diseño para *Query1*. Cambie la consulta: dé doble clic sobre el nombre de campo `major` en la tabla `Student` para agregar la especialidad. Ahora agregue la condición `History` en la línea *Criteria* para `major`, ejecute de nuevo la consulta y note el cambio en los resultados con esta nueva condición. Cierre la ventana de resultados sin guardar.

7. Elija la opción *Create query in design view* (crear consulta en vista diseño). En la ventana *Show Table* (mostrar tabla), resalte `Class`, dé clic en el botón *Add* (agregar), resalte `Enroll` y de nuevo dé clic en el botón *Add*. Dé clic en el botón *Close* (cerrar). Ahora está de vuelta en la ventana de diseño de consulta. Elija `classNumber`, `schedule`, `room` y `stuId` como los campos para la consulta, al dar doble clic sobre cada uno. Agregue la condición de que `stuId` es `S1002` en la línea *Criteria* (criterios) para `stuId`. Ejecute la consulta a fin de ver el horario de clases para el estudiante `S1002`. Cierre y guarde la consulta.

8. Cree una consulta que diseñe usted mismo. Puede variar su diseño al elegir diferentes tablas desde la ventana *Show Table*, y elegir diferentes campos, colocar múltiples condiciones en la misma línea de criterios para indicar AND, o poner condiciones en distintas líneas para indicar OR. Explore las opciones y cree varias consultas. Note que, si quiere comparar cadenas, el caso debe coincidir exactamente, así que tenga cuidado con las mayúsculas.

9. Elija el objeto *Reports* (reportes) en el panel izquierdo. Abra *Class Lists* (listas de clase) con doble clic en su nombre en el panel derecho. Agrande la ventana de reporte para leer el reporte. Cierre el reporte. Elija el ícono *Design* en la barra de herramientas UniversityDB para ver el diseño de dicho reporte. Cierre la pantalla *Design*.

10. Dé clic en *Create report by using wizard* (crear reporte con el asistente). Note la ventana desplegable a la izquierda que menciona todas las tablas y consultas. Podría elegir cualquier combinación de estos objetos para su reporte. Elija la tabla `Student`. Desde la lista de campo justo abajo, resalte `lastName` y presione el botón de flecha derecha para seleccionarla para el reporte (o simplemente dé doble clic en el nombre de campo). También elija `firstName`. Si por accidente elige un campo incorrecto, use la flecha de vuelta para quitar la selección. Cambie la tabla a `Enroll`. Elija `classNumber` y `grade`. Cambie la tabla a `Class`. Elija `schedule` y `room`. Ahora dé clic en el botón *Next* (siguiente) en el fondo de la ventana *Report Wizard*. Tiene la opción de ver sus datos por `Student`, `Enroll` o `Class`. Elija `Student` y dé clic en el botón *Next*. Ahora puede elegir niveles de agrupamiento. Elija `lastName`, luego dé clic en *Next*. Puede elegir ordenar en varios campos. Elija `grade`, *Ascending*, luego clic en *Next*. Elija el estilo por defecto, después *Next*. Escriba el título `Clases por estudiante. Diseñado por <su nombre>`, y ponga su propio nombre en los corchetes. Elija *Finish*. El reporte se debe ejecutar. Examínelo para ver los efectos de sus elecciones.

11. Con el asistente de reporte, diseñe y corra un segundo reporte de su propia elección. Guarde su reporte.

12. Elija *Forms* (formas) del panel de objetos a la izquierda. Elija *Create form by using wizard* (crear forma con el asistente). Elija la tabla `Student` y dé clic en la flecha doble para seleccionar todos sus campos. Elija todas las especificaciones por defecto. Escriba el título `EntradaEstudiante` (`Student Input`) y dé clic en *Finish* para terminar el diseño de la forma. La forma debe aparecer y mostrar cada registro de estudiante. Use la flecha de control de forma para avanzar a través de los registros de estudiantes, uno a la vez al presionar la flecha derecha. También puede ingresar datos usando la forma.

Después del último registro, verá una forma en blanco. Ingrese sus propios datos: establezca una ID (¡recuerde su valor!) e ingrese su nombre, especialidad y créditos. Cierre la forma *EntradaEstudiante*.

13. Elija de nuevo el objeto *Tables* y abra la tabla `Student`. Note que el registro que ingresó usando la forma se guardó en la tabla. Cierre la tabla `Student`. Ahora abra la tabla `Enroll` y agregue dos registros mientras está en la vista normal (hoja de cálculo) para mostrar que usted se inscribió en dos clases. Tenga cuidado de usar el mismo `stuId` que ingresó y use los valores `classNumber` existentes o no será capaz de agregar los registros. Borre el registro `Enroll` del estudiante S1010 en ART103A al mover el cursor a la columna anterior a `stuId` para seleccionar el registro y presione la tecla *Delete* (borrar). Guarde sus cambios y cierre la tabla.
14. Elija el objeto *Reports*. Corra el reporte *ClassLists* con doble clic en su nombre. Note que se mencionan usted y sus nuevas clases, y ya no aparece el registro borrado.
15. Abra la tabla `Faculty`. Cambie la posición del profesor Tanaka a `Asistente` (`Assistant`), al escribir el nuevo valor directamente en la celda. Guarde el cambio.

Ejercicio de laboratorio 1.2: Creación y uso de una nueva base de datos Access

Este laboratorio le proporcionará práctica en:

- Creación de una nueva base de datos Access
- Creación de tablas, y especificación de campos y claves
- Especificación de relaciones entre tablas
- Ingreso de registros
- Edición de registros
- Creación de consultas
- Creación de reportes simples
- Creación de formas simples

Para este laboratorio, creará una base de datos que rastree su colección de música y los amigos a quienes les prestó música.

1. Abra Access y elija *New, Blank Database* (base de datos nueva en blanco) del panel a la derecha de la pantalla de apertura. Elija su directorio en la ventana *Save in* (guardar en) y cambie el nombre del archivo de *db1* a `MiMusica` (`MyMusic`). Asegúrese de que el tipo del archivo esté establecido en *Microsoft Access Databases* y dé clic en el botón *Create*.
2. Asegúrese de que el objeto *Tables* está seleccionado en el panel izquierdo de la ventana *MiMusica*. Dé doble clic en *Create Table in Design View* (crear tabla en vista de diseño). En la ventana *Table1* que aparece, ingrese los nombres de campo y tipos de datos para todos los campos de su tabla. En la primera línea bajo *Field Name* (nombre de campo) escriba `título` (`title`). En la ventana *Data Type* en la misma línea, dé clic en la flecha abajo para ver los tipos de datos disponibles. Elija *text*. Baje a la ventana *Field Properties* (propiedades de campo) y escriba `35` como el tamaño del campo. Muévase a la segunda línea bajo *field name* e ingrese el nombre de campo `artista`, luego tipo de datos *text*, tamaño `20`. Haga el tercer campo `fechaAdquisición` y elija *date/time* como el tipo de datos. Haga el cuarto campo `estatus` con tipo de datos *text*, tamaño `10`. Luego, del menú Access, elija *File*, *Save As*, y escriba el nombre de tabla `Música`. Obtendrá un mensaje que le recuerda que no especificó una clave y le preguntará si le gustaría que Access cree una por usted. Elija *yes*. Access agregará un campo `ID` con tipo de datos *AutoNumber*. Cierre la tabla.

3. Cree una segunda tabla de amigos que pudieran pedirle prestada su música. El primer campo es `apellido`, tipo de datos *text* `20`. Esta vez creará su propia clave que consiste en `apellido`. Para hacerlo, mueva el cursor a la columna justo a la izquierda de `apellido` y dé clic en el ícono *key* en la barra de herramientas Access. Continúe para crear el resto de los campos

```
nombre text 15
códigoPostal text 3
teléfono text 7
```

   Guarde esta tabla como `Amigos`.

4. Cree una tercera tabla, `Préstamo`, con campos

```
ID Number Long Integer
apellido text 20
fechaPréstamo date/time
fechaRegreso date/time
```

   Esta tabla necesita una clave compuesta que consiste en `ID` y `apellido`. Para crear la clave compuesta, mueva el cursor a la primera columna a la izquierda de `ID` y, con el botón del ratón presionado, mueva el ratón abajo una línea para seleccionar tanto la línea `ID` como la de `apellido`. Con ambos resaltados, dé clic en el ícono *key*. Guarde la tabla como `Préstamo`.

5. Para crear relaciones entre las tablas, dé clic en el ícono *Relationships* en la barra de herramientas Access. Consiste en tres rectángulos conectados mediante líneas. En la ventana *Show Table* que aparece, resalte `Música (Music)`, dé clic en el botón *Add*, resalte `Préstamo`, haga clic en *Add*, resalte `Amigos`, haga clic en *Add*, luego clic en *Close*. Se abre la ventana Relationship, que muestra las tres tablas. Dé clic en `apellido` en `Amigos` y arrastre a `apellido` en `Préstamo`. En la ventana *Edit Relationship* que aparece, debe ver ambos campos mencionados. Marque el recuadro *Enforce referential integrity* (fortalecer integridad referencial) y luego dé clic en el botón *Create*. Verá una línea que conecta las dos tablas con 1...∞ en ella. El 1 debe estar cerca de la tabla `Amigos` y el símbolo de infinito cerca de la tabla `Préstamo`. (Si éste no es el caso, cometió un error al diseñar las tablas. Puede dar clic en la línea de relación y presionar la tecla *Delete* para remover la relación. Entonces puede regresar al diseño de tabla y corregir cualquier error.) Arrastre `ID` de `Música` a `Préstamo` y cree otra relación. Elija *Save* y cierre la ventana de relación.

6. Dé doble clic en el nombre de la tabla `Música`. Ahora ingresará datos en esta tabla. Para la tabla `Música`, el sistema ingresará una `ID` (1, 2, 3,...) automáticamente (porque le permitió crear una clave *AutoNumber* por usted), pero debe ingresar los nombres de los álbumes que tenga, el cantante o artista, la fecha en que adquirió el álbum (en Estados Unidos, use la forma mm/dd/aaaa; de otro modo, use la convención local para fechas) y el estatus. Puede dejar el estatus en blanco o ingresar `prestado`, `OK`, `rallado` o cualquier valor cadena adecuado. Cuando haya ingresado varios álbumes, guarde la tabla `Música` y ciérrela.

7. Ingrese datos en su tabla `Amigos`. Esta vez `apellido` es la clave. Debe tener cuidado de no ingresar dos registros con el mismo apellido y recordar los valores que ingresó.

8. Ahora puede ingresar datos en la tabla `Préstamo`. Access comprobará para ver que cualquier ID que ingrese coincide con la ID en la tabla `Música`, y cualquier `apellido` coincide con uno en la tabla `Amigos`, para asegurarse de que los valores son válidos. La comprobación se realiza porque usted informó a *Access* de las relaciones y le pidió reforzar la integridad referencial. Guarde esta tabla.

9. Ahora creará una consulta que usted diseñe. Elija el objeto *Query*. Elija la opción *Create query in design view*. En la ventana *Show Table*, resalte `Música`, dé clic en el botón *Add*, resalte `Préstamo`, dé clic en el botón *Add*, resalte `Amigos` y nuevamente dé clic en el botón *Add*. Dé clic en el botón *Close*. Ahora está en la ventana de diseño de consulta. Elija cualquier campo que quiera incluir en su consulta. Recuerde que puede elegir cualquiera de las tablas, elegir diferentes campos, poner condiciones múltiples en la misma línea de criterios para indicar AND o poner condiciones en distintas líneas para indicar OR. Explore las opciones y cree varias consultas. Note que, si quiere comparar cadenas, el caso debe coincidir exactamente, así que tenga cuidado con las mayúsculas. Diseñe y ejecute varias consultas, guárdelas bajo nombres que elija.
10. Ahora debe diseñar un reporte. Elija el objeto *Reports* en el panel izquierdo. Dé clic en *Create report by using Wizard*. Note la ventana despleglable a la izquierda que menciona todas las tablas y consultas. Podría elegir cualquier combinación de estos objetos para su reporte. Elija una tabla. De la lista de campo justo abajo, resalte cualquier campo que quiera incluir y presione el botón de flecha derecha para seleccionarlo para el reporte (o simplemente dé doble clic en el nombre del campo). Recuerde que, si accidentalmente elige un campo incorrecto, use la flecha de retroceso para quitar la selección. Después de escoger los campos que quiere de las tablas y/o consultas, dé clic en el botón *Next* en el fondo de la ventana *Report Wizard*. Tiene la opción de ver sus datos mediante varios órdenes y niveles de agrupamiento, y elegir un estilo para su reporte. Agregue un título que incluya su nombre. Elija *Finish* para terminar el diseño y corra el reporte.
11. Elija el objeto *Forms* del panel objetos a la izquierda. Elija *Create form by using Wizard*. Elija una tabla y dé clic en la flecha doble para seleccionar todos sus campos. Elija todas las especificaciones por defecto. Debe aparecer la forma y mostrar los registros de la tabla uno a la vez. Recuerde que también puede ingresar datos usando la forma. Después del último registro, verá una forma en blanco sobre la cual debe ingresar sus propios datos.
12. Guarde todos sus cambios y salga de Access.

Nota: Una vez que creó una tabla, siempre puede agregar nuevos registros al simplemente abrir la tabla e ingresar los datos. Puede actualizar un registro existente al mover el cursor al campo a cambiar y escribir el nuevo valor. Puede borrar un registro al resaltarlo y presionar la tecla *Delete* o usar las opciones de menú. Practique cada una de estas operaciones en cualquier tabla que elija. Puede imprimir la tabla o cualquier otro objeto en cualquier momento desde el menú Access al elegir *File, Print*.

PROYECTO DE MUESTRA: LA GALERÍA DE ARTE

Propósito del proyecto de muestra

Las secciones de proyecto de muestra incluidas al final de muchos de los capítulos de este libro proporcionan al estudiante modelos para aplicar los conceptos cubiertos en los capítulos. El proyecto es un ejemplo continuo que ilustra una aplicación práctica de las técnicas de diseño e implementación de bases de datos. Después del proyecto de muestra hay varios proyectos estudiantiles. Los estudiantes deben elegir al menos uno de los proyectos y trabajar sobre su desarrollo conforme avanzan a lo largo del libro. El proyecto de muestra presenta cómo se puede realizar cada paso. El estudiante debe leer la muestra y aplicar los pasos al proyecto elegido.

Descripción general

La Galería de Arte acepta obras de arte originales de artistas contemporáneos vivos para vender sobre una base de comisiones. En la actualidad ofrece obras de más o menos un cen-

tenar de artistas, y vende aproximadamente mil piezas cada año. El precio de venta promedio es de varios miles de dólares. Existen alrededor de cinco mil clientes que compran piezas de la galería. El personal de ventas consiste en el dueño de la galería, Alan Hughes, y cuatro asociados de ventas. Sus actividades las apoya un personal administrativo de dos personas.

Operaciones básicas

Cuando un artista quiere vender obras, contacta a la galería. Alan Hughes, el dueño, visita el estudio del artista y selecciona la obra a vender a través de la galería. Si el artista es bien conocido en la galería, se puede eliminar esta visita y las obras se pueden aceptar automáticamente. Un artista puede enviar una o varias piezas para su venta al mismo tiempo. El artista, que trabaja con Alan, identifica un precio solicitado para cada obra. El personal de ventas intenta vender la obra en dicho precio, o tan cerca de dicho precio como sea posible. Los clientes pueden negociar con el vendedor, de modo que el precio de venta real puede estar por abajo del precio solicitado. Si está por abajo del precio solicitado, el precio de venta final debe ser aprobado por el artista. La comisión que carga la galería es de 10% del precio de venta. La galería divide la comisión con el vendedor que realiza la venta. Cualquier vendedor puede vender cualquier obra en la galería. Sin embargo, los clientes trabajan con un solo vendedor cuando compran cada pieza, de modo que la porción de la comisión del vendedor para una sola pieza va sólo a un vendedor.

La galería promueve las obras al mantener exposiciones que presentan varias piezas. Las exposiciones se publicitan en periódicos y otros medios, y a los clientes potenciales se les envían invitaciones personales. Una exhibición es en realidad una recepción que proporciona una oportunidad para que el público vea las piezas y se encuentre con el artista o artistas cuyas obras se presentan. Una "exposición individual" presenta obras de un solo artista, mientras que una exposición colectiva presenta obras de varios artistas centrados en un solo tema, como "Vistas marinas mediterráneas". Las obras de arte que se presentaron en una exposición permanecen en exhibición hasta que se venden o regresan a los artistas. Una pieza se puede comprar en la exposición o en cualquier momento posterior. Ocasionalmente, una obra se puede comprar de la galería previa a la exposición e incluirse en la exhibición, marcada como "Vendida", para proporcionar al público una mejor vista del trabajo del artista. No todas las obras se promueven mediante exhibiciones. Algunas sólo se muestran en la galería. Si una obra ha estado en la galería durante seis meses sin venderse, Alan contacta al artista y regresa la obra, a menos que ambos acuerden continuar la exhibición de la obra durante un periodo adicional.

En la actualidad, todos los datos relacionados con los artistas, obras no vendidas, exposiciones, ventas y clientes se mantienen en archivos de papel. Para cada obra actualmente en exhibición se elabora una tarjeta descriptiva, que se coloca en la pared o pedestal junto a la pieza. Una copia de la tarjeta también se coloca en un archivo. La tarjeta menciona el nombre del artista, título de la obra, año de creación, tipo, medio, estilo, tamaño y precio solicitado. Cada obra es una pieza original exclusiva producida por un solo artista. Dos artistas no tienen el mismo nombre. El título de la obra debe ser único al artista, pero puede no ser totalmente único para la galería. Por ejemplo, muchos artistas pueden tener obras como "Composición número 5", pero ningún artista tiene dos obras con dicho título. En la galería no se venden impresiones o reproducciones. Un artista puede producir muchas obras en el mismo año. El tipo se refiere al tipo de obra, que puede ser pintura, escultura, collage, y así por el estilo. El medio se refiere a los materiales usados en la obra, como óleo, acuarela, acrílico, mármol o mixto. Una pieza que use más de un medio se categoriza como mixta. El estilo significa el estilo de la obra, que puede ser contemporáneo, impresionista, folk u otro. El tamaño se expresa en unidades adecuadas para la obra; por ejemplo, para una pintura, el

tamaño sería el número de centímetros en ancho y alto, mientras que una escultura tendría tres dimensiones.

Cuando se realiza una compra, se emite un recibo para el comprador, un cheque y un comprobante de pago para el artista, la comisión se asigna entre la galería y el vendedor, y todos los archivos en papel se actualizan de forma individual.

Necesidades de información

Además de los datos acerca de artistas, obras de arte, exhibiciones, ventas y clientes que actualmente se mantienen en archivos de papel, existen otras necesidades de información. Para propósitos de impuesto sobre la renta, a la galería se le requiere reportar la cantidad de ventas por cada artista cada año, una labor que en el presente consume mucho tiempo. Alan se da cuenta de que una base de datos podría proporcionar más información de la que ahora está disponible en archivos de papel. También quiere capturar datos que en la actualidad no se almacenan. Le gustaría seguir la pista a los clientes que han hecho compras e información acerca de la cantidad de sus compras el último año y hasta el momento en este año. Le gustaría tener posibilidad de enviar correos a los potenciales clientes y registrar sus preferencias. Además, prevé que la galería puede comenzar a aceptar obras propiedad de coleccionistas así como obras directamente de artistas. El diseño de la base de datos incluiría la posibilidad de que el propietario no sea el artista.

Pasos del proyecto

- Paso 1.1. Escribir el formato de cada documento de entrada que proporcione información a almacenar en la base de datos.
- Paso 1.2. Escribir el formato de cada reporte de rutina a producir usando la base de datos.
- Paso 1.3. Bosquejar las pantallas de entrada y salida para cada rutina de transacción a realizar contra la base de datos.
- Paso 1.4. Escribir una lista inicial de suposiciones para el proyecto.

Nota: En la vida real, estos pasos estarían precedidos por reuniones y entrevistas con los usuarios del sistema actual y del sistema propuesto para determinar las necesidades de datos y preferencias de los usuarios. Se supondrá que estas reuniones tuvieron lugar y que la información que sigue se desarrolló a partir de ellas. Note que, en este punto, no se hacen suposiciones acerca de la estructura interna de la base de datos. Los reportes y formas se diseñan con base en las necesidades del usuario, no de las estructuras de archivos de la base de datos.

- Paso 1.1. Formato de documentos de entrada

Las siguientes formas se usan para proporcionar información.

1. **Forma de información del artista.** Cuando Alan entrevista a un artista, recopila información de contacto y datos acerca de las obras usuales del artista, como se muestra en la forma de la figura 1.7. Para permitir la posibilidad de que en el futuro los socios de Alan puedan realizar entrevistas, se menciona el nombre del entrevistador.
2. **Forma de información del coleccionista.** Cuando la galería comienza a ofrecer obras propiedad de personas distintas al artista, también se entrevistará a estos coleccionistas. Pueden poseer una o más obras de arte, y sus colecciones pueden o no

LA GALERÍA DE ARTE

Forma de información del artista

Fecha de entrevista _____ Nombre del entrevistador _____

Apellido del artista _____ Apellido del artista _____

Calle _____

Ciudad_____ Estado ___ C.P. _____

Teléfono: Código de área ____ Número _____

R.F.C. _____

Tipo usual _____ Medio usual _____

Estilo usual _____

FIGURA 1.7

Forma de información del artista

tener obras que sean predominantemente de un solo artista o de un solo tipo, estilo o medio. La forma que se muestra en la figura 1.8 la llenará el entrevistador.

3. **Forma de información de la obra de arte.** Para cada obra de arte a considerar, el entrevistador llena la información básica necesaria para la tarjeta descriptiva, como se muestra en la figura 1.9. Si la pieza se elige para ofrecerse a la venta por parte de la galería, se ponen la fecha citada y el precio solicitado.
4. **Factura de venta.** Cuando se vende una obra, el vendedor llena la forma que se muestra en la figura 1.10. En la actualidad, una copia se le da al comprador y el origi-

LA GALERÍA DE ARTE

Forma de información del coleccionista

Fecha de entrevista _____ Nombre del entrevistador _____

Apellido del coleccionista _____ Nombre _____

Calle _____

Ciudad _____ Estado ___ C.P. _____

Teléfono: Código de área ____ Número _____

R.F.C. _____

Si la colección es predominantemente de un artista, o tiene un tipo, medio o estilo distintivo, llenar esta sección.

Apellido del artista _____ Nombre del artista _____

Tipo de colección _____

Medio de la colección _____

Estilo de la colección _____

FIGURA 1.8

Forma de información del coleccionista

LA GALERÍA DE ARTE

Forma de información de la obra de arte

Apellido del artista _____ Nombre del artista _____

Título _____

Año de conclusión _____

Tipo _____

Medio _____

Estilo _____

Tamaño _____

Si el propietario es alguien distinto al artista, por favor complete esta sección con la información del propietario.

Apellido del propietario _____ Nombre del propietario _____

Calle _____

Ciudad _____ Estado ___ C.P. _____

Teléfono: Código de área ____ Número _____

R.F.C. del propietario _____

Si la pieza se elige para ofrecerse en la galería, por favor complete esta sección.

Fecha citada _____

Precio solicitado _____

FIGURA 1.9

Forma de información de la obra de arte

nal se coloca en los archivos. El único número de factura está preimpreso en la forma. Cuando se crea la base de datos, el sistema producirá la factura.

5. **Forma de lista de correos.** La forma que se muestra en la figura 1.11 se deja en varias ubicaciones para que los potenciales consumidores la firmen para una lista de correos.

- Paso 1.2. Formato de rutina de reportes

Los siguientes reportes se producen actualmente o se producirían con el nuevo sistema.

6. **Reporte resumen de artistas activos.** El reporte que se muestra en la figura 1.12 menciona datos de resumen acerca de todos los artistas activos, incluida la cantidad total de las ventas de cada uno durante el último año y este año.
7. **Reporte de ventas por artista individual.** El reporte que se muestra en la figura 1.13 se generaría para un periodo que comience con cualquier fecha que se seleccione (por ejemplo, primero de enero del año actual) y terminaría con otra fecha seleccionada (por ejemplo, la fecha de hoy). Menciona todas las obras del artista que la galería ha recibido desde la fecha de lista especificada hasta la fecha del reporte. El estatus de la obra puede ser vendida, regresada o en venta. Si la obra se vendió, se mencionan la fecha de venta y el precio de venta. Si la obra se regresó, se menciona la fecha de regreso. Si la obra está en venta a la fecha del reporte, se menciona el precio solicitado. Se muestra la cantidad total de ventas de las obras del artista durante el perio-

LA GALERÍA DE ARTE

Factura de venta

Factura número 99999

Título de la obra _____

Artista: Apellido _____ Nombre _____

Propietario: Apellido _____ Nombre _____

Calle _____

Ciudad _____ Estado ___ C.P. _____

Teléfono: Código de área ____ Número _____

R.F.C. del propietario _____

Comprador: Apellido _____ Nombre_____

Calle _____

Ciudad_____ Estado _____ C.P. _____

Teléfono: Código de área ____ Número _____

Precio _____

Impuesto _____

Pago total _____

Firma del vendedor _____ Fecha _____

FIGURA 1.10

Factura de venta

LA GALERÍA DE ARTE

Lista de correos

Fecha _____

Apellido _____ Nombre _____

Calle _____

Ciudad _____ Estado ___ C.P. _____

Teléfono: Código de área ____ Número _____

Por favor indique abajo sus preferencias (si tiene alguna):

Artista preferido _____

Estilo preferido (por ejemplo, contemporáneo, impresionista, folk) _____

Tipo preferido (por ejemplo, pintura, escultura, collage) _____

Medio preferido (por ejemplo, óleo, acuarela, mármol, mixto) _____

FIGURA 1.11

Forma de lista de correos

REPORTE DE ARTISTAS ACTIVOS

Fecha de reporte: mm/dd/aaaa

Nombre	Dirección	Teléfono	Tipo	Medio	Estilo	Ventas año pasado	Ventas a la fecha
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99
. . .							
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99

FIGURA 1.12

Reporte resumen de artistas activos

REPORTE DE VENTAS DE ARTISTA INDIVIDUAL

Fecha de reporte: mm/dd/aaaa

Apellido _____ Nombre_____

Dirección: Calle _____

Ciudad _____ Estado ____ C.P._____

Teléfono: Código de área_____ Número _____

Reporte de obras que comienza con fecha citada de mm/dd/aaaa

Obras vendidas:

Título	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado	Precio de venta	Fecha de venta
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
. . .								
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
						Total de ventas:	999999.99	

Obras regresadas:

Título	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado	Fecha de regreso
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa
. . .							
XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa

Obras en venta:

Título	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado
XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
. . .						
XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
					Total de precios solicitados:	999999.99	

FIGURA 1.13

Reporte de ventas de artista individual

REPORTE RESUMEN DE COLECCIONISTAS

Nombre	Dirección	Teléfono	Artista pref.	Tipo. pref.	Medio pref.	Estilo pref.	Ventas último año	Ventas a la fecha
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99
. . .								
XXXXX	XXXXX	XXX XXX XXXX	XXXXX	XXXXX	XXXXX	XXXXX	999999.99	999999.99

FIGURA 1.14

Reporte resumen de coleccionistas

do. También se proporciona el valor total de los precios solicitados de las obras del artista actualmente en venta. Al elegir fechas que abarcan todo el año, los datos de ventas totales en este reporte también se pueden usar para el reporte de impuestos de fin de año requerido por el gobierno.

8. **Reporte resumen de coleccionistas.** La Galería de Arte planea comenzar la venta de obras propiedad de coleccionistas, además de las obras propiedad del artista que las creó. Cuando estén disponibles las obras pertenecientes a personas distintas al artista, se necesitará el reporte que se muestra en la figura 1.14.
9. **Reporte de ventas por coleccionista individual.** Este reporte, que se muestra en la figura 1.15, es similar al de los artistas individuales. Se necesitará cuando la galería comience a vender obras propiedad de coleccionistas. Proporciona información acerca de las obras que el coleccionista ofreció para su venta a través de la galería. Menciona todas las obras vendidas, las obras regresadas y las obras en venta para dicho coleccionista por el periodo especificado. Las ventas totales para cada coleccionista se envían al gobierno con el propósito de reportar los impuestos a final del año.
10. **Obras en venta.** Este reporte menciona datos acerca de cada obra que actualmente se ofrece en venta en la galería. Si existe, se proporciona la fecha de la exhibición para promover la obra. Se proporciona el total de todos los precios solicitados. El reporte se muestra en la figura 1.16.
11. **Ventas de esta semana.** Este reporte, que se muestra en la figura 1.17, menciona datos acerca de todas las ventas de obras durante la semana actual. Se divide por vendedor, y muestra las obras que cada vendedor vendió esta semana y sus ventas totales. Al final, proporciona el gran total de todas las ventas de la semana.
12. **Reporte de ventas por comprador.** El reporte de ventas por comprador se muestra en la figura 1.18. Los datos del comprador vienen de las facturas. El reporte muestra compradores en orden alfabético por apellido. Las obras que compraron este año se muestran en orden por fecha de compra.
13. **Reporte de clientes favoritos.** A Alan le gustaría ubicar a los clientes potenciales, así como a los actuales, al mantener información acerca de todos aquellos que asisten a las exposiciones, o cuyos nombres se recopilen de la forma de información de cliente potencial. Para cada cliente actual y potencial, le gustaría conservar datos de identificación e información acerca de las preferencias del cliente, como el nombre de un artista, tipo, medio y estilo preferidos por cada cliente. Él espera aumentar las ventas y bajar los costos al usar esta información a fin de elaborar listas de invitación dirigida para exposiciones de obras que coincidan con las preferencias del cliente. Por

REPORTE DE VENTAS POR COLECCIONISTA INDIVIDUAL

Fecha de reporte: mm/dd/aaaa

Apellido _____ Nombre _____

Dirección: Calle _____

Ciudad _____ Estado ____ C.P. _____

Teléfono: Código de área _____ Número _____

Reporte para obras que comienza con la fecha citada de mm/dd/aaaa

Obras vendidas:

Título	Artista	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado	Precio de venta	Fecha de venta
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
. . .									
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	aaaa	9999.99	9999.99	mm/dd/aaaa
							Total de ventas:	999999.99	

Obras regresadas:

Título	Artista	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado	Fecha de regreso
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa
. . .								
XXXXX	XXXXX	mm/dd/aaaa	XXXXX	XXXXX	XXXXX	9999	9999.99	mm/dd/aaaa

Obras en venta:

Título	Artista	Fecha citada	Tipo	Medio	Estilo	Año	Precio solicitado
XXXXX	XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
XXXXX	XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
. . .							
XXXXX	XXXXX	99/99/9999	XXXXX	XXXXX	XXXXX	9999	9999.99
							Total de precios solicitados: 999999.99

FIGURA 1.15

Reporte de ventas por coleccionista individual

OBRAS EN VENTA

Fecha de reporte: _____

Título	Artista	Tipo	Medio	Estilo	Nombre del propietario	Precio solicitado	Fecha mostrada	Fecha citada
XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	9999.99	mm/dd/aaaa	mm/dd/aaaa
XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	9999.99	mm/dd/aaaa	mm/dd/aaaa
. . .								
XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	9999.99	mm/dd/aaaa	mm/dd/aaaa
						Total de precios solicitados:	999999.99	

FIGURA 1.16

Obras en venta

VENTAS PARA SEMANA QUE TERMINA EN MM/DD/AAAA

Vendedor	Artista	Título	Propietario	Comprador	Fecha de venta	Precio de venta	Com
XXXX							
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
		XXX	XXX	XXX	mm/dd	9999.99	999.99
	. . .						
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
					Total:	9999.99	999.99
XXXX							
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
	. . .						
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
. . .					Total:	9999.99	999.99
XXXX							
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
		XXX	XXX	XXX	mm/dd	9999.99	999.99
	. . .						
	XXX	XXX	XXX	XXX	mm/dd	9999.99	999.99
					Total:	9999.99	999.99

Total de todas las ventas para la semana: 99999.99

FIGURA 1.17

Ventas de esta semana

ejemplo, le gustaría ser capaz de obtener un reporte como el que se muestra en la figura 1.19. Este reporte podría correrse para el artista o artistas que se presenten en una exposición. Menciona clientes potenciales cuyas preferencias establecidas mencionan al mismo artista, tipo, medio o estilo de las obras que se exponen.

14. **Reporte de desempeño del vendedor.** El reporte que se muestra en la figura 1.20 se generaría por un periodo que comience con cualquier fecha que se seleccione (por ejemplo, primero de enero del año actual) y termina con otra fecha seleccionada (por ejemplo, la fecha de hoy). Proporciona un listado individual de cada una de las obras vendidas por dicha persona durante el periodo, así como sus ventas totales para el periodo elegido. Por lo general, correría una vez al mes, para permitir a Alan evaluar el desempeño de cada vendedor.
15. **Reporte de obras añejas.** Este reporte, que se muestra en la figura 1.21, se genera al final de cada mes. Menciona las obras de arte que han estado a la venta en la galería durante seis meses o más. Alan la usa con el fin de contactar al artista o coleccionista para determinar si las obras se deben regresar, o permanecer para venta durante un periodo adicional.
16. **Comprobante de pago del propietario.** Cuando una obra se vende, al propietario se le envía un cheque por el 90% del precio de venta. El comprobante que acompaña al cheque se muestra en la figura 1.22.

REPORTE DE VENTAS POR COMPRADORES

Fecha mm/dd/aaaa

Apellido	Nombre	Dirección	Teléfono	Total de compras último año
XXXX	XXXX	XXXX	XXXX	9999.99

Compras de este año:

Fecha de compra	Artista	Título	Precio solicitado	Precio de venta
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
. . .				
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
			———	———
		Total de compras este año:	9999.99	9999.99

Apellido	Nombre	Dirección	Teléfono	Total de compras último año
XXXX	XXXX	XXXX	XXXX	9999.99

Compras de este año:

Fecha de compra	Artista	Título	Precio solicitado	Precio de venta
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
. . .				
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
			———	———
		Total de compras este año:	9999.99	9999.99

. . .
. . .
. . .

Apellido	Nombre	Dirección	Teléfono	Total de compras último año
XXXX	XXXX	XXXX	XXXX	9999.99

Compras de este año:

Fecha de compra	Artista	Título	Precio solicitado	Precio de venta
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
. . .				
mm/dd/aaaa	XXX	XXX	9999.99	9999.99
			———	———
		Total de compras este año:	9999.99	9999.99

FIGURA 1.18

Reporte de ventas por compradores

17. **Detalles de exhibición de arte.** Para cada exposición, este reporte proporciona información acerca de las fechas, artista o tema que se presenta, y obras en exhibición. Aparece en la figura 1.23.

- Paso 1.3. Bosquejo de pantallas para transacciones de rutina

Para todas las transacciones, al usuario se le conmina a elegir de un menú de posibles transacciones, y se proporcionan instrucciones para llenar la información necesaria. La pantalla muestra los resultados, que también se pueden imprimir.

18. **Agregar un nuevo artista.** Un miembro del personal administrativo ingresa los datos de la forma de información del artista. La pantalla tiene la misma plantilla que

REPORTE DE CLIENTES FAVORITOS

Artista	Título	Tipo	Medio	Estilo	Nombre cliente	Dirección	Artista pref.	Tipo pref.	Medio pref.	Estilo pref.
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX
	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX
	. . .									
	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX
. . .										
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX
	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX
	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX
					XXX	XXX	XXX	XXX	XXX	XXX
					. . .					
					XXX	XXX	XXX	XXX	XXX	XXX

FIGURA 1.19

Reporte de clientes favoritos

dicha forma. La pantalla de resultados informa al usuario que se agregó el artista o que el artista ya estaba en la base de datos.

19. **Agregar un nuevo coleccionista.** De igual modo, un trabajador administrativo ingresa datos de la forma de información del coleccionista, en una pantalla con la misma plantilla que la forma. La pantalla de resultados informa al usuario que la persona se ingresó o que ya está en la base de datos.
20. **Agregar una nueva obra de arte.** La información acerca de una nueva obra de arte se toma directamente de la forma de información y se ingresa en una pantalla con la misma plantilla que la forma. La base de datos se comprueba para garantizar que la combinación de nombre de artista y título es única, y luego despliega una pantalla que dice que la obra se agregó.
21. **Transacción de venta.** Los datos que se muestran en la factura, figura 1.10, se ingresan en una pantalla de transacción de ventas que tiene la misma plantilla que la factura. El recibo, que omite la dirección y el número telefónico del propietario, se despliega como respuesta, y un empleado imprime el recibo.

REPORTE DE DESEMPEÑO DEL VENDEDOR

Fecha de inicio de reporte mm/dd/aaaa **Fecha de término de reporte mm/dd/aaaa**

Nombre vendedor	Dir	RFC	Artista	Título	Precio solicitado	Precio de venta	Fecha de venta
XXX	XXX	999-99-9999	XXX	XXX	9999.99	9999.99	mm/dd/aaaa
				XXX	9999.99	9999.99	mm/dd/aaaa
			. . .				
			XXX	XXX	9999.99	9999.99	mm/dd/aaaa
					Ventas totales por periodo:	9999.99	
					Comisión total por periodo:	9999.99	
XXX	XXX	999-99-9999	XXX	XXX	9999.99	9999.99	mm/dd/aaaa
				XXX	9999.99	9999.99	mm/dd/aaaa
			. . .				
			XXX	XXX	9999.99	9999.99	mm/dd/aaaa
					Ventas totales por periodo:	9999.99	
					Comisión total por periodo:	9999.99	

FIGURA 1.20

Reporte de desempeño del vendedor

OBRAS MANTENIDAS DURANTE MÁS DE SEIS MESES

Fecha de reporte mm/dd/aaaa

Nombre propietario	Teléfono propietario	Nombre artista	Título	Fecha citada	Precio solicitado
XXX	XXX XXX XXXX	XXX	XXX	mm/dd/aaaa	9999.99
			XXX	mm/dd/aaaa	9999.99
		. . .			
		XXX	XXX	mm/dd/aaaa	9999.99
XXX	XXX XXX XXXX	XXX	XXX	mm/dd/aaaa	9999.99
		XXX	XXX	mm/dd/aaaa	9999.99
		. . .			
		XXX	XXX	mm/dd/aaaa	9999.99
. . .					
XXX	XXX XXX XXXX	XXX	XXX	mm/dd/aaaa	9999.99
			XXX	mm/dd/aaaa	9999.99
		. . .			
		XXX	XXX	mm/dd/aaaa	9999.99

FIGURA 1.21

Reporte de obras añejas

LA GALERÍA DE ARTE

Pago por venta de obra

Nombre de propietario _____

Dirección de propietario _____

Ciudad _____ Estado ___ C.P. _____

Teléfono: Código área ____ Número _____

R.F.C. del propietario _____

Nombre del artista _____ Título _____

Tipo _____ Medio _____ Estilo _____ Tamaño _____

Vendedor _____

Precio de venta 9999.99 Impuesto: 9999.99 Cantidad total de venta 9999.99

Cantidad remitida a propietario 9999.99

FIGURA 1.22

Comprobante de pago

REPORTE DE EXHIBICIÓN DE ARTE

Título de exhibición _____

Fecha de apertura _____ Fecha de cierre _____

Artista presentado _____ o tema _____

Obras incluidas:

Artista	Título	Precio solicitado	Estatus (*vendido* o *en venta*)
XXX	XXX	9999.99	XXX
	XXX	9999.99	XXX
	. . .		
XXX	XXX	9999.99	XXX
	XXX	9999.99	XXX
	. . .		
. . .			
XXX	XXX	9999.99	XXX
	XXX	9999.99	XXX
	. . .		

FIGURA 1.23

Reporte de exhibición de arte

22. **Agregar un cliente potencial.** Los datos que se muestran en la forma de lista de correos se ingresan para cada cliente potencial. Las personas que compran una obra de arte y los coleccionistas de obras de arte en la galería también se agregan automáticamente al archivo de clientes, usando la información de la factura de ventas y las

formas de información. La pantalla de respuesta confirma que la persona se agregó o que ya estaba en la base de datos.

- Paso 1.4. Lista inicial de suposiciones para el proyecto de la Galería de Arte

1. Los nombres de los artistas son únicos, pero los nombres de los clientes y coleccionistas no lo son.
2. Por razones de privacidad, sólo a las personas que reciben pagos de la galería se les pide proporcionar su registro federal de contribuyentes, porque dichos pagos tienen que reportarse por razones de impuesto sobre la renta. Por tanto, la galería mantiene los R.F.C. de vendedores, coleccionistas y artistas, pero no de los compradores o clientes potenciales.
3. Un artista puede tener muchas obras en venta en la galería.
4. Cada obra es una pieza original exclusiva. No se venden impresiones o reproducciones.
5. Dos obras de arte pueden tener el mismo título, pero la combinación de título y nombre de artista es única.
6. Una obra de arte puede ser propiedad o del artista que la creó o de otra persona, a la que aquí se le refiere como coleccionista.
7. Incluso si la obra de arte es propiedad de un coleccionista, es importante mantener información acerca del artista que la creó, pues éste es un factor para la determinación de su valor.
8. La galería vende una obra de arte sólo una vez. La galería no revende sus propias obras.
9. Una obra de arte puede aparecer en más de una exposición. Algunas obras no aparecen en exposición alguna.
10. El pago por todas las ventas se realiza de inmediato y por completo al momento de la compra. El pago puede ser crédito, efectivo o cheque. Al propietario se le paga el saldo y al vendedor se le paga la comisión al terminar la semana.
11. La base de datos no incluye información de nómina, excepto por la comisión a pagar al vendedor por la venta de obras de arte.
12. Hay listas de valores válidos por tipo, estilo y medio de obras de arte. Cada una tiene un valor "otro" para las obras que no encajan en los valores existentes.
13. La información acerca de las obras no seleccionadas para ser citadas por la galería se descarta.
14. Las listas de artistas, coleccionistas, compradores y clientes potenciales se evalúa periódicamente para determinar si se deben retirar.

PROYECTOS ESTUDIANTILES: INTRODUCCIÓN A LOS PROYECTOS ESTUDIANTILES

En las siguientes páginas se describen varios proyectos. Debe estudiar el proyecto con el que trabajará, leer el proyecto de muestra precedente y usarlo como modelo para llevar a cabo los pasos para su proyecto. Si puede hacerlo, entreviste a personas que estén familiarizadas con el entorno descrito en el proyecto. Con base en sus entrevistas, la descripción escrita y su propio análisis del proyecto que eligió, realice los siguientes pasos. Recuerde que en este

punto no debe hacer suposiciones acerca de la estructura interna de la base de datos. Sus reportes y formas se deben basar en las necesidades del usuario, no en las estructuras de archivo de la base de datos.

- Paso 1.1. Escriba el formato de cada documento de entrada que proporcione información a almacenar en la base de datos.
- Paso 1.2. Escriba el formato de cada reporte de rutina a producir usando la base de datos.
- Paso 1.3. Bosqueje las pantallas de entrada y salida para cada rutina de transacción a realizar contra la base de datos.
- Paso 1.4. Escriba una lista inicial de suposiciones para el proyecto.

Proyecto uno: Colecta anual de la Universidad Beta

Descripción general

La oficina de desarrollo de la Universidad Beta busca obtener donativos para su Colecta Anual a partir de varios donadores. La colecta recauda más de 10 millones de dólares cada año. Los donadores incluyen graduados, alumnos, padres, personal docente, administradores, personal administrativo, corporaciones o amigos de la universidad. Existen aproximadamente 100 000 donadores potenciales. La Colecta Anual la dirige Suzanne Hayes, quien es responsable de recaudar fondos y seguir la pista de las donaciones. Suzanne quiere crear una base de datos que le ayude con estas dos grandes responsabilidades.

Operaciones básicas

Suzanne intenta recaudar fondos de varias formas durante cada año fiscal, que se extiende del 1 de julio al 30 de junio. Cada otoño, todos los potenciales donadores a la Colecta Anual reciben cartas personalizadas de ella, que enfatizan sus lazos cercanos con la Universidad Beta. Las cartas contienen sobres de respuesta y formas en las que los donadores pueden indicar la cantidad que garantizan aportar ese año, y el método de pago que eligen. El pago se puede enviar como un solo cheque en el sobre, los donadores pueden elegir pagos diferidos durante un periodo de un año o pueden proporcionar el número de una tarjeta de crédito para pagar en una sola exhibición. Con frecuencia, el empleador del donador o del cónyuge del donador tiene un programa para hacer un donativo a la universidad, y el donador proporciona la información de contacto en el sobre. Tan pronto como se recibe el compromiso de donación, se envía una carta que agradece el donativo y da las gracias al donador. Suzanne es responsable del seguimiento con el empleador para recoger el donativo, que se paga en una sola exhibición por la corporación.

Durante el año se realizan muchos eventos para recaudar fondos. Suzanne solicita donativos en un carnaval de otoño, una cena-baile y un torneo de golf en primavera, entre otros eventos. Cada generación tiene un coordinador de clase que la ayuda a contactar a los miembros de su generación. El coordinador de clase envía una carta adicional para pedir donativos más grandes a partir de reuniones de generación, de aquellas que marcan un aniversario importante de graduación (ya sea cinco, 10 o más años) previos a su reunión de fin de semana de celebración. Cada primavera hay un fonotón durante el cual los estudiantes actuales y voluntarios llaman a otros potenciales donadores y solicitan garantías de donativos. Todos los alumnos que no han contribuido hacia el final de mayo reciben llamadas telefónicas de su coordinador de clase para pedirles un donativo. Si el coordinador de clase no puede contactar a sus compañeros de clase, Suzanne o un voluntario hacen estas llamadas.

Los donativos se categorizan por el grupo del que provienen, por el año del donador (si es aplicable) y por tamaño. Hay 10 "círculos de donadores", que se categorizan por el tamaño del donativo: Círculo del Presidente para los donativos mayores de 50 000 dólares, Círculo Platino para donativos de más de 25 000 dólares, etc. Los donativos menores a 100 dólares no se mencionan como pertenecientes a un círculo. Durante el verano se publica y envía a todos los donadores, reales y potenciales, un reporte anual que cita todos los donadores por categoría, año y círculo de donador. El reporte no menciona la cantidad real que aportó cada persona.

Necesidades de información

En el presente, Suzanne tiene una lista de correos en un procesador de palabras que se usa para generar etiquetas y cartas a potenciales donadores. A ella le gustaría poder personalizar cada carta al agregar una línea acerca de la cantidad de dinero que el donador dio el año anterior. Para seguir las garantías de donativo y las donaciones, se usa una hoja de cálculo. Las garantías cuantiosas de donadores individuales por lo general se pagan en cuotas mensuales en vez de un solo pago, pero en la actualidad no hay forma de rastrear dichos pagos. Cuando se desarrolle una base de datos, a Suzanne le gustaría poder enviar recordatorios si los pagos tienen un retraso de un mes.

Una forma de Donativo para Colecta Anual se envía con todas las cartas que solicitan fondos, con espacios en blanco para que el donador llene la información aplicable, del modo siguiente:

> **Donativo para la Colecta Anual de la Universidad Beta**. Nombre del donador, dirección del donador, categoría (una casilla de verificación que especifica graduado, alumno, padre, administrador, etc.), año de graduación, fecha de garantía/donativo, cantidad garantizada, cantidad enviada, método de pago, número de pagos elegidos, número de tarjeta de crédito, nombre de la corporación emisora, dirección de la corporación emisora, nombre del cónyuge (si el donativo que se aporta es del empleador del cónyuge).

Cuando los representantes de clase reciben las garantías, o durante el fonotón, se recoge la misma información en formas similares. Los reportes necesitan incluir:

1. **Reporte anual a donadores.** Este reporte se describió anteriormente. Menciona sólo nombres, no cantidades. Sin embargo, los nombres tienen que categorizarse como se indica. El reporte también tiene resúmenes, que incluyen la cantidad total recaudada de todas las fuentes, el total para cada clase, la participación porcentual de cada clase, el total para cada categoría, el gran total para cada círculo de donador y el total de clase por cada círculo donador. Es una importante herramienta de recaudación de fondos para el director del siguiente año, pues se envía por correo a cada donador potencial.
2. **Reporte mensual.** Éste es un reporte interno que Suzanne usa para evaluar el progreso de la recaudación de fondos para el año hasta la fecha. Proporciona los totales y porcentajes de garantías y donativos recibidos durante el mes actual en todas las categorías.
3. **Reporte de vencimiento de pagos.** A Suzanne le gustaría un reporte mensual que mencione los pagos de garantías que vencieron dicho mes pero que no se recibieron. Mencionaría el nombre y dirección del donador, la cantidad vencida, la fecha de vencimiento, la cantidad de la garantía, la cantidad recibida hasta el momento y la fecha del pago anterior, si lo hubiera.

4. **Reporte de eventos.** A Suzanne le gustaría generar reportes que muestren quién asistió a cada uno de los eventos de recaudación de fondos y qué garantías y donativos se recibieron por parte de los asistentes.
5. **Lista de contacto de los representantes de clase.** Para cada representante de clase, a Suzanne le gustaría una lista de compañeros de clase a contactar, incluidos nombre, dirección, número telefónico, información de donación del último año e información de donación de este año.
6. **Lista de contactos de voluntarios para el fonotón.** A cada llamador voluntario se le da una lista con información acerca de los donadores potenciales a los que se espera que llame, incluidos nombre, número telefónico, dirección, categoría, año (si es aplicable) e información de donación del último año.

Además de las formas y reportes mencionados aquí, existen muchos otros que serían útiles. Realice los pasos 1.1-1.4 con base en la información proporcionada y cualesquiera suposiciones adicionales aplicables que necesite hacer acerca de las operaciones de la Colecta Anual.

Proyecto dos: Grupo de teatro de la comunidad Pleasantville

Descripción general

El grupo de teatro de la comunidad Pleasantville es una organización no lucrativa de aproximadamente 200 miembros, aficionados que disfrutan producir y representar obras. Los miembros pagan cuotas de 50 dólares al año. El grupo produce dos obras cada año, pero no todos los miembros son trabajadores activos cada año.

Operaciones básicas

El grupo produce obras en otoño y primavera. Algunos miembros del grupo tienen papeles en las obras, mientras que otros trabajan en escenografía, vestuario, publicidad, programas y otras tareas. El grupo tiene dos reuniones generales por año. Cada otoño se reúnen para elegir oficiales: presidente, vicepresidente, secretario, tesorero y administrador, que funciona todo el año. Al final de la temporada de primavera, el grupo se reúne de nuevo para evaluar las actividades del año anterior y elegir las dos obras y sus productores para el año siguiente. El productor de cada obra es el responsable de administrar todos los aspectos de dicha producción, incluidos reclutamiento de voluntarios, promoción, reparto y más. A veces el grupo obtiene patrocinio de negocios locales para una producción, y siempre imprime un programa con publicidad que ayuda a sufragar algunos costos de la producción. El programa también menciona el reparto, personal y créditos del espectáculo. La mayor parte del trabajo de producción lo realizan los miembros, pero se contratan obreros calificados para tareas específicas según se requiera, como alambrado eléctrico. No se usan actores profesionales. Para un teatro, el grupo usa el auditorio de la preparatoria local, que tiene aproximadamente mil asientos.

Necesidades de información

El grupo quiere tener una base de datos para dar seguimiento a los miembros y las producciones. También necesitan compilar nombres y direcciones de potenciales aficionados al teatro (patronos) para que puedan enviarles anuncios acerca de cada producción, lo que ayuda a la venta de boletos. En el pasado usaban asientos abiertos, pero ahora quieren usar asientos asignados, pues el auditorio identifica los asientos con letras de filas y números de asiento. Existen 26 filas (A-Z) con 40 asientos por fila. Esto permitiría a la compañía tener

suscripciones con asientos asignados. La base de datos rastrearía las obras adecuadas para que tal compañía las produzca. También necesitan compilar información acerca de potenciales o antiguos patrocinadores corporativos. Algunas de las formas o reportes que resultarían útiles son:

1. **Lista de obras.** Las obras que podría producir la compañía se identifican por título, autor, tipo (drama, comedia, musical, etc.) y número de actos.
2. **Programa-Reparto y créditos.** El programa para cada producción debe mencionar los nombres de los actores y las labores que realizó cada miembro para la producción.
3. **Programa-Patrocinadores.** El programa debe mencionar a todas las corporaciones e individuos que donaron dinero, bienes o servicios para cada producción.
4. **Reporte de patronos.** Este reporte interno menciona información de correo para patronos, así como una lista de las producciones para las que compraron boletos en el pasado.
5. **Reporte de ventas de boletos.** Este reporte interno debe mencionar los boletos, junto con el precio y números de asiento, que los patronos ordenaron para las producciones.
6. **Boleto de admisión.** La base de datos debe ser capaz de imprimir boletos cuando un patrono los ordene. El boleto debe mencionar el nombre de la obra, la fecha, horario, precio y asiento.
7. **Reporte de pagos de cuotas de los miembros.** El tesorero necesita un reporte que muestre cuáles miembros pagaron sus cuotas y cuáles miembros todavía las adeudan. Debe proporcionar información de contacto para quienes todavía no pagan sus cuotas.
8. **Estados financieros.** El tesorero es responsable de mantener toda la información acerca de ingresos y gastos por el año. El ingreso proviene de pagos, patrocinios, venta de boletos y otros recursos. Los gastos incluyen costos para las producciones, como honorarios del contratista, renta de equipo, renta de auditorio y otros servicios. A final de año, el balance debe mostrar cuando mucho un beneficio modesto, pero nunca una pérdida. Por tanto, el tesorero debe poder reportar la situación financiera actual en cualquier momento, para que se puedan evaluar los gastos antes de gastar los fondos.
9. **Transacción de venta de boletos.** El proceso de venta de boletos requiere una transacción interactiva. El usuario debe poder ingresar una petición para uno o más asientos de una representación particular, y la pantalla de respuesta debe desplegar suficiente información para permitir al usuario determinar si los asientos están disponibles. Si es así, la transacción se debe completar mediante la reservación de los asientos e imprimir los boletos. Si no, debe dar posibilidad de encontrar asientos alternativos, si es que existen.

Además de las formas y reportes mencionados anteriormente, existen muchos otros que serían útiles. Realice los pasos 1.1-1.4 con base en la información proporcionada aquí y cualquier suposición adicional aplicable que necesite hacer acerca de las operaciones del grupo de teatro de la comunidad Pleasantville.

Proyecto tres: Distribuidor Autos Amistosos

Descripción general

Autos Amistosos es un distribuidor que ofrece automóviles nuevos de un solo fabricante. El distribuidor se ubica en un suburbio de una gran ciudad. Sus ventas brutas superan un millón de dólares anuales. Tiene 10 empleados: Jim Amistoso (dueño/gerente), ocho vendedores y un administrador. La mayoría de sus clientes son del área vecina y saben del distribuidor por comentarios personales; por comerciales en periódico, radio y televisión; por Internet, o por referencia de servicios de compra.

Operaciones básicas

Los clientes potenciales por lo general llegan personalmente a la sala de exhibición para observar y realizar pruebas de manejo de los autos. Ellos comparan la tienda y visitan a muchos distribuidores de varios fabricantes. Por lo general tienen una lista de las características que quieren y cierto conocimiento de los modelos que ofrece el distribuidor. Cuando entran a la sala de exhibición les da la bienvenida cualquier vendedor que esté libre. En pocos casos, especifican con cuál vendedor quieren tratar. Trabajan con un solo vendedor hasta que el trato está completo, porque todas las ventas se realizan sobre una base de comisión. En cada vehículo hay una calcomanía con el precio, que se destaca de forma prominente en la ventana lateral. Los clientes negocian con el vendedor para obtener un mejor precio. Si el precio propuesto está significativamente por abajo del precio de la calcomanía, el vendedor tiene que obtener la aprobación de Jim antes de acordar el trato. El financiamiento se puede arreglar con el fabricante a través del distribuidor, o el cliente puede obtener financiamiento por medio de su banco. Todos los impuestos y cargos de licencia se pagan a través del distribuidor. El cliente puede tener personalización adicional del auto, incluidos adornos especiales, sistema de alarma, sistema de audio y otras características que se realizan en el distribuidor antes de recoger el automóvil. Todos los autos nuevos llegan con una garantía estándar, pero los clientes pueden optar por una garantía extendida a un costo adicional. Se aceptan intercambios como pago parcial por automóviles nuevos. El distribuidor también vende estos vehículos en intercambio como autos usados, que pueden ser modelos de una variedad de fabricantes. Sobre los vehículos de intercambio no se hace mantenimiento; se venden "tal cual", con una garantía limitada de 30 días.

Necesidades de información

El distribuidor tiene un sistema de gestión de base de datos que en la actualidad sigue la pista de los automóviles e información de ventas. Sin embargo, Jim quiere desarrollar una nueva base de datos que pueda proporcionar más información de manera más eficiente que el sistema actual. El sistema actual almacena información acerca de los automóviles, los clientes, los vendedores y las ventas de autos. Se usan las siguientes formas y reportes:

1. **Calcomanía de precio.** La calcomanía de precio que viene con el automóvil cuando se embarca del fabricante contiene toda la información básica acerca del automóvil. Incluye una ID de vehículo que identifica de manera única al vehículo y está físicamente incrustada en la carrocería del automóvil. La calcomanía también proporciona el precio de lista, modelo, fecha de fabricación, lugar de fabricación, número de cilindros, número de puertas, peso, capacidad, opciones, color y otras especificaciones. El distribuidor agrega la fecha cuando se entregó el automóvil y el kilometraje al momento de la entrega.

2. **Datos del cliente.** El vendedor obtiene información de contacto básica del cliente cuando le da la bienvenida a la sala de exhibición. Cuando se realiza una venta se recopila información de cliente adicional. Jim también busca reunir nombres y direcciones de potenciales clientes mediante referencias, tarjetas de respuesta de periódicos y revistas, y otras fuentes. Éstas se usan para enviar por correo material promocional a clientes en potencia.
3. **Licencia, impuestos y documentos de seguro.** El distribuidor necesita enviar información al Estado acerca de cada venta antes de emitir las placas del automóvil. También deben remitir directamente al Estado el impuesto por ventas estatal y el costo de la licencia por cada venta. Se les requiere obtener y enviar pruebas de la cobertura del seguro al Estado antes de liberar el automóvil al nuevo propietario.
4. **Factura de venta.** Cuando el automóvil se entrega al cliente, se le proporciona a éste una factura de venta completa (que muestra la información del cliente, nombre del vendedor, ID del vehículo, kilometraje actual y todas las especificaciones, incluida cualquier personalización adicional, financiamiento, información de garantía, información de licencia y seguro, precio y otros detalles más), y una copia se conserva en la distribuidora. Esta factura de venta es la misma, ya sea que el auto sea nuevo o usado.
5. **Reporte de desempeño de vendedor.** A Jim le gustaría un reporte mensual que resuma las ventas de cada vendedor durante el mes anterior. La cantidad de comisión que gana también se muestra en el reporte.
6. **Encuesta de satisfacción del cliente.** Dentro de un mes posterior a cada venta, el distribuidor envía una encuesta al nuevo propietario, donde plantea preguntas acerca de la opinión que el cliente tiene del automóvil, el distribuidor y el vendedor.

Además de estas formas y reportes, existen muchos otros que serían útiles. Realice los pasos 1.1-1.4 con base en la información proporcionada aquí y cualquier suposición adicional aplicable que necesite hacer acerca de las operaciones de Autos Amistosos.

Proyecto cuatro: Estudio Imágenes Fotográficas

Descripción general

El Estudio Imágenes Fotográficas es un pequeño negocio que proporciona servicios de fotografía a clientes individuales y corporativos. Los servicios incluyen fotografía de bodas, graduaciones, ceremonias de premiación, conferencias empresariales, recepciones y otros eventos. El estudio también ofrece sesiones para retratos individuales, familiares o grupales, que se toman en el estudio o en una locación especificada por el cliente. El estudio fotografía alrededor de 200 eventos y toma casi 1 000 retratos por año. El personal consiste en la gerente/propietaria, Liz Davis, quien es fotógrafa profesional, cinco fotógrafos de planta y un encargado administrativo.

Operaciones básicas

El cliente usualmente contacta al estudio para hacer una cita para reunirse con Liz o su representante. En la primera reunión, el representante muestra ejemplos del trabajo del estudio y responde las preguntas que tenga el cliente. Éste proporciona información, incluidos los servicios deseados, locación, fecha, horario y el nombre del fotógrafo solicitado, si hay alguno. La mayoría de los eventos requieren dos fotógrafos, un principal y un asistente, pero los retratos sólo requieren uno. Además de los seis fotógrafos regulares, el estudio mantiene una lista de fotógrafos independientes para usar en los eventos cuando los fotó-

grafos de planta estén agendados o no disponibles. El representante proporciona una estimación y hace una reservación tentativa. Después de la reunión inicial, se prepara un contrato y se envía por correo al cliente para que lo firme. El cliente regresa el contrato firmado con un depósito y se concreta la reservación. Los fotógrafos cubren el evento o sesión, se revela la película y se producen las pruebas. A cada prueba se le asigna un número de identificación único y al cliente se le presenta un paquete de pruebas. El cliente selecciona las imágenes que desea y realiza el pedido final junto con alguna instrucción especial deseada, como retoque. Se producen las imágenes o álbumes y el paquete final se entrega al cliente.

Los pagos se realizan por trabajos en varias etapas. Por lo general, se entrega un depósito en el momento de la reservación, y los pagos adicionales se efectúan el día del evento o sesión, a la presentación de las pruebas al cliente, y cuando se entrega el paquete final. Están disponibles muchas opciones de paquete, incluidas combinaciones de imágenes de varios tamaños, muchos tipos de álbumes y paquetes digitales. Los paquetes se describen en un cuadernillo impreso y se identifican mediante número. El paquete final puede diferir del original solicitado, de modo que el último pago necesita ajustarse en concordancia. En el caso de que el cliente no esté complacido con las pruebas, tiene la opción de rechazar un paquete final, pero el depósito y los pagos por la sesión no se regresan. Los clientes conservan las pruebas, pero el estudio posee el copyright de las imágenes y guarda todos los negativos y archivos digitales durante seis meses, tiempo en el cual el cliente puede ordenar fotografías adicionales. Al final de los seis meses se desechan los negativos y archivos, a menos que el cliente solicite tiempo adicional.

Necesidades de información

La compañía actualmente conserva registros a mano, pero su negocio ha crecido lo suficiente como para necesitar la ayuda de una base de datos para controlar sus operaciones. El actual sistema manual es poco manejable e ineficiente, y la propietaria quiere desarrollar un sistema de base de datos que el encargado administrativo sea capaz de mantener. El sistema se usará para conservar información de clientes, trabajos y fotógrafos. No incluirá información acerca de suministros, equipo, gastos de oficina o nómina. Las formas usadas para proporcionar información son:

1. **Forma de consulta.** Este documento se llena cuando el cliente se reúne con el gerente. Menciona ítems como información de contacto, servicios solicitados y paquete elegido. Durante la entrevista, el gerente comprueba qué fotógrafos están disponibles en la fecha solicitada y elige uno para ponerlo en la forma. Las entradas se consideran tentativas y sujetas a cambio antes de redactar un contrato.
2. **Contrato.** El contrato contiene datos de la forma de consulta, así como el nombre del fotógrafo asignado al trabajo, datos de pago planeados y cualquier solicitud adicional del cliente. Cada forma de contrato tiene un número único y contiene algún material preimpreso, como el nombre y dirección del estudio, y noticias concernientes a cancelación, responsabilidad y copyright.
3. **Forma de orden de paquete.** La forma de orden de paquete se llena cuando el cliente selecciona las pruebas y decide el paquete final. Si el cliente ordena imágenes o álbumes adicionales durante el periodo de seis meses después de la orden final, se llena una forma de orden adicional. Cada forma de orden tiene un número único.

   Se necesitan los siguientes reportes:

4. **Horario del fotógrafo.** Para cada fotógrafo se imprime un horario para cualquier periodo deseado, por lo general una semana o un mes. El horario proporciona información básica acerca de los eventos o sesiones agendados y se menciona el número

de contrato, que el fotógrafo puede utilizar para obtener información completa acerca de cada evento o sesión calendarizados.

5. **Horario semanal.** El horario semanal resume las actividades agendadas para cada día de la semana, de todos los fotógrafos. Para cada día, menciona las actividades en orden cronológico. El reporte se puede correr cualquier semana deseada, no sólo para la semana actual.
6. **Cuentas por cobrar.** Este reporte resume los pagos vencidos cada mes.
7. **Reporte de cliente.** Este reporte se puede ejecutar como se desee para proporcionar información acerca de clientes individuales. Por lo general se corre para clientes corporativos, con el fin de proporcionar un resumen de los servicios que se les ofrecen.
8. **Transacción de disponibilidad de fotógrafo.** La base de datos debe ser capaz de soportar una transacción en la que el usuario ingrese el nombre del fotógrafo y la fecha, y la pantalla de salida dice las horas que está disponible en dicha fecha.

Además de las formas y reportes aquí mencionados, existen muchas otras que serían útiles. Realice los pasos 1.1-1.4 con base en la información proporcionada aquí y cualquier suposición adicional aplicable que necesite hacer acerca de las operaciones del Estudio Imágenes Fotográficas.

Proyecto cinco: Grupo Médico Clínica Bienestar

Descripción general

La Clínica Bienestar es una instalación que proporciona atención médica en áreas rurales del país. Su personal profesional consiste en cinco médicos, dos enfermeras practicantes que proporcionan atención no especializada y pueden prescribir medicamentos, dos enfermeras registradas, dos parteras que proporcionan atención prenatal y supervisión de parto excepto en casos con complicaciones, un farmacéutico y un técnico médico. Los miembros del personal no profesional incluyen un administrador de oficina, una recepcionista y un contador que trabaja tiempo parcial. La clínica atiende a varios miles de pacientes, quienes pueden visitar la clínica varias veces por año, tanto para cuidado preventivo como son chequeos o inmunizaciones, y para tratamiento de enfermedades. Las instalaciones consisten de una sala de espera con un escritorio de recepción, una oficina administrativa, una estación de enfermeras, 10 salas de exploración con salas de consulta adjuntas, una pequeña sala de operaciones, una sala de partos, una sala de recuperación, una farmacia y un pequeño laboratorio.

Operaciones básicas

La clínica tiene horario regular de operación los días de semana, los sábados en la mañana y dos tardes por semana. Por lo general, durante las horas regulares, en la clínica están dos médicos o un médico y una enfermera practicantes, una enfermera registrada y una partera. Además, los miembros del personal profesional rotan responsabilidades para cubrir llamadas de emergencia 24 horas al día, siete días a la semana. Al final de cada día, el administrador o recepcionista configuran la desviación de llamadas de modo que las llamadas de emergencia se dirijan automáticamente al número telefónico de la persona que proporciona la cobertura de emergencia. Cuando la clínica abre en la mañana, se detiene la desviación de llamadas. Dos de los médicos son cirujanos que realizan cirugía de rutina que no requiere anestesia general en la clínica una mañana a la semana, auxiliados por una enfermera. Otros tienen especialidades en pediatría y medicina interna. Sin embargo, todos los médicos pueden proporcionar cuidado general y de urgencia a cualquiera de los pacien-

tes. Los pacientes que requieren cirugía mayor u otra atención hospitalaria deben acudir a un hospital ubicado fuera del área inmediata atendida por la clínica. Los miembros del personal clínico por lo general no visitan a sus pacientes que están en el hospital, y en vez de ello dejan su atención al personal del hospital con quien la clínica se comunica durante la hospitalización. No obstante, la clínica proporciona atención tanto pre como poshospitalaria para los pacientes.

Las horas de operación se dividen en citas calendarizadas y horas no agendadas que están abiertas para quien ingrese. Por lo general, los pacientes solicitan con mucha anticipación citas para chequeos e inmunizaciones. Los pacientes que sufren de enfermedades crónicas o agudas usualmente pueden solicitar cita en el momento, o pueden llegar durante las horas no agendadas. El administrador es responsable de establecer todos los horarios, tanto para el personal como para los pacientes, y de mantener actualizados los registros. Antes de comenzar cada mes, el administrador hace horarios de cobertura completos para todo el personal. El contador es responsable de elaborar toda la facturación y registrar los pagos. La recepcionista es responsable de hacer citas, dirigir el tráfico y hacer que los registros médicos del paciente estén disponibles en una carpeta durante la visita. La enfermera prepara al paciente, toma la historia médica, realiza algunas rutinas o pruebas médicas, toma muestra para pruebas de laboratorio, actualiza la carpeta y auxilia al practicante (médico, enfermera practicante o partera) durante la visita. En la visita, el practicante examina al paciente, administra tratamiento médico, puede realizar algunas pruebas, tomar muestras para pruebas de laboratorio y escribir prescripciones para medicamentos u órdenes para pruebas de laboratorio adicionales. Cada visita resulta en uno o más diagnósticos, que el practicante agrega a la carpeta del paciente, junto con cualquier comentario u observación. Las recetas se pueden surtir en la farmacia de la clínica a solicitud del paciente. Algunas pruebas de laboratorio las realiza en la clínica el técnico médico, con las muestras tomadas por alguno de los profesionales. Las pruebas más especializadas se realizan en un laboratorio médico en el hospital fuera de la región. Siempre que es posible, los especímenes, como muestras de sangre, se toman en la clínica y luego se envían al laboratorio del hospital. Si la prueba de laboratorio requiere la presencia del paciente y equipo que no está disponible en la clínica, se envía al paciente al laboratorio del hospital para la prueba, y los resultados se envían de vuelta a la clínica.

El cuidado médico se proporciona a todos los pacientes, sin importar su capacidad para pagar. Las facturas se generan con base en los servicios proporcionados, no sobre el método de pago. Los pacientes privados que pueden costear el pago de su bolsillo pueden hacerlo al momento del servicio o cobrarles a fin de cada mes. Quienes tienen seguro médico proporcionan información acerca de las pólizas y se cobra a las compañías aseguradoras. Por lo general, en este caso, los pacientes pagan una pequeña cantidad como deducible, que se determina mediante el tipo de póliza que tienen, al momento de la visita. Quienes no pueden costear el pago por lo general tienen atención médica proporcionada por el gobierno, para lo cual tienen una tarjeta médica emitida por el Estado. Ellos no pagan nada y el gobierno reembolsa a la clínica todo el costo de la visita, incluida cualquier prueba de laboratorio realizada y medicamentos surtidos ahí. Se atiende a un pequeño número de pacientes indigentes que no tienen cobertura de salud y la clínica absorbe el costo hasta que el paciente califica para cobertura proporcionada por el gobierno.

Necesidades de información

En la actualidad toda la información acerca de los pacientes y su atención se llevan de forma manual, y la facturación se realiza usando una hoja de cálculo que se conserva en una computadora personal.

Los médicos usan comunicaciones estándar por correo, fax o teléfono para proporcionar información al hospital y recibir información acerca de pacientes que necesitan atención hospitalaria. Recientemente la clínica actualizó su computadora y tendrá acceso a registros de hospital para sus pacientes, así como sistema en línea que proporcionan las compañías aseguradoras y el gobierno para facturación a terceras partes. La clínica necesita una base de datos que dé seguimiento a todas las actividades de la clínica relacionadas con el paciente, y que proporcione información acerca de facturación y pagos. La base de datos no seguirá los suministros médicos, mantenimiento de planta o información de nómina.

Se necesitan los siguientes reportes o formas.

1. **Horario de cobertura semanal.** Este horario necesita mencionar las horas diarias y el personal profesional y no profesional que están calendarizados para estar en la clínica en horas específicas cada día de la semana. También necesita mencionar el nombre y número telefónico de la persona que cubre las emergencias durante todas las horas cada semana (recuerde que el administrador proporciona la información de cobertura cada mes).
2. **Horario maestro diario.** Éste es un horario maestro para todos los practicantes de cada día. Debe mencionar cada uno de los practicantes que están en dicho día, con todas las citas de paciente agendadas. A la mayoría de las citas se les asignan 10 minutos, de modo que cada hora tiene seis lugares de tiempo. Sin embargo, a algunas citas se les da más de un lugar de tiempo, dependiendo de la naturaleza de la atención requerida. Cada profesional tiene horas dedicadas a los pacientes externos durante las cuales no se asignan citas preagendadas. Conforme los externos ingresan para atención, se les asigna un practicante y el nombre del paciente se agrega al horario. Las enfermeras registradas no tienen citas agendadas y están disponibles para auxiliar a los practicantes con las visitas, o para administrar pruebas o tomar muestras sobre una base sin horario. El técnico de laboratorio tampoco tiene un horario de citas.
3. **Horario diario de practicante individual.** Cada uno de los practicantes debe recibir una copia impresa individualizada del horario para cualquier día que esté en la clínica. Las citas mencionan el nombre del paciente y la razón que da para la visita. La enfermera actualiza manualmente la copia, conforme se realizan las visitas para los pacientes externos.
4. **Formas de declaración para seguro del médico.** Ésta es una forma preimpresa que se usa como recibo principalmente para propósitos de seguro. Menciona el nombre, dirección y número telefónico de la clínica, junto con los nombres y números de registro federal de contribuyentes de todos los profesionales en el personal. También menciona todos los tipos de visitas, los procedimientos que se pueden realizar con un código para cada uno y algunas líneas en blanco para "otro", junto con una línea para ingresar la tarifa para cada uno. También tiene una lista de los diagnósticos y códigos comunes, con algunas líneas en blanco para "otro". En el fondo hay líneas para Cargo total, Cantidad pagada y Saldo. El proveedor usa esta forma durante una visita para registrar tipo de visita, procedimientos realizados y diagnóstico. Cuando el paciente sale después de la visita, la recepcionista llena la tarifa para cada servicio usando una tabla de tarifas, calcula el total y escribe la cantidad pagada, si hay alguna, y el saldo. Una copia se mantiene en la clínica y otra se entrega al paciente. En la actualidad, una tercera copia se envía por correo a la compañía aseguradora o agencia de salud gubernamental, pero en el futuro la información requerida se enviará electrónicamente.
5. **Estado de cuenta mensual del paciente.** Cualquier paciente que tenga un saldo sin pagar recibe un estado de cuenta que se compila al final de cada mes, que menciona

todos los servicios proporcionados dicho mes, cualquier pago recibido y el saldo insoluto.

6. **Etiqueta de prescripción y receta.** Esta forma consta de dos partes. La parte superior está engomada y se usa como etiqueta para el contenedor en la que se dispensa el medicamento. La etiqueta muestra número de receta, nombre del médico, nombre del paciente, dirección del paciente, indicaciones, nombre del fármaco, forma, potencia, cantidad, nombre del farmacéutico, fecha de llenado, fecha original y número de recargas restantes. La parte inferior repite la información de la etiqueta y también menciona el precio total del medicamento, la cantidad cubierta por el seguro o el gobierno, y el saldo del paciente, así como más información acerca del medicamento, indicaciones completas para su uso y advertencias acerca de posibles efectos colaterales e interacciones medicamentosas. La receta se puede usar con el fin de emitir reclamaciones para la cobertura de seguro. En el futuro, esta información también se enviará electrónicamente a las compañías aseguradoras y a la agencia de atención médica gubernamental.
7. **Bitácora de laboratorio diaria.** Esta bitácora se usa para registrar todas las pruebas de laboratorio realizadas cada día.
8. **Horario de sala de operaciones.** Este horario proporciona información acerca de todas las cirugías programadas para el día.
9. **Bitácora de la sala de operaciones.** Ésta registra información acerca de las cirugías que realmente se realizaron en un día dado, incluidos identificación del paciente, cirugía y enfermera, y anotaciones y observaciones acerca de la cirugía.
10. **Bitácora de sala de partos diaria.** Ésta registra información acerca de todos los partos realizados cada día.
11. **Bitácora de sala de recuperación.** Este reporte registra información acerca del uso de la sala de recuperación, incluidos nombre del paciente, practicante que lo atendió, cama, fecha de entrada, hora de entrada, fecha de salida, hora de salida y firma del practicante que da de alta al paciente. Una enfermera registra las horas y resultados de cualquier chequeo médico realizado mientras el paciente está en recuperación.
12. **Reporte de actividad mensual.** Éste es un reporte interno que resume la actividad de la clínica cada mes. Muestra ítems como el número de visitas realizado por cada proveedor, el número de cirugías realizadas, el número de partos, el número de pruebas de laboratorio desglosadas por tipo, el número de recetas surtidas, el tiempo promedio por visita, etcétera.

Éstos son sólo algunos de los muchos reportes y formas que serían útiles para el personal de la clínica. Además de las formas y reportes mencionados aquí, existen muchos otros que serían útiles. Realice los pasos 1.1-1.4 con base en la información proporcionada aquí y cualquier suposición adicional aplicable que necesite hacer acerca de las operaciones de la Clínica Bienestar.

CAPÍTULO 2

Planificación y arquitectura de las bases de datos

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Por qué los datos se ven como un recurso corporativo
- La distinción entre datos e información
- Los cuatro niveles de discusión acerca de los datos
- El significado de los siguientes términos: entidad, conjunto entidad, atributo, relación
- El significado de los términos: metadatos, tipo de registro, tipo ítem de datos, datos agregados, registro, diccionario de datos, instancia de datos y archivo
- Los pasos en el diseño escalonado de bases de datos
- Cómo se construye y usa un diccionario de datos
- Las habilidades y funciones de un administrador de bases de datos
- Las funciones de un sublenguaje de datos y un lenguaje huésped
- La distinción entre lenguaje de definición de datos (DDL) y lenguaje de manipulación de datos (DML)
- Las razones y contenidos de la arquitectura de base de datos en tres niveles

- El significado de la independencia lógica y física de los datos
- Las características de varios modelos de datos: entidad-relación, relacional, orientada a objeto, objeto-relacional y semiestructurada

2.1 Exploración de una herramienta de diagramación

2.2 Exploración de una herramienta de gestión de proyecto

2.3 Construcción de un diccionario de datos simple

PROYECTO DE MUESTRA: Aplicación de técnicas de planificación al proyecto de Galería de Arte

PROYECTOS ESTUDIANTILES: Aplicación de las técnicas de planificación a los proyectos estudiantiles

2.1 Los datos como un recurso

Si se le pide identificar los recursos de una organización empresarial común, probablemente incluiría equipo de capital, activos financieros y personal, pero es posible que no piense en los datos como un recurso. Cuando existe una base de datos corporativa, los datos que contiene son un genuino recurso corporativo. Puesto que la base de datos contiene datos acerca de las operaciones de la organización (llamados **datos operativos**) que utilizan muchos departamentos, y dado que la maneja profesionalmente un ABD, existe un creciente aprecio por el valor de los datos en sí, independiente de las aplicaciones que los usen. Un **recurso** es cualquier activo que es de valor para una organización y que incurre en costos. Los datos operativos de una organización claramente encajan con esta definición. Para apreciar el valor de los datos de una organización con más amplitud, imagine lo que ocurriría si los datos se perdieran o cayeran en manos de un competidor. Muchas organizaciones, como los bancos y las casas de corretaje, dependen enormemente de los datos, y fracasarían rápidamente si los perdieran. La mayoría de las empresas sufrirían grandes pérdidas si sus datos operativos no estuvieran disponibles. De hecho, una organización depende de la disponibilidad de los datos operativos para administrar sus otros recursos. Por ejemplo, las decisiones acerca de compras, rentas o uso de equipo, inversiones y rendimientos financieros, y necesidades de personal se deben realizar sobre la base de información acerca de las operaciones de la organización. El reconocimiento de los datos como un recurso corporativo es un importante objetivo en el desarrollo de un entorno de base de datos. La base de datos protege los recursos de datos al proporcionar controles de seguridad, integridad y confiabilidad de datos mediante el DBMS.

2.2 Características de los datos

Con la finalidad de apreciar la importancia de los datos como un recurso corporativo, es necesario examinar sus características con más detalle.

2.2.1 Datos e información

Con frecuencia se piensa en los datos como información, pero estos dos términos tienen significados ligeramente diferentes. El término **datos** se refiere a los hechos brutos registrados en la base de datos. Pueden ser ítems acerca de personas, lugares, eventos o conceptos. La **información** consiste en datos procesados que están en una forma que es útil para tomar decisiones. La información se deriva de los datos almacenados al reordenar, seleccionar, combinar, resumir o realizar otras operaciones sobre los datos. Por ejemplo, si simplemente se imprimen todos los ítems almacenados en la base de datos que se muestra en la figura 1.1, sin identificar lo que representan, se tienen datos. Sin embargo, si se imprime un repor-

te formateado como el de la figura 1.3, que muestra los datos en cierto orden que ayuda a tomar una decisión, se tiene información. En la práctica, la mayoría de las personas usan los dos términos de manera intercambiable.

2.2.2 Niveles de discusión de datos

Cuando se discuten datos, es importante distinguir entre el mundo real, la pequeña parte del mundo real por la que se preocupa la base de datos, la estructura de la base de datos y los datos almacenados en la base de datos. En realidad existen cuatro niveles de discusión o abstracción a considerar cuando se habla acerca de bases de datos.

Comience con el **mundo real** o realidad. En este nivel, se habla de la **empresa**, la organización para la que se diseña la base de datos. La empresa puede ser una corporación, agencia gubernamental, universidad, banco, casa de corretaje, escuela, hospital u otra organización que en verdad exista en el mundo real. Conforme la organización funciona en este mundo, es imposible obtener información necesaria para la toma de decisiones mediante observación directa de la realidad. Hay demasiados pormenores involucrados para seguir la pista de todos los detalles. En vez de ello, se desarrolla un modelo o una visión de la realidad en la que se representan dichas facetas de la empresa que es necesario seguir para la toma de decisiones. La parte del mundo real que se representará en la base de datos se llama **minimundo** o **universo de discurso**. Para el minimundo, comience por desarrollar un **modelo conceptual**, que forma el segundo nivel de discusión de datos. Las **entidades** se identifican como representaciones de personas, lugares, eventos, objetos o conceptos acerca de los que se recopilan datos. Para las organizaciones mencionadas anteriormente, se podrían elegir entidades como clientes, empleados, estudiantes, cuentas bancarias, inversiones, clases o pacientes. Las entidades similares se agrupan en **conjuntos de entidad**. Por ejemplo, para el conjunto de todos los clientes se forma el conjunto de entidad que se puede llamar Clientes. De igual modo, se pueden tener conjuntos de entidad llamados Empleados, Estudiantes, Cuentas, Inversiones, Clases y Pacientes, y cada uno consiste en todas las instancias de entidad del tipo correspondiente. Un modelo conceptual puede tener muchos conjuntos de entidad. Cada entidad tiene ciertos **atributos**, que son características o propiedades para describir la entidad y que la organización considera importante. Cada conjunto de entidad puede tener muchos atributos para describir sus miembros. Para el conjunto de entidad Estudiante, los atributos pueden incluir ID de estudiante, nombre, dirección, número telefónico, especialidad, créditos aprobados, calificación promedio y consejero. Para el conjunto entidad Cuenta Bancaria, los atributos pueden incluir número de cuenta, fecha de apertura, nombre del propietario, nombre del copropietario y saldo. Algunas entidades pueden tener **relaciones** o asociaciones con otras entidades. Por ejemplo, en una universidad, los estudiantes se relacionan a clases al inscribirse en dichas clases, y los miembros del personal docente se relacionan a clases al enseñarlas. Los estudiantes y los miembros del personal docente pueden relacionarse mutuamente mediante la relación profesor-estudiante o mediante la relación consejero-estudiante. Los estudiantes también se pueden relacionar mutuamente mediante compañeros de cuarto. El modelo conceptual representará estas relaciones al relacionar los conjuntos de entidad en alguna otra forma. Los conceptos de entidad, atributo y relación se discutirán con más detalle en el capítulo 3. La base de datos debe diseñarse para ser un modelo útil de la organización y sus operaciones para el minimundo de interés. Debe representar cada entidad, junto con sus atributos y las relaciones en las que participa. En el mundo real, los cambios se hacen a los objetos representados por las entidades, atributos o relaciones. Por ejemplo, los empleados dejan la organización, los clientes cambian sus direcciones y los estudiantes se inscriben en diferentes clases. Para seguir la pista de los hechos acerca de dichos objetos y tales cambios, es necesario desarrollar un modelo conceptual que permita no sólo la representación de las entidades, atributos y rela-

ciones básicas, sino también que permite hacer cambios que reflejen los cambios en la realidad.

La estructura de la base de datos, llamado **modelo lógico** de la base de datos, es el tercer nivel de discusión. En este nivel se habla acerca de **metadatos**, o datos acerca de datos. Para cada conjunto de entidad en el modelo conceptual, se tiene un **tipo de registro** en el modelo lógico de la base de datos. Por ejemplo para el conjunto de entidad Estudiante en la universidad, se tendría un tipo de registro `Student`. Un tipo de registro contiene muchos **tipos ítem de datos**, cada uno de los cuales representa un atributo de una entidad. Para el tipo de registro `Student`, los tipos ítem de datos podrían ser `stuId`, `stuName`, `address`, `phone`, `major`, `credits`, `gpa` (promedio) y `adviser` (consejero). Un **ítem de datos** es la unidad más pequeña nominada de los datos almacenados. Otras palabras que a veces se usan para los ítems de datos son elemento de datos, campo o atributo. Por lo general, **campo** significa un conjunto de bytes adyacentes identificados como la ubicación física para un ítem de datos, así que tiene un significado más físico que el término ítem de datos. **Atributo** por lo general se refiere a una característica de una entidad en el modelo conceptual, pero, dado que hay una correspondencia entre atributos e ítems de datos, con frecuencia las dos palabras son intercambiables. Los ítems de datos en ocasiones se agrupan para formar **agregados de datos**, que son grupos nominados de ítems de datos dentro de un registro. Para un registro `Empleado`, puede haber un agregado de datos llamado `empAdd`, que consiste en los ítems de datos `street`, `city`, `state` y `zip`. Los agregados de datos permiten hacer referencia al grupo de ítems de datos como un todo o a los ítems individuales en el grupo. Un **registro** es una colección nominada de ítems de datos relacionados y/o agregados de datos. Como se mencionó antes, usualmente hay un tipo de registro para cada conjunto de entidad, y un tipo ítem de dato para cada atributo. Las relaciones también se pueden representar mediante tipos de registro, pero hay otras formas para representar relaciones.

La información acerca de la estructura lógica de la base de datos se almacena en un **diccionario de datos**, también llamado **directorio de datos** o **catálogo del sistema**. Este depósito de información contiene descripciones de los tipos de registro, tipos ítem de datos y agregados de datos en la base de datos, así como otra información. Por ejemplo, el diccionario/directorio de datos puede contener una entrada para el tipo de registro `Empleado`, y establece que consiste en los ítems de datos `empId`, `jobTitle` (nombre del puesto), `salary`, `dept` y `mgr`, y los agregados de datos `empName` y `empAdd`. Para cada uno de los ítems de datos, habría una entrada descriptiva que muestre el nombre de ítem de datos (por ejemplo, `empId`), su tipo de datos (por ejemplo, `char(5)`) y cualquier **sinónimo**, que son otros nombres utilizados para el ítem de datos (por ejemplo, `emp#`). Para agregados de datos, el diccionario/directorio mencionaría los componentes. Por ejemplo, para `empAdd`, establecería que sus componentes son los ítems de datos `street`, `city`, `state` y `zip`, cada uno de los cuales se mencionaría como ítem de datos. El diccionario/directorio de datos es en realidad una base de datos acerca de la base de datos. Sin embargo, los diccionarios de datos por lo general hacen mucho más que simplemente almacenar la descripción de la estructura de base de datos. Para algunos sistemas están involucrados de manera activa en todos los accesos de la base de datos.

El cuarto nivel de discusión tiene que ver con los datos reales en la base de datos en sí. Consiste en **instancias de datos** u ocurrencias. Para cada objeto en el minimundo que se representa como una entidad, habrá una ocurrencia de un registro correspondiente en la base de datos. Para cada estudiante en la universidad, hay una ocurrencia de un registro de estudiante. De este modo, mientras sólo hay un tipo de registro `Student`, que se describe en el diccionario de datos y corresponde al conjunto de entidad Student, puede haber miles de ocurrencias de registro `Student`, que corresponden a entidades estudiante individuales, en la base de datos en sí. De igual modo, habrá muchas instancias de cada uno de los tipos

Reino	Objetos	Ejemplos
Mundo real que contiene minimundo	Empresa	Corporación, universidad, banco
	Algunos aspectos de la empresa	Recursos humanos Inscripción de estudiante Clientes y cuentas
Modelo conceptual	Entidad Atributo Conjunto de entidad Relación	Un estudiante, un nombre de clase, horarios de todos los estudiantes, todas las clases de entidad Student se relacionan con la entidad Class al inscribirse en ellas
Modelo lógico Metadatos: definiciones de datos, almacenados en Diccionario de datos	Tipo de registro	Tipo de registro Student, tipo de registro Class
	Tipo ítem de datos	stuId, classNumber
	Agregado de datos	Dirección, que consiste en calle, ciudad, estado, CP
Ocurrencias de datos almacenados en la base de datos	Ocurrencia de registro Student	Registro del estudiante Tom Smith
	Ocurrencia de ítem de datos	'S1001', Smith', 'Tom', 'History',90
	Archivo	Archivo Student con 5 000 registros Student
	Base de datos	Base de datos University, que contiene el archivo Student, archivo Class, archivo Faculty, . . .

FIGURA 2.1

Cuatro niveles de discusión de datos

de ítem de datos que corresponden a atributos. Un **archivo** (a veces llamado conjunto de datos) es una colección nominada de ocurrencias de registro. Por lo general un archivo consiste en todas las ocurrencias de un tipo de registro. Por ejemplo, el archivo `Student` puede contener 5 000 registros `Student`. Finalmente, la **base de datos** se puede considerar como una colección nominada de archivos relacionados. La figura 2.1 resume los cuatro niveles de discusión de datos.

2.2.3 Sublenguajes de datos

El lenguaje que se usa para describir una base de datos a un DBMS es parte de un **sublenguaje de datos**. Un sublenguaje de datos consiste en dos partes: un **lenguaje de definición de datos** (DDL) y un **lenguaje de manipulación de datos** (DML). El DDL se usa para describir la base de datos, mientras que el DML se usa para procesar la base de datos. Estos lenguajes se llaman sublenguajes de datos porque los lenguajes de programación de propó-

sito general se extendieron para proporcionar operaciones de bases de datos, de modo que los comandos para definición y manipulación de objetos de base de datos forman un subconjunto del lenguaje de programación en sí, que se llama **lenguaje huésped**. Existen estándares para C, C++, C#, Java, COBOL, Fortran y Ada como lenguajes huésped para un sublenguaje de datos estándar llamado SQL. Sin embargo, muchos sistemas de gestión de bases de datos tienen sus propios sublenguajes únicos que no se conforman a estándar alguno, y no todos los lenguajes de propósito general tienen extensiones de base de datos. En muchos sistemas de base de datos no hay una conexión cercana entre el lenguaje huésped y el sublenguaje de datos, de modo que comandos en el sublenguaje de datos se marcan, remueven del programa en lenguaje huésped y se sustituyen por llamadas de subrutinas antes de la compilación del programa. Luego se compilan mediante el DBMS, se colocan en un módulo objeto y el módulo objeto se ejecuta en el momento adecuado. Cuando los comandos del sublenguaje de datos ocurren como parte de un programa en un lenguaje huésped, se dice que el sublenguaje está **embebido** (embedded) en el lenguaje huésped. Además, la mayoría de los sublenguajes de datos no permite empotramiento o comandos interactivos para acceso desde estaciones de trabajo.

2.3 Etapas en el diseño de bases de datos

El proceso de analizar la organización y su entorno, desarrollar un modelo de base de datos que refleje con precisión el funcionamiento de la organización en el mundo real y la implementación de dicho modelo mediante la creación de una base de datos requieren una metodología adecuada. Los análisis de sistemas tradicionales proporcionan un posible abordaje, pero un abordaje de diseño escalonado de base de datos ofrece una mejor solución.

2.3.1 Abordaje de análisis de sistemas

Un proyecto de diseño e implementación de base de datos se podría ver como un proyecto de desarrollo de sistemas. Tradicionalmente, los sistemas de software se desarrollaron usando un abordaje de análisis de sistemas que identifica los pasos en el diseño y la implementación de un sistema. Existe una suposición de que cada sistema tiene un **ciclo de vida**, un periodo durante el cual el sistema se diseña, crea, usa y luego se sustituye por un nuevo sistema. Un ciclo de vida típico se extiende durante muchos años y consiste en las etapas que se muestran en la figura 2.2. Al usar el abordaje del ciclo de vida tradicional, el sistema a la larga fallará para satisfacer las necesidades del usuario, se identificarán problemas y el ciclo comenzará de nuevo.

2.3.2 Abordaje de diseño escalonado de base de datos

Una suposición básica detrás del abordaje del ciclo de vida del análisis de sistemas es que los sistemas finalmente se volverán obsoletos y tendrán que sustituirse. En el entorno de base de datos hay razón para cuestionar esta suposición. La base de datos se puede diseñar en tal forma que pueda evolucionar y cambiar para satisfacer futuras necesidades de información de la organización. Esta evolución es posible cuando el diseñador crea un verdadero modelo conceptual de la organización con las siguientes características:

- El modelo refleja fehacientemente las operaciones de la organización.
- Es lo bastante flexible para permitir cambios conforme surjan nuevas necesidades de información.
- Apoya muchas visiones de usuario diferentes.
- Es independiente de la implementación física.

Etapa	Actividades
Investigación preliminar	Entrevista a usuarios, reportes de estudio, transacciones, procedimientos, software, documentación para identificar problemas en el sistema actual y metas del nuevo sistema.
Estudio de factibilidad	Estudio de alternativas, estimación de costos, horarios de estudio, beneficios. Hacer recomendaciones.
Diseño preliminar	Trabajo con usuarios para desarrollo de diseño del sistema general. Elección del mejor diseño. Desarrollo de diagramas de flujo del sistema. Identificación de hardware, software y necesidades personales. Revisión de estimaciones.
Diseño detallado	Realización de diseño técnico. Planificación de módulos de programa, algoritmos, archivos, bases de datos, formas I/O. Revisión de estimaciones.
Implementación del sistema	Programación de módulos, conversión de archivos, prueba del sistema, escritura de documentación, desarrollo de procedimientos operativos, capacitación de personal, elaboración de operaciones paralelas, recorte sobre el nuevo sistema.
Operación del sistema	Evaluación de sistemas. Monitorización y modificación del sistema según se necesite.

FIGURA 2.2

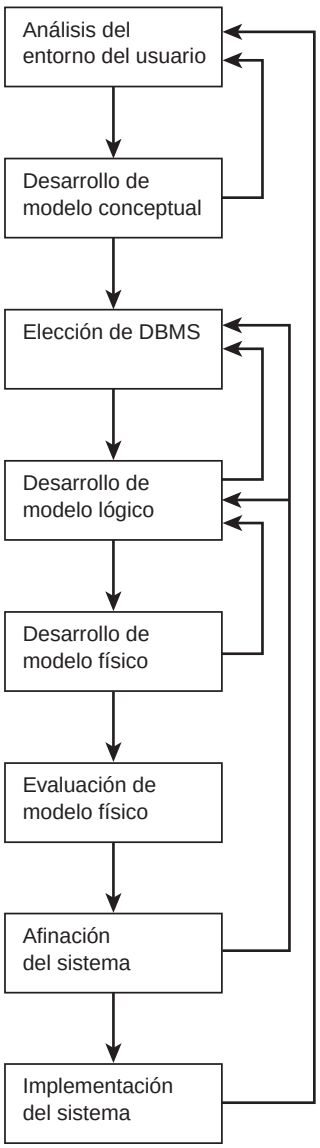
Etapas en el ciclo de vida de análisis de sistemas tradicional

- No depende del modelo de datos usado por un sistema de gestión de base de datos particular.

Un modelo de base de datos conceptual bien diseñado protege la fuente de datos al permitirle evolucionar de modo que sirva a las necesidades de información actuales y del mañana. Si el sistema es verdaderamente independiente de su implementación física, entonces se puede mover a nuevo hardware para sacar ventaja de desarrollos técnicos. Incluso si se sustituye el sistema de gestión de base de datos elegido para su implementación, el modelo lógico puede cambiar, pero el modelo conceptual de la empresa puede sobrevivir. El abordaje del diseño escalonado de base de datos es un método de arriba abajo que comienza con enunciados generales de las necesidades y avanza a la consideración de problemas cada vez más detallados. En diferentes fases del proyecto se consideran distintos problemas. Cada etapa usa herramientas de diseño que son adecuadas al problema en dicho nivel. La figura 2.3 muestra las principales etapas de diseño. Éstas son

1. Análisis del entorno del usuario.
   El primer paso en el diseño de una base de datos es determinar el entorno de usuario actual. El diseñador estudia todas las aplicaciones actuales, determina sus entradas y salidas, examina todos los reportes generados por el sistema actual y entrevista a los usuarios para determinar cómo usan el sistema. Después de que el sistema actual se entiende a profundidad, el diseñador trabaja de cerca con los usuarios actuales y con los potenciales usuarios del nuevo sistema para identificar sus necesidades. El diseñador considera no sólo las necesidades actuales sino las posibles nuevas aplicaciones o usos futuros de la base de datos. El resultado de este análisis es un modelo del entorno y las necesidades del usuario.

FIGURA 2.3
Pasos en el diseño escalonado de bases de datos



2. Desarrollo de un modelo de datos conceptual.
   Al usar el modelo del entorno del usuario el diseñador desarrolla un modelo conceptual detallado de la base de datos: identifica las entidades, atributos y relaciones que se representarán. Además del modelo conceptual, el diseñador tiene que considerar cómo se usará la base de datos. Se deben especificar los tipos de aplicaciones y transacciones, los tipos de acceso, el volumen de transacciones, el volumen de datos, la frecuencia de acceso y otros datos cuantitativos. También se deben identificar otras restricciones como restricciones presupuestarias y necesidades de desempeño. El resultado de esta fase es un conjunto de especificaciones de base de datos.
3. Elección de un DBMS.
   El diseñador usa las especificaciones y su conocimiento de los recursos de hardware y software disponibles para evaluar sistemas de gestión de bases de datos alternativos. Cada sistema de gestión de base de datos impone sus propias restricciones. El diseñador intenta elegir el sistema que satisface mejor las especificaciones para el entorno.

4. Desarrollo del modelo lógico.
   El diseñador mapea el modelo conceptual al modelo de datos utilizado por el DBMS elegido, lo que crea el modelo lógico.
5. Desarrollo del modelo físico.
   El diseñador planifica las plantillas de datos al considerar las estructuras soportadas por el DBMS elegido, y los recursos de hardware y software disponibles.
6. Evaluación del modelo físico.
   Luego el diseñador estima el rendimiento de todas las aplicaciones y transacciones, y considera los datos cuantitativos anteriormente identificados y las prioridades dadas a las aplicaciones y transacciones. Puede ser útil desarrollar un **prototipo**, que implemente una porción seleccionada de la base de datos de modo que las visiones de usuario se puedan validar y el desempeño se pueda medir con más precisión.
7. Realización de una afinación si lo indica la evaluación.
   Para mejorar el rendimiento se pueden realizar ajustes como modificación de estructuras físicas u optimización del software.
8. Implementación del modelo físico.
   Si la evaluación es positiva, entonces el diseñador implementa el diseño físico y la base de datos se vuelve operativa.

Las iteraciones en la figura 2.3 proporcionan oportunidades para retroalimentación y cambios en varias etapas en el proceso de diseño. Por ejemplo, después de desarrollar el modelo conceptual, el diseñador se comunica con los grupos de usuarios para asegurarse de que sus requisitos de datos están representados de manera adecuada. Si no es así, debe ajustar el modelo conceptual. Si el modelo conceptual no mapea bien un modelo de datos particular, se debe considerar otro. Para un DBMS particular, puede haber varios mapeos lógicos posibles que se puedan evaluar. Si el modelo físico no es aceptable, se puede considerar un mapeo diferente o un DBMS distinto. Si los resultados de la evaluación de desempeño no son satisfactorios, se pueden realizar afinación y evaluación adicionales. El modelo físico se puede cambiar si la afinación no produce el desempeño requerido. Si la afinación y la optimización repetida no son suficientes, puede ser necesario cambiar la forma en que se mapea el modelo lógico al DBMS o considerar un sistema de gestión de base de datos diferente. Incluso después de implementar el sistema, pueden ser necesarios cambios para responder a las cambiantes necesidades del usuario o a un entorno cambiante.

2.4 Herramientas de diseño

Existen muchas metodologías que pueden facilitar el proceso de diseño de la base de datos tanto para diseñadores como para usuarios. Las metodologías varían desde las técnicas generales descritas en la literatura, hasta productos comerciales cuya meta es automatizar el proceso de diseño. Por ejemplo, los paquetes **CASE** (Computer-Aided Software Engineering: software de ingeniería asistido por computadora), que incluyen varias herramientas para análisis de sistemas, gestión de proyectos, diseño y programación, están disponibles con muchos proveedores. Estos paquetes proporcionan herramientas que pueden ser muy útiles en el proceso de diseño de bases de datos. Las herramientas por lo general se categorizan como upper-CASE, lower-CASE o integradas. Las herramientas upper-CASE se usan en la planificación de bases de datos para recolección y análisis de datos, diseño de modelos de datos y diseño de aplicaciones. La herramientas lower-CASE se usan para implementar la base de datos, incluidos la elaboración de prototipos, conversión de datos, generación de código de aplicación, generación de reportes y pruebas. Las herramientas CASE integradas cubren ambos niveles. Un diccionario de datos orientado al usuario es una herramienta que

se puede desarrollar con o sin un paquete CASE. El **software de gestión de proyectos** es otro tipo de herramienta que se puede aplicar de manera efectiva al desarrollo de las bases de datos.

2.4.1 Diccionario de datos

Un diccionario de datos, que se discute en la sección 2.2.2, es un depósito de información que describe la estructura lógica de la base de datos. Tiene entradas para tipos de registro, tipos de ítem de datos y agregados de datos, junto con otra información. La cantidad de información y la forma en que se usa la información varían con el sistema. La mayoría de los proveedores de DBMS ofrece diccionarios o directorios de datos que almacenan la descripción de la base de datos y se usan en la creación y procesamiento de la base de datos. El diccionario de datos contiene **metadatos**, o datos acerca de los datos en la base de datos. Si el diccionario de datos es parte del DBMS, se le conoce como diccionario de datos **integrado** o catálogo del sistema. Un diccionario de datos integrado siempre es consistente con la estructura de la base de datos real, porque el sistema la mantiene automáticamente. Los diccionarios de datos integrados realizan muchas funciones a lo largo de la vida de la base de datos, no sólo en la fase de diseño. Si el diccionario de datos está disponible sin un DBMS particular, se le conoce como diccionario de datos **independiente** (freestanding). Un diccionario de datos independiente puede ser un producto comercial o un simple archivo desarrollado y mantenido por el diseñador. Por ejemplo, los paquetes CASE con frecuencia incluyen una herramienta de diccionario de datos, y los diccionarios de datos independientes están disponibles incluso para entornos que no son de bases de datos. Tanto los diccionarios de datos integrados como los independientes tienen ventajas y desventajas, pero un diccionario independiente puede ser preferible en las etapas de diseño iniciales, antes de que el diseñador elija un DBMS particular. Dado que no está ligado a un DBMS, permite al diseñador disfrutar las ventajas de tener esta herramienta sin comprometerse con una implementación particular. Una gran desventaja es que, una vez creada la base de datos, los ajustes a su estructura pueden no ingresarse en el diccionario de datos independiente y, con el tiempo, el diccionario no reflejará con precisión la estructura de la base de datos.

Un diccionario de datos independiente es útil en las primeras etapas de diseño para recopilar y organizar información acerca de los datos. Se deben determinar cada ítem de datos, su fuente, su nombre, sus usos, sus significados, sus relaciones con otros ítems, su formato y la identidad de la persona o grupo responsable de ingresarlo, actualizarlo y mantenerlo correcto. El diccionario de datos proporciona una efectiva herramienta para lograr estas tareas. El administrador de base de datos (ABD) puede comenzar a desarrollar el diccionario de datos al identificar ítems de datos, asegurar concordancia de usuarios acerca de una definición de cada ítem e ingresar el ítem en el diccionario. Dado que los usuarios no deben estar al tanto de los datos de otros usuarios, el ABD debe controlar el acceso al diccionario.

Un diccionario de datos es útil para lo siguiente:

- Recopilar y almacenar información acerca de datos en una ubicación central. Esto ayuda al administrador a ganar control sobre los datos como un recurso.
- Asegurar la concordancia de los usuarios y diseñadores acerca de los significados de los ítems de datos. Se debe desarrollar una definición exacta y acordada de cada ítem para almacenar en el diccionario de datos.
- Comunicación con usuarios. El diccionario de datos facilita enormemente la comunicación, pues se almacenan significados exactos. El diccionario también identifica personas, departamentos o grupos que tienen acceso a o interés en cada ítem.
- Identificación de redundancia e inconsistencia en los nombres de ítem de datos. En el proceso de identificar y definir ítems de datos, el ABD puede descubrir **sinónimos**,

que son diferentes nombres para el mismo ítem. La base de datos puede aceptar sinónimos, pero el sistema debe estar al tanto de ellos. El ABD también puede descubrir **homónimos**, que son nombres idénticos para diferentes ítems de datos. Éstos nunca se permiten en una base de datos.

- Seguir la huella de cambios a la estructura de la base de datos. Los cambios como creación de nuevos ítems de datos y tipos de registro o la alteración a descripciones de ítem de datos se deben registrar en el diccionario de datos.
- Determinación del impacto de los cambios a la estructura de la base de datos. Dado que el diccionario de datos registra cada ítem, todas sus relaciones y todos sus usuarios, el ABD puede ver qué efectos tendría un cambio.
- Identificación de las fuentes de y responsabilidad por la exactitud de cada ítem. Una regla general para entrada de datos es que los datos se deben capturar tan cerca de su fuente como sea posible. Las personas o departamentos que generen o capturen valores para cada ítem de datos y quienes son responsables de actualizar cada ítem se deben mencionar en el diccionario de datos.
- Registro de los esquemas externo, lógico e interno y los mapeos entre ellos. Este aspecto se discutirá en la sección 2.6.
- Registro de acceso de información de control. Un diccionario de datos puede registrar las identidades de todos aquellos que tienen permiso para ingresar cada ítem, junto con el tipo de recuperación de acceso, inserción, actualización o borrado.
- Proporcionar información de auditoría. El diccionario de datos del sistema (es decir, el directorio de datos o catálogo del sistema) se puede usar para registrar cada acceso, lo que permite el uso de estadísticas a recopilar para auditar al sistema y marcar los intentos de violación de seguridad.

Note que no todos los diccionarios de datos proporcionan soporte para todas estas funciones, y algunos proporcionan funciones adicionales. Algunos sólo almacenan el esquema, algunos, documentación de control y algunos directorios de datos del sistema son "metasistemas" que controlan el acceso a la base de datos, generan código de sistema y también mantienen estadísticas acerca del uso de la base de datos.

2.4.2 Software de gestión de proyecto

Este tipo de software proporciona un conjunto de herramientas que se pueden usar para planificar y gestionar un proyecto, especialmente cuando existen muchas personas que trabajan en él. Por lo general existen muchos tipos de gráficos y gráficas disponibles, como las **gráficas de Gantt**, que se muestran en la figura 2.12, y las **gráficas PERT**, que son similares. El usuario especifica los objetivos y ámbito del proyecto, identifica las tareas y fases principales, indica dependencias entre las tareas, identifica los recursos disponibles y establece un cronograma para completar las tareas y fases del proyecto. El software se puede usar para generar calendarios, producir gráficos con muchas vistas diferentes del progreso del proyecto y proporcionar un medio de comunicación entre el personal del proyecto, ya sea a través de intranet o acceso a Internet. Un ejemplo es Microsoft Project.

2.5 Administración de bases de datos

El administrador de la base de datos es responsable del diseño, operación y gestión de la base de datos. En muchos casos, el diseño conceptual lo realiza un diseñador de bases de datos y el ABD implementa el diseño, desarrolla el sistema y lo gestiona. El ABD debe ser técnicamente competente, un buen administrador, un experto comunicador y debe tener

excelentes habilidades interpersonales. Las habilidades de gestión se requieren para planificar, coordinar y realizar una multitud de tareas durante todas las fases del proyecto de la base de datos, y para supervisar al personal. Las habilidades técnicas son necesarias porque el ABD debe ser capaz de entender los complicados conflictos de hardware y software involucrados con la finalidad de diseñar, desarrollar y gestionar la base de datos, y para trabajar con expertos en sistemas y aplicaciones con el fin de resolver problemas. Las habilidades interpersonales son necesarias para comunicarse con los usuarios para determinar sus necesidades, negociar acuerdos acerca de definiciones de datos y derechos de acceso a la base de datos, para asegurar acuerdos acerca de los cambios a la estructura u operaciones de la base de datos que afecten a los usuarios, y para mediar entre los usuarios cuando los conflictos lo requieran. Para todas estas actividades se necesitan excelentes habilidades de comunicación. El ABD tiene muchas funciones que varían de acuerdo con la etapa del proyecto de la base de datos. Dado que hay tantas tareas a realizar, en especial durante las fases de diseño y creación, es posible que el ABD necesite delegar algunas de estas responsabilidades. Las funciones principales incluyen planificación, diseño, desarrollo y gestión de la base de datos.

2.5.1 Planificación y diseño

- **Planificación preliminar de la base de datos.** Si el ABD o el diseñador de la base de datos es elegido temprano en el proyecto, debe participar en la investigación preliminar y en el estudio de factibilidad. Si el ABD todavía no se elige, uno de los líderes de estos estudios puede convertirse en candidato para la posición.
- **Identificación de los requisitos de los usuarios.** El ABD o el diseñador examina todos los reportes generados por el sistema actual y consulta con los usuarios para determinar si los reportes satisfacen sus necesidades de información. Puede trabajar con usuarios presentes y potenciales para diseñar nuevos reportes que les gustaría que produzca el sistema propuesto. También se puede preguntar a los usuarios qué transacciones en línea les gustaría realizar. El ABD estudia todas las aplicaciones actuales, en especial sus entradas y salidas. También se registran la frecuencia de los reportes y transacciones y el marco temporal dentro del cual se deben producir. El ABD usa su conocimiento de los objetivos de la organización a largo y corto plazos para dar prioridad a las necesidades de los usuarios.
- **Desarrollo y mantenimiento del diccionario de datos.** Conforme determina las necesidades de los usuarios, el ABD o diseñador almacena los nombres, fuentes, significados, usos y sinónimos de ítem de datos en el diccionario de datos. El ABD revisa el diccionario de datos para incluir más información acerca de la base de datos conforme avance el proyecto.
- **Diseño del modelo conceptual.** El ABD o diseñador identifica todas las entidades, atributos y relaciones que se deben representar en la base de datos, y desarrolla un modelo conceptual que es un reflejo preciso del minimundo, y captura las operaciones de la organización en el mundo real que son de interés para la base de datos.
- **Elección de un DBMS.** El ABD considera el modelo conceptual y otras especificaciones de la base de datos y el hardware y software de computadora disponibles para la base de datos y elige el DBMS que mejor se ajusta al entorno y satisface las especificaciones.
- **Desarrollo del modelo lógico.** Una vez que se elige el DBMS, existen muchas formas en que se puede mapear el modelo conceptual al modelo de datos utilizado por el DBMS. El ABD elige aquel que parezca ser el más natural y apropiado, sin considerar las limitaciones del DBMS.

- **Desarrollo del modelo físico.** Existen muchas formas en que el modelo lógico se puede mapear a las estructuras de datos proporcionadas por el DBMS y a dispositivos físicos. El ABD evalúa cada mapeo al estimar el desempeño de las aplicaciones y transacciones. El mejor mapeo se convierte en el modelo físico.

2.5.2 Desarrollo de la base de datos

- **Creación y carga de la base de datos.** Una vez desarrollado el modelo físico, el ABD crea la estructura de la base de datos usando el lenguaje de definición de datos para el DBMS elegido. Establece conjuntos de datos físicos, crea bibliotecas y carga los datos en la base de datos, por lo general con el programa de utilidades del DBMS que acepta archivos existentes o convertidos, coloca los datos en las ubicaciones adecuadas y construye índices y/o establece valores de puntero conforme se cargan los registros.
- **Desarrollo de vistas de usuario.** El ABD intenta satisfacer las necesidades de datos de todos los usuarios. Una vista de usuario puede ser idéntica a la que se solicita en las etapas iniciales del diseño. Sin embargo, con frecuencia las solicitudes de los usuarios cambian conforme comienzan a entender mejor el sistema. Si la vista no coincide con la solicitud del usuario, el ABD debe presentar buenas razones por las que la petición no se satisfizo y asegurar un acuerdo acerca de la visión real. Dado que el soporte al usuario es vital para el éxito del proyecto de la base de datos, es esencial que los usuarios sientan que la base de datos les funciona bien.
- **Escritura y mantenimiento de documentación.** Idealmente, la documentación de la base de datos se escribe en forma automática mediante el sistema de diccionario de datos conforme avanza el proyecto. Cuando se crea la base de datos, el ABD se asegura de que la documentación refleja con precisión la estructura de la base de datos.
- **Desarrollo y fortalecimiento de estándares de datos.** Dado que la base de datos la comparten muchos usuarios, es importante definir y fortalecer estándares para beneficio de todos. Los usuarios responsables de insertar y actualizar datos deben seguir un formato estándar para ingresar datos. La interfaz de usuario se debe diseñar para facilitar a los usuarios seguir los estándares. Por ejemplo, las pantallas de entrada deben desplegar valores por defecto, mostrar rangos aceptables para los ítems, y cosas por el estilo. Los estándares de datos típicos incluyen especificaciones para valores nulos, abreviaturas, códigos, puntuación y uso de mayúsculas. El sistema puede verificar automáticamente en busca de errores y restricciones de rango. Otras restricciones que puede comprobar el DBMS antes de aceptar actualizaciones involucran la unicidad de valores clave y las relaciones entre valores de datos en un solo registro, entre registros en el mismo archivo y entre registros en diferentes archivos.
- **Desarrollo y fortalecimiento de estándares de programa de aplicación.** El ABD debe desarrollar estándares para programas de aplicación, de modo que obedezcan las restricciones de seguridad y privacidad de la base de datos, estén sujetas a mecanismos de auditoría, hagan uso adecuado del lenguaje de manipulación de datos de alto nivel y encajen con las facilidades de desarrollo de aplicaciones proporcionadas por el DBMS. Estos estándares son pertinentes tanto a aplicaciones antiguas que se convierten para uso con la base de datos y con nuevas aplicaciones.
- **Desarrollo de procedimientos operativos.** El ABD es responsable de establecer procedimientos para arranque diario del DBMS (si es necesario), suavizar el corrido de las operaciones de la base de datos, anotación de errores (logging) de transacciones, respaldos periódicos, procedimientos de seguridad y autorización, registro de fallas de hardware y software, tomar mediciones de desempeño, parar la base de datos en

una forma ordenada en caso de falla, reinicio y recuperación después de la falla, y parar al final de cada día (si es necesario). Dado que estos procedimientos los realizan operadores, el ABD debe consultar con los gestores de operaciones para asegurar que los operadores están capacitados en todos los aspectos de las operaciones de la base de datos.

- **Realización de capacitación de usuarios.** Los usuarios finales, programadores de aplicación y programadores de sistemas que ingresen a la base de datos deben participar en programas de capacitación para que puedan aprender a usarla más efectivamente. Las sesiones las puede dirigir el ABD, el proveedor del DBMS u otros capacitadores técnicos, ya sea en el centro de trabajo o en un centro de capacitación.

2.5.3 Gestión de bases de datos

- **Monitoreo del desempeño.** El ABD es responsable de recopilar y analizar estadísticas acerca del desempeño de la base de datos y responder a las quejas y sugerencias de los usuarios acerca del desempeño. Se debe medir el tiempo de corrido para aplicaciones y el tiempo de respuesta para consultas interactivas, de modo que el ABD pueda marcar los problemas en el uso de la base de datos. Por lo general, el DBMS proporciona facilidades para registrar esta información. El ABD compara continuamente el desempeño para las necesidades y hace ajustes cuando es necesario.
- **Ajuste y reorganización.** Si el desempeño comienza a degradarse conforme se hacen cambios a los datos almacenados, el ABD puede responder al agregar o cambiar índices, reorganizar archivos, usar dispositivos de almacenamiento más rápidos u optimizar el software. Para problemas serios de desempeño, es posible que tenga que cambiar el modelo físico y recargar toda la base de datos.
- **Mantenerse al corriente en mejoras a la base de datos.** El ABD debe estar al tanto de nuevas características y nuevas versiones del DBMS que estén disponibles. Debe evaluar estos nuevos productos y otros desarrollos de hardware y software para determinar si proporcionarían beneficios sustanciales a la organización.

2.6 La arquitectura en tres niveles de las bases de datos

En la sección 2.3 se presentó el proceso de diseño escalonado de las bases de datos que comenzó con el desarrollo de un modelo conceptual y terminó con un modelo físico. Ahora está listo para examinar éstos y otros conceptos relacionados más de cerca. Cuando se discute una base de datos, es necesario algún medio para describir diferentes aspectos de su estructura. Una primera proposición para un vocabulario estandarizado y arquitectura para sistemas de bases de datos se desarrolló y publicó en 1971 por el Database Task Group citado por la Conferencia acerca de Sistemas y Lenguajes de Datos, o **CODASYL DBTG**. Un vocabulario y arquitectura similares se desarrolló y publicó en 1975 por parte del Standards Planning and Requirements Committee del American National Standards Institute Committee on Computers and Information Processing, o **ANSI/X3/SPARC**. Como resultado de éstos y anteriores reportes, las bases de datos se pueden ver en tres niveles de abstracción. Los niveles forman una **arquitectura de tres niveles** en capas y se describen mediante tres **esquemas**, que son descripciones escritas de sus estructuras. El propósito de la arquitectura de tres niveles es separar el modelo del usuario de la estructura física de la base de datos. Existen muchas razones por las que es deseable esta separación:

- Diferentes usuarios necesitan distintas vistas de los mismos datos.

FIGURA 2.4
Arquitectura en tres niveles simplificada para sistemas de base de datos

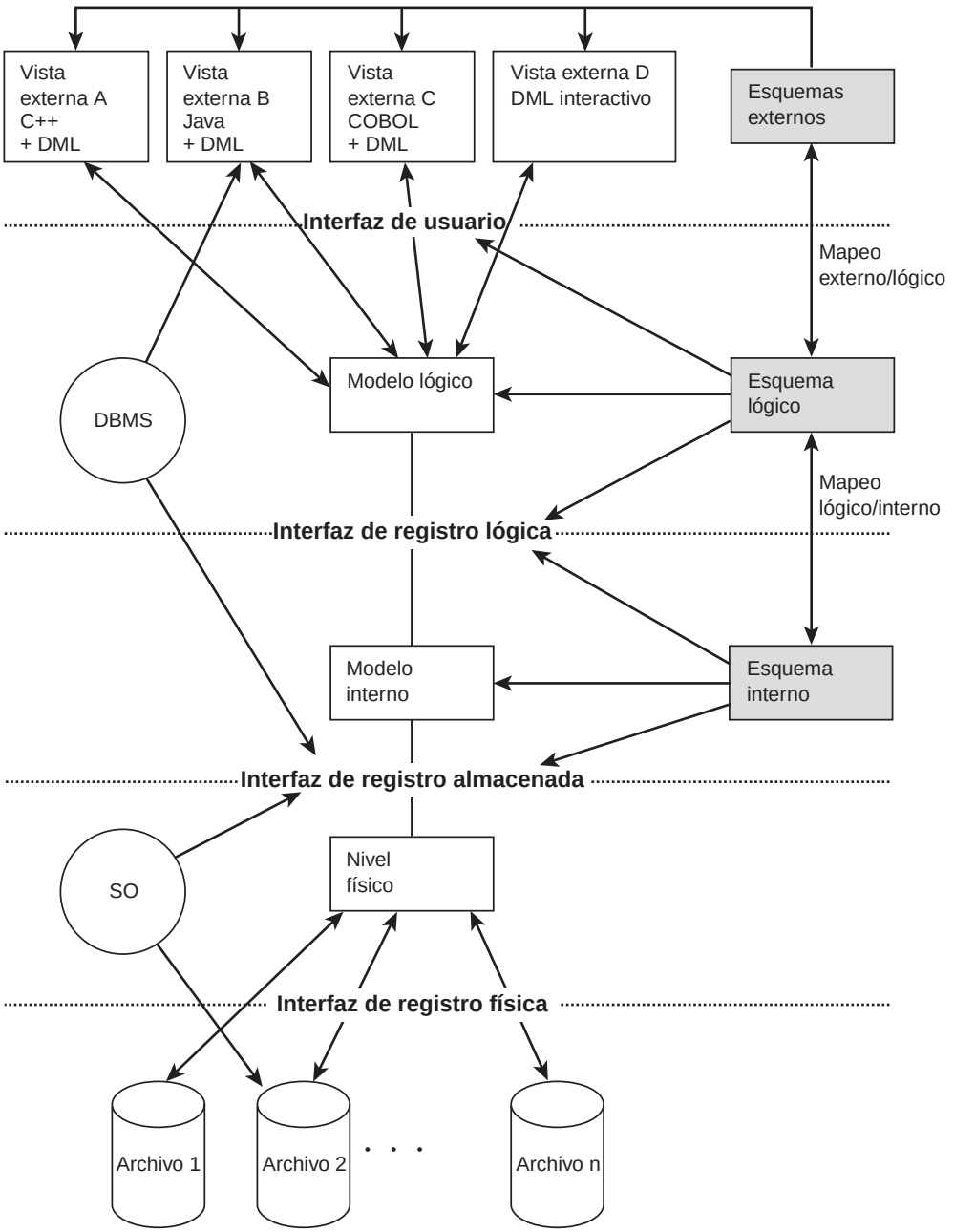
- La forma en que un usuario particular necesita ver los datos puede cambiar con el tiempo.
- Los usuarios no deben tener que lidiar con las complejidades de las estructuras de almacenamiento de la base de datos.
- El ABD debe ser capaz de efectuar cambios al modelo conceptual de la base de datos sin afectar a todos los usuarios.
- El ABD debe ser capaz de cambiar el modelo lógico, y las estructuras de datos y archivos, sin afectar el modelo conceptual o las vistas de los usuarios.
- Los cambios a los aspectos físicos del almacenamiento, como los cambios a los dispositivos de almacenamiento, no deben afectar a la estructura de la base de datos.

La figura 2.4 muestra la arquitectura en tres niveles de los sistemas de la base de datos. La forma en que los usuarios piensan acerca de los datos se llama **nivel externo**. El **nivel interno** es la forma en que los datos se almacenan realmente usando estructuras de datos y organizaciones de archivo estándar (vea el apéndice A). Sin embargo, existen muchas diferentes visiones de usuario y muchas estructuras físicas, de modo que debe haber algún método para mapear las visiones externas a las estructuras físicas. Un mapeo directo es indeseable, pues los cambios hechos a las estructuras físicas o dispositivos de almacenamiento requerirían un cambio correspondiente en el exterior al mapeo físico. Por tanto, hay un nivel medio que proporciona tanto el mapeo como la independencia deseada entre los niveles externo y físico. Éste es el nivel **lógico**.

2.6.1 Vistas externas

El **nivel externo** consiste en muchas **vistas externas** o **modelos externos** diferentes de la base de datos. Cada usuario tiene un modelo del mundo real representado en una forma que es adecuada para dicho usuario. La figura 2.5 presenta una imagen más detallada de la arquitectura de bases de datos en tres niveles. El nivel superior de la figura 2.5 muestra varias vistas externas. Un usuario particular interactúa sólo con ciertos aspectos del minimundo y está interesado sólo en algunas entidades, y sólo algunos de sus atributos y relaciones. Por tanto, la vista de dicho usuario contendrá sólo información acerca de dichos aspectos. Otras entidades u otros atributos o relaciones en realidad pueden representarse en la base de datos, pero el usuario no estará al tanto de ellos. Además de incluir diferentes entidades, atributos y relaciones, distintas visitas pueden tener diferentes representaciones de los mismos datos. Por ejemplo, un usuario puede creer que las fechas se almacenan en la forma mes, día, año, mientras que otro puede creer que se representan como año, mes, día. Algunas vistas pueden incluir datos **virtuales** o calculados, que son datos que en realidad

FIGURA 2.5
Arquitectura de base de datos en tres niveles



no se almacenan como tales, sino que se crean cuando es necesario. Por ejemplo, la edad realmente no se puede almacenar, pero dateOfBirth (fecha de nacimiento) sí se puede, y el sistema puede calcular la edad cuando el usuario la refiera. Incluso las vistas pueden incluir datos combinados o calculados a partir de varios registros. Un **registro externo** es un registro como lo ve un usuario particular, una parte de su vista externa. Una vista externa es en realidad una colección de registros externos. Las vistas externas se describen en **esquemas externos** (también llamados **subesquemas**) que se escriben en el lenguaje de definición de datos (DDL). El esquema de cada usuario da una descripción completa de cada tipo de registro externo que aparece en la vista de dicho usuario. Los esquemas se compilan mediante el DBMS y se almacenan en forma de objeto para uso del diccionario/directorio de datos del sistema al recuperar registros. También se deben mantener en forma de fuente

como documentación. El DBMS usa el esquema externo para crear una **interfaz de usuario**, que es tanto una facilidad como una barrera. Un usuario individual ve la base de datos a través de esta interfaz. Define y crea el entorno de trabajo para dicho usuario, y acepta y despliega información en el formato que el usuario espera. También actúa como una frontera por abajo de la cual al usuario no se le permite ver. Esconde al usuario los detalles lógicos, internos y físicos.

Para desarrollar una vista de usuario, el ABD primero entrevista al usuario y examina los reportes y transacciones que crea o recibe. Después de planificar el diseño completo de la base de datos, el ABD determina cuáles datos estarán disponibles a dicho usuario y qué representación verá el usuario, y escribe un esquema externo para dicho usuario. Siempre que sea posible, este esquema externo incluye toda la información que solicitó el usuario. Con el tiempo, las necesidades del usuario pueden cambiar y se pueden hacer modificaciones a la vista. Con frecuencia, los nuevos datos requeridos ya están presentes en la base de datos, y el esquema externo del usuario se rescribe para permitir el acceso a él.

2.6.2 Modelos lógico y conceptual

El nivel medio en la arquitectura de tres niveles es el **nivel lógico**, como se muestra en la figura 2.5. Este modelo incluye toda la estructura de información de la base de datos, como la ve el ABD. Es la "vista comunitaria" de los datos e incluye una descripción de todos los datos que están disponibles para compartir. Es un modelo o vista abarcadora de las operaciones de la organización en el minimundo. Todas las entidades, con sus atributos y relaciones, se representan en el modelo lógico usando el modelo de datos que soporta el DBMS. El modelo incluye cualesquiera restricciones sobre los datos e información semántica acerca de los significados de los datos. El modelo lógico soporta las vistas externas en que cualquier dato disponible a cualquier usuario debe estar presente en o derivar del modelo lógico. El modelo lógico es relativamente constante. Cuando el ABD originalmente lo diseña, intenta determinar las necesidades de información presentes y futuras y trata de desarrollar un modelo duradero de la organización. En consecuencia, conforme surgen nuevas necesidades de datos, el modelo lógico puede ya contener los objetos requeridos. Si éste no es el caso, el ABD expande el modelo lógico para incluir los nuevos objetos. Un buen modelo lógico será capaz de acomodar este cambio y todavía soportar las antiguas vistas externas. Sólo los usuarios que necesiten acceso a los nuevos datos deben resultar afectados por el cambio. El **esquema lógico** es una descripción completa del contenido de información de la base de datos. Se escribe en DDL, lo compila el DBMS y se almacena en forma de objeto en el diccionario/directorio de datos y en forma fuente como documentación. El DBMS usa el esquema lógico para crear la **interfaz de registro lógica**, que es una frontera por abajo de la cual todo es invisible al nivel lógico y que define y crea el entorno operativo para el nivel lógico. Ningún detalle interno o físico, sea cómo se almacenan o secuencian los registros, cruza esta frontera. El modelo lógico es en realidad una colección de registros lógicos. El modelo de datos lógico es el corazón de la base de datos. Soporta todas las vistas externas y, a su vez, lo soporta el modelo interno. Sin embargo, el modelo interno es simplemente la implementación física del modelo lógico. El modelo lógico en sí mismo se deriva del modelo conceptual. Desarrollar el modelo conceptual es la parte más desafiante, interesante y gratificante del diseño de bases de datos. El diseñador de bases de datos debe ser capaz de identificar, clasificar y estructurar objetos en el diseño. El proceso de **abstracción**, que significa identificar propiedades comunes de un conjunto de objetos en lugar de enfocarse en los detalles, se usa para categorizar datos. En la ciencia de la computación, la abstracción se usa para simplificar conceptos y esconder complejidades. Por ejemplo, los tipos de datos abstractos se consideran aparte de su implementación; el comportamiento de las consultas y pilas (stacks) se puede describir sin considerar cómo se representan. El diseñador puede observar diferentes niveles de abstracción, de modo que un objeto abstracto en un nivel se

vuelve un componente de un nivel de abstracción superior. Durante el proceso de diseño conceptual, puede haber muchos errores y varios inicios falsos. Como muchos otros procesos de resolución de problemas, el diseño de bases de datos conceptuales es un arte, guiado por el conocimiento. Puede haber muchas posibles soluciones, pero algunas son mejores que otras. El proceso en sí es una situación de aprendizaje. El diseñador gradualmente llega a entender las operaciones de la organización y el significado de sus datos, y expresa dicha comprensión en el modelo elegido. Si el diseñador produce un buen modelo conceptual, es una tarea relativamente sencilla convertirlo en un modelo lógico y completar los diseños interno y físico. Si el modelo conceptual es bueno, también son fáciles de definir las vistas externas. Si algún dato en el que pueda estar interesado un usuario se incluye en el modelo conceptual, es una tarea sencilla ponerlo en la vista externa del usuario. Por otra parte, un modelo conceptual deficiente puede ser difícil de implementar, en particular si los datos y las relaciones no están bien definidos. También será inadecuado proporcionar todos los modelos externos necesarios. Continuará causando problemas durante el tiempo de vida de la base de datos, porque tendrá que ser "parchada" siempre que surgen diferentes necesidades de información. La habilidad para ajustarse a los cambios es uno de los hitos del buen diseño conceptual. Por tanto, vale la pena gastar todo el tiempo y energía necesarios para producir el mejor diseño conceptual posible. La paga se sentirá no sólo en las etapas de diseño lógico e interno, sino en el futuro.

2.6.3 Modelo interno

El **nivel interno** cubre la implementación física de la base de datos. Incluye las estructuras de datos y organizaciones de archivo utilizadas para almacenar datos en dispositivos de almacenamiento físicos. El DBMS elegido determina, en gran medida, cuáles estructuras están disponibles. Funciona con los métodos de acceso del sistema operativo para colocar los datos en los dispositivos de almacenamiento, construir los índices y/o establecer los punteros que se usarán para recuperación de datos. Por ende, en realidad hay un nivel físico por abajo del que es responsabilidad del DBMS, uno que es gestionado por el sistema operativo bajo la dirección del DBMS. La línea entre las responsabilidades del DBMS y las responsabilidades del sistema operativo no es clara y en realidad varía de sistema a sistema. Algunos DBMS sacan ventaja de muchas de las facilidades de los métodos de acceso del sistema operativo, mientras que otros ignoran todo excepto los gestores I/O más básicos y crean sus propias organizaciones de archivos alternativas. El ABD debe estar al tanto de las posibilidades para mapear el modelo lógico al modelo interno, y elegir un mapeo que soporte la visión lógica y proporcione desempeño adecuado. El **esquema interno**, escrito en DDL, es una descripción completa del modelo interno. Incluye ítems de cómo se representan los datos, cómo se secuencian los registros, qué índices existen, qué punteros existen y cuál esquema de claves (hashing), si hay alguno, se utiliza. Un **registro interno** es un solo registro almacenado. Es la unidad que se transmite al nivel interno. La **interfaz de registro almacenada** es la frontera entre el nivel físico, del que puede ser responsable el sistema operativo, y el nivel interno, del que es responsable el DBMS. Esta interfaz la proporciona al DBMS el sistema operativo. En algunos casos donde el DBMS realiza algunas funciones del sistema operativo, el DBMS en sí puede crear esta interfaz. El nivel físico por abajo de esta interfaz consiste en ítems que sólo conoce el sistema operativo, tales como la manera exacta en que se implementa la secuenciación y si los campos de registros internos en realidad se almacenan como bytes contiguos en el disco. El sistema operativo crea la **interfaz de registro física**, que es una frontera inferior donde se esconden detalles de almacenamiento, tales como exactamente qué porción de qué pista contiene cuál dato.

El diccionario/directorio de datos no sólo almacena los esquemas completos externo, lógico e interno, sino también almacena los mapeos entre ellos. El **mapeo externo/lógico** dice al

DBMS cuáles objetos en el nivel lógico corresponden a cuáles objetos en una vista externa de un usuario particular. Puede haber diferencias en los nombres de registros, nombres de ítem de datos, orden de ítem de datos, tipos de datos, etc. Si los cambios se hacen o a una vista externa o a un modelo lógico, los mapeos se deben cambiar. De igual modo, el **mapeo lógico/interno** da la correspondencia entre los objetos lógicos y los internos, en cuanto que dice cómo se representan físicamente los objetos lógicos. Si cambia la estructura almacenada, el mapeo debe cambiar en concordancia.

Para entender las distinciones entre los tres niveles, se examinará qué recibe y pasa cada nivel cuando se solicita el registro de un empleado particular. Consulte la figura 2.6 junto con la figura 2.7. Cuando el usuario A solicita un registro como el registro (externo) del Empleado 101, el DBMS intercepta la solicitud. Si la solicitud se ingresa en línea o en una estación de trabajo usando el lenguaje de manipulación de datos interactivo (DML), el

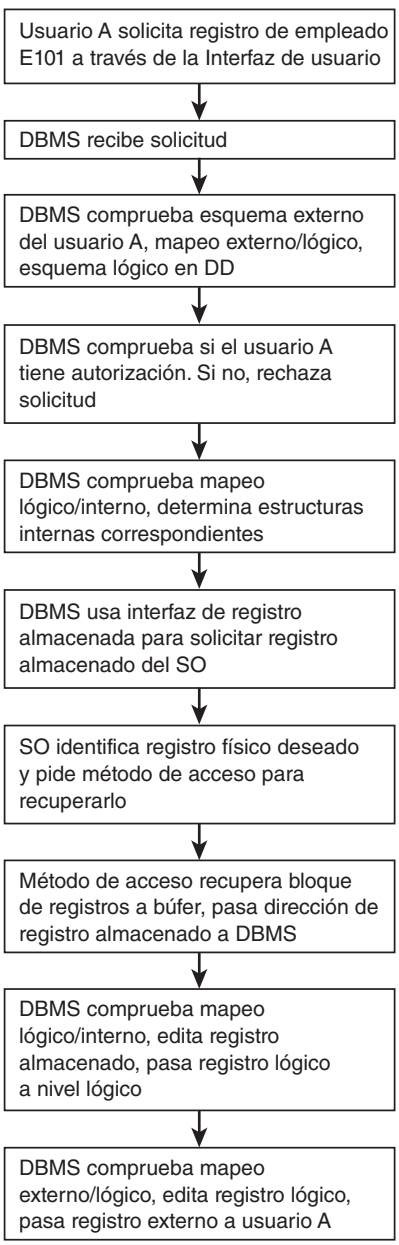


FIGURA 2.6
Recuperación de registro de E101 para el usuario A

FIGURA 2.7
Diferencias en registro externo, lógico, almacenado y físico

Registro de empleado externo:

employeeName	empNumber	dept
JACK JONES	E101	Marketing

Registro de empleado lógico:

empId	lastName	firstName	dept	salary
E101	Jones	Jack	12	55000

Registro de empleado almacenado:

empId	lastName	firstName	dept	salary	Puntero adelante	Puntero atrás
E101bbbbbbbb	**Jones**bbbbbbb	**Jack**bbbbbbbbb	**12**bbbbbbbbb	**55000**bbbbb	**10101**bbbbbbb	**10001**

Registro físico:

Cabecera de bloque	rec de E90	rec de E95	**reg de E101**	rec de E125

DBMS recibe la solicitud del sistema de comunicaciones de datos. Si la solicitud se envía como parte de un programa, el DBMS lo saca del programa y lo sustituye con una llamada de subrutina. En cualquier caso, una vez que el DBMS recibe la solicitud, comprueba la vista externa del usuario y el mapeo externo/lógico. Tanto las vistas como el mapeo se almacenan en forma de objeto en el diccionario/directorio de datos, de modo que la comprobación se puede completar fácilmente. El DBMS también comprueba la autorización del usuario para acceder a los ítems y realizar las operaciones solicitadas. Si no hay tal autorización, se niega la solicitud. Si el usuario tiene autorización, el DBMS nota cuáles objetos del nivel lógico se necesitan, y la solicitud pasa al nivel lógico. A continuación, se comprueba el mapeo lógico/interno para ver cuáles estructuras internas corresponden a los ítems lógicos. Una vez más, el diccionario/directorio de datos almacena los modelos y el mapeo en forma de objeto. El DBMS identifica los objetos internos que se requieren y pasa la solicitud al sistema operativo.

En el nivel físico, un registro de empleado se contiene en un registro físico, una página o bloque puede contener varios registros de empleado. Ésta es la unidad que se lleva al búfer desde el disco. El sistema operativo es responsable de realizar el trabajo básico de localizar el bloque correcto para el registro solicitado y gestionar su recuperación. Cuando la recuperación se completa, el bloque adecuado está en el búfer, y el sistema operativo pasa al DBMS la ubicación exacta dentro del búfer donde aparece el registro almacenado. El DBMS accede sólo al registro de empleado solicitado, no a todos los registros en el bloque. Sin embargo, recibe el registro almacenado completo, exactamente como se codificó o encriptó, junto con cualesquier punteros que puedan aparecer en él, pero sin sus cabeceras. Ésta es una descripción de un registro interno, que es la unidad que pasa a través de la interfaz de registro almacenada. El DBMS usa el mapeo lógico/interno para decidir cuáles ítems pasan a través de la interfaz de registro lógica hacia el nivel lógico.

En el nivel lógico, el registro aparece como un registro lógico, al que se remueven el encriptado y la codificación especial. Los punteros que se usan para establecer relaciones no aparecen en el nivel lógico, pues dicho nivel sólo se preocupa por la existencia de las relaciones, no de cómo se implementan. De igual modo, los punteros utilizados para secuenciar no son de interés en el nivel lógico, ni tampoco los índices. Por tanto, el registro del nivel lógico contiene sólo la información para dicho empleado particular, pero contiene todos los campos almacenados para el empleado, en el orden en el que se almacenaron. El DBMS com-

prueba el mapeo externo/lógico para decidir cómo se debe ver el registro externo del usuario.

Cuando el registro pasa a través de la interfaz de usuario al nivel externo, ciertos campos se pueden ocultar, algunos pueden cambiar de nombres, otros se pueden reordenar, algunos pueden aparecer en una forma diferente a la de su forma almacenada y otros pueden ser virtuales, creados a partir del registro almacenado. Algunos registros externos pueden ser combinaciones de registros almacenados o el resultado de operaciones como cálculos en los registros almacenados. Entonces el usuario realiza operaciones sobre los registros externos. Esto es: puede manipular sólo aquellos ítems que aparecen en el registro externo. Estos cambios, si son legales, a la larga se realizan sobre el registro almacenado. La figura 2.6 resume los pasos en este proceso, y la figura 2.7 ilustra las diferencias en la apariencia del registro de empleado conforme pasa al nivel externo.

2.6.4 Independencia de datos

Una razón principal de la arquitectura de tres niveles es proporcionar **independencia de datos**, lo que significa que los niveles superiores no son afectados por los cambios en los niveles inferiores. Existen dos tipos de independencia de datos: **lógica** y **física**. La independencia de datos lógica se refiere a la inmunidad de los modelos externos a cambios en el modelo lógico. Los cambios de modelo lógico, como agregar nuevos tipos de registro, nuevos ítems de datos y nuevas relaciones, deben ser posibles sin afectar las vistas externas existentes. Desde luego, los usuarios para quienes se hacen los cambios deben estar al tanto de ellos, pero otros usuarios no deben estarlo. En particular, los programas de aplicación existentes no se deben rescribir cuando se hagan cambios en el nivel lógico.

La independencia de datos física se refiere a la inmunidad del modelo lógico a los cambios en el modelo interno. Los cambios internos o físicos, como una diferente secuenciación física de registros, cambio de un método de acceso a otro, cambio del algoritmo de hashing, uso de diferentes estructuras de datos y el uso de nuevos dispositivos de almacenamiento no debe tener efecto sobre el modelo lógico. En el nivel externo, el único efecto que se puede sentir es un cambio en el desempeño. De hecho, un deterioro en el desempeño es la razón más común para cambios en el modelo interno. La figura 2.8 muestra dónde ocurre cada tipo de independencia de datos.

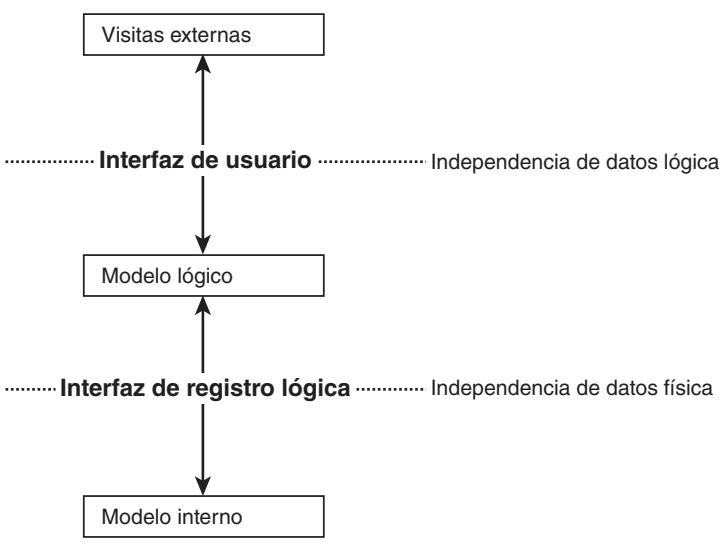


FIGURA 2.8
Independencia de datos lógica y física

2.7 Panorama de los modelos de datos

Un **modelo de datos** es una colección de herramientas que usualmente incluyen un tipo de diagrama y vocabulario especializado para describir la estructura de la base de datos. Un modelo de datos proporciona una descripción de la estructura de la base de datos, incluidos los datos, las relaciones dentro de los datos, las restricciones sobre los datos y a veces semántica o significados de los datos. Esta estructura se llama **intensión** de la base de datos y es relativamente permanente. Se describe en el esquema de la base de datos. El esquema puede cambiar ocasionalmente si surgen nuevas necesidades de datos, un proceso llamado **evolución del esquema**. Los datos almacenados en la base de datos en un momento dado se llaman **extensión** de la base de datos, **instancia de la base de datos** o **estado de la base de datos**. La extensión cambia siempre que se agregan, borran o actualizan registros. La extensión siempre debe ser un **estado válido**, lo cual significa que debe satisfacer todas las restricciones especificadas en el esquema. La intensión de la base de datos es en realidad una estructura de datos abstracta compleja que formalmente define todas las extensiones posibles.

Los modelos de datos en esta sección describen métodos de representación. Existe mucho desacuerdo acerca de qué constituye un modelo de datos, y esto se refleja en las docenas de modelos propuestos y metodologías que se encuentran en la literatura.

2.7.1 Modelo entidad-relación

El modelo entidad-relación es un ejemplo de lo que se llama **modelo semántico**. Los modelos semánticos se usan para describir los niveles conceptual y externo de datos, y son independientes de los aspectos interno y físico. Además de especificar lo que se representará en la base de datos, intentan incorporar algunos significados o aspectos semánticos de los datos como la representación explícita de objetos, atributos y relaciones, categorización de objetos, abstracciones y restricciones explícitas de datos. Algunos de los conceptos del modelo E-R se introdujeron en la sección 2.2.2, cuando se describieron los cuatro niveles de abstracción en la discusión de datos. El modelo lo introdujo Chen a mediados de la década de 1970 y se usa ampliamente para diseño conceptual. Se basa en la identificación de objetos llamados entidades, que son representaciones de objetos reales en el minimundo. Las entidades se describen mediante sus atributos y se conectan mediante relaciones. Las entidades se describen como personas, lugares, eventos, objetos o conceptos acerca de los cuales se recopilan datos. Una descripción más adecuada es que una entidad es cualquier objeto que existe y se distingue de otros objetos. Los atributos describen las entidades y las distinguen unas de otras. Un conjunto de entidad se define como una colección de entidades del mismo tipo. Ahora también se define un conjunto de relación como un conjunto de relaciones del mismo tipo, y se agrega el hecho de que las relaciones mismas pueden tener atributos descriptivos. El modelo E-R también permite expresar las restricciones sobre las entidades o relaciones. El capítulo 3 contiene una descripción más completa del modelo E-R, incluidos detalles acerca de las restricciones.

Una de las características más útiles y atractivas del modelo E-R es que proporciona un método gráfico para mostrar la estructura conceptual de la base de datos. Los diagramas E-R contienen símbolos para entidades, atributos y relaciones. La figura 2.9 muestra algunos de los símbolos, junto con sus nombres, significados y usos. La figura 2.10 ilustra un diagrama E-R simple para una base de datos de estudiantes y clases similar a la que se discutió en la sección 1.2. Muestra un conjunto de entidad llamado Student, con los atributos stuId, lastName, firstName, major y credits. La información de clase, a conservar para cada clase impartida durante el periodo actual, incluye classNumber, schedule y room. Los conjuntos entidad Student y Class se conectan mediante un conjunto relación, Enroll, que dice

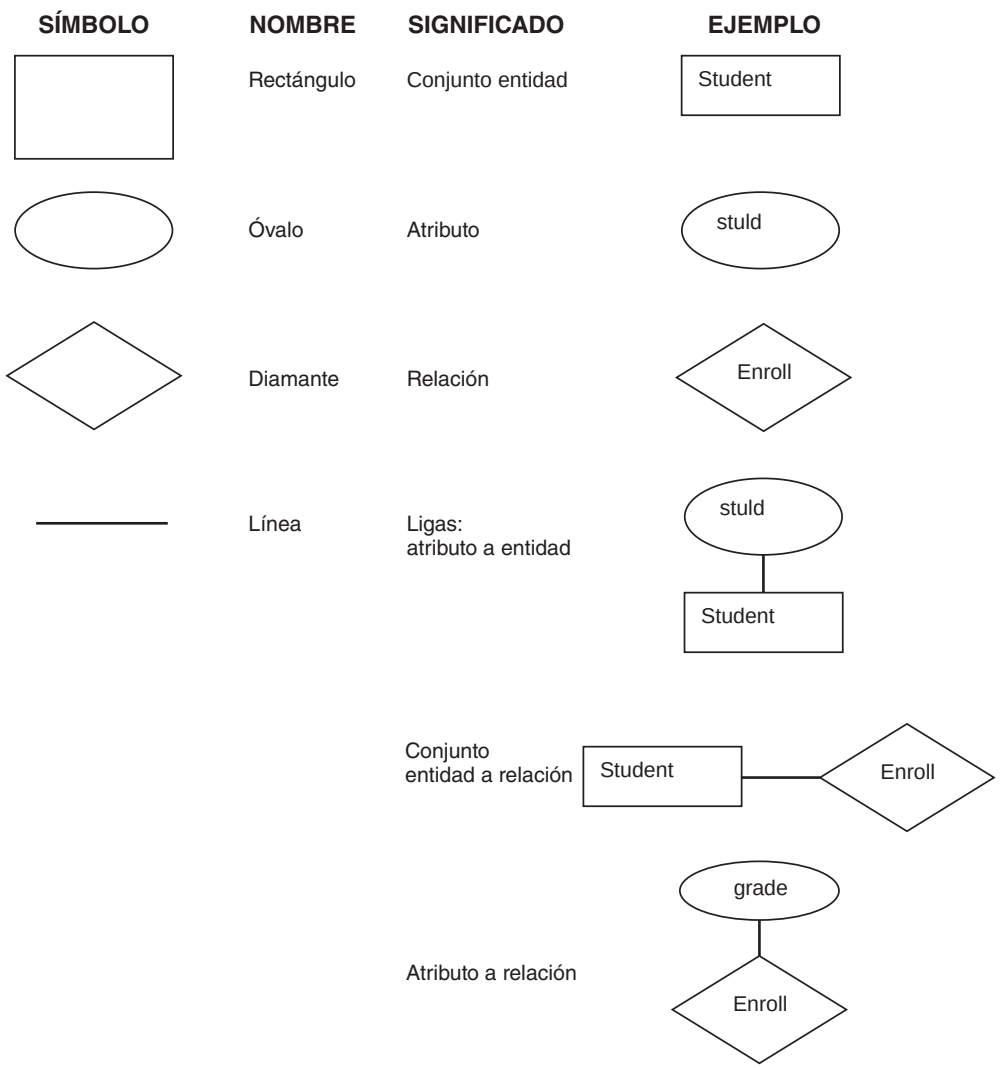


FIGURA 2.9

Símbolos básicos para diagramas E-R

cuáles estudiantes están inscritos en cuáles clases. Tiene su propio atributo descriptivo, grade. Note que grade no es un atributo de Student, pues conocer la calificación para un estudiante es insignificante a menos que también se conozca el curso. De igual modo, grade no es un atributo de Class, pues saber que se dio una calificación particular para una clase es insignificante a menos que se sepa a cuál estudiante se dio la calificación. En consecuencia, dado que la calificación sólo tiene significación para una combinación particular de estudiante y clase, pertenece al conjunto de relación. Puesto que el modelo E-R sólo describe una estructura conceptual para la base de datos, no se intenta describir cómo se podría o debería representar internamente el modelo. Por tanto, el material en la sección 2.2.2, en la que se describen ítems de datos, registros y archivos, no es parte del modelo E-R en sí.

2.7.2 Modelo relacional

El modelo relacional es un ejemplo de un **modelo basado en registro**. Los modelos basados en registro se usan para describir los niveles externo, lógico y, en cierta medida, interno de

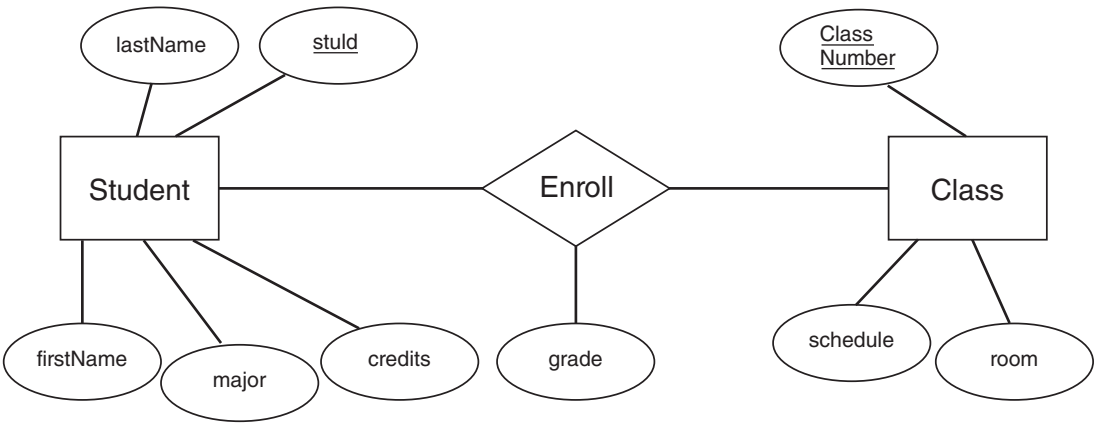


FIGURA 2.10
Diagrama E-R simplificado

la base de datos. Permiten al diseñador desarrollar y especificar la estructura lógica y proporcionar algunas opciones para la implementación del diseño. Se implementan con el uso de una variedad de sistemas de bases de datos. Sin embargo, no proporcionan mucha información semántica como categorización de objetos, relaciones, abstracción o restricciones de datos. El modelo relacional lo propuso Codd en 1970 y continúa siendo el más ampliamente usado, debido a su simplicidad desde el punto de vista del usuario y su poder. El modelo relacional comenzó con el uso de la teoría de relaciones en matemáticas y la adaptó para su uso en la teoría de bases de datos. El mismo tipo de desarrollo teórico de la materia, completada con notación formal, definiciones, teoremas y pruebas que usualmente se encuentran en matemáticas, se pueden aplicar a las bases de datos usando este modelo. Los resultados de este desarrollo teórico se aplican entonces a consideraciones prácticas de implementación. En el modelo relacional, las entidades se representan como relaciones, que se representan físicamente como tablas o arreglos bidimensionales, y los atributos como columnas de dichas tablas. Las relaciones también se pueden representar como relaciones o tablas, pues este modelo considera una relación como un tipo especial de entidad. La figura 1.1 (a)-(d) mostró un ejemplo de base de datos relacional para datos acerca de estudiantes y sus clases. Se tiene una tabla para el conjunto de entidad estudiante, uno para el conjunto de entidad clase, uno para el conjunto de entidad personal docente y uno para la relación entre el estudiante y la clase. Un estudiante se relaciona con una clase al inscribirse en dicha clase. Las columnas de la tabla `Student` tienen cabeceras para los atributos `Student`: `stuId`, `lastName`, `firstName`, `major` y `credits`. La tabla `Class` tiene una columna para cada uno de sus atributos: `classNumber`, `facId`, `schedule` y `room`. La tabla `Faculty` tiene columnas para `facId`, `name`, `department` y `rank`. La tabla `Enroll` se usa para mostrar la relación entre las tablas `Student` y `Class` al incluir las columnas `stuId` y `classNumber`, las principales entradas de estas dos tablas. También tiene el atributo `grade`, que pertenece a la relación. Note que los registros en la tabla `Enroll` muestran cuáles estudiantes están inscritos en cuáles clases. En posteriores capítulos se estudiará el modelo relacional, incluido un lenguaje estándar para el modelo llamado SQL. Dos antiguos modelos basados en registro son los modelos **red** y **jerárquico** mencionados en la sección 1.7. Son principalmente de interés histórico, pues ya no se usan con amplitud para desarrollar nuevas bases de datos. Sin embargo, todavía existen muchas bases de datos legadas con base en estos modelos, con código que todavía se usa y que se debe mantener. El apéndice B discute el modelo red y el apéndice C cubre el modelo jerárquico.

2.7.3 Modelo orientado a objeto

El modelo orientado a objeto es un modelo semántico similar al E-R. Extiende los conceptos E-R al agregar **encapsulado**, un medio de incorporar datos y funciones en una unidad donde están protegidos de modificación desde el exterior. En oposición a las entidades que tienen atributos como en E-R, este modelo usa **objetos** que tienen tanto un **estado** como un **comportamiento**. El estado de un objeto se determina mediante los valores de sus atributos (variables de instancia). El comportamiento es el conjunto de métodos (funciones) definidos por el objeto. Al crear una base de datos orientada a objeto, el diseñador comienza por definir **clases**, que especifican los atributos y métodos para un conjunto de objetos. Entonces cada objeto se crea al crear una instancia de la clase, usando uno de los propios métodos de la clase llamado **constructor**. La estructura de los objetos puede ser bastante compleja. Cada objeto en una base de datos debe tener un **identificador de objeto** único que funcione como una clave primaria permanente, pero ésta no toma su valor de alguno de los atributos del objeto. Las clases que se relacionan una con otra se agrupan para formar **jerarquías de clase**. Los lectores familiarizados con la programación orientada a objeto reconocerán estos conceptos. Una diferencia importante entre los objetos de programa y los objetos de base de datos es la **persistencia**. A diferencia de un objeto de programa que existe sólo mientras el programa se ejecuta, un objeto de base de datos permanece en existencia después de que la ejecución de un programa de aplicación se completa.

2.7.4 Modelo objeto-relacional

El modelo objeto-relacional extiende el modelo relacional al agregarle algunos tipos de datos y métodos complejos. En lugar de atributos atómicos de un solo valor, como se requiere en el modelo relacional, este modelo permite que los atributos se estructuren y tengan conjuntos o arreglos de valores. También permite herencia de métodos y tipo. El lenguaje SQL se extendió en 1999 para crear y manipular los tipos de datos más complejos que soporta este modelo. Por tanto, el lenguaje utilizado para administrar una base de datos objeto-relacional está más cerca del tipo de lenguaje que se utiliza para las bases de datos relacionales que el utilizado para bases de datos estrictamente orientadas a objeto.

2.7.5 Modelo de datos semiestructurado

La mayoría de los modelos de datos requieren que los tipos de entidad (u objetos o registros, dependiendo del modelo) tengan la misma estructura. La estructura se define en el esquema y permanece invariable a menos que el ABD cambie el esquema. En contraste, el modelo semiestructurado permite una colección de nodos, cada uno conteniendo datos, posiblemente con diferentes esquemas. El nodo en sí contiene información acerca de la estructura de sus contenidos. Las bases de datos semiestructuradas son especialmente útiles cuando se deben integrar bases de datos existentes que tengan distintos esquemas. Las bases de datos individuales se pueden ver como documentos, y a cada documento se pueden agregar etiquetas (tags) XML (Extensible Markup Language = lenguaje de marcas extensible) para describir sus contenidos. XML es un lenguaje similar a HTML (Hypertext Markup Language = lenguaje de marcas de hipertexto), pero se usa como un estándar para intercambio de datos en vez de para presentación de datos. Las etiquetas XML se usan para identificar elementos, subelementos y atributos que almacenan datos. El esquema se puede identificar usando un Document Type Definition (DTD, definición de tipo de documento) o mediante un esquema XML que identifique los elementos, sus atributos y sus relaciones mutuas.

2.8 Resumen del capítulo

Los datos operativos de una corporación son un recurso corporativo, un activo que es valioso para la organización e incurre en costos. Una base de datos, compartida por muchos usuarios, protege los datos operativos al proporcionar seguridad de datos, restricciones de integridad, controles de confiabilidad y gestión profesional por un ABD.

Datos significa hechos, mientras que la información son datos procesados en una forma útil para tomar decisiones. Existen cuatro niveles de discusión acerca de los datos: **realidad** (el mundo real) que contiene el **minimundo** (Universo de Discurso) que se modela, el **modelo conceptual** del minimundo, **metadatos** (datos acerca de datos) y **datos de base de datos** (instancias de datos). A partir de los objetos en el minimundo se identifican **entidades**, **conjuntos de entidad**, **atributos** y **relaciones**. La estructura de la base de datos se llama **modelo lógico de la base de datos**. El **esquema** de la base de datos, una descripción de la estructura, se registra en el diccionario de datos. El diccionario de datos contiene **metadatos**, o datos acerca de datos, y dice cuáles **tipos de registro**, **tipos de ítem de datos** y **agregados de datos** existen en la base de datos. La base de datos en sí contiene **instancias de datos** u ocurrencias. Existe un registro en la base de datos para cada entidad en el mundo real. Un **archivo** es una colección nominada de ocurrencias de registro. Una **base de datos** es una colección de archivos.

Un abordaje escalonado al diseño de bases de datos es un abordaje de arriba abajo que permite al diseñador desarrollar un modelo conceptual que refleje las operaciones de la organización, permita cambios, soporte muchas vistas de usuario, sea independiente de la implementación física y no dependa del modelo de un DBMS particular. Este abordaje permite la evolución de la base de datos según cambien las necesidades. En el diseño escalonado de la base de datos el diseñador debe analizar el entorno del usuario, desarrollar un modelo de datos conceptual, elegir el DBMS, crear el modelo lógico al mapear el modelo conceptual al modelo de datos del DBMS, desarrollar los modelos interno y físico, evaluar el modelo físico, realizar afinación si se necesita e implementar el modelo físico. Los pasos se pueden repetir hasta que el diseño sea satisfactorio.

Las metodologías de diseño pueden ser las técnicas generales descritas en la literatura o productos comerciales como los paquetes **CASE**. El **diccionario de datos** puede ser **integrado** (parte del DBMS) o **independiente** (del DBMS). El DBMS puede usar un diccionario de datos integrado siempre que el DBMS acceda a la base de datos y la actualice automáticamente. Un diccionario independiente es útil en las primeras etapas del diseño, antes de elegir un DBMS. Un diccionario de datos es una herramienta valiosa para recopilar y organizar información acerca de los datos. Es útil para recopilar información acerca de los datos en una ubicación central, asegurar acuerdo acerca de los significados de los ítems de datos, comunicarse con los usuarios, identificar redundancia e inconsistencia (incluidos **sinónimos** y **homónimos**), registrar cambios a la estructura de la base de datos, determinar el impacto de tales cambios, identificar fuentes de y responsabilidad para la exactitud de los ítems, registrar modelos externos, lógicos y físicos y sus mapeos, registrar información de control de acceso y proporcionar información de auditoría.

El **administrador de la base de datos** debe ser técnicamente competente, un buen administrador y tener excelentes habilidades interpersonales y de comunicación. Su responsabilidad principal es planificar, diseñar, desarrollar y gestionar la base de datos operativa. En las etapas de planificación y diseño, las responsabilidades incluyen planificación preliminar, identificación de necesidades del usuario, desarrollo y mantenimiento del diccionario de datos, diseño del modelo conceptual, elección de un DBMS, desarrollo del modelo lógico y de los modelos interno y físico. En la fase de desarrollo, las responsabilidades son crear y cargar la base de datos, desarrollar vistas de usuario, escribir y mantener documentación, desarrollar

y fortalecer estándares de datos, crear y fortalecer estándares de programas de aplicación, desarrollo de procedimientos operativos y capacitar a usuarios. Las responsabilidades de gestionar la base de datos incluyen monitorear el desempeño, afinar y reorganizar y mantenerse actualizado en las mejoras a la base de datos.

La arquitectura estándar de bases de datos usa tres niveles de abstracción: **externo**, **lógico** e **interno**. Una **vista externa** es el modelo visto por un usuario particular y consiste en **registros externos**, que pueden ser subconjuntos o combinaciones o registros reales. Un **esquema externo** es la descripción de un modelo externo escrito en el **lenguaje de definición de datos**, que es parte del **sublenguaje de datos** para el DBMS particular que se usa. La **interfaz de usuario** crea el entorno operativo del usuario y oculta los niveles inferiores al usuario. El **esquema lógico** es una descripción DDL completa del modelo lógico. Especifica el contenido de información de la base de datos, incluidos todos los tipos de registro y campos, pero no contiene detalles del almacenamiento o la representación física. El modelo lógico es el corazón de la base de datos. El diseño conceptual es una tarea desafiante y gratificante. Un buen diseño conceptual es fácil de implementar y soporta las vistas externas deseadas. La **interfaz de registro lógica** es una frontera por abajo de la cual el nivel lógico no puede ver. El **esquema interno** es una descripción DDL del modelo interno. Especifica cómo se representan los datos, cómo se secuencian los registros, qué índices y punteros existen, y cuál esquema de claves (hashing) se usa. La **interfaz de registro almacenada** es la frontera por abajo de la cual el DBMS no ve. El sistema operativo debe responsabilizarse de todos los detalles físicos por abajo de esta interfaz, incluida la creación de una **interfaz de registro física** para que la misma manipule los detalles físicos de bajo nivel, con la colocación de pista. El **mapeo externo/lógico** dice cómo las vistas externas corresponden a los ítems lógicos. El **mapeo lógico/interno** dice cómo los ítems lógicos corresponden a los internos. La **independencia de datos** hace a cada nivel inmune a los cambios en los niveles inferiores. La **independencia de datos lógica** significa que el nivel lógico se puede cambiar sin cambiar las vistas externas. La **independencia de datos física** significa que los cambios internos y físicos no afectarán al modelo lógico.

Algunos modelos de datos son los modelos **entidad-relación**, **orientada a objeto**, **objeto-relacional** y **semiestructurado**. También existen modelos **basados en registro**, incluidos el **relacional** así como los modelos más antiguos **red** y **jerárquico**. El modelo entidad-relación usa **diagramas E-R** para mostrar conjuntos de entidad, atributos de entidades, conjuntos de relación y atributos descriptivos de relaciones. El modelo relacional usa **tablas** para representar datos y relaciones. El modelo orientado a objeto usa el concepto de **clase**, que tiene atributos y métodos. Un **objeto** se crea como una instancia de una clase. Las bases de datos orientadas a objetos contienen objetos **persistentes**. Las bases de datos objeto-relacional son extensiones de las bases de datos del modelo relacional para permitir objetos y métodos complejos. Las bases de datos semiestructuradas consisten en nodos que son autodescriptivos al usar XML.

Ejercicios

2.1 Mencione cuatro recursos de una organización empresarial común.

2.2 Distinga entre datos e información.

2.3 Identifique los cuatro niveles de abstracción en la discusión de datos. Para cada uno, proporcione un ejemplo de un ítem que aparezca en dicho nivel.

2.4 Distinga entre un conjunto de entidad y una instancia de entidad, proporcione un ejemplo de cada uno.

2.5 Distinga entre un tipo de registro y una ocurrencia de registro, proporcione un ejemplo de cada uno.

2.6 ¿Qué nivel de abstracción de datos se representa en el diccionario de datos? Proporcione ejemplos de los tipos de entradas que se almacenarían ahí.

2.7 Explique por qué un abordaje de diseño escalonado es más adecuado para el diseño de bases de datos que el abordaje tradicional de análisis de sistemas.

2.8 Mencione cinco características deseables de un modelo conceptual de una empresa.

2.9 Mencione las ocho principales etapas de diseño en el diseño escalonado de bases de datos, junto con las actividades y posibles resultados de cada una.

2.10 Explique qué se entiende al decir que el diseño escalonado de las bases de datos es un método "arriba abajo".

2.11 Explique cómo se puede usar un paquete CASE en el diseño de bases de datos.

2.12 Mencione dos ventajas y dos desventajas de lo siguiente:

a. diccionarios de datos integrados

b. diccionarios de datos independientes

2.13 Explique por qué los usuarios no deben tener acceso al diccionario de datos.

2.14 Mencione ocho usos de un diccionario de datos.

2.15 ¿Qué tipos de habilidades debe poseer un administrador de bases de datos? ¿Para qué tareas se necesitan?

2.16 Mencione las principales funciones del ABD.

2.17 Defina cada uno de los siguientes términos:

a. datos operativos
b. recurso corporativo
c. metadato
d. entidad
e. atributo
f. ítem de datos
g. agregado de datos
h. registro de datos
i. archivo de datos
j. ciclo de vida del sistema
k. prototipo
l. afinación del sistema
m. CASE
n. diccionario de datos integrado
o. sinónimo de diccionario de datos
p. homónimo de diccionario de datos
q. estándares de datos
r. ABD

2.18 Describa la arquitectura en tres niveles para las bases de datos.

2.19 Describa las dos partes de los sublenguajes de datos.

2.20 Proporcione cinco razones por las que es deseable separar la representación física de la estructura conceptual de una base de datos.

2.21 Distinga entre lo siguiente: interfaz de usuario, interfaz de registro lógica, interfaz de registro almacenada, interfaz de registro física.

2.22 Describa cada uno de los siguientes: esquema externo, esquema lógico, esquema interno.

2.23 Explique el propósito del mapeo externo/lógico y del mapeo lógico/interno.

2.24 Distinga entre independencia de datos lógica y física, y proporcione ejemplos de los posibles cambios que permiten.

2.25 Explique por qué el modelo lógico se llama el corazón de la base de datos.

2.26 Describa abstracción y explique cómo se usa en el diseño de base de datos.

2.27 Distinga entre la intensión y la extensión de una base de datos.

2.28 Explique cómo los modelos de datos basados en registros difieren de los modelos semánticos.

Ejercicios de laboratorio

Ejercicio de laboratorio 2.1: Exploración de una herramienta de diagramación

Para este ejercicio de laboratorio debe usar software de herramientas de dibujo como SmartDraw, Microsoft Visio o productos similares. Si éstos no están disponibles, puede usar la herramienta de dibujo de Microsoft Word. Dado que cada herramienta tiene requisitos y convenciones ligeramente diferentes, explore los menús para encontrar los símbolos correctos a usar.

Con la herramienta de dibujo, trace un diagrama E-R para el ejemplo University similar al que se muestra en la figura 2.10. Si en la herramienta no están disponibles todos los elementos, escriba a mano los elementos faltantes.

Ejercicio de laboratorio 2.2: Exploración de una herramienta de gestión de proyecto

Con una herramienta de gestión de proyecto como Microsoft Project, explore las opciones de la herramienta, y examine los varios tipos de diagramas y gráficos disponibles. Construya un diagrama que ilustre las fases de un proyecto de programación, con un conjunto de recursos (programadores) para cada tarea, y un cronograma para completar cada fase. Si tiene Microsoft Project o Visio, elija una gráfica de Gantt o PERT para este ejercicio. Si ninguna de éstas está disponible, use un programa de hoja de cálculo como Microsoft Excel. Use la figura 2.12 como guía.

Ejercicio de laboratorio 2.3: Construcción de un diccionario de datos simple

Con un procesador de palabra, elabore una lista de los ítems de datos en el ejemplo University, y proporcione una definición de cada ítem de datos. Indique los ítems agrupados mediante un nombre del grupo y una lista de los ítems individuales en el grupo.

PROYECTO DE MUESTRA: APLICACIÓN DE TÉCNICAS DE PLANIFICACIÓN AL PROYECTO DE GALERÍA DE ARTE

- Paso 2.1. Diseño del diccionario de datos para la Galería de Arte

Escriba un diccionario de datos orientado al usuario, que consista en una lista alfabética de cada ítem de datos referenciado en cualquier reporte o transacción de rutina, y una definición informal para cada término.

El diccionario de datos orientado al usuario para la Galería de Arte es el siguiente:

amountRemittedtoOwner Cantidad de dinero en dólares enviada a un propietario por la venta de una obra de arte.

artistAddress Dirección de correo de un artista.

artistAreaCode Código de área telefónica de un artista.

artistCity Ciudad de la dirección de correo de un artista.

artistFirstName Nombre dado que usa un artista.

artistInterviewDate Fecha cuando un representante de la galería entrevistó a un artista.

artistInterviewerName Nombre completo del representante de la galería que entrevistó al artista.

artistLastName Apellido (sobrenombre) de un artista.

artistPhone Número telefónico completo del artista.

artistSalesLastYear Cantidad total, en dólares, de las ventas de las obras de un artista durante todo el año anterior.

artistSalesYearToDate Cantidad total, en dólares, de las ventas de las obras de un artista desde el primer día del año actual hasta la fecha del reporte o transacción en la que aparece.

artistSocialSecurityNumber Registro federal de contribuyentes de un artista.

artistState Estado de la dirección de correo de un artista.

artistStreet Número de casa y calle de la dirección de correo de un artista.

artistTelephoneNumber Número telefónico de un artista, sin incluir código de área.

artistTotalAskingPriceforPeriod Valor total en dólares de las obras no vendidas de un artista para venderse en la galería por el periodo cubierto en un reporte o transacción, calculado como la suma de sus precios solicitados.

artistTotalSalesforPeriod Cantidad total, en dólares, de las ventas de las obras de un artista para el periodo cubierto en un reporte o transacción.

artistZip Código postal de la dirección de correo de un artista.

askingPrice Precio solicitado de una obra de arte.

buyerAddress Dirección de correo de un comprador de una obra de arte de la galería.

buyerAreaCode Código de área telefónica de un comprador de una obra de arte de la galería.

buyerCity Ciudad de la dirección de correo de un comprador de una obra de arte de la galería.

buyerFirstName Nombre de un comprador de una obra de arte de la galería.

buyerLastName Apellido (sobrenombre) de un comprador de una obra de arte de la galería.

buyerPhone Número telefónico completo de un comprador de una obra de arte de la galería.

buyerState Estado de la dirección de correo de un comprador de una obra de arte de la galería.

buyerStreet Número de casa y calle de la dirección de correo de un comprador de una obra de arte de la galería.

buyerTelephoneNumber Número telefónico de un comprador de una obra de arte de la galería, no incluye código de área.

buyerZip Código postal del comprador de una obra de arte.

collectionArtistFirstName Nombre del artista que se presenta en un grupo de obras de arte propiedad de un coleccionista.

collectionArtistLastName Apellido del artista que se presenta en un grupo de obras de arte propiedad de un coleccionista.

collectionMedium Medio utilizado por un grupo de obras de arte propiedad de un coleccionista.

collectionStyle Estilo de un grupo de obras de arte propiedad de un coleccionista.

collectionType Tipo de un grupo de obras de arte propiedad de un coleccionista.

collectorAddress Dirección de correo de un coleccionista de obras de arte.

collectorAreaCode Código de área telefónico de un coleccionista de obras de arte.

collectorCity Ciudad de la dirección de correo de un coleccionista de obras de arte.

collectorFirstName Nombre dado por un coleccionista de obras de arte.

collectorInterviewDate Fecha cuando un representante de la galería entrevistó a un coleccionista de obras de arte.

collectorInterviewerName Nombre completo (sobrenombre) del representante de la galería que entrevistó a un coleccionista de obras de arte.

collectorLastName Apellido (sobrenombre) de un coleccionista de obras de arte.

collectorPhone Número telefónico completo de un coleccionista de obras de arte.

collectorSalesLastYear Cantidad total, en dólares, de las ventas de las obras de arte del coleccionista durante todo el año anterior.

collectorSalesYeartoDate Cantidad total, en dólares, de las ventas de las obras de arte del coleccionista desde el primer día del año actual hasta la fecha del reporte o transacción en la que aparece.

collectorSocialSecurityNumber Registro federal de contribuyentes de un coleccionista de obras de arte.

collectorState Estado de la dirección de correo de un coleccionista de obras de arte.

collectorStreet Número de casa y calle de la dirección de correo de un coleccionista de obras de arte.

collectorTelephoneNumber Número telefónico de un coleccionista de obras de arte, no incluye código de área.

collectorTotalAskingPriceforPeriod Valor total, en dólares, de las obras no vendidas de un coleccionista para venta en la galería por el periodo cubierto en un reporte o transacción, calculado como la suma de sus precios solicitados.

collectorTotalSalesforPeriod Cantidad total, en dólares, de las obras del coleccionista para el periodo cubierto en un reporte o transacción.

collectorZip Código postal de la dirección de correo de un coleccionista de obras de arte.

dateListed Fecha cuando una obra de arte se ofreció a la venta por primera vez en la galería.

dateOfReport Fecha cuando se generó un reporte.

dateReturned Fecha cuando la obra de arte se regresó a su propietario.

dateShown Fecha cuando una obra de arte se presentó en una exposición en la galería.

medium Medio de una obra de arte. Ejemplos de valores válidos son óleo, pastel, acuarela, medio acuoso, acrílico, mármol, acero, cobre, madera, fibra, otro.

ownerAddress Dirección de correo del propietario de una obra de arte.

ownerAreaCode Código de área telefónico del propietario de una obra de arte.

ownerCity Ciudad de la dirección de correo del propietario de una obra de arte.

ownerFirstName Nombre dado que usa el propietario de una obra de arte.

ownerLastName Apellido (sobrenombre) del propietario de una obra de arte.

ownerPhone Número telefónico completo del propietario de una obra de arte.

ownerSocialSecurityNumber Registro federal de contribuyentes del propietario de una obra de arte.

ownerState Estado de la dirección de correo del propietario de una obra de arte.

ownerStreet Número de casa y calle de la dirección de correo del propietario de una obra de arte.

ownerTelephoneNumber Número telefónico del propietario de una obra de arte, no incluye código de área.

ownerZIP Código postal de la dirección de correo del propietario de una obra de arte.

potentialCustomerAddress Dirección de correo de un potencial cliente de la galería.

potentialCustomerAreaCode Código de área telefónico de un potencial cliente de la galería.

potentialCustomerCity Ciudad de la dirección de correo de un potencial cliente de la galería.

potentialCustomerDateFilledIn Fecha cuando un cliente llenó la forma de información.

potentialCustomerFirstName Nombre de un potencial cliente de la galería.

potentialCustomerLastName Apellido (sobrenombre) de un potencial cliente de la galería.

potentialCustomerPhone Número telefónico completo de un potencial cliente de la galería.

potentialCustomerState Estado de la dirección de correo de un potencial cliente de la galería.

potentialCustomerStreet Número de casa y calle de la dirección de correo de un potencial cliente de la galería.

potentialCustomerTelephoneNumber Número telefónico de un potencial cliente de la galería, no incluye código de área.

potentialCustomerZip Código postal de la dirección de correo de un potencial cliente de la galería.

preferredArtist Nombre del artista elegido como preferencia por un potencial cliente de la galería.

preferredMedium Medio elegido como preferido por un potencial cliente de la galería.

preferredStyle Estilo elegido como preferencia por un potencial cliente de la galería.

preferredType Tipo elegido como preferido por un potencial cliente de la galería.

purchasesLastYear Cantidad total, en dólares, de las ventas a un comprador durante todo el año anterior.

purchasesYearToDate Cantidad total, en dólares, de ventas a un comprador desde el primer día del año actual hasta la fecha del reporte o transacción en la que aparece.

reportStartingDate Fecha elegida como la fecha más antigua para usar información en un reporte.

reportEndingDate Fecha elegida como la fecha más reciente para usar la información en un reporte.

saleDate Fecha cuando la galería vendió una obra de arte.

saleInvoiceNumber Número impreso en la factura para una venta de una obra de arte.

salePrice Precio al que la galería vendió una obra de arte.

salesPersonAddress Dirección completa de un socio de ventas que trabaja en la galería.

salesPersonFirstName Nombre dado de un socio de ventas que trabaja en la galería.

salesPersonLastName Apellido (sobrenombre) de un socio de ventas que trabaja en la galería.

salesPersonSocialSecurityNumber Registro federal de contribuyentes de un socio de ventas que trabaja en la galería.

saleSalesPersonCommission Cantidad en dólares de la comisión para un vendedor por la venta de una obra de arte.

saleSalesPersonName Nombre y apellido del vendedor que vendió una obra de arte.

saleTax Cantidad en dólares del impuesto de ventas por la venta de una obra de arte.

saleTotal Cantidad total, en dólares, de una venta, incluidos precio e impuesto, por una obra de arte.

salespersonCommissionforPeriod Cantidad total, en dólares, de la comisión ganada por un vendedor por un periodo específico.

salespersonTotalSalesforPeriod Cantidad total, en dólares, de ventas, no incluidos impuestos, hechos por un vendedor durante un periodo específico.

showClosingDate Fecha cuando se cierra una exposición al público.

showFeaturedArtist Nombre de un artista presentado en una exposición.

showTheme Tema de una exposición.

showTitle Título dado a una exposición.

showOpeningDate Fecha de apertura de una exposición al público.

size Tamaño de una obra de arte, expresada en pulgadas. Para obras bidimensionales, largo por ancho; para obras tridimensionales, largo por ancho por altura.

status Estatus de ventas de una obra de arte. Los posibles valores son vendida o sin vender.

style Estilo artístico de una obra de arte. Ejemplos de valores válidos son contemporáneo, impresionista, folk, otro.

title Título de una obra de arte.

totalAllSalesforWeek Cantidad total, en dólares, de las ventas de la galería durante una semana específica, sin incluir impuestos.

totalAskingPriceForPeriod Suma de los precios solicitados para todas las obras durante el periodo elegido para un reporte.

type Tipo de obra de arte. Ejemplos de valores válidos son pintura, escultura, collage, otro.

usualMedium Medio que generalmente usa el artista. Ejemplos de valores válidos son óleo, pastel, acuarela, medio acuoso, acrílico, mármol, acero, cobre, madera, fibra, otro.

usualStyle Estilo artístico usual de las obras del artista. Ejemplos de valores válidos son contemporáneo, impresionista, folk, otro.

usualType Tipo de obra de arte que el artista produce normalmente. Ejemplos de valores válidos son pintura, escultura, collage, otro.

yearCompleted El año que se terminó una obra de arte.

- Paso 2.2. Modifique la lista de suposiciones (según se requiera)

  La lista de suposiciones no tiene cambios en este punto. Permanece como se muestra en el paso 1.4.

- Paso 2.3. Escriba una tabla de referencias cruzadas (que muestre cuáles ítems de datos aparecen en cuáles formatos, reportes o transacciones)

Para construir la tabla de referencia cruzada, escriba los nombres de todas las formas, reportes y transacciones como encabezados de columnas a través de la parte superior de la tabla. Escriba los ítems del diccionario de datos abajo de la primera columna, y haga una forma similar a una hoja de cálculo. Si un ítem de datos en una fila dada aparece en una forma, reporte o transacción particulares, coloque una marca de verificación en la celda para la correspondiente intersección columna-fila. La tabla de referencia cruzada para la Galería de Arte aparece en la figura 2.11.

- Paso 2.4. Cree una gráfica de gestión de proyecto con formato de Gantt o PERT

Puede usar una herramienta de gestión de proyecto como Microsoft Project para hacer un gráfico que mencione los pasos principales del proyecto y asigne un cronograma para completar todo el proyecto, y si lo realiza un individuo o un grupo. La figura 2.12 muestra un diagrama simplificado para completar las partes del proyecto de base de datos para la Galería de Arte como una gráfica de Gantt creada con Microsoft Visio. Suponga que tres personas (mencionadas bajo Recursos) trabajan en el proyecto.

PROYECTOS ESTUDIANTILES: APLICACIÓN DE LAS TÉCNICAS DE PLANIFICACIÓN A LOS PROYECTOS ESTUDIANTILES

- Paso 2.1. Diseñe el diccionario de datos para el proyecto estudiantil

Escriba un diccionario de datos orientado a usuario, que consista en una lista alfabética de todos los ítems de datos referenciados en cualquier reporte o transacción de rutina, y una definición informal para cada término.

- Paso 2.2. Modifique la lista de suposiciones

  Realice los cambios necesarios a las suposiciones.

- Paso 2.3. Escriba una tabla de referencias cruzadas

Para construir la tabla de referencias cruzadas, escriba los nombres de todas las formas, reportes y transacciones como encabezados de columnas a través de la parte superior de la tabla. Escriba los ítems del diccionario de datos abajo de la primera columna, y haga una forma similar a una hoja de cálculo. Si un ítem de datos en una fila dada aparece en una forma, reporte o transacción particulares, coloque una marca de verificación en la celda para la correspondiente intersección columna-fila.

- Paso 2.4. Planificación de gestión de proyecto para el proyecto estudiantil

  Con una herramienta de gestión de proyecto como Microsoft Project o una hoja de cálculo, elabore un gráfico que mencione las tareas principales del proyecto y asigne un cronograma para completar todo el proyecto. Divida las tareas principales en subtareas. Si el proyecto lo realiza un grupo, asigne las subtareas a los miembros del grupo. Indique la dependencia de una tarea sobre otra mediante el dibujo de flechas. Establezca fechas límite según requiera para completar el proyecto a tiempo.

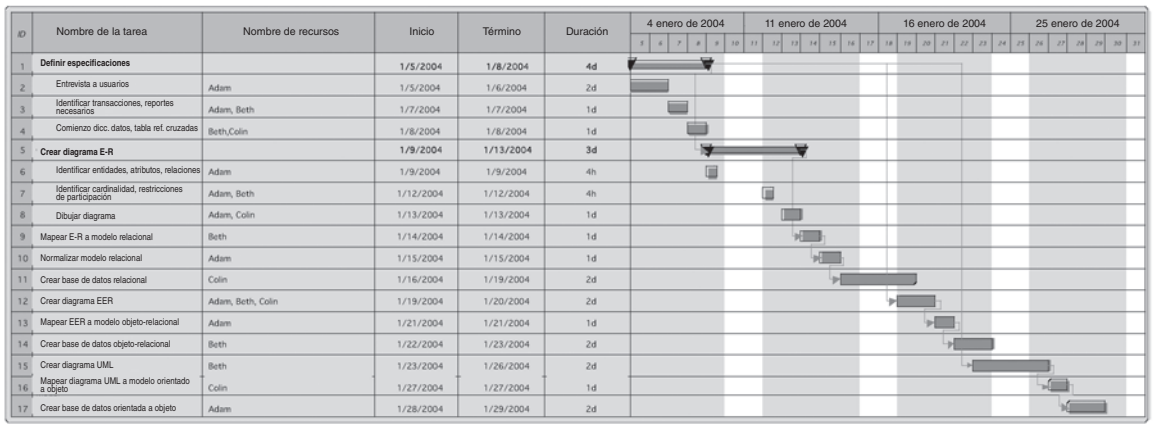
	Documento o forma en la que aparece el ítem																
	ArtistInfo	CollInfo	ArtwrkInfo	SaleInv	MailList	ActvArtsts	IndArtstSa	CollSum	IndColSal	WksForSal	SalThisWk	BuyerSals	PrefCust	SalpPerfor	AgedArt	PaymtStub	ArtShow
Ítem																	
amountRemittedtoOwner																X	
artistAddress	X					X	X										
artistAreaCode	X					X	X										
artistCity	X					X	X										
artistFirstName	X		X	X		X	X		X	X	X	X	X	X	X	X	X
artistInterviewDate	X																
artistInterviewerName	X																
artistLastName	X		X	X		X	X		X	X	X	X	X	X	X	X	X
artistPhone	X					X	X										
artistSalesLastYear						X											
artistSalesYearToDate						X											
artistSocialSecurityNumber	X																
artistState	X					X	X										
artistStreet	X					X	X										
artistTelephoneNumber	X					X	X										
artistTotalSalesforPeriod							X										
artistTotalAskingPriceforPeriod							X										
artistZip	X					X											
askingPrice			X				X		X	X					X		X
buyerAddress				X								X					
buyerAreaCode				X								X					
buyerCity				X								X					
buyerFirstName				X							X	X					
buyerLastName				X							X	X					
buyerphone				X								X					
buyerState				X								X					
buyerStreet				X								X					
buyerTelephoneNumber				X								X					
buyerZip				X								X					
collectionArtistFirstName		X															
collectionArtistLastName		X															
collectionMedium		X															
collectionStyle		X															
collectionType		X															
collectorAddress		X						X	X								

FIGURA 2.11

Tabla de referencias cruzadas

	ArtistInfo	CollInfo	ArtwrkInfo	SaleInv	MailList	ActvArtsts	IndArtstSa	CollSum	IndColSal	WksForSal	SalThisWk	BuyerSals	PrefCust	SalpPerfor	AgedArt	PaymtStub	ArtShow
collectorAreaCode		X						X	X								
collectorCity		X						X	X								
collectorFirstName		X						X	X								
collectorInterviewDate		X															
collectorInterviewName		X															
collectorLastName		X						X	X								
collectorPhone		X						X	X								
collectorSalesLastYear								X									
collectorSalesYearToDate								X									
collectorSocialSecurityNumber		X															
collectorState		X						X	X								
collectorStreet		X						X	X								
collectorTelephoneNumber		X						X	X								
collectorTotalSalesforPeriod									X								
collectorTotalAskingPriceforPeriod									X								
collectorZip		X						X	X								
dateListed			X				X		X	X					X		
dateReturned			X				X		X								
dateShown										X							
dateOfReport						X	X		X	X		X			X		
medium			X				X		X	X			X			X	
ownerAddress			X	X												X	
ownerAreaCode			X	X													
ownercity			X	X												X	
ownerFirstName			X	X							X				X	X	
ownerLastName			X	X							X				X	X	
ownerhone			X	X											X		
ownerSocialSecurityNumber			X													X	
ownerState			X													X	
ownerStreet			X													X	
ownerTelephoneNumber			X												X		
ownerZip			X										X				
potentialCustomerAddress					X								X				
potentialCustomerAreaCode					X												
potentialCustomerCity					X								X				
potentialCustomerDateFilledIn					X												
potentialCustomerFirstName					X								X				
potentialCustomerLastName					X								X				
potentialCustomerPhone					X												

	ArtistInfo	CollInfo	ArtwrkInfo	SaleInv	MailList	ActvArtsts	IndArtstSa	CollSum	IndColSal	WksForSal	SalThisWk	BuyerSals	PrefCust	SalpPerfor	AgedArt	PaymtStub	ArtShow
potentialCustomerState					X								X				
potentialCustomerStreet					X								X				
potentialCustomerTelephoneNumber					X												
potentialCusomerZip					X								X				
preferredArtist					X			X					X				
preferredMedium					X			X					X				
preferredStyle					X			X					X				
preferredType					X			X					X				
purchasedLastYear												X					
purchasesYearToDate												X					
reportStartingDate														X			
reportEndingDate														X			
saleDate				X		X			X		X			X			
saleInvoiceNumber				X													
salePrice				X		X			X		X			X		X	
salesPersonAddress														X			
salesPersonFirstName				X							X			X		X	
salesPersonLastName				X							X			X		X	
salesPersonSocialSecurityNumber														X			
saleSalesPersonCommission											X						
saleSalesPersonName				X							X					X	
saleTax				X												X	
saleTotal				X												X	
salespersonCommissionForPeriod														X			
salespersonTotalSalesForPeriod											X			X			
showClosingDate																	X
showFeaturedArtist																	X
showTheme																	X
showTitle																	X
showOpeningDate																	X
size			X													X	
status																	X
style			X			X			X	X			X			X	
title			X	X		X			X	X	X	X	X	X	X	X	X
totalAllSalesForWeek											X						
totalAskingPriceForPeriod						X			X	X		X					
type			X			X			X	X			X			X	
usualMedium	X				X												
usualStyle	X				X												
usualType	X				X												
yearCompleted			X			X			X								



ID	Nombre de la tarea	Nombre de recursos	Inicio	Término	Duración
1	**Definir especificaciones**		**1/5/2004**	**1/8/2004**	**4d**
2	Entrevista a usuarios	Adam	1/5/2004	1/6/2004	2d
3	Identificar transacciones, reportes necesarios	Adam, Beth	1/7/2004	1/7/2004	1d
4	Comienzo dicc. datos, tabla ref. cruzadas	Beth,Colin	1/8/2004	1/8/2004	1d
5	**Crear diagrama E-R**		**1/9/2004**	**1/13/2004**	**3d**
6	Identificar entidades, atributos, relaciones	Adam	1/9/2004	1/9/2004	4h
7	Identificar cardinalidad, restricciones de participación	Adam, Beth	1/12/2004	1/12/2004	4h
8	Dibujar diagrama	Adam, Colin	1/13/2004	1/13/2004	1d
9	Mapear E-R a modelo relacional	Beth	1/14/2004	1/14/2004	1d
10	Normalizar modelo relacional	Adam	1/15/2004	1/15/2004	1d
11	Crear base de datos relacional	Colin	1/16/2004	1/19/2004	2d
12	Crear diagrama EER	Adam, Beth, Colin	1/19/2004	1/20/2004	2d
13	Mapear EER a modelo objeto-relacional	Adam	1/21/2004	1/21/2004	1d
14	Crear base de datos objeto-relacional	Beth	1/22/2004	1/23/2004	2d
15	Crear diagrama UML	Beth	1/23/2004	1/26/2004	2d
16	Mapear diagrama UML a modelo orientado a objeto	Colin	1/27/2004	1/27/2004	1d
17	Crear base de datos orientada a objeto	Adam	1/28/2004	1/29/2004	2d

FIGURA 2.12

Gráfica de Gantt para el proyecto de base de datos de la Galería de Arte

CAPÍTULO 3

El modelo entidad-relación

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Qué es un esquema de empresa
- El significado de tipo de entidad, conjunto de entidades e instancia de entidad
- Cómo representar entidades en el diagrama entidad-relación (E-R)
- El significado de atributo
- Cómo los atributos se asocian con las entidades
- El significado de dominio de atributo
- Qué es un valor nulo
- El significado de atributos multivaluados, compuestos y derivados
- El significado de superclave, clave candidata, clave primaria, clave alternativa, clave secundaria y clave externa
- El significado de tipo de relación, conjunto de relación e instancia de relación
- Cómo representar conjuntos de relaciones como pares, tripletas o *n*-tuplas ordenadas
- Cómo representar conjuntos de relación y sus atributos en un diagrama E-R

- El significado y representación de la cardinalidad de una relación
- El significado y representación de restricciones de participación
- Cuándo y cómo indicar roles en un diagrama E-R
- El significado de existencia de dependencia y conjuntos de entidad débiles

3.1 Propósito del modelo E-R

El modelo entidad-relación lo desarrolló P. P. Chen en 1976 para facilitar el diseño de bases de datos al permitir al diseñador expresar las propiedades conceptuales de la base de datos en un **esquema de empresa**. La palabra empresa se usa ampliamente en las discusiones de bases de datos para significar la organización para la cual se conserva la base de datos. La empresa podría ser un pequeño negocio, una corporación, una universidad, una agencia gubernamental, un hospital o alguna otra organización. El esquema de empresa es una descripción que corresponde al modelo conceptual. Es independiente de cualquier DBMS particular. Por tanto, no está limitado al lenguaje de definición de datos de algún DBMS particular. Usa sus propios diagramas E-R para expresar la estructura del modelo. Algunos de los símbolos de diagrama y sus usos se describieron en las figuras 2.9 y 2.10. Debido a su independencia de un DBMS, el esquema de empresa será válido sin importar el sistema de gestión elegido, y puede permanecer correcto incluso si cambia el DBMS. A diferencia de un esquema escrito en un DDL, los diagramas E-R que se usarán por lo general no están disponibles para su uso por el DBMS para crear la estructura lógica o hacer relaciones externas/lógicas o lógicas/internas. Note que hay herramientas de software que en realidad usan diagramas E-R para crear las estructuras de bases de datos. Sin embargo, se verán los diagramas como herramientas de diseño y se discutirá cómo se pueden usar para implementar una diversidad de sistemas. También note que la discusión del modelo E-R aquí difiere ligeramente del modelo de Chen. Se agregaron conceptos y se usó terminología que será útil en discusiones posteriores.

En la sección 2.7 se clasificó el modelo E-R como un modelo semántico, uno que intenta capturar significados así como estructura. Existe un esfuerzo real por hacer que los ítems en el modelo representen "cosas" en el minimundo, la parte del mundo real que modelará la base de datos, y por expresar las relaciones entre "cosas" del mundo real mediante relaciones en el modelo. El modelo describe el entorno del minimundo en términos de **entidades**, **atributos** y **relaciones**. La figura 2.9 muestra los símbolos básicos para diagramas E-R. Un **rectángulo** se usa para representar una entidad, un **óvalo** para representar un atributo y un **diamante** para representar una relación. Estos elementos se conectan mediante líneas, como se muestra en las figuras 2.9 y 2.10. La figura 3.1 es una versión mejorada de la figura 2.10, que muestra un diagrama E-R simplificado para una base de datos universitaria que representa información acerca de estudiantes, personal docente y clases. El conjunto de entidades Student, que se muestra como un rectángulo, tiene los atributos stuId, lastName, firstName, major y credits, y cada uno se muestra como un óvalo conectado al rectángulo para la entidad Student. El conjunto de entidades Faculty tiene los atributos facId, lastName, firstName y rank. El conjunto de entidades Class tiene los atributos classNumber, schedule y room. El diamante marcado Enroll muestra que hay una relación entre estudiantes y las clases que cursan. La relación Teaches (enseña) conecta al personal docente con las clases que imparten.

3.2 Entidades

No se dará una definición formal del término **entidad** sino que, informalmente, se describirá como algún objeto que existe y se puede distinguir de otros objetos. Puede representar una persona, lugar, evento, objeto o concepto en el mundo real que se planea modelar en la base de datos. Puede ser un objeto físico o una abstracción. Diferentes diseñadores pueden no estar de acuerdo acerca de qué entidades existen en el minimundo. Las **instancias** de entidad representan a un estudiante particular, una clase específica, un cliente individual, un empleado particular, una cuenta, un paciente, una conferencia, un invento o un club,

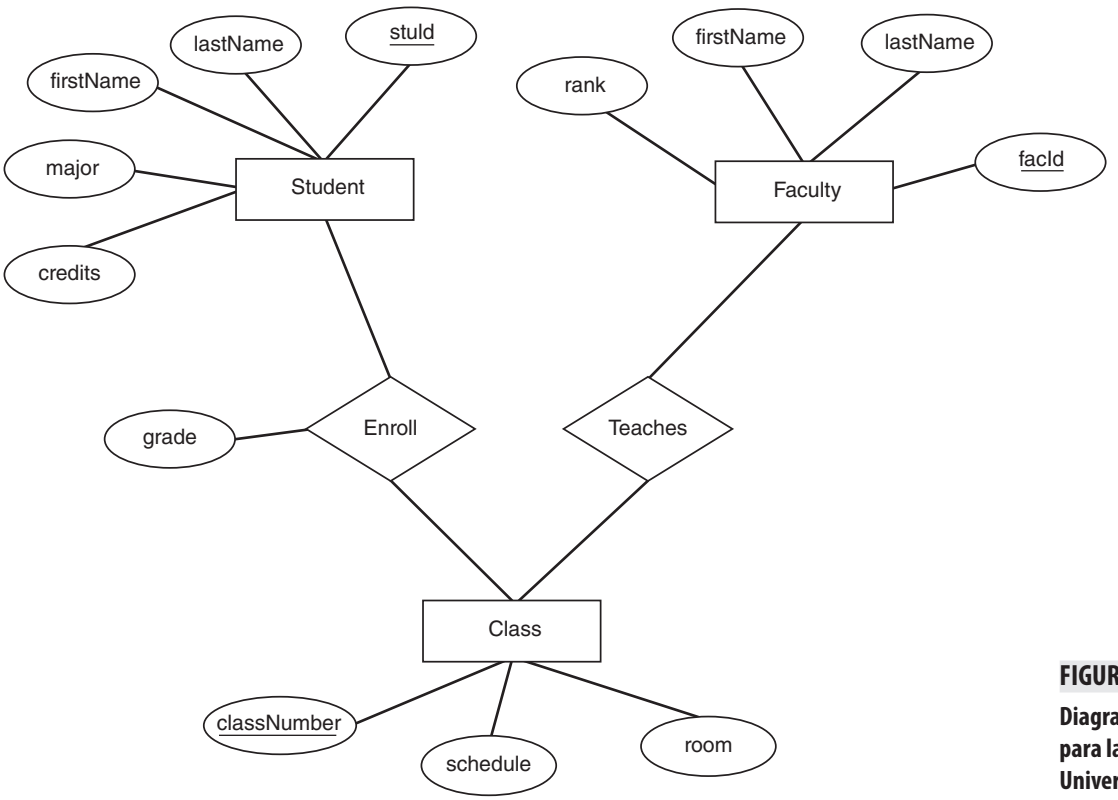


FIGURA 3.1
Diagrama E-R simplificado para la base de datos University

dependiendo de cuál sea la empresa y qué partes de ella se desee representar. Al aplicar abstracción es posible identificar las propiedades comunes de las instancias de entidades que son de interés en la base de datos y definir un **tipo de entidad**, que es una representación en el modelo de datos de una categoría de entidades. Por ejemplo, si la empresa es una universidad, se puede considerar a todos los estudiantes en la universidad e identificar las propiedades comunes de interés para el tipo de entidad Student. El tipo de entidad forma la **intensión** de la entidad, la parte de definición permanente. Una colección de entidades del mismo tipo se llama **conjunto de entidades**. El conjunto debe estar **bien definido**, lo que significa que debe ser posible determinar si una instancia de entidad particular pertenece a ella o no. Todas las instancias de entidad que satisfacen la definición en el momento forman la **extensión** de la entidad. Los miembros del conjunto de entidades Student cambian conforme los estudiantes entran y salen, pero el tipo de entidad Student permanece constante. Los conjuntos de entidades pueden intersecarse, esto es, tener miembros comunes. Por ejemplo, en el modelo de la universidad se puede tener un tipo de entidad faculty y un tipo de entidad administrator. Una persona particular puede satisfacer la definición de ambos tipos y ser simultáneamente tanto un miembro del personal docente como un administrador en la universidad, y por tanto, sería una instancia en estos dos conjuntos de entidades. Como se muestra en la figura 3.1, un tipo de entidad se representa en el diagrama E-R mediante un rectángulo que tiene el nombre de la entidad en su interior.

3.3 Atributos

Los **atributos** de una entidad representan las propiedades definitorias o cualidades del tipo de entidad. Para el tipo de entidad student, las propiedades definitorias pueden ser la ID, nombre, especialidad y número de créditos acumulados del estudiante. Los atributos son la

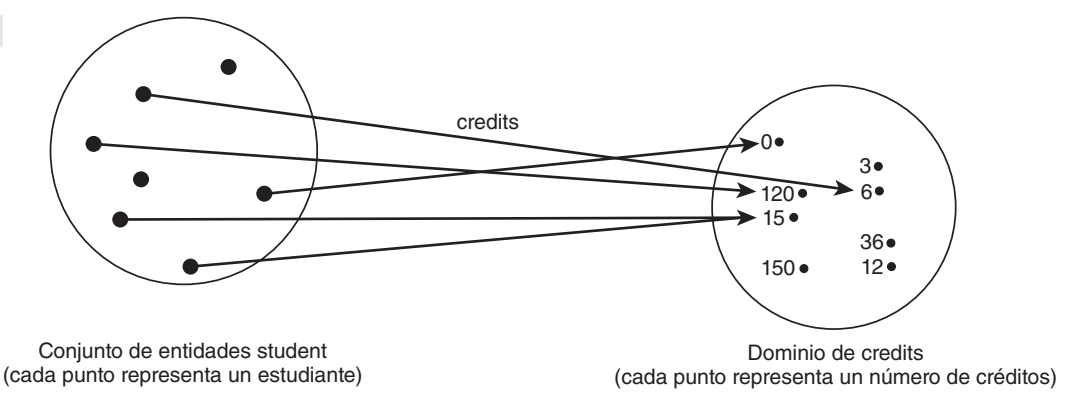
representación en el modelo de dichas propiedades, a saber: stuId, stuLastName, stuFirstName, major y credits. Por lo general, una entidad tendrá un valor para cada uno de sus atributos. Un atributo se representa en un diagrama E-R mediante un óvalo con el nombre del atributo en el interior. Una línea conecta el óvalo del atributo con el rectángulo del conjunto de entidades que describe. La figura 3.1 muestra varios ejemplos de atributos conectados a sus tipos de entidad. Tal como con las entidades, diferentes diseñadores pueden no estar de acuerdo acerca de los atributos para un conjunto de entidades. Por ejemplo, otro diseñador puede elegir incluir stuAddress, stuPhone y stuStatus, pero no credits. Además, lo que parece ser un atributo para un diseñador puede ser una entidad para otro. Por ejemplo, major puede verse como una entidad en un diseño diferente. Al elegir si un objeto debe ser una entidad o un atributo, el diseñador consideraría si el objeto describe a otro objeto y si tiene valores para sus instancias. En dicho caso, es mejor representar el objeto como un atributo. Si es difícil identificar los posibles valores, es más probable que el objeto sea una entidad.

3.3.1 Dominios

El conjunto de valores permitidos para cada atributo se llama **dominio** de dicho atributo. Para el ejemplo Student, el dominio del atributo credits puede ser el conjunto de valores enteros entre 0 y 150 inclusive, dependiendo de cómo la universidad calcula las horas de créditos. El dominio del atributo stuLastName es un poco más difícil de definir, pues consiste de todos los apellidos legales de estudiantes. Ciertamente es una cadena, pero puede consistir no sólo de letras sino de apóstrofos, espacios en blanco, guiones u otros caracteres especiales. Diferentes atributos pueden tener los mismos dominios. Por ejemplo, un subconjunto del conjunto de enteros positivos con frecuencia se usa como dominio para atributos con significados muy distintos, como créditos y edad. En realidad un atributo mapea un conjunto de entidades en el dominio del atributo. Por ejemplo, el atributo credits es una función que toma el conjunto de estudiantes y mapea cada estudiante a un valor específico en el dominio {0,…, 150}. La figura 3.2 ilustra credits como una función que relaciona el conjunto de entidades Student con el dominio credits. (Nota: Esta figura no es parte de un diagrama E-R, sino un medio de ilustrar visualmente este concepto.) Es posible que note que la palabra dominio, como se usa aquí, no coincide con la noción matemática de dominio como el conjunto sobre el cual se define una función. De hecho, el dominio del atributo es en realidad el rango de una función matemática. Una instancia de entidad particular se podría describir como un conjunto de pares ordenados, donde cada punto es el nombre de un atributo y el valor del atributo. Para un estudiante específico, tal conjunto puede ser {(stuId, S1001), (stuLastName, Smith), (stuFirstName, Jack), (major, History), (credits, 90)}. El atributo nombrado en sí y su dominio son parte de la intensión del modelo, mientras que

FIGURA 3.2

Credits como el mapeo de la entidad student con el dominio credits



los valores del atributo son parte de la extensión. Note que algunos miembros de ambos conjuntos no están mapeados en la figura 3.2.

3.3.2 Valores nulos

En ocasiones el valor de dicho atributo se desconoce en el momento actual o no está definido para una instancia particular. En una base de datos, a algunos atributos se les puede permitir tener **valores nulos** para algunas instancias de entidades. En dicho caso, la instancia de entidad no se mapeará al dominio del atributo, aunque otras instancias del mismo conjunto de entidades se mapearán al dominio de atributos. En la figura 3.2 algunos miembros del conjunto de entidades student no se conectaron mediante flechas al dominio de credits. Note que los valores de cero o una cadena en blanco para un campo de cadena de caracteres se consideran como entradas no nulas. Nulo significa sin valor.

3.3.3 Atributos multivaluados

Algunos atributos pueden tener **valores múltiples** para una instancia de entidad. Por ejemplo, los estudiantes pueden tener más de una dirección de correo electrónico. Si es posible que alguna instancia de entidad tenga valores múltiples para un atributo particular, se usa un **óvalo doble** alrededor del nombre del atributo. El óvalo doble no se debe interpretar como que todas las instancias deban tener valores múltiples, sólo que algunas instancias pueden tenerlos. La figura 3.3 ilustra cómo aparecería en un diagrama E-R múltiples direcciones de correo electrónico (emailAddress) para un estudiante.

3.3.4 Atributos compuestos

Algunos atributos se pueden descomponer en elementos más pequeños. Por ejemplo, la dirección se puede descomponer en calle, ciudad, estado y código postal. Si se examina classNumber, se ve que consiste en un código de departamento, un número de curso dentro de dicho departamento y una letra para una sección. Si se usa stuName como atributo, se podría descomponer en firstName y lastName. De igual modo, telephoneNumber se puede descomponer en areaCode, phoneNumber o en countryCode, areaCode, exchange y extension. Un atributo es un atributo **compuesto** si es posible descomponerlo todavía más. Se indica que un atributo es compuesto al escribir su nombre en un óvalo en la forma usual y luego dibujar óvalos para los componentes individuales, que se conectan mediante líneas al óvalo del atributo compuesto. La figura 3.4 ilustra la dirección como un atributo compuesto.

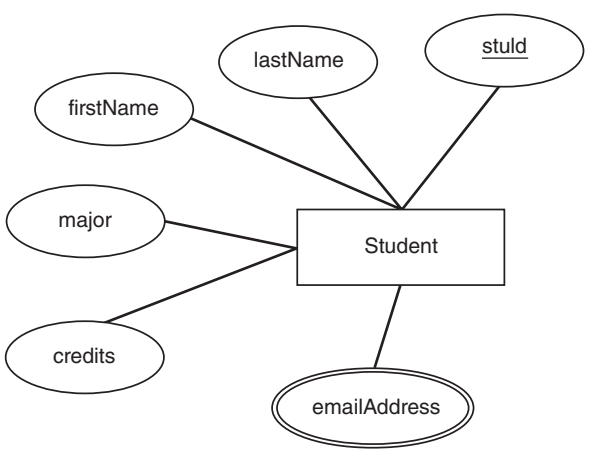


FIGURA 3.3
Conjunto de entidades Student con atributo multivaluado emailAddress

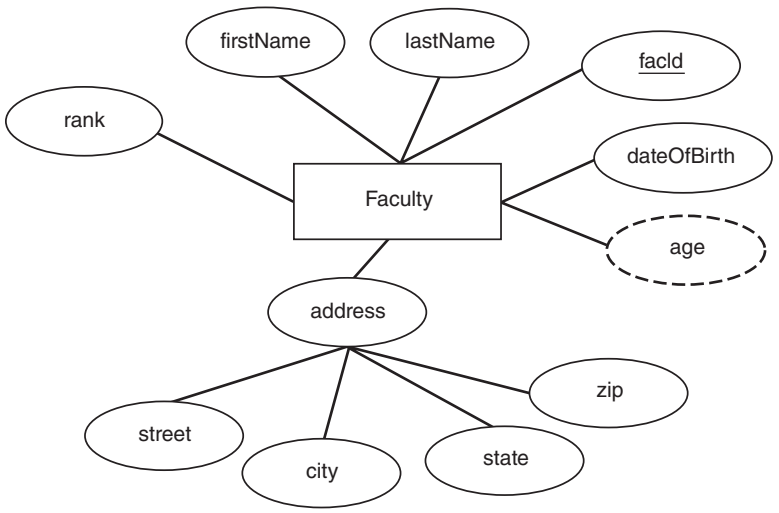


FIGURA 3.4
Conjunto de entidades Faculty con atributo compuesto dirección (address) y atributo derivado edad (age)

3.3.5 Atributos derivados

A veces es posible que quiera incluir en un diseño un atributo cuyo valor se pueda calcular cuando sea necesario. Por ejemplo, es posible que quiera tratar la edad como si fuese un atributo, pero si ya almacenó dateOfBirth (fecha de nacimiento) no hay necesidad de también almacenar la edad, pues se calcula fácilmente. Los atributos que no se almacenarán, pero cuyos valores se calcularán u obtendrán a partir de otras fuentes, se llaman **derivados**. Éstos se citan en un diagrama E-R mediante un **óvalo con rayas**. La figura 3.4 también muestra la edad como un atributo derivado. Los atributos también se pueden derivar a partir de otras entidades o de relaciones. Por ejemplo, se podría tener un atributo currentEnrollment (inscripción actual) para Class, que muestre el número de estudiantes inscritos. El valor se podría derivar a partir del número de relaciones de inscripción (enrollment) para la entidad class. Si se almacenó Department como entidad, se podría tener un atributo derivado, numberOfFaculty (número de profesores), que haría un conteo de los miembros del personal docente para cada departamento. Esto se podría derivar a partir de la entidad faculty.

3.4 Claves

Intuitivamente, se piensa en una **clave** como en un ítem de datos que permite diferenciar los registros. Es necesaria una definición más exacta del concepto de clave. Comience con la noción de superclave.

3.4.1 Superclaves

Una **superclave** es un atributo o un conjunto de atributos que identifican de manera única una entidad. Eso significa que siempre permite diferenciar una instancia de entidad de otra. Por ejemplo, para el conjunto de entidades Student, {stuId} es una superclave porque se puede usar para identificar de manera única cada estudiante. Si tiene una lista de todos los estudiantes, con sus ID, apellidos, nombres, especialidades y créditos, y se le dice el valor stuId, sólo hay un estudiante en la lista con dicho valor. Si se le dice el apellido del estudiante, es posible que no esté seguro de cuál estudiante elegir, pues dos o más estudiantes pueden tener el mismo apellido. Por tanto, {stuLastName} no es una superclave. Si tiene una superclave, entonces cualquier conjunto de atributos que contenga dicha superclave tam-

bién es una superclave. En consecuencia, la combinación de stuId y credits, escrito {stuId,credits}, también es una superclave. Note que una superclave proporciona identificación única para todas las extensiones de la base de datos, no sólo para uno o dos ejemplos. Puede usar ejemplos para auxiliarse a reparar las ideas acerca de las superclaves, pero un ejemplo puede ser engañoso. Por ejemplo, puede suceder que, en el momento, ningún par de estudiantes en la universidad tenga el mismo apellido y pueda inferir de manera incorrecta a partir de la extensión particular que {stuLastName} es una superclave. Para identificar una superclave requiere considerar el significado de los atributos, una noción semántica, antes de decidir si es única sobre todas las extensiones. Las superclaves representan una restricción que evita que dos entidades tengan alguna vez el mismo valor para dichos atributos. Representa una suposición hecha acerca del minimundo que se usa en el modelo.

3.4.2 Claves candidatas

Dado que una superclave como {stuId,credits} puede contener atributos adicionales que no son necesarios para identificación única de instancias de entidad, el interés está en encontrar superclaves que no contengan estos atributos adicionales. En este ejemplo, el atributo adicional es claramente credits. Una **clave candidata** es aquella que no contiene atributos adicionales. Una clave candidata se define como una superclave tal que ningún subconjunto propio de sus atributos sea por sí mismo una superclave. En el ejemplo, {stuId,credits} no es una clave candidata porque contiene un subconjunto, {stuId}, que es una superclave. Sin embargo, {stuId} por sí mismo es una clave candidata, pues no tiene subconjunto propio que identifique entidades. Puede haber muchas claves candidatas para un conjunto de entidades. Si se almacenan números de seguridad social de estudiantes, entonces {socSecNo} también sería una clave candidata, siempre que cada estudiante tenga un número de seguridad social. Note que una clave candidata puede consistir en un solo atributo, como {stuId} y {socSecNo}, o puede ser una combinación de atributos. Por ejemplo, la combinación {lastName,firstName,Address}, si siempre es única, puede ser una clave candidata para algún conjunto de entidades. Cuando una clave consiste en más de un atributo, se le llama **clave compuesta**. Por conveniencia, ahora se retirarán las llaves en las claves de identificación y simplemente se citará(n) el (los) atributo(s) en la clave.

3.4.3 Claves primarias

Un conjunto de entidades puede tener varias claves candidatas. El diseñador de la base de datos elige entre ellas e identifica una como la forma normal de identificar entidades y acceder a los registros. Ésta se convierte en la **clave primaria**. En otras palabras, la clave primaria es la clave candidata "triunfadora", aquella que en realidad se elige. La clave primaria puede ser una sola clave de atributo o una clave compuesta. Con frecuencia, las otras claves candidatas se convierten en **claves alternativas**, cuyos valores únicos proporcionan otro método de acceder a los registros. El término **clave secundaria** por lo general significa un atributo o conjunto de atributos cuyos valores, no necesariamente únicos, se usan como un medio de acceder a los registros. En el ejemplo, stuId puede ser la clave primaria para el conjunto de entidades Student. Si socSecNo también se almacena, puede ser una clave alternativa. El atributo lastName se puede usar como una clave secundaria. A pesar de que se permitan apellidos duplicados, puede usar el apellido para auxiliarse a encontrar el registro de un estudiante si no se conoce la ID del estudiante o su número de seguridad social. Por lo general se crea un índice en un campo de clave secundaria, lo que permite rápida recuperación de registros con un valor particular del atributo indexado. Entonces los registros se pueden examinar individualmente para encontrar el deseado. Para el conjunto de entidades Class, course# puede ser la clave primaria, y facId puede ser la clave primaria para el conjunto de entidades Faculty. Una característica importante de una clave primaria es que nin-

guno de sus atributos tendrá valores nulos. Si permitiera valores nulos en las claves, no tendría posibilidad de diferenciar las entidades, pues dos entidades con valores nulos para el mismo atributo de clave serían indistinguibles. Esto se sigue de la definición de clave candidata, de modo que todos los atributos en una clave candidata son necesarios para una identificación única de las instancias de entidad. Para garantizar la exactitud de los datos, también se debe insistir en que ningún atributo de una clave candidata tiene permiso de poseer valores nulos. Al crear una base de datos, por lo general se identifica la clave primaria de modo que el sistema fortalecerá las propiedades no nula y única, y se reforzará el estatus de clave de las claves candidatas al especificar que deben ser únicas. La clave primaria de una entidad se subraya en el diagrama E-R, como se muestra en la figura 3.1.

Por cuestiones completivas, aquí se anota que existe un concepto llamado **clave externa**. Sin embargo, pertenece al modelo relacional y no es parte del modelo entidad-relación. Se discutirá en el capítulo 4.

3.5 Relaciones

Con frecuencia las entidades se ligan mediante asociaciones o **relaciones**, que son conexiones o interacciones entre las instancias de entidad. Un estudiante se relaciona con una clase al inscribirse en dicha clase. Por abstracción, es posible identificar las propiedades comunes de ciertas relaciones y definir un **tipo de relación** y un correspondiente **conjunto de relaciones** bien definido como la colección de relaciones de dicho tipo. Las relaciones que satisfacen los requisitos de membresía en el conjunto de relaciones en cualquier momento son las **instancias**, o miembros, del conjunto de relaciones. Como con entidades y atributos, el tipo de relación es parte de la **intensión** y las instancias son parte de la **extensión** del modelo. Por ejemplo, se tiene un tipo de relación que vincula a cada estudiante con cada una de las clases en las que está inscrito. Entonces se tiene un conjunto de relaciones Enroll, que contiene todas las instancias de la relación. Si no se proporciona un nombre, la relación se referirá mediante los nombres de las entidades relacionadas, con un guión entre ellas, por ejemplo, Student-Class o Department-Faculty. La figura 3.1 muestra dos relaciones, Enroll y Teaches, representados mediante diamantes en el diagrama E-R.

3.5.1 Tipos de relaciones

Enroll es un ejemplo de un conjunto de relaciones **binario**, que vincula dos conjuntos de entidades. Es posible describir este conjunto de relaciones más formalmente como un conjunto de pares ordenados, en el que el primer elemento representa a un estudiante y el segundo una clase que el estudiante cursa. En un momento dado, las instancias de este conjunto de relaciones se puede considerar como

```
Enroll = { (El estudiante con Id S1001, La clase con classNumber ART103A),
           (El estudiante con Id S1020, La clase con classNumber CS201A),
           (El estudiante con Id S1002, La clase con classNumber CS201A),
           (El estudiante con Id S1010, La clase con classNumber ART103A),
           (El estudiante con Id S1002, La clase con classNumber ART103A),
           (El estudiante con Id S1020, La clase con classNumber MTH101B),
           (El estudiante con Id S1001, La clase con classNumber HST205A),
           (El estudiante con Id S1010, La clase con classNumber MTH103C),
           (El estudiante con Id S1002, La clase con classNumber MTH103C) }
```

Cada par ordenado muestra que un estudiante particular está inscrito en una clase particular. Por ejemplo, el primer par muestra que el estudiante cuyo ID es S1001 está inscrito en la clase con número de clase ART103A. El conjunto de los pares ordenados de las entidades

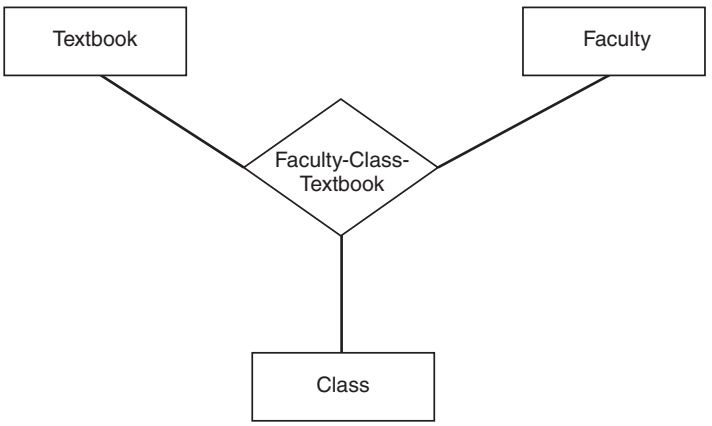


FIGURA 3.5
Una relación ternaria

relacionadas es el conjunto de relaciones, y cada par ordenado es una instancia de la relación.

Los conjuntos de entidades involucrados en un conjunto de relaciones no necesitan ser distintos. Por ejemplo, se podría definir una relación de compañero de cuarto (roommate) dentro del conjunto de entidades Student. Tal relación se llama **recursiva**. Si supone sólo dos estudiantes que comparten una habitación, ésta sería una relación binaria llamada Roommate de la forma

```
Roommate = {(Student1, Student2) | Student1 ∈ Student, Student2 ∈
Student, Student1 es el compañero de clase de Student2}.
```

Las instancias de esta relación serían pares ordenados de estudiantes, como en

```
Roommate = { (El estudiante con Id S1001, El estudiante con Id S1020),
             (El estudiante con Id S1020, El estudiante con Id S1001),
             (El estudiante con Id S1002, El estudiante con Id S1005),
             . . . }
```

Si se tiene un conjunto de entidades de empleados, podría tener una relación Manages (supervisión) recursiva que vincule a cada empleado con su supervisor, quien también es un empleado. Dicho conjunto se definiría como

```
Manages = {(Employee1, Employee2) | Employee1 ∈ Employee, Employee2 ∈ Employee,
Employee1 es el supervisor de Employee2}.
```

Las instancias serían pares ordenados de empleados, como en

```
Manages = { (El empleado con Id E101, El empleado con Id E301),
            (El empleado con Id E101, El empleado con Id E421),
            El empleado con Id E102, El empleado con Id E555),
            . . . }
```

Una relación puede involucrar más de dos conjuntos de entidades. Por ejemplo, podría tener una relación ternaria que involucre tres conjuntos de entidades y vincule clases, personal docente y libros de texto usados en la clase. Entonces el conjunto de relaciones se podría definir como un conjunto de tripletas ordenadas en las que el primer elemento representa una clase, el segundo un miembro del personal docente y el tercero un libro de texto. Al usar números de curso para representar las entidades class, ID de personal docente para representar entidades faculty y números isbn para representar libros de texto, esta relación, que se llamará Class-Faculty-Text, se puede considerar como

```
Class-Faculty-Text= { (Class ART103A, Faculty F101, Text '0-89134-573-6'),
                      (Class CSC201A, Faculty F105, Text '0-13-101634-1'),
```

```
(Class CSC203A, Faculty F105, Text '0-13-090955-5'),
(Class HST205A, Faculty F115, Text '0-78-531220-4'),
(Class MTH101B, Faculty F110, Text '0-618-04239-3'),
(Class MTH103C, Faculty F110, Text '0-618-733214-6')}
```

Aquí, cada tripleta ordenada es una instancia que muestra que un texto particular se utiliza en una clase particular que la imparte un profesor particular. Por ejemplo, la tripleta ordenada (Class ART103A, Faculty F101, Text '0-89134-573-6') significa que la clase ART103A, que la imparte el profesor cuyo ID es F101, usa el texto con isbn '0-89134-573-6' como libro de texto. La figura 3.5 ilustra cómo aparece una relación ternaria en un diagrama E-R.

Aunque la mayoría de las relaciones en un modelo de datos son binarias o cuando mucho ternarias, se podría definir un conjunto de relaciones que vincule cualquier número de conjuntos de entidad. Por tanto, el conjunto de relaciones general se puede considerar como un subconjunto de una relación *n*-aria de la forma

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

donde E_i son los conjuntos de entidades, e_i son las instancias de entidad y cada *n*-tupla ordenada representa una instancia de la relación. Se puede reconocer la notación utilizada para el producto cruz o producto cartesian o de conjuntos en matemáticas. De hecho, un conjunto de relaciones es un subconjunto del producto cartesiano de los conjuntos de entidades involucrados; esto es, si R es un conjunto de relaciones y E_1, E_2,…, E_n son conjuntos de entidades, entonces

$$R \subset E_1 \times E_2 \times \ldots \times E_n$$

3.5.2 Atributos de conjuntos de relaciones

A veces un conjunto de relaciones tiene **atributos descriptivos** que pertenecen a la relación en vez de alguna de las entidades involucradas. Por ejemplo, en la figura 3.1, el atributo grade es un atributo descriptivo para el conjunto de relaciones Enroll. En un diagrama E-R, se coloca un atributo descriptivo de una relación en un óvalo y se le conecta al diamante de relación. Dado que los estudiantes todavía están inscritos en las clases, se supondrá que este atributo representa la calificación de mitad de semestre. El atributo grade no describe la entidad Student, pues cada estudiante puede tener calificaciones para muchas clases, ni describe a la entidad Class, pues en una clase particular se otorgan diferentes calificaciones para distintos estudiantes. Para que una calificación tenga significado, debe asociarse con un estudiante particular para una clase particular. Dado que la calificación es un atributo de Enroll, se puede describir como un mapeo de instancias de Enroll al dominio de grade. La figura 3.6 muestra grade como función del mapeo de la instancia (Student S1001,Class ART103A) al dominio de grade. Note que la relación Enroll se dibuja como un conjunto de pares ordenados. Este diagrama no es parte del diagrama E-R, sino que se incluye sólo para ilustrar este concepto de relación.

3.5.3 Cardinalidad de una relación

Es importante identificar restricciones sobre las relaciones de modo que las posibles extensiones de la relación corresponden a conexiones o asociaciones del mundo real. Otros dos tipos de restricciones sobre las relaciones son las restricciones en la participación y la cardinalidad. La **cardinalidad** de una relación es el número de entidades a las que otra entidad puede mapear bajo dicha relación. Sean X y Y conjuntos de entidades y R una relación binaria de X a Y. Si no hubiera restricciones de cardinalidad sobre R, entonces cualquier número de entidades en X podría relacionarse con cualquier número de entidades en Y. Sin embar-

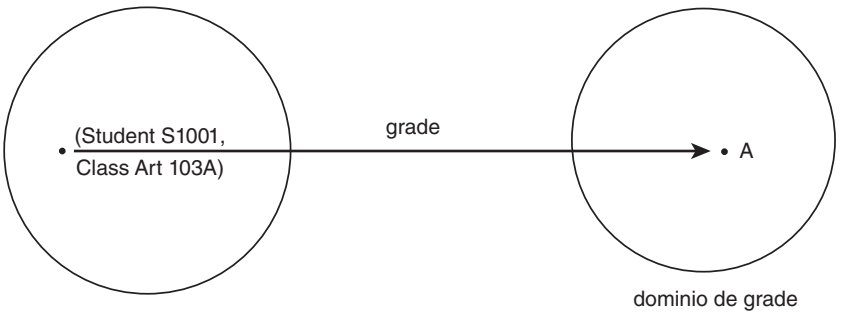


FIGURA 3.6
Atributo de conjunto de relaciones como función

go, por lo general, existen restricciones sobre el número de entidades correspondientes. Se distinguen cuatro tipos de relaciones binarias.

1. **uno a uno.** Una relación R de X a Y es **uno a uno** si cada entidad en X se asocia con cuando mucho una entidad en Y e, inversamente, cada entidad en Y se asocia con cuando mucho una entidad en X. Un ejemplo de relación uno a uno es la relación de Chairperson (jefe) a Department. Cada jefe dirige al menos un departamento, y cada departamento tiene cuando mucho un jefe. En la vida familiar, un ejemplo de relación uno a uno es la relación de matrimonio monógamo. En cualquier momento dado, cada esposo está casado con sólo una esposa, y cada esposa con un esposo.
2. **uno a muchos.** Una relación R de X a Y es **uno a muchos** si cada entidad en X se puede asociar con muchas entidades en Y, pero cada entidad en Y se asocia con cuando mucho una entidad en X. La palabra "muchos" se aplica al posible número de entidades con las que se asocia otra. Para una instancia dada, pueden existir cero, uno, dos o más entidades asociadas, pero si *alguna* vez es posible tener más de una, se usa la palabra "muchas" para describir la asociación. La relación entre Faculty y Class es uno a muchas, si supone que no hay docencia colectiva. Cada miembro del personal docente puede impartir muchas clases, pero cada clase se imparte por sólo un miembro del personal docente. En una familia, la tradicional relación madre a hijo es uno a muchas. Una madre puede tener muchos hijos, pero cada hijo sólo tiene una madre (de nacimiento).
3. **muchos a uno.** Una relación R de X a Y es **muchos a uno** si cada entidad en X se asocia con cuando mucho una entidad en Y, pero cada entidad en Y se puede asociar con muchas entidades en X. La relación entre Student y su Department de especialidad es muchos a uno, si supone que no hay dobles o triples especialidades. Cada estudiante puede tener cuando mucho una especialidad, pero un departamento puede tener muchas especialidades de estudiantes en ella. En la vida familiar, la relación hijo a madre (de nacimiento) es muchos a una. Cada hijo tiene cuando mucho una madre de nacimiento, pero cada madre puede tener muchos hijos. Una relación muchos a uno es en realidad lo mismo que uno a muchos, pero vista desde el lado opuesto.
4. **muchos a muchos.** Una relación R de X a Y es **muchos a muchos** si cada entidad en X se puede asociar con muchas entidades en Y y cada entidad en Y se puede asociar con muchas entidades en X. La relación entre Student y Class es muchos a muchos. Cada estudiante puede inscribirse en muchas clases (es decir, más de una) y cada clase puede tener muchos estudiantes inscritos. En una familia, la relación abuelo a nieto ilustra una relación muchos a muchos. Un abuelo puede tener muchos nietos, y un nieto puede tener muchos abuelos. La relación padre a hijo también es muchos a muchos, pues cada hijo puede tener dos padres, y cada padre muchos hijos.

Si los conjuntos de entidades A, B y C se relacionan en una relación ternaria R, se pueden determinar las restricciones de cardinalidad para la participación de cada conjunto de enti-

dades en la relación. Para cada combinación particular de las entidades B y C, si sólo existe un valor A, entonces A participa como "uno". Si puede haber más de un valor A para una combinación B-C particular, entonces A participa como "muchos". De igual modo, B participa como "uno" o "muchos" dependiendo de cuántos valores B puedan existir para cada combinación A-C, y C participa como "uno" o "muchos" dependiendo del número de valores C para cada combinación A-B. La noción se puede extender a cualquier grado de relación.

3.5.4 Muestra de cardinalidades en un diagrama E-R

En un diagrama E-R, las líneas que conectan los rectángulos representantes de los conjuntos de entidades a los diamantes representantes de los conjuntos de relaciones muestran sus asociaciones. Existen muchos métodos alternativos de mostrar la cardinalidad de la relación. La tradicional, que se muestra como Método 1 en la parte superior de la figura 3.7, es poner un "1" para mostrar una cardinalidad de relación "uno" y una "M" o "N" para mostrar una cardinalidad "muchos" en la línea que conecta una entidad con un conjunto de relaciones. Por ende, si un miembro docente se asocia con cada clase, la línea entre el rectángulo del conjunto de entidades Faculty y el diamante del conjunto de relaciones Faculty-Class tiene "1" sobre ella. Puesto que muchas clases se pueden asociar con el miembro del personal docente, la línea entre el diamante del conjunto de relaciones Faculty-Class y el rectángulo Class tiene "M" sobre ella. Si elige representar la relación uno a uno Chairperson-Department, pondría "1" en la línea que conecta el rectángulo Chairperson al diamante del conjunto de relaciones Chairperson-Department, y otro "1" sobre la línea que conecta el diamante del conjunto de relaciones al rectángulo Department. Para la relación Enroll, tendría una "M" desde el rectángulo del conjunto de entidades Student hasta el diamante Enroll y una "N" desde el diamante Enroll hasta el rectángulo del conjunto de entidades Class.

Una notación alternativa al "1" es poner una flecha sencilla sobre la línea desde un diamante de relación que apunte hacia un rectángulo de conjunto de entidades que participa como "uno" y una flecha doble sobre la línea que apunta hacia un conjunto de entidades que participa como "muchos". Algunos autores o herramientas de software para dibujar diagramas usan una flecha sencilla para "uno" y ninguna flecha para "muchos", mientras que otros usan "punto grande" al final de una línea "muchos" y nada en el extremo de una línea "uno". Incluso otra notación utiliza líneas para conectar entidades relacionadas, sin diamante, con una "pata de gallo" ramificada al final de "muchos", y un extremo no ramificado para "uno". La figura 3.7 resume estos métodos aplicados a estos tres ejemplos.

Para relaciones ternarias, se coloca "1" o "M" o "N" en el arco desde un conjunto de entidades hasta uno de relaciones, dependiendo de si la entidad participa como "uno" o como "muchos". La participación se determina al preguntar: "Para cada combinación de las otras dos entidades, ¿cuántas de éstas participan?" La figura 3.8 muestra las cardinalidades para la relación ternaria Faculty-Class-Textbook. Indica que, para cada combinación Faculty-Class, puede haber muchos textos, para cada combinación de docente y texto existen muchas clases, pero para cada combinación de clase y texto sólo existe un docente.

3.5.5 Restricciones de participación

Es posible que no todos los miembros de un conjunto de entidades participen en una relación. Por ejemplo, algunos miembros del personal docente pueden no impartir este semestre, o algunos estudiantes pueden no estar inscritos en alguna clase este semestre, aunque conserven su estatus de estudiante. Si todo miembro de un conjunto de entidades debe participar en una relación, a esto se le conoce como **participación total** del conjunto de enti-

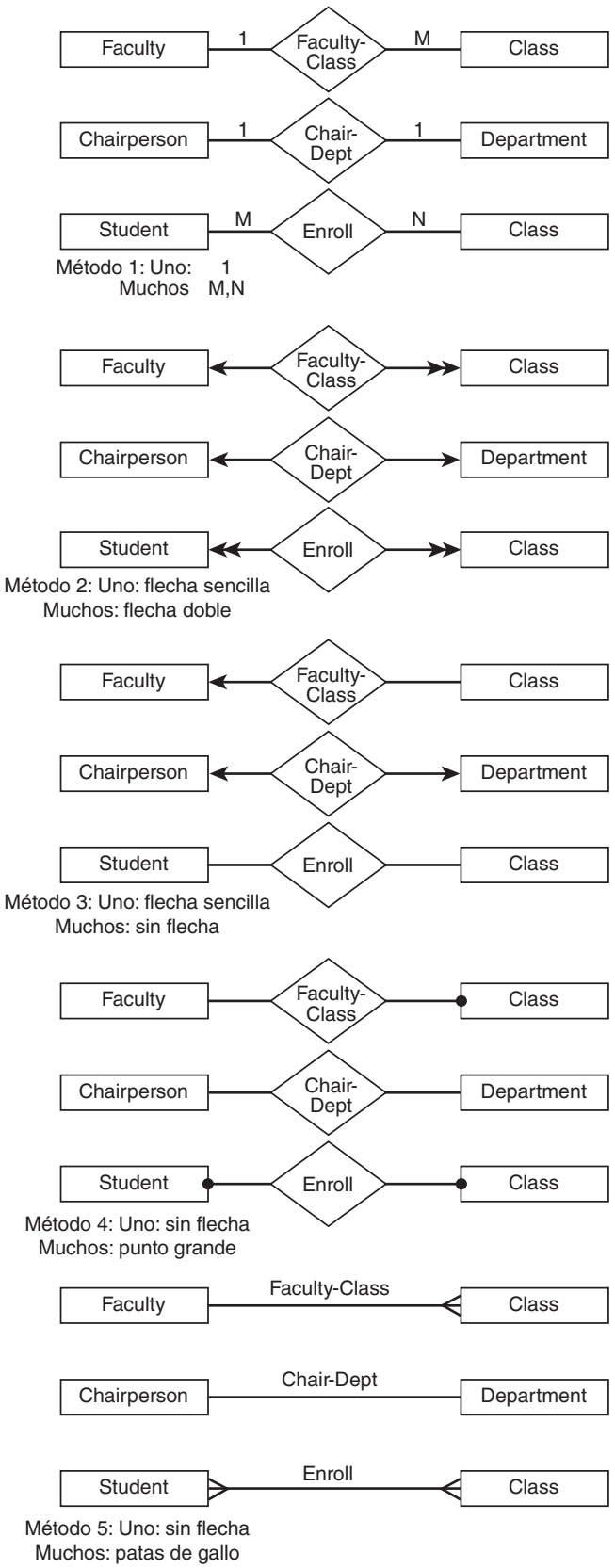


FIGURA 3.7
Muestra de cardinalidades en diagrama E-R

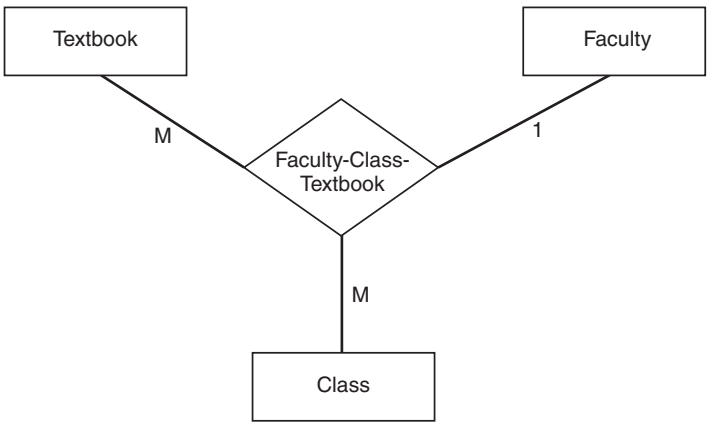


FIGURA 3.8
Cardinalidades para relación ternaria

dades en la relación. Esto se denota al dibujar una línea doble desde el rectángulo de entidades hasta el diamante de relación. Una línea sencilla indica que algunos miembros del conjunto de entidades no deben participar en la relación, una situación llamada **participación parcial**. Por ejemplo, si existen estudiantes que no tomen clases, la participación de Student en la relación Enroll es parcial, y la línea que liga el rectángulo del conjunto de entidades Student hasta el diamante de relaciones Enroll es una línea sencilla. Una línea doble implicaría que una persona no es estudiante a menos que esté inscrita en alguna clase. Las líneas sencillas de la figura 3.9 muestran que algunos estudiantes pueden no estar inscritos en alguna clase, algunas clases pueden no tener estudiantes inscritos y algunos miembros del personal docente pueden no impartir clase alguna. Sin embargo, la línea doble muestra que toda clase debe tener un miembro del personal docente para impartirla. También existen métodos alternativos para representar restricciones en la participación.

3.6 Roles

En una relación, cada entidad tiene una función llamada **rol** en la relación. Por lo general es claro a partir del contexto qué rol juega una entidad en una relación. Por tanto, en la relación que conecta Faculty y Class, se entiende que la entidad Faculty juega el rol de "es profesor de" en la relación, mientras que la entidad Class juega el rol "la imparte..." Sin embargo,

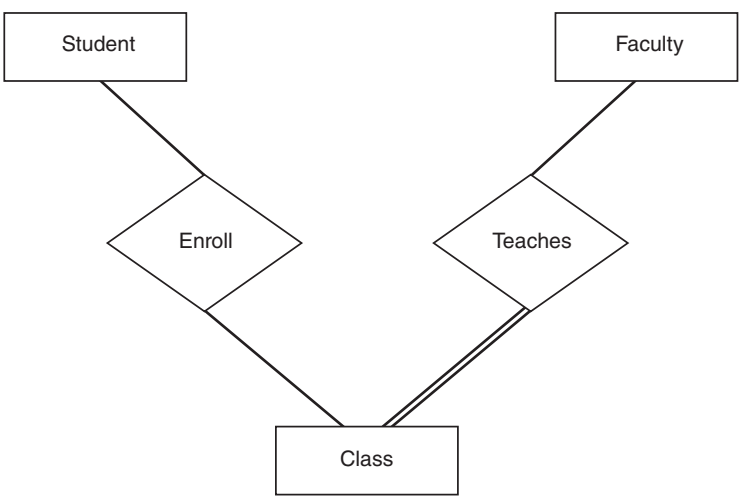


FIGURA 3.9
Participación total frente a parcial en una relación

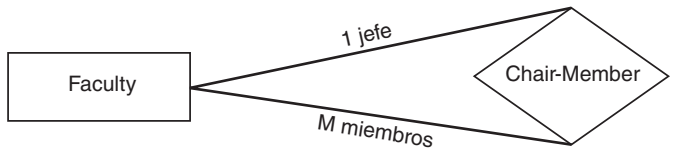


FIGURA 3.10 (a)
Relación recursiva

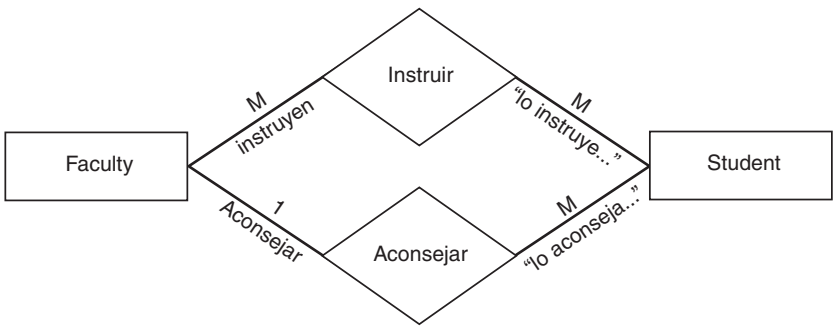


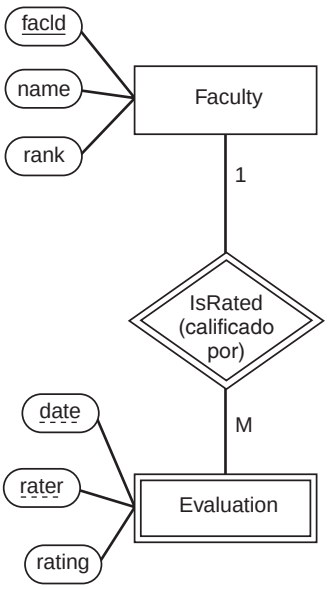
FIGURA 3.10 (b)
Conjuntos de entidades con dos relaciones

no todas las relaciones involucran conjuntos de entidades distintos. Cuando un conjunto de entidades se relaciona consigo mismo, se tiene una relación **recursiva**, y es necesario indicar los roles que juegan los miembros en la relación. Por ejemplo, un miembro del personal docente en cada departamento es el jefe. Si elige no representar al jefe como una entidad separada, sino conservarlo en el conjunto de entidades Faculty, entonces tendrá que representar la relación Chair-Member que vincula a otros miembros del departamento con su jefe. La figura 3.10(a) muestra el conjunto de relaciones Chair-Member definido en el conjunto de entidades Faculty, con los roles para el jefe y los otros miembros. La relación Roommate en el conjunto de entidades Student, que se describió en la sección 3.5.1, es una relación recursiva uno a uno, en la que un estudiante tiene el rol "tiene compañero de cuarto" y otro tiene el rol "es compañero de cuarto de". La relación Manages entre un empleado supervisor y los otros empleados a los que supervisa, que se muestra en la sección 3.5.1, es una relación recursiva uno a muchos. El supervisor juega el rol "supervisa" y los otros empleados juegan el papel "supervisado por". Otra instancia donde los nombres de rol son útiles ocurre cuando los mismos dos conjuntos de entidades se relacionan en dos formas diferentes. Por ejemplo, suponga que quiere representar dos tipos de relaciones diferentes entre los miembros del personal docente y los estudiantes. Un tipo sería la usual relación Profesor-Estudiante, en la que la entidad Faculty tiene el rol "instruye" y la entidad Student el rol "lo instruye..." Otra es la relación Advisor (consejero), en la que la entidad Faculty tiene el rol "aconseja" y la entidad Student tiene el rol "lo aconseja..." Cuando los conjuntos de entidades están conectados por múltiples relaciones, los roles se pueden escribir en las ligas apropiadas. La figura 3.10(b) muestra dos conjuntos de entidades, Faculty y Student, conectados mediante dos relaciones, con roles que aparecen en las ligas. Aunque no es necesario en todos los casos, se pueden usar nombres de roles para ayudar a clarificar cualquier relación, ya sea que se requiera o no.

3.7 Dependencia de existencia y entidades débiles

En ocasiones es necesario almacenar datos de una entidad en la que no estaría interesado a menos que ya tuviese una entidad relacionada en la base de datos. Por ejemplo, no necesitaría almacenar datos acerca de órdenes de ventas a menos que tuviese clientes. Entre dos entidades puede ocurrir una restricción de existencia, o **dependencia de existencia**. Si X y Y son entidades y cada instancia de Y debe tener una instancia correspondiente de X,

FIGURA 3.11
Conjunto de entidades débiles



entonces se dice que Y es **dependiente de la existencia** de X. Esto significa que una entidad Y no puede existir sin alguna entidad X. Si Y depende de la existencia de X, entonces Y debe tener **participación total** en su conjunto de relaciones con X. Un tipo especial de dependencia de existencia ocurre cuando el conjunto de entidades dependientes no tiene una clave candidata y sus instancias son indistinguibles sin una relación con otra entidad. Por ejemplo, suponga que los docentes más antiguos o el decano realizan evaluaciones de enseñanza a los miembros del personal docente, y que las calificaciones (ratings) se almacenan en la base de datos. Por simplicidad, suponga que por cada evaluación se otorga una calificación, de modo que una entidad Evaluation (evaluación) tiene atributos Date (fecha), RaterName (nombre del calificador) y Rating (calificación). Dado que puede haber muchas instancias con valores idénticos para los tres atributos, una entidad Evaluation debe asociarse con la instancia Faculty correcta para que tenga significado. En este caso, a X (Faculty, en este ejemplo) se le conoce como la **entidad fuerte** (también llamada padre, propietario o entidad dominante) y Y (Rating, en este ejemplo) como la **entidad débil** (también llamada hijo, dependiente o entidad subordinada). Una entidad débil se muestra en el diagrama E-R al dibujar un rectángulo doble alrededor de la entidad, y el diamante de relación se convierte en diamante doble. La figura 3.11 muestra el diagrama E-R con la entidad Evaluation débil y su relación identificatoria. Cuando una entidad es débil, no tiene clave primaria propia, sino que es única sólo en referencia a su relación con su propietaria. Sin embargo, con frecuencia tiene una clave parcial, también llamada discriminador, que permite identificar de manera única las entidades débiles que pertenecen al mismo propietario. La clave parcial puede ser un atributo solo o uno compuesto. Para Evaluation, Date solo no puede ser suficiente, si es posible que un miembro del personal docente sea evaluado por dos personas diferentes en la misma fecha. Por tanto, se usará la combinación Date y RaterName como el discriminador. El discriminador se indica mediante una subraya punteada en el diagrama E-R.

3.8 Diagrama E-R de muestra

La figura 3.12 muestra un diagrama E-R que incorpora muchas de las entidades y relaciones discutidas en este capítulo. Las entidades y atributos son:

- Student: stuId, lastName, firstName, major, credits

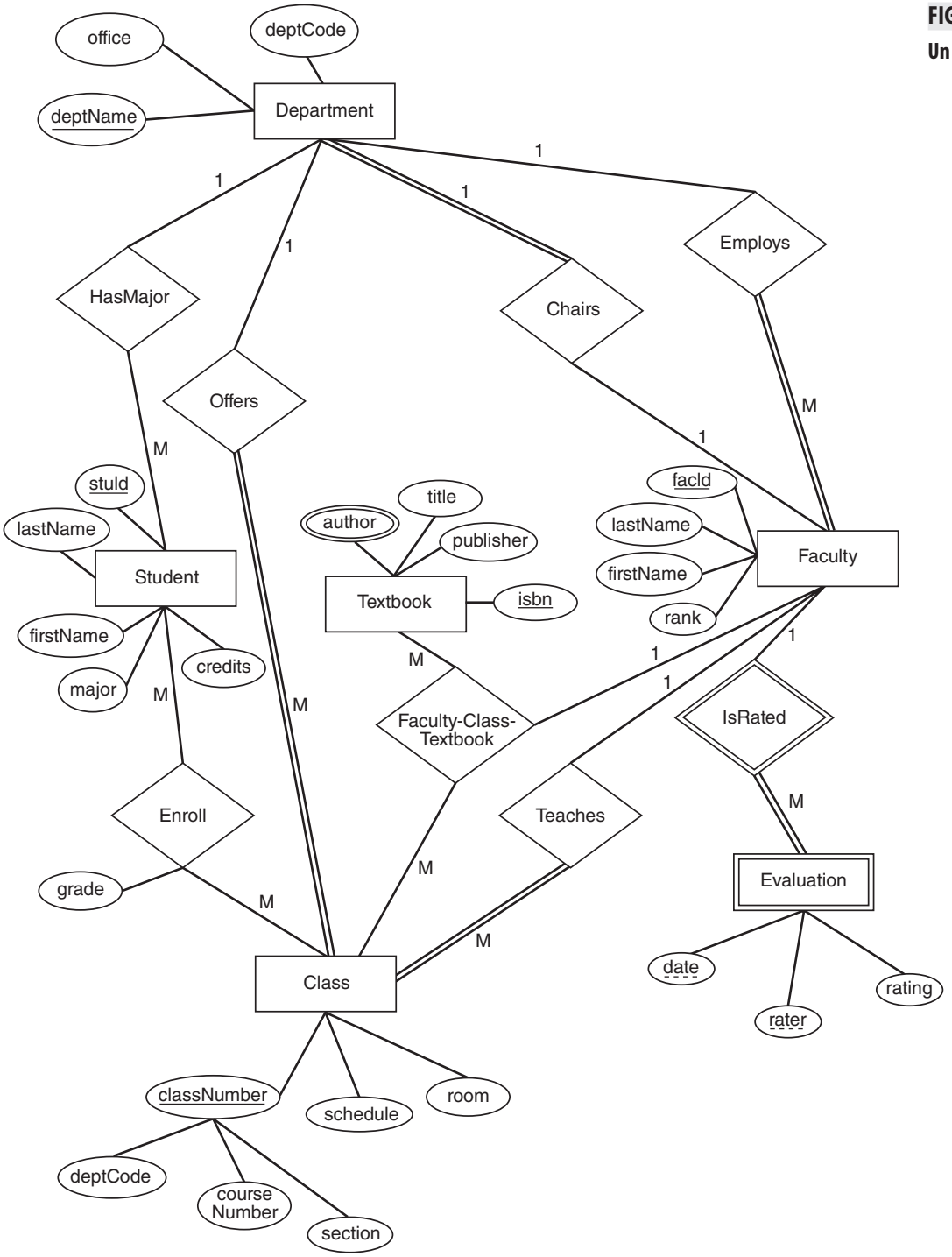


FIGURA 3.12
Un diagrama E-R

Se supone que cada estudiante tiene una ID única y tiene cuando mucho una especialidad.

- Department: deptCode, deptName, office

Se supone que cada departamento tiene un código único y un nombre único, y que cada departamento tiene una oficina designada como oficina departamental.

- Faculty: facId, lastName, firstName, rank

Se supone que FacId es única y que todo miembro del personal docente debe pertenecer a un departamento. Un miembro del personal docente en cada departamento es el jefe.

- Class: classNumber, sched, room

Se supone que classNumber consiste en un código de departamento, un número que identifica el curso dentro del departamento y un código de sección. Sólo se conservan datos para las clases actuales. Si se quieren almacenar datos históricos acerca de clases anteriores que se ofrecieron o datos acerca de clases contempladas, se necesitaría agregar un atributo de fecha.

- Textbook: isbn, author, title, publisher

Se supone que un libro puede tener varios autores.

- Evaluation: date, rater, rating

La evaluación es una entidad débil, dependiente de Faculty.

Note que la empresa no se hace una de las entidades. No se tiene un conjunto de entidades llamado universidad, pues ésta es toda la empresa. Todo en el diagrama representa alguna faceta de la universidad. Los conjuntos de relaciones son:

1. HasMajor (tiene especialidad), que es una relación uno a muchos que conecta a los estudiantes con sus departamentos de especialidad. Se supone que los estudiantes tienen cuando mucho una especialidad. No todo departamento tiene especialidades, y no todo estudiante tiene una especialidad declarada.
2. Offers (ofrece), que es una relación uno a muchos que conecta departamentos a las clases que ofrece. En esta base de datos sólo se conservan los ofrecimientos del semestre actual. Un departamento puede no tener clases ofrecidas este semestre, pero toda clase tiene un departamento que la ofrece.
3. Enroll (inscripción), que es una relación muchos a muchos que conecta estudiantes con las clases en las que están inscritos. Se supone que sólo se conservan las inscripciones actuales. Grade (calificación) es un atributo descriptivo para este conjunto de relaciones. Dado que los estudiantes todavía están inscritos en las clases, este atributo representa una calificación de mitad de semestre. Algunos estudiantes no están inscritos en clase alguna, y algunas clases ofrecidas pueden no tener estudiantes inscritos.
4. Employs (emplea), que es una relación uno a muchos que conecta departamentos a miembros del personal docente asignados a ellos. Los miembros del personal docente deben pertenecer exactamente a un departamento. Un departamento puede o no tener docentes asignados, pero también puede tener muchos docentes.
5. Chairs (cátedras), que es una relación uno a uno que conecta departamentos con docentes. Un docente en cada departamento es el jefe del departamento. Todo departamento debe tener un jefe, pero no todo miembro del personal docente debe ser jefe. Se supone que un jefe es miembro regular del personal docente, que tiene la responsabilidad adicional de dirigir el departamento.
6. Teaches (imparte), que es una relación uno a muchos que conecta miembros del personal docente con las clases que imparten. Se supone que un docente puede impartir cero o muchas clases, y cada clase tiene exactamente un docente designado.
7. IsRated (calificado por), es una relación uno a muchos entre Faculty y la entidad débil, Evaluation. No todo docente tiene una calificación, pero toda calificación debe pertenecer a un miembro del personal docente.
8. Faculty-Class-Textbook, que es una relación ternaria entre las tres entidades. Podría usar dos relaciones binarias, la Teaches que ya existe y conectar Faculty y Class, y una

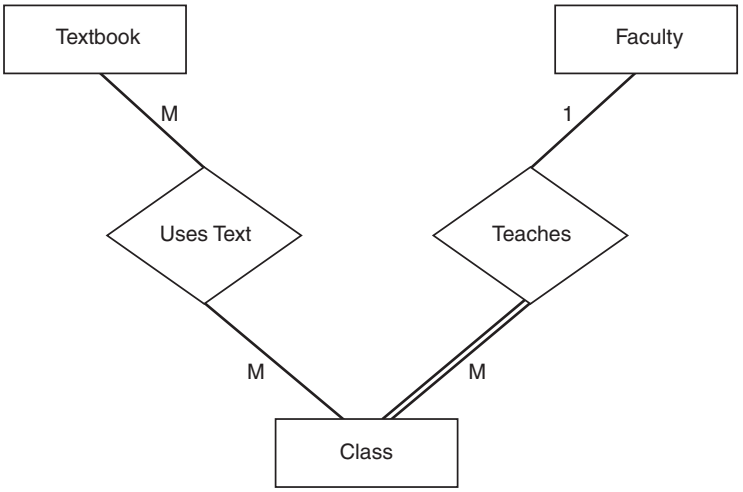


FIGURA 3.13
Ejemplo de dos relaciones binarias que sustituyen la relación ternaria

nueva UsesText, que conecta Class y Textbook, como se indica en la figura 3.13. Sin embargo, el significado sería ligeramente diferente. La relación binaria UsesText indicaría que las clases usarían dicho texto sin importar quién sea el instructor, mientras que la relación ternaria indica que el texto depende del instructor. Las cardinalidades son más difíciles de identificar en una relación ternaria que en una binaria, pero las que se muestran en la figura 3.12 conservan los significados. Para identificar la cardinalidad para cada entidad, es necesario determinar cuántas instancias de dicha entidad se relacionan con cada combinación de instancias de las otras dos entidades.

3.9 Resumen del capítulo

El modelo **entidad-relación** usa **diagramas E-R** para representar un esquema de empresa, una descripción a nivel conceptual que es independiente de cualquier DBMS. Una **entidad** es cualquier objeto distinguible en el minimundo que modela la base de datos. Las entidades se categorizan en **tipos de entidad**, y una colección de entidades del mismo tipo forma un **conjunto de entidades**. Las entidades individuales que pertenecen al conjunto en un momento dado son **instancias de entidad**. En un diagrama E-R, un **rectángulo** representa un tipo de entidad. Los **atributos** son representaciones de propiedades de las entidades del mundo real. Se representan como **óvalos** en un diagrama E-R. El conjunto de valores permitidos para un atributo son su dominio. El atributo es en realidad un mapeo del conjunto de entidades en el dominio del atributo. Los valores **nulos** ocurren cuando a una instancia de entidad le falta un valor para un atributo particular. Los atributos también pueden ser **multivaluados**, **compuestos** y/o **derivados**.

Una **superclave** es un conjunto de atributos que identifica de manera única instancias de entidad. Una superclave mínima, aquella sin un subconjunto propio que también es una superclave, se llama **clave candidata**. La **clave primaria** de una entidad es la clave candidata que el diseñador elige para identificación única. Las otras claves candidatas se pueden convertir en **claves alternativas**. Una **clave secundaria**, que proporciona otra forma de acceder a los registros, puede o no tener valores únicos. Una **clave compuesta** es aquella que tiene más de un atributo. Ningún atributo de una clave primaria puede tener valores nulos.

Una **relación** es una asociación o interacción entre entidades. Un **conjunto de relaciones** consiste en todas las relaciones de un tipo de relación dado. Las relaciones pueden ser **binarias,** que ligan dos entidades, **ternarias**, que ligan tres entidades, o ***n*-arias**, que ligan *n* enti-

dades. Las instancias de relación binaria se pueden representar como pares ordenados, las instancias ternarias como tripletas ordenadas y las instancias *n*-arias como *n*-tuplas ordenadas de instancias de entidad. Un conjunto de relaciones es un subconjunto del producto cartesiano de los conjuntos de entidad relacionados. Un **diamante** se usa para representar un conjunto de relaciones en un diagrama E-R. Un conjunto de relaciones puede tener atributos descriptivos. En un diagrama E-R, un atributo descriptivo aparece en un óvalo conectado con el diamante de relación. Las relaciones tienen **restricciones de cardinalidad**, que especifican cuántas instancias de entidad se pueden relacionar. Para relaciones binarias, éstas pueden ser uno a uno, uno a muchos, muchos a uno o muchos a muchos. Para relaciones ternarias o de orden superior, la cardinalidad está determinada por cada entidad al examinar cuántas instancias de dicha entidad pueden ocurrir para cada combinación de las otras entidades en la relación. Las cardinalidades se pueden mostrar en los diagramas E-R en varias formas, la usual es escribir "1" o "M" en el arco desde cada conjunto de entidades hacia el diamante de relación para indicar la cardinalidad de la participación de dicho conjunto de entidades. Las relaciones también pueden tener restricciones de participación, que pueden ser **total**, lo que indica que todos los miembros del conjunto de entidades deben participar en la relación, o **parcial**, si no todos los miembros tienen que participar.

Si una relación es **recursiva**, es decir, está definida en un solo conjunto de entidades, o si dos conjuntos de entidades se relacionan en más de una forma, se puede identificar el **rol** o función que juega una entidad en una relación. Esto se hace al colocar el nombre del rol en el arco desde el conjunto de entidades hasta el diamante de relaciones en el diagrama E-R.

Una entidad es **dependiente de la existencia** de otra si no puede existir en la base de datos sin una instancia correspondiente de la otra entidad. Si tal entidad no tiene claves propias, sino que debe usar el atributo de clave primaria de la entidad de la que depende, se llama **débil**. La entidad de la que depende se llama **fuerte**. Una entidad débil se muestra en un diagrama E-R dentro de un rectángulo doble con su relación de identificación mostrada como diamante doble.

Ejercicios

3.1 Defina cada uno de los siguientes términos.

a. tipo de entidad
b. conjunto de entidades
c. conjunto bien definido
d. intensión de una entidad
e. extensión de una entidad
f. atributo
g. dominio de un atributo
h. valor nulo
i. superclave
j. clave candidata
k. clave compuesta
l. clave primaria
m. clave alternativa
n. clave secundaria
o. tipo de relación

p. conjunto de relaciones
q. relación binaria
r. relación ternaria
s. relación *n*-aria
t. cardinalidad de una relación
u. relación recursiva
v. dependencia de existencia
w. entidad débil

3.2 Considere el conjunto de entidades Empleado con atributos empId, socSecNo, empNombre, titulopuesto y salario

a. Muestre cómo el conjunto de entidades y sus atributos se representarían en un diagrama E-R.
b. Describa el dominio del atributo salario y haga las suposiciones necesarias.
c. Identifique una superclave para el conjunto de entidades Empleado.
d. Identifique todas las claves candidatas para el conjunto de entidades.
e. Identifique una clave primaria para el conjunto de entidades y subráyelo en el diagrama E-R.

3.3 a. Suponga en la misma empresa del ejercicio 3.2, que existe un conjunto de entidades llamado Proyecto con atributos proyNombre, fechaInicio, fechaFin y presup. Muestre cómo se representarían este conjunto de entidades y su relación con Empleado en el diagrama E-R. Suponga que quiere representar el número de horas que se asignan a un empleado para trabajar en un proyecto y muéstrelo en el diagrama.

b. Al hacer las suposiciones necesarias, tome una decisión acerca de la cardinalidad y las restricciones de participación de la relación y agregue los símbolos adecuados al diagrama E-R.
c. Suponga que debe agregar otra entidad llamada Departamento. Cada empleado trabaja sólo para un departamento. Los proyectos no los patrocina directamente un departamento. Elabore los atributos necesarios y agregue esta entidad y las relaciones adecuadas al diagrama.

3.4 Diseñe una base de datos para conservar los datos de estudiantes universitarios, sus consejeros académicos, los clubes a los que pertenecen, los moderadores de los clubes y las actividades que patrocina cada club. Suponga que a cada estudiante se le asigna un consejero académico, pero un consejero aconseja a muchos estudiantes. Los consejeros no tienen que ser miembros del personal docente. Cada estudiante puede pertenecer a cualquier número de clubes, y los clubes pueden patrocinar cualquier número de actividades. El club debe tener algunos miembros estudiantes con la finalidad de existir. Cada actividad está patrocinada por exactamente un club, pero puede haber muchas actividades programadas para un día. Cada club tiene un moderador, quien puede o no ser miembro del personal docente. Dibuje un diagrama E-R completo para este ejemplo. Incluya todas las restricciones.

3.5 El consultorio de un dentista necesita conservar información acerca de pacientes, el número de visitas que hacen al consultorio, el trabajo que se debe realizar, los procedimientos realizados durante las visitas, los cargos y pagos por el tratamiento y los suministros de laboratorio y servicios. Suponga que sólo hay un dentista, de modo que no hay necesidad de almacenar información acerca del dentista en la base de

datos. Existen muchos cientos de pacientes. Los pacientes pueden hacer muchas visitas y la base de datos debe almacenar información acerca de los servicios realizados durante cada visita, y los cargos por cada uno de los servicios. Existe una lista estándar de cargos, que se mantiene fuera de la base de datos. El consultorio usa tres laboratorios dentales que proporcionan suministros y servicios, como fabricar dentaduras. Dibuje un diagrama E-R completo para este ejemplo.

3.6 Una firma de diseño de interiores quiere tener una base de datos para representar sus operaciones. Un cliente solicita que la firma realice un trabajo como decorar una casa nueva, redecorar habitaciones, encontrar y comprar mobiliario, y cosas por el estilo. Uno de los decoradores de la firma está a cargo de cada trabajo. Para cada trabajo, la firma proporciona una estimación de la cantidad de tiempo y dinero requeridos para todo el trabajo. Parte de las actividades de un trabajo, como planear la colocación de los muebles, la realiza el decorador encargado del trabajo. Además, la firma puede contratar contratistas para laborar por día u hora en un trabajo particular. Un trabajo también puede incluir muchas actividades, como pintar, instalar pisos, fabricar cortinajes, papel tapiz, construir, instalar gabinetes, etc. Estas actividades las realizan contratistas contratados por la firma. El contratista proporciona una estimación para cada actividad. Una actividad o trabajo también pueden requerir materiales como pintura o madera, y la firma tiene que dar seguimiento al costo de los materiales para cada actividad o trabajo, con la finalidad de cobrar al cliente. La base de datos debe almacenar los costos estimados y los costos reales de todas las actividades y todos los trabajos. Dibuje un diagrama E-R completo para este ejemplo.

3.7 Un taller de hojalatería automotriz necesita conservar información acerca de sus operaciones. Los clientes inicialmente llevan sus vehículos al taller para un presupuesto de las reparaciones. Un mecánico observa el automóvil y estima el costo y tiempo requeridos para todo el trabajo. Si el cliente acepta la estimación, se le asigna un número de trabajo y se registran el nombre e información de contacto del cliente; el número de placas, marca, modelo y año del automóvil; y una lista de las reparaciones necesarias. Luego el cliente hace una cita para llevar el auto en una fecha específica. Cuando el auto se lleva para reparaciones, comienza el trabajo. El taller da seguimiento a los cargos para partes y mano de obra conforme se acumulan. Sólo un mecánico labora en el vehículo durante todo el trabajo. Un trabajo puede incluir varias reparaciones (por ejemplo, cambiar el guardafangos izquierdo, pintar la puerta del pasajero). El tiempo que realmente se emplea en cada reparación se registra y usa para calcular el costo de la mano de obra, mediante una tarifa horaria fija. Dibuje un diagrama E-R completo para este ejemplo.

3.8 Se necesita una base de datos para seguir las operaciones de un centro de terapia física. A cada paciente lo remite un médico y tiene una receta para terapia física con la finalidad de recibir tratamiento. Un paciente puede tener distintos médicos en diferentes momentos. La base de datos conserva toda la información acerca de recetas y tratamientos, tanto pasadas como actuales. Cuando se hacen las citas, se registra la información acerca de la fecha y hora programadas. Ningún paciente se programa para dos visitas en un día. El centro tiene muchos terapeutas físicos y un paciente puede recibir tratamiento de diferentes terapeutas físicos en distintas visitas. Cuando un paciente hace una visita en un horario programado, se registran el nombre del terapeuta, el tratamiento, la fecha, la hora y el equipo utilizados para dicha visita. Cada uno de éstos tiene sólo un valor para la visita. Esta información se usará más tarde para el cobro del seguro, que no es parte de esta base de datos. Dibuje un diagrama E-R completo para este ejemplo.

Ejercicios de laboratorio: Dibujo de diagramas E-R

Para este ejercicio de laboratorio debe usar software de herramientas de dibujo como SmartDraw, Microsoft Visio o productos similares. Si no los tiene disponibles, puede usar el programa Paint o la herramienta de dibujo de Microsoft Word.

Con la herramienta de dibujo, dibuje un diagrama E-R para el ejemplo Empleado-Proyecto-Departamento descrito en los ejercicios 3.2 y 3.3.

PROYECTO DE MUESTRA: CREACIÓN DEL DIAGRAMA E-R PARA EL PROYECTO GALERÍA DE ARTE

- Paso 3.1. Elabore una lista de todas las entidades y sus atributos asociados.

Esto puede requerir varios intentos y diferentes diseñadores llegarán a distintas soluciones. Para identificar las entidades, examine el diccionario de datos y la tabla de referencias cruzadas que desarrolló en pasos previos. La mayoría de los ítems de datos representan atributos en vez de entidades. Debe evitar la tentación de hacer todas las entidades de reportes o transacciones. Su trabajo es intentar usar abstracción con el fin de agrupar los atributos en entidades para el minimundo que modela, que cubre sólo una pequeña parte de las actividades de la galería. Además de examinar los documentos desarrollados en el capítulo 2, necesita pensar en la empresa y preguntarse cuáles son las personas, lugares, eventos, objetos o conceptos en el minimundo de los que quiere conservar información. La tabla de referencias cruzadas puede ayudarlo. Si muchos atributos tienden a aparecer juntos en los reportes, pueden ser atributos de la misma entidad. El diccionario de datos original puede tener algunos ítems que no necesita almacenar en la base de datos. Se pueden eliminar de la lista de atributos. Al examinar el diccionario de datos y preguntarse cuáles personas son importantes en la Galería de Arte, puede identificar con certeza artistas, coleccionistas y compradores como entidades, y probablemente aquellos clientes potenciales que llenaron las formas y quiera colocar en la lista de correos, junto con los compradores. Al pensar en cuáles eventos son importantes, la venta de una obra de arte es un evento central, y una exposición es un evento de cierta importancia. El vendedor que vende una obra de arte también podría ser una entidad. La obra de arte es un objeto de gran importancia para la galería, de modo que podría ser una entidad. Note que la galería en sí no es una entidad, pues es la empresa.

Las entidades son:

```
Artist
Artwork
Buyer
Collector
Potential customer
Sale
Show
Salesperson
Owner
```

1. Artist (artista)
   Al identificar los atributos para una entidad, se intenta encontrar los ítems de datos que dicen un solo hecho acerca de una instancia de entidad. Para la entidad Artist, se buscan los atributos cuyo valor dirían un trozo de información acerca de un artista particular. Al agrupar los ítems del diccionario de datos, los atributos que parecen

describir al artista (en oposición a sus obras de arte o la venta o exposición de las mismas), son:

`artistAddress, artistAreaCode, artistCity, artistFirstName, artistInterviewDate, artistInterviewerName, artistLastName, artistPhone, artistSalesLastYear, artistSalesYearToDate, artistSocialSecurityNumber, artistState, artistStreet, artistTelephoneNumber, artistTotalSalesforPeriod, artistTotalAskingPriceforPeriod, usualMedium, usualStyle, usualType.`

Al examinar esto más de cerca, se ve que algunos de ellos forman atributos compuestos, incluidos:

`artistAddress`, que consiste de `artistStreet, artistCity, artistState, artistZip`
`artistName`, que consiste de `artistFirstName, artistLastName`
`artistPhone`, que consiste de `artistAreaCode, artistTelephoneNumber`

Note que algunos de los ítems son producto de un solo reporte y no tienen significado si no se conocen los parámetros para el reporte, de modo que se deben considerar datos temporales (efímeros) que no se almacenarán. Como datos efímeros se incluyen artistTotalSalesforPeriod y artistTotalAskingPriceForPeriod, pues son dependientes de las fechas de inicio y fin que el usuario elige para correr el reporte, y tienen poco significado sin el reporte. Puede considerar tratar artistSalesLastYear y artistSalesYearToDate de la misma manera. Sin embargo, el valor de artistSalesLastYear podría calcularse al final de cada año, como se requiere para propósitos de declarar impuestos, y es una constante para todo el año que sigue, de modo que es un valor que podría calcular una vez y almacenar. De igual modo, artistSalesYearToDate se podría calcular una vez cada semana o una vez cada mes y almacenarse. Estos dos ítems tienen significado independiente del reporte que los produce siempre que, desde luego, se actualicen regularmente. Con estos cambios, la lista de atributos se recorta. Al eliminar el prefijo de artista que se mencionó en el diccionario de datos para algunos de los atributos, ahora se tiene:

`Artist` con atributos: `address(street, city, state, zip), interviewDate, interviewerName, name(first, last), phone(areaCode, number), salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType`

Normalmente se elegiría socialSecurityNumber como la clave primaria, pues tiene valores únicos. Si la galería vende sólo obras propiedad del artista que las crea, esto podría funcionar bien, pues el Registro Federal de Contribuyentes (R.F.C.) tendría que proporcionarlo el artista para declarar sus impuestos, de modo que siempre estaría disponible. Sin embargo, no se puede asegurar que se tendrán los R.F.C. de los artistas cuyas obras sean propiedad de coleccionistas. También es posible que se tendrán obras de artistas extranjeros, quienes no tienen R.F.C. Por tanto, se pueden tener valores nulos para este atributo, de modo que no se les puede usar como la clave primaria. En vez de ello, se elige el nombre del artista, que se supone es único y siempre debería estar disponible.

2. Artwork (obra de arte)

   Cuando examina el diccionario de datos para ítems que describen la obra de arte, encuentra que los candidatos para atributos son:

   `askingPrice, dateListed, dateReturned, dateShown, status, medium, size, style, title, type, yearCompleted`

   En la lista no se incluyen los atributos que describen al artista, que es una entidad diferente. También se ignoran los atributos que describen al propietario de la obra, pues dicha persona será o el artista o un coleccionista, que será una entidad separada. Esto deja sólo algunos atributos para la obra.

Artwork con atributos: askingPrice, dateListed, dateReturned, dateShown, status, medium, size, style, title, type, yearCompleted

Note que Artwork no tiene una clave. Si se conoce al artista de la obra, entonces title sería la clave. Dado que requiere el uso de la clave de otra entidad, Artwork es una entidad débil, dependiente de Artist. Use title como una clave parcial, un discriminador para esta entidad.

3. Buyer (comprador)

   Los atributos potenciales son:

   buyerAddress, buyerAreaCode, buyerCity, buyerFirstName, buyerLastName, buyerState, buyerStreet, buyerTelephoneNumber, buyerZip, purchasesLastYear, purchasesYearToDate

   Al agrupar los compuestos, como antes, y eliminar el prefijo buyer, se obtiene lo siguiente:

   Buyer con atributos: name(firstName, lastName), address(street, city, state, zip), phone(areaCode, telephoneNumber), purchasesLastYear, purchasesYearToDate

   No se tiene número de R.F.C., que proporcionaría valores únicos para una clave, y no se quiere pedir a los compradores que proporcionen sus R.F.C. por razones de privacidad. Se supone que los nombres de los compradores no son únicos, de modo que no se usan nombres como clave. Phone aparece como una posibilidad para una clave, pero sólo si dos compradores no tienen el mismo número telefónico. Para permitir la posibilidad de que, por ejemplo, dos miembros del mismo hogar puedan ser compradores, se agregará el nombre al número telefónico y se usará la combinación como la clave primaria compuesta.

4. Collector (coleccionista)

   Al buscar en el diccionario de datos atributos de Collector, los candidatos incluyen:

   collectionArtistFirstName, collectionArtistLastName, collectionMedium, collectionStyle, collectionType, collectorAddress, collectorAreaCode, collectorCity, collectorFirstName, collectorInterviewDate, collectorInterviewerName, collectorLastName, collectorPhone, collectorSalesLastYear, collectorSalesYearToDate, collectorSocialSecurityNumber, collectorState, collectorStreet, collectorTelephoneNumber, collectorTotalSalesforPeriod, collectorTotalAskingPriceforPeriod

   Algunos de éstos forman atributos compuestos, incluidos:

   address, que consiste de street, city, state, y zip
   name, que consiste de firstName y lastName
   phone, que consiste de areaCode y telephoneNumber

   Se eliminan datos efímeros, incluidos collectorTotalSalesforPeriod y collectorTotalAskingPriceforPeriod, pero se conservan collectorSalesLastYear y collectorSalesYearToDate, que tienen significado independiente de los reportes que los generan.

   De nuevo se elimina el prefijo, lo que deja lo siguiente:

   Collector con atributos: name(firstName, lastName), address(street, city, state, zip), interviewDate, interviewerName, phone(areaCode, telephonenumber), salesLastYear, salesYearToDate, collectionArtistFirstName, collectionArtistLastName, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate, SocialSecurityNumber

Note que, a diferencia de los compradores, los coleccionistas deben proporcionar su R.F.C., pues la galería está obligada a reportar al gobierno el R.F.C. del recibo y la cantidad de cualquier pago por las obras vendidas. El R.F.C. se usará como la clave primaria.

5. Potential customer (cliente potencial)
   Recuerde que un cliente potencial es una persona que llenó una forma que indica interés en las obras de la galería, pero que todavía no compra alguna obra de arte. Los atributos a considerar son los siguientes:

   potentialCustomerAddress, potentialCustomerAreaCode, potentialCustomerCity, potentialCustomerDateFilledIn, potentialCustomerFirstName, potentialCustomerLastName, potentialCustomerState, potentialCustomerStreet, potentialCustomerTelephoneNumber, potentialCustomerPreferredArtist, potentialCustomerPreferredMedium, potentialCustomerPreferredStyle, potentialCustomerPreferredType, potentialCustomerZip

   Al agrupar los atributos compuestos y eliminar el prefijo, se tiene lo siguiente:

   Potential customer con atributos: address(street, city, state, zip), phone(areaCode, telephoneNumber), name(firstName, lastName),dateFilledIn, preferredArtist, preferredMedium, preferredStyle, preferredType.

   Por las mismas razones que con comprador, se agregará el nombre al número telefónico y la combinación se usará como la clave primaria compuesta.

6. Show (exposición)
   Los atributos potenciales son: `showFeaturedArtist, showClosingDate, showTheme, showTitle, showOpeningDate`

   Note que una exposición siempre tiene un título único y puede tener o la presentación de un artista o un tema. Ninguno de los atributos es compuesto o efímero, de modo que se les conserva todos. Dado que el título es único, se le usa como la clave. Se tiene: `Show` con atributos `FeaturedArtist, ClosingDate, Theme, Title, OpeningDate`

7. Sale (venta)

   Los atributos potenciales son: `amountRemittedtoOwner saleDate, saleInvoiceNumber, salePrice, saleSalesPersonCommission saleTax, SaleTotal`

   Podría considerar el título de la obra de arte, el nombre del comprador, el nombre del artista, el nombre del coleccionista (si hay alguno) y el nombre del vendedor, pero note que existen entidades separadas para artwork, buyer, artist, collector y salesperson, de modo que aquí no se incluyen estos atributos. Se conservará la comisión del vendedor, pues dicho atributo describe una sola venta y tendrá diferentes cantidades para distintas ventas para el mismo vendedor.

   Puesto que ninguno de los atributos mencionados son datos compuestos o efímeros, se dejan todos, de modo que se tiene: `Sale` con atributos `amountRemittedToOwner, saleDate, InvoiceNumber, salePrice, saleSalesPersonCommission, saleTax, SaleTotal`. Se usará el número de factura como la clave primaria.

8. Salesperson (vendedor)
   Al elegir atributos para Salesperson no se incluirá información acerca de las ventas individuales que hizo un vendedor, pues los datos de venta ya se mencionan para dicha entidad. Salesperson tiene entonces atributos potenciales como:
   salesPersonAddress, salesPersonFirstName, salesPersonLastName, salesPersonName, salespersonSocialSecurityNumber, salespersonCommissionForPeriod, salespersonTotalSalesForPeriod

Puede especificar los componentes individuales de la dirección y el número telefónico, y hacer un atributo compuesto para el nombre. Se eliminan las ventas y el total de comisiones para un periodo, pues éstos son datos efímeros. Esto deja la entidad como: `Salesperson` con atributos `name(firstName, lastName)`, `socialSecurityNumber` , `address(street, city, state, zip)`.

Se usará socialSecurityNumber como la clave primaria, pues siempre se tendrá dicho valor para cualquier empleado de la galería.

9. Owner (propietario)

   Los atributos potenciales son: `ownerAddress, ownerAreaCode, ownercity, ownerFirstName, ownerLastName, ownerPhone, ownerSocialSecurityNumber, ownerState, ownerStreet, ownerTelephone, Number,ownerZip`

   Sin embargo, dado que un propietario es o el artista de la obra o un coleccionista, y se tienen ítems de datos que corresponden a estas dos entidades, ya se tienen los valores de todos los ítems de datos owner, de modo que se elimina la entidad Owner.

Al examinar el diccionario de datos para ver si existe algún atributo sin contar, se ve que `dateOfReport`, `reportEndingDate`, `reportStartingDate`, `totalAskingPriceforPeriod`, `totalAllSalesforWeek`, aparecen en los reportes. Note que todos éstos son o datos calculados o efímeros que no tienen que almacenarse. También se ven muchos atributos para el propietario de una obra de arte, pero observe que, dado que un propietario es o un artista o un coleccionista, ya se tienen los valores de dichos atributos almacenados para dichas entidades.

- Paso 3.2. Haga una lista de relaciones a representar y cualquier atributo descriptivo para ellas.

  Las entidades son Artist, Collector, Buyer, PotentialCustomer, Artwork, Show, Sale y Salesperson. Al buscar relaciones entre ellas, se encuentran las siguientes.

1. Creates (crea): Artist relacionado con Artwork. Toda obra de arte en la galería tiene un artista que la creó. De hecho, Artwork no tiene una clave sin Artist, pues title no es único. Por tanto, Artwork será una entidad débil, dependiente de Artist a través de la relación Creates.
2. Owns (posee): En algunas instancias, la obra de arte no es propiedad del artista que la creó. Para estas instancias de entidad, Artwork se relaciona con Collector, a través de la relación Owns.
3. SoldIn: Artwork relacionada con Sale.
4. SoldBy: Sale relacionada con Salesperson.
5. SoldTo: Sale relacionada con Buyer.
6. ShownIn (expuesta en): Artwork relacionada con Show.
7. PreferredBy (preferida por): PotentialCustomer no parece estar fuertemente relacionada con alguna otra entidad. Sin embargo, un cliente potencial puede identificar un artista como una preferencia, de modo que se podría relacionar PotentialCustomer con Artist.
8. CollectedBy (coleccionada por): Dado que un coleccionista puede coleccionar obras predominantemente de un artista, se puede agregar una relación entre Artist y Collector.
9. FeaturedIn (presentación en): Esta relación se agrega para conectar un artista con la exposición "unipersonal" donde se presenta la obra de dicha persona.

Puesto que no quedaron atributos en el diccionario de datos, no existen atributos que dependan de las relaciones en este ejemplo.

- Paso 3.3. Dibuje un diagrama E-R para representar la empresa. Asegúrese de identificar la participación en relaciones y las restricciones de cardinalidad, cualquier conjunto de entidades débil y los nombres de rol, si es necesario.

En la figura 3.14 se muestra un diagrama E-R. Para construirlo, use los siguientes pasos. Comience el diagrama con la entidad Artist. Anteriormente se concluyó que Artwork es una entidad débil con referencia a Artist, de modo que se dibuja un rectángulo doble para Artwork y un diamante doble para la relación Creates. Cada artista puede tener cero o muchas obras de arte en la galería. Se usa una línea sencilla para permitir la posibilidad de que a un artista lo hayan entrevistado pero que ninguna de sus obras se haya seleccionado todavía. Cada obra de arte debe tener exactamente un artista, de modo que Artwork tiene participación total en la relación. Un artista puede crear muchas obras de arte, pero una obra de arte sólo tiene un artista. Ésta es una relación 1:M.

A continuación se agrega la entidad Collector. La relación Owns entre Collector y Artwork también es 1:M. Un coleccionista puede no tener todavía una obra de arte en la galería (participación parcial), pero una obra de arte debe tener un propietario. Sin embargo, el propietario puede ser el artista en vez de un coleccionista, de modo que también se elige participación parcial para Artwork.

La relación CollectedBy conecta un artista con el coleccionista, si hay alguno, que colecciona sus obras. Cada artista puede tener muchos coleccionistas, pero un coleccionista opcionalmente puede nombrar un artista en la forma, lo que hace a esta relación 1:M con participación parcial en ambos lados.

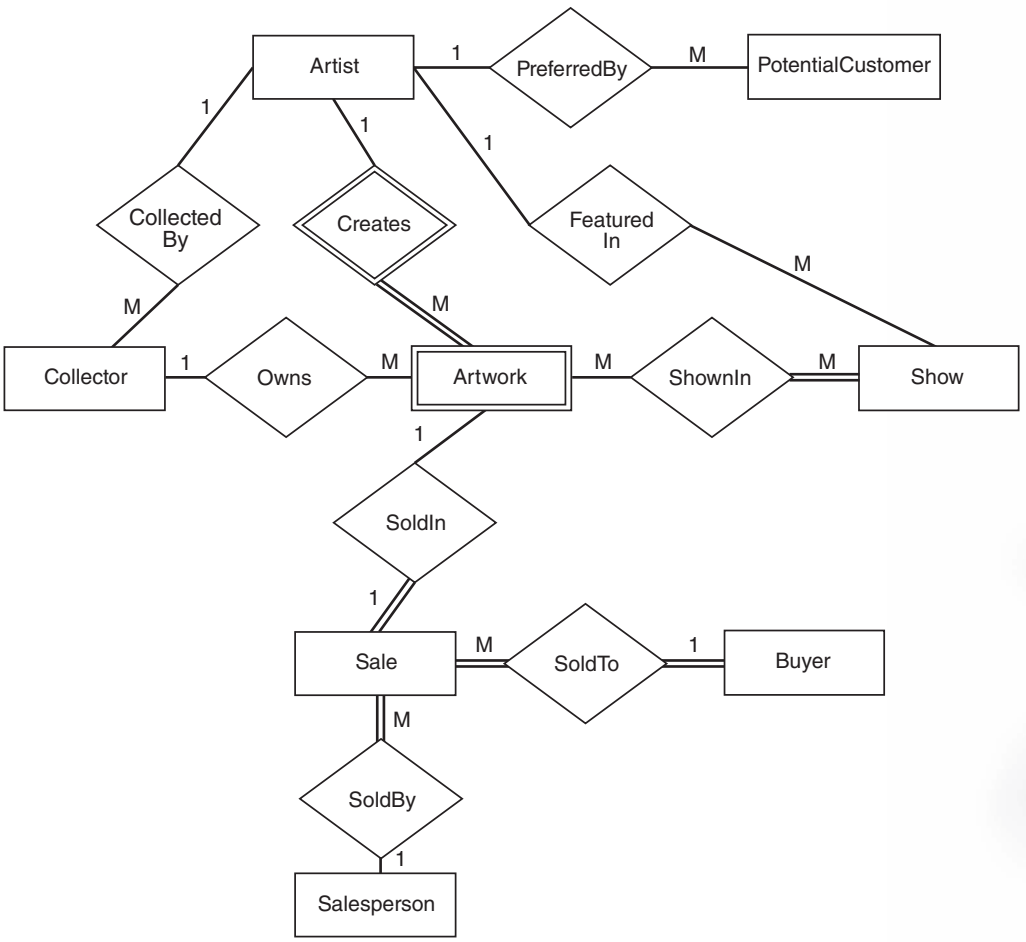
Ahora se agrega la entidad Sale. IsSold, la relación Artwork a Sale, es 1:1. En la lista de suposiciones, se dijo que cada obra de arte se vende una vez, cuando mucho, y cada venta es por una obra de arte. Una obra de arte puede o no venderse (participación parcial), pero una venta debe tener una obra de arte (participación total).

A continuación se agrega la entidad Show. La relación de Artwork a Show, ShownIn, es M:N, pues una obra de arte puede aparecer en más de una exposición, y cada exposición puede exhibir muchas obras de arte. Es participación parcial de Artwork, pues no todas las obras de arte aparecen en las exposiciones, y total para Show, pues una exposición debe incluir al menos una obra de arte.

La relación de Artist con Show, FeaturedIn, es 1:M, pues una exposición tiene, cuando mucho, un artista presentado, pero un artista podría presentarse en más de una exposición. La participación es parcial en ambos lados, pues un artista no tiene que presentarse en exposición alguna, y una exposición no tiene que presentar un artista.

Se puede agregar la entidad Salesperson. Sale a Salesperson, SoldBy, es M:1. Cada venta la realiza sólo un vendedor, pero un vendedor puede hacer muchas ventas. Una venta requiere un vendedor (participación total), pero un nuevo vendedor puede no tener ventas todavía (participación parcial). También se puede agregar la entidad Buyer. SoldTo, la relación de Sale a Buyer, es M:1. Una venta es para un comprador, pero el mismo comprador puede estar involucrado en muchas ventas. Una venta debe tener un comprador (participación total) y un comprador, por definición, debe ser alguien que haya comprado una obra de arte (participación total).

Finalmente se agrega la entidad PotentialCustomer. PreferredBy, la relación de Artist a PotentialCustomer, es 1:M, pues un cliente potencial puede identificar un artista, pero un artista puede tener muchos clientes potenciales que lo prefieran. Ésta es una relación parcial en ambos lados, pues un cliente potencial no tiene que establecer una preferencia por artista alguno, y un artista puede no tener clientes potenciales que lo elijan como preferido.



Artist: address (street, city, state, zip), interviewDate, interviewerName, name(firest, last), phone(areaCode, telephoneNumber), salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType

Artwork: askingPrice, dateListed, dateReturned, dateShown, status,workMedium, workSize, workStyle, workTitle, workType, workYearCompleted

Buyer: name(firstName, lastName), address(street, city, state, zip), phone(areaCode, telephoneNumber), purchasesLastYear, purchasesYearToDate

Collector: name(firstName, lastName), address(street, city, state, zip), interviewDate, interviewerName, phone(areaCode, telephonenumber), salesLastYear, salesYearToDate, collectionArtistFirstName, collectionArtistLastName, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate, SocialSecurityNumber

PotentialCustomer: address(street, city, state, zip), phone(areaCode, telephoneNumber), name(firstName, lastName), dateFilledIn, preferredArtist, preferredMedium, preferredStyle, preferredType.

Show: showFeatureArtist, showClosingDate, showTheme, showTitle, showOpeningDate

Sale: amoutRemittedToOwner, saleDate, InvoiceNumber, salePrice, sale SalesPersonCommission, saleTax, SaleTotal (*calculated field*)

Salesperson: name(firstName, lastName), socialSecurityNumber, address(street, city, state, zip)

FIGURA 3.14

Diagrama E-R para la Galería de Arte

- Paso 3.4. Actualizar el diccionario de datos y la lista de suposiciones según se necesite.

Revise el diccionario de datos. Los cambios están en *cursivas*.

amountRemittedtoOwner Cantidad de dinero en dólares enviada a un propietario por la venta de una obra de arte.

artistAddress Dirección de correo de un artista; *compuesto que consiste en artistStreet, artistCity, artistState, artistZip*.

artistAreaCode Código de área telefónica de un artista; *parte del compuesto artistPhone*.

artistCity Ciudad de la dirección de correo de un artista; *parte del compuesto artistAddress*.

artistFirstName Nombre dado que usa un artista; *parte del compuesto artistName*.

artistInterviewDate Fecha cuando un representante de la galería entrevistó a un artista.

artistInterviewerName Nombre completo del representante de la galería que entrevistó al artista.

artistLastName Apellido (sobrenombre) de un artista; *parte del compuesto artistName*.

artistName Nombre completo de un artista; *compuesto que consiste en artistFirstName, artistLastName*.

artistPhone Número telefónico completo del artista; *compuesto que consiste en artistAreaCode y artistTelephoneNumber*.

artistSalesLastYear Cantidad total, en dólares, de las ventas de las obras de un artista durante todo el año anterior.

artistSalesYearToDate Cantidad total, en dólares, de las ventas de las obras de un artista desde el primer día del año actual hasta el presente.

artistSocialSecurityNumber Registro federal de contribuyentes de un artista.

artistState Estado de la dirección de correo de un artista; *parte del compuesto artistAddress*.

artistStreet Número de casa y calle de la dirección de correo de un artista; *parte del compuesto artistAddress*.

artistTelephoneNumber Número telefónico de un artista, sin incluir código de área; *parte del compuesto artistPhone*.

artistTotalSalesforPeriod *Cantidad total, en dólares, de las ventas de las obras de un artista para el periodo cubierto en un reporte o transacción. (Ítem borrado.)*

artistTotalAskingPriceforPeriod *Valor total en dólares de las obras no vendidas de un artista para venderse en la galería por el periodo cubierto en un reporte o transacción, calculado como la suma de sus precios solicitados. (Ítem borrado.)*

artistZip Código postal de la dirección de correo de un artista; *parte del compuesto artistAddress*.

askingPrice Precio solicitado de una obra de arte.

buyerAddress Dirección de correo de un comprador de una obra de arte de la galería; *compuesto que consiste en buyerStreet, buyerCity, buyerState, buyerZip*.

buyerAreaCode Código de área telefónica de un comprador de una obra de arte de la galería; *parte del compuesto buyerPhone*.

buyerCity Ciudad de la dirección de correo de un comprador de una obra de arte de la galería; *parte del compuesto buyerAddress*.

buyerFirstName Nombre de un comprador de una obra de arte de la galería; *parte del compuesto buyerName*.

buyerLastName Apellido de un comprador de una obra de arte de la galería; *parte del compuesto buyerName*.

buyerName Nombre completo de un comprador de una obra de arte de la galería; *compuesto que consiste en buyerFirstName, buyerLastName*.

buyerPhone Número telefónico completo de un comprador de una obra de arte de la galería; *compuesto que consiste en buyerAreaCode y buyerTelephoneNumber*.

buyerState Estado de la dirección de correo de un comprador de una obra de arte de la galería; *parte del compuesto buyerAddress*.

buyerStreet Número de casa y calle de la dirección de correo de un comprador de una obra de arte de la galería; *parte del compuesto buyerAddress*.

buyerTelephoneNumber Número telefónico de un comprador de una obra de arte de la galería, no incluye código de área; *parte del compuesto buyerPhone*.

buyerZip Código postal del comprador de una obra de arte; *parte del compuesto buyerAddress*.

collectionArtistFirstName Nombre del artista que se presenta en un grupo de obras de arte propiedad de un coleccionista; *parte del compuesto collectionArtistName*.

collectionArtistLastName Apellido del artista que se presenta en un grupo de obras de arte propiedad de un coleccionista; *parte del compuesto collectionArtistName*.

collectionArtistName Nombre completo del artista que se presenta en un grupo de obras de arte propiedad de un coleccionista; *compuesto que consiste en collectionArtistFirstName, collectionArtistLastName*.

collectionMedium Medio utilizado por un grupo de obras de arte propiedad de un coleccionista.

collectionStyle Estilo de un grupo de obras de arte propiedad de un coleccionista.

collectionType Tipo de un grupo de obras de arte propiedad de un coleccionista.

collectorAddress Dirección de correo de un coleccionista de obras de arte; *compuesto que consiste en collectorStreet, collectorCity, collectorState, collectorZip*.

collectorAreaCode Código de área telefónico de un coleccionista de obras de arte; *parte del compuesto collectorPhone*.

collectorCity Ciudad de la dirección de correo de un coleccionista de obras de arte; *parte del compuesto collectorAddress*.

collectorFirstName Nombre dado por un coleccionista de obras de arte; *parte del compuesto collectorName*.

collectorInterviewDate Fecha cuando un representante de la galería entrevistó a un coleccionista de obras de arte.

collectorInterviewerName Nombre completo del representante de la galería que entrevistó a un coleccionista de obras de arte.

collectorLastName Apellido (sobrenombre) de un coleccionista de obras de arte; *parte del compuesto collectorName*.

collectorName Nombre completo del coleccionista de obras de arte; *compuesto que consiste en collectorFirstName, collectorLastName*.

collectorPhone Número telefónico completo de un coleccionista de obras de arte; *compuesto que consiste en collectorAreaCode, collectorTelephoneNumber*.

collectorSalesLastYear Cantidad total, en dólares, de las ventas de las obras de arte del coleccionista durante todo el año anterior.

collectorSalesYearToDate Cantidad total, en dólares, de las ventas de las obras de arte del coleccionista desde el primer día del año actual hasta el presente.

collectorSocialSecurityNumber Registro federal de contribuyentes de un coleccionista de obras de arte.

collectorState Estado de la dirección de correo de un coleccionista de obras de arte; *parte del compuesto collectorAddress*.

collectorStreet Número de casa y calle de la dirección de correo de un coleccionista de obras de arte; *parte del compuesto collectorAddresss*.

collectorTelephoneNumber Número telefónico de un coleccionista de obras de arte, no incluye código de área; *parte del compuesto collectorPhone*.

collectorTotalAskingPriceforPeriod *Valor total, en dólares, de las obras no vendidas de un coleccionista para venta en la galería por el periodo cubierto en un reporte o transacción, calculado como la suma de sus precios solicitados. (Ítem borrado.)*

collectorTotalSalesforPeriod *Cantidad total, en dólares, de las ventas de las obras del coleccionista para el periodo cubierto en un reporte o transacción. (Ítem borrado.)*

collectorZip Código postal de la dirección de correo de un coleccionista de obras de arte; *parte del compuesto collectorAddress*.

dateListed Fecha cuando una obra de arte se ofreció a la venta por primera vez en la galería.

dateOfReport *Fecha cuando se generó un reporte. (Ítem no almacenado.)*

dateReturned Fecha cuando la obra de arte se regresó a su propietario.

dateShown Fecha cuando una obra de arte se presentó en una exposición en la galería.

medium Medio de una obra de arte. Ejemplos de valores válidos son óleo, pastel, acuarela, medio acuoso, acrílico, mármol, acero, cobre, madera, fibra, otro.

ownerAddress *Dirección de correo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerAreaCode *Código de área telefónico del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerCity *Ciudad de la dirección de correo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerFirstName *Nombre dado que usa el propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerLastName *Apellido (sobrenombre) del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerPhone *Número telefónico completo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerSocialSecurityNumber *Registro federal de contribuyentes del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerState *Estado de la dirección de correo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerStreet *Número de casa y calle de la dirección de correo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerTelephoneNumber *Número telefónico del propietario de una obra de arte, no incluye código de área. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

ownerZip *Código postal de la dirección de correo del propietario de una obra de arte. (Ítem borrado; owner se sustituye o por Artist o por Collector.)*

potentialCustomerAddress Dirección de correo de un potencial cliente de la galería; *compuesto que consiste en potentialCustomerStreet, potentialCustomerCity, potentialCustomerState, potentialCustomerZip*.

potentialCustomerAreaCode Código de área telefónico de un potencial cliente de la galería; *parte del compuesto potentialCustomerPhone*.

potentialCustomerCity Ciudad de la dirección de correo de un potencial cliente de la galería; *parte del compuesto potentialCustomerAddress*.

potentialCustomerDateFilledIn Fecha cuando un cliente llenó la forma de información.

potentialCustomerFirstName Nombre de un potencial cliente de la galería; *parte del compuesto potentialCustomerName*.

potentialCustomerLastName Apellido (sobrenombre) de un potencial cliente de la galería; *parte del compuesto potentialCustomerName*.

potentialCustomerName Nombre completo de un potencial cliente de la galería; *compuesto que consiste en potentialCustomerFirstName, potentialCustomerLastName*.

potentialCustomerPhone Número telefónico completo de un potencial cliente de la galería; *compuesto que consiste en potentialCustomerAreaCode, potentialCustomerTelephoneNumber*.

potentialCustomerState Estado de la dirección de correo de un potencial cliente de la galería; *parte del compuesto potentialCustomerAddress*.

potentialCustomerStreet Número de casa y calle de la dirección de correo de un potencial cliente de la galería; *parte del compuesto potentialCustomerAddresss*.

potentialCustomerTelephoneNumber Número telefónico de un potencial cliente de la galería, no incluye código de área; *parte del compuesto potentialCustomerPhone*.

potentialCustomerZip Código postal de la dirección de correo de un potencial cliente de la galería; *parte del compuesto potentialCustomerAddress*.

preferredArtist Nombre del artista elegido como preferencia por un potencial cliente de la galería.

preferredMedium Medio elegido como preferido por un potencial cliente de la galería.

preferredStyle Estilo elegido como preferencia por un potencial cliente de la galería.

preferredType Tipo elegido como preferido por un potencial cliente de la galería.

purchasesLastYear Cantidad total, en dólares, de las ventas a un comprador durante todo el año anterior.

purchasesYearToDate Cantidad total, en dólares, de ventas a un comprador desde el primer día del año actual hasta el presente.

reportEndingDate *Fecha elegida como la fecha más reciente para usar la información en un reporte. (Ítem borrado.)*

reportStartingDate *Fecha elegida como la fecha más antigua para usar información en un reporte. (Ítem borrado.)*

saleDate Fecha cuando la galería vendió una obra de arte.

saleInvoiceNumber Número impreso en la factura para una venta de una obra de arte.

salePrice Precio al que la galería vendió una obra de arte.

salesPersonAddress Dirección completa de un socio de ventas que trabaja en la galería; *compuesto que consiste en salesPersonStreet, salesPersonCity, salesPersonState, salesPersonZip*.

salesPersonCommissionForPeriod *Cantidad total, en dólares, de la comisión ganada por un vendedor por un periodo específico. (Ítem borrado.)*

salesPersonFirstName Nombre dado de un socio de ventas que trabaja en la galería; *parte del compuesto salesPersonName*.

salesPersonLastName Apellido (sobrenombre) de un socio de ventas que trabaja en la galería; *parte del compuesto salesPersonName*.

salesPersonName Nombre completo de un socio de ventas que trabaja en la galería; *compuesto que consiste en salesPersonFirstName, salesPersonLastName*.

salesPersonSocialSecurityNumber Registro federal de contribuyentes de un socio de ventas que trabaja en la galería.

salesPersonTotalSalesForPeriod *Cantidad total, en dólares, de ventas, no incluidos impuestos, hechos por un vendedor durante un periodo específico. (Ítem borrado.)*

saleSalesPersonCommission Cantidad en dólares de la comisión para un vendedor por la venta de una obra de arte.

saleTax Cantidad en dólares del impuesto de ventas por la venta de una obra de arte.

saleTotal Cantidad total, en dólares, de una venta, incluidos precio e impuesto, por una obra de arte. *(Ítem borrado, se calcula a partir de salePrice y saleTax.)*

showClosingDate Fecha cuando se cierra una exposición al público.

showFeaturedArtist Nombre de un artista presentado en una exposición.

showTheme Tema de una exposición.

showTitle Título dado a una exposición.

showOpeningDate Fecha de apertura de una exposición al público.

size Tamaño de una obra de arte, expresada en pulgadas. Para obras bidimensionales, largo por ancho; para obras tridimensionales, largo por ancho por altura.

status Estatus de ventas de una obra de arte. Los posibles valores son vendida o sin vender.

style Estilo artístico de una obra de arte. Ejemplos de valores válidos son contemporáneo, impresionista, folk, otro.

title Título de una obra de arte.

totalAllSalesForWeek *Cantidad total, en dólares, de las ventas de la galería durante una semana específica, sin incluir impuestos. (Ítem borrado.)*

totalAskingPriceForPeriod *Suma de los precios solicitados para todas las obras durante el periodo elegido para un reporte. (Ítem borrado.)*

type Tipo de obra de arte. Ejemplos de valores válidos son pintura, escultura, collage, otro.

usualMedium Medio que generalmente usa el artista. Ejemplos de valores válidos son óleo, pastel, acuarela, medio acuoso, acrílico, mármol, acero, cobre, madera, fibra, otro.

usualStyle. Estilo artístico usual de las obras del artista. Ejemplos de valores válidos son contemporáneo, impresionista, folk, otro.

usualType Tipo de obra de arte que el artista produce normalmente. Ejemplos de valores válidos son pintura, escultura, collage, otro.

yearCompleted Año cuando se concluyó una obra de arte.

La lista de suposiciones tiene algunos cambios menores, que se indican en cursivas del modo siguiente:

1. Los nombres de los artistas son únicos, pero los nombres de los clientes y coleccionistas no lo son.
2. Por razones de privacidad, sólo a las personas que reciben pagos de la galería se les pide proporcionar su registro federal de contribuyentes, porque dichos pagos tienen que reportarse por razones de impuesto sobre la renta. Por tanto, la galería mantiene los R.F.C. de vendedores, coleccionistas y artistas, pero no de los compradores o clientes potenciales.
3. Un artista puede tener muchas obras en venta en la galería.
4. Cada obra es una pieza original exclusiva. No se venden impresiones o reproducciones.
5. Dos obras de arte pueden tener el mismo título, pero la combinación de título y nombre de artista es única.
6. Una obra de arte puede ser propiedad o del artista que la creó o de otra persona, a la que aquí se le refiere como coleccionista.
7. Incluso si la obra de arte es propiedad de un coleccionista, es importante mantener información acerca del artista que la creó, pues éste es un factor para la determinación de su valor.
8. La galería vende una obra de arte sólo una vez. La galería no revende sus propias obras.
9. Una obra de arte puede aparecer en más de una exposición. Algunas obras no aparecen en exposición alguna.
10. El pago por todas las ventas se realiza de inmediato y por completo al momento de la compra. El pago puede ser crédito, efectivo o cheque. Al propietario se le paga el saldo y al vendedor se le paga la comisión al terminar la semana.
11. La base de datos no incluye información de nómina, excepto por la comisión a pagar al vendedor por la venta de obras de arte.
12. Hay listas de valores válidos por tipo, estilo y medio de obras de arte. Cada una tiene un valor "otro" para las obras que no encajan en los valores existentes.
13. La información acerca de las obras no seleccionadas para ser listadas por la galería se descarta.
14. Las listas de artistas, coleccionistas, compradores y clientes potenciales se evalúa periódicamente para determinar si se deben retirar.
15. *Los títulos de las exposiciones son únicos. Cada exposición tiene o un solo artista presentado o un tema.*
16. *Una exposición por lo general presenta muchas obras de arte, pero es posible que se exhiba una sola pieza de arte importante.*

PROYECTOS ESTUDIANTILES: CREACIÓN DE DIAGRAMAS E-R PARA LOS PROYECTOS ESTUDIANTILES

Para el proyecto que haya elegido, haga lo siguiente:

- Paso 3.1. Elabore una lista de todas las entidades y sus atributos asociados.
- Paso 3.2. Elabore una lista de relaciones a representar, y cualquier atributo descriptivo para ellas.
- Paso 3.3. Dibuje un diagrama E-R para representar la empresa. Asegúrese de identificar la participación en relaciones y las restricciones de cardinalidad, conjuntos de entidades débiles y nombres de rol si es necesario.
- Paso 3.4. Actualice el diccionario de datos y mencione las suposiciones según se requiera.

CAPÍTULO 4

El modelo relacional

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Los orígenes del modelo relacional
- Algunas ventajas del modelo relacional
- Cómo se usan las tablas para representar datos
- La conexión entre relaciones matemáticas y relaciones del modelo relacional
- Características de las relaciones de base de datos
- Cómo identificar claves de relación
- El significado de la integridad de entidad y la integridad referencial
- Cómo escribir un esquema relacional
- Las categorías de los lenguajes relacionales
- Cómo escribir consultas en álgebra relacional
- Cómo se expresan las consultas en cálculo relacional
- Cómo se definen las vistas en el modelo relacional
- Cómo transformar un diagrama E-R en un modelo relacional para una base de datos

4.1 Breve historia del modelo relacional

El modelo relacional lo propuso por primera vez Codd en 1970, en un ensayo titulado "A Relational Model of Data for Large Shared Data Banks". Gran parte de las primeras investigaciones acerca del modelo la realizaron Codd y sus asociados en el laboratorio de investigación de IBM, anteriormente en San José, California. En aquella época, el mercado estaba dominado por los sistemas de gestión de base de datos con modelo jerárquico y de red, que usaban estructuras de datos y almacenamiento complejas y eran difíciles de entender para los usuarios. Un prototipo de sistema de gestión de base de datos relacional, llamado **System R** (sistema R), lo desarrollaron investigadores de IBM a finales de la década de 1970. Este proyecto se diseñó para demostrar la practicidad del modelo relacional al proporcionar una implementación de sus estructuras de datos y operaciones. También resultó ser una excelente fuente de información acerca de técnicas de implementación relacionadas como control de concurrencia, optimización de consulta, gestión de transacción, seguridad e integridad de datos, recuperación, factores humanos e interfaces de usuario, y condujo a muchos ensayos de investigación y otros prototipos. System R es la base de DB2, el sistema de gestión de base de datos relacional de IBM. El **Peterlee Relational Test Vehicle** (vehículo de prueba relacional Peterlee) fue un primer proyecto más teórico desarrollado en el laboratorio científico de IBM en el Reino Unido.

INGRES fue otro primer proyecto de investigación de modelo relacional, desarrollado en la Universidad de California en Berkeley aproximadamente al mismo tiempo que el proyecto System R. La investigación condujo a una versión "universitaria" de INGRES, así como a un producto comercial. **ORACLE** se desarrolló y comercializó usando muchos de los resultados de System R. Entre los primeros sistemas de gestión de bases de datos relacionales basados en microcomputadoras estaban **dBase**, **R:base** y **Paradox**. El **Access** de Microsoft, que usa el modelo relacional, es ahora el sistema de gestión de base de datos basado en microcomputadora más ampliamente utilizado. **Oracle**, **DB2**, **Informix**, **Sybase** y el **SQL Server** de Microsoft, todos los cuales usan el modelo relacional, en la actualidad son los sistemas de gestión de bases de datos empresariales más populares.

4.2 Ventajas del modelo relacional

El modelo relacional se basa en la noción matemática de **relación**. Codd y otros extendieron la noción para aplicarla al diseño de bases de datos. Por ende, fueron capaces de sacar ventaja del poder de la abstracción matemática y de la expresividad de la notación matemática para desarrollar una estructura simple, pero poderosa, para las bases de datos. El modelo relacional es el tema de una gran colección de literatura de investigación.

Mucha de la literatura trata el modelo teóricamente y desarrolla aspectos del modelo con el uso del enfoque matemático de teorema y prueba. Este enfoque abstracto tiene la ventaja de que los resultados son generales, lo que significa que no dependen de un ejemplo particular, y se pueden aplicar a muchas aplicaciones diferentes. Por fortuna, se pueden usar los resultados de la teoría para diseñar modelos relacionales sin necesariamente seguir el intrincado proceso de probar un teorema. La teoría ayuda en el diseño de la base de datos al identificar potenciales fallos en los diseños propuestos, y ofrece herramientas para permitir corregir dichos fallos. El capítulo 6 contiene una discusión detallada de algunas de estas técnicas.

La estructura básica del modelo relacional es simple, lo que lo hace sencillo de entender desde el punto de vista intuitivo. Permite la separación de los niveles conceptual y físico, de modo que el diseño conceptual se puede realizar sin considerar estructuras de almacenamiento. Los usuarios y diseñadores encuentran que el modelo les permite expresar nociones de datos conceptuales en una forma que se entiende con facilidad. Las operaciones de datos

también se expresan de una manera sencilla y no requieren que el usuario esté familiarizado con las estructuras de almacenamiento utilizadas. El modelo usa pocos comandos muy poderosos para lograr manipulaciones de datos que varían de lo simple a lo complejo. Por estas razones, el modelo relacional se ha convertido en el modelo más popular para las bases de datos.

4.3 Estructuras de datos relacionales

Las estructuras de datos utilizadas en el modelo relacional son tablas con relación entre ellas (figura 4.1).

4.3.1 Tablas

El modelo relacional se basa en el concepto de **relación**, que se representa físicamente como una **tabla** o arreglo bidimensional. En este modelo, las tablas se usan para contener información acerca de los objetos a representar en la base de datos. Al usar los términos del modelo entidad-relación, los conjuntos de entidades y de relaciones se muestran usando tablas. Una relación se representa como una tabla bidimensional en la que las filas de la tabla corresponden a registros individuales y las columnas corresponden a atributos. Por ejemplo, la relación Student se representa mediante la tabla `Student`, que tiene columnas para los atributos `stuId, lastName, firstName, major` y `credits`. La figura 4.1(a), que es una copia de la figura 1.1(a), muestra una instancia de la tabla `Student`. Note que los nombres de columna, `stuId`, `lastName`, `firstName`, `major` y `credits`, son los mismos que los nombres del atributo. Como puede ver en este ejemplo, una columna contiene valores de un solo atributo; por ejemplo, la columna `stuId` contiene sólo las ID de estudiantes. El **dominio** de un atributo es el conjunto de valores permisibles para

FIGURA 4.1
Las tablas University

FIGURA 4.1(a)
La tabla Student

Student				
stuId	**lastName**	**firstName**	**major**	**credits**
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15

FIGURA 4.1(b)
La tabla Faculty

Faculty			
facId	**name**	**department**	**rank**
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

FIGURA 4.1(c)
La tabla Class

Class			
classNo	**facId**	**schedule**	**room**
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
CSC203A	F105	MThF12	M110
HST205A	F115	MWF11	H221
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

FIGURA 4.1(d)
La tabla Enroll

Enroll		
stuId	**classNo**	**grade**
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

dicho atributo. Los dominios pueden ser distintos, o dos o más atributos pueden tener el mismo dominio. Cada fila de la tabla corresponde a un registro individual o instancia de entidad. En el modelo relacional, cada fila se llama **tupla**.

Una tabla que representa una relación tiene las siguientes características:

- Cada celda de la tabla contiene sólo un valor.
- Cada columna tiene un nombre distinto, que es el nombre del atributo que representa.
- Todos los valores en una columna provienen del mismo dominio, pues todos son valores del atributo correspondiente.
- Cada tupla o fila es distinta; no hay tuplas duplicadas.
- El orden de las tuplas o filas es irrelevante.

Para mostrar lo que significan estas restricciones, use la tabla `Student` como ejemplo. Dado que cada celda debe contener sólo un valor, es ilegal almacenar valores múltiples para un atributo de una sola entidad. Por ejemplo, no se pueden almacenar dos especialidades para un estudiante en una sola celda. Los nombres de columna escritos en la parte superior de las columnas corresponden a los atributos de la relación. Los valores en la columna `stuId` son todos del dominio de `stuId`; no se permitiría que el nombre de un estudiante apareciera en esta columna. No puede haber filas duplicadas, porque cada estudiante individual está representado sólo una vez. Por ejemplo, la fila que contiene (S1001, Smith, Tom, History, 90) aparece sólo una vez. Las filas se pueden intercambiar a voluntad, de modo que los registros de S1001 y S1002 se pueden cambiar, sin variar la relación.

4.3.2 Relaciones matemáticas

Para entender el significado estricto del término relación, es necesario revisar algunas nociones de matemáticas. Suponga que tiene dos conjuntos, $\mathbf{D_1}$ y $\mathbf{D_2}$, con $\mathbf{D_1} = \{1,3\}$ y $\mathbf{D_2} = \{a,b,c\}$. Puede formar el producto cartesiano de estos dos conjuntos, $\mathbf{D_1} \times \mathbf{D_2}$, que es el conjunto de todos los pares ordenados tales que el primer elemento es un miembro de $\mathbf{D_1}$ y el segundo elemento es un miembro de $\mathbf{D_2}$. Otra forma de pensar en esto es encontrar todas las combinaciones de elementos con el primero de $\mathbf{D_1}$ y el segundo de $\mathbf{D_2}$. Por ende, se encuentra lo siguiente:

$D_1 \times D_2 = \{(1,a), (1,b), (1,c), (3,a), (3,b), (3,c)\}$

Una relación es simplemente algún subconjunto de este producto cartesiano. Por ejemplo,

$R = \{(1,a), (3,a)\}$

es una relación. Con frecuencia, se especifica cuáles pares ordenados estarán en la relación al proporcionar alguna regla para su selección. Por ejemplo, **R** incluye a todos aquellos pares ordenados en los que el segundo elemento es **a**, de modo que **R** se puede escribir como,

$R = \{(x,y) \mid x \in D_1, y \in D_2, y\ y = a\}$

Al usar estos mismos conjuntos, podría formar otra relación, **S**, en la que el primer elemento siempre es 3. Por tanto,

$S = \{(x,y) \mid x \in D_1, y \in D_2, y\ x = 3\}$

De manera alternativa, **S** = {(3, a), (3, 6), (3, c)}, pues sólo existen tres pares ordenados en el producto cartesiano que satisfacen la condición.

La noción de relación se podría extender a tres conjuntos en una forma natural. Sean $\mathbf{D_1}$, $\mathbf{D_2}$ y $\mathbf{D_3}$ los tres conjuntos. Entonces el producto cartesiano $\mathbf{D_1} \times \mathbf{D_2} \times \mathbf{D_3}$ de estos tres conjuntos es el conjunto de todas las tripletas ordenadas tales que el primer elemento es de $\mathbf{D_1}$, el segundo elemento es de $\mathbf{D_2}$ y el tercer elemento es de $\mathbf{D_3}$. Entonces, una relación es cualquier subconjunto de este producto cartesiano. Por ejemplo, suponga que los conjuntos se definen como

$D_1 = \{1,3\} \quad D_2 = \{2,4,6\} \quad D_3 = \{3,6,9\}$

Entonces $D_1 \times D_2 \times D_3 = \{(1,2,3), (1,2,6), (1,2,9), (1,4,3), (1,4,6), (1,4,9), (1,6,3), (1,6,6), (1,6,9), (3,2,3), (3,2,6), (3,2,9), (3,4,3), (3,4,6), (3,4,9), (3,6,3), (3,6,6), (3,6,9)\}$

Una relación es cualquier subconjunto de estas tripletas ordenadas. Por ejemplo, se podría definir una relación **T** como aquellas tripletas ordenadas en las que el tercer elemento es la suma de los dos primeros. Entonces se tiene

$\mathbf{T} = \{(x,y,z) \mid x \in \mathbf{D_1}, y \in \mathbf{D_2}, z \in \mathbf{D_3}\ y\ z = x + y\}$
o $\mathbf{T} = \{(1,2,3), (3,6,9\}$.

Se puede ir más allá de tres conjuntos y definir una relación general sobre *n* dominios. Sean $\mathbf{D_1}, \mathbf{D_2}, \ldots, \mathbf{D_n}$ los *n* conjuntos. Su producto cartesiano se define como

$\mathbf{D_1} \times \mathbf{D_2} \times \ldots \times \mathbf{D_n} = \{(d_1,d_2, \ldots, d_n) | d_1 \in \mathbf{D_1}, \mathbf{d_2} \in \mathbf{D_2}, \ldots, \mathbf{d_n} \in \mathbf{D_n}\}$

Usualmente, el producto cartesiano se escribe

$\Pi_{i=1}^{n} D_i$

Una relación sobre los *n* conjuntos es cualquier conjunto de *n*-tuplas elegidas de este producto cartesiano. Note que, al definir relaciones, se tienen que especificar los dominios o los conjuntos de los que se escogen valores.

4.3.3 Relaciones y tablas de bases de datos

Al aplicar estos conceptos a las bases de datos, sean $\mathbf{A_1}$, $\mathbf{A_2}$, . . ., $\mathbf{A_n}$ los atributos con dominios $\mathbf{D_1}$, $\mathbf{D_2}$, . . ., $\mathbf{D_n}$. Un esquema de relación **R** es un conjunto de atributos con sus correspondientes dominios. De esta manera, el conjunto {$\mathbf{A_1}$, $\mathbf{A_2}$, . . ., $\mathbf{A_n}$} con dominios correspondientes {$\mathbf{D_1}$, $\mathbf{D_2}$, . . ., $\mathbf{D_n}$} es un esquema de relación. Una **relación r** sobre un esquema de relación **R** es un conjunto de mapeos de los nombres de atributo a sus dominios correspondientes. Por tanto, la relación **r** es un conjunto de *n*-tuplas ($\mathbf{A_1}$:d_1, $\mathbf{A_2}$:d_2, . . ., $\mathbf{A_n}$:d_n) tales que $d_1 \in \mathbf{D_1}$, $d_2 \in \mathbf{D_2}$, . . ., $d_n \in \mathbf{D_n}$. Cada elemento en una de estas *n*-tuplas consiste en un atributo y un valor de dicho atributo. Por lo general, cuando una relación se escribe como tabla, los nombres de los atributos se listan como encabezados de columna y las tuplas se escriben simplemente usando los valores elegidos de los dominios adecuados, de modo que las *n*-tuplas se considera que tienen la forma (d_1, d_2, . . ., d_n). De esta forma, se puede considerar una relación en el modelo relacional como cualquier subconjunto del producto cartesiano de los dominios de los atributos. Una tabla es simplemente una representación de tal relación.

Para el ejemplo de universidad, la relación `Student` tendría los atributos `stuId`, `lastName`, `firstName`, `major` y `credits`, cada uno con sus dominios correspondientes. La relación `Student` es cualquier subconjunto del producto cartesiano de los dominios, o cualquier conjunto de 5-tuplas en las que el primer elemento es una ID de estudiante, el segundo es el apellido de un estudiante, el tercero es el nombre del estudiante, el cuarto es una especialidad y el quinto es un número de créditos. Algunas de las 5-tuplas son:

```
{
(S1001, Smith, Tom, History, 90),
(S1002, Chin,  Ann, Math,    36)
. . .}
```

De manera más adecuada, estas 5-tuplas son:

```
{
(stuId:S1001, lastName:Smith, firstName:Tom, major:History, credits:90),
(stuId:S1002, lastName:Chin, firstName:Ann, major:Math, credits:36)
. . .}
```

La tabla `Student`, que se muestra en la figura 4.1(a), es una forma conveniente de escribir todas las 5-tuplas que satisfacen la relación en el momento. Desde luego, las 5-tuplas son las filas de la tabla, lo que explica por qué las filas de tabla en el modelo relacional se llaman **tuplas**. La tabla, con todas sus filas escritas, es una instancia o **extensión** de la relación. La estructura de la tabla, junto con una especificación de los dominios y cualesquiera otras restricciones sobre los posibles valores, muestra la **intensión** de la relación, también llamada **esquema de la base de datos**. Estrictamente hablando, el esquema también incluye dominios, vistas, conjuntos de caracteres, restricciones, procedimientos almacenados, autorizaciones y otra información relacionada. Los esquemas de relación se pueden representar al dar el nombre de cada relación, seguido por los nombres de atributos entre paréntesis, como en

```
Student (stuId, lastName, firstName, major, credits)
```

El atributo `stuId` está subrayado porque es obligatorio destacar la clave primaria en el esquema de relación.

4.3.4 Propiedades de las relaciones

La mayoría de las características especificadas para las tablas resulta de las propiedades de las relaciones. Dado que una relación es un conjunto, el orden de los elementos no cuenta.

Por tanto, en una tabla, el orden de las filas es irrelevante. En un conjunto, ningún elemento se repite. De igual modo, en una tabla no hay filas duplicadas. Cuando se encuentran los productos cartesianos de conjuntos con elementos simples de un solo valor, como los enteros, cada elemento en cada tupla tenía valor sencillo. De igual modo, cada celda de una tabla contiene sólo un valor. En una relación, los posibles valores para una posición dada están determinados por el conjunto o dominio sobre el cual se define la posición. En una tabla, los valores en cada columna deben provenir del mismo dominio de atributo. En una relación matemática, el orden de los elementos en una tupla es importante. Por ejemplo, el par ordenado (1,2) es muy diferente del par ordenado (2,1), como se dará cuenta cuando ubique los dos puntos en un plano con coordenadas cartesianas. Sin embargo, en las tablas, el orden de las columnas es irrelevante. La tabla `Student` se considera como la misma incluso si se ponen las columnas en orden diferente, en tanto se usen los mismos datos. La razón es que los encabezados de columna indican a cuál atributo pertenece el valor. Sin embargo, una vez elegida la estructura de la tabla, el orden de los elementos dentro de las filas de la extensión debe coincidir con el orden de los nombres de columna.

4.3.5 Grado y cardinalidad

El número de columnas en una tabla se llama **grado** de la relación. El grado de la relación `Student` es cinco, pues la tabla `Student` tiene cinco columnas. Esto significa que cada fila de la tabla es una 5-tupla, que contiene cinco valores. Una relación con una sola columna tendría grado uno y se llamaría relación **unaria**. Una relación con dos columnas se llama **binaria**, una con tres columnas se llama **ternaria** y, después de ella, por lo general se usa el término ***n*-aria**. El grado de una relación es parte de la intensión de la relación y nunca cambia.

En contraste, el número de filas en una tabla, llamada la **cardinalidad** de la relación, cambia conforme se agregan nuevas tuplas o se borran las antiguas. La cardinalidad es una propiedad de la extensión de la relación, la instancia particular de la relación en cualquier momento dado.

4.3.6 Claves de la relación

Dado que una relación es un conjunto y un conjunto no tiene elementos duplicados, siempre es posible separar los elementos de un conjunto. En una relación de base de datos, las tuplas son los elementos del conjunto, de modo que siempre debe ser posible separar las tuplas. Esto significa que siempre debe haber algún atributo, o alguna combinación de atributos, que haga distintas a las tuplas. Por tanto, siempre habrá una superclave para cada relación, y esto implica que siempre habrá una superclave mínima, lo que significa que siempre hay una clave candidata. Por tanto, siempre se puede encontrar una clave primaria para cada relación. En el peor caso, todo el conjunto de atributos necesitaría examinarse para separar las tuplas. Sin embargo, por lo general, algún conjunto más pequeño de atributos es suficiente para distinguir entre tuplas. Al aplicar las definiciones del capítulo 3, una **superclave** para una relación es un conjunto de atributos que identifica de manera única una tupla, una **clave candidata** es una superclave tal que ningún subconjunto propio es una superclave, y una **clave primaria** es la clave candidata que en realidad se elige para identificar de manera única las tuplas. Note que una instancia de la tabla no se puede usar para probar que un atributo o combinación de atributos es una clave candidata. El hecho de que no haya duplicados para los valores que aparecen en un momento particular no garantiza que los duplicados no sean posibles. Sin embargo, la presencia de duplicados en una instancia se puede usar para mostrar que alguna combinación de atributos no es una clave candidata. Identificar una clave candidata requiere considerar el significado de los atributos involucrados, de

modo que se pueda tomar una decisión acerca de si los duplicados son posibles en el mini-mundo. Sólo al usar esta información semántica e identificar la suposición para el modelo se puede asegurar que una combinación de atributos es una clave candidata.

Una **clave externa** es un atributo o combinación de atributos de una relación que no es la clave primaria de dicha relación, pero que es la clave primaria de alguna relación, por lo general una diferente. Las claves externas son muy importantes en el modelo relacional, porque se usan para representar conexiones lógicas entre relaciones. Considere una relación Employee con esquema

```
Employee(empId, lastName, firstName, departmentName)
```

y también una relación Department con esquema

```
Department(departmentName, office)
```

El atributo `departmentName`, que es una clave primaria en la relación `Department`, es una clave externa en `Employee`. Note que, aunque tiene valores únicos en la tabla `Department`, no es única en la tabla `Employee`, pues muchos empleados pueden pertenecer al mismo departamento. Tener dicho atributo en la tabla `Employee` permite decir a cuál departamento pertenece un empleado. Cuando se tiene una clave externa, a la relación en la que el atributo o combinación es la clave primaria se le llama **relación base** (home). `Department` es la relación base para `departmentName`.

En las tablas que se muestran en las figuras 4.1(a)-(d), `facId` es una clave externa en `Class`. No es la clave primaria de dicha tabla, pero es la clave primaria de una tabla diferente, `Faculty`. Tener dicho atributo en la tabla `Class` permite conectar una clase con el miembro del personal docente que la imparte, y muestra la relación entre estas tablas. En la tabla `Enroll`, `stuId` es una clave externa. No es, por sí misma, la clave primaria de `Enroll`, pero es la clave primaria de `Student`. De igual modo, `classNo` es una clave externa en `Enroll`, pero es la clave primaria de `Class`. Estos dos atributos en `Enroll` permiten decir en cuáles clases está inscrito un estudiante, y cuáles estudiantes están inscritos en cada clase.

La relación base no tiene que ser una tabla separada. Por ejemplo, suponga que un esquema de empleados aparece como

```
Employee(empId, firstName, lastName, managerEmpId)
```

Aunque `empId` es la clave primaria de esta tabla, `managerEmpId` es una clave externa que en realidad se refiere al `empId` de otra tupla en la misma tabla. El registro de cada empleado contiene el `empId` del supervisor de dicha persona, que desde luego ocurre en el registro empleado del supervisor como su `empId`.

4.4 Restricciones de integridad: dominio, clave, clave externa, restricciones generales

Es importante preservar la **integridad**, que significa la exactitud y consistencia interna, de los datos en la base de datos, al no permitir a los usuarios ingresar datos que haría incorrecta la base de datos. El modelo relacional permite definir **restricciones de integridad** (**IC**, por sus siglas en inglés), que son reglas o restricciones que se aplican a todas las instancias de la base de datos. Un **estado legal** de la base de datos es aquel que obedece todas las restricciones de integridad. Parte del trabajo de un sistema de gestión de base de datos es reforzar las restricciones de integridad, para asegurar que cualquier dato ingresado cree una instancia legal de la base de datos.

Dado que todo atributo tiene un dominio asociado, existen restricciones sobre el conjunto de valores permitidos para los atributos de las relaciones. A éstas se le llaman **restricciones de dominio** y se pueden reforzar en SQL al definir el dominio para un atributo cuando se crea una tabla. Una alternativa es usar uno de los tipos de datos interconstruidos y agregar una restricción llamada opción CHECK (verificación), que permite especificar que el valor de un atributo debe estar dentro de un conjunto específico de valores, o que debe obedecer una condición que se especifica al usar predicados lógicos.

Una restricción de clave primaria, llamada **integridad de entidad**, establece que, en una relación, **ningún atributo de una clave primaria puede tener un valor nulo**. Por definición, una clave primaria es un identificador mínimo que se usa para reconocer tuplas de manera única. Esto significa que ningún subconjunto de la clave primaria es suficiente para proporcionar identificación única de tuplas. Si se permitiera un valor nulo para cualquier parte de la clave primaria, se demostraría que no todos los atributos son necesarios para distinguir entre tuplas, lo que contradeciría la definición. Todas las claves de relación, tanto primarias como candidatas, deben tener valores únicos. En SQL, la clave primaria se puede identificar al usar una restricción de clave primaria cuando se crea la tabla. Entonces el sistema reforzará automáticamente las restricciones de no nulo y exclusividad. Para claves candidatas, la mayoría de los sistemas permiten especificar tanto la restricción de exclusividad como la restricción de no nulo.

Una regla de integridad, llamada **integridad referencial**, se aplica a claves externas. La integridad referencial afirma que, si en una relación existe una clave externa, entonces o el valor de la clave externa debe coincidir con el valor de la clave primaria de alguna tupla en su relación base, o el valor de la clave externa debe ser completamente nulo. SQL permite especificar restricciones de clave externa cuando se crea una tabla.

Existen muchos otros tipos de restricciones, conocidos como **restricciones generales**, que se discutirán en capítulos posteriores. Muchas de tales restricciones se pueden expresar en SQL como **restricciones de tabla**, que son reglas que gobiernan los valores permitidos dentro de una tabla, o como **aserciones SQL**, que son enunciados que especifican que un predicado lógico dado debe ser verdadero para todos los estados de la base de datos. Las restricciones generales se refuerzan mediante el sistema de gestión de base de datos siempre que se hacen cambios a los datos. El sistema verifica que los cambios no violan las restricciones antes de permitirles volverse permanentes.

4.5 Representación de esquemas de bases de datos relacionales

Un esquema de base de datos relacional puede tener cualquier número de relaciones. Los esquemas de relación se pueden representar al dar el nombre de cada relación, seguido de los nombres de atributo entre paréntesis, con la clave primaria subrayada. No hay una forma estándar simple de indicar sin ambigüedades las claves externas, pero en el texto se usarán *cursivas* para indicar que un atributo es una clave externa, o parte de una clave externa. Desafortunadamente, este método no siempre dirá con claridad cuál es la relación base. Además, si se tienen claves externas compuestas, y los atributos se traslapan, se tendría dificultad para identificar con exactitud cuál es la clave externa. La forma más clara de mostrar las claves externas es mediante el dibujo de flechas desde las claves externas hasta las claves primarias a las que hacen referencia. Recuerde que el esquema en realidad también incluye dominios, vistas, conjuntos de caracteres, restricciones, procedimientos almacenados, autorizaciones y otra información relacionada, de modo que la representación realmente sólo es una parte del esquema. Para el ejemplo de universidad, una base de datos simple puede

contener relaciones para `Student`, `Class`, `Faculty` y `Enroll`. Los esquemas de relación se escribirían como:

```
Student (stuId, lastName, firstName, major, credits)
Class (classNo, facId, schedule, room)
Faculty (facId, name, department, rank)
Enroll(stuId,classNo, grade)
```

El esquema de relación proporciona el nombre de la tabla para cada relación y encabezado de columna para cada uno de sus atributos. El esquema de modelo lógico es el conjunto de todos estos esquemas para la base de datos. La figura 4.1 muestra una instancia de esta base de datos.

Note que algunos atributos, como las claves externas, aparecen en más de una relación. Por ejemplo, `stuId` aparece en `Student` y `Enroll`. Cuando se quiere distinguir entre las dos apariciones de `stuId`, se usa el nombre de la relación, seguido por un punto, seguido por el nombre del atributo. Los dos nombres calificados para `stuId` son `Student.stuId` y `Enroll.stuId`. Un nombre todavía más explícito comienza con el nombre de la base de datos, luego el nombre de la tabla, después el nombre del atributo, como en `University.Student.stuId`. Cuando un atributo aparece en más de una relación, su aparición por lo general representa una relación o interacción entre tuplas de las dos relaciones. Estos atributos comunes juegan un importante papel al realizar manipulación de datos, como se verá en secciones posteriores.

4.6 Lenguajes de manipulación de datos relacionales

Existe una diversidad de lenguajes utilizados para los sistemas de gestión de base de datos relacionales.

4.6.1 Categorías de los DML

Algunos de los lenguajes son **procedurales** o proscriptivos, lo que significa que el usuario le dice al sistema exactamente cómo manipular los datos. El **álgebra relacional** es un ejemplo de lenguaje procedural que se discutirá.

Otros son **no procedurales** o declarativos, lo que significa que el usuario establece cuáles datos son necesarios mas no exactamente cómo se les localiza. El **cálculo relacional** y **SQL** son lenguajes no procedurales. Algunos lenguajes son **gráficos**, lo que permite al usuario dar un ejemplo o ilustración de cuáles datos se deben encontrar. **QBE** (QueryByExample: consulta por ejemplo) es un lenguaje gráfico que permite a los usuarios proporcionar ejemplos de datos que les gustaría recuperar. Otra categoría es el **lenguaje de cuarta generación** (4GL), que permite crear una aplicación personalizada completa mediante algunos comandos en un ambiente amigable para el usuario, con frecuencia activado por menú. Algunos sistemas aceptan una variedad de **lenguajes naturales**, en ocasiones llamados **lenguajes de quinta generación**, una versión restringida del inglés natural.

Tanto el álgebra relacional como el cálculo relacional son lenguajes formales, no amigables para el usuario. No se implementan en su forma nativa en los sistemas de gestión de base de datos, pero ambos se han usado como la base para otros lenguajes de manipulación de datos de nivel superior para bases de datos relacionales. Son de interés porque ilustran las operaciones básicas requeridas por cualquier lenguaje de manipulación de datos, y porque sirven como el estándar de comparación para otros lenguajes relacionales.

4.6.2 Álgebra relacional

El álgebra relacional es un lenguaje teórico con operadores que se aplican en una o dos relaciones para producir otra relación. Por ende, tanto los operandos como el resultado son tablas. Date originalmente propuso ocho operaciones, pero se han desarrollado muchas otras. Tres operaciones muy básicas, SELECT, PROJECT y JOIN, permiten realizar la mayoría de las operaciones de recuperación de datos que interesan. Se usará el ejemplo de base de datos universitaria de la figura 4.1 para ilustrar estas operaciones. Existen muchas variaciones en la sintaxis de los comandos del álgebra relacional, y aquí se usará uno muy simple parecido al inglés y se le presentará de manera informal. También se incluirá la notación simbólica más formal para los comandos. Existen muchas variaciones de las operaciones que se incluyen en el álgebra relacional. Suponga que se tiene la opción de asignar los resultados a una tabla que se nombra en el comando. [**Nota:** En una discusión más formal se usaría una operación "rename" (renombrar) especial para hacer esto.]

El operador SELECT, σ

El comando SELECT se aplica a una sola tabla y toma filas que satisfacen una condición específica, y las copia en una nueva tabla. De manera informal, la forma general es:

SELECT *nombreTabla* WHERE *condición* [GIVING *nombreTablaNueva*]

Simbólicamente, la forma es:

[*nombreTablaNueva* =] $\sigma_{\text{predicado}}$(*nombreTabla*)

o simplemente

σ_{θ}(*nombreTabla*)

Por ejemplo, si quiere encontrar toda la información en la tabla `Student` acerca del estudiante S1013, informalmente se escribiría

```
SELECT Student WHERE stuId = 'S1013' GIVING Result
```

o simbólicamente

$\texttt{Result} = \sigma_{\texttt{STUID='S1013'}}\ \texttt{(Student)}$

El comando produce una nueva tabla que se llama `Result`, que se parece a esto:

Result:

stuId	lastName	firstName	major	credits
S1013	McCarthy	Owen	Math	0

La operación SELECT se puede usar para encontrar más de una fila. Por ejemplo, para encontrar todas las clases que se imparten en el salón H225, se escribiría

```
SELECT Class WHERE room = 'H225' GIVING Answer
```

o simbólicamente

$\texttt{Answer} = \sigma_{\texttt{room='H225'}}\ \texttt{(Class)}$

Este comando produce la siguiente tabla:

Answer:

classNo	facId	schedule	room
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

Note que la operación se realiza sobre una tabla existente, *nombre tabla*, y produce una nueva tabla que es un subconjunto horizontal de la antigua y que se puede nombrar e indi-

car por *nuevo nombre tabla*. Los corchetes indican que la parte del enunciado encerrado en ellos es opcional. La tabla antigua todavía existe bajo su nombre anterior y tanto ella como la tabla nueva están disponibles para operaciones adicionales. Si simplemente se desea encontrar y desplegar las filas que satisfacen la condición, pero no planea posteriores operaciones sobre la nueva tabla, omita su nombre. Al predicado de selección se le conoce como **condición theta**. La letra griega theta, θ, con frecuencia se usa en matemáticas para representar cualquier tipo de operador. Los desarrolladores de la teoría relacional la usaron para representar predicados que involucren algunos de los operadores de comparación, a saber,

$<, \leq, >, \geq, =, \neq, \wedge$ (AND), $\vee$ (OR), $\neg$ (NOT)

Se pueden formar predicados de selección compleja con el uso de más de un operador de comparación en la condición θ. Por ejemplo, para encontrar todas las especializaciones matemáticas que tengan más de 30 créditos, se escribe:

```
SELECT Student WHERE major = 'Math' AND credits> 30
```

o simbólicamente

$\sigma_{major='Math' \wedge credits > 30}(Student)$

Este comando produce la siguiente tabla sin nombre:

stuId	lastName	firstName	major	credits
S1002	Chin	Ann	Math	36
S1015	Jones	Mary	Math	42

El operador PROJECT, Π

El comando PROJECT también opera sobre una sola tabla, pero produce un subconjunto vertical de la tabla, extrae los valores de columnas específicas, elimina duplicados y coloca los valores en una nueva tabla. De manera informal, su forma es

PROJECT *nombreTabla* OVER (*nombreCol, . . . , nombreCol*) [GIVING *nombreTablaNueva*]

o, simbólicamente,

[*nombreTablaNueva* =] $\Pi_{nombreCol, \ldots, nombreCol}(nombreTabla)$

Para ilustrar la proyección sobre una sola columna, encuentre todas las diferentes especialidades que los estudiantes tienen mediante el comando

```
PROJECT Student OVER major GIVING Temp
```

o Temp = Π_{major}(Student)

La tabla resultante, `Temp`, se parece a esto:

Temp	major
	History
	Math
	Art
	CSC

Note que se obtienen todos los valores que aparecen en la columna `major` de la tabla `Student`, pero se eliminan los duplicados.

Cuando se proyecta sobre dos o más columnas, se eliminan los duplicados de las combinaciones de valores. Por ejemplo, suponga que quiere encontrar el salón donde enseña cada docente. Podría hacerlo al usar el comando

```
PROJECT Class OVER (facd,room)
```

o

$\Pi_{facId,room}$ (Class)

Esto da el resultado en una tabla sin nombre,

facId	room
F101	H221
F105	M110
F115	H221
F110	H225

Aunque se tiene repetición en los valores del salón, pues H221 aparece dos veces, note que los valores de salón repetidos aparecen con diferentes valores `facId`. En tanto la combinación no aparezca antes, se agrega a la proyección. Note que la tercera y última filas de `Class` no contribuyen al resultado de la proyección, pues su combinación de valores para `facId` y `room` apareció anteriormente.

Es posible combinar las operaciones SELECT y PROJECT para obtener sólo columnas específicas de ciertas filas, pero hacerlo requiere dos pasos. Por ejemplo, suponga que quiere ver los nombres e ID de todas las especialidades en historia. Dado que sólo se quieren especialidades en historia, es necesario un SELECT, pero, como se quieren sólo ciertas columnas de dichos registros, se necesita un PROJECT. La consulta se puede expresar como

```
SELECT Student WHERE major = 'History' GIVING Temp
PROJECT Temp OVER (lastName, firstName, stuId) GIVING Result
```

Después de ejecutar el primer comando, se tiene la tabla

Temp:

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1005	Lee	Perry	History	3

El segundo comando se realiza sobre esta tabla temporal, y el resultado es

Result:

lastname	firstname	stuid
Smith	Tom	S1001
Lee	Perry	S1005

Note que la operación PROJECT permitió invertir el orden de las dos columnas en el resultado final. Pudo haber escrito los comandos simbólicamente como

$$\Pi_{\texttt{lastName,firstName,stuId}}(\sigma_{\texttt{major='History'}}(\texttt{Student}))$$

Note que se pueden componer las operaciones usando el resultado de la primera como el argumento de la segunda. La tabla intermedia que resulta de la operación de selección, que se llamó `Temp` cuando se usó la sintaxis inglesa, no necesita un nombre cuando se usa notación simbólica, pues la expresión se usa como un operando. Observe que no podría invertir el orden de SELECT y PROJECT. Si hubiera hecho PROJECT primero, tendría una tabla intermedia sólo con las columnas `lastName`, `firstName` y `stuId`, así que luego no podría intentar hacer una SELECT sobre major en la tabla intermedia, pues no contiene dicho atributo.

Producto y combinaciones (joins): combinación theta, equicombinación, combinación natural, semicombinación y combinación exterior Dadas dos tablas, A y B, se puede formar su producto, A POR B o A $\times$ B, en forma muy parecida a como se formó el producto cartesiano de los conjuntos en la sección 4.3.2. A $\times$ B es una tabla formada al concatenar todas las filas de A con todas las filas de B. Sus columnas son las columnas de A seguidas por las columnas de B, y su ancho es el ancho de A más el ancho de B. Se puede producir en varias formas. Un método es usar un algoritmo de "bucle anidado". Comience con la primera fila de A, combínela con la primera fila de B, luego con la segunda fila de B y así por el estilo, hasta que se hayan formado todas las combinaciones de la primera fila de A con todas

las filas de B. Luego el procedimiento se repite para la segunda fila de A, luego la tercera fila, etc. Si A tiene x filas y B tiene y filas, entonces A POR B, escrito A × B, tiene x*y filas.

Suponga que se forma el producto de `Student` y `Enroll`, escrito `Student` × `Enroll`. Esta tabla tendrá siete columnas, pero dos de ellas se llamarán `stuId`. Para distinguir estas dos use los nombres calificados de las tablas originales, `Student.stuId` y `Enroll.stuId`. El producto, que tiene 63 filas, se muestra en la figura 4.2.

FIGURA 4.2
Student X Enroll

Student X Enroll

Student.stuId	lastname	firstName	major	Enroll.stuId	classNo	grade
S1001	Smith	Tom	History	S1002	ART103A	D
S1001	Smith	Tom	History	S1002	MTH103C	B
S1001	Smith	Tom	History	S1010	ART103A	
S1001	Smith	Tom	History	S1020	MTH101B	A
S1001	Smith	Tom	History	S1001	HST205A	C
S1001	Smith	Tom	History	S1002	CSC201A	F
S1001	Smith	Tom	History	S1010	MTH103C	
S1001	Smith	Tom	History	S1001	ART103A	A
S1001	Smith	Tom	History	S1020	CSC201A	B
S1002	Chin	Ann	Math	S1002	MTH103C	B
S1002	Chin	Ann	Math	S1010	ART103A	
S1002	Chin	Ann	Math	S1002	CSC201A	F
S1002	Chin	Ann	Math	S1010	MTH103C	
S1002	Chin	Ann	Math	S1002	ART103A	D
S1002	Chin	Ann	Math	S1001	HST205A	C
S1002	Chin	Ann	Math	S1001	ART103A	A
S1002	Chin	Ann	Math	S1020	MTH101B	A
S1002	Chin	Ann	Math	S1020	CSC201A	B
S1005	Lee	Perry	History	S1010	MTH103C	
S1005	Lee	Perry	History	S1001	HST205A	C
S1005	Lee	Perry	History	S1002	MTH103C	B
S1005	Lee	Perry	History	S1020	CSC201A	B
S1005	Lee	Perry	History	S1002	ART103A	D
S1005	Lee	Perry	History	S1010	ART103A	
S1005	Lee	Perry	History	S1020	MTH101B	A
S1005	Lee	Perry	History	S1001	ART103A	A
S1005	Lee	Perry	History	S1002	CSC201A	F
S1010	Burns	Edward	Art	S1002	ART103A	D
S1010	Burns	Edward	Art	S1001	HST205A	C
S1010	Burns	Edward	Art	S1002	CSC201A	F
S1010	Burns	Edward	Art	S1001	ART103A	A

Student X Enroll

Student.stuId	lastname	firstName	major	Enroll.stuId	classNo	grade
S1010	Burns	Edward	Art	S1002	MTH103C	B
S1010	Burns	Edward	Art	S1010	MTH103C	
S1010	Burns	Edward	Art	S1020	MTH101B	A
S1010	Burns	Edward	Art	S1020	CSC201A	B
S1010	Burns	Edward	Art	S1010	ART103A	
S1013	McCarthy	Owen	Math	S1010	ART103A	
S1013	McCarthy	Owen	Math	S1002	ART103A	D
S1013	McCarthy	Owen	Math	S1001	HST205A	C
S1013	McCarthy	Owen	Math	S1002	MTH103C	B
S1013	McCarthy	Owen	Math	S1020	CSC201A	B
S1013	McCarthy	Owen	Math	S1010	MTH103C	
S1013	McCarthy	Owen	Math	S1002	CSC201A	F
S1013	McCarthy	Owen	Math	S1001	ART103A	A
S1013	McCarthy	Owen	Math	S1020	MTH101B	A
S1015	Jones	Mary	Math	S1001	ART103A	A
S1015	Jones	Mary	Math	S1020	MTH101B	A
S1015	Jones	Mary	Math	S1001	HST205A	C
S1015	Jones	Mary	Math	S1020	CSC201A	B
S1015	Jones	Mary	Math	S1010	MTH103C	
S1015	Jones	Mary	Math	S1002	ART103A	D
S1015	Jones	Mary	Math	S1010	ART103A	
S1015	Jones	Mary	Math	S1002	CSC201A	F
S1015	Jones	Mary	Math	S1002	MTH103C	B
S1020	Rivera	Jane	CSC	S1001	HST205A	C
S1020	Rivera	Jane	CSC	S1002	ART103A	D
S1020	Rivera	Jane	CSC	S1010	ART103A	
S1020	Rivera	Jane	CSC	S1020	CSC201A	B
S1020	Rivera	Jane	CSC	S1001	ART103A	A
S1020	Rivera	Jane	CSC	S1002	CSC201A	F
S1020	Rivera	Jane	CSC	S1002	MTH103C	B
S1020	Rivera	Jane	CSC	S1020	MTH101B	A
S1020	Rivera	Jane	CSC	S1010	MTH103C	

Con base en el producto de tablas se pueden definir varias operaciones. La más general se llama **COMBINACIÓN THETA** (theta join). La combinación theta se define como el resultado de realizar una operación SELECT sobre el producto. Por ejemplo, es posible que quie-

ra sólo las tuplas del producto donde el valor credits sea mayor que 50. Aquí, theta es >, y la consulta se escribiría como

```
Student TIMES Enroll WHERE credits > 50
```

Esto es equivalente a

```
Student TIMES Enroll GIVING Temp
SELECT Temp WHERE credits > 50
```

o simbólicamente

$$\sigma_{credits>50}\ (Student \times Enroll)$$

A veces se usa el símbolo X_θ para representar la combinación theta. Note que, para cualesquiera dos relaciones, X y Y, la combinación theta se define simbólicamente como

$$A\ X_\theta\ B = \sigma_\theta\ (A \times B)$$

Dado que el producto es una operación lenta que requiere, en el ejemplo anterior, 63 concatenaciones, sería más eficiente realizar primero la selección y luego el producto, siempre que la secuencia de operaciones produzca el mismo resultado. Una forma más eficiente, que requiere sólo 18 concatenaciones, es:

```
SELECT Student WHERE credits > 50 GIVING Temp2
Temp2 TIMES Enroll
```

o,

$$(\sigma_{credits>50}\ (Student)) \times Enroll$$

Aunque el producto es una operación válida, raramente se estaría interesado en formar concatenaciones de filas de `Student` con filas de `Enroll` que tengan un `stuId` diferente. Una operación más común que involucra el producto de tablas es aquella en donde se piden sólo aquellas filas del producto en las que los valores de las columnas comunes sean iguales. Cuando la theta es la igualdad en las columnas comunes, se tiene la **EQUIJOIN** (equicombinación) de tablas. Para formar la equicombinación, entonces, comience con dos tablas que tengan una o más columnas en común. Compare cada tupla de la primera con cada tupla de la segunda y elija sólo aquellas concatenaciones en las que los valores en las columnas comunes sean iguales. La equicombinación de `Student` y `Enroll` se formaría al elegir las tuplas del producto con valores `stuId` coincidentes. La figura 4.3 muestra:

```
Student EQUIJOIN Enroll
```

que se escribe simbólicamente como

$$Student\ x_{Student.stuId=Enroll.stuId}\ Enroll$$

Note que esto es equivalente a

```
Student TIMES Enroll GIVING Temp3
SELECT Temp3 WHERE Student.stuId = Enroll.stuId
```

o

$$\sigma\ Student.stuId=Enroll.stuId\ (Student \times Enroll)$$

Si se tuviera más de una columna común, ambos conjuntos de valores tendrían que coincidir.

Usted puede notar que, por definición, en una equicombinación, siempre se tienen al menos dos columnas idénticas. Puesto que parece innecesario incluir la columna repetida, se le puede eliminar y definir una **NATURAL JOIN** (combinación natural) como una equicombinación en la que se elimina la columna repetida. Ésta es la forma más común de la operación JOIN (combinación), tan común, de hecho, que esto es lo que usualmente se entiende por JOIN. Cuando se pretende la combinación natural, simplemente se escribe:

nombreTabla1 JOIN *nombreTabla2* [GIVING *nombreTablaNueva*]

Student Equijoin Enroll

Student.stuId	lastName	firstName	major	credits	Enroll.stuId	classNo	grade
S1001	Smith	Tom	History	90	S1001	HST205A	C
S1001	Smith	Tom	History	90	S1001	ART103A	A
S1002	Chin	Ann	Math	36	S1002	MTH103C	B
S1002	Chin	Ann	Math	36	S1002	CSC201A	F
S1002	Chin	Ann	Math	36	S1002	ART103A	D
S1010	Burns	Edward	Art	63	S1010	MTH103C	
S1010	Burns	Edward	Art	63	S1010	ART103A	
S1020	Rivera	Jane	CSC	15	S1020	MTH101B	A
S1020	Rivera	Jane	CSC	15	S1020	CSC201A	B

FIGURA 4.3

EQUICOMBINACIÓN de Student y Enroll

Puede usar el símbolo |x| para la combinación natural, como en

```
nombreTabla1 |x| nombreTabla2
```

La combinación natural de `Student` y `Enroll` produciría una tabla idéntica a la de la figura 4.3, excepto que la segunda columna `stuId` se eliminaría. Dado que `Faculty` y `Class` tienen una columna común, `facId`, se encontrará su combinación natural. El resultado del comando

```
Faculty JOIN Class
```

o

```
Faculty |x| Class
```

se muestra en la figura 4.4. La tabla resultante da todos los detalles acerca de los miembros del personal docente y las clases que imparten. La combinación permite recombinar trozos de información acerca de una entidad, aun cuando aparezcan en diferentes tablas. Los datos acerca del personal docente y las clases no se mantienen en una sola tabla, porque se tendrían muchas repeticiones en la tabla. Note que los datos acerca de Tanaka y Byrne se repiten, porque cada uno imparte dos clases.

Las consultas más complicadas pueden requerir el uso de los comandos SELECT, PROJECT y JOIN. Por ejemplo, suponga que quiere encontrar las clases y calificaciones de la estudiante Ann Chin. Note que la tabla `Student` contiene el nombre de la estudiante, pero no las

Faculty Natural Join Class

facId	name	department	rank	classNo	schedule	room
F101	Adams	Art	Professor	ART103A	MWF9	H221
F105	Tanaka	CSC	Instructor	CSC203A	MThf12	M110
F105	Tanaka	CSC	Instructor	CSC201A	TuTHF10	M110
F110	Byrne	Math	Assistant	MTH103C	MWF11	H225
F110	Byrne	Math	Assistant	MTH101B	MTuTh9	H225
F115	Smith	History	Associate	HST205A	MWF11	H221

FIGURA 4.4

Faculty |x| Class

Combinación natural (|x|) de Faculty y Class

clases o calificaciones, mientras que la tabla `Enroll` contiene las clases y calificaciones, pero no el nombre. Siempre que necesite usar más de una tabla para responder una consulta, debe usar una combinación o producto. Si el resultado necesita contener sólo algunas columnas de las tablas originales, pero no todas, también se necesita un PROJECT. Si no usará todas las filas, necesita también un SELECT. Para decidir cuáles operaciones realizar y en qué orden, examine las tablas para ver cómo respondería la pregunta "a mano" y luego intente formular las operaciones requeridas en términos de comandos de álgebra relacional. Aquí, comenzaría con la tabla `Student`, y localizaría el registro Ann Chin (una operación SELECT). Luego consultaría la tabla `Enroll`, buscaría los registros con `stuId` que coincida con el `stuId` de Ann Chin (una operación JOIN) y leería sólo los valores `classNo` y `grade` (una operación PROJECT). Una forma de encontrar los datos requeridos es la siguiente, expresada con el lenguaje parecido al inglés:

```
SELECT Student WHERE lastName='Chin' AND firstName ='Ann' GIVING Temp1
Temp1 JOIN Enroll GIVING Temp2
PROJECT Temp2 OVER (classNo, grade) GIVING Answer
```

o simbólicamente como

$$\Pi_{classNo,grade}((\sigma_{lastName='Chin' \wedge firstName='Ann'}(Student))|x|Enroll)$$

Después de SELECT en la primera línea se tiene este resultado:

Temp1:

stuid	lastname	firstname	major	credits
S1002	Chin	Ann	Math	36

Después de JOIN en la segunda línea se tiene este resultado:

Temp2:

stuId	lastName	firstName	major	credits	classNo	grade
S1002	Chin	Ann	Math	36	ART103A	D
S1002	Chin	Ann	Math	36	CSC201A	F
S1002	Chin	Ann	Math	36	MTH103C	B

Después de PROJECT en la tercera línea se tiene este resultado:

Answer:

classNo	grade
CSC201A	F
ART103A	D
MTH103C	B

Es posible que haya observado que todo lo realmente necesario de `Temp1` era la columna `stuId`, pues ésta se usó para la comparación de la combinación, y ninguna de las otras columnas se utilizó más tarde. Por tanto, si lo prefiere, podría usar un PROJECT en `Temp1` para obtener sólo la columna `stuId` antes de realizar la JOIN, como en

$$\Pi_{classNo,grade}(\Pi_{stuId}\ (\sigma_{lastName='Chin' \wedge firstName='Ann'}(Student))\ |x|Enroll)$$

Una tercera forma de realizar la consulta es

```
JOIN Student, Enroll GIVING Tempa
SELECT Tempa WHERE lastName='Chin' AND firstName = 'Ann' GIVING Tempb
PROJECT Tempb OVER (classNo, grade)
```

o

$$\Pi_{classNo,grade}(\sigma_{lastName='Chin' \wedge firstName='Ann'}(Student|x|Enroll))$$

Como puede observar, este método es menos eficiente, pues la primera línea requiere 54 concatenaciones para formar una tabla combinada, una operación relativamente lenta.

Hacer la selección primero reduce el número de comparaciones a nueve, lo que por tanto optimiza la consulta.

Es posible hacer combinaciones de tablas combinadas. Suponga que quiere encontrar las ID de todos los estudiantes en las clases del profesor Adam. Necesitaría datos de las tablas `Faculty`, `Class` y `Enroll` para responder esta consulta. Un método es

```
SELECT Faculty WHERE name = 'Adams' GIVING Temp1
Temp1 JOIN Class GIVING Temp2
Temp2 JOIN Enroll GIVING Temp3
PROJECT Temp3 OVER stuId GIVING Result
```

o simbólicamente

$$\Pi_{stuId}(((\sigma_{name='Adams'}(Faculty)) \mid x \mid Class) \mid x \mid Enroll)$$

En este ejemplo, `Temp2` tiene siete columnas y `Temp3` tiene nueve. Dado que sólo una de las columnas de `Faculty`, `facId`, se usa para la combinación y ninguna de sus otras columnas se necesita de modo posterior, se podría hacer un PROJECT antes de la primera combinación. Alternativamente, dado que sólo una de las columnas de `Temp2`, `classNo`, se necesita para la combinación y ninguna de sus otras columnas se usa de nuevo, podría hacer un PROJECT entre las dos combinaciones.

Se pueden definir muchos otros tipos de operadores combinación. Una variación es la **SEMIJOIN** (semicombinación) de dos tablas. Si A y B son tablas, entonces la semicombinación izquierda (left-semijoin) A |x B se encuentra al tomar la combinación natural de A y B y luego proyectar el resultado en los atributos de A. El resultado será justo aquellas tuplas de A que participan en la combinación. Para las tablas `Student` y `Enroll` que se muestran en la figura 4.1, la semicombinación izquierda de `Student` por `Enroll`, escrita como

```
Student LEFT-SEMIJOIN Enroll
```

o simbólicamente

```
Student |x Enroll
```

se muestra en la figura 4.5. Note que la semicombinación no es conmutativa. Por ejemplo, Student LEFT SEMIJOIN Enroll es diferente de Enroll LEFT SEMIJOIN Student. Además de la semicombinación izquierda se puede definir la semicombinación derecha (right-semijoin) de A y B, A x| B, que se encuentra al tomar la combinación natural de A y B y luego proyectar el resultado en los atributos de B. Para Student x| Enroll, ésta es la proyección en la tabla `Enroll` de la combinación natural, esto es, de aquellas tuplas de `Enroll` que participan en la combinación (a saber, todas ellas). Sus columnas son `stuId`, `classNo` y `grade`, y tiene las mismas tuplas que la tabla `Enroll`.

Otro tipo de operación combinación es la OUTERJOIN (combinación exterior). Esta operación es una extensión de una operación THETA JOIN, una EQUIJOIN o una NATURAL JOIN. Cuando se forma cualquiera de estas combinaciones, cualquier tupla de una de las tablas originales para las que no hay coincidencia en la segunda tabla no entra al resultado.

Student LEFT-SEMIJOIN Enroll

stuId	lastName	firstName	major
S1001	Smith	Tom	History
S1002	Chin	Ann	Math
S1010	Burns	Edward	Art
S1020	Rivera	Jane	CSC

FIGURA 4.5
SEMICOMBINACIÓN-IZQUIERDA de Student y Enroll

Por ejemplo, en una equicombinación para tablas con una columna común, una fila en la primera tabla no participa en el resultado a menos que exista una fila en la segunda tabla con el mismo valor para la columna común. Se vio, por ejemplo, que la fila "S1015 Jones Mary Math 42" de la tabla `Student` no se representó en la tabla `Student EQUIJOIN Enroll`, figura 4.3, porque no hay una fila de `Enroll` que tuviera S1015 como el valor de `stuId`. Ni había filas para S1005 ni S1013. En una combinación exterior aparecen tales filas sin coincidencia, con valores nulos para los atributos que la otra tabla aporta al resultado. Por ejemplo, forme la combinación exterior de `Student` y `Faculty` donde compare `Student.lastName` con `Faculty.name`. Como se muestra en la figura 4.6(a), se incluyen todas las filas en Student EQUIJOIN Faculty (esto es, todas las tuplas estudiante con las tuplas de personal docente que tengan el mismo apellido), y se agregan en las filas de `Student` que no tengan filas `Faculty` coincidentes, y coloca valores nulos en las columnas `facId`, `name`, `department` y `rank`. También se incluye cualquier fila de `Faculty` para la cual el valor de nombre no tenga una coincidencia en la tabla `Student`.

Una variación de la equicombinación exterior mostrada es una LEFT-OUTER-EQUIJOIN (equicombinación-exterior-izquierda), lo que significa que sólo las filas sin coincidencia de

FIGURA 4.6
Combinaciones exteriores

FIGURA 4.6(a)
EQUICOMBINACIÓN-EXTERIOR de Student y Faculty

Student OUTER-EQUIJOIN Faculty

stuId	lastName	firstName	major	facId	name	department	rank
S1001	Smith	Tom	History	F221	Smith	CSC	Professor
S1001	Smith	Tom	History	F115	Smith	History	Associate
S1002	Chin	Ann	Math				
S1005	Lee	Perry	History				
S1010	Burns	Edward	Art				
S1013	McCarthy	Owen	Math				
S1015	Jones	Mary	Math				
S1020	Rivera	Jane	CSC				
				F101	Adams	Art	Professor
				F105	Tanaka	CSC	Instructor
				F110	Byrne	Math	Assistant

FIGURA 4.6(b)
EQUICOMBINACIÓN-EXTERIOR-IZQUIERDA de Student y Faculty

Student LEFT-OUTER-EQUIJOIN Faculty

stuId	lastName	firstName	major	facId	name	department	rank
S1001	Smith	Tom	History	F221	Smith	CSC	Professor
S1001	Smith	Tom	History	F115	Smith	History	Associate
S1002	Chin	Ann	Math				
S1005	Lee	Perry	History				
S1010	Burns	Edward	Art				
S1013	McCarthy	Owen	Math				
S1015	Jones	Mary	Math				
S1020	Rivera	Jane	CSC				

Student RIGHT-OUTER-EQUIJOIN Faculty

stuId	lastName	firstName	major	facId	name	department	rank
S1001	Smith	Tom	History	F221	Smith	CSC	Professor
S1001	Smith	Tom	History	F115	Smith	History	Associate
				F101	Adams	Art	Professor
				F105	Tanaka	CSC	Instructor
				F110	Byrne	Math	Assistant

FIGURA 4.6(c)
EQUICOMBINACIÓN-EXTERIOR-DERECHA de Student y Faculty

la primera tabla (izquierda) aparecen en el resultado. La equicombinación-exterior-izquierda de `Student` y `Faculty` se muestra en la figura 4.6(b). En una RIGHT-OUTER-EQUIJOIN (equicombinación-exterior-derecha) se incluyen las filas sin coincidencia de la segunda tabla (derecha), como se muestra para `Student` y `Faculty` en la figura 4.6(c). También se puede definir la combinación theta exterior general, la combinación theta exterior izquierda y la combinación theta exterior derecha en una forma similar.

La combinación natural exterior es similar a la equicombinación exterior, excepto que se eliminan las columnas repetidas, como es usual para una combinación natural. Si las filas combinadas tienen un valor no nulo igual para una columna repetida, se usa dicho valor en la columna común. Si ambas tienen valores nulos, se usa uno nulo, y si una tiene un valor nulo y la otra no, se usa el valor no nulo en el resultado.

División

La división es una operación binaria que se puede definir sobre dos relaciones donde toda la estructura de una (el divisor) es una porción de la estructura de la otra (el dividendo). La operación dice cuáles valores en los atributos que aparecen sólo en el dividendo aparecen con todas las filas del divisor. Un ejemplo se muestra en la figura 4.7. Aquí, la estructura de la tabla `Stu`, que se muestra en la figura 4.7(b), está contenida dentro de la estructura de la tabla `Club`, que se muestra en la figura 4.7(a), de modo que se puede dividir `Club` por `Stu`. Los resultados, que se muestran en la figura 4.7(c) serán aquellos valores de `ClubName` que aparezcan con cada valor de `StuNumber` y `StuLastName`, esto es, los nombres de los clubes a los que pertenecen todos los estudiantes.

Club

ClubName	StuNumber	StuLastName
Computing	S1001	Smith
Computing	S1002	Chin
Drama	S1001	Smith
Drama	S1002	Chin
Drama	S1005	Lee
Karate	S1001	Smith
Karate	S1002	Chin
Karate	S1005	Lee

FIGURA 4.7
Tablas de división

FIGURA 4.7(a)
La tabla Club

FIGURA 4.7(b)
La tabla Stu

Stu	
StuNumber	**StuLastName**
S1001	Smith
S1002	Chin
S1005	Lee

FIGURA 4.7(c)
Club ÷ Stu

Club DividedBy Stu
ClubName
Drama
Karate

Note que esta división es equivalente a las siguientes operaciones:

```
PROJECT Club OVER (ClubName) GIVING Temp1
Temp1 TIMES Stu GIVING Temp2
Temp2 MINUS Club GIVING Temp3
PROJECT Temp3 OVER ClubName GIVING Temp4
Temp1 MINUS Temp4 GIVING Quotient
```

Operaciones de conjuntos: unión, diferencia, intersección

Dado que las relaciones básicamente son conjuntos de *n*-tuplas, el álgebra relacional incluye una versión de las operaciones de conjunto básicas: unión, intersección y diferencia de conjuntos. Para que estas operaciones binarias sean posibles, las dos relaciones sobre las que se realizan deben ser **compatibles en unión**. Esto significa que es posible realizar una operación unión dado que tienen la misma estructura básica. En particular, deben tener el mismo grado y atributos en la posición correspondiente y ambas relaciones deben tener el mismo dominio. Por ejemplo, la tercera columna en la primera tabla debe tener el mismo dominio que la tercera columna en la segunda tabla, aunque los nombres de columna podrían ser diferentes. El resultado de cada una de las operaciones de conjunto es una nueva tabla con la misma estructura que las dos tablas originales. Las cuatro tablas con las que ha trabajado tienen todas distintas estructuras, de modo que ningún par de ellas es compatible en unión. Por tanto, se usarán las dos tablas de la figura 4.8(a) para operaciones de conjuntos. Suponga que la tabla MainFac contiene registros de miembros del personal docente que imparten en el mismo campus, mientras que la tabla BranchFac contiene registros de quienes imparten en el campus alterno de la universidad. Algunos miembros del personal docente dan clases en ambas ubicaciones.

La unión de dos relaciones es el conjunto de tuplas en cualquiera o en ambas relaciones. Por ejemplo, puede encontrar la unión de `MainFac` y `BranchFac` del modo siguiente:

```
MainFac UNION BranchFac
```

o simbólicamente

```
MainFac ∪ BranchFac
```

El resultado se muestra en la figura 4.8(b).

La intersección de dos relaciones es el conjunto de tuplas en ambas relaciones simultáneamente. La intersección de `MainFac` y `BranchFac`,

```
MainFac INTERSECTION BranchFac
```

MainFac

FacID	name	department	rank
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F221	Smith	CSC	Professor

BranchFac

FacId	name	department	rank
F101	Adams	Art	Professor
F110	Byre	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

FIGURA 4.8
Las operaciones de conjuntos

FIGURA 4.8(a)
Relaciones compatibles en unión MainFac y BranchFac

MainFac UNION BranchFac

FacId	name	department	rank
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

FIGURA 4.8(b)
UNIÓN de MainFac y BranchFac

MainFac INTERSECTION BranchFac

FacId	name	department	rank
F101	Adams	Art	Professor
F221	Smith	CSC	Professor

FIGURA 4.8(c)
INTERSECCIÓN de MainFac y BranchFac

MainFac MINUS BranchFac

FacId	name	department	rank
F105	Tanaka	CSC	Instructor

FIGURA 4.8(d)
DIFERENCIA de MainFac y BranchFac

o simbólicamente

```
MainFac ∩ BranchFac
```

se muestra en la figura 4.8(c).

La diferencia entre dos relaciones es el conjunto de tuplas que pertenecen a la primera relación pero no a la segunda. Por tanto,

```
MainFac MINUS BranchFac
```

o simbólicamente

```
MainFac - BranchFac
```

es la tabla que se muestra en la figura 4.8(d).

Existen muchas extensiones del álgebra relacional. A los operadores estándar se han agregado métodos para tratar los valores nulos de manera sistemática, y varios investigadores han definido funciones agregadas como SUM, AVG, MAX, MIN y COUNT.

4.6.3 Cálculo relacional

El **cálculo relacional** es un lenguaje de manipulación de datos relacionales formal no procedural, en el que el usuario simplemente especifica cuáles datos se deben recuperar, pero no cómo recuperarlos. Es un estándar alternativo para los lenguajes de manipulación de datos relacionales. Esta sección sólo presenta una breve discusión con propósitos de comparación, y no tiene la intención de ser un tratamiento completo del lenguaje. El cálculo relacional no se relaciona con los familiares cálculos diferencial e integral de matemáticas, pero usa una rama de la lógica simbólica llamada **cálculo de predicados**. Cuando se aplica a las bases de datos, viene en dos formas: cálculo relacional **orientado a tuplas** y cálculo relacional **orientado a dominio**. Ambos usan conceptos de la lógica simbólica.

En lógica, un **predicado** es una oración declarativa que puede ser verdadera o falsa. Por ejemplo, las oraciones "Mary Jones es estudiante" y "Mary Jones tiene 500 créditos" son predicados, pues se puede decidir si son verdaderos o falsos. Por otra parte, "¡Vaya día!" no es un predicado. Si un predicado contiene una variable, como en "x es estudiante", debe haber un conjunto de sustitución asociado o **rango** para x. Cuando algunos valores del rango se sustituyen por x, el predicado puede ser verdadero; para otros valores puede ser falso. Por ejemplo, si el rango es el conjunto de todas las personas, y x se sustituye con Mary Jones, el predicado resultante, "Mary Jones es estudiante", es verdadero. Si x se sustituye con el profesor Adams, el predicado probablemente sea falso.

Si se usa P para representar un predicado, entonces {x|P(x)} significa el conjunto de todos los valores x tales que P es verdadero. Podría usar Q, R o cualquiera otra letra para el predicado. De igual modo, podría usar y, z, w o cualquiera otra letra para representar la variable, de modo que

```
{t|P(t)}, {s|Q(s)}, {x|R(x)}
```

significan el conjunto de valores en el conjunto de sustitución para los que el predicado correspondiente es verdadero.

Los predicados se pueden conectar mediante los conectivos lógicos AND (∧), OR (∨) y NOT (¬) para formar **predicados compuestos** como:

```
P(x)AND Q(x)     P(x) OR Q(x)   NOT(P(x)) OR Q(x)
```

que se pueden escribir como

```
P(x) ∧ Q(x)      P(x) ∨ Q(x)    ¬P(x) ∨ Q(x)
```

Una **conjunción** consiste en predicados conectados con AND, una **disyunción** consiste en predicados conectados con OR y una negación es un predicado precedido por un NOT. En lógica existen dos **cuantificadores** usados con predicados para decir a cuántas instancias se aplica el predicado. El cuantificador **existencial**, **EXISTS** (existe), significa precisamente eso: "existe". Se usa en **aseveraciones**, o enunciados que deben ser ciertos para al menos una instancia, como:

```
EXISTS x (P(x)}
```

"Existe al menos un valor variable x tal que P(x) es verdadero." A veces se usa el símbolo ∃ para "existe", de modo que la aseveración se podría escribir como

```
∃ x (P(x))
```

El cuantificador **universal**, **FORALL** (para todo), significa "para todo". En ocasiones se escribe ∀. Se usa en aseveraciones acerca de toda instancia, como:

```
FORALL s(P(s)) o ∀ s(P(s))
```

que significa que P(s) es verdadero para todos los valores, s, en el rango. Una variable sin cuantificador (∃ o ∀) se llama **variable libre**, y una con cuantificador se llama **variable acotada**.

Cálculo relacional orientado a tupla En el cálculo relacional orientado a tupla se usan **variables tupla**, que son variables que toman las tuplas de alguna relación o relaciones como valores. Intuitivamente, una **consulta** (query) se expresa en una forma como:

```
{S | P(S)}
```

que significa "Encontrar el conjunto de todas las tuplas, por decir s, tales que `P(S)` sea verdadera cuando S=s". Aquí, S es la variable tupla y `P(S)` es una **fórmula** que describe S. Dado que S representa una tupla de una relación, a un atributo de la tupla se le puede referir mediante la notación punto. Por ejemplo:

```
{S | S ∈ Student ∧ S.credits > 50}
```

significa "Encontrar todos los estudiantes que tengan más de 50 créditos". De igual modo,

```
{S.stuId | S ∈ Student ∧ S.major='History' ∧ S.credits < 30}
```

significa "Encontrar el stuId de todos los estudiantes que tengan especialidad en historia con menos de 30 créditos". Estas consultas se evalúan al instanciar la variable tupla, S, con cada tupla, s, de su rango, `Student`, a su vez, y poner a prueba el predicado para dicha tupla. Las tuplas para las que el predicado sea verdadero forman el resultado de la consulta.

Ahora se definirá de manera más formal el concepto de fórmula. Comience con la definición de una **fórmula atómica** o **átomo**. Sean s, t representaciones de variables tupla que tienen atributos `a` y `b`, respectivamente; sea θ la representación de los operadores de comparación ($<$, $<=$, $>$, $>=$, $=$, $\neq$), y sea `r` la representación de una relación. Una fórmula atómica es aquella que tiene una de las siguientes formas:

- $S \in \mathbf{r}$
- S.a θ R.b
- S.a θ *constante*

Las **fórmulas** se construyen de manera recursiva a partir de fórmulas atómicas usando estas reglas:

- Una fórmula atómica es una fórmula
- Si F_1 y F_2 son fórmulas, entonces también lo son $F_1 \wedge F_2$, $F_1 \vee F_2$ y $\neg F_1$

Si S es una variable tupla que aparece como variable libre en una fórmula F, entonces

- $\exists$ S (**F**(S)) es una fórmula y
- $\forall$ S(**F**(S)) es una fórmula

Una expresión del cálculo relacional de tuplas, como las consultas que se mostraron anteriormente, tiene la forma

$$\{S_1.a_1, S_2.a_2, \ldots, S_n.a_n \mid F(S_1, S_2, \ldots, S_n)\}$$

donde S_1 son las variables tuplas que tienen rangos que son las relaciones $S_1, S_2, \ldots, S_n$ y los a_1 son atributos de dichas relaciones. F es una fórmula, como se definió anteriormente.

Ejemplos de cálculo relacional orientado a tuplas

- Ejemplo 1. Encuentre los nombres de todos los miembros del personal docente que sean profesores en el departamento CSC.

  ```
  {F.name | F ∈ Faculty ∧ F.department = 'CSC' ∧ F.rank ='Professor'}
  ```

- Ejemplo 2. Encuentre los apellidos de todos los estudiantes inscritos en CSC201A.

  ```
  {S.lastName | S ∈ Student ∧ EXISTS E (E ∈ Enroll ∧ E.stuId =
  S.stuId ∧ E.classNo = 'CSC201A')}
  ```

- Ejemplo 3. Encuentre los nombres y apellidos de todos los estudiantes inscritos en al menos una clase que se imparta en el salón H221.

```
{S.firstName, S.lastName | S ∈ Student ∧ EXISTS E (E ∈ Enroll
∧ E.stuId = S.stuId ∧ EXISTS C (C ∈ Class ∧C.classNo = E.classNo
∧ C.room = 'H221'))}
```

- Ejemplo 4. Encuentre los nombres de los docentes que estén en el departamento CSC pero no impartan CSC201A.

```
{F.name | F ∈ Faculty ∧ F.department= 'CSC' ∧ ¬ EXISTS C (C ∈ Class
∧ C.facId = F.facId ∧ C.classNo = 'CSC201A')}
```

- Ejemplo 5. Encuentre los apellidos de los estudiantes inscritos en todas las clases ofrecidas.

```
{S.lastName | S ∈ Student ∧ C ∈ Class ∧ FORALL C (EXISTS E (E ∈ Enroll ∧ E.stuId =
S.stuId ∧ E.classNo = C.classNo))}
```

Al crear una expresión en cálculo relacional de tuplas es importante que sea **segura**, esto es, las condiciones a poner a prueba se restrinjan a un conjunto finito de posibilidades, de modo que no se pierda tiempo intentando poner a prueba un predicado que no se pueda probar en una cantidad finita de tiempo. Por ejemplo, la expresión

```
{S | ¬ (S ∈ Student)}
```

no es segura, pues existe un número infinito de posibles tuplas que no están en Student. Para evitar tales consultas se define el **dominio** de una expresión como el conjunto de todos los valores que o aparecen explícitamente en la expresión (por ejemplo, constantes como 'H221') o que aparecen en una o más de las relaciones que se presentan en la expresión. Una expresión es segura si todos los valores en su resultado están en su dominio. En el texto, las consultas se limitan a expresiones seguras.

Se puede demostrar que el álgebra relacional es lógicamente equivalente al cálculo relacional seguro, de modo que cualquier expresión en uno se puede traducir en una expresión equivalente en la otra.

Cálculo relacional orientado a dominio

En el cálculo relacional orientado a dominio se usan variables que toman sus valores de dominios en lugar de tuplas de relaciones. Si $P(x_1, x_2, \ldots, x_m)$ representa un predicado con variables $x_1, x_2, \ldots, x_n$, donde n <= m, entonces

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_m)\}$$

significará el conjunto de todas las variables dominio $x_1, x_2, \ldots, x_n$ para las que el predicado $P(x_1, x_2, \ldots, x_m)$ es verdadero. El predicado debe ser una fórmula compuesta de átomos. En el cálculo relacional de dominio, un **átomo** debe tener una de las siguientes formas:

- $<x_1, x_2, \ldots, x_n> \in R$, donde R es una relación que tiene n atributos y cada x_i es una variable dominio o una constante.
- $x_i \theta x_j$, donde x_i y x_j son variables de dominio y θ es uno de los operadores de comparación (<, <=, >, >=, =, ≠), siempre que x_i y x_j tengan dominios que se puedan comparar usando θ.
- $x_i \theta c$, donde c es una constante en el dominio de x_i y θ es un operador comparación como antes.

Una **fórmula** se construye a partir de átomos usando reglas, del modo siguiente:

- Un átomo es una fórmula.
- Si F_1 y F_2 son fórmulas, entonces también lo son $F_1 \wedge F_2$, $F_1 \vee F_2$ y $\neg F_1$.
- Si F(x) es una fórmula que tiene variable de dominio x, entonces $\exists$ x (F(x)) es una fórmula y $\forall$ x (F(x)) son fórmulas.

Ejemplos

En el cálculo relacional orientado a dominio, con frecuencia se quiere poner a prueba una **condición de membresía** para determinar si los valores pertenecen a una relación. La expresión `<x,y,z> ∈ X` evalúa su certeza si, y sólo si, existe una tupla en la relación `X` con valores `x`, `y`, `z` para sus tres atributos.

En los ejemplos del 6 al 10 se usarán las variables de dominio `SI` para `stuId`, `LN` para `lastName`, `FN` para `firstName`, `MJ` para `major`, `CR` para `credits`, `CN` para `classNo`, `FI` para `facId`, `SH` para `schedule`, `RM` para `room`, `DP` para `department`, `RK` para `rank` y `GR` para `grade`. Note que, cuando un atributo como `LN` aparece en más de una relación, se necesita sólo una variable de dominio para ambas apariciones, pues se usa el mismo dominio. Se supondrá que se puede usar `LN` para nombres de docentes así como apellidos de estudiantes. Se usará la notación $\exists$ x, y, z (F(x, y, z)) como una abreviatura para $\exists$ x($\exists$ y($\exists$ z(F(x, y, z)))).

- Ejemplo 1. Encontrar los nombres de todos los miembros del personal docente que sean profesores en el departamento CSC.

```
{LN | ∃ FI,DP,RK(<FI,LN,DP,RK> ∈ Faculty ∧ RK = 'Professor' ∧
DP = 'CSC')}
```

- Ejemplo 2. Encontrar los apellidos de todos los estudiantes inscritos en CSC201A.

```
{LN | ∃ SI,FN,MJ,CR(<SI,LN,FN,MJ,CR> ∈ Student ∧ ∃ CN,GR(<SI,CN,GR>
∈ Enroll ∧ CN = 'CSC201A'))}
```

- Ejemplo 3. Encontrar los nombres y apellidos de todos los estudiantes que estén inscritos en al menos una clase que se imparta en el salón H221.

```
{FN,LN | ∃ SI,MJ,CR(<SI,LN,FN,MJ,CR> ∈ Student ∧ ∃ CN,GR(<SI,CN,GR ∈ Enroll
∧ ∃ FI,SH,RM(<CN,FI,SH,RM> ∈ Class ∧ RM = 'H221')))}
```

- Ejemplo 4. Encontrar los nombres de los docentes que estén en el departamento CSC pero no impartan CSC201A. (**Nota:** Para este ejemplo, sea `FI'` también la representación de facId.)

```
{LN | ∃ FI,DP,RK(<FI,LN,DP,RK> ∈ Faculty ∧ DP = 'CSC' ∧ ¬ ∃
CN,FI',SH,RM(<CN,FI',SH,RM> ∈ CLASS ∧ CN = 'CSC201A' ∧ FI' = FI))}
```

- Ejemplo 5. Encontrar los apellidos de los estudiantes inscritos en cada clase ofrecida. (**Nota:** Para este ejemplo, sea `SI'` también la representación de studId.)

```
{LN | ∃ SI,FN,MJ,CR(<SI,LN,FN,MJ,CR> ∈ Student ∧ ∀ CN ∃
SI',GR(<SI',CN,GR> ∈ Enroll ∧ SI'= SI))}
```

Todas las consultas del cálculo relacional de dominio que se muestran aquí son seguras. Se puede garantizar la seguridad del cálculo relacional de dominio, como se hizo para cálculo relacional de tupla, al restringir los valores en las tuplas de la expresión a aquellos en el dominio de la fórmula.

4.7 Vistas

En la arquitectura estándar de tres niveles, el nivel superior consiste en vistas externas. Una vista externa es la estructura de la base de datos como aparece a un usuario particular. En el modelo relacional, la palabra "vista" tiene un significado ligeramente diferente. En lugar de ser todo el modelo externo de un usuario, una vista es una tabla virtual, una tabla que en realidad no existe pero se puede construir al realizar operaciones como selección, proyección, combinación u otros cálculos del álgebra relacional sobre los valores de las tablas existentes. Por ende, un modelo externo puede consistir tanto en tablas a nivel conceptual reales como en vistas derivadas de las tablas.

El mecanismo de vista es deseable porque permite ocultar porciones de la base de datos de ciertos usuarios. El usuario no está al tanto de la existencia de algunos atributos que faltan de su vista. También permite a los usuarios acceder a los datos en una forma "personalizada". La vista se debe diseñar para crear un modelo externo que el usuario encuentre familiar y confortable. Por ejemplo, un usuario puede necesitar registros `Enroll` que contengan nombres de estudiante así como los tres atributos que ya están en `Enroll`. Esta vista se crearía al combinar las tablas `Enroll` y `Student`, y luego proyectar sobre los cuatro atributos de interés. Otro usuario puede necesitar ver registros `Student` sin el atributo `credits`. Para este usuario, una proyección se realiza de modo que su vista no tenga la columna `credits`. Los atributos pueden renombrarse, así que el usuario acostumbrado a llamar a la ID de los estudiantes por el nombre `stuNumber` puede ver dicho encabezado de columna, y el orden de las columnas puede cambiar, de modo que `lastName`, `firstName` pueden aparecer como la primera y segunda columnas en una vista. Las operaciones de selección también se pueden usar para crear una vista. Por ejemplo, un jefe de departamento puede ver registros de docentes sólo para dicho departamento. Aquí, una operación Select se realiza de modo que sólo se vea un subconjunto horizontal de la tabla `Faculty`. Las vistas de tabla no se almacenan en forma permanente como tales. En vez de ello, sus definiciones se almacenan en el diccionario de datos y el sistema crea la vista de manera dinámica conforme el usuario la solicite. Cuando el usuario termina con una vista, la tabla de vista se borra, pero las tablas subyacentes a partir de las que se creó permanecen.

Aunque todos los ejemplos previos demuestran que una vista proporciona independencia lógica, las vistas permiten más independencia lógica significativa cuando el nivel lógico se reorganiza. Si a una tabla se agrega una nueva columna, los usuarios existentes pueden no estar al tanto de su existencia si sus vistas se definen para excluirla. Cuando se agregan nuevas tablas, desde luego no hay cambio a los modelos externos de los usuarios existentes. Sin embargo, si una tabla existente se reordena o divide, se puede definir una vista de modo que los usuarios puedan continuar viendo sus antiguos modelos. En el caso de dividir una tabla, la tabla anterior se puede recrear al definir una vista a partir de la combinación de las nuevas tablas, siempre que la división se realice de tal forma que la original se pueda reconstruir. Se puede asegurar que esto es posible al colocar la clave primaria en las nuevas tablas. Por ende, si originalmente una tabla `Student` tenía la forma

```
Student(stuId, lastName, firstName, ssn, major, credits)
```

ésta se podría reorganizar en dos nuevas tablas,

```
PersonalStu(stuId, lastName, firstName, ssn)
AcademicStu(stuId, major, credits)
```

Los usuarios y las aplicaciones todavía podrían ingresar a los datos con el uso de la antigua estructura de tabla, que se recrearía al definir una vista llamada `Student` como la combinación natural de `personalStu` y `academicStu`, con `stuId` como la columna común.

Muchas vistas actúan como "ventanas" en las tablas, lo que permite al usuario ver porciones de tablas reales. Otras que contienen información combinada, calculada o resumida de las tablas reales no son ventanas, sino más como "instantáneas", imágenes de los datos como existían cuando la vista se invocó durante la sesión. En el caso "ventana", la vista es dinámica, lo que significa que los cambios hechos a la tabla real que afectan los atributos de la vista se reflejan inmediatamente en la vista. Cuando los usuarios realizan cambios permitidos a la vista "ventana", dichos cambios se hacen a las tablas subyacentes. Existen restricciones sobre los tipos de modificaciones que se pueden hacer a través de las vistas. Por ejemplo, una vista que no contiene la clave primaria no debe ser actualizable en absoluto. Además, las vistas construidas a partir de información resumen no son actualizables. Cuando un usuario de vista escribe un comando DML usa el nombre de la vista, y el sistema automáticamente convierte el comando en uno equivalente en las tablas subyacentes. El necesario mapeo de información externo/lógico se almacena en el diccionario de datos. Las operaciones (permitidas) se realizan sobre las tablas reales y el sistema regresa resultados en la forma que un usuario espera, con base en la vista que usa.

4.8 Mapeo de un modelo E-R a un modelo relacional

Un diagrama E-R se puede convertir a un modelo relacional con bastante facilidad. Para convertir el diagrama E-R University a un modelo relacional, consulte la figura 3.12.

- Los **conjuntos de entidades fuertes**, representados mediante rectángulos, se convierten en relaciones representadas por tablas. El nombre de la tabla es el mismo que el de la entidad, que es el nombre escrito dentro del rectángulo. Para conjuntos de entidades fuertes, **atributos univaluados no compuestos**, representados mediante óvalos simples, se convierten en atributos de la relación, o encabezados de columna de la tabla. Los conjuntos de entidades fuertes, `Department`, `Student` y `Faculty` se pueden representar de inmediato mediante las siguientes tablas:

  ```
  Department (deptName, deptCode, office)
  Student (stuId, lastName, firstName, major, credits)
  Faculty (facId, lastName, firstName, rank)
  ```

- Para atributos E-R que son **compuestos**, el modelo relacional puro no permite representar directamente el hecho de que el atributo es compuesto. En vez de ello, sencillamente se puede hacer una columna para cada uno de los atributos simples que forman el compuesto, o se puede elegir dejar el compuesto como un solo atributo. Por ejemplo, si se tiene un compuesto, `address`, no se tendría una columna con dicho nombre, pues en su lugar se tendrían columnas individuales para sus componentes: `street`, `city`, `state` y `zip`. Por ejemplo, si incluye una dirección en la tabla `Student`, se podría usar el esquema

  ```
  Student1 (stuId, lastName, firstName, street, city, state, zip, major, credits)
  ```

  o conservar la dirección como un solo atributo

  ```
  Student2 (stuId, lastName, firstName, address, major, credits)
  ```

  La segunda opción haría más difícil seleccionar registros sobre la base del valor de las partes de la dirección, como código postal o estado. En la figura 3.12 se vio que `classNo` es en realidad un compuesto que consiste en `deptCode`, `courseNumber` y `section`. Aquí se eligió dejar dicho compuesto como un solo atributo. Por tanto, para el presente se forma la tabla `Class` como

  ```
  Class (classNo, schedule, room)
  ```

- Los atributos **multivaluados** imponen un problema especial cuando se convierten a un modelo relacional puro, que no permite valores múltiples en una celda. La solución usual es removerlos de la tabla y crear una relación separada en la que se coloque la clave primaria de la entidad, junto con el atributo multivaluado. La clave de esta nueva tabla es la combinación de la clave de la tabla original y el atributo multivaluado. Si existen atributos multivaluados múltiples en la tabla original se tiene que crear una nueva tabla para cada una. Estas nuevas tablas se tratan como si fueran entidades débiles, con la tabla original (sin los atributos multivaluados) que actúa como la entidad propietaria. Es conveniente nombrar las nuevas tablas usando la forma plural del nombre del atributo multivaluado. Como ilustración de cómo manejar atributos multivaluados, considere qué haría si los estudiantes pudieran tener más de una especialidad. Para representar esto se cambiaría la tabla `Student` al remover `major` de ella y crear en su lugar dos tablas

  ```
  Student3(stuId, lastName, firstName, credits)
  StuMajors(stuId, major)
  ```

  Sin embargo, se supone que los estudiantes tienen cuando mucho una especialidad, de modo que en vez de ello se conservará la tabla `Student` original. Otra solución es poner columnas adicionales para los valores múltiples en la tabla original. Con esta solución, si los estudiantes pudieran tener cuando mucho dos especialidades, la tabla `Student` se convertiría en

  ```
  Student4(stuId, lastName, firstName, major1, major2, credits)
  ```

  Si se incluyeran números telefónicos para el personal docente, y los miembros de dicho personal pudieran tener más de un número telefónico almacenado en la base de datos, la solución de múltiples columnas produciría el esquema

  ```
  Faculty (facId, lastName, firstName, rank, phone, alternatePhone)
  ```

  En la figura 3.12 se vio que el atributo `author` para `Textbook` puede tener valores múltiples. Se elegirá el primer método, representar los autores en una tabla separada, lo que da

  ```
  Textbook(isbn, title, publisher)
  TextAuthors(isbn, author)
  ```

- Los **conjuntos de entidades débiles** también se representan mediante tablas, pero requieren atributos adicionales. Recuerde que una entidad débil es dependiente de otra entidad (propietaria) y no tiene clave candidata que consista sólo en sus propios atributos. Por tanto, la clave primaria de la entidad propietaria correspondiente se usa para mostrar de cuál instancia de la entidad propietaria depende una instancia de entidad débil. Para representar la entidad débil use una tabla cuyos atributos incluyan todos los atributos de la entidad débil, más la clave primaria de la entidad propietaria. La entidad débil en sí debe tener un discriminante: algún atributo o conjunto de atributos que, cuando se acople con la clave primaria de la propietaria, permita separar las instancias. Se usa como la clave la combinación de la clave primaria de la propietaria y el discriminante. En la figura 3.12, al conjunto de entidades débil, `Evaluation`, se le dio la clave primaria de su conjunto de entidad propietario, `Faculty`, lo que resulta en la tabla

  ```
  Evaluation(facId, date, rater, rating)
  ```

- Los **conjuntos de relaciones** también se pueden traducir directamente en tablas. Un conjunto de relaciones se puede representar mediante una tabla que tenga las claves primarias de las entidades asociadas como atributos. Si el conjunto de relaciones no tiene atributos descriptivos (no clave), puede construir la correspondiente tabla de

relaciones al crear encabezados de columna que consistan en las claves primarias de las entidades asociadas. Si el conjunto de relaciones tiene atributos descriptivos, éstos también se convierten en atributos de la relación, de modo que se tienen columnas para ellos así como para las claves primarias de las entidades relacionadas.

- Para **relaciones binarias 1-1 o 1-M**, puede elegir no representar conjuntos de relaciones mediante una tabla de relación separada. Siempre que la relación sea uno a uno o uno a muchos, es posible usar claves externas para mostrar la relación.
- Si A:B es **uno a muchos**, puede colocar la clave de A (el lado uno) en la tabla para B, el lado muchos, donde se convierte en una clave externa. `Teaches` es un conjunto de relaciones uno a muchos que conecta los conjuntos de entidad fuertes, `Class` y `Faculty`. Esta relación se podría representar mediante la tabla

  ```
  Teaches(classNo, facId)
  ```

  Se elegiría `classNo` como la clave de `Teaches`, pues `facId` no proporciona valores únicos para esta combinación de atributos. En vez de ello, se usará una clave externa para representar esta relación, al colocar `facId`, la clave del lado "uno", `Faculty`, en la tabla para el lado "muchos", `Class`. En consecuencia, la tabla `Class` cambia a

  ```
  Class (classNo, facId, schedule, room)
  ```

  y no se almacena la tabla `Teaches`.
- Para una **entidad débil**, la relación con la propietaria ya está representada, porque la clave primaria de la entidad propietaria ya está en la tabla para la entidad débil, así que no es necesario usar otra tabla para representar su conexión.
- Todos los conjuntos de entidad que tienen una relación **uno a uno** deben examinarse cuidadosamente para determinar si en realidad son la misma entidad. Si lo son se deben combinar en una sola entidad. Si A y B son entidades en verdad separadas que tienen una relación uno a uno, entonces se puede poner la clave de cualquier relación en la otra tabla para mostrar la conexión (es decir: o poner la clave de A en la tabla B, o viceversa, mas no ambas). Por ejemplo, si los estudiantes pueden tener un permiso de estacionamiento para estacionar un automóvil en el campus, se puede agregar una entidad `Car` que tendría una relación uno a uno con `Student`. Cada automóvil pertenece exactamente a un estudiante y cada estudiante tiene cuando mucho un automóvil. El esquema inicial para `Car` puede ser

  ```
  Car(licNo, make, model, year, color)
  ```

  Se supone que `licNo` incluye el estado donde el carro tiene licencia. Para almacenar la relación uno a uno con `Student` se podría poner `stuId` en la tabla `Car`, que produce

  ```
  Car(licNo, make, model, year, color, stuId).
  ```

  De manera alternativa, se podría agregar `licNo` a la tabla `Student`.
- Para una relación binaria, el único caso donde es imposible eliminar el conjunto de relaciones es el caso **muchos a muchos**. Aquí, la única forma en que se puede mostrar la conexión es mediante una tabla. La relación `Enroll` representa una relación muchos a muchos. La tabla que la representa debe contener las claves primarias de las entidades asociadas `Student` y `Class`, `stuId` y `classNo`, respectivamente. Dado que la relación también tiene un atributo descriptivo, `grade`, también se incluye éste. Note que necesitaría una tabla de relación incluso si no hubiera tal atributo descriptivo para la relación M:M. La tabla de relación es

  ```
  Enroll (stuId, classNo, grade)
  ```

- Cuando se tiene una **relación ternaria**, o **n-aria** que involucre tres o más conjuntos de entidades, debe construir una tabla para la relación, en la que coloque las claves primarias de las entidades relacionadas. Si la relación ternaria o n-aria tiene un atributo descriptivo, va en la tabla de relación. La figura 3.12 muestra una relación ternaria, `Faculty-Class-Textbook`. Se le representa mediante la tabla

  ```
  Faculty-Class-Textbook(facId, classNo, isbn)
  ```

  Se elige la combinación de `classNo` e `isbn` como la clave porque, como muestra el diagrama, para cada combinación de valores `classNo` e `isbn`, sólo hay un `facId`.

- Cuando se tiene una relación **recursiva**, su representación depende de la cardinalidad. Siempre se puede representar tal relación mediante una tabla separada, sin importar su cardinalidad. Si la cardinalidad es muchos a muchos, debe crear una tabla de relaciones correspondiente. Si es uno a uno o uno a muchos, puede usar el mecanismo de clave externa. Por ejemplo, `Chair-Member` podría incluirse como una relación recursiva uno a muchos en `Faculty`. Se podría representar mediante una tabla que muestra pares de valores `facId`, en la que el primer `facId` representa a un miembro del personal docente y el segundo representa al jefe de departamento para dicho docente. Por tanto, la tabla de relación tendría la forma

  ```
  Chair-Member (memberFacId, chairFacId)
  ```

  Los dos atributos tienen el mismo dominio, el conjunto de valores `facId`, pero se les renombró para distinguir entre miembros y jefes de departamento. La clave es `memberFacId`, pues cada miembro del personal docente tiene sólo un jefe de departamento, pero dicho jefe puede asociarse con muchos miembros del departamento. La alternativa sería colocar `chairFacId` como una clave externa en la tabla `Faculty`. La tabla `Faculty` tendría entonces la estructura

  ```
  Faculty (facID, lastName, firstName, department, rank, chairFacId).
  ```

  Si supone que cada estudiante puede tener exactamente un compañero de cuarto, podría representar la relación recursiva Roommate uno a uno o como una tabla separada

  ```
  Roommate(stuId, roommateStuId)
  ```

  o al agregar un atributo a la tabla Student, como en

  ```
  Student(stuId, lastName, firstName, major, credits,
  roommateStuId)
  ```

Siempre que tenga una opción de cómo representar relaciones, como en los casos uno a uno o uno a muchos para relaciones binarias, ¿cómo sabe si usar o no una tabla separada? La respuesta puede depender de las aplicaciones para las que se diseña una base de datos particular. Tener una tabla de relaciones separadas proporciona máxima flexibilidad, lo que permite cambiar las asociaciones con facilidad. Sin embargo, requiere realizar combinaciones siempre que se use la relación, lo que puede producir deficiente rendimiento si muchas aplicaciones requieren las combinaciones. El diseñador debe elegir entre flexibilidad y eficiencia, dependiendo de los criterios para las aplicaciones.

Existen ocho conjuntos de relaciones representados en la figura 3.12 mediante diamantes. El conjunto de relaciones 1:M `HasMajor` que conecta `Department` con `Student` tiene atributos `deptname` y `stuId`, las claves primarias de los conjuntos de entidad asociados. Por tanto, si elige representar la relación como una tabla, usaría

```
HasMajor(deptName, stuId)
```

Se subrayaría `stuId` como la clave primaria, pues este atributo proporciona valores únicos para cada tupla, incluso sin `deptname`. Se elegiría no crear esta tabla de relación en absolu-

to, pues es una relación uno a muchos y hacerlo representará la relación mediante una clave externa. Para hacerlo, coloque la clave primaria del lado "uno", `Department`, en la tabla para el lado "muchos", `Student`. Note que `Student` ya contiene `major`, un atributo que en realidad es una clave externa. (Aquí se supone que los nombres de todos los programas de especialización en realidad sólo son los nombres de departamento. Si éste no es el caso, es decir, los programas de especialidad pueden diferir de los nombres de departamento, entonces necesitaría agregar el nombre del departamento como una clave externa en `Student`.) Si decide crear la tabla de relaciones podría eliminar el atributo `major` de la tabla `Student`. Entonces tendría que efectuar una combinación siempre que necesite encontrar la especialidad de un estudiante. La elección será conservar `major` en `Student` y no usar una tabla separada para esta relación.

La relación `Enroll` representa una relación muchos a muchos, que siempre requiere una tabla separada. Además, tiene un atributo descriptivo, cuya presencia indica que es deseable una tabla separada. Esta tabla debe contener las claves primarias de las entidades asociadas `Student` y `Class`, `stuId` y `classNo`, respectivamente. La tabla de relación es

```
Enroll (stuId, classNo, grade)
```

Note que, en una relación muchos a muchos, necesita las claves primarias tanto para las entidades relacionadas como para las claves primarias de la relación. Recuerde que la calificación aquí significa la calificación de mitad de semestre, pues estos registros son para el semestre actual.

`Offers` es una relación uno a muchos que conecta `Department` con `Class`. Esta relación se puede representar mediante la tabla

```
Offers(deptname, classNo)
```

Una alternativa es poner la clave de `Department`, `deptName`, en la tabla `Class`. Aquí se elegirá usar la tabla de relación. `Teaches` es un conjunto de relaciones uno a muchos que conecta los conjuntos de entidades fuertes, `Class` y `Faculty`. Esta relación podría representarse mediante la tabla

```
Teaches (classNo, facId)
```

Se elegiría `classNo` como la clave, pues `facId` no proporciona valores únicos para esta combinación de atributos. En vez de ello se usará una clave externa para representar esta relación, al colocar `facId`, la clave del lado "uno", `Faculty`, en la tabla para el lado "muchos", `Class`. Por tanto, la tabla `Class` cambia a

```
Class (classNo, facId, schedule, room)
```

`Employs` es una relación uno a muchos que representa la asociación entre `Department` y `Faculty`. Tiene la opción de representar explícitamente la relación mediante la tabla

```
Employs (deptName, facId)
```

De nuevo, elija no crear esta tabla y use el mecanismo de la clave externa y agregue `deptName` como un atributo en la tabla `Faculty`, lo que hace a dicha tabla

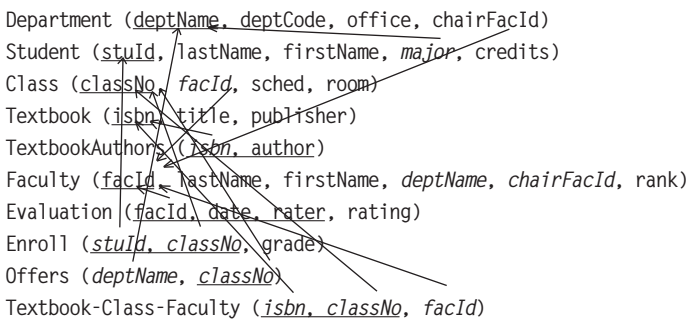
```
Faculty (facId, lastName, firstName, deptName, rank)
```

`Chairs` es una relación uno a uno que conecta `Department` con `Faculty`. La clave primaria de `Faculty` se empujará a `Department` como una clave externa, lo que hace el esquema `Department`

```
Department (deptName, deptCode, office, chairFacId)
```

Como se indicó anteriormente, la relación ternaria que conecta `Textbook`, `Faculty` y `Class` se debe representar mediante una tabla. La relación `isRated` conecta la entidad débil `Evaluation` con su propietaria, `Faculty`. Se representará al tener la clave primaria de la entidad propietaria en la tabla de la entidad débil como parte de su clave primaria.

Entonces, todo el esquema es



4.9 Reglas de Codd para un sistema de gestión de base de datos relacional

En dos artículos de 1985, Codd publicó reglas o principios que debe usar un sistema de gestión de base de datos para ser considerado "completamente relacional". (Codd, E. F., "Is Your DBMS Really Relational?", *Computerworld*, 14 de octubre de 1985; "Does Your DBMS Run by the Rules?", *Computerworld*, 21 de octubre de 1985.) Codd quería mantener la integridad del modelo relacional y dejar en claro que colocar una interfaz de usuario relacional por encima de un sistema que utilizaba algún otro modelo como su modelo de datos básico no era suficiente para hacer verdaderamente relacional un DBMS. Él identificó 12 reglas, junto con una regla abarcadora fundamental que llamó Regla Cero. Las reglas proporcionan un conjunto de estándares para juzgar si un DBMS es completamente relacional. Las reglas, que fueron tema de mucho debate, se resumen a continuación:

- **Regla Cero.** Un sistema de gestión de bases de datos relacional debe gestionar sus datos almacenados sólo con el uso de sus capacidades relacionales. Éste es el principio fundamental sobre el que se basan las 12 reglas restantes.
- **Regla 1-Representación de información.** Toda información debe representarse, en el nivel lógico, sólo como valores en tablas.
- **Regla 2-Acceso garantizado.** Debe ser posible acceder a cualquier ítem de datos en la base de datos al proporcionar su nombre de tabla, nombre de columna y valor de clave primaria.
- **Regla 3-Representación de valores nulos.** El sistema debe ser capaz de representar valores nulos en una forma sistemática, sin importar el tipo de datos del ítem. Los valores nulos deben ser distintos de cero o cualquier otro número, y de cadenas vacías.
- **Regla 4-Catálogo relacional.** El catálogo del sistema, que contiene la descripción lógica de la base de datos, debe representarse de la misma forma que los datos ordinarios.
- **Regla 5-Sublenguaje de datos amplio.** Sin importar el número de otros lenguajes que soporte, la base de datos debe incluir un lenguaje que permite enunciados expresados como cadenas de caracteres para soportar definición de datos, definición de vistas, manipulación de datos, reglas de integridad, autorización de usuario y un método de identificación de unidades para recuperación.
- **Regla 6-Actualización de vistas.** Cualquier vista que sea teóricamente actualizable en realidad la puede actualizar el sistema.

- **Regla 7-Operaciones Insert, Delete y Update.** Cualquier relación que se pueda manejar como un solo operando para recuperación (retrieval) también se puede manejar de esa forma para operaciones de inserción, borrado y actualización.
- **Regla 8-Independencia física de datos.** Los programas de aplicación son inmunes a cambios hechos a representaciones de almacenamiento o métodos de acceso.
- **Regla 9-Independencia lógica de datos.** Los cambios efectuados a nivel lógico, como dividir tablas o combinar tablas, que no afectan el contenido de información a nivel lógico, no requieren modificación de aplicaciones.
- **Regla 10-Reglas de integridad.** Las restricciones de integridad como la integridad de entidad y la integridad referencial deben especificarse en el sublenguaje de datos y almacenarse en el catálogo. Para expresar estas restricciones no se deben usar enunciados de programa de aplicación.
- **Regla 11-Independencia de distribución.** El sublenguaje de datos debe ser tal que, si la base de datos se distribuye, los programas de aplicación y los comandos de los usuarios no necesitan cambiar.
- **Regla 12-No subversión.** Si el sistema permite un lenguaje que soporte acceso a registro a la vez, cualquier programa que use este tipo de acceso no puede pasar por alto las restricciones de integridad expresadas en el lenguaje de nivel superior.

4.10 Resumen del capítulo

Una **relación** matemática se define como un subconjunto del producto cartesiano de conjuntos. En términos de base de datos, una relación es cualquier subconjunto del producto cartesiano de los dominios de los atributos. Una relación normalmente se escribe como un conjunto de **n-tuplas**, en las que cada elemento se elige del dominio adecuado. Las relaciones se representan físicamente como **tablas**, con las filas correspondientes a registros individuales y las columnas a los atributos de la relación. La estructura de la tabla, con especificaciones de dominio y otras restricciones es la **intensión** de la base de datos, mientras que la tabla con todas sus filas escritas es una instancia o **extensión** de la base de datos. Las propiedades de las relaciones de las bases de datos son: cada celda tiene un solo valor, los nombres de columna son distintos, los valores de una columna vienen todos del mismo dominio, el orden de las filas es irrelevante y no hay filas duplicadas.

El **grado** de una relación es el número de atributos o columnas. Una relación **unaria** tiene una columna, una relación **binaria** tiene dos, una relación **ternaria** tiene tres y una relación **n-aria** tiene *n* columnas. La **cardinalidad** de una relación es el número de filas o tuplas. El grado es una propiedad de la intensión, mientras que la cardinalidad es una propiedad de la extensión de una relación. Una **superclave** es un conjunto de atributos que identifica de manera única las tuplas de la relación, mientras que una **clave candidata** es una superclave mínima. Una **clave primaria** es la clave candidata elegida para usar en la identificación de las tuplas. Una relación siempre debe tener una clave primaria. Una **clave externa** es un atributo de una relación que no es la clave primaria de dicha relación, pero es la clave primaria de alguna relación (usualmente otra), llamada su **relación base**. La **integridad de entidad** es una restricción que establece que ningún atributo de una clave primaria puede ser nulo. La **integridad referencial** afirma que los valores de las claves externas deben coincidir con los valores de la clave primaria de alguna tupla en la relación base o ser completamente nulos. Otras reglas de integridad incluyen **restricciones de dominio**, **restricciones de tabla** y **restricciones generales**.

Los lenguajes de manipulación de datos relacionales pueden ser **procedurales** o **no procedurales**, **gráficos**, **de cuarta generación** o **de quinta generación**. El **álgebra relacional** es un lenguaje procedural formal. Sus operadores incluyen selección, proyección, producto, unión, intersección, diferencia, división y varios tipos de combinaciones, semicombinaciones y combinaciones exteriores. El **cálculo relacional** es un lenguaje no procedural formal que usa predicados. El tipo más común de consulta en el cálculo relacional orientado a tupla tiene la forma $\{x \mid P(x)\}$, que significa "encontrar el conjunto de todas las variables de tupla x tales que el predicado $P(x)$ sea verdadero". Una consulta típica en el cálculo relacional orientado a dominio tiene la forma $\{<x_1,x_2, \ldots, x_n> \mid P(x_1, x_2, \ldots, x_n)\}$, que significa "encontrar el conjunto de todas las variables de dominio para las que el predicado sea verdadero". El álgebra relacional es lógicamente equivalente a un subconjunto seguro de cálculo relacional.

Una **vista** en el modelo relacional no es un modelo externo completo, sino una **tabla virtual**. La vista protege la seguridad y permite al diseñador personalizar el modelo de un usuario. Las vistas se crean dinámicamente cuando el usuario hace una petición de datos.

Al convertir un modelo E-R a uno relacional, las entidades fuertes se convierten en tablas que tienen una columna por cada uno de los atributos simples univaluados de la entidad. Los atributos compuestos pueden almacenarse como un solo atributo o representar cada uno de los atributos simples que componen al compuesto mediante una columna, sin representación del compuesto. Cada atributo multivaluado se remueve y coloca en una tabla separada, junto con la clave primaria de la tabla original. Las tablas para entidades débiles tienen columnas para los atributos clave de la entidad propietaria asociada, así como para los atributos de la entidad débil. Las tablas de relación tienen columnas para los atributos de clave primaria de las entidades relacionadas, más una columna por cada atributo descriptivo de la relación. Las relaciones muchos a muchos requieren una tabla de relación separada, pero las relaciones uno a uno y uno a muchos se pueden representar mediante claves externas en lugar de mediante tablas. El diseñador puede elegir cualquier método que se ajuste mejor a la aplicación, y negociar flexibilidad por eficiencia. Las relaciones ternarias y n-arias se representan mejor como tablas separadas. Las relaciones recursivas se pueden representar mediante claves externas, siempre que sean uno a uno o uno a muchos, o mediante una tabla de relaciones separada, que se requiere si son relaciones muchos a muchos.

Ejercicios

4.1 Sean S = {rojo, amarillo, verde} y T = {cuadros, tiras, punto}. Encuentre el producto cartesiano de S y T.

4.2 Sean Q = {Tom, Mary, Jim} y R = {caminar, correr}. Cree una relación con Q y R como dominios.

4.3 Considere el esquema de relación que contiene datos de libros para una librería: `Book (title, author, isbn, publisher, pubDate, city, qtyOnHand)`

a. Escriba la tabla para una instancia de esta relación.

b. Identifique una superclave, una clave candidata y la clave primaria, y escriba cualesquiera suposiciones que necesite hacer para justificar su elección.

4.4 Considere la siguiente instancia de base de datos que contiene información acerca de empleados y los proyectos a los que se asignan:

Emp

empId	lastName
E101	Smith
E105	Jones
E110	Adams
E115	Smith

Proj

projNo	projName	budget
P10	Hudson	500000
P15	Columbia	350000
P20	Wabash	350000
P23	Arkansas	600000

Assign

empId	projNo	hours
E101	P10	200
E101	P15	300
E105	P10	400
E110	P15	700
E110	P20	350
E115	P10	300
E115	P20	400

Muestre TODAS las tablas (incluidas las intermedias) que produciría cada uno de los siguientes comandos de álgebra relacional:

a. `SELECT Emp WHERE lastName = 'Adams' GIVING T1`

`JOIN T1, Assign GIVING T2`

Simbólicamente esto es,

$(\sigma_{\text{lastName='Adams'}}(\text{Emp}))\ |X|\ \text{Assign}$

b. `SELECT Proj WHERE budget > 400000 GIVING T1`

`JOIN T1, Assign GIVING T2`

`PROJECT T2 OVER empId GIVING T3`

Simbólicamente esto es,

$\Pi_{\text{empId}}((\sigma_{\text{budget}>400000}(\text{Proj}))|X|\ \text{Assign})$

c. `PROJECT Assign OVER projNo GIVING T1`

`JOIN T1, Proj GIVING T2`

`PROJECT T2 OVER budget GIVING T3`

Simbólicamente esto es,

$\Pi_{\text{budget}}(\Pi_{\text{projNo}}(\text{Assign})\ |X|\ \text{Proj})$

4.5 Use las tablas Emp, Proj y Assign que se muestran en el ejercicio 4.4, y muestre qué resultados se producirían por cada uno de los siguientes comandos de cálculo relacional de tupla:

a. $\{E.empId \mid E \in Emp \wedge E.lastName = \text{'Smith'}\}$

b. $\{P \mid P \in Proj \wedge P.budget < 500000\}$

c. $\{E.lastName \mid E \in Emp \wedge \exists A(A \in Assign \wedge (A.empId = E.empId) \wedge (A.projNo = \text{'P10'}))\}$

d. $\{E \mid E \in Emp \wedge \exists A(A \in Assign \wedge (E.empId = A.empId) \wedge \exists P(P \in Proj \wedge (A.projNo = P.projNo) \wedge (P.projName = \text{'Columbia'})))\}$

4.6 Con las tablas `Emp`, `Proj` y `Assign` del ejercicio 4.4, muestre cuáles resultados se producirían por cada uno de los siguientes comandos de álgebra relacional de dominio. Suponga que las variables `EI` y `EI'` representan `empId`, `LN` representa `lastName`, `PN` representa `projNo`, `PM` representa `projName`, `BG` representa `budget` (presupuesto) y `HR` representa `hours`.

a. $\{EI \mid \exists LN(\langle EI,LN\rangle \in Emp \wedge LN = \text{'Smith'}\}$

b. {LN | ∃ EI(<EI,LN> ∈ Emp ∧ ∃ PN,HR(<EI,PN,HR> ∈ Assign ∧ PN = 'P15'))}

c. {EI| ∃ LN(<EI,LN> ∈ Emp ∧ ∃ PN,HR(<EI,PN,HR ∈ Assign ∧ ∃ PM,BG(<PN,PM,BG> ∈ Proj ∧ BG>= 500000)))}

d. {LN | ∃ EI(<EI,LN> ∈ Emp ∧ ¬∃ EI',PN,HR(<EI',PN,HR> ∈ Assign ∧ PN = 'P20' ∧ EI′ = EI))}

4.7 Considere el siguiente esquema para una base de datos que conserva información acerca de viajes de negocios y sus gastos asociados por empleado:

EMPLEADO (NSS, Nombre, DeptNo, TituloPuesto, Salario)

VIAJE (ViajeId, CiudadPartida, CiudadDestino, FechaPartida, FechaRegreso, NSS)

GASTOS (ViajeId, Concepto, Fecha, Cantidad)

Escriba consultas de álgebra relacional para cada uno de los siguientes:

a. Obtener una lista de todas las diferentes ciudades de destino donde realizaron viajes los empleados.

b. Encontrar toda la información de empleado para los empleados que trabajen en el Departamento 10.

c. Obtener registros de viaje completos (pero no sus gastos asociados) para todos los viajes con fechas de partida después del 1 de enero del año actual.

d. Encontrar los nombres de todos los empleados que partieron en viajes desde Londres.

e. Encontrar el NSS de todos los empleados que tengan cualquier concepto de gasto individual de más de $1 000 para cualquier viaje.

f. Encontrar los nombres de todos los empleados que tengan cualquier concepto de gasto con valor "Entretenimiento".

g. Encontrar las ciudades de destino de todos los viajes realizados por los empleados que tengan el título de puesto "Consultor".

h. Encontrar los nombres y departamentos de todos los empleados que tengan un concepto de gasto individual por más de $1 000 desde el 1 de enero de este año.

i. Encontrar los conceptos y cantidades de todos los gastos para un viaje al Cairo que comience el 3 de enero de este año realizado por el empleado Jones.

j. Encontrar los nombres, departamentos y títulos de puesto de todos los empleados que tengan cualquier gasto con valor "Cargo de servicio" para viajes realizados a Melbourne el año pasado, junto con la fecha y cantidad del gasto.

4.8 Con el esquema `Empleado`, `Viaje`, `Gastos` del ejercicio 4.7, escriba comandos de cálculo relacional de tupla para cada una de las siguientes consultas.

a. Encontrar los nombres de todos los empleados que trabajen en el Departamento 10.

b. Encontrar los números de seguridad social y nombres de todos los empleados que hayan realizado un viaje de negocios a Hong Kong.

c. Encontrar los números de seguridad social y nombres de todos los empleados que tuvieron cualquier gasto para un concepto llamado "Misceláneo".

d. Encontrar todos los conceptos de gasto y cantidades para cualquier gasto de negocio incurrido por el empleado Smith.

4.9 Con el esquema `Empleado`, `Viaje`, `Gastos` del ejercicio 4.7, escriba comandos de cálculo relacional de dominio para cada una de las siguientes consultas.

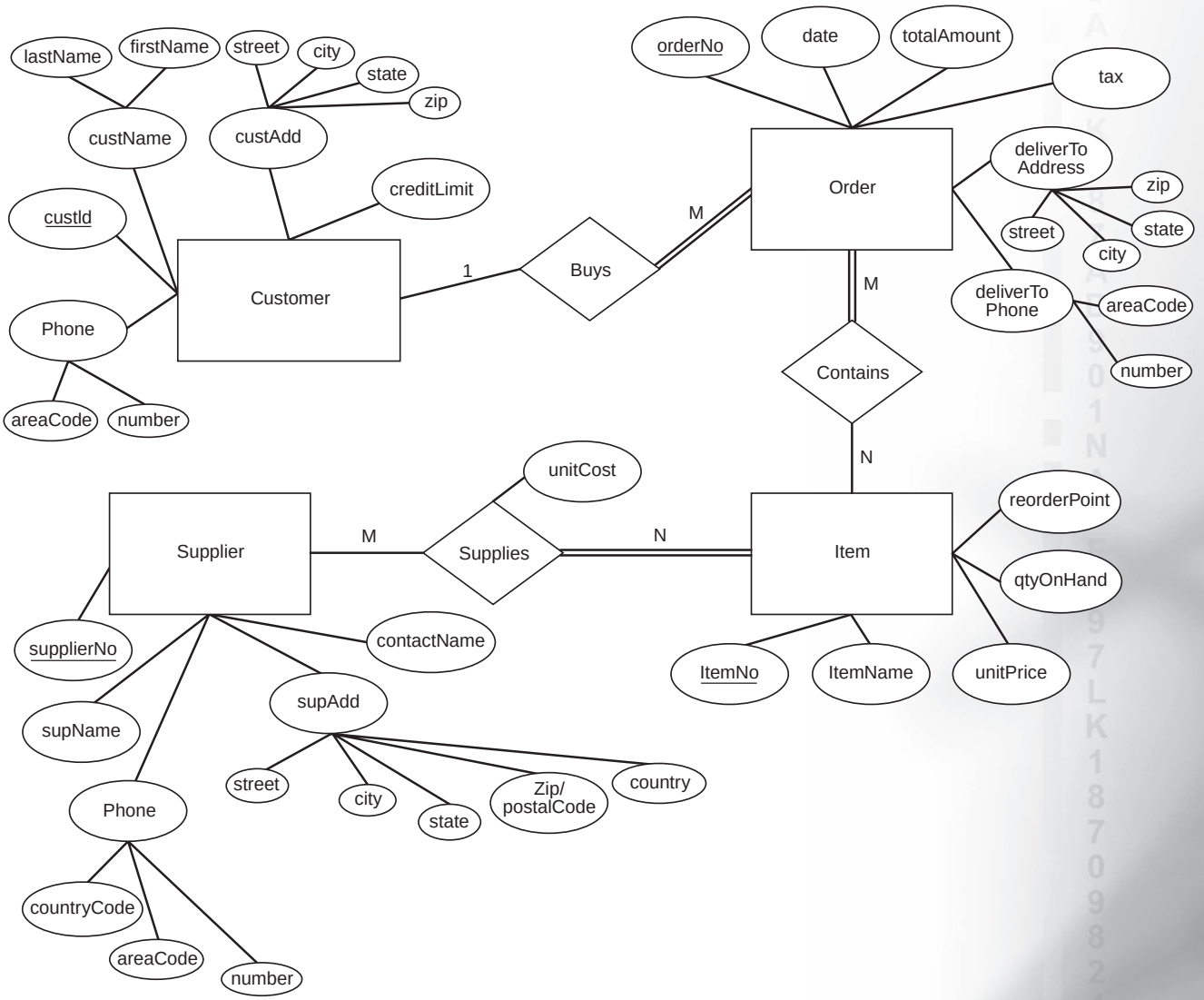


FIGURA 4.9

Diagrama E-R para el ejemplo CustomerOrder (pedido de cliente)

a. Encontrar los números de seguridad social de todos los empleados que tengan título de puesto Programador.

b. Encontrar los nombres de todos los empleados que tengan un viaje de negocios con partida desde San Francisco.

c. Encontrar los nombres de los empleados que no hayan realizado viajes de negocios.

d. Encontrar todos los conceptos de gastos para un viaje de negocios realizado por el empleado Jones con fecha de partida del 10 de enero de este año.

4.10 Diseñe un esquema de base de datos relacional correspondiente al diagrama que se muestra en la figura 4.9.

4.11 Diseñe un esquema de base de datos relacional para los datos descritos en los ejercicios 3.2 y 3.3.

4.12 Diseñe un esquema de base de datos relacional para los datos descritos en el ejercicio 3.4.

4.13 Diseñe un esquema de base de datos relacional para los datos descritos en el ejercicio 3.5.

4.14 Diseñe un esquema de base de datos relacional para los datos descritos en el ejercicio 3.6.

4.15 Diseñe un esquema de base de datos relacional para los datos descritos en el ejercicio 3.7.

4.16 Diseñe un esquema de base de datos relacional para los datos descritos en el ejercicio 3.8.

PROYECTO DE MUESTRA: MAPEO INICIAL DEL MODELO E-R A TABLAS PARA LA GALERÍA DE ARTE

- Paso 4.1. Mapee el modelo E-R desarrollado al final del capítulo 3 a modelo relacional, usando los lineamientos presentados en la sección 4.9.

El diagrama E-R desarrollado al final del capítulo 3 mostró las entidades fuertes `Artist`, `PotentialCustomer`, `Collector`, `Show`, `Sale`, `Buyer` y `Salesperson`, y una entidad débil, `Artwork`.

Las entidades fuertes se mapean a las siguientes tablas. Note que los atributos compuestos se sustituyeron por sus componentes simples y que se subrayaron las claves primarias. Aunque no es necesario, se acostumbra mencionar las claves primarias como las primeras columnas en las tablas.

```
Artist(firstName, lastName, street, city, state, zip, interviewDate, interviewerName,
areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber,
usualMedium, usualStyle, usualType)

PotentialCustomer(firstName, lastName, street, city, state, zip, areaCode, telephoneNumber,
dateFilledIn, preferredArtist, preferredMedium, preferredStyle, preferredType)

Collector(SocialSecurityNumber, firstName, lastName, street, city, state, zip,
interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate,
collectionArtistFirstName, collectionArtistLastName, collectionMedium, collectionStyle,
collectionType, SalesLastYear, SalesYearToDate)

Show(showTitle, showFeaturedArtist, showClosingDate, showTheme, showOpeningDate)

Sale(invoiceNumber, amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission,
saleTax, SaleTotal)

Buyer(firstName, lastName, street, city, state, zip, areaCode, telephoneNumber,
purchasesLastYear, purchasesYearToDate)

Salesperson(socialSecurityNumber, firstName, lastName, street, city, state, zip)
```

Dado que la entidad débil `Artwork` depende de `Artist`, se agrega la clave de `Artist` a la tabla `Artwork` y se combina con la clave parcial de la entidad débil para formar una clave primaria, del modo siguiente:

```
Artwork(artistLastName, artistFirstName, workTitle, askingPrice, dateListed, dateReturned,
dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted)
```

Los conjuntos de relaciones son: `PreferredBy`, `CollectedBy`, `Creates`, `FeaturedIn`, `Owns`, `ShownIn`, `SoldIn`, `SoldTo` y `SoldBy`.

La relación uno a muchos `PreferredBy` podría ser una tabla separada que tenga las claves primarias de `Artist` y `PotentialCustomer` como columnas, pero en vez de ello se elige representarla mediante una clave externa. Por tanto, es necesario poner `artistLastName` y `artistFirstName` en la tabla `PotentialCustomer`. Note que `preferredArtist` ya está en dicha tabla. Se le sustituye por los dos atributos, que se llamarán `preferredArtistLastName` y `preferredArtistFirstName`, que juntos formarán una clave externa, indicada mediante cursivas en el esquema que se muestra a continuación en negrillas.

La relación `CollectedBy` también es uno a muchos, y se puede representar al colocar la clave de `Artist` en `Collector`. Note que `Collector` ya tiene `collectionArtistLastName`, `collectionArtistFirstName`, así que la relación ya está representada.

La relación `Creates` ya se representó al colocar la clave primaria de `Artist` en `Artwork`. Con cursivas se muestra que estos atributos, aunque parte de la clave primaria, también forman una clave externa en `Artwork`.

`FeaturedIn` es una relación uno a muchos que se puede representar con el uso de una clave externa. Note que `Show` ya tiene un atributo llamado `showFeaturedArtist`. Se cambia a `showFeaturedArtistLastName`, `showFeaturedArtistFirstName`.

La relación uno a muchos `Owns` se puede representar al colocar la clave primaria de `Collector` en la tabla `Artwork`, así que a dicha tabla se agrega `collectorSocialSecurityNumber`, con cursivas para mostrar que es una clave externa.

La relación `ShownIn` es muchos a muchos, así que se debe construir una tabla con las claves primarias de `Artwork` y `Show` para representarla. Dado que no existen atributos descriptivos en el diagrama E-R, ésta es una tabla "toda claves", sin atributos no clave. Note también que sus atributos son claves externas y se refieren a las tablas donde son claves primarias.

La relación `SoldIn` es uno a uno. Si no quiere construir una nueva tabla, tiene las opciones de colocar la clave de `Artwork` en `Sale`, o de `Sale` en `Artwork`. Note que no necesita hacer ambas cosas. Aquí se elegirá la primera alternativa.

La relación `SoldTo` es muchos a uno. Aquí se elige representarla mediante la colocación de la clave del lado "uno", `Buyer`, en la tabla para el lado "muchos", `Sale`, donde es la clave externa `buyerLastName`, `buyerFirstName`, `areaCode`, `telephoneNumber`.

De igual modo, para representar la relación uno a muchos `SoldBy`, coloque la clave primaria de `Salesperson` en la tabla `Sale`, donde se convierte en la clave externa `salespersonSocialSecurityNumber`.

Las tablas resultantes en el esquema relacional a nivel conceptual son las siguientes:

```
Artist (firstName, lastName, street, city, state, zip, interviewDate, interviewerName,
areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber,
usualMedium, usualStyle, usualType)

PotentialCustomer(firstName, lastName, street, city, state, zip, areaCode, telephoneNumber,
dateFilledIn, preferredArtistLastName, preferredArtistFirstName, preferredMedium,
preferredStyle, preferredType)

Artwork(artistLastName, artistFirstName, workTitle, askingPrice, dateListed, dateReturned,
dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted,
collectorSocialSecurityNumber)

ShownIn (artistLastName, artistFirstName, workTitle, showTitle)

Collector (SocialSecurityNumber, firstName, lastName, street, city, state, zip,
interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate,
```

collectionArtistFirstName, *collectionArtistLastName*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

Show (showTitle, *showFeaturedArtistLastName*, *showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

Sale (InvoiceNumber, *artistLastName*, *artistFirstName*, *workTitle*, amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, SaleTotal, *buyerLastName*, *buyerFirstName*, *buyerAreaCode*, *buyerTelephoneNumber*, *salespersonSocialSecurityNumber*)

Buyer(firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, purchasesLastYear, purchasesYearToDate)

Salesperson (socialSecurityNumber, firstName, lastName, street, city, state, zip)

Note que las claves primarias de `Artist`, `Buyer` y `PotentialCustomer` constan de dos o más atributos de cadena de caracteres. Observe que se vuelve pesado incluir estos atributos múltiples cuando los usa como claves externas. Este conflicto se abordará en un capítulo posterior.

PROYECTOS ESTUDIANTILES: MAPEO INICIAL A TABLAS PARA PROYECTOS ESTUDIANTILES

- Paso 4.1. Mapee el modelo E-R desarrollado al final del capítulo 3 para un modelo relacional, usando los lineamientos presentados en la sección 4.8 e ilustrados en el Proyecto de Muestra.

CAPÍTULO 5

Normalización

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Por qué las relaciones se deben normalizar
- El significado de dependencia funcional y su relación con las claves
- Cómo se pueden usar las reglas de inferencia para dependencias funcionales
- La definición de la primera forma normal y cómo lograrla
- El significado de dependencia funcional completa
- La definición de la segunda forma normal y cómo lograrla
- El significado de la dependencia transitiva
- Definición de la tercera forma normal y cómo lograrla
- Definición de la forma normal Boyce-Codd y cómo lograrla
- El significado de las dependencias multivaluadas
- Definición de la cuarta forma normal y cómo lograrla
- El significado de dependencia de combinación
- Definición de la quinta forma normal (forma normal proyección-combinación)

- Definición de la forma normal dominio-clave
- Cuándo detener el proceso de normalización

5.1 Objetivos de la normalización

El objetivo básico del modelado lógico es desarrollar una "buena" descripción de los datos, sus relaciones y sus restricciones. Para el modelo relacional, esto significa que debe identificar un conjunto adecuado de relaciones. Sin embargo, la tarea de elegir las relaciones es difícil, porque existen muchas opciones para que el diseñador las considere. Este capítulo explica algunos métodos para mejorar el diseño lógico. Las técnicas que aquí se presentan se basan en un gran conjunto de investigación en el proceso de diseño lógico generalmente llamado normalización.

El propósito de la normalización es producir un conjunto estable de relaciones que sea un modelo fiel de las operaciones de la empresa. Al seguir los principios de la normalización, se logra un diseño que es muy flexible, lo que permite al modelo extenderse cuando necesite representar nuevos atributos, conjuntos de entidades y relaciones. La base de datos se diseña en tal forma que se pueden fortalecer con facilidad ciertos tipos de restricciones de integridad. También se puede reducir la redundancia en la base de datos, tanto para ahorrar espacio como para evitar inconsistencias en los datos. También asegura que el diseño esté libre de ciertas anomalías de actualización, inserciones y borrado. Una anomalía es un estado inconsistente, incompleto o contradictorio de la base de datos. Si estas anomalías estuvieran presentes sería incapaz de representar cierta información, podría perder información cuando ciertas actualizaciones se realicen y correría el riesgo de que los datos se vuelvan inconsistentes con el tiempo.

5.2 Anomalías de inserción, actualización y borrado

Considere la siguiente relación

```
NewClass(classNo, stuId, stuLastName, facId, schedule, room, grade)
```

Una instancia de esta relación aparece en la figura 5.1. En este ejemplo se supondrá que solamente existe un miembro del personal docente para cada clase (esto es: no hay enseñanza en equipo). También se supone que cada clase siempre tiene asignado el mismo salón. Esta relación muestra anomalías de actualización, inserción y borrado.

- **Anomalía de actualización.** Suponga que quiere cambiar el horario de ART103A a MWF12. Es posible que pueda actualizar los dos primeros registros de la tabla `NewClass`, pero no el tercero, lo que resulta en un estado inconsistente en la base de datos. Entonces sería imposible decir el verdadero horario para dicha clase. Ésta es una anomalía de actualización.

courseNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TUTHF10	M110	A
CSC201A	S1006	Lee	F105	TUTHF10	M110	G
HST205A	S1001	Smith	F202	MWF11	H221	

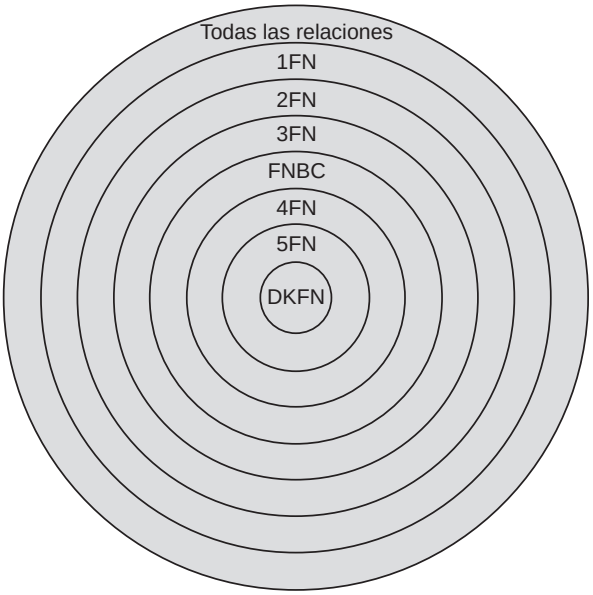
FIGURA 5.1
La tabla NewClass

- **Anomalía de inserción.** Una anomalía de inserción ocurre cuando intenta agregar información acerca de un curso para el cual todavía no hay estudiantes registrados. Por ejemplo, suponga que crea una nueva clase, con valores MTH110A, F110, MTuTh10, H225 para `classNumber`, `facId`, `schedule` y `room`. No es posible registrar la información del curso, aun cuando tenga los valores para estos atributos. Dado que la clave es `{courseNo,stuId}`, no tiene permiso para insertar un registro con un valor nulo para `stuId`. Puesto que no tiene posibilidad de representar esta información de clase, tiene una anomalía de inserción. El mismo problema ocurriría si intenta insertar información acerca de un estudiante que todavía no esté registrado en curso alguno.
- **Anomalía de borrado.** Cuando borra el registro del único estudiante que toma un curso particular, ocurre una anomalía de borrado. Por ejemplo, si el estudiante S1001 se retira de HST205A, perdería toda la información acerca de dicho curso. Sería deseable conservar la información del curso, pero no puede hacerlo sin un `stuId` correspondiente. De igual modo, si un estudiante abandona el único curso que toma, se pierde toda la información acerca de dicho estudiante.

La investigación acerca de estas anomalías la realizó por primera ocasión Codd, quien identificó las causas y definió las primeras tres "formas normales". Una relación es una forma normal específica si satisface el conjunto de requisitos o restricciones para dicha forma. Note que las restricciones que se discuten son restricciones de esquema, propiedades permanentes de la relación, no simplemente de alguna instancia de la relación. Son propiedades de la intensión, no sólo de una extensión particular. Investigación posterior de Boyce y Codd condujo a un refinamiento de la tercera de estas formas. Investigación adicional de Fagin, Zaniolo y Delobel (cada uno de manera independiente) resultó en la definición de tres nuevas formas normales. Todas las formas normales son anidadas (nested), en cuanto a que satisfacen las restricciones de las anteriores, pero es una "mejor" forma porque cada una elimina los fallos que se encuentran en la forma previa. La figura 5.2 muestra cómo se relacionan las siete formas normales. El círculo más grande representa todas las relaciones. Entre el conjunto de todas las relaciones, aquellas que satisfacen ciertas condiciones están en la primera forma normal y se representan mediante el segundo círculo más grande. De las que están en la primera forma normal, existen algunas que tienen condiciones adicionales y que también están en la segunda forma normal; éstas caen en el siguiente círculo, y así por el estilo. El objetivo del diseño debe ser poner el esquema en la forma normal más alta, que es práctica y adecuada para los datos en la base de datos. Normalización significa poner una relación en una forma normal superior. La normalización requiere tener un uso claro de la semántica del modelo. Examinar simplemente una instancia o extensión no es suficiente, porque una instancia no proporciona suficiente información acerca de todos los posibles valores o combinaciones de valores para atributos de relación.

Al intentar se puntualicen las causas de las anomalías de actualización, inserción y borrado, los investigadores identificaron tres tipos de dependencias: **dependencias funcionales**,

FIGURA 5.2
Relaciones de las formas normales



dependencias multivaluadas y **dependencias de combinación**. En la literatura de investigación aparecen dependencias adicionales.

5.3 Dependencia funcional

Una **dependencia funcional (DF)** es un tipo de relación entre atributos, como se describe mediante la siguiente definición:

> **Definición:** Si R es un esquema de relación y A y B son conjuntos de atributos no vacíos en R, se dice que B es **funcionalmente dependiente** en A si y sólo si cada valor de A en R tiene asociado exactamente un valor de B en R.

Esto se escribe como

$$A \rightarrow B$$

que se lee como "A determina funcionalmente a B". La definición dice que si dos tuplas en una extensión de R tienen el mismo valor para A, también deben tener los mismos valores para B. De manera más formal, para cada par de tuplas, t_1 y t_2, en cada instancia de R, se tienen las siguientes reglas:

Si $t_1.A = t_2.A$, entonces $t_1.B = t_2.B$

Aquí, la notación $t_1.A$ significa la proyección de la tupla t_1 sobre los atributos del conjunto A. Esta regla no significa que A **causa a** B o que el valor de B se puede calcular a partir del valor de A mediante una fórmula, aunque a veces éste es el caso. Simplemente significa que, si se conoce el valor de A y se examina la tabla de relación R, se encontrará sólo un valor de B en todas las filas que tienen el valor dado de A en cualquier momento dado. Por tanto, cuando dos filas tienen el mismo valor A, también deben tener el mismo valor B. Sin embargo, para un valor B dado, existen varios valores diferentes de A. Una dependencia funcional es en realidad una **relación muchos a uno** del conjunto de atributos A al conjunto de atributos B. Es una **restricción de integridad** que toda instancia de la base de datos debe obedecer. Note que A y B pueden ser conjuntos que consisten de un solo atributo. Cuando existe una dependencia funcional, el conjunto de atributos en el lado izquierdo de la flecha (A en este ejemplo) se llama **determinante** y el conjunto de atributos en el lado derecho de la flecha se llama **dependiente**.

NewStudent

stuId	LastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.3

Instancia de la tabla NewStudent (suponga que cada estudiante tiene sólo una especialidad)

Para ilustrar la dependencia funcional considere la siguiente relación:

```
NewStudent(stuId, lastName, major, credits, status, socSecNo)
```

La figura 5.3 muestra una instancia de la relación. Esta relación almacena información acerca de los estudiantes en un colegio. Aquí, se supondrá que cada estudiante tiene ID y número de seguridad social únicos, y que cada estudiante tiene cuando mucho una especialidad. Se supondrá que los apellidos no son únicos, que dos estudiantes diferentes pueden tener el mismo apellido. El atributo `credits` significa el número de créditos completados y `status` se refiere al año en el que está el estudiante: freshman (1°), sophomore (2°), junior (3°) y senior (4°). Aunque esta instancia se usa para ayudar a detectar las dependencias funcionales, se intentará determinar características permanentes de la relación, no simplemente de la instancia. Al examinar la tabla, se ve que, si se proporciona un valor específico de `stuId`, sólo hay un valor de `lastName` asociado con dicho `stuId` particular. Por ejemplo, para el `stuId` S1006, el `lastName` asociado es Lee. Puesto que se entiende que cada estudiante tiene una ID única, se sabe que es una característica de la relación, no sólo de esta instancia, de modo que `{lastName}` es funcionalmente dependiente de `{stuId}` y se puede escribir

```
{stuId} → {lastName}
```

Sin embargo, para un valor dado de `lastName`, puede haber más de un `stuId`. Note que, para el `lastName` Jones, existen dos valores `stuId`, S1003 y S1060. Por tanto, no se puede voltear la dependencia funcional y escribir `{lastName}` → `{stuId}`. Para cada uno de los otros atributos, existe sólo un valor asociado con un valor particular de `stuId`, de modo que todos los atributos son funcionalmente dependientes de `stuId`. Se tiene

```
{stuId}→ {stuId}
{stuId}→ {lastName}
{stuId}→ { major}
{stuId}→ {credits}
{stuId}→ {status}
{stuId}→ {socSecNo}
```

que en forma abreviada se escribirá como

```
{stuId }→ {stuId, lastName, major, credits, status, socSecNo}
```

para indicar que cada uno de los atributos a la derecha es funcionalmente dependiente de `stuId`. De igual modo se tiene

```
{socSecNo} → {socSecNo, stuId, lastName, major, credits, status}
```

Note también que `status` es funcionalmente dependiente de `credits`. Por ejemplo, si se sabe que el valor de `credits` es 15, el `status` de manera automática es freshman. Se escribe

```
{credits} → {status}
```

Para conjuntos que consisten en un solo atributo como éste, se escribirá justo el nombre del atributo y se eliminarán las llaves. Para este caso de dependencia funcional, el determinante, `credits`, **no** determina de manera funcional todos los otros atributos de la relación. Note también que el valor de `credits` no necesariamente es único. Muchos estudiantes podrían tener el mismo número de créditos, de modo que la exclusividad no es una característica necesaria de los determinantes. La instancia en la figura 5.3 en realidad no demuestra la falta de exclusividad de `credits`, lo que muestra que debe tener cuidado al hacer juicios sólo a partir de las instancias. Es necesario concentrarse en los significados de los atributos y sus restricciones al identificar dependencias funcionales. Note que no se tiene la DF `status` → `credits`, pues, por ejemplo, dos alumnos de primer año pueden tener un número diferente de créditos, como se ve en los registros de S1006 y S1060.

Ciertas dependencias funcionales se llaman **triviales** porque siempre se satisfacen en cada relación. En las dependencias funcionales triviales, el dependiente es un subconjunto del determinante. Si todos los atributos en el conjunto del lado derecho están incluidos en el conjunto del lado izquierdo de la dependencia, o si los dos lados son el mismo, la DF es trivial. Por ejemplo, las siguientes DF son triviales:

```
{A,B}→ A
{A,B}→ B
{A,B}→ {A,B}
```

5.4 Superclaves, claves candidatas y claves primarias

Recuerde del capítulo 3 que una **superclave** es un atributo o conjunto de atributos que identifica de manera única una entidad. En una tabla, una superclave es cualquier columna o conjunto de columnas cuyos valores se pueden usar para distinguir una fila de otra. En consecuencia, una superclave es cualquier identificador único. Dado que una superclave identifica de manera única a cada entidad, determina funcionalmente a todos los atributos de una relación. Para la tabla `Student` de la figura 5.3, `{stuId}` es una superclave. Al igual que la combinación de `{stuId,lastName}`. De hecho, `{stuId, cualquier otro atributo}` es una superclave para esta relación. Lo mismo es cierto para `{socSecNo, cualquier otro atributo}`. De hecho, cualquier conjunto de atributos que contengan una superclave también es una superclave. Sin embargo, una superclave puede contener atributos adicionales que no son necesarios para exclusividad, y el interés está en encontrar superclaves que no contengan tales atributos adicionales.

Una **clave candidata** es una superclave tal que ningún subconjunto propio de sus atributos sea por sí mismo una superclave. Por tanto, una clave candidata debe ser un identificador **mínimo**. En el ejemplo, la superclave `{stuId, lastName}` no es una clave candidata porque contiene un subconjunto propio, `{stuId}`, que es una superclave. Sin embargo, `stuId` por sí mismo es una clave candidata. De igual modo, `socSecNo` es una clave candidata. Si a ningún par de estudiantes se le permite tener la misma combinación de valores para nombre y especialidad, la combinación `{lastName,major}` también sería una clave candidata. Por tanto, se ve que una relación puede tener varias claves candidatas. Se usará el término clave compuesta para referir una clave que consista en más de un atributo.

Una **clave primaria** es una clave candidata que en realidad se usa para identificar tuplas en una relación. En el ejemplo, `stuId` puede ser la clave primaria y `socSecNo` una clave alterna. Estos dos atributos se determinan de modo funcional uno a otro, y todos los atributos son funcionalmente dependientes de cada uno de ellos. Cuando se toma una decisión acerca de cuál clave candidata usar como la clave primaria, es importante considerar cuál elección es una mejor representación del mundo real en el que funciona la empresa. Se elige `stuId` en lugar de `socSecNo` porque, dado que la universidad asigna valores `stuId`,

siempre puede estar seguro de tener dicho valor para cada estudiante. Es posible que no se tenga el número de seguridad social de cada estudiante. Por ejemplo, los estudiantes extranjeros pueden no tener números de seguridad social. También existen reglas de privacidad que limitan el uso de los números de seguridad social como identificadores, de modo que es mejor evitar usarlos para la base de datos universidad. Aunque no se establece de manera explícita en la definición, una importante característica de una clave primaria es que ninguno de sus atributos puede tener valores nulos. Si en las claves se permiten valores nulos, sería incapaz de separar los registros, dado que dos registros con valores nulos en el mismo campo de clave pueden ser indistinguibles. Para claves candidatas se especificará que sus valores también son únicos en la base de datos. Esto ayuda a asegurar la calidad de los datos pues se sabe que los valores duplicados serían incorrectos para estos ítems. También es deseable reforzar la regla de "no nulos" para claves candidatas, siempre que pueda asegurar la disponibilidad del valor de la clave candidata.

5.5 El proceso de normalización usando claves primarias

En la práctica, los diseñadores de bases de datos por lo general desarrollan un modelo lógico inicial para una base de datos relacional al mapear un diagrama E-R u otro modelo conceptual a un conjunto de relaciones. Identifican la clave primaria y posiblemente dependencias funcionales durante el proceso de diseño conceptual. Al mapear el diagrama E-R a un modelo relacional, con el método descrito en la sección 4.8, el diseñador crea un modelo relacional que ya está bastante bien normalizado. Para completar el proceso de diseño relacional, el diseñador comprueba cada relación, identifica un conjunto de dependencias funcionales por si existe alguna otra que involucre las claves primarias y las normaliza más si es necesario. Un conjunto de reglas empíricas bien desarrolladas, o heurística, se usa para guiar el proceso de normalización con base en claves primarias. En la siguiente sección se describe este proceso, usando definiciones informales de las formas normales. En secciones posteriores se formalizarán estas nociones.

5.5.1 Primera forma normal

Para describir la primera forma normal se usará un contraejemplo. Si supone que a un estudiante se le permite tener más de una especialidad, y se intenta almacenar especialidades múltiples en el mismo campo del registro del estudiante, la tabla `NewStu` puede parecerse a la de la figura 5.4(a). Este ejemplo viola la definición de la primera forma normal, que es la siguiente.

> **Definición:** Una relación está en la **primera forma normal** (1FN) si y sólo si cada atributo tiene valor sencillo para cada tupla.

Esto significa que cada atributo en cada fila, o cada "celda" de la tabla, contiene sólo un valor. Una forma alternativa de describir la primera forma normal es decir que los dominios de los atributos de la relación son **atómicos**. Esto significa que en el dominio no se permiten conjuntos, listas, campos repetidos o grupos. Los valores en el dominio deben ser valores únicos que no se puedan descomponer más. En la tabla de la figura 5.4(a) se ve la violación de esta regla en los registros de los estudiantes S1006 y S1010, quienes ahora tienen dos valores mencionados para `major`. Éste es el primer ejemplo visto, porque todas las relaciones consideradas hasta ahora estaban en la primera forma normal. De hecho, la mayor parte de la teoría relacional se basa en relaciones en al menos primera forma normal, aunque en la literatura aparecen relaciones que no están en la primera forma normal. Es importante tener primera forma normal de modo que los operadores relacionales, como se les definió, funcionen de manera correcta. Por ejemplo, si realiza la operación del álgebra

relacional SELECT NewStu WHERE major = 'Math' en la tabla de la figura 5.4(a), ¿se debe incluir el registro del estudiante S1006? Si fuera a realizar una combinación natural con alguna otra tabla usando `major` como la columna de combinación, ¿este registro se aparearía sólo con aquellos que tengan tanto CSC como Math como `major`, o con aquellos que tengan o CSC o Math? ¿Qué ocurriría con la combinación si otro registro tuviera una doble especialidad citada como Math CSC? Para evitar estas ambigüedades se insistirá en que toda relación se escriba en la primera forma normal. Si una relación no está ya en 1FN se le puede rescribir y crear una nueva relación que consista en la clave de la relación original, más el atributo multivaluado. Por tanto, la tabla `NewStu` se rescribirá como en la figura 5.4(b) y se creará una nueva tabla `Majors` (especialidades). Note también que la clave de la nueva tabla no es `stuId`, pues cualquier estudiante con más de una especialidad aparece al menos dos veces. Se necesita `{stuId, major}` para identificar de manera única un registro en dicha tabla.

Otro método de normalización a primera forma normal en el caso donde se conoce el número máximo de repeticiones que puede tener un atributo es agregar nuevas columnas

FIGURA 5.4

Ejemplos de la primera forma normal

FIGURA 5.4(a)

Tabla NewStu (suponga que los estudiantes pueden tener doble especialidad)

StuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC Math	15	Freshman	088520876
S1010	Burns	Art English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.4(b)

Tablas NewStu2 y Majors

NewStu2

stuId	lastName	credits	status	socSecNo
S1001	Smith	90	Senior	100429500
S1003	Jones	95	Senior	010124567
S1006	Lee	15	Freshman	088520876
S1010	Burns	63	Junior	099320985
S1060	Jones	25	Freshman	064624738

Majors

stuId	major
S1001	History
S1003	Math
S1006	CSC
S1006	Math
S1010	Art
S1010	English
S1060	CSC

stuId	lastName	major1	major2	credits	status	socSecNo
S1001	Smith	History		90	Senior	100429500
S1003	Jones	Math		95	Senior	010124567
S1006	Lee	CSC	Math	15	Freshman	088520876
S1010	Burns	Art	English	63	Junior	099320985
S1060	Jones	CSC		25	Freshman	064624738

FIGURA 5.4(c)
Tabla newStu3 con dos atributos para major

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1006	Lee	Math	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1010	Burns	English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.4(d)
Tabla NewStu2 rescrita en 1FN, con {stuId,major} como clave primaria

para el atributo. Por ejemplo, si sabe que los estudiantes pueden tener cuando mucho dos especialidades, `NewStu` se rescribiría con dos columnas para la especialidad, `major1` y `major2`, como se muestra en la figura 5.4(c). La desventaja de este enfoque es que usted debe conocer el máximo número de repeticiones, y que las consultas se vuelven más complejas. Para formar una selección en álgebra relacional sobre la especialidad, necesitaría probar tanto la columna `major1` como la `major2`.

Un método alternativo para convertir en 1FN la tabla original es hacer al atributo multivaluado parte de la clave. Con este método, la nueva tabla contendría filas múltiples para estudiantes con múltiples especialidades, como se muestra en la figura 5.4(d). Esta solución puede hacer que surjan dificultades cuando el diseñador intente poner la relación en formas normales superiores.

5.5.2 Dependencia funcional completa y segunda forma normal

Para la relación que se muestra en las figuras 5.1 y 5.5(a), se tienen las siguientes dependencias funcionales además de las triviales:

```
{courseNo,stuId} → {lastName}
{courseNo,stuId} → {facId}
{courseNo,stuId} → {schedule}
{courseNo,stuId} → {room}
{courseNo,stuId} → {grade}
```

Dado que no hay otra clave candidata, se elige `{courseNo,stuId}` para la clave primaria. De nuevo, al ignorar las dependencias funcionales triviales, también se tienen las dependencias funcionales

```
courseNo → facId
courseNo → schedule
courseNo → room
stuId → lastName
```

De modo que se encuentran atributos que son funcionalmente dependientes en la combinación `{courseNo,stuId}`, pero también funcionalmente dependientes en un subconjunto de dicha combinación. Se dice que tales atributos no son **por completo** dependientes funcionales de la combinación.

> **Definición:** En una relación R, el atributo A de R es **completamente dependiente funcional** sobre un atributo o conjunto de atributos X de R si A es funcionalmente dependiente sobre X pero no funcionalmente dependiente sobre cualquier subconjunto propio de X.

En el ejemplo, aunque `lastName` es funcionalmente dependiente sobre `{courseNo, stuId}`, también es funcionalmente dependiente sobre un subconjunto propio de dicha combinación, a saber `stuId`. De igual modo, `facId`, `schedule` y `room` son funcionalmente dependientes sobre el subconjunto propio `courseNo`. Note que `grade` es por completo dependiente funcional sobre la combinación `{courseNo,stuId}`.

> **Definición:** Una relación está en **segunda forma normal** (2FN) si y sólo si está en primera forma normal y todos los atributos no clave son completamente dependientes funcionales sobre la clave.

Claro está, si una relación es 1FN y la clave consiste en un solo atributo, la relación es automáticamente 2FN. Tiene que preocuparse por 2FN sólo cuando la clave sea compuesta. A partir de la definición, se ve que la relación `Class` no está en segunda forma pues, por ejemplo, `lastName` no es completamente dependiente funcional sobre la clave `{courseNo, stuId}`. A pesar de que aquí existen otras dependencias no completamente funcionales, una es suficiente para mostrar que la relación no es 2FN.

Una relación 1FN que no es 2FN se puede transformar en un conjunto equivalente de relaciones 2FN. La transformación se efectúa al realizar **proyecciones** sobre la relación original en tal forma que es posible regresar al original al tomar la combinación de las proyecciones.

FIGURA 5.5

Ejemplo de segunda forma normal

FIGURA 5.5(a)

La tabla NewClass no en 2FN

classNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TUTHF10	M110	A
CSC201A	S1006	Lee	F105	TUTHF10	M110	C
HST205A	S1001	Smith	F202	MWF11	H221	

Register

classNo	stuI	grade
ART103A	S1001	A
ART103A	S1010	
ART103A	S1006	B
CSC201A	S1003	A
CSC201A	S1006	C

Stu

stuId	stuLastName
S1001	Smith
S1010	Burns
S1006	Lee
S1003	Jones

Class2

classNo	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TUTHF10	M110
HST205A	F202	MWF11	H221

FIGURA 5.5(b)

Las tablas Register, Stu y Class2 en 2FN

Las proyecciones de este tipo se llaman **proyecciones sin pérdida** (lossless) y se discutirán con más detalle en la sección 5.6.3. En esencia, para hacer una relación 2FN, identifique cada una de las dependencias no completamente funcionales y forme proyecciones al remover los atributos que dependan de cada uno de los determinantes identificados como tales. Estos determinantes se colocan en relaciones separadas junto con sus atributos dependientes. La relación original todavía contendrá la clave compuesta y cualesquier atributos que sean completamente dependientes funcionales sobre ella. Incluso si no existen atributos completamente dependientes funcionales sobre la clave de la relación original, es importante conservar la relación (incluso sólo con los atributos clave) con la finalidad de poder reconstruir la relación original mediante una combinación. Esta "relación de conexión" muestra cómo se relacionan las proyecciones.

Al aplicar este método al ejemplo primero identifique todas las dependencias funcionales de interés. Por brevedad, se combinarán los atributos que aparezcan como dependientes y tengan el mismo determinante, y se eliminarán las llaves en ambos lados. Es importante notar que, cuando se usa esta notación menos formal, los atributos a la derecha de la flecha se pueden "descomponer" y citar como DF separadas, pero los atributos en el lado izquierdo deben permanecer unidos, pues es su combinación la que es determinante. Las dependencias funcionales son

```
courseNo → facId, schedule, room
stuId → lastName
courseNo,stuId → grade (y, desde luego, facId,schedule,room,lastName)
```

Al usar proyección, se descompone la relación `NewClass` en el siguiente conjunto de relaciones:

```
Register (courseNo, stuId, grade)
Class2 (courseNo, facId, schedule, room)
Stu (stuId, stuLastName)
```

Las relaciones resultantes se muestran en la figura 5.5(b). Note que podría reconstruir la relación original al tomar la combinación natural de estas tres relaciones. Incluso si no hubiera atributo `grade`, necesitaría la relación `Register(courseNo,stuId)` con la finalidad de mostrar cuáles estudiantes están inscritos en cuáles cursos. Sin ella, no podría combinar `Class2` y `Stu`, que no tienen atributos comunes. Al usar estas nuevas relaciones se eliminan las anomalías de actualización, inserción y borrado discutidas anteriormente. Puede cambiar el horario del curso de ART103A al actualizar la columna `schedule` en un solo registro en `Class2`. Puede agregar nueva información de curso a `Class2` sin tener algún estudiante inscrito en el curso, y puede insertar nueva información de estudiante al agregar un registro a `Stu`, sin tener que insertar información de curso para dicho estudiante. De igual modo, puede eliminar el registro de matriculación de `Register` y aun así conservar la información acerca del curso en `Class2` y acerca del estudiante en `Stu`.

5.5.3 Dependencia transitiva y tercera forma normal

Aunque las relaciones de la segunda forma normal son mejores que las de la primera forma normal, todavía pueden tener anomalías de actualización, inserción y borrado. Considere la siguiente relación:

```
NewStudent (stuId, lastName, major, credits, status)
```

La figura 5.6(a) muestra una instancia de esta relación. Aquí, la única clave candidata es `stuId` y se usará como la clave primaria. Todo otro atributo de la relación es funcionalmente dependiente de la clave, así que se tiene la siguiente dependencia funcional, entre otras,

```
stuId → credits
```

Sin embargo, dado que el número de créditos determina el `status`, como se discutió en la sección 5.3, también se tiene

$$\texttt{credits} \rightarrow \texttt{status}$$

Por tanto, `stuId` de manera funcional determina `status` en dos formas, directa y transitivamente, a través del atributo no clave `status`. Al usar transitividad se tiene

$$(\texttt{stuId} \rightarrow \texttt{credits}) \wedge (\texttt{credits} \rightarrow \texttt{status}) \Rightarrow (\texttt{stuId} \rightarrow \texttt{status})$$

> **Definición:** Si A, B y C son atributos de la relación R, tales que A → B y B → C, entonces C es **transitivamente dependiente** de A.

Para la tercera forma normal se quiere eliminar ciertas dependencias transitivas. Las dependencias transitivas causan anomalías de inserción, borrado y actualización. Por ejemplo, en la tabla `NewStudent` de la figura 5.6(a), no se puede insertar la información de que cualquier estudiante con 30 créditos tenga estatus Soph hasta que se tenga a tal estudiante, puesto que eso requeriría insertar un registro sin un `stuId`, lo que no está permitido. Si se borra el registro del único estudiante con cierto número de créditos, se pierde la información acerca del estatus asociado con dichos créditos. Si tiene varios registros con el mismo valor `credits` y cambia el estatus asociado con dicho valor (por ejemplo, hacer que 24 créditos ahora tenga el estatus de Soph), de manera accidental puede fallar al actualizar todos los registros, lo que deja la base de datos en un estado inconsistente. Debido a estos problemas, es deseable remover las dependencias transitivas y crear un conjunto de relaciones que satisfagan la siguiente definición.

> **Definición:** Una relación está en **tercera forma normal** (3FN) si, siempre que exista una dependencia funcional no trivial X → A, entonces o X es una superclave o A es un miembro de alguna clave candidata.

Parafraseando a Kent (vea las referencias), usted puede recordar las características de la tercera forma normal al decir que cada atributo no clave debe depender de la clave, toda la clave y nada más que la clave.

FIGURA 5.6
Ejemplo de tercera forma normal

FIGURA 5.6(a)
Tabla NewStudent no en 3FN

NewStudent

stuId	lastName	major	credits	status
S1001	Smith	History	90	Senior
S1003	Jones	Math	95	Senior
S1006	Lee	CSC	15	Freshman
S1010	Burns	Art	63	Junior
S1060	Jones	CSC	25	Freshman

FIGURA 5.6(b)
Tablas NewStu2 y Stats en 3FN

NewStu2

stuId	lastName	major	credits
S1001	Smith	History	90
S1003	Jones	Math	95
S1006	Lee	CSC	15
S1010	Burns	Art	63
S1060	Jones	CSC	25

Stats

credits	status
15	Freshman
25	Freshman
63	Junior
90	Senior
95	Senior

Al comprobar la tercera forma normal, se busca si algún atributo no clave candidata (o grupo de atributos) es funcionalmente dependiente de otro atributo no clave (o grupo). Si existe tal dependencia funcional, se remueve de la relación el atributo funcionalmente dependiente, y se le coloca en una nueva relación con su determinante. El determinante puede permanecer en la relación original. Para el ejemplo `NewStudent`, dado que la dependencia indeseable es `credits` → `status`, y `status` no es parte de alguna clave candidata, se forma el conjunto de relaciones:

```
NewStu2 (stuId, lastName, major, credits)
Stats (credits, status)
```

Esta descomposición se muestra en la figura 5.6(b). De hecho, se puede decidir no almacenar `status` en la base de datos, y calcular el `status` para aquellas vistas que la necesiten. En este caso, simplemente se elimina la relación `Status`. Este ejemplo no involucra múltiples claves candidatas. Si en la relación original se tiene una segunda clave candidata, `socialSecurityNumber`, se tendría

```
socialSecurityNumber → status
```

pero esto es permisible dado que `socialSecurityNumber` es una superclave para la relación, de modo que se tendría el número de seguridad social en la relación `NewStu2`, junto con `stuId` y los otros atributos ahí.

La definición de la tercera forma normal es la original desarrollada por Codd. Es suficiente para relaciones que tengan una sola clave candidata, pero se descubrió que es deficiente cuando existen múltiples claves candidatas que son compuestas o se traslapan. Por tanto, para considerar todos los casos se formuló una definición mejorada de la tercera forma normal, que lleva el nombre de sus desarrolladores, Boyce y Codd.

5.5.4 Forma normal de Boyce-Codd

La forma normal Boyce-Codd es ligeramente más estricta que 3FN.

> **Definición:** Una relación está en forma Boyce/Codd (FNBC) si, siempre que existe una dependencia funcional no trivial X → A, entonces X es una superclave.

Por tanto, para comprobar la existencia de FNBC, simplemente identifique todos los determinantes y verifique que son superclaves.

Al regresar a los primeros ejemplos para comprobar directamente FNBC se ve que, para la relación `NewStudent` que se muestra en la figura 5.6(a), los determinantes son `stuId` y `credits`. Dado que `credits` no es una superclave, esta relación no es FNBC. Al realizar las proyecciones como se hizo en la sección anterior, las relaciones resultantes son FNBC. Para la relación `NewClass` que se muestra en la figura 5.5(a) se encuentran los determinantes `courseNo`, que (por sí mismo) no es una superclave, y `stuId`, tampoco una superclave. Por tanto, la relación `Class` no es FNBC. Sin embargo, las relaciones que resultan de las proyecciones son FNBC. Considere un ejemplo en el que se tenga 3FN mas no FNBC.

```
NewFac (facName, dept, office, rank, dateHired)
```

Para este ejemplo, que se muestra en la figura 5.7(a), se supondrá que, aunque los nombres del personal docente no son únicos, dos miembros del personal docente dentro de un solo departamento no tienen el mismo nombre. También se supone que cada miembro del personal docente sólo tiene una oficina, identificada en office. Un departamento puede tener varias oficinas de personal docente, y los docentes del mismo departamento pueden compartir oficinas.

FIGURA 5.7

Ejemplo de forma normal Boyce-Codd

FIGURA 5.7(a)

Tabla NewFac en 3FN, mas no FNBC

Faculty				
facName	**dept**	**office**	**rank**	**dateHired**
Adams	Art	A101	Professor	1975
Byrne	Math	M201	Assistant	2000
Davis	Art	A101	Associate	1992
Gordon	Math	M201	Professor	1982
Hughes	Math	M203	Associate	1990
Smith	CSC	C101	Professor	1980
Smith	History	H102	Associate	1990
Tanaka	CSC	C101	Instructor	2001
Vaughn	CSC	C105	Associate	1995

FIGURA 5.7(b)

Fac1 y Fac2 en FNBC

Fac1	
office	**dept**
A101	Art
C101	CSC
C105	CSC
H102	History
M201	Math
M203	Math

Fac2			
facName	**office**	**rank**	**dateHired**
Adams	A101	Professor	1975
Byrne	M201	Assistant	2000
Davis	A101	Associate	1992
Gordon	M201	Professor	1982
Hughes	M203	Associate	1990
Smith	C101	Professor	1980
Smith	H102	Associate	1990
Tanaka	C101	Instructor	2001
Vaughn	C105	Associate	1995

A partir de estas suposiciones se tienen las siguientes DF. De nuevo, se eliminan las llaves y se combinan los dependientes con el mismo determinante.

```
office → dept
facName,dept → office, rank, dateHired
facName,office → dept, rank, dateHired
```

Se tienen claves candidatas que se traslapan de `{facName,dept}` y `{facNam,office}`. Si elige `{facName,dept}` como clave primaria, queda el determinante, `office`, que no es una superclave. Esto viola la FNBC. Note que la relación es 3FN pues `office` es parte de una clave candidata. Para llegar a FNBC puede descomponer la relación `Faculty` mediante proyección en

```
Fac1 (dept, office)
Fac2 (facName, office, rank, dateHired)
```

La clave de la primera relación es `office`, pues un departamento puede tener varias oficinas, mas cada oficina pertenece sólo a un departamento. Claramente es FNBC, pues el único determinante es la clave. La clave de la segunda es `{facName,office}`. También

es FNBC, dado que su único determinante es la clave. Sin embargo, note que el esquema final no preserva la dependencia funcional `facName,dept` → `office,rank, dateHired (fecha de contratación)`, dado que estos atributos no permanecen en la misma relación.

(Nota: Si se hubiese elegido `facName,office` como la clave primaria de la relación `NewFac` original, se tendría `office` → `dept`. Dado que `office` no es una superclave, la relación no sería FNBC. De hecho, no sería 2FN, pues `dept` no sería completamente dependiente funcional sobre la clave, `facName,office`.)

Cualquier relación que no es FNBC se puede descomponer en relaciones FNBC mediante el método recién ilustrado. No obstante, no siempre puede ser deseable transformar la relación en FNBC. En particular, si existe dependencia funcional que no se preserva cuando se realiza la descomposición, entonces se vuelve difícil fortalecer la dependencia funcional en la base de datos pues dos o más tablas tendrían que unirse para verificar que se refuerza, y se pierde una importante restricción. En este caso es preferible asentar 3FN, que siempre permite preservar dependencias. La relación `NewFac` proporciona un ejemplo en el que se pierde una dependencia funcional al normalizar a FNBC. En las relaciones resultantes, los atributos `facName` y `dept` aparecieron en diferentes relaciones, y no se tenía forma de expresar el hecho de que ellas determinaban todos los otros atributos.

5.5.5 Ejemplo comprensivo de dependencias funcionales

Para resumir las varias formas normales definidas mediante dependencias funcionales, considere la siguiente relación que almacena información acerca de proyectos en una gran empresa:

```
Work (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept,
rating)
```

La figura 5.8(a) muestra una instancia de esta relación.

Se hacen las siguientes suposiciones:

1. Cada proyecto tiene un nombre único.
2. Aunque los nombres de proyecto son únicos, los nombres de los empleados y supervisores no lo son.
3. Cada proyecto tiene un supervisor, cuyo nombre se almacena en `projMgr`.
4. Muchos empleados se pueden asignar para trabajar en cada proyecto, y un empleado se puede asignar a más de un proyecto. El atributo `hours` dice el número de horas por semana que un empleado particular se asigna para trabajar en un proyecto particular.
5. `budget` almacena la cantidad presupuestada para un proyecto, y `startDate` da la fecha de inicio para un proyecto.
6. `salary` proporciona el salario anual de un empleado.
7. `empMgr` proporciona el nombre del supervisor del empleado, que puede no ser el mismo que el supervisor del proyecto.
8. `empDept` proporciona el departamento del empleado. Los nombres de departamento son únicos. El supervisor del empleado es el supervisor del departamento del empleado.
9. `rating` proporciona la calificación del empleado para un proyecto particular. El supervisor del proyecto asigna la calificación al final del trabajo del empleado en dicho proyecto.

FIGURA 5.8

Ejemplo de normalización

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/04	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/04	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/04	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/04	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/04	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/04	45000	Jones	15	

FIGURA 5.8(a)

La tabla Work

Proj

projName	projMgr	budget	startDate
Jupiter	Smith	100000	01/15/04
Maxima	Lee	200000	03/01/04

Dept

empDept	empMgr
10	Levine
12	Jones
15	Jones

Emp1

EmpId	empName	salary	empDept
E101	Jones	60000	10
E105	Adams	55000	12
E110	Rivera	43000	10
E120	Tanaka	45000	15

Work1

projName	empId	hours	rating
Jupiter	Jones	25	9
Jupiter	Adams	40	
Jupiter	Rivera	10	8
Maxima	Jones	15	
Maxima	Rivera	30	
Maxima	Tanaka	15	

FIGURA 5.8(b)

Las tablas normalizadas que sustituyen Work

Al usar estas suposiciones se encuentran las siguientes dependencias funcionales para comenzar:

```
projName → projMgr, budget, startDate
empId → empName, salary, empMgr, empDept
projName, empId → hours, rating
```

Dado que se supone que los nombres de las personas no son únicos, `empMgr` no determina funcionalmente `empDept`. [Dos supervisores diferentes pueden tener el mismo nombre y supervisar distintos departamentos, o posiblemente un supervisor puede supervisar varios departamentos, vea a Jones en la figura 5.8(a).] De igual modo, `projMgr` no determina `projName`. Sin embargo, dado que los nombres de departamento son únicos y cada departamento sólo tiene un supervisor, es necesario agregar

```
empDept → empMgr
```

Usted puede preguntarse si `projMgr` → `budget`. Aunque puede ser el caso de que el supervisor determina el presupuesto en el sentido de la palabra, lo cual significa que el supervisor presenta las cifras para el presupuesto, debe recordar que la dependencia funcional no significa causar o calcular. De igual modo, aunque el supervisor del proyecto asigna una calificación al empleado, no hay dependencia funcional entre `projMgr` y `rating`. (Si la hubiera, significaría que cada supervisor siempre da las mismas calificaciones a quienes evalúa. Por ejemplo, significaría que si el nombre del supervisor es Levine, el empleado siempre obtiene una calificación de 9.)

Como se aprecia que cada atributo es funcionalmente dependiente de la combinación `projName`, `empId`, se elegirá dicha combinación como la clave primaria, y vea qué forma normal tiene. Comience por verificar si ya está en FNBC.

FNBC: Observe si existe un determinante que no sea una superclave. Cualesquiera de `empId`, `empDept` o `projName` es suficiente para mostrar que la relación `Work` no es FNBC. Puesto que se sabe que no es FNBC, comience el proceso de normalización por comprobar las formas normales inferiores, y normalice las relaciones conforme avance.

Primera forma normal: Con la clave compuesta, cada celda sería de un solo valor, de modo que `Work` está en 1FN.

Segunda forma normal: Se encuentran dependencias parciales (no completas).

```
projName → projMgr, budget, startDate
empId → empName, salary, empMgr, empDept
```

Puede hacerse cargo de esto, al transformar la relación en un conjunto equivalente de relaciones 2FN mediante proyección, lo que resulta en

```
Proj (projName, projMgr, budget, startDate)
Emp (empId, empName, salary, empMgr, empDept)
Work1 (projName, empId, hours, rating)
```

Tercera forma normal: Con el conjunto de proyecciones, `{Proj,Emp,Work1}`, se prueba cada relación para ver si se tiene 3FN. Al examinar `Proj` se ve que ningún atributo no clave determina funcionalmente otro atributo no clave, así que `Proj` es 3FN. En `Emp` se tiene una dependencia transitiva, pues `empDept` → `empMgr`, como se explicó anteriormente. Como `empDept` no es una superclave, ni `empMgr` es parte de una clave candidata, esto viola 3FN. Por tanto, es necesario rescribir `Emp` como

```
Emp1 (empId, empName, salary, empDept)
Dept (empDept, empMgr)
```

`Work1` no tiene dependencia transitiva que involucre horas o calificación, de modo que la relación ya es 3FN. En consecuencia, el nuevo conjunto de relaciones 3FN es

```
Proj (projName, projMgr, budget, startDate)
Emp1 (empId, empName, salary, empDept)
Dept (empDept, empMgr)
Work1 (projName, empId, hours, rating)
```

Forma normal Boyce-Codd revisada: El nuevo conjunto de relaciones 3FN también es FNBC, pues, en cada relación, el único determinante es la clave primaria.

La figura 5.8(b) muestra las nuevas tablas que sustituyen la tabla `Work` original. Note que se pueden unir para producir exactamente la tabla original.

5.6 Propiedades de las descomposiciones relacionales

Aunque la normalización se puede realizar a partir del uso del enfoque heurístico basado en claves primarias y que se demostró en las secciones anteriores, un enfoque más formal al diseño de bases de datos relacionales se basa estrictamente en dependencias funcionales y otros tipos de restricciones. Este enfoque usa algoritmos de normalización formal para crear esquemas de relación. Para comenzar, todos los atributos en la base de datos se colocan en una sola relación grande llamada **relación universal**. Al usar dependencias funcionales y otras restricciones, la relación universal se descompone en esquemas relacionales más pequeños hasta que el proceso alcanza un punto donde ya no se prefiere más descomposición. Se quiere que los resultados del proceso de descomposición tengan algunas cualidades importantes, si es posible. Es deseable tener cada esquema de relación en FNBC o al menos 3FN. Otras propiedades importantes incluyen preservación de atributo, preservación de dependencia y combinaciones sin pérdida.

5.6.1 Preservación de atributo

Cuando la relación universal se construye contiene, por definición, todo atributo en la base de datos. En el proceso de descomponer la relación universal en relaciones más pequeñas y mover atributos hacia ellas se quiere garantizar que cada atributo aparezca en al menos una de las relaciones, de modo que no se pierden ítem de datos. Como se vio en los ejemplos, el esquema de base de datos usualmente contiene alguna repetición de atributos con la finalidad de representar relaciones entre las tablas. Sin embargo, tienen que colocarse en relaciones en una forma que preserve toda la información, no sólo todos los atributos.

5.6.2 Preservación de dependencia

Una dependencia funcional representa una restricción que se debe reforzar en la base de datos. Siempre que se realice una actualización, el DBMS debe comprobar que no se viola la restricción. Es mucho más fácil verificar las restricciones dentro de una tabla que verificar una que involucre múltiples tablas, lo que requeriría realizar primero una combinación. Para evitar que deban hacerse tales combinaciones se quiere asegurar que, en una descomposición, las dependencias funcionales involucren atributos que estén todos en la misma tabla, si es posible. Dada una descomposición de una relación R, con un conjunto de dependencias funcionales en ella, en un conjunto de relaciones individuales $\{R_1, R_2, \ldots, R_n\}$, para cada dependencia funcional $X \rightarrow Y$ es deseable que todos los atributos en $X \cup Y$ aparezcan en la misma relación, R_i. Esta propiedad se llama **preservación de dependencia**. Siempre es posible encontrar una dependencia que preserve la descomposición que sea 3FN, pero no siempre es posible encontrar una que sea FNBC, como se ilustra en el ejemplo de la figura 5.7 y que se discutió en la sección 5.5.4.

5.6.3 Descomposición sin pérdida

Al dividir las relaciones mediante proyección en los ejemplos anteriores se fue muy explícito acerca del método de descomposición a usar. En particular, se tuvo cuidado al usar proyecciones que pudieran deshacerse al combinar las tablas resultantes, de manera que resultaría la tabla original. Por el término tabla original no se entiende simplemente la estructura de la tabla, esto es, los nombres de columna, sino también las tuplas reales. Tal descomposición se llama **descomposición sin pérdida**, porque conserva toda la información en la relación original. Aunque se usó la palabra descomposición, no se escribió una definición formal, y ahora se hace.

> **Definición:** Una **descomposición** de una relación R es un conjunto de relaciones $\{R_1, R_2, \ldots, R_n\}$ tal que cada R_i es un subconjunto de R y la unión de todas las R_i es R.

Ahora está listo para la definición de descomposición sin pérdida.

> **Definición:** Una descomposición $\{R_1, R_2, \ldots, R_n\}$ de una relación R se llama **descomposición sin pérdida** de R si la combinación natural de $R_1, R_2, \ldots, R_n$ produce exactamente la relación R.

No todas las descomposiciones son sin pérdida, porque existen proyecciones cuya combinación no da de vuelta la relación original. Como ejemplo de una proyección con pérdida, considere la relación `EmpRoleProj (empName,role,projName)` que se muestra en la figura 5.9(a). La tabla muestra cuáles empleados juegan qué papel para cuál proyecto. Se puede descomponer la tabla mediante proyección en las dos tablas, `Table 1` y `Table 2` que se muestran en la figura 5.9(b). Sin embargo, cuando se combinan estas dos tablas, en la figura 5.9(c), se obtiene una tupla adicional que no aparece en la tabla original. Ésta es una tupla **espuria** (falsa), creada por los procesos de proyección y combinación. Puesto que, sin la tabla original, no habría forma de identificar cuáles tuplas son genuinas y cuáles son espurias, en realidad se perdería información (aun cuando se tengan más tuplas) si las proyecciones se sustituyen por la relación original.

Se puede garantizar la descomposición sin pérdida al asegurarse de que, por cada par de relaciones que se combinarán, el conjunto de atributos comunes es una superclave de una de las relaciones. Esto se puede hacer al colocar atributos funcionalmente dependientes en una relación con sus determinantes y conservar los determinantes ellos mismos en la relación original.

De manera más formal, para descomposiciones binarias, si R se descompone en dos relaciones $\{R_1,R_2\}$, entonces la combinación no tiene pérdida si y sólo si alguno de los siguientes se mantiene en el conjunto de DF para R, o se implica mediante las DF en R:

$$R1 \cap R2 \rightarrow R1 - R2$$

o

$$R1 \cap R2 \rightarrow R2 - R1$$

En el ejemplo que se muestra en la figura 5.9, `role` no fue un determinante para `projName` o `empName`, de modo que la intersección de las dos proyecciones, `role`, no determina funcionalmente alguna proyección. Cuando se normalizaron relaciones se vieron muchos ejemplos de proyección sin pérdida.

Para una descomposición que involucra más de dos relaciones, la prueba anterior no se puede usar, así que se presenta un algoritmo para probar el caso general.

Algoritmo para probar la combinación sin pérdida

Dada una relación R(A1, A2, . . . , An), un conjunto de dependencias funcionales, F, y una descomposición de R en las relaciones R1, R2, . . . , Rm, para determinar si la descomposición tiene una combinación sin pérdida

1. *Construya una tabla m por n, S, con una columna para cada uno de los n atributos en R y una fila por cada una de las m relaciones en la descomposición.*
2. *Para cada celda S(i,j) de S,*

   si el atributo para la columna, Aj, está en la relación para la fila, Ri,

   entonces coloque el símbolo a(j) en la celda

   de otro modo coloque ahí el símbolo b(i,j)

FIGURA 5.9

Ejemplo de proyección con pérdida

FIGURA 5.9(a)

Tabla original EmpRoleProj

EmpRoleProj

empName	role	projName
Smith	diseñador	Nilo
Smith	programador	Amazonas
Smith	diseñador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.9(b)

Proyecciones de EmpRoleProj

Table 1

empName	role
Smith	diseñador
Smith	programador
Jones	diseñador

Table 2

role	projName
diseñador	Nilo
programador	Amazonas
diseñador	Amazonas

FIGURA 5.9(c)

Combinación de Table 1 y Table 2

empName	role	projName	
Smith	diseñador	Nilo	
Smith	diseñador	Amazonas	
Smith	programador	Amazonas	
Jones	diseñador	Nilo	←tupla espuria
Jones	diseñador	Amazonas	

3. *Repita el siguiente proceso hasta que ya no se hagan más cambios a S:*

   para cada DF X → Y en F

   para todas las filas en S que tengan los mismos símbolos en las columnas correspondientes a los atributos de X, iguale los símbolos para las columnas que representan atributos de Y mediante la regla siguiente:

   si alguna fila tiene un valor a, a(j), entonces establezca el valor de dicha columna en todas las otras filas igual a a(j)

   si ninguna fila tiene un valor a, entonces escoja cualquiera de los valores b, por decir b(i,j), y establezca todas las otras filas iguales a b(i,j)

4. *si, después de que se han hecho todos los posibles cambios sobre S, una fila está constituida completamente con símbolos a, a(1), a(2), . . . , a(n), entonces la combinación es sin pérdida. Si no hay tal fila, la combinación es con pérdida.*

Ejemplo:

Considere la relación *R(A,B,C,D,E)* que tiene descomposición que consiste en *R1(A,C)*, *R2(A,B,D)* y *R3(D,E)* con DF $A \rightarrow C$, $AB \rightarrow D$ y $D \rightarrow E$. La figura 5.10 ilustra el algoritmo. Con referencia a la figura 5.10(a) construya una fila por cada relación en la descomposición y una columna por cada uno de los cinco atributos de *R*. Para cada fila, coloque el valor *a* con el subíndice de columna en alguna columna cuyo encabezado represente un atributo en dicha relación, y el valor *b* con el subíndice usual de fila y columna en la columna para cualquier atributo que no esté en dicha relación. Por ejemplo, en la primera fila, para la relación

FIGURA 5.10

Prueba para combinación sin pérdida

R(A, B, C, D, E)
Descomposición: R1(A, C), R2(A, B, D), R3(D, E)
DF: A → C, AB → D, D → E

	A	B	C	D	E
R1(A, C)	a(1)	b(1, 2)	a(3)	b(1, 4)	b(1, 5)
R2(A, B, D)	a(1)	a(2)	b(2, 3)	a(4)	b(2, 5)
R3(D, E)	b(3, 1)	b(3, 2)	b(3, 3)	a(4)	a(5)

FIGURA 5.10(a)
Colocación inicial de valores

	A	B	C	D	E
R1(A, C)	a(1)	b(1, 2)	a(3)	b(1, 4)	b(1, 5)
R2(A, B, D)	a(1)	a(2)	a(3)	a(4)	a(5)
R3(D, E)	b(3,1)	b(3, 2)	b(3, 3)	a(4)	a(5)

FIGURA 5.10(b)
Tabla después de considerar todas las DF

R1(A,C), se coloca *a(1)* en la primera columna, para *A*, y *a(3)* en la tercera columna, para *C*. Dado que *B* no aparece en *R1*, se coloca *b(1,2)* en su columna. De igual modo, se coloca *b(1,4)* en la columna *D* y *b(1,5)* en la columna *E*, pues estos atributos no aparecen en *R1*. Ahora considere la DF $A \rightarrow C$, y busque las filas que concuerden con el valor del lado izquierdo, *A*. Se encuentra que las filas 1 y 2 concuerdan con el valor *a(1)*. Por tanto, se pueden igualar los valores *C*. Se encuentra que la fila 1 tiene un valor *a*, *a(3)*, en la columna *C*, de modo que en la figura 5.10(b) el valor de la columna *C* de la fila 2 se iguala a *a(3)*. Al considerar la segunda DF, $AB \rightarrow D$, no se pueden encontrar dos filas que concuerden en sus valores *A* y *B*, de modo que no hay posibilidad de realizar cambios. Ahora, al considerar la DF $D \rightarrow E$ se encuentra que las filas 2 y 3 concuerdan en sus valores *D*, *a(4)*, de modo que se pueden igualar sus valores *E*. Puesto que la fila 3 tiene un valor *E* de *a(5)*, el valor *E* de la fila 2 se cambia a *a(5)*, así como en la figura 5.10(b). Ahora se encuentra que la segunda fila de la figura 5.10(b) tiene toda valores *a*, y se concluye que la proyección tiene la propiedad de combinación sin pérdida.

La prueba para la propiedad de combinación sin pérdida para el caso general es mucho más compleja que la del caso de descomposición binaria. Sin embargo, si las proyecciones se limitan a proyecciones binarias sucesivas, cada una de las cuales es sin pérdida, el resultado final también será sin pérdida. Si se tiene una descomposición D que consiste en el conjunto $\{R_1,R_2\}$, que tiene la propiedad de combinación sin pérdida (confirmada fácilmente mediante la prueba del caso binario), y R_2 a su vez tiene una proyección sin pérdida $\{T_1,T_2\}$ (también probada fácilmente pues es binaria), entonces la descomposición D_2 que consiste en $\{R_1,T_1,T_2\}$ es sin pérdida.

Siempre es posible encontrar una descomposición FNBC que sea sin pérdida. En la sección 5.7.8 se presentará tal algoritmo.

5.7 Diseño relacional formal

5.7.1 Reglas de inferencia: axiomas de Armstrong

Para comenzar con el abordaje más formal de la normalización es necesario un conjunto de axiomas que proporcionen reglas para trabajar con las dependencias funcionales. Las reglas

de inferencia para dependencias funcionales, llamados **axiomas de inferencia** o **axiomas de Armstrong**, en honor a su desarrollador, se pueden usar para encontrar todas las DF lógicamente implicadas por un conjunto de DF. Estas reglas son **sonoras**, lo que significa que son una consecuencia inmediata de la definición de dependencia funcional y que cualquier dependencia funcional que se pueda derivar a partir de un conjunto dado de DF, usándolas, es verdadera. También son **completas**, lo que significa que se pueden usar para derivar toda inferencia válida acerca de las dependencias, de modo que, si una DF particular no se puede derivar a partir de un conjunto dado de DF usando estas reglas, entonces el conjunto dado de DF no implica dicha DF particular.

Sean A, B, C y D subconjuntos de atributos de una relación R. Los siguientes axiomas se sostienen (note que aquí AC significa la unión del conjunto A y el conjunto C):

- **Reflexividad.** Si B es un subconjunto de A, entonces $A \rightarrow B$. Esto también implica que $A \rightarrow A$ siempre se sostiene. Las dependencias funcionales de este tipo se llaman **dependencias funcionales triviales**.
- **Aumento.** Si $A \rightarrow B$, entonces $AC \rightarrow BC$.
- **Transitividad.** Si $A \rightarrow B$ y $B \rightarrow C$, entonces $A \rightarrow C$.

Las siguientes reglas se pueden derivar a partir de las tres anteriores:

- **Aditividad** o unión. Si $A \rightarrow B$ y $A \rightarrow C$, entonces $A \rightarrow BC$.
- **Proyectividad** o descomposición. Si $A \rightarrow BC$, entonces $A \rightarrow B$ y $A \rightarrow C$.
- **Pseudotransitividad.** Si $A \rightarrow B$ y $CB \rightarrow D$, entonces $AC \rightarrow D$.

Estas reglas se pueden usar para desarrollar una teoría formal de las dependencias funcionales, pero en vez de ello el texto se concentrará en sus aplicaciones prácticas.

5.7.2 Clausura de un conjunto de dependencias funcionales

Para la normalización es necesario identificar superclaves, claves candidatas y otros determinantes, lo que se puede hacer si se identifican todas las dependencias funcionales en una relación. También es necesario poder razonar acerca de todas las dependencias funcionales implicadas por un conjunto dado de dependencias funcionales. Hacer esto requiere la noción de la clausura de un conjunto de DF. Si F es un conjunto de dependencias funcionales para una relación R, entonces el conjunto de todas las dependencias funcionales que se pueden derivar de F, F^+, se llama **clausura de F**. Los axiomas de Armstrong son suficientes para calcular todas las F^+; esto es, si se fueran a aplicar estas reglas repetidamente, se encontrarían todas las dependencias funcionales en F^+. Sin embargo, la tarea obviamente sería muy compleja y tomaría mucho tiempo. Las cosas se simplificarían si se pudiera encontrar un conjunto más pequeño de DF que se pudiera usar en lugar de todas las F^+.

5.7.3 Clausura de un atributo

Dado un conjunto de dependencias funcionales F de una relación R, con frecuencia se tiene interés en encontrar todos los atributos en R que son funcionalmente dependientes de cierto atributo o conjunto de atributos, A, en R. A este conjunto de atributos se le llama **clausura de A** o A^+. Claro es, si A^+ es todo de R, entonces A es una superclave para R. A^+ se podría encontrar al calcular todos los F^+ y luego elegir sólo aquellas dependencias funcionales donde A es el determinante, pero hay un atajo. Se puede usar el siguiente algoritmo para encontrar A^+, dado un conjunto F de dependencias funcionales:

Algoritmo de clausura para conjunto de atributos A

resultado ← A;
while (resultado cambia) do
  for each dependencia funcional B → C en F
    if B está contenido en resultado then resultado ← resultado ∪ C;
end;
A+ ← resultado;

Por ejemplo, sea R una relación con atributos W, X, Y, Z y dependencias funcionales

W → Z
{Y,Z} → X
{W,Z} → Y

Calcule {W∪Z}+. Asigne {W∪Z} a `resultado` e ingrese el while por primera vez. Busque una DF cuyo determinante esté contenido en `resultado`, que al momento sólo tiene W∪Z. La DF {W,Z} → Y satisface el requisito, así que se asigna {W∪Z} ∪ Y a `resultado`. Dado que se tiene un cambio en resultado, se ingresa de nuevo el while. Ahora busque una DF donde W, Z, Y o cualquier combinación de estas tres sea un determinante. Puede usar {Y,Z} → X y ahora asigne {{W∪Z} ∪Y)} ∪ X a `resultado`. Puesto que se encontró que todo atributo de R está en {W∪Z}+, W∪Z es una superclave.

Para encontrar W^+ asigne W a `resultado` e ingrese el while por primera vez. La DF W → Z tiene W como determinante, así que asigne W∪Z a `resultado`. Dado que se tiene un cambio a `resultado`, ingrese el while de nuevo. Esta vez use {W,Z} → Y, así que {W∪Z} ∪ Y se asigna a `resultado`. Ahora se busca una DF donde cualquier combinación de los atributos W, Z, Y aparezca como el determinante. Esta vez puede usar {Y,Z} → X, y ahora asigne {{W∪Z}∪Y} ∪ X a `resultado`. Dado que se encontró que todo atributo de R está en W^+, W es una superclave para esta relación. Puesto que no tiene subconjunto propio que sea también una superclave, W también es una clave candidata. Note que ahora se sabe que W∪Z no es una clave candidata.

Es fácil verificar que {Y∪Z} no es una superclave. Comience con `resultado` ←{Y∪Z}. Al usar {Y,Z} → X, `resultado` se convierte en {Y∪Z}∪X. Ahora se busca una DF donde alguna combinación de Y, Z, X es el determinante. Dado que no existe alguna, no se puede agregar algún atributo nuevo a `resultado`. Esto significa que {Y∪Z}+ sólo es {{Y∪Z}∪X}, de modo que W no es funcionalmente dependiente de {Y∪Z}, lo cual significa que {Y∪Z} no es una superclave.

El algoritmo de clausura también permite determinar si existe una dependencia funcional particular en R. Para los conjuntos de atributos A y B, si quiere determinar si A → B, puede calcular A^+ y ver si incluye a B.

5.7.4 Identificación de dependencias funcionales redundantes

Dado un conjunto de dependencias funcionales se desea poder sustituirlas por un conjunto de DF equivalente pero más pequeño. Una forma de hacerlo es determinar si alguna de ellas es **redundante**, lo que significa que se puede derivar de las otras. Para hacerlo, puede usar el siguiente algoritmo:

Algoritmo para determinación de redundancia en un conjunto de DF

1. Elija una DF candidata, por decir X → Y, y remuévala del conjunto de DF.
2. resultado ← X;

```
while (resultado cambia y Y no está contenida en resultado) do
    for each DF, A → B, que permanece en el conjunto reducido de DF
        if A es un subconjunto de resultado, then resultado ← ∪ B
end
```

3. si Y es un subconjunto de resultado, entonces la DF X → Y es redundante.

Entonces se podría remover la DF X → Y del conjunto, pues se puede derivar de las otras DF. Al probar cada DF en el conjunto en turno y remover cualquiera que se pueda derivar de las otras, entonces repetir este proceso con las restantes DF puede encontrar un conjunto no redundante de DF equivalente al conjunto original.

Por ejemplo, suponga que se tienen los siguientes conjuntos de DF:

(1) W → Z
(2) W → Y
(3) {Y,Z} → X
(4) {W,Z} → Y

Comience por probar (1), W → Z. Asigne W a `resultado`. Ahora busque una DF (distinta a (1)) en la que W sea el determinante. Se encuentra una en (2), W → Y. Ahora asigne W ∪ Y a `resultado`. Al buscar una DF que tenga un determinante que esté contenido en W ∪ Y, no se encuentra alguna. Por tanto, no hay posibilidad de demostrar que Z está contenida en `resultado` y se concluye que (1) es no redundante.

A continuación pruebe (2), W → Y. Asigne W a `resultado`. Al buscar una DF distinta a (2) cuyo determinante sea W, se encuentra una en (1), así que se puede asignar W∪Z a `resultado`. Ahora busque una DF cuyo determinante esté contenido en W∪Z. Se encuentra (4), {W,Z} → Y, y ahora se puede asignar {W∪Z}∪Y a `resultado`. Dado que Y ahora está contenida en `resultado`, se puede salir del while y concluir que (2) es redundante. Ahora se elimina (2) del conjunto de DF.

Al probar (3) se asigna Y∪Z a `resultado`. Busque una DF distinta a (3) o (2) (que se eliminó en el paso anterior) cuyo determinante esté contenido en Y∪Z. Al no encontrar alguno se concluye que (3) no es redundante.

Al probar (4) se asigna W∪Z a `resultado`. En (1) se ve que el determinante, W, está contenido en `resultado`, así que se podría agregar el lado derecho de (1), Z, a `resultado`, pero ya está ahí. No hay otra DF cuyo determinante esté contenido en W∪Z, excepto para (2), que se eliminó. Por tanto, se concluye que (4) no es redundante en el conjunto reducido de DF pues no es posible obtener Y en `resultado`. El conjunto final de DF es

(1) W → Z
(3) {Y,Z} → X
(4) {W,Z} → Y

5.7.5 Cubiertas (covers) y conjuntos equivalentes de DF

Si F y G son dos conjuntos de DF para alguna relación R, entonces F es una **cubierta** (cover) para G si toda DF en G también está en F^+. Esto significa que toda DF en G se puede derivar a partir de las DF en F, o que G^+ es un subconjunto de F^+. Para probar que F es una cubierta para G, examine cada DF X → Y en G. Luego calcule X^+ en F y demuestre que X^+ contiene los atributos de Y. Si eso se mantiene verdadero para todas las DF en G, entonces F es una cubierta para G. Para una relación R, dos conjuntos de DF, F y G, se dice que son **equivalentes** si y sólo si F es una cubierta para G y G también es una cubierta para F (es decir, $F^+ = G^+$). Para probar equivalencia, se prueba que F y G son cubiertas una de otra.

Si G es un conjunto de DF en R, y G es mayor, se desea tener la capacidad de encontrar un conjunto más pequeño de DF tal que todas las DF en G también están implicadas por dicho conjunto más pequeño, esto es, el conjunto más pequeño es una cubierta para G.

5.7.6 Conjunto mínimo de dependencias funcionales

Se dice que un conjunto de DF, F, es **mínimo** si satisface estas condiciones:

- El lado derecho de cada DF en F tiene un solo atributo. Esta forma se conoce como estándar o **canónica** para DF.
- Ningún atributo en el lado izquierdo de cualquier DF en F es **extraña**. Esto significa que, si $X \rightarrow Y$ es una DF en F, entonces no hay subconjunto propio S de X tal que $S \rightarrow Y$ se pueda usar en lugar de $X \rightarrow Y$ y el conjunto resultante es equivalente a F.
- F no tiene DF redundantes.

5.7.7 Cómo encontrar una cubierta mínima para un conjunto de DF

Se dice que una cubierta F para G es una **cubierta mínima** (también llamada cubierta **no redundante**) si F es una cubierta para G mas ningún subconjunto propio de F es una cubierta para G. Un conjunto de DF puede tener muchas cubiertas mínimas, pero siempre se puede encontrar una de ellas. Para hacerlo, comience con el conjunto de DF, G. Cada DF en G se expresa en forma canónica, esto es, con un atributo en el lado derecho. Luego se examina el lado izquierdo de cada DF, y se verifica cada atributo A en el lado izquierdo para ver si borrarlo no afecta a G^+. Si el borrado de A no tiene efecto, se borra del lado izquierdo. Esto elimina atributos extraños de todas las DF. A continuación se examina cada DF restante y se verifica para ver si es redundante, esto es, si borrarla no tiene efecto sobre G^+. Si es redundante, se elimina. El conjunto final de DF, que se llamará F, es irreductible y equivalente al conjunto original. A continuación se presenta el algoritmo.

Algoritmo para encontrar una cubierta mínima, F, para un conjunto dado de DF, G

1. *Sea* F $\leftarrow$ *G*;
2. *Para cada DF en* F *que no esté en forma canónica, es decir, de la forma* $X \rightarrow \{Y1, Y2, \ldots Yn\}$, *sustitúyala por las n DF* $X \rightarrow Y1$, $X \rightarrow Y2, \ldots, X \rightarrow Yn$;
3. Elimina *atributos extraños:*

   for each FD $X \rightarrow Y$ *en* F

   for each atributo A *que es un elemento de* X

   if $((F-\{X \rightarrow Y\}) \cup \{(X - \{A\}) \rightarrow Y\}$ *es equivalente a* F

   then sustituya $X \rightarrow Y$ *con* $X \rightarrow Y$ *con* $\{X - \{A\}) \rightarrow Y$ *en* F;
4. Elimine DF *redundantes:*

   for each DF restante $X \rightarrow Y$ *en* F

   si $(F-\{X \rightarrow Y\})$ *es equivalente a* F

   then *remueva* $X \rightarrow Y$ *de* F

5.7.8 Algoritmo de descomposición para la forma normal Boyce-Codd con combinación sin pérdida

Siempre es posible encontrar una descomposición, D, que sea forma normal Boyce-Codd y que tenga la propiedad de combinación sin pérdida. El proceso involucra encontrar cada

violación de FNBC y removerla al descomponer la relación que la contiene en dos relaciones. El proceso se repite hasta que todas tales violaciones se remueven. El algoritmo es

Dada una relación universal R y un conjunto de dependencias funcionales en los atributos de R:

1. D ← R;
2. while existe algún esquema de relación S en D que no sea ya FNBC
   {
   a. Encuentre una dependencia funcional X → Y en S que viole FNBC
   b. Sustituya S con dos esquemas de relación (S-Y) y (X,Y)
   }

5.7.9 Algoritmo de síntesis para descomposición de tercera forma normal

Siempre se puede encontrar una descomposición de tercera forma normal que sea sin pérdida y que preserve dependencias. El algoritmo para la descomposición de la tercera forma normal es:

Dada una relación universal R y un conjunto de dependencias funcionales, G, en R,

1. Encuentre una cubierta mínima F para G, usando el algoritmo dado en la sección 5.7.7.
2. Examine los lados izquierdos de todas las dependencias funcionales. If hay más de una dependencia funcional en F con el mismo lado izquierdo; por ejemplo, existen algún atributo o conjunto de atributos X, y uno o más atributos Ai tales que

   X→ A1, X→ A2, . . ., X→ An;

   then combine X con los atributos en el lado derecho para formar una relación que tenga el esquema R1(X, A1, A2,...,An) en el que X es la clave.

   Repita esto para todos tales determinantes en F.
3. if ninguno de los esquemas de relación resultantes contiene una clave para la relación universal R, then cree una nueva relación que contenga atributos que formen una clave para R.

5.8 Dependencias multivaluadas y cuarta forma normal

Aunque la forma normal Boyce-Codd es suficiente para remover cualquier anomalía debida a dependencias funcionales, mayor investigación por parte de Fagin condujo a la identificación de otro tipo de dependencia que puede causar similares problemas de diseño. Éstas son dependencias multivaluadas. Para ilustrar el concepto de dependencias multivaluadas, considere la siguiente relación:

```
JointAppoint(facId, dept, comité)
```

Aquí se supondrá que un miembro del personal docente puede pertenecer a más de un departamento. Por ejemplo, un profesor puede contratarse conjuntamente por los departamentos CSC y matemáticas. Un docente puede pertenecer a varios comités en toda la universidad, cada uno identificado por el nombre del comité. No existe relación entre departamento y comité. La figura 5.11(a) muestra una versión no normalizada de esta relación. Con la finalidad de hacer la relación 1FN se le debe rescribir de modo que cada celda tenga sólo un valor. Aunque el método preferido es crear una relación separada para cada atributo multivaluado, otro método para crear 1FN es mediante "aplanamiento" de la tabla, como se muestra en la figura 5.11(b). Note que uno está forzado a escribir todas las combinaciones de valores `dept` con valores `comité` para cada miembro docente, o de otro modo parecería que hay alguna relación entre `dept` y `comité`. Por ejemplo, sin la segunda fila, parecería que F101 está en el comité de presupuesto sólo como miembro del departa-

mento CSC, mas no como miembro del departamento Mat. Note también que la clave de la relación es ahora `{facId,dept,comité}`. La relación resultante es FNBC, pues el único determinante es la clave. Aunque se atendieron todas las dependencias funcionales, todavía hay anomalías de actualización, inserción y borrado. Si se quiere actualizar un `comité` al que pertenezca F101 de Presupuesto a Promoción, se necesita hacer dos cambios. Si quiere insertar el registro de un miembro del personal docente que no esté en algún comité, no podrá hacerlo, pues comité es parte de la clave, de modo que en dicha columna no se permiten valores nulos. Ésta es una anomalía de inserción. De igual modo, si F221 deja la membresía en el comité Biblioteca, se pierde el resto de la información almacenada para él, pues no está permitido tener un valor nulo para un atributo de la clave. Anteriormente se encontró que problemas similares eran causados por dependencias funcionales, pero no hay ninguna en este ejemplo, así que es necesario identificar una nueva causa. Aunque un miembro del personal docente no está asociado sólo con un departamento, ciertamente está asociado con un conjunto particular de departamentos. De igual modo, un docente está asociado con un conjunto específico de comités en algún momento dado. El conjunto de departamentos para una `facId` particular es independiente del conjunto de comités para dicho miembro del personal docente. Esta independencia es la causa de los problemas. Para ver cómo se pueden corregir, es necesaria otra definición.

> **Definición:** Sea R una relación que tiene atributos o conjuntos de atributos A, B y C. Existe una **dependencia multivaluada** del atributo B sobre el atributo A si y sólo si el conjunto de valores B asociados con un valor A dado es independiente de los valores C.

Esto se escribe como A —>> B y se lee como A multidetermina B. Si R tiene al menos tres atributos, A, B y C, entonces en R(A,B,C), si A —>> B, entonces A —>> C también.

A diferencia de las reglas para dependencias funcionales, que hacen ilegales ciertas tuplas en las relaciones, las dependencias multivaluadas hacen esenciales ciertas tuplas en una relación. En la tabla `JointAppoint1` normalizada que se muestra en la figura 5.11(b), uno está forzado a escribir ciertas tuplas porque se incluyeron otras. Por ejemplo, cuando se escribió la combinación de F101 con valores de departamento CSC y Mat, tuvieron que escribirse dos tuplas por cada uno de los valores de comité, Presupuesto y Currículum, y colocar cada valor de departamento en una tupla con cada valor de comité. Una definición exacta de la dependencia multivaluada describe las tuplas que deben aparecer.

Definición alternativa de dependencia multivaluada

Más generalmente, si R es una relación con dependencia multivaluada,

```
A —>> B
```

entonces, en cualquier tabla para R, si dos tuplas, t1 y t2, tienen el mismo valor A, entonces deben existir otras dos tuplas t3 y t4 que obedezcan las siguientes reglas:

1. t3 y t4 tienen el mismo valor A que t1 y t2
2. t3 tiene el mismo valor B que t1
3. t4 tiene el mismo valor B que t2
4. si R – B representa los atributos de R que no están en B, entonces t2 y t3 tienen los mismos valores para R – B y
5. t1 y t4 tienen los mismos valores para R – B

La dependencia A —>> B se llama **dependencia multivaluada trivial** si B es un subconjunto de A o A ∪ B es toda de R. Ahora está listo para considerar la cuarta forma normal.

FIGURA 5.11

Ejemplo de cuarta forma normal

FIGURA 5.11(a)

Tabla JointAppoint no en 1FN

JointAppoint

facId	dept	comité
F101	CSC Mat	Currículum
F221	Biología	Biblioteca
F330	Inglés	Admisiones

FIGURA 5.11(b)

Tabla JointAppoint1 en 1FN que muestra dependencias multivaluadas

JointAppoint1

facId	dept	comité
F101	CSC	Presupuesto
F101	Mat	Presupuesto
F101	CSC	Currículum
F101	Mat	Currículum
F221	Biología	Biblioteca
F330	Inglés	Presupuesto
F330	Inglés	Admisiones

FIGURA 5.11(c)

Tablas Appoint1 y Appoint2 en 4FN

Appoint1

facId	dept
F101	CSC
F101	Mat
F221	Biología
F330	Inglés

Appoint2

facId	Comité
F101	Presupuesto
F101	Currículum
F221	Biblioteca
F330	Admisiones
F330	Presupuesto

> **Definición:** Una relación está en **cuarta forma normal** (4FN) si y sólo si es una forma normal Boyce-Codd y no hay dependencias multivaluadas no triviales.

La relación `JointAppoint1` que se muestra en la figura 5.11(b) no está en cuarta forma normal debido a las dos dependencias multivaluadas no triviales,

```
facId —>> dept
facId —>> comité
```

Cuando una relación es FNBC mas no 4FN, se le puede transformar en un conjunto equivalente de relaciones 4FN mediante proyección. Se forman dos relaciones separadas, y en cada una se coloca el atributo que multidetermina a las otras, junto con uno de los atributos multideterminados.

Para la relación `JointAppoint1`, se forman las dos proyecciones,

```
Appoint1 (facId, dept)
Appoint2 (facId, comité)
```

Estas dos relaciones están en 4FN. Se muestran en la figura 5.11(c).

5.9 Descomposición sin pérdida y quinta forma normal

Como se discutió en la sección 5.6.3, no todas las descomposiciones son sin pérdida, porque existen proyecciones cuya combinación no da de vuelta la relación original. Como ejemplo de una proyección con pérdida se usó la relación `EmpRoleProj (empName, role, projName)` que se muestra de nuevo en la figura 5.12(a). La tabla muestra cuáles empleados juegan cuáles papeles para cuáles proyectos. La tabla se puede descomponer mediante proyección en las dos tablas, `Table 1` y `Table 2`, que se muestran en la figura 5.12(b). (Por el momento, ignore la `Table 3`.) Sin embargo, cuando se combinan estas dos tablas, en la figura 5.12(c), se obtiene una tupla espuria adicional que no aparecía en la tabla original y por tanto se pierde información. La tabla original se puede recrear sólo mediante la combinación del resultado con `Table 3`, como se muestra en la figura 5.12(d). La recrea-

FIGURA 5.12
Ejemplo de dependencia de combinación

EmpRoleProj

empName	role	projName
Smith	diseñador	Nilo
Smith	programador	Amazonas
Smith	diseñador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.12(a)
Tabla original EmpRoleProj

Table 1

empName	role
Smith	diseñador
Smith	programador
Jones	diseñador

Table 2

role	proName
diseñador	Nilo
programador	Amazonas
diseñador	Amazonas

Table 3

empName	projName
Smith	Nilo
Smith	Amazonas
Jones	Amazonas

FIGURA 5.12(b)
Proyecciones de EmpRoleProj

empName	role	projName	
Smith	diseñador	Nilo	
Smith	diseñador	Amazonas	
Smith	programador	Amazonas	
Jones	diseñador	Nile	←tupla espuria
Jones	diseñador	Amazonas	

FIGURA 5.12(c)
Primera combinación usando Table 1 y Table 2

empName	role	projName
Smith	diseñador	Nilo
Smith	diseñador	Amazonas
Smith	programador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.12(d)
Combinación de primera combinación con Table3 sobre EmpName, projName

ción depende de esta combinación; esto es: la combinación final es necesaria para obtener de vuelta la tabla. Una **dependencia de combinación** existe cuando, para una relación R con subconjuntos de sus atributos A, B, . . . , Z, R es igual a la combinación de sus proyecciones sobre A, B, . . . , Z.

> **Definición:** Una relación está en **quinta forma normal** si toda dependencia de combinación es implicada por las claves candidatas.

En esencia, esto significa que las únicas descomposiciones válidas son aquellas que involucran claves candidatas. Las dependencias de combinación se relacionan con dependencias multivaluadas, pero pueden ser muy difíciles de identificar porque son sutiles. Si un diseño consiste en relaciones que son todas 5FN, están en su forma útil más simple de modo que no hay nada a ganar al descomponerlas aún más, pues esto resultaría en una pérdida de información. Por desgracia, no hay una prueba simple para 5FN. Se cree que las dependencias de combinación son relativamente raras, de modo que los diseñadores con frecuencia detienen el proceso de normalización en 4FN, FNBC o 3FN (para preservar dependencias funcionales).

5.10 Forma normal dominio-clave

La forma normal final a definir por Fagin involucra los conceptos de dominio, clave y restricción. Fagin demostró que una relación en esta forma no puede tener anomalías de actualización, inserción y borrado. Por tanto, esta forma representa la forma normal final con respecto a estos defectos.

> **Definición:** Una relación está en **forma normal dominio-clave** (FNDC) si toda restricción es una consecuencia lógica de las restricciones de dominio o restricciones de clave.

La definición usa los términos dominio, clave y restricción. Como siempre, el dominio de un atributo es el conjunto de valores permisibles para dicho atributo. Fagin usa la palabra clave para dar a entender lo que se describió como superclave, un identificador único para cada entidad. La restricción es un término general que significa una regla o restricción que se puede verificar al examinar estados estáticos de la base de datos. Entonces, para que exista una restricción, debe ser capaz de establecerla como un predicado lógico, así como determinar si dicho predicado es verdadero o falso al examinar instancias de la relación. Aunque las dependencias funcionales, dependencias multivaluadas y dependencias de combinación son restricciones, también existen otros tipos, llamados restricciones generales. Es posible tener reglas acerca de relaciones entre atributos de una relación (restricciones intrarrelación) que no se expresan como dependencias, o puede haber reglas acerca de relaciones entre relaciones (restricciones interrelación). Por ejemplo, considere la relación Student (`stuId, ..., credits`). Suponga que se tiene una regla de que `stuId` tiene un prefijo que cambia conforme el estudiante avanza; por ejemplo, todo alumno de primer año (freshman) tiene un 1, todos los de segundo año (sophomores) un 2, etc., al comienzo de sus ID. Esto se podría expresar como la restricción general

- Si el primer dígito de `stuId` es 1, entonces `credits` debe estar entre 0 y 30.
- Si el primer dígito de `stuId` es 2, . . .
- . . .

Un ejemplo familiar de una restricción interrelación resulta una restricción de integridad referencial. Considere `Student (stuId, lastName, ...)` y `Enroll (stuId, courseNo, grade)`. Aquí, una restricción interrelación es

- Para cada tupla en Enroll debe haber una tupla en `Student` con el mismo `stuId`.

Para que una relación sea FNDC, las restricciones intrarrelación deben ser expresables como restricciones de dominio o restricciones de clave. La restricción sobre `Student` se podría expresar al dividir `Student` en cuatro relaciones diferentes. Por ejemplo, para alumnos de primer año, se puede tener

`Stu1(stuId, . . ., credits)` con las restricciones de dominio

- `stuId` debe comenzar con un 1
- `credits` debe estar entre 0 y 30.

Entonces se tendría `STU2` con restricciones de dominio similares para estudiantes de segundo año, `STU3` para los de tercer año y `STU4` para los de cuarto año. Para la restricción interrelación, tendría que restringir el dominio de `stuId` en `Student` a los valores realmente representados en `Enroll`. Sin embargo, la definición de Fagin no se extiende a restricciones interrelación, pues su objetivo fue definir una forma que permitiera la comprobación de restricciones generales dentro de una relación al verificar sólo las restricciones de dominio y clave de dicha relación. Puesto que una restricción interrelación como una restricción de integridad referencial involucra dos relaciones, no se considera una FNDC.

Desafortunadamente, aunque el concepto de forma normal dominio-clave es simple, no existe un método probado de convertir un diseño a esta forma.

5.11 El proceso de normalización

Como se afirmó al comienzo del capítulo, el objetivo de la normalización es encontrar un conjunto estable de relaciones que sea un modelo fiel de la empresa. Se encontró que la normalización elimina algunos problemas de la representación de datos y resulta en un buen esquema para la base de datos. Existen dos procesos diferentes que se podrían usar para desarrollar el conjunto de relaciones normalizadas, llamado análisis y síntesis. La mayoría de los diseñadores en vez de ello eligen comenzar con un diagrama entidad-relación y trabajar desde ahí.

5.11.1 Análisis

El abordaje de análisis o descomposición comienza con una lista de todos los atributos a representar en la base de datos y supone que todos ellos están en una relación sencilla llamada relación universal para la base de datos. Entonces el diseñador identifica dependencias funcionales entre los atributos y usa las técnicas de descomposición explicadas en este capítulo para dividir la relación universal en un conjunto de relaciones normalizadas. Los ejemplos de normalización en la sección 5.5 fueron todos ejemplos de descomposición. En particular, en la sección 5.5.5 se consideró un ejemplo en el que se tenía una relación sencilla llamada `Work`. En realidad, `Work` es una relación universal, pues todos los atributos a almacenar en la base de datos estaban en ella. Luego se escribieron las suposiciones y se identificaron cuatro dependencias funcionales. Las dependencias funcionales permitieron la realización de proyecciones sin pérdida, de modo que se desarrolló un conjunto de relaciones en FNBC que preservó todas las dependencias funcionales. Dado que no existían dependencias multivaluadas que estuvieran también en 4FN, y dado que no había proyecciones sin pérdida no triviales restantes, están en 5FN. Todas las restricciones expresadas en la lista de suposiciones y todas las dependencias identificadas en la discusión se consideraron en el desarrollo del conjunto de relaciones final. Las restricciones se representan como restricciones de clave en las relaciones resultantes, de modo que en realidad se tiene un conjunto de relaciones FNDC. Este ejemplo ilustra el proceso de análisis a partir de una relación universal. El algoritmo dado en la sección 5.7.8 para encontrar una descomposición FNBC es un algoritmo de análisis.

5.11.2 Síntesis

La síntesis es, en un sentido, el opuesto de análisis. En el análisis se comienza con una sola relación grande y se descompone hasta que alcanza un conjunto de relaciones normalizadas más pequeñas. En la síntesis se comienza con atributos y se les combina en grupos relacionados, usando dependencias funcionales para desarrollar un conjunto de relaciones normalizadas. Si el análisis es un proceso arriba-abajo, entonces la síntesis es abajo-arriba. El algoritmo que se presentó en la sección 5.7.9 para crear un conjunto de relaciones 3FN es un algoritmo de síntesis.

5.11.3 Normalización desde un diagrama entidad-relación

En la práctica, cuando existe un diagrama E-R completo, el proceso de normalización es significativamente más eficiente que o el análisis puro o la síntesis pura. Al usar las técnicas de mapeo descritas en la sección 4.8, el conjunto de tablas resultante por lo general está bastante bien normalizado. El diseñador puede comenzar por comprobar cada relación para ver si es FNBC. Si no lo es, se pueden examinar los determinantes que causan los problemas y la relación normalizada. Este proceso se ilustra en el proyecto de muestra para la Galería de Arte al final de este capítulo.

5.12 Cuándo detener la normalización

Sin importar el proceso utilizado, el resultado final debe ser un conjunto de relaciones normalizadas que preserven las dependencias y formen combinaciones sin pérdida sobre atributos comunes. Una pregunta importante es cuán lejos ir en el proceso de normalización. Idealmente, se intenta llegar a FNDC. Sin embargo, si eso resulta en una descomposición que no conserva dependencias, entonces se conforma con pérdidas. De igual modo, si intenta 5FN, 4FN o FNBC, es posible que no sea capaz de obtener una descomposición que conserve dependencias, así que uno se conformaría con 3FN en este caso. Siempre es posible encontrar una dependencia que preserve la descomposición para 3FN. Incluso así, puede haber razones válidas para no elegir también implementar 3FN. Por ejemplo, si tiene atributos que casi siempre se usen juntos en aplicaciones y terminen en diferentes relaciones, entonces siempre tendrá que hacer una operación de combinación cuando se les recupere. Un ejemplo familiar de esto ocurre al almacenar direcciones. Suponga que almacena nombre y dirección de un empleado en la relación

```
Emp(empId, apellido, nombre, calle, ciudad, estado, cp)
```

Como siempre, se supone que los nombres no son únicos. Se tiene la dependencia funcional

```
cp → ciudad,estado
```

que significa que la relación no es 3FN. Se le podría normalizar mediante descomposición en

```
Empl(empId,nombre,calle,cp)
Códigos(cp,ciudad,estado)
```

Sin embargo, esto significa que tendría que hacer una combinación siempre que quiera una dirección completa para una persona. En este caso, puede conformarse con 2FN e implementar la relación Emp original. En general, se deben tomar en cuenta los requisitos de desempeño para decidir cuál será la forma final. El proyecto de muestra de la Galería de Arte ilustra las negociaciones en la normalización.

5.13 Resumen del capítulo

Este capítulo trata con un método para desarrollar un conjunto adecuado de relaciones para el modelo lógico de una base de datos relacional. Se encontraron tres tipos de dependencias, **dependencias funcionales**, **dependencias multivaluadas** y **dependencias de combinación** para producir problemas en la representación de información en la base de datos. Muchos problemas involucraban **anomalías de actualización**, **inserción** y **borrado**. Se dice que un conjunto de atributo B es **funcionalmente dependiente** sobre un conjunto de atributos A en una relación R si cada valor A tiene exactamente un valor B asociado con él.

La normalización con frecuencia se realiza comenzando con un conjunto de relaciones diseñadas a partir del mapeo de un diagrama E-R y el uso de métodos heurísticos para crear un conjunto equivalente de relaciones en una forma normal superior. La **primera forma normal** significa que una relación no tiene atributos de valor múltiple. Una relación que no esté ya en 1FN se puede normalizar al crear una nueva tabla que consiste en la clave y el atributo multivaluado, al agregar columnas adicionales para el número esperado de repeticiones, o mediante "aplanamiento", al hacer el atributo multivaluado parte de la clave. Las relaciones en 1FN pueden tener anomalías de actualización, inserción y borrado, debido a **dependencias parciales** sobre la clave. Si un atributo no clave no es **completamente dependiente funcional** sobre toda la clave, la relación no está en **segunda forma normal**. En este caso se le normaliza mediante proyección al colocar cada determinante que sea un subconjunto propio de la clave en una nueva relación con sus atributos dependientes. La clave compuesta original también debe mantenerse en otra relación. Las relaciones en 2FN todavía pueden tener anomalías de actualización, inserción y borrado. Las **dependencias transitivas**, en las que uno o más atributos no clave determinan funcionalmente otro atributo no clave, causan tales anomalías. La **tercera forma normal** se logra al eliminar las dependencias transitivas usando proyección. El determinante que causa la dependencia transitiva se coloca en una relación separada con los atributos que determina. También se mantiene en la relación original. En la tercera forma normal, cada atributo no clave es funcionalmente dependiente de la clave, toda la clave y nada más que la clave. La **forma normal Boyce-Codd** requiere que cada determinante sea una superclave. Esta forma también se logra mediante proyección, pero en algunos casos la proyección separará los determinantes de sus atributos funcionalmente dependientes, lo que resulta en la pérdida de una restricción importante. Si esto ocurre, es preferible 3FN.

Tres propiedades deseables para las descomposiciones relacionales son **preservación de atributo**, **preservación de dependencia** y **descomposición sin pérdida**. Las proyecciones con pérdida pueden perder información porque, cuando se deshacen mediante una combinación, pueden resultar tuplas espurias. Para proyecciones binarias, puede garantizar que una descomposición es sin pérdida si la intersección de las dos proyecciones es un determinante para una de ellas. Para descomposiciones binarias, existe una prueba simple para la descomposición sin pérdida. También existe un algoritmo para probar la descomposición sin pérdida en el caso general. Sin embargo, si la descomposición se hace mediante repetidas descomposiciones binarias, la prueba más simple se puede aplicar a cada descomposición sucesiva. Las dependencias funcionales se pueden manipular formalmente con el uso de los **axiomas de Armstrong** o reglas de inferencia. El conjunto F^+ de todas las dependencias funcionales implicadas lógicamente por un conjunto dado F de dependencias funcionales se llama su **clausura**. La clausura se puede construir mediante la aplicación repetida de los axiomas de Armstrong, pero es un procedimiento largo. De igual modo, la clausura A^+ de un atributo A en un conjunto de dependencias funcionales es el conjunto de todos los atributos que A determina funcionalmente. Se puede calcular A^+ mediante un algoritmo simple. El algoritmo para clausura de atributo se puede usar para responder algunas preguntas muy importantes, como si A es una superclave, si A es una clave candidata y si una

dependencia funcional aparece en F^+. Para un conjunto de DF, F, existe un algoritmo simple para determinar si una DF particular es **redundante** en el conjunto, lo que significa que se puede derivar a partir de las otras DF en F. Si F y G son dos conjuntos de DF para alguna relación R, entonces F es una **cubierta** para G si cada DF en G también está en F^+. Si F es una cubierta para G y G también es una cubierta para F, entonces se dice que F y G son **equivalentes**, y $F^+ = G^+$. Un conjunto de dependencias funcionales es **mínimo** si se escribe en forma canónica (es decir, toda DF tiene sólo un atributo en el lado derecho), ningún atributo de un determinante es extraño y no hay DF redundantes. Se dice que una cubierta F para G es una **cubierta mínima** (cubierta **no redundante**) si F es una cubierta para G mas ningún subconjunto propio de F es una cubierta para G. Un conjunto de DF puede tener varias cubiertas mínimas, pero siempre es posible encontrar una de ellas mediante un algoritmo simple. Si una relación no es ya FNBC, se puede usar un algoritmo de descomposición para encontrar un conjunto equivalente de relaciones FNBC que forman una proyección sin pérdida. No siempre es posible encontrar una descomposición FNBC que también preserve dependencias. Sin embargo, siempre es posible encontrar una descomposición 3FN que sea tanto sin pérdida como conservadora de la dependencia. Para hacerlo existe un algoritmo de síntesis que comienza con una cubierta mínima para el conjunto de DF.

Las **dependencias multivaluadas** pueden ocurrir cuando existen tres atributos en una clave de relación, y dos de ellas tienen valores múltiples independientes para cada valor distinto del tercero. La cuarta forma normal requiere que una relación sea FNBC y no tenga dependencias multivaluadas. Se puede lograr 4FN mediante proyección, pero usualmente se hace sólo si las proyecciones resultantes preservan dependencias funcionales. Una relación está en quinta forma normal si no hay proyecciones sin pérdida no triviales restantes. De manera alternativa, una relación está en quinta forma normal si cada dependencia de combinación es implicada por las claves candidatas. No hay un método probado para lograr la quinta forma normal. La **forma normal dominio-clave** requiere que toda restricción sea consecuencia de restricciones de dominio o restricciones de clave. No existe método probado para producir forma normal dominio-clave.

En análisis se comienza con una relación, se identifican dependencias y se usa proyección para lograr una forma normal superior. El enfoque opuesto, llamado síntesis, comienza con atributos, encuentra dependencias funcionales y agrupa dependencias funcionales con el mismo determinante, y forma relaciones.

Al decidir cuál forma normal elegir para la implementación, considere la preservación de atributo, la proyección sin pérdida y la preservación de dependencia. Por lo general se elige la forma normal superior que permite todo esto. En la implementación también debe equilibrar desempeño contra normalización, así que a veces se acepta un diseño que está en una forma normal inferior por razones de desempeño.

Ejercicios

5.1 Considere la siguiente relación universal que conserva información acerca de libros en una librería:

```
Libros(titulo, isbn, autor, nombreEditorial, direcEditorial, totalCopiasOrdenadas,
copiasEnStock, fechaPublicacion, categoria, precioVenta, costo)
```

Suponga:

- El `isbn` identifica de manera única a un libro. (Sin embargo, no identifica cada copia del libro.)
- Un libro puede tener más de un autor.

- Un autor puede tener más de un libro.
- Cada nombre de editorial es único. Cada editorial tiene una dirección única: la dirección de las oficinas centrales de la firma.
- Los títulos no son únicos.
- `totalCopiasOrdenadas` es el número de copias de un libro particular que la librería solicitó alguna vez, mientras que `copiasEnStock` es el número que todavía no se vende en la librería.
- Cada libro tiene sólo una fecha de publicación. A la revisión de un libro se le da un nuevo ISBN.
- La categoría puede ser biografía, ciencia ficción, poesía, etc. El título solo no es suficiente para determinar la categoría. El `precioVenta`, que es la cantidad que carga la librería por un libro siempre está 20% arriba del costo, que es la cantidad que la librería paga a la editorial o distribuidor por el libro.

  a. Con estas suposiciones, y el establecimiento de las que considere necesarias, mencione todas las dependencias funcionales no triviales para esta relación.

  b. ¿Cuáles son las claves candidatas para esta relación? Identifique la clave primaria.

  c. ¿La relación está en tercera forma normal? Si no, encuentre una 3FN mediante descomposición de combinación sin pérdida de Libros que preserve dependencias.

  d. ¿La relación o conjunto resultante de relaciones está en forma normal Boyce-Codd? Si no, encuentre una descomposición por combinación sin pérdida que esté en FNBC. Identifique cualesquiera dependencias funcionales que no se conserven.

5.2 Considere la siguiente relación que almacena información acerca de estudiantes que viven en dormitorios en una universidad:

```
Universidad(apellido, stuId, dirCasa, telCasa, habitacNum, nombreCompañero, dirDorm,
status, planComida, cargoHabitac, cargoPlanComida)
```

Suponga:

- A cada estudiante se le asigna un número de habitación y puede tener varios compañeros de habitación.
- Los nombres de los estudiantes no son únicos.
- La universidad tiene varios dormitorios. `habitacNum` contiene un código para el dormitorio y el número de la habitación particular asignada al estudiante. Por ejemplo, A221 significa Adams Hall, habitación 221. Los nombres de los dormitorios son únicos.
- `dirDorm` es la dirección del edificio de dormitorios. Cada edificio tiene su propia dirección única. Por ejemplo, Adams Hall puede estar en 123 Main Street, Anytown, NY 10001.
- `status` menciona el estatus del estudiante: freshman (primer año), sophomore (segundo año), junior (tercer año), senior (cuarto año) o graduado.
- `planComida` menciona cuántas comidas por semana eligió el estudiante como parte de su plan alimenticio. Cada plan alimenticio tiene un solo `cargoPlanComida` asociado con él.
- `cargoHabitac` es diferente para distintos dormitorios, pero todos los estudiantes en el mismo dormitorio pagan la misma cantidad.

Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

5.3 Considere los siguientes atributos para tablas en un modelo relacional diseñado para seguir la información de una compañía de mudanza que mueve clientes residenciales, usualmente de una casa o departamento a otro:

`clienteId, clienteNombre, clienteDirActual, clienteTelActual, clienteNuevaDir, clienteNuevoTel, ubicacionLevanta, ubicacionDeja, fechaDeMudanza, horaInicio, pesoEstimado, costoEstimado, camion#Asignado, conductorNombre, conductorLicNo, costoReal, cantidadDeDaños, capacidadCamion, costoCasetas, impuestos, cantidadFinal, facturaNumero, cantidadPagada, fechaPago, chequeNumero, saldo`

Suponga:

- Aunque en la mayoría de los casos la `ubicacionLevanta` es la antigua dirección del cliente y la `ubicacionDeja` es la nueva dirección, existen excepciones, como cuando el mobiliario se mueve hacia o desde un almacén.
- Antes de la mudanza se proporciona una estimación usando una factura preimpresa que contiene un número de factura único. El costo real se registra en la misma forma una vez que la mudanza está completa. El costo real puede diferir del costo estimado. La cantidad final incluye el costo real de la mudanza, más impuestos y casetas.
- En la mayoría de los casos, el cliente paga la cantidad final de manera inmediata, pero en ocasiones paga sólo parte de la cuenta, en especial si la cantidad de daños es elevada. En dichos casos, existe una cantidad saldo, que es la diferencia entre la cantidad final y la cantidad pagada.

(a)-(d) Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

(e) ¿Cuáles tablas implementaría realmente? Explique cualquier desnormalización, omisión o adición de atributos.

5.4 El propietario de una compañía de alimentos y bebidas quiere una base de datos para seguir la huella de varios aspectos del negocio. Los clientes pueden ser individuos o empresas que hagan arreglos para eventos como bodas, bailes, cenas corporativas, recaudación de fondos, etc. La compañía es propietaria de un salón en el que sólo puede desarrollarse un evento a la vez. Además, los clientes pueden usar los servicios de la compañía para realizar eventos en sus hogares, centros de trabajo o lugares que el cliente renta, como mansiones históricas. Muchos de estos eventos fuera del salón tienen lugar al mismo tiempo. Los clientes que rentan el salón deben garantizar un mínimo de 150 invitados, pero el salón puede acomodar hasta a 200 personas. Las reservaciones para el salón se aceptan hasta con dos años de anticipación. Los clientes que proporcionen su propio espacio pueden tener cualquier número de invitados. Las reservaciones para los eventos externos por lo general se realizan con muchos meses de anticipación. La firma proporciona la comida, vajilla (platos, cubertería y vasos), mantelería, meseros y barman para el evento, sin importar si es en el salón o en alguna otra parte. Los clientes pueden elegir el color de la mantelería. La firma tiene menúes preparados que se identifican por número. Para un número de menú dado, existe un aperitivo, una ensalada, un plato principal y un postre. (Por ejemplo, el menú número 10 puede incluir coctel de camarones, ensalada César, prime rib y *mousse* de chocolate.) La firma cita un precio total por sus servicios, con base en el número de invitados, el menú, la ubicación y factores intangibles como cuán ocupados están en dicha fecha. La firma también puede contratar arreglos florales, músicos, animadores y fotógrafos, si el cliente lo solicita. El cliente paga a la compañía, que entonces paga al contratista. El precio que se carga al cliente por cada uno de éstos siempre está 10% arriba del costo a la compañía. Suponga que los nombres son

únicos. Suponga también que los músicos y animadores proporcionan sólo un tipo de música o entretenimiento.

Suponga que se identificaron los siguientes atributos:

clienteApellido, cNombre, cTel, cCalle, cCiudad, cEstado, cCp, eventoFecha, eventoHoraInicio, eventoDuracion, eventoTipo, numeroInvitados, ubicacionNombre, ubicacionCalle, ubicacionCiudad, ubicacionEstado, ubicacionCp, mantelColorSolicita, numeroMeseros, numeroBarman, precioTotal, floristaNombre, floristaTel, floristaCosto, floristaPrecio, musicaContacto, musicaTelContacto, musicaTipo, musicaCosto, musicaPrecio, animadorNombre, animadorTel, animadorTipo, animadorCosto, animadorPrecio, fotografoNombre, fotografoTel, fotografoCosto, fotografoPrecio, menuNumeroElegido, menuAperitivo, menuEnsalada, menuPrincipal, menuPostre.

(a)-(d) Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

(e) ¿Cuáles tablas implementaría realmente? Explique cualquier desnormalización, omisión o adición de atributos.

5.5 Considere las siguientes relaciones e identifique la forma normal más alta para cada una, como se proporcionan, y enuncie cualquier suposición que necesite realizar.

a. Work1 (empId, empName, dateHired, jobTitle, jobLevel)

b. Work2 (empId, empName, jobTitle, ratingDate, raterName, rating)

c. Work3 (empId, empName, projectNo, projectName, projBudget, empManager, hoursAssigned)

d. Work4 (empId, empName, schoolAttended, degreeAwarded, graduationDate)

e. Work5 (empId, empName, socialSecurityNumber, dependentName, dependentAddress, relationToEmp)

5.6 Para cada una de las relaciones del ejercicio 5.5 identifique una clave primaria y,

a. si la relación no está en tercera forma normal, encuentre una 3FN por descomposición de combinación sin pérdida que preserve dependencias.

b. si la relación o conjunto de relaciones resultante no está en forma normal Boyce-Codd, encuentre una descomposición por combinación sin pérdida que esté en FNBC. Identifique cualquier dependencia funcional que no se conserve.

5.7 Considere instancias de relación R(A, B, C, D) como se muestra en la figura 5.13. Para cada uno de los siguientes conjuntos de DF, determine si cada una de las instancias es consistente con las DF dadas:

a. $A \rightarrow B,C$
   $B \rightarrow D$

b. $AB \rightarrow C$
   $B \rightarrow D$

c. $AB \rightarrow CD$
   $AC \rightarrow B$

5.8 Dado el siguiente conjunto S de DF:

$S = \{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\}$

a. Encuentre la clausura de A, A^+.

b. ¿A es una superclave? Explique.

c. ¿B es una superclave? ¿Cómo lo sabe?

d. La DF $B \rightarrow D$, ¿está en S^+? ¿Cómo lo sabe?

e. Encuentre la clausura del conjunto de DF, S^+.

FIGURA 5.13
Instancias de R(A,B,C,D)

Instancia 1

A	B	C	D
a1	b1	c1	d1
a2	b2	c1	d2
a3	b1	c2	d3
a4	b3	c2	d3

Instancia 2

A	B	C	D
a1	b1	c1	d1
a2	b1	c2	d1
a3	b2	c1	d2
a4	b2	c2	d2

Instancia 3

A	B	C	D
a1	b1	c1	d1
a2	b1	c2	d1
a1	b2	c3	d2
a2	b2	c4	d2

5.9 Examine cada uno de los siguientes conjuntos de DF y encuentre cualquier DF redundante en cada uno. Proporcione una cobertura mínima para cada uno.

a. B → D
E → C
AC → D
CD → A
BE → A

b. A → CDE
B → CE
AD → E
CD → F
BD → A
CED → ABD

c. D → C
AB → C
AD → B
BD → A
AC → B

5.10 Considere la relación

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

con DF

A → B
BC → D
D → BC
C → A

a. Identifique las claves candidatas de esta relación.

b. Suponga que la relación se descompone en

R1 (A, B)

R2 (B, C, D)

¿Esta descomposición tiene una combinación sin pérdida?

PROYECTO DE MUESTRA: NORMALIZACIÓN DEL MODELO RELACIONAL PARA LA GALERÍA DE ARTE

En la sección de proyecto de muestra al final del capítulo 3 se creó un diagrama E-R para la Galería de Arte. Al final del capítulo 4 se vio cómo mapear dicho diagrama a tablas. En la presente sección se quiere normalizar las relaciones. Aunque el proceso de normalización podría comenzar por elaborar una sola relación universal que mencione todos los atributos

y luego intentar aislar las dependencias funcionales entre los atributos, el trabajo se puede hacer más sencillo al usar las entidades y relaciones identificadas en el capítulo 3 y las tablas a las que se mapearon al final del capítulo 4. En la práctica, las tablas que resultan de tales mapeos por lo general ya casi están normalizadas. Por tanto, use las tablas mapeadas como punto de partida.

- Paso 5.1. Comience con la lista de las tablas a las que las entidades y relaciones del diagrama E-R se mapearon de manera natural, a partir de la sección del proyecto de muestra al final del capítulo 4. Para cada tabla en la lista, identifique las dependencias funcionales y normalice la relación a FNBC. Luego decida si las tablas resultantes se deben implementar en dicha forma. Si no, explique por qué.

Del mapeo resultaron las siguientes tablas:

Artist(firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

PotentialCustomer(firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName,* preferredMedium, preferredStyle, preferredType)

Artwork(*artistLastName, artistFirstName,* workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

ShownIn(*artistLastName, artistFirstName, workTitle, showTitle*)

Collector(socialSecurityNumber, firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, *collectionArtistFirstName, collectionArtistLastName,* collectionMedium, collectionStyle, collectionType, salesLastYear, salesYearToDate)

Show(showTitle, *showFeaturedArtistLastName, showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

Sale(invoiceNumber, *artistLastName, artistFirstName, workTitle,* amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, saleTotal, *buyerLastName, buyerFirstName, salespersonSocialSecurityNumber*)

Buyer(firstName, lastName, street, city, state, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

Salesperson(socialSecurityNumber, firstName, lastName, street, city, state, zip)

Para la tabla `Artist`, identifique las DF.

firstName + lastName → todos los atributos

Parecería que

socialSecurityNumber → todos los atributos

Recuerde que FNBC permite a los determinantes que son claves candidatas permanecer en la tabla, así que no hay problema con dejar `socialSecurityNumber` en la tabla. Sin embargo, recuerde que se supuso que no siempre se tendría el registro federal de contribuyentes de los artistas cuyas obras fueran propiedad de coleccionistas, así que el valor de este atributo puede ser nulo para algunos artistas. No obstante, cuando aparece, es único en la tabla.

zip → city, state

Es posible que quiera considerar

areaCode → ? city, state

Se concluye que, con teléfonos celulares, el código de área no necesariamente está asociado con la ciudad y el estado de residencia o estudio del artista, de modo que ésta no se tiene como dependencia funcional. También considere si el número telefónico completo determina la dirección.

```
areaCode + telephoneNumber → ? street, city, state, zip
```

¿Es posible que dos artistas registrados con el mismo número telefónico tengan dos direcciones diferentes? Si el teléfono es de una casa o estudio, las direcciones deben ser las mismas (la dirección de la ubicación del teléfono). Si es un teléfono celular, los dos artistas tendrían que compartir el mismo número para aparecer en dos registros diferentes, de modo que es probable que también compartan la misma dirección en dicho caso. Esto se tendrá como una dependencia funcional. Entonces se puede preguntar

```
areaCode + telephoneNumber → ? todos los atributos
```

Se decide que éste no es el caso, pues se considera la posibilidad de que dos artistas puedan compartir el mismo hogar o estudio y el mismo número telefónico ahí, pero aún así tener diferentes nombres, estilos, etc. La tabla está en 1FN y 2FN, mas no en 3FN o FNBC, debido a las dependencias mencionadas. Por tanto, la tabla Artist se descompone del modo siguiente:

```
Artist1(firstName, lastName, interviewDate, interviewerName, areaCode,
telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber,
usualMedium, usualStyle, usualType)

Phones(areaCode, telephoneNumber, street, zip)

Zips(zip, city, state)
```

Sin embargo, este diseño requeriría usar el número telefónico con la finalidad de obtener la calle y código postal de un artista, y que se hagan dos combinaciones siempre que se quiera obtener la dirección completa de un artista. Por cuestiones de eficiencia se llegará al acuerdo y la calle y código postal se pondrán de vuelta en la tabla Artist1. Se elige dejar la tabla Zips como está, y se nota que las tablas de códigos postales completos están disponibles para compras en forma electrónica. Ahora la forma para las tablas Artist es

```
Artist2 (firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber,
street, zip, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle,
usualType)

Zips(zip, city, state)
```

Note que la clave primaria de Artist2 es compuesta, y que se convertirá en clave externa compuesta en otras tablas. Las claves externas compuestas requieren verificar múltiples campos en las combinaciones. En la práctica, es mejor usar un campo numérico para un valor de clave que una cadena de caracteres, que está sujeta a diferencias en pronunciación, mayúsculas y puntuación al ingresar datos. Los errores o variaciones de entrada de datos al ingresar datos cadena pueden causar errores cuando se intenta comparar valores; en particular, cuando los campos se usan como claves externas. Por tanto, se creará un identificador numérico único para cada artista, que se llamará `artistId`, y ésa será la clave primaria de la primera tabla. Todos los atributos de la primera tabla serán funcionalmente dependientes sobre ella. Este tipo de clave se llama **clave subrogada** y la mayoría de los sistemas de gestión de base de datos tienen un mecanismo para generar valores y seguir la pista de los valores para claves subrogadas. Oracle usa un concepto llamado **secuencia** para este propósito. Microsoft Access usa un tipo de dato **autonúmero** para el mismo propósito. De hecho, Access conmina al usuario a permitirle generar un campo clave de este tipo si el usuario rechaza especificar uno. En consecuencia, como tablas finales de artista se tiene

(1) Artist3(artistId, firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

(2) Zips(zip, city, state)

Para `PotentialCustomer` se tienen muchas de las mismas DF que se vieron para Artist.

firstName + lastName + areaCode + telephoneNumber → todos los atributos

areaCode + telephoneNumber → street, city, state, zip

zip → city, state

Al usar el mismo razonamiento como se hizo para `Artist` se agregará un identificador único, `potentialCustomerId`, para usar como la clave primaria. Puede preguntar si existe una dependencia funcional entre el artista preferido y otras preferencias. Aunque puede haber alguna conexión, no es una verdadera dependencia funcional pues, por ejemplo, dos personas que admiren al mismo artista pueden preferir obras que estén en diferentes medios o estilos, acaso incluso producidas por el mismo artista.

Para FNBC estricta `PotentialCustomer` se descompondría del modo siguiente:

Customer1(potentialCustomerId, firstName, lastName, *areaCode, telephoneNumber,* dateFilledIn, *preferredArtistLastName, preferredArtistFirstName,* preferredMedium, preferredStyle, preferredType)

Phones(areaCode, telephoneNumber, street, *zip*)

Zips(zip, city, state)

Al usar la misma lógica que se utilizó para la información del artista, se elige crear una tabla que conserve la calle y código postal con los otros datos del cliente, y usar la tabla `Zips` que ya existe para determinar ciudad y estado. También se quiere usar `preferredArtistId` en lugar del apellido y el nombre. En consecuencia, al diseño de la tabla se agrega

(3) PotentialCustomer2 (potentialCustomerId, firstName, lastName, areaCode, telephoneNumber, street, *zip,* dateFilledIn, *preferredArtistId,* preferredMedium, preferredStyle, preferredType)

Para `Artwork`, se tienen las siguientes DF:

artistLastName + artistFirstName + workTitle → todos los atributos

Aquí se usa el nombre del artista como una clave externa. Puesto que se cambió la clave primaria de `Artist` a `artistId`, también se cambiará la clave externa, lo que sustituye el nombre por la Id en la tabla `Artwork`. Aunque existe alguna conexión lógica entre las fechas no hay dependencia funcional entre ellas. También se creará un identificador único para cada obra, `artworkId`,

(4) Artwork (artworkId, *artistId,* workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

Para la tabla `ShowIn` no existen dependencias funcionales no triviales, así que la tabla se podría conservar en su forma actual. Sin embargo, se desea usar artworkId para identificar la obra de arte.

(5) ShownIn (*artworkId, showTitle*)

En la tabla `Collector` se tienen las DF

socialSecurityNumber → todos los atributos

así como las DF vistas anteriormente que involucran números telefónicos y códigos postales. También se quiere usar `artistId` en lugar del nombre y apellido del artista. Por tanto, se elige crear la tabla que se muestra a continuación y usar la tabla `Zips` existente

(6) Collector1 (socialSecurityNumber, firstName, lastName, street, *zip*, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, *collectionArtistId*, collectionMedium, collectionStyle, collectionType, salesLastYear, salesYearToDate)

Para la tabla `Show` se tiene la DF

showTitle → todos los atributos

Sin embargo, se quiere sustituir el `artistId` por el nombre, como se hizo para las tablas anteriores. Puede existir alguna conexión entre el artista presentado y el título de la exposición, pero no necesariamente son dependientes funcionales. Por ejemplo, dos diferentes exposiciones pueden ambas presentar al mismo artista, pero sus títulos serán distintos. Lo mismo es cierto del tema y el título. También existe alguna conexión entre la fecha de apertura y la fecha de cierre de la exposición. Si se supone que sólo una exposición puede estar abierta en una fecha dada, entonces sólo habrá una fecha de cierre asociada con dicha fecha de apertura, y se tendría una dependencia transitiva que necesitaría remover. Agregue a las suposiciones que más de una exposición puede abrir al mismo tiempo. Ésta es una suposición razonable si la galería es suficientemente grande. En este acaso, puede haber exposiciones que tengan la misma fecha de apertura pero diferentes fechas de cierre. Dadas estas suposiciones, la tabla `Show` ya está normalizada.

(7) Show (showTitle, *showFeaturedArtistId*, showClosingDate, showTheme, showOpeningDate)

Para `Buyer`, se tiene la DF

firstName + lastName + areaCode + telephoneNumber → todos los atributos

así como las DF que involucran números telefónicos y códigos postales, como se vio anteriormente para `Artist` y para `Collector`. Como se hizo para `Artist` se creará una clave primaria numérica, `buyerId`. Al usar el mismo patrón para dichas tablas se diseña una nueva tabla `Buyer` y se usa la tabla `Zips` diseñada anteriormente.

(8) Buyer (buyerId, firstName, lastName, street, *zip*, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

Para la tabla `Sale` se tiene la DF

invoiceNumber → todos los atributos

Dado que cada obra de arte se vende cuando mucho una vez, también se tiene

artworkId → todos los atributos

Dado que es permisible conservar una clave candidata en la relación, esto no presenta un problema. De nuevo, se sustituirá `artworkId` por el nombre del artista y el título, y la `buyerId` por el nombre del comprador como claves externas. Si la comisión es un porcentaje constante del precio de venta, entonces se tiene

salePrice → saleSalesPersonCommission

No se supone que el precio de venta determina el impuesto, pues algunos compradores, como las organizaciones no lucrativas, pueden estar exentos de impuestos. Sin embargo, la venta total es sólo la suma del precio de venta e impuesto, así que se tiene

salePrice + tax → saleTotal

En realidad, cualquiera de estos dos determina al otro. Puede pensar que se tiene la DF entre `salePrice` y la `amountRemittedToOwner`. Sin embargo, es posible que pueda

haber una demora entre el tiempo cuando una obra se vende y el tiempo en que se le paga al propietario, así que se podrían tener dos ventas con el mismo precio de venta y diferentes cantidades remitidas al propietario, puesto que una todavía no se paga. Al remover las DF identificadas aquí, se forma la nueva tabla

(9) `Sale1 (InvoiceNumber, artworkId, amountRemittedToOwner, saleDate, salePrice, saleTax, buyerId, salesPersonSocialSecurityNumber)`

También podría tener la tabla

`Commissions(salePrice, saleSalesPersonCommission)`

pero no se necesitaría almacenar ésta, pues la comisión se calcula fácilmente. De igual modo, debido a la relación aritmética entre los atributos, no necesita almacenar la tabla.

`Totals(salePrice, saleTax, saleTotal)`

Para `Salesperson` se tiene

`socialSecurityNumber` → todos los atributos
`zip` → `city, state`

Al remover la dependencia transitiva y usar la tabla `Zips` existente, se tiene

(10) `Salesperson (socialSecurityNumber, firstName, lastName, street, zip)`

Las tablas numeradas 1-10 se usarán como el conjunto final de tablas para el diseño relacional de esta base de datos.

- Paso 5.2. Actualice el diccionario de datos y cite las suposiciones según se requiera.

Es necesario agregar al diccionario de datos los nuevos identificadores creados, del modo siguiente:

artistId Identificador numérico único creado para cada artista.

artworkId Identificador numérico único creado para cada obra de arte.

buyerId Identificador numérico único creado por cada comprador.

potentialCustomerId Identificador numérico único creado por cada cliente potencial.

Se hace una nota de los ítems calculados que no se almacenan.

saleSalesPersonCommission Cantidad en dólares de la comisión para una vendedor por la venta de una obra de arte. *Ítem calculado*.

saleTotal Cantidad total, en dólares, de una venta, incluido precio e impuesto, para una obra de arte; *calculado a partir de salePrice y saleTax. Ítem calculado.*

Se ve que `Artist`, `Buyer`, `Collector`, `PotentialCustomer` y `Salesperson` comparten atributos que tienen el mismo significado, así que se eliminan las distinciones con base en las fuentes para los siguientes atributos: `name`, `firstName`, `lastName`, `address`, `street`, `city`, `state`, `zip`, `phone`, `areaCode`, `telephoneNumber` y `socialSecurityNumber`.

No hay cambios a la lista de suposiciones.

PROYECTOS ESTUDIANTILES: NORMALIZACIÓN DEL MODELO RELACIONAL PARA LOS PROYECTOS ESTUDIANTILES

- Paso 5.1. Comience con la lista de las tablas a las que las entidades y relaciones del diagrama E-R se mapearon de manera natural, a partir de la sección del proyecto de muestra al final del capítulo 4. Para cada tabla en la lista, identifique las dependen-

cias funcionales y normalice la relación a FNBC. Luego decida si las tablas resultantes se deben implementar en dicha forma. Si no, explique por qué. Proporcione el esquema final para el modelo relacional.

- Paso 5.2. Actualice el diccionario de datos y la lista de suposiciones según se requiera.

CAPÍTULO 6

Sistemas de gestión de bases de datos relacionales y SQL

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- La historia de los sistemas de bases de datos relacionales y SQL
- Cómo se implementa la arquitectura de tres niveles en los sistemas de gestión de bases de datos relacionales
- Cómo crear y modificar una estructura de base de datos a nivel lógico usando SQL DDL
- Cómo recuperar y actualizar datos en una base de datos relacional usando SQL DML
- Cómo reforzar las restricciones en las bases de datos relacionales
- Cómo terminar transacciones relacionales
- Cómo se usa SQL en un ambiente de programación
- Cómo crear vistas relacionales
- Cuándo y cómo realizar operaciones sobre vistas relacionales
- Estructura y funciones de un catálogo de sistema de una base de datos relacional
- Las funciones de los varios componentes de un sistema de gestión de base de datos relacional

PROYECTO DE MUESTRA: Creación y manipulación de una base de datos relacional para la Galería de Arte
PROYECTOS ESTUDIANTILES: Creación y uso de una base de datos relacional para los proyectos estudiantiles

6.1 Breve historia de SQL en sistemas de bases de datos relacionales

Como se describe en el capítulo 4, el modelo relacional fue propuesto por primera vez por E. F. Codd en 1970. D. D. Chamberlin y otros en el Laboratorio de investigación San José de IBM desarrollaron un lenguaje ahora llamado SQL, o Structured Query Language (lenguaje de consulta estructurado) como un sublenguaje de datos para el modelo relacional. Originalmente nombrado SEQUEL, el lenguaje se presentó en una serie de ponencias que comenzaron en 1974, y se usó en un sistema relacional prototipo llamado System R, que desarrolló IBM a finales de la década de 1970. Otros primeros sistemas de gestión de bases de datos relacionales prototipos incluyeron INGRES, que se desarrolló en la Universidad de California en Berkeley, y el Peterlee Relational Test Vehicle, creado en el Laboratorio científico IBM del Reino Unido. El System R se evaluó y refinó durante un periodo de varios años y se convirtió en la base del primer sistema IBM de gestión de base de datos relacional disponible comercialmente, SQL/DS, que se anunció en 1981. Otro primer sistema de gestión de base de datos comercial, Oracle, se desarrolló a finales de la década de 1970 con el uso de SQL como su lenguaje. El DB2 de IBM, que también usa SQL como su lenguaje, se lanzó en 1983. Microsoft SQL Server, MySQL, Informix, Sybase, dBase, Paradox, r:Base, FoxPro y muchos otros sistemas de gestión de bases de datos relacionales han incorporado SQL.

Tanto el American National Standards Institute (ANSI) como la International Standards Organization (ISO) adoptaron SQL como un lenguaje estándar para bases de datos relacionales y publicaron especificaciones para el lenguaje SQL en 1986. Este estándar usualmente se llama SQL1. Una revisión menor, llamada SQL-89, se publicó tres años después. ANSI e ISO adoptaron una revisión mayor, SQL2, en 1992. Las primeras partes del estándar SQL3, que se conocen como SQL: 1999, se publicaron en 1999. Las grandes nuevas características incluyeron capacidades de gestión de datos orientados a objetos y tipos de datos definidos por el usuario. La mayoría de los proveedores de sistemas de gestión de bases de datos relacionales usan sus propias extensiones del lenguaje, lo que crea una variedad de dialectos alrededor del estándar.

SQL tiene un lenguaje de definición de datos (DDL) completo y lenguaje de manipulación de datos (DML) descritos en este capítulo, y un lenguaje de autorización, descrito en el capítulo 9. Los lectores deben notar que diferentes implementaciones de SQL varían ligeramente de la sintaxis estándar que se presentó aquí, pero las nociones básicas son las mismas.

6.2 Arquitectura de un sistema de gestión de bases de datos relacional

Los sistemas de gestión base de datos relacional soportan la arquitectura estándar en tres niveles descrita en la sección 2.6. Como se muestra en la figura 6.1, las bases de datos relacionales proporcionan independencia de datos tanto lógica como física, porque separan los niveles externo, lógico e interno. El nivel lógico para bases de datos relacionales consiste en tablas base que se almacenan físicamente. Estas tablas se crean mediante el administrador

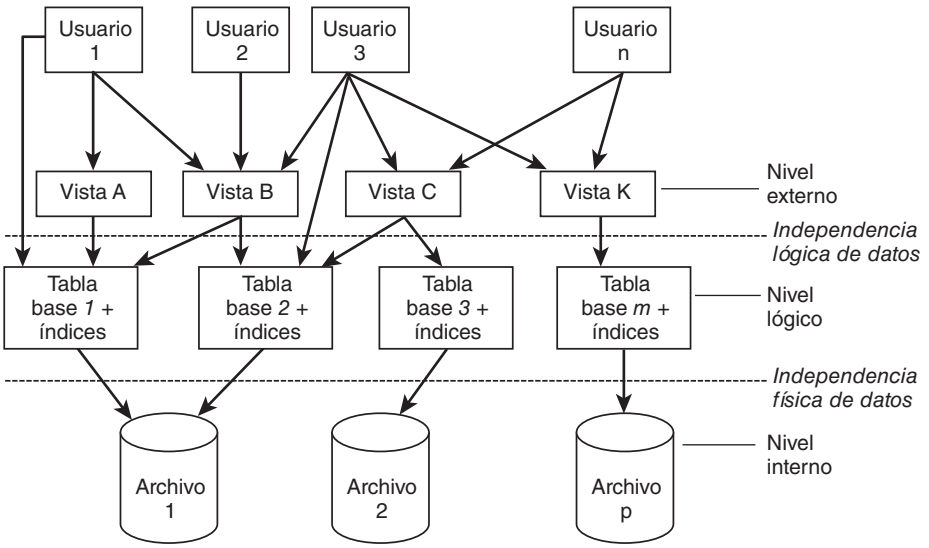


FIGURA 6.1
Arquitectura en tres niveles para bases de datos relacionales

de base de datos con el uso de un comando CREATE TABLE (crear tabla), como se describe en la sección 6.3. Una tabla base puede tener cualquier número de índices, creados por el ABD usando el comando CREATE INDEX (crear índice). Un índice se usa para acelerar la recuperación de registros con base en el valor en una o más columnas. Un índice menciona los valores que existen para la columna indexada y la ubicación de los registros que tienen dichos valores. La mayoría de los sistemas de gestión de bases de datos relacionales usan árboles B o árboles B+ para los índices (vea el apéndice A). En el nivel físico, las tablas base se representan, junto con sus índices, en archivos. La representación física de las tablas puede no corresponder exactamente con la noción de una tabla base como un objeto bidimensional que consiste en filas y columnas. Sin embargo, las filas de la tabla sí corresponden con los registros almacenados físicamente, aunque su orden y otros detalles de almacenamiento pueden ser diferentes del concepto de ellos. El sistema de gestión de bases de datos, no el sistema operativo, controla la estructura interna de archivos e índices. El usuario generalmente no está al tanto de cuáles índices existen, y no tiene control sobre cuáles índices usará para localizar un registro. Una vez creadas las tablas base, el ABD puede crear "vistas" para los usuarios, con el uso del comando CREATE VIEW (crear vista), descrita en la sección 6.8. Una vista puede ser un subconjunto de una sola tabla base, o bien crearse al combinar tablas base. Las vistas son "tablas virtuales", que no se almacenan de modo permanente, sino que se crean cuando el usuario necesita acceder a ellas. Los usuarios no están al tanto del hecho de que sus vistas no se almacenan de manera física en forma de tabla. Esto no es exactamente lo mismo que el término "vista externa", que significa la base de datos como aparece a un usuario particular. En esta terminología, una vista externa puede consistir en varias tablas base y/o vistas.

Una de las características más útiles de una base de datos relacional es que permite definición dinámica de base de datos. El ABD, y los usuarios a quienes autoriza, pueden crear nuevas tablas, agregar columnas a las anteriores, crear nuevos índices, definir vistas y eliminar cualquiera de estos objetos en cualquier momento. En contraste, muchos otros sistemas requieren que toda la estructura de la base de datos se defina en el momento de creación, y que todo el sistema se detenga y vuelva a cargar (reload) cuando se haga cualquier cambio estructural. La flexibilidad de las bases de datos relacionales alienta a los usuarios a experimentar con varias estructuras y permite la modificación del sistema para satisfacer sus necesidades cambiantes. Esto permite al ABD a asegurar que la base de datos sea un modelo útil de la empresa a lo largo de su ciclo de vida.

6.3 Definición de la base de datos: SQL DDL

Los comandos más importantes del lenguaje de definición de datos (DDL) SQL son los siguientes:

CREATE TABLE
CREATE INDEX
ALTER TABLE
RENAME TABLE
DROP TABLE
DROP INDEX

Estos enunciados se usan para crear, cambiar y destruir las estructuras lógicas que constituyen el modelo lógico. Estos comandos se pueden usar en cualquier momento para realizar cambios a la estructura de la base de datos. Existen comandos adicionales disponibles para especificar detalles físicos de almacenamiento, pero no se les discutirá aquí, pues son específicos al sistema.

Se aplicarán estos comandos al siguiente ejemplo, que se usó en capítulos anteriores:

```
Student (stuId, lastName, firstName, major, credits)
Faculty (facId, name, department, rank)
Class (classNumber, facId, schedule, room)
Enroll (classNumber, stuId, grade)
```

6.3.1 Create Table (crear tabla)

Este comando se usa para crear las tablas base que forman el corazón de una base de datos relacional. Dado que se puede usar en cualquier momento durante el ciclo de vida del sistema, el desarrollador de la base de datos puede comenzar con un pequeño número de tablas y agregarlas conforme se planeen y desarrollen aplicaciones adicionales. Una tabla base está bastante cercana a la noción abstracta de una tabla relacional. Consiste de uno o más **encabezados (headings) de columna**, que proporcionan el **nombre de columna** y **tipo de datos**, y cero o más **filas de datos**, que contienen un valor de datos del tipo de datos especificado para cada una de las columnas. Como en el modelo relacional abstracto, las filas se consideran sin orden. Sin embargo, las columnas están ordenadas de izquierda a derecha, para coincidir con el orden de las definiciones de columna en el comando CREATE TABLE. La forma del comando es:

```
CREATE TABLE nombre tabla base (nombre_col tipo_dato [restricciones columna -
NULL/NOT NULL, DEFAULT . . . , UNIQUE, CHECK . . . , PRIMARY KEY . . .]]
[,nombre_col tipo_dato [restricciones columna]
. . .
[restricciones tabla - PRIMARY KEY . . . , FOREIGN KEY . . . , UNIQUE
. . . , CHECK . . .]
[especificaciones almacenamiento]);
```

Aquí, *nombre tabla base* es un nombre proporcionado por el usuario para la tabla. No se pueden usar palabras clave SQL, y el nombre de la tabla debe ser único dentro de la base de datos. Para cada columna, el usuario debe especificar un nombre que sea único dentro de la tabla, y un tipo de datos. La sección opcional de especificaciones de almacenamiento del comando CREATE TABLE permite al ABD nombrar el espacio de la tabla donde se almacenará la tabla. Si el espacio de tabla no se especifica, el sistema de gestión de base de datos creará un espacio por defecto para la tabla. Quienes quieran pueden ignorar los detalles del sistema y quienes deseen más control pueden ser muy específicos acerca de las áreas de almacenamiento.

La figura 6.2 muestra los comandos para crear las tablas base para una base de datos para el ejemplo University.

6.3.1.1 Tipos de datos

Los tipos de datos disponibles incluyen varios tipos numéricos, cadenas de caracteres de longitud fija y de longitud variable, cadenas de bits y tipos definidos por el usuario. Los tipos de datos disponibles varían de DBMS a DBMS. Por ejemplo, los tipos más comunes en Oracle son CHAR(N), VARCHAR2(N), NUMBER(N,D), DATE y BLOB (gran objeto binario). En DB2, los tipos incluyen SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, REAL, DOUBLE, CHAR(N), VARCHAR(N), LONG VARCHAR, CLOB, GRAPHIC, DBCLOB, BLOB, DATE, TIME y TIMESTAMP. Microsoft SQL Server incluye NUMERIC, BINARY, CHAR, VARCHAR, DATE-TIME, MONEY, IMAGE y otros. Microsoft Access soporta varios tipos de NUMBER, así como TEXT, MEMO, DATE/TIME, CURRENCY, YES/NO y otros. Además, algunos sistemas como Oracle permiten a los usuarios crear nuevos dominios, y construir sobre los tipos de datos existentes. En lugar de usar uno de los tipos de datos disponibles, los usuarios pueden especificar dominios por adelantado y pueden incluir una condición de verificación para el dominio. SQL:1999 permite la creación de nuevos tipos de datos **distintos** usando uno de los tipos anteriormente definidos como el tipo fuente. Por ejemplo, se podría escribir:

```
CREATE DOMAIN creditValues INTEGER
   DEFAULT 0
   CHECK (VALUE >=0 AND VALUE <150);
```

Una vez creado el dominio, puede usarlo como un tipo de datos para atributos. Por ejemplo, cuando se crea la tabla Student, para la especificación de `credits` se podría escribir entonces:

```
credits creditValues, . . .
```

en lugar de

```
credits SMALLINT DEFAULT 0,
. . .
CONSTRAINT Student_credits_cc CHECK (credits>=0 AND credits < 150)
```

Sin embargo, cuando se crean tipos distintos, SQL:1999 no permite comparar sus valores con valores de otros atributos que tengan el mismo tipo fuente subyacente. Por ejemplo, si usa el dominio `creditValues` para `credits`, no se puede comparar `credits` con otro atributo cuyo tipo también sea SMALLINT, por ejemplo, con `age`, si tiene almacenado dicho atributo. No se pueden usar las funciones SQL disponibles como COUNT, AVERAGE, SUM, MAX o MIN en tipos distintos, aunque se pueden escribir las definiciones de las funciones para los nuevos tipos.

6.3.1.2 Restricciones (constraints) de columna y tabla

El sistema de gestión de base de datos tiene facilidades para reforzar la exactitud de los datos, las que debe usar el ABD cuando cree tablas. Recuerde de la sección 4.4 que el modelo relacional usa restricciones de integridad para proteger la exactitud de la base de datos y permitir la creación sólo de instancias legales. Estas restricciones protegen el sistema de errores en entrada de datos que crearían datos inconsistentes. Aunque el nombre de la tabla, los nombres de columna y los tipos de datos son las únicas partes requeridas en el comando CREATE TABLE, se pueden y deben agregar restricciones opcionales, tanto a nivel columna como a nivel tabla.

Las restricciones de columna incluyen opciones para especificar NULL/NOT NULL, UNIQUE, PRIMARY KEY, CHECK y DEFAULT para cualquier columna, inmediatamente des-

FIGURA 6.2
Enunciados DDL SQL para crear tablas para el ejemplo University

```
CREATE TABLE Student (
stuId CHAR(6),
lastName CHAR(20) NOT NULL,
firstName CHAR(20) NOT NULL,
major CHAR(10),
credits SMALLINT DEFAULT 0,
CONSTRAINT Student_stuId_pk PRIMARY KEY (stuId),
CONSTRAINT Student_credits_cc CHECK (CREDITS>=0 AND credits < 150);

CREATE TABLE Faculty (
facId CHAR(6),
name CHAR(20) NOT NULL,
department CHAR(20) NOT NULL,
rank CHAR(10),
CONSTRAINT Faculty_facId_pk PRIMARY KEY (facId));

CREATE TABLE Class (
classNumber CHAR(8),
facId CHAR(6) NOT NULL,
schedule CHAR(8),
room CHAR(6),
CONSTRAINT Class_classNumber_pk PRIMARY KEY (classNumber),
CONSTRAINT Class_facId_fk FOREIGN KEY (facId) REFERENCES Faculty (facId));

CREATE TABLE Enroll (
stuId CHAR(6),
classNumber CHAR(8),
grade CHAR(2),
CONSTRAINT Enroll_classNumber_stuId_pk PRIMARY KEY (classNumber, stuId),
CONSTRAINT Enroll_classNumber_fk FOREIGN KEY (classNumber) REFERENCES Class (classNumber),
CONSTRAINT Enroll_stuId_fk FOREIGN KEY (stuId) REFERENCES Student (stuId) ON DELETE CASCADE);
```

pués de la especificación del nombre de columna y el tipo de datos. Si no se especifica NOT NULL, el sistema permitirá que la columna tenga valores nulos, lo que significa que el usuario puede insertar registros que no tengan valores para dichos campos. Cuando un valor nulo aparezca en un campo de un registro, el sistema es capaz de distinguirlo de una cadena en blanco o valor cero, y tratarlos de manera diferente en cálculos y comparaciones lógicas. Es deseable poder insertar valores nulos en ciertas situaciones; por ejemplo, cuando un estudiante universitario todavía no haya declarado una especialidad es posible que quiera establecer un campo `major` a nulo. Sin embargo, el uso de valores nulos puede crear complicaciones, especialmente en operaciones como combinaciones, así que debe usar NOT NULL cuando sea adecuado. También puede especificar un valor por defecto para una columna, si quiere hacerlo así. Entonces, a todo registro que se inserte sin un valor para el

campo se le dará de manera automática el valor por defecto. Opcionalmente puede especificar que un campo dado tenga valores únicos al escribir la restricción UNIQUE. En este caso, el sistema rechazará la inserción de un nuevo registro que tenga en dicho campo el mismo valor que un registro ya presente en la base de datos. Si la clave primaria no es compuesta, también es posible especificar PRIMARY KEY como una restricción de columna, simplemente al agregar las palabras PRIMARY KEY después del tipo de datos para la columna. Claramente no se puede permitir valores duplicados para la clave primaria. También se deshabilitan los valores nulos, pues no se podría distinguir entre dos diferentes registros si ambos tuvieran valores nulos, así que la especificación de PRIMARY KEY en SQL lleva implícita una restricción NOT NULL, así como una restricción UNIQUE. Sin embargo, tal vez también quiera asegurar la exclusividad de las claves candidatas, y debe especificar UNIQUE para ellas cuando cree la tabla. El sistema automáticamente comprueba cada registro que intenta insertar para garantizar que los ítems de datos para columnas descritas como únicas no tengan valores que dupliquen cualquier otro ítem de datos en la base de datos para dichas columnas. Si es posible una duplicación rechazará la inserción. También es deseable especificar una restricción NOT NULL para claves candidatas, cuando es posible asegurar que los valores para dichas columnas siempre estarán disponibles. La restricción CHECK se puede usar a fin de verificar que los valores proporcionados para los atributos sean adecuados. Por ejemplo, podría escribir:

```
credits SMALLINT DEFAULT 0 CHECK ((credits>=0) AND (credits < 150)),
```

Las restricciones de tabla, que aparecen después de declarar todas las columnas, pueden incluir la especificación de una clave primaria, claves externas, exclusividad, comprobaciones y restricciones generales que se pueden expresar como condiciones a verificar. Si la clave primaria es compuesta, debe identificarla usando una restricción de tabla en lugar de una restricción de columna, aunque incluso una clave primaria que consista en una sola columna se puede identificar como una restricción de tabla. La restricción PRIMARY KEY fortalece la exclusividad y las restricciones de no nulo para la columna identificada como la clave primaria. La restricción FOREIGN KEY requiere la identificación de la tabla referenciada donde la columna o combinación de columnas sea una clave primaria. El estándar SQL permite especificar qué se hará con los registros que contengan los valores de clave externa cuando los registros que relaciona se actualicen o borren en su tabla origen (home). Para el ejemplo University, ¿qué debe ocurrir a un registro `Class` cuando el registro de los miembros del personal docente asignados a impartir la clase se borren o se actualice el `facId` del registro `Faculty`? Para el caso de borrado, el DBMS automáticamente podría:

- Borrar (delete) todos los registros `Class` para dicho miembro docente, una acción que se realiza cuando se especifica ON DELETE CASCADE (al borrar cascada) en la especificación de clave externa en SQL.
- Establecer el `facId` en el registro `Class` a un valor nulo, una acción que se realiza cuando se escribe ON DELETE SET NULL (al borrar establecer nulo) en SQL.
- Establecer el `facId` a algún valor por defecto como F999 en la tabla `Class`, una acción que se realiza cuando se escribe ON DELETE SET DEFAULT (al borrar establecer por defecto) en SQL (esta opción requiere que se use la restricción de columna DEFAULT para esta columna previo a la especificación de la clave externa).
- No permitir el borrado de un registro `Faculty` si existe un registro `Class` que se refiera a él, una acción que se realiza cuando especifica ON DELETE NO ACTION (al borrar no acción) en SQL.

Las mismas acciones, con significados similares, se pueden especificar en una cláusula ON UPDATE (al actualizar); esto es

```
ON UPDATE CASCADE/SET NULL/SET DEFAULT/NO ACTION
```

Tanto para borrado como para actualización, el defecto es NO ACTION, lo que en esencia no permite los cambios a un registro en una relación origen que causaría inconsistencia con los registros que se refieran a él. Como se muestra en la figura 6.2, para la tabla `Class` se eligió el defecto. Note también las opciones realizadas para la tabla `Enroll`, para los cambios hechos tanto a `classNumber` como a `stuId`.

El mecanismo de restricción de exclusividad de tabla se puede usar para especificar que los valores en una combinación de columnas deben ser únicos. Por ejemplo, para garantizar que ningunas dos clases tengan exactamente el mismo horario y salón, se escribiría:

```
CONSTRAINT Class_schedule_room_uk UNIQUE (schedule, room)
```

Recuerde de la sección 4.4 que la restricción de exclusividad permite especificar claves candidatas. La restricción anterior dice que `{schedule,room}` es una clave candidata para `Class`. También se podría especificar que `{facId,schedule}` es una clave candidata mediante

```
CONSTRAINT Class_facId_schedule_uk UNIQUE (facId, schedule)
```

pues un miembro del personal docente no puede impartir dos clases exactamente en el mismo horario.

A las restricciones, ya sean a nivel columna o tabla, en forma opcional se les puede dar un nombre, como se ilustra en los ejemplos. Si no se les nombra, el sistema generará un nombre de restricción único para cada restricción. La ventaja de las restricciones de nomenclatura es que luego se puede hacer referencia a ellas con facilidad. Existen comandos SQL para permitir la deshabilitación, habilitación, alteración o eliminación de restricciones a voluntad, siempre que se conozcan sus nombres. Es una buena práctica usar un patrón consistente en la nomenclatura de las restricciones. El patrón que se ilustra aquí es el nombre de tabla, nombre de columna y una abreviatura para el tipo de restricción (pk, fk, nn, uk, cc), separados por guiones inferiores.

6.3.2 Create Index (crear índice)

Opcionalmente puede crear índices para las tablas con el fin de facilitar la rápida recuperación de registros con valores específicos en una columna. Un índice sigue la pista de cuáles valores existen para la columna indexada y cuáles registros tienen dichos valores. Por ejemplo, si se tiene un índice en la columna `lastName` de la tabla `Student`, y se escribe una solicitud de consulta para todos los estudiantes con apellido Smith, el sistema no tendrá que explorar todos los registros `Student` para elegir los deseados. En lugar de ello, leerá el índice, que apuntará a los registros con el nombre deseado. Una tabla puede tener cualquier número de índices, que se almacenan como B-trees (árboles B) o B+ trees (árboles B+) en archivos índice separados, por lo general cerca de las tablas que indexan (vea el apéndice A para una descripción de los índices árbol). Los índices pueden crearse en campos sencillos o combinaciones de campos. Sin embargo, dado que el sistema debe actualizar los índices cada vez que se actualizan las tablas subyacentes, se requiere uso de sistema adicional. Además de elegir cuáles índices existirán, los usuarios no tienen control sobre el uso o mantenimiento de los índices. El sistema elige cuál, si hay alguno, índice usar en la búsqueda de registros. Los índices no son parte del estándar SQL, pero la mayoría de los DBMS soportan su creación. El comando para crear un índice es:

CREATE [UNIQUE] INDEX *nombre índice* ON *nombre tabla base* (*nombre_col* [*order*] [,*colname* [*orden*]] . . .) [CLUSTER] ;

Si se usa la especificación UNIQUE, el sistema reforzará la exclusividad del campo o combinación de campos indexados. Aunque los índices se pueden crear en cualquier momento,

puede tener problemas si intenta crear un índice único después de que la tabla tenga registros almacenados en ella, porque los valores almacenados para el campo o campos indexados pueden ya contener duplicados. En este caso, el sistema no permitirá la creación del índice único. Para crear el índice en `lastName` para la tabla `Student` se escribiría:

```
CREATE INDEX Student_lastName ON STUDENT (lastName);
```

El nombre del índice se debe elegir para indicar la tabla y el campo o campos usados en el índice. Cualquier número de columnas, sin importar dónde aparecen en la tabla, se puede usar en un índice. La primera columna nombrada determina el orden mayor, la segunda da el orden menor, etc. Para cada columna es posible especificar que el orden es ascendente, ASC, o descendente, DESC. Si elige no especificar el orden, ASC es el defecto. Si escribe

```
CREATE INDEX Faculty_department_name ON Faculty (department ASC, name ASC);
```

entonces se creará el archivo índice llamado `Faculty_Department_Name` para la tabla `Faculty`. Las entradas estarán en orden alfabético por departamento. Dentro de cada departamento, las entradas estarán en orden alfabético por nombre de docente.

Algunos DBMS permiten una especificación CLUSTER (grupo) opcional sólo para un índice para cada tabla. Si usa esta opción, el sistema almacenará registros con los mismos valores para el campo indexado que estén juntos físicamente, en la misma página o páginas adyacentes si es posible. Si crea un índice agrupado para el campo que se utiliza con más frecuencia para recuperación, puede mejorar sustancialmente el rendimiento de aquellas aplicaciones que necesiten dicho orden particular de recuperación, pues se minimizará el tiempo de búsqueda y el tiempo de lectura. Sin embargo, es el sistema, no el usuario, el que elige usar un índice particular, incluso uno agrupado, para la recuperación de datos.

Oracle crea automáticamente un índice en la clave primaria de cada tabla que se crea. El usuario debe crear índices adicionales en cualquier campo que se use con frecuencia en consultas, para acelerar la ejecución de dichas consultas. Los campos de clave externa, que se usan con frecuencia en combinaciones, son buenas candidatas para indexar.

6.3.3 ALTER TABLE, RENAME TABLE

Una vez creada una tabla, los usuarios pueden encontrarla más útil si contiene un ítem de datos adicional, no tiene una columna particular o tiene diferentes restricciones. Aquí, la naturaleza dinámica de una estructura de base de datos relacional hace posible cambiar las tablas base existentes. Por ejemplo, para agregar una nueva columna a la derecha de la tabla, use un comando de la forma:

ALTER TABLE *nombre tabla base* ADD *nombre_col tipo_dato*;

Note que no se puede usar la especificación NULL para la columna. Un comando ALTER TABLE ...ADD (alterar tabla ...agregar) hace que el nuevo campo se agregue a todos los registros ya almacenados en la tabla, y los valores nulos se asignen a dicho campo en todos los registros existentes. Los registros recientemente insertados, desde luego, tendrán el campo adicional, pero no se tiene permiso para especificar no nulos incluso para ellos.

Suponga que quiere agregar una nueva columna, `cTitle`, a la tabla `Class`. Esto se puede hacer al escribir

```
ALTER TABLE Class ADD cTitle CHAR(30);
```

El esquema de la tabla `Class` sería entonces:

```
Class(classNumber,facId,schedule,room,cTitle)
```

Todos los antiguos registros `Class` ahora tendrían valores nulos para `cTitle`, pero se podría proporcionar un título para cualquier nuevo registro `Class` que se inserte, y actua-

lizar los antiguos registros `Class` al agregarles títulos. También se pueden eliminar columnas de las tablas existentes mediante el comando:

```
ALTER TABLE nombre tabla base DROP COLUMN nombre columna;
```

Para eliminar la columna `cTitle` y regresar a la estructura original para la tabla `Class`, se escribiría:

```
ALTER TABLE Class DROP COLUMN cTitle;
```

Si quiere agregar, eliminar o cambiar una restricción, puede usar el mismo comando ALTER TABLE. Por ejemplo, si creó la tabla `Class` y despreció hacer `facId` una clave externa en `Class`, podría agregar la restricción en cualquier momento al escribir:

```
ALTER TABLE Class ADD CONSTRAINT Class_facId_fk FOREIGN KEY (facId) REFERENCES Faculty
(facId));
```

Podría eliminar una restricción nombrada existente al usar el comando ALTER TABLE. Por ejemplo, para eliminar la condición check (verificar) en el atributo `credits` de `Student` que se creó antes, podría escribir:

```
ALTER TABLE Student DROP CONSTRAINT Student_credits_cc;
```

Puede cambiar fácilmente el nombre de una tabla existente mediante el comando:

```
RENAME TABLE nombre tabla anterior TO nombre tabla nueva;
```

6.3.4 Enunciados DROP

Las tablas se pueden eliminar en cualquier momento mediante el comando SQL:

```
DROP TABLE nombre tabla base;
```

Cuando se ejecuta este enunciado se remueven la tabla en sí y todos los registros contenidos en ella. Además, todos los índices y, como verá más tarde, todas las vistas que dependen de ella se eliminan. Naturalmente, el ABD conferencia con los usuarios de la tabla antes de tomar tan drástico paso. Cualquier índice existente puede destruirse mediante el comando:

```
DROP INDEX nombre índice;
```

El efecto de este cambio puede o no verse en el rendimiento. Recuerde que los usuarios no pueden especificar cuándo el sistema usa un índice para recuperación de datos. Por tanto, es posible que exista un índice que realmente nunca se usó, y su destrucción no tendría efecto sobre el rendimiento. Sin embargo, la pérdida de un índice eficiente que el sistema use para muchas recuperaciones ciertamente afectaría el rendimiento. Cuando se elimina un índice, cualquier plan de acceso para aplicaciones que dependan de ella se marca como inválido. Cuando una aplicación las llama, se proyecta un nuevo plan de acceso para sustituir el anterior.

6.4 Manipulación de la base de datos: DML SQL

El lenguaje de consulta de SQL es **declarativo**, también llamado **no procedural**, lo que significa que permite especificar cuáles datos se recuperan sin dar los procedimientos para recuperarlos. Se puede usar como un lenguaje interactivo para consultas, incrustado en un lenguaje de programación huésped, o como un lenguaje completo en sí para cálculos con el uso de SQL/PSM (Persistent Stored Modules = módulos de almacenamiento persistentes).

Los enunciados DML SQL son:

SELECT
UPDATE
INSERT
DELETE

6.4.1 Introducción al enunciado SELECT

El enunciado SELECT se usa para recuperar datos. Es un comando poderoso, que realiza el equivalente de SELECT, PROJECT y JOIN del álgebra relacional, así como otras funciones, en un solo enunciado simple. La forma general de SELECT es

```
SELECT    [DISTINCT] nombre col [AS nombre nuevo], [,nombre col..] . . .
FROM      nombre tabla [alias] [,nombre tabla] . . .
[WHERE  predicado]
[GROUP BY nombre col [,nombre col] . . . [HAVING predicado]
```

o

```
[ORDER BY nombre col [,nombre col] . . .];
```

El resultado es una tabla que puede tener filas duplicadas. Dado que los duplicados se permiten en tal tabla, no es una relación en el sentido estricto, pero se le conoce como **multiconjunto** o **bolsa**. Como se indica por la ausencia de corchetes se requieren las cláusulas SELECT y FROM, pero no WHERE u otras cláusulas. Las muchas variaciones de este enunciado se ilustrarán mediante los ejemplos que siguen, con el uso de las tablas `Student`, `Faculty`, `Class` y/o `Enroll` como aparecen en la figura 6.3.

- Ejemplo 1. Recuperación simple con condición

  Pregunta: Obtener nombres, identificaciones y número de créditos de todos los que tienen especialidad Math.

  Solución: La información solicitada aparece en la tabla `Student`. Para dicha tabla se seleccionan sólo las filas que tienen un valor de 'Math' para `major`. Para dichas filas, sólo se despliegan las columnas `lastName`, `firstName`, `stuId` y `credits`. Note que se hace el equivalente SELECT (al encontrar las filas) y PROJECT (al desplegar sólo ciertas columnas) del álgebra relacional. También se reordenan las columnas.

  Consulta SQL:

```
SELECT      lastName, firstName, stuId, credits
FROM        Student
WHERE       major = 'Math';
```

  Resultado:

lastName	firstName	stuId	credits
Chin	Ann	S1002	36
McCarthy	Owen	S1013	0
Jones	Mary	S1015	42

Note que el resultado de la consulta es una tabla o un multiconjunto.

- Ejemplo 2. Uso de notación asterisco para "todas las columnas"

  Pregunta: Obtener toda la información acerca de Faculty CSC.

  Solución: Se quiere todo el registro `Faculty` de cualquier miembro del personal docente cuyo departamento sea 'CSC'. Dado que muchas recuperaciones SQL requieren todas las columnas de una sola tabla, existe una forma corta de expresar "todas

Student

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15

Faculty

facId	name	department	rank
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

Class

classNumber	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
CSC203A	F105	MThF12	M110
HST205A	F115	MWF11	H221
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

Enroll

stuId	classNumber	grade
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

FIGURA 6.3

La base de datos University (igual que la figura 1.1)

las columnas", a saber, con el uso de un asterisco en lugar de los nombres de columna en la línea SELECT.

Consulta SQL:

```
SELECT      *
FROM        Faculty
WHERE       department = 'CSC';
```

Resultado:

```
facId       name        department      rank
F105        Tanaka      CSC             Instructor
F221        Smith       CSC             Professor
```

Los usuarios que acceden a una base de datos relacional a través de un lenguaje huésped por lo general se les aconseja evitar el uso de la notación asterisco. El peligro es que se puede agregar una columna adicional a una tabla después de escribir un programa. Entonces el programa recuperará el valor de dicha nueva columna con cada registro y no tendrá una variable de programa que coincida con el valor, lo que causa una pérdida de correspon-

dencia entre variables de base de datos y variables de programa. Es más seguro escribir la consulta como:

```
SELECT     facId, name, department, rank
FROM       Faculty
WHERE      department = 'CSC';
```

- Ejemplo 3. Recuperación sin condición, uso de "distinto", uso de nombres calificados

  Pregunta: Obtener el número de curso de todos los cursos en los que están inscritos los estudiantes.

  Solución: Vaya a la tabla `Enroll` en lugar de a la tabla `Class`, porque es posible que exista un registro `Class` para una clase planeada en la que nadie esté inscrito. A partir de la tabla `Enroll` podría pedir una lista de todos los valores `classNumber`, del modo siguiente.

  Consulta SQL:

  ```
  SELECT   classNumber
  FROM     Enroll;
  ```

  Resultado:

  ```
  classNumber
  ART103A
  CSC201A
  CSC201A
  ART103A
  ART103A
  MTH101B
  HST205A
  MTH103C
  MTH103C
  ```

Dado que no necesita un predicado, no se usa la línea WHERE. Note que existen muchos duplicados en el resultado; es un multiconjunto, no una verdadera relación. A diferencia del PROJECT del álgebra relacional, el SELECT de SQL no elimina duplicados cuando se "proyecta" sobre las columnas. Para eliminar los duplicados, necesita usar la opción DISTINCT en la línea SELECT. Si escribe

```
SELECT DISTINCT classNumber
FROM Enroll;
```

El resultado sería:

```
classNumber
ART103A
CSC201A
HST205A
MTH101B
MTH103C
```

En cualquier recuperación, especialmente si existe una posibilidad de confusión debido a que el mismo nombre de columna aparece en dos tablas diferentes, especifique *nombretabla.nombrecol*. En este ejemplo se pudo haber escrito:

```
SELECT     DISTINCT Enroll.classNumber
FROM       Enroll;
```

Aquí, no es necesario usar el nombre calificado, pues la línea FROM dice al sistema que use la tabla `Enroll`, y los nombres de columna siempre son únicos dentro de una tabla. Sin embargo, nunca es equivocado usar un nombre calificado, y a veces es necesario hacerlo cuando dos o más tablas aparecen en la línea FROM.

- Ejemplo 4. Recuperación de una tabla completa

  Pregunta: Obtener toda la información acerca de todos los estudiantes.

  Solución: Puesto que se quieren todas las columnas de la tabla `Student`, use la notación asterisco. Puesto que se quieren todos los registros en la tabla, omita la línea WHERE.

  Consulta SQL:

```
SELECT     *
FROM       Student;
```

  Resultado: El resultado es toda la tabla `Student`.

- Ejemplo 5. Uso de "ORDER BY" y AS

  Pregunta: Obtener nombres e identificaciones de todos los miembros de `Faculty`, ordenados en orden alfabético por nombre. Solicite las columnas resultantes `FacultyName` y `FacultyNumber`.

  Solución: La opción ORDER BY (ordenar por) en el SELECT de SQL permite ordenar los registros recuperados en orden ascendente (ASC, por defecto) o descendente (DESC) en cualquier campo o combinación de campos, sin importar si dicho campo aparece en los resultados. Si se ordenan por más de un campo, el que se nombró primero determina orden mayor, el siguiente orden menor, etcétera.

  Consulta SQL:

```
SELECT     name AS FacultyName, facId AS
           FacultyNumber
FROM       Faculty
ORDER BY   name;
```

  Resultado:

FacultyName	FacultyNumber
Adams	F101
Byrne	F110
Smith	F202
Smith	F221
Tanaka	F105

Los encabezados de columna se cambian a los especificados en la cláusula AS. Se puede renombrar cualquier columna o columnas para desplegarla de esta forma. Note el nombre duplicado de 'Smith'. Dado que no se especifica orden menor, el sistema ordenará estas dos filas en cualquier orden que elija. Podría romper el "lazo" para dar un orden menor, como sigue:

```
SELECT   name AS FacultyName, facId AS
         FacultyNumber
FROM     Faculty
ORDER BY name, department;
```

Ahora los registros Smith se invertirán, pues F221 está asignado a CSC, que alfabéticamente está antes que History. Note también que el campo que determina el orden no necesita ser uno de los desplegados.

- Ejemplo 6. Uso de condiciones múltiples

  Pregunta: Obtener nombres de todos quienes tienen especialidades Math que tengan más de 30 créditos.

  Solución: De la tabla `Student`, elija aquellas filas donde la especialidad sea 'Math' y el número de créditos sea mayor que 30. Estas dos conexiones se expresan al conectarlas con 'AND'. Sólo se despliegan `lastName` y `firstName`.

Consulta SQL:

```
SELECT      lastName, firstName
FROM        Student
WHERE       major = 'Math'
            AND credits > 30;
```

Resultado:

lastName	firstName
Jones	Mary
Chin	Ann

El predicado puede ser tan complejo como sea necesario con el uso de los operadores de comparación estándar =, <>, <, <=, >, >= y los operadores lógicos estándar AND, OR y NOT, con paréntesis, si se necesita o desea, para mostrar orden de evaluación.

6.4.2 SELECT usando tablas múltiples

- Ejemplo 7. Combinación natural

  Pregunta: Encontrar las ID y nombres de todos los estudiantes que toman ART103A.

  Solución: Esta pregunta requiere el uso de dos tablas. Primero se busca en la tabla `Enroll` para registros donde el `classNumber` es 'ART103A'. Luego se busca en la tabla `Student` los registros con valores `stuId` coincidentes, y se combinan dichos registros en una nueva tabla. A partir de esta tabla, se encuentran `lastName` y `firstName`. Esto es similar a la operación JOIN en el álgebra relacional. SQL permite hacer una combinación natural, como se describe en la sección 4.6.2, al nombrar las tablas involucradas y expresar en el predicado la condición que deben cumplir los registros en el campo común.

  Consulta SQL:

```
SELECT      Enroll.stuId, lastName, firstName
FROM        Student, Enroll
WHERE       classNumber = 'ART103A'
            AND Enroll.stuId = Student.stuId;
```

  Resultado:

stuId	lastName	firstName
S1001	Smith	Tom
S1010	Burns	Edward
S1002	Chin	Ann

Note que se usó el nombre calificado por `stuId` en la línea SELECT. Pudo haberse escrito `Student.stuId` en lugar de `Enroll.stuId`, pero necesitaba usar uno de los nombres de tabla, porque `stuId` aparece en ambas tablas en la línea FROM. No necesita usar el nombre calificado para `classNumber` porque no aparece en la tabla `Student`. El hecho de que aparezca en la tabla `Class` es irrelevante, pues dicha tabla no se mencionó en la línea FROM. Desde luego, tiene que escribir ambos nombres calificados para `stuId` en la línea WHERE.

¿Por qué es necesaria la condición "`Enroll.stuId=Student.stuId`"? La respuesta es que es esencial. Cuando un sistema de base de datos relacional realiza una combinación, actúa como si primero forma un producto cartesiano, como se describió en la sección 4.6.2, de modo que (teóricamente) se forma una tabla intermedia que contiene las combinaciones de todos los registros de la tabla `Student` con los registros de la tabla `Enroll`. Incluso si el sistema se restringe a sí mismo a registros en `Enroll` que satisfagan

la condición "`classNumber`='ART103A'", la tabla intermedia numera 6*3 o 18 registros. Por ejemplo, uno de estos registros intermedios es:

S1015 Jones Mary Math 42 ART103A S1001 A

No se tiene interés en este registro, pues este estudiante no es una de las personas en la clase ART103A. Por tanto, se agrega la condición de que los valores `stuId` deben ser iguales. Esto reduce la tabla intermedia a tres registros.

- Ejemplo 8. Combinación natural con ordenamiento

  Pregunta: Encontrar `stuId` y `grade` de todos los estudiantes que tomen cualquier curso impartido por el miembro `Faculty` cuyo `facId` es F110. Ordenar en orden por `stuId`.

  Solución: Necesita buscar en la tabla `Class` para encontrar el `classNumber` de todos los cursos impartidos por F110. Luego se busca en la tabla `Enroll` los registros con valores `classNumber` coincidentes y obtener la combinación de las tablas. A partir de esto se encuentran los correspondientes `stuId` y `grade`. Puesto que se usan dos tablas, esto se escribirá como una combinación.

  Consulta SQL:

```
SELECT      stuId,grade
FROM        Class,Enroll
WHERE       facId = 'F110' AND Class.classNumber
            = Enroll.classNumber
ORDER BY    stuId ASC;
```

  Resultado:

stuId	grade
S1002	B
S1010	
S1020	A

- Ejemplo 9. Combinación natural de tres tablas

  Pregunta: Encontrar los números de curso y los nombres y especialidades de todos los estudiantes inscritos en los cursos impartidos por el miembro `Faculty` F110.

  Solución: Como en el ejemplo anterior, necesita empezar en la tabla `Class` para encontrar el `classNumber` de todos los cursos impartidos por F110. Luego se comparan éstos con los valores `classNumber` en la tabla `Enroll` para encontrar los valores `stuId` de todos los estudiantes en dichos cursos. Luego se busca en la tabla `Student` para encontrar los nombres y especialidades de todos los estudiantes inscritos en ellos.

  Consulta SQL:

```
SELECT    Enroll.classNumber, lastName,
          firstName, major
FROM      Class, Enroll, Student
WHERE     facId = 'F110'
          AND Class.classNumber =
          Enroll.classNumber
          AND Enroll.stuId = Student.stuId;
```

  Resultado:

classNumber	lastName	firstName	major
MTH101B	Rivera	Jane	CSC
MTH103C	Burns	Edward	Art
MTH103C	Chin	Ann	Math

Ésta fue una combinación natural de tres tablas y requirió dos conjuntos de columnas comunes. Se usó la condición de igualdad para ambos conjuntos en la línea WHERE. Es posible que haya notado que el orden de los nombres de la tabla en la línea FROM correspondía al orden en el que aparecieron en el plan de solución, pero esto no es necesario. SQL ignora el orden en el que las tablas se nombran en la línea FROM. Lo mismo es cierto del orden en el que se escriben las varias condiciones que constituyen el predicado en la línea WHERE. La mayoría de los sistemas de gestión de base de datos relacionales sofisticados eligen cuál tabla usar primero y cuál condición verificar primero, usando un optimizador para identificar el método más eficiente de lograr cualquier recuperación antes de elegir un plan.

- Ejemplo 10. Uso de alias

  Pregunta: Obtener una lista de todos los cursos que se imparten en el mismo salón, con sus horarios y números de salón.

  Solución: Esto requiere comparar la tabla `Class` consigo misma, y sería útil si hubiera dos copias de la tabla de modo que se podría usar una combinación natural. Puede pretender que existen dos copias de una tabla al darle dos "alias", por ejemplo, COPY y COPY2, y luego tratar estos nombres como si fuesen los nombres de dos tablas distintas. Los "alias" se introducen en la línea FROM al escribirlos inmediatamente después de los nombres de tabla reales. Luego se tienen los alias disponibles para usar en las otras líneas de la consulta.

  Consulta SQL:

```
SELECT   COPY1.classNumber, COPY1.schedule, COPY1.room,
         COPY2.classNumber, COPY2.schedule
FROM     Class COPY1, Class COPY2
WHERE    COPY1.room = COPY2.room
         AND COPY1.classNumber < COPY2.classNumber ;
```

  Resultado:

COPY1.classNumber	COPY1.schedule	COPY1.room	COPY2.classNumber	COPY2.schedule
ART103A	MWF9	H221	HST205A	MWF11
CSC201A	TUTHF10	M110	CSC203A	MTHF12
MTH101B	MTUTH9	H225	MTH103C	MWF11

Note que debe usar los nombres calificados en la línea SELECT incluso antes de introducir los "alias". Esto es necesario porque cada columna en la tabla `Class` ahora aparece dos veces, una vez en cada copia. Se agregó la segunda condición "COPY1.`classNumber` < COPY2.COURSE#" para evitar incluir a cada curso, pues cada curso obviamente satisface el requisito de que se imparte en el mismo salón como él mismo. También evita que aparezcan registros con los dos cursos invertidos. Por ejemplo, puesto que se tiene

ART103A MWF9 H221 HST205A MWF11

no se necesita el registro

HST205A MWF11 H221 ART103A MWF9

De modo incidental se pueden introducir alias en cualquier SELECT, aun cuando no se requieren.

- Ejemplo 11. Combinación sin condición de igualdad

  Pregunta: Encontrar todas las combinaciones de estudiantes y `Faculty` donde la especialidad del estudiante sea diferente del departamento del miembro `Faculty`.

  Solución: Esta solicitud inusual es para ilustrar una combinación en la que la condición no sea una igualdad en un campo común. En este caso, los campos que se exa-

minan, `major` y `department`, incluso no tienen el mismo nombre. Sin embargo, se les puede comparar ya que tienen el mismo dominio. Dado que no se dice cuáles columnas mostrar en el resultado, se usa su criterio.

Consulta SQL:

```
SELECT   stuId, lastName, firstName, major, facId, name, department
FROM     Student, Faculty
WHERE    Student.major <> Faculty.department;
```

Resultado:

```
stuId    lastName  firstName  major    facId  name    department
S1001    Smith     Tom        History  F101   Adams   Art
S1001    Smith     Tom        History  F105   Tanaka  CS
S1001    Smith     Tom        History  F110   Byrne   Math
S1001    Smith     Tom        History  F221   Smith   CS
S1010    Burns     Edward     Art      F202   Smith   History
.....................................
.....................................
.....................................
S1013    McCarthy  Owen       Math     F221   Smith   CS
```

Como en el álgebra relacional, una combinación se puede realizar sobre cualesquiera dos tablas al simplemente formar el producto cartesiano. Aunque por lo general se quiere la combinación natural como en los ejemplos anteriores, se puede usar cualquier tipo de predicado como la condición para la combinación. Sin embargo, si se quieren comparar dos columnas, deben tener los mismos dominios. Note que se usaron nombres calificados en la línea WHERE. Esto realmente no era necesario, porque cada nombre de columna era único, pero se hizo para que la condición fuese más fácil de seguir.

- Ejemplo 12. Uso de una subconsulta con igualdad

  Pregunta: Encontrar los números de todos los cursos impartidos por Byrne del departamento de matemáticas.

  Solución: Ya se sabe cómo hacer esto con el uso de una combinación natural, pero existe otra forma de encontrar la solución. En lugar de imaginar una combinación a partir de la cual elegir registros con el mismo `facId`, esto se podría visualizar como dos consultas separadas. Para la primera, se iría a la tabla `Faculty` y se encontrarían los registros con nombre de Byrne y `department` Math. Podría hacer una anotación del correspondiente `facId`. Luego podría tomar el resultado de dicha consulta, a saber F110, y buscar la tabla `Class` para registros con dicho valor en `facId`. Una vez encontrados se mostraría el `classNumber`. SQL permite secuenciar estas consultas de modo que el resultado de la primera se pueda usar en la segunda, como se muestra a continuación:

  Consulta SQL:

```
SELECT    classNumber
FROM      Class
WHERE     facId =
          (SELECT  facId
          FROM     Faculty
          WHERE    name = 'Byrne'
                   AND department = 'Math');
```

  Resultado:

```
classNumber
MTH101B
MTH103C
```

Note que este resultado podría producirse mediante la siguiente consulta SQL, usando una combinación:

```
SELECT      classNumber
FROM        Class, Faculty
WHERE       name = 'Byrne' AND department = 'Math' AND Class.facId = Faculty.
            facId;
```

En lugar de una combinación se puede usar una subconsulta, siempre que el resultado a mostrar esté contenido en una sola tabla y los datos recuperados de la subconsulta consistan sólo de una columna. Cuando escriba una subconsulta que involucre dos tablas, sólo nombre una tabla en cada SELECT. La consulta a realizar primero, la subconsulta, es la que está entre paréntesis, después de la primera línea WHERE. La consulta principal se realiza usando el resultado de la subconsulta. Normalmente se quiere que el valor de algún campo en la tabla mencionada en la consulta principal coincida con el valor de algún campo de la tabla en la subconsulta. En este ejemplo se sabía que sólo se obtendría un valor de la subconsulta, pues `facId` es la clave de `Faculty`, de modo que se produciría un valor único. Por tanto, se tiene posibilidad de usar igualdad como el operador. Sin embargo, se pueden usar condiciones distintas a la igualdad. Es posible usar cualquier operador de comparación sencillo en una subconsulta de la que usted sepa que se producirá un solo valor. Dado que la subconsulta se realiza primero, el SELECT . . . FROM . . . WHERE de la subconsulta en realidad se sustituye por el valor recuperado, de modo que la consulta principal cambia a lo siguiente:

```
SELECT      classNumber
FROM        Class
WHERE       facId = ('F110');
```

- Ejemplo 13. Subconsulta usando 'IN'

  Pregunta: Encontrar los nombres e ID de todos los miembros de `Faculty` que impartan una clase en el salón H221.

  Solución: Se necesitan dos tablas, `Class` y `Faculty`, para responder esta pregunta. También se ve que los nombres e ID aparecen ambos en la tabla `Faculty`, así que se tiene la opción de o una combinación o una subconsulta. Si se usa una subconsulta, comience con la tabla `Class` con el fin de encontrar los valores `facId` para cualquier curso que se imparta en el salón H221. Se encuentran dos de tales entradas, así que se hace una anotación de dichos valores. Luego se va a la tabla `Faculty` y se compara el valor `facId` de cada registro en dicha tabla con los dos valores ID de `Class` y se muestran los correspondientes `facId` y `name`.

  Consulta SQL:

```
SELECT    name, facId
FROM      Faculty
WHERE     facId IN
          (SELECT    facId
          FROM       Class
          WHERE      room = 'H221');
```

  Resultado:

name	facId
Adams	F101
Smith	F202

En la línea WHERE de la consulta principal se usó IN, en lugar de =, porque el resultado de la subconsulta es un conjunto de valores en vez de un solo valor. Se pide que el `facId` en `Faculty` coincida con cualquier miembro del conjunto de valores obtenidos de la subcon-

sulta. Cuando la subconsulta se sustituye con los valores recuperados, la consulta principal se convierte en

```
SELECT      name, facId
FROM        Faculty
WHERE       FACID IN ('F101','F202');
```

El IN es una forma más general de subconsulta que el operador comparación, que está restringido al caso donde se produce un solo valor. También se puede usar la forma negativa 'NOT IN', que se evaluará verdadero si el registro tiene un valor de campo que no está en el conjunto de valores recuperados por la subconsulta.

- Ejemplo 14. Subconsultas anidadas

  Pregunta: Obtener una lista alfabética de los nombres de ID de todos los estudiantes en cualquier clase que imparta F110.

  Solución: Se necesitan tres tablas, `Student`, `Enroll` y `Class`, para responder esta pregunta. Sin embargo, los valores a mostrar aparecen en una tabla, `Student`, así que se puede usar una subconsulta. Primero verifique la tabla `Class` para encontrar el `classNumber` de todos los cursos impartidos por F110. Se encuentran dos valores, MTH101B y MTH103C. A continuación se va a la tabla `Enroll` para encontrar el `stuId` de todos los estudiantes en cualquiera de estos cursos. Se encuentran tres valores: S1020, S1010 y S1002. Ahora se busca en la tabla `Student` para encontrar los registros con valores `stuId` coincidentes, y se despliegan `stuId`, `lastName` y `firstName` en orden alfabético por apellido.

  Consulta SQL:

```
SELECT     lastName, firstName, stuId
FROM       Student
WHERE      stuId IN
           (SELECT     stuId
           FROM        Enroll
           WHERE       classNumber IN
                       (SELECT  classNumber
                       FROM     Class
                       WHERE    facId = 'F110'))
ORDER BY lastName, firstName ASC;
```

  Resultado:

lastName	firstName	stuId
Burns	Edward	S1010
Chin	Ann	S1002
Rivera	Jane	S1020

En la ejecución, el SELECT más profundamente anidado se realiza primero, y se sustituye por los valores recuperados, así que se tiene:

```
SELECT    lastName, firstName, stuId
FROM      Student
WHERE     stuId IN
          (SELECT  stuId
          FROM     Enroll
          WHERE    classNumber IN
                   ('MTH101B', 'MTH103C'))
ORDER BY  lastName, firstName ASC;
```

A continuación se realiza la subconsulta sobre `Enroll` y se obtiene:

```
SELECT      lastName, firstName, stuId
FROM        Student
WHERE       stuId IN
            ('S1020', 'S1010', 'S1002')
ORDER BY    lastName, firstName ASC;
```

Finalmente, se realiza la consulta principal, y se obtiene el resultado que se mostró con anterioridad. Note que el ordenamiento se refiere al resultado final, no a cualquier paso intermedio. Observe también que cualquier parte de la operación se podía realizar como una combinación natural y la otra parte como una subconsulta, al mezclar ambos métodos.

- Ejemplo 15. Consulta usando EXISTS

  Pregunta: Encontrar los nombres de todos los estudiantes inscritos en CSC201A.

  Solución: Ya se sabe cómo escribir esto usando una combinación o una subconsulta con IN. Sin embargo, otra forma de expresar esta consulta es usar el cuantificador existencial, EXISTS, con una subconsulta.

  Consulta SQL:

```
SELECT    lastName, firstName
FROM      Student
WHERE     EXISTS
          (SELECT  *
          FROM     Enroll
          WHERE    Enroll.stuId = Student.stuId
          AND      classNumber = 'CSC201A');
```

  Resultado:

lastName	firstName
Rivera	Jane
Chin	Ann

Esta consulta se podría parafrasear como "Encuentre `lastName` y `firstName` de todos los estudiantes tales que exista un registro `Enroll` que contenga su `stuId` con un `classNumber` de CSC201A". La prueba para inclusión es la existencia de tal registro. Si existe, el "EXISTS (SELECT FROM...;" evalúa a verdadero.

Note que necesita usar el nombre de la tabla de consulta principal (`Student`) en la subconsulta para expresar la condición `Student.stuId = Enroll.stuId`. En general, se evita mencionar una tabla no citada en el FROM para dicha consulta particular, pero es necesario y permisible hacerlo en este caso. Esta forma se llama subconsulta **correlacionada**, pues la tabla en la subconsulta se compara con la tabla en la consulta principal.

- Ejemplo 16. Consulta usando NOT EXISTS

  Pregunta: Encontrar los nombres de todos los estudiantes que no están inscritos en CSC201A.

  Solución: A diferencia del ejemplo anterior, no se puede expresar fácilmente esto usando una combinación o una subconsulta IN. En vez de ello, se usará NOT EXISTS.

  Consulta SQL:

```
SELECT    lastName, firstName
FROM      Student
WHERE     NOT EXISTS
          (SELECT     *
```

```
FROM      Enroll
WHERE     Student.stuId = Enroll.stuId
AND       classNumber = 'CSC201A');
```

Resultado:

lastName	firstName
Smith	Tom
Burns	Edward
Jones	Mary
McCarthy	Owen
Lee	Perry

Esta consulta se podría parafrasear como "Seleccionar nombres de estudiante de la tabla `Student` tales que no exista registro `Enroll` que contenga sus valores `stuId` con `classNumber` de CSC201A".

6.4.3 SELECT con otros operadores

- Ejemplo 17. Consulta usando UNION

  Pregunta: Obtener las ID de todos los `Faculty` que estén asignados al departamento de historia o que den clase en el salón H221.

  Solución: Es fácil escribir una consulta para cualquiera de las condiciones y los resultados de las dos consultas se pueden combinar usando un operador UNION. La UNION en SQL es el operador estándar del álgebra relacional para unión de conjuntos y funciona en la forma esperada, lo que elimina duplicados.

  Consulta SQL:

```
SELECT    facId
FROM      Faculty
WHERE     department = 'History'
UNION
SELECT    facId
FROM      Class
WHERE     room ='H221';
```

  Resultado:

facId
F115
F101

- Ejemplo 18. Uso de funciones

  Pregunta: Encontrar el número total de estudiantes inscritos en ART103A.

  Solución: Aunque ésta es una pregunta simple, no se tiene posibilidad de expresarla como una consulta SQL en el momento, porque todavía no se ha visto alguna forma de operar sobre colecciones de filas o columnas. Se necesitan algunas funciones para hacerlo. SQL tiene cinco funciones internas: COUNT, SUM, AVG, MAX y MIN. Se usará COUNT, que regresa el número de valores en una columna.

  Consulta SQL:

```
SELECT    COUNT (DISTINCT stuId)
FROM      Enroll
WHERE     classNumber = 'ART103A';
```

  Resultado:

```
3
```

Las funciones internas operan sobre una sola columna de una tabla. Cada una de ellas elimina primero los valores nulos y sólo opera sobre los restantes valores no nulos. Las funciones regresan un solo valor, que se definen del modo siguiente:

COUNT	regresa el número de valores en la columna
SUM	regresa la suma de los valores en la columna
AVG	regresa el promedio de los valores en la columna
MAX	regresa el valor más grande en la columna
MIN	regresa el valor más pequeño en la columna

COUNT, MAX y MIN se aplican a campos numéricos y no numéricos, pero SUM y AVG sólo se pueden usar en campos numéricos. La secuencia de recopilación se usa para determinar el orden de datos no numéricos. Si se quiere eliminar valores duplicados antes de comenzar, use la palabra DISTINCT antes del nombre de columna en la línea SELECT. COUNT(*) es un uso especial de COUNT. Su propósito es contar todas las filas de una tabla, sin importar si ocurren valores nulos o duplicados. Excepto por COUNT(*), siempre debe usar DISTINCT con la función COUNT, como se hizo en el ejemplo anterior. Si usa DISTINCT con MAX o MIN no tendrá efecto, pues el valor más grande o más pequeño permanece igual incluso si dos tuplas lo comparten. Sin embargo, DISTINCT usualmente tiene un efecto sobre el resultado de SUM o AVG, así que el usuario debe entender si los duplicados se incluyen o no para calcular éste. Las referencias a función aparecen en la línea SELECT de una consulta o una subconsulta.

Ejemplos de funciones adicionales:

Ejemplo (a) Encontrar el número de departamentos que tienen `Faculty` en ellos. Puesto que no se quiere contar un departamento más de una vez, aquí se usa DISTINCT.

```
SELECT    COUNT(DISTINCT department)
FROM      Faculty;
```

Ejemplo (b) Encontrar el número promedio de créditos que tienen los estudiantes. Aquí no se quiere usar DISTINCT, porque si dos estudiantes tienen el mismo número de créditos, ambos se deben contar en el promedio.

```
SELECT    AVG(credits)
FROM      Student;
```

Ejemplo (c) Encontrar al estudiante con el número más grande de créditos. Puesto que se quiere que los créditos del estudiante sean igual al máximo, es necesario encontrar dicho máximo primero, así que se usa una subconsulta para encontrarlo.

```
SELECT     stuId, lastName, firstName
FROM       Student
WHERE      credits =
           (SELECT   MAX(credits)
           FROM      Student);
```

Ejemplo (d) Encontrar la ID del estudiante con la calificación más alta en cualquier curso. Puesto que se quiere la calificación más alta, puede parecer que aquí se debe usar la función MAX. Un vistazo más cercano a la tabla revela que las calificaciones son letras, A, B, C, etc. Para esta escala, la mejor calificación es aquella que está primero en el abecedario, así que en realidad se quiere MIN. Si las calificaciones fuesen numéricas se habría querido MAX.

```
SELECT    stuId
FROM      Enroll
WHERE     grade =
          (SELECT  MIN(grade)
          FROM     Enroll);
```

Ejemplo (e) Encontrar los nombres e ID de los estudiantes que tienen menos que el número promedio de créditos.

```
SELECT    lastName, firstName, stuId
FROM      Student
WHERE     credits <
          (SELECT  AVG(credits)
          FROM     Student);
```

- Ejemplo 19. Uso de una expresión y una constante cadena

  Pregunta: Si supone que cada curso tiene tres créditos, para cada estudiante, el número de cursos que ha completado.

  Solución: Puede calcular el número de cursos al dividir el número de créditos entre tres. Se puede usar la expresión `credits/3` en SELECT para mostrar el número de cursos. Dado que no se tiene tal nombre de columna se usará una constante cadena como etiqueta. Las constantes cadena que aparecen en la línea SELECT simplemente se imprimen en el resultado.

  Consulta SQL:

```
SELECT       stuId, 'Number of courses =', credits/3
FROM         Student;
```

  Resultado:

```
stuId
S1001 Number of courses =  30
S1010 Number of courses =  21
S1015 Number of courses =  14
S1005 Number of courses =   1
S1002 Number of courses =  12
S1020 Number of courses =   5
S1013 Number of courses =   0
```

Al combinar constantes, nombres de columna, operadores aritméticos, funciones internas y paréntesis, el usuario puede personalizar las recuperaciones.

- Ejemplo 20. Uso de GROUP BY

  Pregunta: Para cada curso, mostrar el número de estudiantes inscritos.

  Solución: Se quiere usar la función COUNT, pero es necesario aplicarla a cada curso individualmente. El GROUP BY permite poner juntos todos los registros con un solo valor en el campo especificado. Luego se puede aplicar cualquier función a cualquier campo en cada grupo, siempre que el resultado sea un solo valor para el grupo.

  Consulta SQL:

```
SELECT       classNumber, COUNT(*)
FROM         Enroll
GROUP BY     classNumber;
```

  Resultado:

```
classNumber
ART103A         3
CSC201A         2
MTH101B         1
HST205A         1
MTH103C         2
```

Note que, en esta consulta, podría usar COUNT(DISTINCT `stuId`) en lugar de COUNT(*).

- Ejemplo 21. Uso de HAVING

  Problema: Encontrar todos los cursos en los que estén inscritos menos de tres estudiantes.

  Solución: Ésta es una pregunta acerca de una característica de los grupos formados en el ejemplo anterior. HAVING se usa para determinar cuáles grupos tienen alguna cualidad, tal como WHERE se usa con tuplas para determinar cuáles registros tienen alguna cualidad. No se permite usar HAVING sin un GROUP BY, y el predicado en la línea HAVING debe tener un solo valor para cada grupo.

  Consulta SQL:

```
SELECT     classNumber
FROM       Enroll
GROUP BY   classNumber
HAVING     COUNT(*) < 3 ;
```

  Resultado:

```
classNumber
CSC201A
MTH101B
HST205A
MTH103C
```

- Ejemplo 22. Uso de LIKE

  Problema: Obtener detalles de todos los cursos MTH.

  Solución: No se quiere especificar los números de curso exactos, lo que se quiere es que las tres primeras letras de `classNumber` sean MTH. SQL permite el uso de LIKE en el predicado con el fin de mostrar una cadena patrón para campos carácter. Se recuperarán los registros cuyas columnas especificadas coincidan con el patrón.

  Consulta SQL:

```
SELECT     *
FROM       Class
WHERE      classNumber LIKE 'MTH%';
```

  Resultado:

```
classNumber     facId        schedule       room
MTH101B         F110         MTUTH9         H225
MTH103C         F110         MWF11          H225
```

En la cadena patrón se pueden usar los siguientes símbolos:

- % El carácter porcentaje representa cualquier secuencia de caracteres de cualquier longitud >= 0.
- _ El carácter guión bajo representa cualquier carácter solo.

Todos los otros caracteres en el patrón se representan ellos mismos.

Ejemplos:

- **`classNumber` LIKE `'MTH%'`** significa que las tres primeras letras deben ser MTH, pero el resto de la cadena puede ser cualquier carácter.
- **`stuId` LIKE `'S_ _ _ _'`** significa que debe haber cinco caracteres, el primero de los cuales debe ser S.
- **`schedule` LIKE `'%9'`** significa cualquier secuencia de caracteres, de longitud al menos uno, con el último carácter un nueve.

- **`classNumber` LIKE `'%101%'`** significa una secuencia de caracteres de cualquier longitud que contenga 101. Note que 101 podría ser el primero, último o únicos caracteres, así como estar en cualquier parte en medio de la cadena.
- **`name` NOT LIKE `'A%'`** significa que el nombre no puede comenzar con A.

- Ejemplo 23. Uso de NULL

  Pregunta: Encontrar el `stuId` y `classNumber` de todos los estudiantes cuyas calificaciones faltan en dicho curso.

  Solución: De la tabla `Enroll` se puede ver que existen dos de tales registros. Puede pensar que podría acceder a ellos al especificar que las calificaciones no sean A, B, C, D o F, pero éste no es el caso. Una calificación nula se considera que tiene "desconocido" como valor, así que es imposible juzgar si es igual o no es igual a otra calificación. Si se pusiera la condición "WHERE grade <>'A' AND grade <>'B' AND grade <>'C' AND grade <>'D' AND grade <>'F'" se obtendría de vuelta una tabla vacía, en lugar de los dos registros que se quieren. SQL usa la expresión lógica

  nombre_columna IS [NOT] NULL

  a fin de probar para valores nulos en una columna.

  Consulta SQL:

```
SELECT     classNumber,stuId
FROM       Enroll
WHERE      grade IS NULL;
```

  Resultado:

classNumber	stuId
ART103A	S1010
MTH103C	S1010

Note que es ilegal escribir "WHERE `grade` = NULL", porque un predicado que involucre operadores comparación con NULL evaluará para "desconocido" en lugar de "verdadero" o "falso". Además, la línea WHERE es la única en la que NULL puede aparecer en un enunciado SELECT.

- Ejemplo 24. Consultas recursivas

  SQL: 1999 permite consultas recursivas, que son consultas que se ejecutan repetidamente hasta que no se encuentran nuevos resultados. Por ejemplo, considere una tabla CSCCourse, como se muestra en la figura 6.4(a). Su estructura es:

  `CSCCourse(cursoNumero, cursoTitulo, creditos, prerrequisitoCursoNumero)`

  Por simplicidad, se supone que un curso puede tener cuando mucho un curso prerrequisitos inmediato. El número de curso prerrequisitos funciona como una clave externa para la tabla CSCCourse, que se refiere a la clave primaria (número de curso) de un curso diferente.

  Problema: Encontrar todos los prerrequisitos de un curso, incluidos prerrequisitos de prerrequisitos para dicho curso.

  Consulta SQL:

```
WITH RECURSIVE
Prereqs (cursoNumero, prerrequisitoCursoNumero) AS
  (  SELECT cursoNumero, prerrequisitoCursoNumero
     FROM CSCCourse
     UNION
     SELECT (COPY1.cursoNumero, COPY2.prerrequisitoCursoNumero
     FROM Prereqs COPY1, CSCCourse COPY2
     WHERE COPY1.prerrequisitoCursoNumero = COPY2.cursoNumero);
```

```
SELECT *
FROM Prereqs
ORDER BY cursoNumero, prerrequisitoCursoNumero;
```

Esta consulta desplegará cada número de curso, junto con todos los prerrequisitos de dicho curso, incluidos los prerrequisitos de prerrequisitos, etc., toda la ruta hasta el curso inicial en la secuencia de sus prerrequisitos. El resultado se muestra en la figura 6.4(b).

6.4.4 Operadores para actualización: UPDATE, INSERT, DELETE

El operador UPDATE (actualizar) se usa para cambiar valores en registros ya almacenados en una tabla. Se usa en una tabla a la vez y puede cambiar cero, uno o muchos registros, dependiendo del predicado. Su forma es:

```
UPDATE    nombre tabla
SET       nombre_columna = expresión
          [nombre_columna = expresión] . . .
[WHERE    predicado];
```

Note que no es necesario especificar el valor actual del campo, aunque el valor presente se puede usar en la expresión para determinar el nuevo valor. El enunciado SET es en realidad un enunciado de asignación y funciona en la forma usual.

- Ejemplo 1. Actualización de un solo campo de un registro

  Operación: Cambiar la especialidad de S1020 a Music.

  Comando SQL:

  ```
  UPDATE  Student
  SET     major = 'Music'
  WHERE   stuId = 'S1020';
  ```

- Ejemplo 2. Actualización de varios campos de un registro

  Operación: Cambiar el departamento de Tanaka a MIS y clasificarlo como asistente (Assistant).

  Comando SQL:

  ```
  UPDATE  Faculty
  SET     department = 'MIS'
          rank = 'Assistant'
  WHERE   name = 'Tanaka';
  ```

CSCCourse

cursoNumero	cursoTitulo	creditos	prerrequisitoCursoNumero
101	Introducción a la computación	3	
102	Aplicaciones de computadoras	3	101
201	Programación 1	4	101
202	Programación 2	4	201
301	Estructuras de datos y algoritmos	3	202
310	Sistemas operativos	3	202
320	Sistemas de bases de datos	3	301
410	Sistemas operativos avanzados	3	310
420	Sistemas de base de datos avanzados	3	320

FIGURA 6.4(a)
Tabla CSCCurso para demostrar las consultas recursivas

FIGURA 6.4(b)
Resultado de consulta recursiva

Prereqs

cursoNumero	prerrequisitoCursoNumero
101	
102	
102	101
201	
201	101
202	
202	101
202	201
301	
301	101
301	201
301	202
310	
310	101
310	201
310	202
320	
320	101
320	201
320	202
320	301
410	
410	101
410	201
410	202
410	310
420	
420	101
420	201
420	202
420	301
420	320

- Ejemplo 3. Actualización usando NULL

  Operación: Cambiar la especialidad de S1013 de Math a NULL. Para insertar un valor nulo en un campo que ya tiene un valor real, debe usar la forma:

  SET *nombre_columna* = NULL

  Comando SQL:

```
UPDATE  Student
SET     major = NULL
WHERE   stuId = 'S1013';
```

- Ejemplo 4. Actualización de varios registros

  Operación: Cambiar las calificaciones de todos los estudiantes en CSC201A a A.

  Comando SQL:

```
UPDATE     Enroll
SET        grade = 'A'
WHERE      classNumber = 'CSC201A';
```

- Ejemplo 5. Actualización de todos los registros.

  Operación: Dar a todos los estudiantes tres créditos adicionales.

  Comando SQL:

```
UPDATE  Student
SET     credits = credits + 3;
```

Note que no se necesita la línea WHERE, porque se actualizaron todos los registros.

- Ejemplo 6. Actualización con una subconsulta

  Operación: Cambiar el salón a B220 para todos los cursos que imparte Tanaka.

  Comando SQL:

```
UPDATE  Class
SET     room = 'B220'
WHERE   facId =
        (SELECT     facId
        FROM        Faculty
        WHERE       name = 'Tanaka');
```

El operador INSERT se usa para poner nuevos registros en una tabla. Por lo general, no se usa para cargar toda una tabla, porque el sistema de gestión de base de datos usualmente tiene una utilidad de carga (load) para manejar dicha tarea. Sin embargo, INSERT es útil para agregar uno o algunos registros a una tabla. Su forma es:

```
INSERT
INTO      nombre_tabla [(nombre_col [,nombre_col]. . .)]
VALUES    (constante [,constante] . . .);
```

- Ejemplo 1. Insertar un solo registro, con todos los campos especificados

  Operación: Insertar un nuevo registro `Faculty` con ID de F330, nombre Jones, departamento CSC y clasificación de Instructor.

  Comando SQL:

```
INSERT
INTO    Faculty (facId, name, department, rank)
VALUES  ('F330', 'Jones', 'CSC', 'Instructor');
```

- Ejemplo 2. Insertar un solo registro, sin especificar campos

  Operación: Insertar un nuevo registro de estudiante con ID de S1030, nombre Alice Hunt, especialidad art y 12 créditos

  Consulta SQL:

```
INSERT
INTO    Student
VALUES  ('S1030', 'Hunt', 'Alice', 'Art', 12);
```

Note que no fue necesario especificar nombres de campo, porque el sistema supone que se quieren todos los campos en la tabla. Podría hacer lo mismo para el ejemplo anterior.

- Ejemplo 3. Insertar un registro con valor nulo en un campo

  Operación: Insertar un nuevo registro de estudiante con ID de S1031, nombre Maria Bono, cero créditos y sin especialidad.

  Comando SQL:

```
INSERT
INTO    Student (lastName, firstName, stuId, credits)
VALUES  ('Bono', 'Maria', 'S1031', 0);
```

Note que se reordenaron los nombres de campo, pero no hay confusión porque se entiende que el orden de los valores coincide con el orden de los campos mencionados en el INTO, sin importar su orden en la tabla. Note también que el cero es un valor real para `credits`, no un valor nulo. `major` se establecerá en nulo, pues se excluye de la lista de campos en la línea INTO.

- Ejemplo 4. Insertar múltiples registros

  Operación: Crear y llenar una nueva tabla que muestre cada curso y el número de estudiantes inscritos en ellos.

  Comando SQL:

```
CREATE TABLE Enrollment
  (classNumber CHAR(7) NOT NULL,
  Students SMALLINT);
INSERT
INTO Enrollment        (classNumber, Students)
  SELECT               classNumber, COUNT(*)
  FROM                 Enroll
  GROUP BY             classNumber;
```

Aquí se creó una nueva tabla, `Enrollment`, y se llenó al tomar datos de una tabla existente, `Enroll`. Si `Enroll` es como aparece en la figura 6.3, `Enrollment` ahora se ve como esto:

Enrollment

classNumber	Students
ART103A	3
CSC201A	2
MTH101B	1
HST205A	1
MTH103C	2

La tabla `Enrollment` ahora está disponible para que el usuario la manipule, tal como lo sería cualquiera otra tabla. Se puede actualizar según se requiera, pero no se actualizará automáticamente cuando `Enroll` se actualice.

El DELETE se usa para borrar registros. El número de registros borrados puede ser cero, uno o muchos, dependiendo de cuántos satisfagan el predicado. La forma de este comando es:

```
DELETE
FROM        nombre_tabla
WHERE       predicado;
```

- Ejemplo 1. Borrado de un solo registro

  Operación: Borrar el registro del estudiante S1020.

  Comando SQL:

```
DELETE
FROM        Student
WHERE       stuId = 'S1020';
```

- Ejemplo 2. Borrado de muchos registros

  Operación: Borrar todos los registros de inscripción para el estudiante S1020.

  Comando SQL:

```
DELETE
FROM        Enroll
WHERE       stuId = 'S1020';
```

- Ejemplo 3. Borrado de todos los registros de una tabla

  Operación: Borrar todos los registros de clase.

Si borra los registros `Class` y permite que permanezcan sus correspondientes registros `Enroll` se perdería integridad referencial, porque los registros `Enroll` se referirían entonces a clases que ya no existen. Sin embargo, si las tablas se crearon con Oracle y los comandos que se muestran en la figura 6.2, el comando DELETE no funcionará en la tabla `Class` a menos que primero se borren los registros `Enroll` para cualquier estudiante registrado en la clase, porque se escribió

```
CONSTRAINT Enroll_classNumber_fk FOREIGN KEY (classNumber) REFERENCES Class
(classNumber)
```

para la tabla `Enroll`. No obstante, si supone que se borraron los registros `Enroll`, entonces se pueden borrar los registros `Class`.

Comando SQL:

```
DELETE
FROM       Class;
```

Esto removería todos los registros de la tabla `Class`, pero su estructura permanecería, así que en ella podría agregar nuevos registros en cualquier momento.

- Ejemplo 4. DELETE con una subconsulta

  Operación: Borrar todos los registros de inscripción para Owen McCarthy.

  Comando SQL:

```
DELETE
FROM        Enroll
WHERE       stuId =
            (SELECT    stuId
            FROM       Student
            WHERE      lastName = 'Mc Carthy'
                       AND firstName = 'Owen');
```

Puesto que no existían tales registros, este enunciado no tendrá efecto sobre `Enroll`.

6.5 Bases de datos activas

El DBMS tiene más poder al asegurar la integridad en una base de datos que las restricciones de columna y tabla discutidas en la sección 6.3. Una **base de datos activa** es aquella en la que el DBMS monitorea a los contenidos con la finalidad de evitar que ocurran estados ilegales, usando restricciones y disparadores (triggers).

6.5.1 Habilitación y deshabilitación de restricciones

Las restricciones de columna y tabla descritas en la sección 6.3 se identifican cuando se crea la tabla, y se verifican siempre que se realiza un cambio a la base de datos, para garantizar

que el nuevo estado es válido. Los cambios que podrían resultar en estados inválidos incluye inserción, borrado y actualización de registros. Por tanto, el DBMS verifica las restricciones siempre que se realiza una de estas operaciones, por lo general después de cada enunciado SQL INSERT, DELETE o UPDATE. A esto se le llama el modo IMMEDIATE (inmediato) de verificación de restricción, y es el modo por defecto. Sin embargo, hay ocasiones cuando una transacción o aplicación involucra varios cambios y la base de datos será temporalmente inválida mientras los cambios están en progreso, mientras algunos aunque no todos los cambios ya se realizaron. Por ejemplo, suponga que tiene una tabla `Department` en el ejemplo University con una columna `chairPerson` en la que se menciona el `facId` del jefe de departamento, y se usa una especificación NOT NULL para dicho campo y se le convierte en clave externa al escribir:

```
CREATE TABLE Department (
   deptName       CHAR(20),
   chairPerson    CHAR(20) NOT NULL,
   CONSTRAINT Department_deptName_pk PRIMARY KEY(deptName),
   CONSTRAINT Department_facId_fk FOREIGN KEY REFERENCES Faculty (facId));
```

Por la definición de tabla `Faculty` en la figura 6.2, ya se tiene un NOT NULL para el departamento, pero ahora se podría convertir a `department` en una clave externa con la nueva tabla `Department` como la tabla de origen, al escribir:

```
ALTER TABLE Faculty ADD CONSTRAINT Faculty_department_fk FOREIGN KEY
department REFERENCES Department(deptName);
```

¿Qué ocurrirá cuando intente crear un nuevo departamento y agregar al personal docente para dicho departamento a la tabla `Faculty`? No se puede insertar el registro del departamento a menos que ya se tenga el registro del jefe del mismo en la tabla `Faculty`, pero no se puede insertar dicho registro a menos que el registro del departamento ya esté ahí, así que cada enunciado SQL INSERT fallaría. Para tales situaciones, SQL permite diferir la verificación hasta el final de toda la transacción, usando enunciados como

```
SET CONSTRAINT Department_facId_fk DEFERRED;
```

o

```
SET CONSTRAINT Faculty_department_fk DEFERRED;
```

Las restricciones se pueden habilitar o deshabilitar para permitir el éxito de tales transacciones, usando un enunciado como:

```
DISABLE CONSTRAINT Department_facId_fk;
```

Al final de la transacción escribiría

```
ENABLE CONSTRAINT Department_facId_fk;
```

para permitir de nuevo el fortalecimiento. Aunque no se recomienda, SQL permite escribir

```
DISABLE ALL CONSTRAINTS;
```

que suspende toda la verificación de integridad hasta encontrar un comando

```
ENABLE ALL CONSTRAINTS;
```

correspondiente. Estos enunciados se pueden usar tanto interactivamente como dentro de las aplicaciones.

6.5.2 Disparadores (triggers) SQL

Como las restricciones, los **disparadores** (triggers) permiten al DBMS monitorear la base de datos. Sin embargo, son más flexibles que las restricciones, se aplican a un rango más amplio de situaciones y permiten una mayor variedad de acciones. Un disparador consiste en tres partes:

- Un **evento**, que normalmente es algún cambio hecho a la base de datos
- Una **condición**, que es un predicado lógico que evalúa a verdadero o falso
- Una **acción**, que es algún procedimiento que se realiza cuando ocurre el evento y la condición evalúa a verdadero, también llamado disparar el disparador

Un disparador tiene acceso a los datos insertados, borrados o actualizados que causaron su disparo (es decir, a activarse o elevarse), y los valores de datos se pueden usar en el código tanto para la condición como para la acción. El prefijo :`OLD` se usa para hacer referencia a los valores en una tupla recién borrada o a los valores sustituidos en una actualización. El prefijo :`NEW` se usa para referirse a los valores en una tupla recién insertada o a los nuevos valores en una actualización. Los disparadores se pueden disparar o antes o después de la ejecución de la operación insertar, borrar o actualizar. También puede especificar si el disparador se dispara sólo una vez por cada enunciado de disparo, o por cada fila que cambie por el enunciado (recuerde que, por ejemplo, un enunciado de actualización puede cambiar muchas filas). En Oracle se especifica el comando SQL (INSERT, DELETE o UPDATE) que es el evento; la tabla involucrada; el nivel del disparador (ROW, fila, o STATEMENT, enunciado); la temporización (BEFORE, antes, o AFTER, después), y la acción a realizar, que se puede escribir como uno o más comandos SQL en PL/SQL. La sección 6.7 discute PL/SQL con más detalle. La sintaxis de disparador Oracle tiene la forma:

```
CREATE OR REPLACE TRIGGER nombre_disparador
  [BEFORE/AFTER] [INSERT/UPDATE/DELETE] ON nombre_tabla
  [FOR EACH ROW] [WHEN condición]
  BEGIN
      cuerpo del disparador
  END;
```

Por ejemplo, agregue a la tabla `Class` dos atributos adicionales, `currentEnroll`, que muestra el número de estudiantes actualmente inscritos en cada clase, y `maxEnroll`, que es el número máximo de estudiantes con permiso para inscribirse. La nueva tabla, `RevClass`, se muestra en la figura 6.5, junto con una nueva versión de la tabla `Enroll`, `RevEnroll`. Puesto que `currentEnroll` es un atributo derivado, dependiente de la tabla `RevEnroll`, su valor se debe actualizar cuando existan cambios relevantes hechos a `RevEnroll`. Los cambios que afectan a `currentEnroll` son:

1. Un estudiante que se inscribe en una clase
2. Un estudiante que da de baja una clase
3. Un estudiante que cambia de una clase a otra

En una base de datos activa debe haber disparadores para cada uno de estos cambios. Para el cambio (1) debe incrementar el valor de `currentEnroll` por uno. Se puede referir al `classNumber` del nuevo registro `RevEnroll` con el uso del prefijo :NEW. El disparador correspondiente se muestra en la figura 6.5(b). Note que no se usó WHEN porque siempre se quiere hacer este cambio después de insertar un nuevo registro de inscripción, así que no se necesitó condición alguna. Para el cambio (2) se necesita disminuir el valor de `currentEnroll` por uno, y se usa el prefijo: OLD para referirse al registro `RevEnroll` a borrar. El disparador se muestra en la figura 6.5(c). El cambio (3) se podría tratar como una baja seguida por una inscripción, pero en vez de ello se escribirá un disparador para una actualización a fin de demostrar una acción con dos partes, como se muestra en la figura 6.5(d).

Aunque estos disparadores son suficientes si hay espacio en una clase, también se necesita considerar lo que ocurriría si la clase ya tiene inscripción completa. Debió examinar el valor de `maxEnroll` antes de permitir a un estudiante su inscripción en la clase, así que necesita verificar dicho valor antes de hacer los cambios (1) o (3). Suponga que si un cambio causaría que una clase tuviera inscripción en exceso, se llamaría un procedimiento llamado

FIGURA 6.5(a)

Tablas para disparadores

RevClass

classNumber	facId	schedule	room	currentEnroll	maxEnroll
ART103A	F101	MWF9	H221	3	25
CSC201A	F105	TuThF10	M110	2	20
CSC203A	F105	MThF12	M110	0	20
HST205A	F115	MWF11	H221	1	35
MTH101B	F110	MTuTh9	H225	1	25
MTH103C	F110	MWF11	H225	2	25

RevEnroll

stuId	classNumber	grade
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

FIGURA 6.5(b)

Disparador para inscripción de estudiante en una clase

```
CREATE TRIGGER ADDENROLL
AFTER INSERT ON RevEnroll
FOR EACH ROW
BEGIN
    UPDATE RevClass
    SET currentEnroll = currentEnroll +1
    WHERE RevClass.classNumber = :NEW.classNumber;
END;
```

FIGURA 6.5(c)

Disparador para un estudiante que se da de baja en una clase

```
CREATE TRIGGER DROPENROLL
AFTER DELETE ON RevEnroll
FOR EACH ROW
BEGIN
    UPDATE RevClass
    SET currentEnroll = currentEnroll −1
    WHERE RevClass.classNumber = OLD.classNumber;
END;
```

```
CREATE TRIGGER SWITCHENROLL
AFTER UPDATE OF classNumnber ON RevEnroll
FOR EACH ROW
    BEGIN
        UPDATE RevClass
        SET currentEnroll = currentEnroll + 1
        WHERE RevClass.classNumber = :NEW.classNumber;
        UPDATE RevClass
        SET currentEnroll = currentEnroll –1
        WHERE RevClass.classNumber = :OLD.classNumber;
    END;
```

FIGURA 6.5(d)
Disparador para estudiante que cambia de clases

```
CREATE TRIGGER ENROLL_REQUEST
BEFORE INSERT OR UPDATE OF classNumber ON REVenroll
FOR EACH ROW
DECLARE
    numStu number;
    maxStu number;
BEGIN
    select   maxEnroll into maxStu
    from     RevClass
    where   RevClass.classNumber = :NEW.classNumber;

    select  currentEnroll +1 into numStu
    from     RevClass
    where   RevClass.classNumber = :NEW.classNumber;

    if numStu > maxStu
    RequestClosedCoursePermission(:NEW.stuId, :NEW.classNumber, RevClass.currentEnroll, RevClass.
   maxEnroll);
    end if;
END;
```

FIGURA 6.5(e)
Disparador para verificar la inscripción en exceso antes de inscribir a un estudiante

RequestClosedCoursePermission (solicitud de permiso de curso cerrada) que tomaría como parámetros la ID del estudiante, el número de clase, la inscripción actual y la inscripción máxima. La acción se debe tomar antes de hacer el cambio. El disparador se muestra en la figura 6.5(e).

Los disparadores se habilitan automáticamente cuando se crean. Se pueden deshabilitar con el enunciado:

ALTER TRIGGER *nombre_disparador* DISABLE;

Después de deshabilitar se pueden habilitar de nuevo mediante el enunciado:

ALTER TRIGGER *nombre_disparador* ENABLE;

Se pueden eliminar mediante el enunciado:

DROP TRIGGER *nombre_disparador*;

Oracle proporciona una forma INSTEAD OF (en lugar de) que es especialmente útil cuando un usuario intenta actualizar la base de datos a través de una vista. Esta forma especifica

una acción a realizar en lugar de la inserción, borrado o actualización que el usuario solicita. Se discutirá en la sección 6.8. Los disparadores también se pueden usar a fin de proporcionar una vía de auditoría para una tabla, registrar todos los cambios, el momento cuando se hicieron y la identidad del usuario que los hizo, una aplicación que se discutirá en la sección 9.6.

6.6 Uso de los enunciados COMMIT y ROLLBACK

Cualquier cambio hecho a una base de datos usando comandos SQL no es permanente hasta que el usuario escribe un enunciado COMMIT (compromiso). Como se discute en el capítulo 10, una transacción SQL termina cuando se encuentra o un enunciado COMMIT o un enunciado ROLLBACK (retrocesión). El COMMIT hace permanentes los cambios realizados desde el comienzo de la transacción actual, que es o el comienzo de la sesión o el momento desde el último COMMIT o ROLLBACK. El ROLLBACK deshace los cambios realizados por la transacción actual. Es aconsejable escribir COMMIT con frecuencia para guardar los cambios mientras trabaja.

6.7 Programación SQL

Para los enunciados SQL interactivos mostrados hasta el momento, se supone que había una interfaz de usuario que proporcionaba un entorno para aceptar, interpretar y ejecutar directamente los comandos SQL. Un ejemplo es la facilidad SQLPlus de Oracle. Aunque algunos usuarios pueden interactuar con la base de datos de esta forma, la mayoría de los accesos a la base de datos es a través de programas.

6.7.1 SQL incrustado (embedded)

Una forma en que se puede usar SQL es incrustarlo en programas escritos en un lenguaje de programación de propósito general como C, C++, Java, COBOL, Ada, Fortran, Pascal, PL/1 o M, a los que se les conoce como **lenguaje huésped**. Cualquier comando SQL interactivo como los discutidos, se puede usar en un programa de aplicación, como modificaciones menores. Los programadores escriben el código fuente usando tanto enunciados de lenguaje huésped, que proporcionan las estructuras de control, como enunciados SQL, que gestionan el acceso a la base de datos. Los enunciados SQL ejecutables están precedidos por un prefijo como la palabra clave EXEC SQL y terminan con un terminador como un punto y coma. Un enunciado SQL ejecutable puede aparecer en cualquier parte que pueda aparecer un enunciado en lenguaje huésped ejecutable. El DBMS proporciona un precompilador, que barre todo el programa y extrae los enunciados SQL, identificados por el prefijo. El SQL se compila por separado en algún tipo de módulo de acceso y los enunciados SQL se sustituyen, por lo general mediante llamadas simples de función en el lenguaje huésped. El programa resultante en lenguaje huésped puede entonces compilarse como siempre. La figura 6.6 ilustra este proceso.

El intercambio de datos entre el programa de aplicación y la base de datos se logra a través de las variables de programa en lenguaje huésped, para cuyos atributos de registros de base de datos se proporcionan valores o de los cuales reciben sus valores. Estas **variables compartidas** se declaran dentro del programa en una sección de declaración SQL como la siguiente:

```
EXEC SQL BEGIN DECLARE SECTION;
char stuNumber[5];
char stuLastName[15];
char stuFirstName[12];
```

```
char stuMajor[10];
int stuCredits;
char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

Las primeras cinco variables de programa declaradas aquí están diseñadas para coincidir con los atributos de la tabla `Student`. Se pueden usar para recibir valores de las tuplas `Student`, o suministrar valores a las tuplas. Puesto que los tipos de datos del lenguaje huésped pueden no coincidir exactamente con las de los atributos de la base de datos, se puede usar un tipo de operación cast (molde) para pasar valores apropiados entre variables de programa y atributos de base de datos. SQL:1999 proporciona vínculos de lenguaje que especifican la correspondencia entre tipos de datos SQL y los tipos de datos de cada lenguaje huésped.

La última variable declarada, SQLSTATE, es un arreglo carácter que se usa para comunicar condiciones de error al programa. Cuando se llama una función de biblioteca SQL se coloca un valor en SQLSTATE que indica cuál condición de error pudo haber ocurrido cuando se accedió a la base de datos. Un valor de '00000' indica no error, mientras que un valor '02000' indica que el enunciado SQL se ejecutó correctamente pero no se encontró tupla para una consulta. El valor de SQLSTATE se puede probar en enunciados de control de lenguaje huésped. Por ejemplo, se puede realizar algún procesamiento si el valor es '00000', o se puede escribir una iteración para leer un registro hasta que regrese el valor de '02000'.

Los enunciados SELECT de SQL incrustado que operan en una sola fila de una tabla son muy similares a los enunciados interactivos que se vieron anteriormente. Por ejemplo, un enunciado SELECT que recupera una sola tupla se modifica para el caso incrustado al identificar las variables compartidas en una línea INTO. Note que, cuando se hace referencia a variables compartidas dentro de un enunciado SQL, están precedidas por dos puntos para distinguirlas de los atributos, que pueden o no tener los mismos nombres. El siguiente ejemplo comienza con un enunciado en lenguaje huésped que asigna un valor a la variable huésped `stuNumber`. Luego usa un enunciado SQL SELECT para recuperar una tupla de

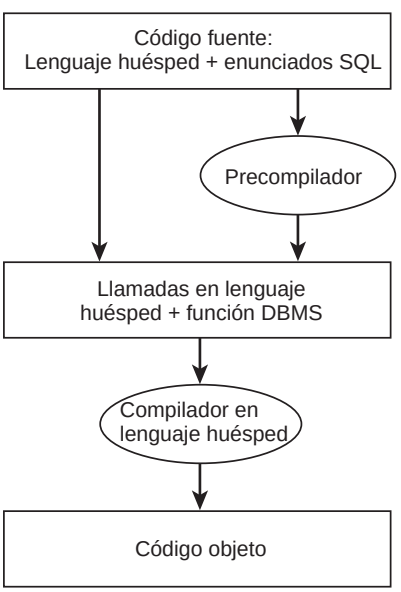


FIGURA 6.6
Procesamiento de programas SQL incrustados

la tabla `Student` cuyo `stuId` coincida con la variable compartida `stuNumber` (referida como :`stuNumber` en el enunciado SQL). Pone los valores de atributo de tupla en cuatro variables compartidas que se declararon previamente:

```
stuNumber = 'S1001';
EXEC SQL  SELECT  Student.lastName, Student.firstName, Student.major,
Student.credits
   INTO :stuLastName, :stuFirstName, :stuMajor, :stuCredits
   FROM Student
   WHERE Student.stuId = :stuNumber;
```

Este segmento se debe seguir mediante una verificación en lenguaje huésped del valor de SQLSTATE.

Para insertar una tupla de base de datos, puede asignar valores a variables compartidas en el lenguaje huésped y luego usar un enunciado SQL INSERT. Los valores de atributo se toman de las variables huésped. Por ejemplo, se podría escribir:

```
stuNumber = 'S1050';
stuLastName = 'Lee';
stuFirstName = 'Daphne';
stuMajor = 'English';
stuCredits = 0;
EXEC SQL     INSERT
           INTO Student (stuId, lastName, firstName, major, credits)
           VALUES(:stuNumber,:stuLastName, :stuFirstName,:stuMajor,
:stuCredits);
```

Puede borrar cualquier número de tuplas que pueda identificar usando el valor de una variable huésped:

```
stuNumber = 'S1015';
EXEC SQL    DELETE
        FROM Student
        WHERE stuId = :stuNumber;
```

También puede actualizar cualquier número de tuplas en una forma similar:

```
stuMajor = 'History';
EXEC SQL UPDATE Student
        SET CREDITS = CREDITS + 3
        WHERE major = :stuMajor;
```

En lugar de verificar el valor de SQLSTATE después de cada enunciado SQL, puede usar el enunciado manipulador de error WHENEVER (siempre que) que tiene la forma:

```
EXEC SQL WHENEVER [SQLERROR/NOT FOUND] [CONTINUE/GO TO enunciado];
```

Como se mencionó antes, una condición NOT FOUND significa que el valor de SQLSTATE es 02000, mientras que la condición SQLERROR es cualquier excepción. Un WHENEVER permanece en efecto durante todo el programa para la condición especificada, a menos que aparezca un nuevo WHENEVER para la misma condición, que pasa por encima de la primera.

Un problema especial llamado **desajuste de impedancia** ocurre cuando un enunciado SELECT regresa más de una tupla (un multiconjunto). Aunque el modelo relacional usa conjuntos, un lenguaje huésped generalmente es capaz de manipular sólo un registro a la vez en lugar de todo un conjunto de registros. Es necesario un dispositivo llamado **cursor**, un puntero simbólico que apunta a una fila de una tabla o multiconjunto a la vez, proporciona a la aplicación acceso a dicha fila, y luego se mueve hacia la siguiente fila. Para una consulta SQL que regresa un multiconjunto, el cursor se puede usar para avanzar a través de

los resultados de la consulta, lo que permite la dotación de valores al lenguaje huésped tupla por tupla. Un cursor se crea y coloca de modo que puede apuntar a una nueva fila en la tabla o en el multiconjunto a la vez. La forma para declarar un cursor es:

EXEC SQL DECLARE *nombre_cursor* [INSENSITIVE] [SCROLL] CURSOR FOR *consulta*
[FOR {READ ONLY | UPDATE OF *nombre_atributos*}];

Por ejemplo, para crear un cursor que más tarde se usará para ir a través de los registros de estudiantes CSC que sólo se planea recuperar, se escribiría:

```
EXEC SQL DECLARE CSCstuCursor CURSOR FOR
   SELECT stuId, lastName, firstName, major, credits
   FROM student
   WHERE major='CSC';
```

Note que ésta es una declaración, no un enunciado ejecutable. La consulta SQL no se ejecuta todavía. Después de declarar el cursor se escribe un enunciado para abrirlo. Éste ejecuta la consulta de modo que se crean los resultados multiconjunto. Abrir el cursor también lo coloca justo antes de la primera tupla del conjunto de resultados. Para este ejemplo, escriba:

```
EXEC SQL OPEN CSCstuCursor;
```

Para recuperar la primera fila de los resultados se usa entonces el comando FETCH (lectura de instrucción) que tiene la forma

```
EXEC SQL FETCH cursorname INTO hostvariables;
```

como en

```
EXEC SQL FETCH CSCstuCursor INTO :stuNumber,:stuLastName, :stuFirstName,:stuMajor,
   :stuCredits
```

El enunciado FETCH avanza el cursor y asigna los valores de los atributos mencionados en el enunciado SELECT a las correspondientes variables compartidas mencionadas en la línea INTO. Una iteración controlada por el valor de SQLSTATE (por ejemplo, WHILE (SQLSTATE=00000)) se debe crear en el lenguaje huésped de modo que se acceda a filas adicionales. La iteración también contiene enunciados en lenguaje huésped para hacer cualquier procesamiento que sea necesario. Después de recuperar todos los datos, se sale de la iteración y se cierra el cursor en un enunciado como:

```
EXEC SQL CLOSE CSCstuCursor;
```

Si planea actualizar múltiples filas usando el cursor, inicialmente debe declararlo como actualizable, con el uso de una declaración más completa como:

```
EXEC  SQL DECLARE stuCreditsCursor CURSOR FOR
      SELECT stuId, credits
      FROM Student
FOR UPDATE OF credits;
```

Se debe nombrar, tanto en el enunciado SELECT como en la lista de actualización de atributos, cualquier atributo que se planee actualizar usando el cursor. Una vez que el cursor se abra y active, puede actualizar la tupla donde se coloca el cursor, llamado el **actual** (current) del cursor, al escribir un comando como:

```
EXEC SQL UPDATE Student
SET credits = credits +3
WHERE CURRENT OF stuCreditsCursor;
```

De igual modo, la tupla actual se puede borrar mediante un enunciado como:

```
EXEC SQL DELETE FROM Student
WHERE CURRENT OF stuCreditCursor;
```

Es posible que los usuarios quieran poder especificar el tipo de acceso que necesitan en el tiempo de corrido en lugar de en una forma estática usando código compilado del tipo recién descrito. Por ejemplo, es posible que quiera un frente gráfico donde el usuario pueda ingresar una consulta que se pueda usar para generar enunciados SQL que se ejecuten dinámicamente, como el SQL interactivo descrito antes. Además de la versión de SQL apenas discutida, que se clasifica como estática, existe un **SQL dinámico**, que permite especificar el tipo de acceso a base de datos en el tiempo de corrida en lugar de en el momento de compilar. Por ejemplo, al usuario se le puede conminar a ingresar un comando SQL que luego se almacene como una cadena en lenguaje huésped. El comando SQL PREPARE (preparar) dice al sistema de gestión de la base de datos que analice sintéticamente (parse) y compile la cadena como un comando SQL y asigne el código ejecutable resultante a una variable SQL nominada. Luego se usa un comando EXECUTE para correr el código. Por ejemplo, en el siguiente segmento la variable en lenguaje huésped `userString` se asigna a un comando SQL de actualización, el código correspondiente se prepara y liga al identificador SQL `userCommand`, y entonces se ejecuta el código.

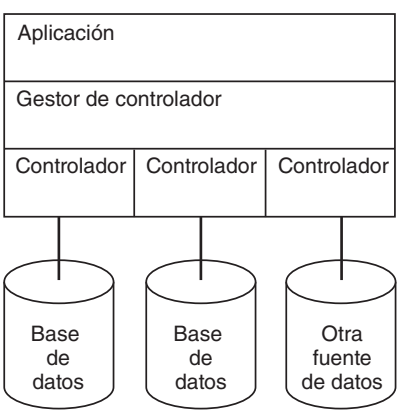
```
char userString[ ]='UPDATE Student SET credits = 36 WHERE stuId= S1050';
EXEC SQL PREPARE userCommand FROM :userString;
EXEC SQL EXECUTE userCommand;
```

6.7.2 API, ODBC y JDBC

Para SQL incrustado, un precompilador suministrado por el DBMS compila el código SQL, y lo sustituye con llamadas de función en el lenguaje huésped. Un enfoque más flexible es que el DBMS proporcione una **Interfaz de Programas de Aplicación, API,** que incluye un conjunto de funciones de biblioteca estándar para procesamiento de bases de datos. Entonces la aplicación puede llamar las funciones de biblioteca, que se escriben en un lenguaje de propósito general. Los programadores usan estas funciones de biblioteca esencialmente en la misma forma como usan la biblioteca del lenguaje en sí. Las funciones permiten a la aplicación realizar operaciones estándar como conectar a la base de datos, ejecutar comandos SQL, presentar tuplas una a la vez, etc. Sin embargo, la aplicación todavía tendría que usar el precompilador para un DBMS particular y ligarse a la librería API para dicho DBMS, de modo que el mismo programa podría no usarse con otro DBMS.

La **conectividad de base de datos abierta (ODBC)** y la **conectividad de base de datos JAVA (JDBC)** proporcionan formas estándar de integrar código SQL y lenguajes de propósito general al proporcionar una interfaz común. Esta estandarización permite que las aplicaciones accedan a múltiples bases de datos usando diferentes DBMS.

FIGURA 6.7
Arquitectura ODBC/JDBC



El estándar proporciona un alto grado de flexibilidad, lo que permite el desarrollo de aplicaciones cliente-servidor que funcionan con una variedad de DBMS, en lugar de estar limitadas a un proveedor API particular. La mayoría de los proveedores ofrecen controladores ODBC o JDBC que se conforman al estándar. Una aplicación que use una de estas interfaces estándar puede usar el mismo código para acceder a diferentes bases de datos sin recompilación. La arquitectura ODBC/JDBC requiere cuatro componentes: la aplicación, gestor de controlador, controlador y fuente de datos (normalmente una base de datos), como se ilustra en la figura 6.7. La aplicación inicia la conexión con la base de datos, envía los datos solicitados como enunciados SQL al DBMS, recupera los resultados, realiza procesamiento y termina la conexión, todo usando el API estándar. Un gestor de controlador carga y descarga controladores a petición de la aplicación, y pasa las llamadas ODBC o JDBC al controlador seleccionado. El controlador de base de datos vincula la aplicación a la fuente de datos, traduce las llamadas ODBC o JDBC a llamadas específicas de DBMS y manipula la traducción de datos necesaria debido a cualquier diferencia entre el lenguaje de datos DBMS y el estándar ODBC/JDBC, y las diferencias de manipulación de error que surgen entre la fuente de datos y el estándar. La fuente de datos es la base de datos (u otra fuente, como una hoja de cálculo) a la que se accede, junto con su entorno, que consiste en sus DBMS y plataforma. Existen distintos niveles de conformidad definidos para controladores ODBC y JDBC, dependiendo del tipo de relación entre la aplicación y la base de datos.

6.7.3 PSM SQL

La mayoría de los sistemas de gestión de bases de datos incluyen una extensión del mismo SQL, llamada **módulos de almacenamiento permanente (PSM)**, para permitir a los usuarios escribir procedimientos almacenados, llamados **rutinas internas**, dentro del espacio de proceso de la base de datos, en lugar de externamente. Estas facilidades se usan para escribir rutinas SQL que se puedan guardar con el esquema de la base de datos y se invoquen cuando se necesite (en contraste, los programas escritos en un lenguaje huésped se conocen como rutinas externas). El PL/SQL de Oracle, al que se puede acceder desde dentro del entorno SQLPlus, es este tipo de facilidad. El estándar SQL/PSM está diseñado para proporcionar facilidades de lenguaje de programación completo, incluidas declaraciones, estructuras de control y enunciados de asignación.

Los módulos SQL/PSM incluyen funciones, procedimientos y relaciones temporales. Para declarar un procedimiento, se escribe:

```
CREATE PROCEDURE nombre_procedimiento (lista_parámetros)
    declaraciones de variables locales
    código de procedimiento
```

Cada parámetro en la lista de parámetros tiene tres ítems: modo, nombre y tipo de datos. El modo puede ser IN, OUT o INOUT, dependiendo de si es un parámetro de entrada, un parámetro de salida o ambos. También se deben proporcionar el nombre y tipo de datos del parámetro. A continuación se tienen declaraciones de variables locales, si hay alguna, y el código para el procedimiento, que puede contener enunciados SQL del tipo visto anteriormente para SQL incrustado, más enunciados de asignación, enunciados de control y manipulación de error.

Una función tiene una declaración similar, del modo siguiente:

```
CREATE FUNCTION nombre_función (lista_parámetros)
    RETURNS tipo de datos SQL
declaraciones de variables locales
código de función (debe incluir un enunciado RETURN)
```

Las funciones sólo aceptan parámetros con modo IN, para evitar efectos colaterales. El único valor regresado debe ser el especificado en el enunciado RETURN.

Las declaraciones tienen esta forma:

```
DECLARE identificador de tipo de datos;
```

como en,

```
DECLARE     status                VARCHAR2;
DECLARE     number_of_courses     NUMBER;
```

El enunciado de asignación SQL, SET, permite que el valor de una constante o una expresión se asigne a una variable. Por ejemplo, podría escribir:

```
SET status = 'Freshman'; //However, Oracle uses := for assignment
SET number_of_courses = credits/3;
```

Las ramificaciones (branches) tienen la forma:

```
IF (condición) THEN enunciados;
   ELSEIF (condición) enunciados;
   . . .
   ELSEIF (condición) enunciados;
   ELSE enunciados;
END IF;
```

por ejemplo,

```
IF (Student.credits <30) THEN
   SET status = 'Freshman';
   ELSEIF (Student.credits <60) THEN
      SET status = 'Sophomore';
   ELSEIF (Student.credits <90) THEN
      SET status = 'Junior';
   ELSE SET status = 'Senior';
END IF;
```

El enunciado CASE se puede usar para selección con base en el valor de una variable o expresión:

```
CASE selector
   WHEN valor_1          THEN enunciados;
   WHEN valor_2          THEN enunciados;
   . . .
END CASE;
```

La repetición se controla mediante estructuras LOOP... ENDLOOP, WHILE... DO... END WHILE, REPEAT... UNTIL... END REPEAT, y FOR... DO... END FOR. Puede usar cursores como se hizo para SQL incrustado. Por ejemplo, podría escribir:

```
...
DECLARE CSCstuCursor CURSOR FOR
   SELECT stuId, lastName, firstName, major, credits
   FROM student
   WHERE major= 'CSC';
OPEN CSCstuCursor;
WHILE (SQLCODE = '00000') DO
   FETCH CSCstuCursor INTO stuNumber,stuLastName,stuFirstName,
   stuMajor,stuCredits;
      //statements to process these values
END WHILE;
CLOSE CSCstuCursor;
```

El lenguaje también proporciona manejadores de excepción predefinidos y también permite al usuario crear excepciones definidas por el usuario.

Una vez creado un procedimiento se puede ejecutar mediante este comando:

```
EXECUTE nombre_procedimiento(lista_parámetros_real);
```

Como con la mayoría de los lenguajes, la lista de parámetros real consiste en los valores o variables a pasar al procedimiento, en oposición a la lista de parámetros formal que aparece en la declaración del procedimiento.

Una función se invoca mediante su nombre, por lo general en un enunciado de asignación. Por ejemplo:

```
SET newVal = MyFunction (val1, val2);
```

6.8 Creación y uso de vistas

Las vistas son herramientas importantes a fin de proporcionar a los usuarios un entorno simple y personalizado, y para ocultar datos. Como se explicó en la sección 6.2, una vista relacional no corresponde exactamente con la vista externa general, sino que es una tabla virtual derivada de una o más tablas base subyacentes. No existe en almacenamiento en el sentido como lo hacen las tablas base, sino que se crean mediante selección de filas y columnas específicas de las tablas base, y posiblemente al realizar operaciones sobre ellas. La vista se produce dinámicamente conforme el usuario trabaja con ella. Para integrar una vista, el ABD decide a cuáles atributos necesita acceso el usuario, determina cuáles tablas base los contienen y construye una o más vistas para mostrar en forma de tabla los valores que el usuario debe ver. Las vistas permiten fácilmente la creación de un modelo externo dinámico. Las razones para proporcionar vistas en lugar de permitir a los usuarios trabajar con las tablas base son las siguientes:

- Las vistas permiten a diferentes usuarios ver los datos en distintas formas, y permiten un modelo externo que difiere del modelo lógico.
- El mecanismo de vista proporciona un dispositivo sencillo de control de autorización, creado fácilmente y reforzado en forma automática por el sistema. Los usuarios de vista no están al tanto de, y no pueden acceder, a ciertos ítems de datos.
- Las vistas pueden liberar a los usuarios de complicadas operaciones DML, especialmente en el caso donde las vistas involucran combinaciones. El usuario escribe un enunciado SELECT simple usando la vista como la tabla nominada y el sistema se encarga de los detalles de las correspondientes operaciones más complicadas sobre las tablas base para soportar la vista.
- Si la base de datos se reestructura en el nivel lógico, la vista se puede utilizar para mantener constante el modelo del usuario. Por ejemplo, si la tabla se divide mediante proyección y la clave primaria aparece en cada una de las nuevas tablas resultantes, la tabla original siempre se puede reconstruir cuando se necesite al definir una vista que es la combinación de las nuevas tablas.

La siguiente es la forma más común del comando usado para crear una vista:

```
CREATE VIEW nombre_vista
    [(nombre_vista [,nombre_vista] . . .)]
    AS  SELECT    nombre_columna [,nombre_columna] . . .
        FROM      nombre_tabla_base [,nombre_tabla_base] . . .
        WHERE     condición;
```

El nombre de la vista se elige usando las mismas reglas que el nombre de tabla y debe ser único dentro de la base de datos. Los nombres de columna en la vista pueden ser diferentes de los correspondientes nombres de columna en las tablas base, pero deben obedecer las mismas reglas de construcción. Si se elige hacerlos iguales es necesario no especificarlos dos veces, de modo que se deja la línea *nombre_col_vista*. En la línea AS SELECT se mencionan los nombres de las columnas de las tablas base subyacentes que se quieren incluir en la vista. El orden de estos nombres debe corresponder exactamente con los *nombre_col_vista*, si los mismos se especifican. Sin embargo, las columnas elegidas de las tablas base pueden reordenarse en cualquier forma deseada en la vista. Como en el usual SELECT... FROM... WHERE, la condición es un predicado lógico que expresa alguna restricción sobre los registros a incluir. Una forma más general de CREATE VIEW usa cualquier subconsulta válida en lugar del SELECT descrito.

- Ejemplo 1. Elección de un subconjunto vertical y horizontal de una tabla

  Suponga que un usuario necesita ver los ID y nombres de todos quienes tienen especialidad history. Puede crear una vista para este usuario del modo siguiente:

```
CREATE VIEW HISTMAJ (last, first, StudentId)
  AS SELECT    lastName, firstName,stuId
     FROM      Student
     WHERE     major = 'History';
```

Aquí se renombraron las columnas de la tabla base. El usuario de esta vista no necesita saber los nombres de columna reales.

- Ejemplo 2. Elección de un subconjunto vertical de una tabla

Si quisiera una tabla de todos los cursos con sus horarios y salones podría crearla del modo siguiente:

```
CREATE VIEW      ClassLoc
AS SELECT        classNumber, schedule, room
FROM             Class;
```

Note que aquí no se necesitó una condición, pues se querían estas partes de todos los registros `Class`. Esta vez se conservan los nombres de las columnas como aparecen en la tabla base.

- Ejemplo 3. Una vista usando dos tablas

Suponga que un usuario necesita una tabla que contenga las ID y nombres de todos los estudiantes en el curso CSC101. La tabla virtual se puede crear al elegir registros en `Enroll` que tengan `classNumber` de CSC101, coincidir el `stuId` de dichos registros con el `stuId` de los registros `Student` y tomar los correspondientes `lastName` y `firstName` de `Student`. Esto se podría expresar como una combinación o una subconsulta.

```
CREATE VIEW ClassList
   AS  SELECT    Student.stuId,lastName,
               firstName
       FROM      Enroll,Student
       WHERE     classNumber = 'CSC101'
               AND Enroll.stuId =
               Student.stuId;
```

- Ejemplo 4. Una vista de una vista

Es posible definir una vista derivada de una vista. Por ejemplo, puede pedir un subconjunto de la tabla (virtual) `ClassLoc` al escribir:

```
CREATE VIEW ClassLoc2
  AS SELECT     classNumber, room
     FROM       ClassLoc;
```

- Ejemplo 5. Una vista usando una función

  En el enunciado SELECT en la línea AS puede incluir funciones internas y opciones GROUP BY. Por ejemplo, si quiere una vista de `Enroll` que proporcione `classNumber` y el número de estudiantes inscritos en cada clase, escriba:

```
CREATE VIEW ClassCount (classNumber, TotCount)
  AS SELECT     classNumber, COUNT(*)
     FROM       Enroll
                GROUP BY classNumber;
```

Note que se debe proporcionar un nombre para la segunda columna de la vista, pues no hay alguno disponible de la tabla base.

- Ejemplo 6. Operaciones sobre vistas

Una vez creada una vista, el usuario puede escribir enunciados SELECT para recuperar datos a través de la vista. El sistema se encarga de mapear los nombres de usuario a los nombres de tabla base y nombres de columna subyacentes, y realiza cualquier función que se requiera para producir el resultado en la forma que el usuario espera. Los usuarios pueden escribir consultas SQL que se refieran a combinaciones, ordenamiento, agrupamiento, funciones internas, etc., de vistas tal como si estuviese operando sobre las tablas base. Dado que la operación SELECT no cambia las tablas base subyacentes, no hay restricción para usarlas con las vistas. El siguiente es un ejemplo de una operación SELECT sobre la vista `ClassLoc`:

```
SELECT     *
FROM       ClassLoc
WHERE      room LIKE 'H%';
```

INSERT, DELETE y UPDATE presentan ciertos problemas con las vistas. Por ejemplo, suponga que se tiene una vista de registros de estudiante como:

```
StudentVwl(lastName, firstName, major, credits)
```

Si tuviese permiso de ingresar registros, cualquier registro creado mediante esta vista en realidad serían registros `Student`, pero no contendrían `stuId`, que es la clave de la tabla `Student`. Dado que `stuId` tendría la restricción NOT NULL deberá rechazar cualquier registro sin este campo. Sin embargo, si tiene la siguiente vista:

```
StudentVw2(stuId,lastName, firstName,credits)
```

no debe tener problema para insertar registros, pues insertaría registros `Student` con un campo `major` nulo, lo que está permitido. Podría lograr esto al escribir:

```
INSERT
INTO    StudentVw2
VALUES  ('S1040' 'Levine', 'Adam', 30);
```

Sin embargo, el sistema en realidad insertaría el registro en la tabla `Student`. Puede usar un disparador INSTEAD OF para asegurar que esto ocurra.

```
CREATE TRIGGER InsertStuVw2
  INSTEAD OF INSERT ON StudentVw2
  FOR EACH ROW
   BEGIN
       INSERT
       INTO Student
       VALUES (:NEW.stuId, :NEW.lastName,
       :NEW.firstName, NEW. Credits);

   END;
```

Ahora considere insertar registros en la vista `ClassCount`, como se describió anteriormente en el ejemplo 5. Esta vista usó la función COUNT en grupos de registros en la tabla `Enroll`. Obviamente, esta tabla tenía la intención de ser un resumen dinámico de la tabla `Enroll`, en lugar de ser un subconjunto fila y columna de dicha tabla. No tendría sentido permitir la inserción de nuevos registros `ClassCount`, pues éstos no corresponden a filas o columnas individuales de una tabla base.

Los problemas identificados para INSERT se aplican con cambios menores también a UPDATE y DELETE. Como regla general, estas tres operaciones se pueden realizar sobre vistas que consisten en filas y columnas reales de tablas base subyacentes, siempre que la clave primaria se incluya en la vista y no se violen otras restricciones. Se pueden usar disparadores INSTEAD OF para asegurar que la base de datos actualiza las tablas subyacentes.

6.9 El catálogo del sistema

El **catálogo del sistema** o **diccionario de datos del sistema** se puede considerar como una base de datos de información acerca de las bases de datos. Contiene, en forma de tabla, un resumen de la estructura de cada base de datos como aparece en un momento dado. Siempre que una tabla base, vista, índice, restricción, módulo almacenado u otro ítem de un esquema de base de datos se crea, altera o elimina, el DBMS automáticamente actualiza sus entradas en el catálogo. El sistema también usa el catálogo para verificar autorizaciones y almacenar información para planes de acceso para aplicaciones. Los usuarios pueden consultar el diccionario de datos usando comandos SQL SELECT. Sin embargo, dado que el diccionario de datos se mantiene mediante el DBMS mismo, los comandos SQL UPDATE, INSERT y DELETE no se pueden usar en él.

El **diccionario de datos Oracle** contiene información acerca de todos los objetos de esquema, pero el acceso a él se proporciona mediante tres vistas diferentes, llamadas USER (usuario), ALL (todos) y DBA (administrador de base de datos). En Oracle, a cada usuario se le proporciona automáticamente acceso a todos los objetos que crea. La **vista USER** proporciona a un usuario información acerca de todos los objetos creados por dicho usuario. Los usuarios pueden tener acceso a objetos creados por otros. La **vista ALL** proporciona información acerca de dichos objetos además de los que creó el usuario. La **vista DBA**, que proporciona información acerca de todos los objetos de base de datos, está disponible al administrador de la base de datos. Cada una de las vistas se invoca mediante el uso del término apropiado como un prefijo para el objeto nombrado en la cláusula FROM en una consulta. Por ejemplo, si un usuario quiere una lista de los nombres de todas las tablas que creó, la consulta es:

```
SELECT TABLE_NAME
FROM USER_TABLES;
```

Las consultas se pueden escribir usando los prefijos adecuados (USER_, ALL_, DBA_) en la cláusula FROM, seguido por una de las categorías CATALOG, CONSTRAINTS, CONS_COLUMNS (columnas que tienen restricciones), DICTIONARY, IND_COLUMNS (columnas que tienen índices, INDEXES, OBJECTS, TAB_COLUMNS (columnas de tabla), TRIGGERS, ROLES, PROFILES, SEQUENCES, SOURCE (código fuente para un módulo), SYS_PRIVS, USERS, TABLES, TABLESPACES, VIEWS y otros objetos en el esquema. Para cada una de estas categorías, cada una de las tres vistas (USER, ALL, DBA) tiene varias columnas. Por lo general, puede suponer que cada categoría tiene una columna para el nombre del objeto, que puede usar con el fin de escribir una consulta como:

```
SELECT VIEW_NAME
FROM USER_VIEWS;
```

Para determinar cuáles son todas las columnas disponibles, puede usar el comodín (*) en la cláusula SELECT. Por ejemplo,

```
SELECT*
FROM USER_TAB_COLUMNS;
```

desplegará toda la información registrada acerca de las columnas en las tablas que usted creó. Luego puede usar los nombres de columna de las vistas (por ejemplo, COLUMN_NAME, DATA_TYPE) en una consulta más específica, como

```
SELECT COLUMN_NAME, DATA_TYPE
FROM USER_TAB_COLUMNS
WHERE TABLE_NAME = 'STUDENT';
```

Otra forma de aprender acerca de los objetos es usar el comando DESCRIBE. Una vez que conozca el nombre de un objeto (por ejemplo, una tabla, restricción, columna), que puede obtener mediante uno de los métodos recién ilustrados, puede solicitar una descripción de él. Por ejemplo, puede escribir

```
DESCRIBE STUDENT;
```

para ver lo que se conoce acerca de la tabla `Student`, o

```
DESCRIBE HISTMAJ;
```

para ver lo que se conoce acerca de la vista `HISTMAJ` que se creó en la sección 6.8. Si quiere aprender qué información está disponible acerca de las restricciones en la vista USER, escribiría:

```
DESCRIBE USER_CONSTRAINTS;
```

Entonces puede escribir una consulta con el uso de los nombres de las columnas que se desplegaron, como:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM USER_CONSTRAINTS;
```

Para información acerca de los disparadores, el comando

```
SELECT TRIGGER_NAME, TRIGGERING_EVENT, TRIGGER_TYPE
FROM USER_TRIGGERS;
```

proporciona información útil acerca de ellos.

La **base de datos universal DB2** de IBM tiene un catálogo de sistema que también se mantiene en la forma de tablas en un esquema llamado SYSIBM, usualmente con acceso restringido. Dos vistas de las tablas, SYSCAT y SYSSTAT, están disponibles para los usuarios. El **esquema SYSCAT** tiene muchas tablas, todas ellas de sólo lectura. Algunas de las más importantes son las siguientes:

```
TABLES (TABSCHEMA, TABNAME, DEFINER, TYPE, STATUS, COLCOUNT, KEYCOLUMNS,
CHECKCOUNT, ...)
COLUMNS (TABSCHEMA, TABNAME, COLNAME, COLNO, TYPENAME, LENGTH, DEFAULT,
NULLS, ...)
INDEXES (INDSCHEMA, INDNAME, DEFINER, TABSCHEMA, TABNAME, COLNAMES, UNIQUERULE,
COLCOUNT, ...)
TRIGGERS (TRIGSCHEMA, TRIGNAME, DEFINER, TABSCHEMA, TABNAME, TRIGTIME,
TRIGEVENT, ...)
VIEWS (VIEWSCHEMA, VIEWNAME, DEFINER, TEXT ...)
```

Puede escribir consultas para estas tablas usando SQL, como en

```
SELECT TABSCHEMA, TABNAME
FROM TABLES
WHERE DEFINER = 'JONES';
```

que da los nombres de todas las tablas creadas por Jones.

La consulta

```
SELECT *
FROM COLUMNS
WHERE TABNAME = 'STUDENT'
GROUP BY COLNAME;
```

da toda la información disponible acerca de las columnas de la tabla `Student`.

6.10 Resumen del capítulo

Oracle, DB2 de IBM, MySQL, SQL Server y otros sistemas de gestión de bases de datos relacionales usan **SQL**, un DDL y DML relacional estándar. En el nivel lógico, cada relación se representa mediante una **tabla base**. El nivel externo consiste en **vistas**, que se crean a partir de subconjuntos, combinaciones u otras operaciones en las tablas base. Una tabla base puede tener **índices**, uno de los cuales puede ser un índice agrupado, definido en ella. La definición de base de datos dinámica permite que la estructura cambie en cualquier momento.

Los comandos **DDL SQL CREATE TABLE** y **CREATE INDEX** se usan para crear las tablas de datos y sus índices. Están disponibles muchos **tipos de datos** internos y los usuarios también pueden definir nuevos tipos. En el nivel de columna o tabla se pueden especificar **restricciones**. El comando **ALTER TABLE** permite cambios a tablas existentes, como agregar una nueva columna, eliminar una columna, cambiar tipos de datos o cambiar restricciones. El comando **RENAME TABLE** permite al usuario cambiar un nombre de tabla. **DROP TABLE** y **DROP INDEX** remueven tablas e índices, junto con todos los datos en ellos, de la base de datos.

Los comandos **DML** son **SELECT**, **UPDATE**, **INSERT** y **DELETE**. El comando SELECT tiene varias formas y realiza el equivalente de las operaciones SELECT, PROJECT y JOIN del álgebra relacional. Las opciones incluyen **GROUP BY**, **ORDER BY**, **GROUP BY**... **HAVING**, **LIKE** y funciones internas **COUNT**, **SUM**, **AVG**, **MAX** y **MIN**. El enunciado SELECT puede operar sobre combinaciones de tablas y puede manejar **subconsultas**, incluidas subconsultas correlacionadas. También son posibles expresiones y operaciones de conjuntos. El comando UPDATE se puede usar para actualizar uno o más campos en uno o más registros. El comando INSERT puede insertar uno o más registros, posiblemente con valores nulos para algunos campos. El operador DELETE borra registros, mientras deja intacta la estructura de la tabla.

Una base de datos activa es aquella donde el DBMS monitorea activamente los cambios para garantizar que sólo se crean instancias legales de la base de datos. Para crear una base de datos activa se puede usar una combinación de restricciones y **disparadores** (triggers).

El enunciado **COMMIT** hace permanentes todos los cambios que se han hecho mediante la transacción actual. El enunciado **ROLLBACK** deshace todos los cambios que se realizaron mediante la transacción actual. La transacción actual comienza inmediatamente después del último COMMIT o ROLLBACK, o, si ninguno de éstos ocurrió, entonces al comienzo de la actual sesión de usuario.

SQL se usa con frecuencia en un entorno de programación en lugar de interactivamente. Se puede **incrustar** (embedded) en un lenguaje de programación huésped y compilar por separado mediante un **precompilador**. Se puede usar con un API estándar a través de **ODBC** o **JDBC**. También se puede usar como un lenguaje completo usando sus propios **PSM SQL**. El PL/SQL de Oracle es un ejemplo de un entorno de programación completo para creación de PSM SQL.

El comando **CREATE VIEW** se usa para definir una tabla virtual, mediante la selección de campos de tablas base existentes o vistas definidas con antelación. La operación SELECT se puede usar en vistas, pero otros comandos DML están restringidos a ciertos tipos de vistas. Un **disparador INSTEAD OF** es útil para sustituir comandos DML del usuario escritos en una vista con comandos correspondientes en la(s) tabla(s) base usada(s) para la vista. Una definición de vista se puede destruir mediante un comando DROP VIEW.

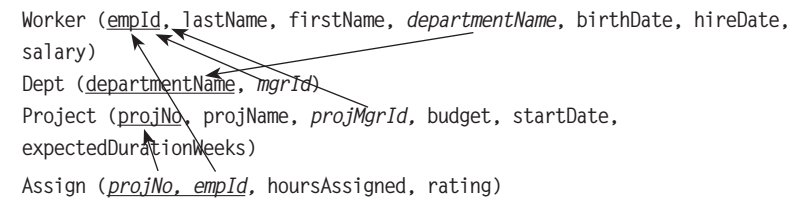
El **catálogo del sistema** o **diccionario de datos del sistema** es una base de datos que contiene información acerca de la base de datos del usuario. Sigue la pista de las tablas, columnas, índices y vistas que existen, así como información de autorización y otros datos. El sistema automáticamente actualiza el catálogo cuando se realizan cambios estructurales y otras modificaciones.

Ejercicios

6.1 Escriba los comandos necesarios para crear índices para las tablas `Student`, `Faculty`, `Class` y `Enroll` de este capítulo.

6.2 Para cada uno de los ejemplos de combinación (ejemplos 7-11) de la sección 6.4.2, sustituya la combinación mediante una subconsulta, si es posible. Si no es posible, explique por qué no.

Instrucciones para los ejercicios 6.3-6.25: Para el esquema que sigue, escriba los comandos indicados en SQL. La figura 6.8 muestra el DDL para crear estas tablas. Muestra que `departmentName` es una clave externa en la tabla `Worker`, que `mgrId` es una clave externa en la tabla `Dept`, que `projMgrId` es una clave externa en la tabla `Project`, y que `projNo` y `empId` son claves externas en la tabla `Assign`. Suponga que cada departamento tiene un supervisor (manager) y que cada proyecto tiene un supervisor, pero éstos no necesariamente están relacionados. (Nota: Se recomienda que realice el Ejercicio de laboratorio 1 en conjunto con estos ejercicios. Si Oracle no está disponible, puede usar otro DBMS relacional, incluido el freeware MySQL, que puede descargar. Dependiendo del producto es posible que necesite hacer algunos cambios al DDL. Si no planea realizar el Ejercicio de laboratorio 1, simplemente puede escribir los comandos.)



6.3 Obtenga los nombres de todos los trabajadores en el departamento accounting (contabilidad).

6.4 Obtenga una lista alfabética de nombres de todos los trabajadores asignados al proyecto 1001.

6.5 Obtenga el nombre del empleado (employee) en el departamento research (investigación) que tenga el salario más bajo (lowest).

6.6 Obtenga detalles del proyecto con el presupuesto (budget) más alto (highest).

6.7 Obtenga los nombres y departamentos de todos los trabajadores en el proyecto 1019.

6.8 Obtenga una lista alfabética de nombres y calificaciones (ratings) correspondientes de todos los trabajadores en cualquier proyecto que esté supervisado (managed) por Michael Burns.

FIGURA 6.8

DDL y enunciados Insert para el ejemplo Worker-Project-Assign

```
CREATE TABLE Dept (
departmentName VARCHAR2 (15),
mgrId Number(6),
CONSTRAINT Dept_departmentName_pk PRIMARY KEY (departmentName));
CREATE TABLE Worker (
empId NUMBER (6),
lastName VARCHAR2 (20) NOT NULL,
firstName VARCHAR2 (15) NOT NULL,
departmentName  VARCHAR2 (15),
birthDate DATE,
hireDate DATE,
salary NUMBER (8, 2),
CONSTRAINT Worker_empid_pk PRIMARY KEY (empid),
CONSTRAINT Worker_departmentNumber_fk FOREIGN KEY (departmentNumber) REFERENCES
Dept (departmentName));

ALTER TABLE Dept Add CONSTRAINT Dept_mgrId_fk FOREIGN KEY (mgrId) REFERENCES Worker (empId)) ON UPDATE

CREATE TABLE Project (
projNo NUMBER (6),
projName VARCHAR2 (20),
projMgrId VARCHAR2 (20),
budget NUMBER (8, 2),
startDate DATE,
expectedDurationWeeks NUMBER (4),
CONSTRAINT Project_projNo_pk PRIMARY KEY (projNo),
CONSTRAINT Project_projMgrId_fk FOREIGN KEY (projMgrid) REFERENCES WORKER (empId));

CREATE TABLE Assign (
projNo NUMBER (6),
empId NUMBER (6),
hoursAssigned NUMBER (3),
rating NUMBER (1),
CONSTRAINT Assign_projNo_empId_pk PRIMARY KEY (projNo, empId),
CONSTRAINT Assign_projNo_fk FOREIGN KEY (projNo) REFERENCES Project (projNo) ON UPDATE
CONSTRAINT Assign_empId_fk FOREIGN KEY (empId) REFERENCES Worker (empId) ON
ON DELETE CASCADE);

INSERT INTO Dept VALUES ('Accounting',);
INSERT INTO Dept VALUES ('Research',);

INSERT INTO Worker VALUES(101, 'Smith', 'Tom', 'Accounting', '01-Feb-1960', '06-Jun-1983', 50000);
INSERT INTO Worker VALUES(103, 'Jones', 'Mary' , 'Accounting', '15-Jun-1965', '20-Sep-1985', 48000);
INSERT INTO Worker VALUES(105, 'Burns', 'Jane', 'Accounting', '21-Sep-1970' , '12-Jun-1990', 39000);
INSERT INTO Worker VALUES(110, 'Burns', 'Michael', 'Research', '05-Apr-1967', '10-Sep-1990', 70000);
INSERT INTO Worker VALUES(115, 'Chin', 'Amanda', 'Research', '22-Sep-1965', '19-Jun-1985', 60000);

UPDATE Dept SET mgrId = 101 WHERE departmentName = 'Accounting';
UPDATE Dept SET mgrId = 101 WHERE departmentName = 'Research';

INSERT INTO Project VALUES (1001, 'Jupiter', 101, 300000, '01-Feb-2004', 50);
INSERT INTO Project VALUES (1005, 'Saturn', 101, 400000, '01-Jun-2004', 35);
INSERT INTO Project VALUES (1019, 'Mercury', 110, 350000, '15-Feb-2004', 40);
INSERT INTO Project VALUES (1025, 'Neptune', 110, 600000, '01-Feb-2005', 45);
INSERT INTO Project VALUES (1030, 'Pluto', 110, 380000, '15-Sept-2004', 50);
```

(*continúa*)

```
INSERT INTO Assign(projNo, empId, hoursAssigned) VALUES (1001, 101, 30);
INSERT INTO Assign VALUES (1001, 103, 20, 5);
INSERT INTO Assign(projNo, empId, hoursAssigned)VALUES (1005, 103, 20);
INSERT INTO Assign(projNo, empId, hoursAssigned)VALUES (1001, 105, 30);
INSERT INTO Assign VALUES (1001, 115, 20, 4);
INSERT INTO Assign VALUES (1019, 110, 20, 5);
INSERT INTO Assign VALUES (1019, 115, 10, 4);
INSERT INTO Assign(projNo, empId, hoursAssigned)VALUES (1025, 110, 10);
INSERT INTO Assign(projNo, empId, hoursAssigned)VALUES (1030, 110, 10);
```

FIGURA 6.8
Continuación

6.9 Cree una vista que tenga número de proyecto y nombre de cada proyecto, junto con las ID y nombres de todos los trabajadores asignados a él.

6.10 Con la vista creada en el ejercicio 6.9, encuentre el número de proyecto y nombre de proyecto de todos los proyectos a los que está asignado el empleado 110.

6.11 Agregue un nuevo trabajador llamado Jack Smith con ID de 1999 al departamento research.

6.12 Cambie las horas que tiene asignadas el empleado 110 al proyecto 1019, de 20 a 10.

6.13 Para todos los proyectos que comiencen (starting) después del 1 de mayo de 2004, encuentre el número de proyecto y las ID y nombres de todos los trabajadores asignados a ellos.

6.14 Para cada proyecto, haga una lista del número de proyecto y cuántos trabajadores se asignan a él.

6.15 Encuentre los nombres de empleado y nombres de supervisor de departamento de todos los trabajadores que no estén asignados a proyecto alguno.

6.16 Encuentre los detalles de cualquier proyecto con la palabra "urn" en cualquier parte en su nombre.

6.17 Obtenga una lista de números de proyecto y nombres y fechas de inicio de todos los proyectos que tienen la misma fecha de inicio.

6.18 Agregue un campo llamado `status` a la tabla `Project`. Los valores muestra para este campo son `active` (activo), `completed` (completado), `planned` (planificado), `cancelled` (cancelado). Luego escriba el comando para deshacer este cambio.

6.19 Obtenga la ID de empleado y número de proyecto de todos los empleados que no tengan calificaciones en dicho proyecto.

6.20 Si supone que `salary` ahora contiene salario anual, encuentre la ID, nombre y salario mensual de cada trabajador.

6.21 Agregue un campo llamado `numEmployeesAssigned` a la tabla `Project`. Use el comando UPDATE a fin de insertar valores en el campo para que corresponda con la información actual en la tabla `Assign`. Luego escriba un disparador que actualice el campo correctamente siempre que se realice, elimine o actualice una asignación. Escriba el comando para hacer permanentes estos cambios.

6.22 a. Escriba una consulta de diccionario de datos Oracle a fin de mostrar los nombres de todas las columnas en una tabla llamada `Customers` (clientes).

b. Escriba una consulta correspondiente para las tablas DB2 UDB SYSCAT para este ejemplo.

6.23 a. Escriba una consulta de diccionario de datos Oracle para encontrar toda la información acerca de todas las columnas llamadas PROJNO.

b. Escriba una consulta correspondiente para las tablas DB2 UDB SYSCAT para este ejemplo.

6.24 a. Escriba una consulta de diccionario de datos Oracle para obtener una lista de nombres de las personas que hayan creado tablas, junto con el número de tablas que creó cada una. Suponga que tiene privilegios ABD.

b. Escriba una consulta correspondiente para las tablas DB2 UDB SYSCAT para este ejemplo.

6.25 a. Escriba una consulta de diccionario de datos Oracle para encontrar los nombres de las tablas que tengan más de dos índices.

b. Escriba una consulta correspondiente para las tablas DB2 UDB SYSCAT para este ejemplo.

Ejercicios de laboratorio

Ejercicio de laboratorio 6.1. Exploración de la base de datos Oracle para el ejemplo Worker-Dept-Project-Assign

Un script para crear una base de datos Oracle para el ejemplo usado en los ejercicios 6.3-6.25 aparece en la figura 6.8 y en el website que acompaña a este libro. El script se escribió usando Notepad. Encuentre el script en el website, cópielo en su propio directorio y ábralo con Notepad. Abra la facilidad SQLPlus de Oracle. Cambiará de ida y vuelta entre Notepad y SQLPlus, porque el editor en SQLPlus es difícil de usar.

a. En Notepad, resalte y copie el comando para crear la primera tabla, luego cambie a SQLPlus y pegue dicho comando en la ventana SQLPlus y ejecute el comando. Debe ver el mensaje "Table created" (tabla creada). Si en vez de ello obtiene un mensaje de error, regrese al archivo Notepad y corrija el error.

b. Continúe para crear las tablas restantes una a la vez en la misma forma.

c. Corra los comandos INSERT para poblar las tablas. Explique por qué se usaron los dos enunciados UPDATE.

d. Con esta implementación, ejecute los enunciados SQL Oracle para los ejercicios 6.3-6.25.

Ejercicio de laboratorio 6.2. Creación y uso de una base de datos simple en Oracle

a. Escriba los comandos DDL para crear las tablas `Student`, `Faculty`, `Class` y `Enroll` para la base de datos University que se muestra en la figura 6.2.

b. Con el comando INSERT agregue los registros conforme aparezcan en la figura 6.3 a su nueva base de datos Oracle.

c. Escriba consultas SQL para las siguientes preguntas y ejecútelas.

i. Encuentre los nombres de todas las especialidades history.

ii. Encuentre el número de clase (class number), horario (schedule) y salón (room) para todas las clases que imparte Smith del departamento historia (history).

iii. Encuentre los nombres de todos los estudiantes que tienen menos que el número promedio de créditos.

iv. Encuentre los nombres de todos los profesores que tiene Ann Chin, junto con todas sus clases y calificaciones de mitad de semestre de cada una.

v. Para cada estudiante, encuentre el número de clases en las que está inscrito.

PROYECTO DE MUESTRA: CREACIÓN Y MANIPULACIÓN DE UNA BASE DE DATOS RELACIONAL PARA LA GALERÍA DE ARTE

En la sección del proyecto de muestra al final del capítulo 5 creó un modelo relacional normalizado para la base de datos de la Galería de Arte. Al renombrar ligeramente las tablas se concluyó que el siguiente modelo se debe implementar:

(1) Artist (artistId, firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

(2) Zips (zip, city, state)

(3) PotentialCustomer (potentialCustomerId, firstname, lastName, areaCode, telephoneNumber, street, *zip*, dateFilledIn, *preferredArtistId*, preferredMedium, preferredStyle, preferredType)

(4) Artwork (artworkId, *artistId*, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

(5) ShownIn (*artworkId,showTitle*)

(6) Collector (socialSecurityNumber, firstName, lastName, street, *zip*, interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear, salesYearToDate, *collectionArtistId*, collectionMedium, collectionStyle, collectionType, SalesLastYear, SalesYearToDate)

(7) Show (showTitle, *showFeaturedArtistId*, showClosingDate, showTheme, showOpeningDate)

(8) Buyer (buyerId, firstName, lastName, street, *zip*, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

(9) Sale (InvoiceNumber, *artworkId*, amountRemittedToOwner, saleDate, salePrice, saleTax, *buyerId*, *salespersonSocialSecurityNumber*)

(10) Salesperson (socialSecurityNumber, firstName, lastName, street, *zip*)

- Paso 6.1. Actualice el diccionario de datos y lista de suposiciones si se necesita. Para cada tabla escriba el nombre de la tabla y los nombres, tipos de datos y tamaños de todos los ítems de datos, identifique cualquier restricción y utilice las convenciones del DBMS que usará para la implementación.

A la lista de suposiciones no se hicieron cambios. No se necesitan más cambios al diccionario de datos. Para una base de datos Oracle, las tablas tendrán las siguientes estructuras:

TABLE Zips

Item	Datatype	Size	Constraints	Comments
Zip	CHAR	5	PRIMARY KEY	
city	VARCHAR2	15	NOT NULL	
state	CHAR	2	NOT NULL	

TABLE Artist

Item	Datatype	Size	Constraints	Comments
artistId	NUMBER	6	PRIMARY KEY	
firstName	VARCHAR2	15	NOT NULL; (firstName, lastName) UNIQUE	
lastName	VARCHAR2	20	NOT NULL; (firstName, lastName) UNIQUE	
interviewDate	DATE			
interviewerName	VARCHAR2	35		
areaCode	CHAR	3		
telephoneNumber	CHAR	7		
street	VARCHAR2	50		
zip	CHAR	5	FOREIGN KEY REF Zips	
salesLastYear	NUMBER	8,2		
salesYearToDate	NUMBER	8,2		
socialSecurityNumber	CHAR	9	UNIQUE	
usualMedium	VARCHAR	15		
usualStyle	VARCHAR	15		
usualType	VARCHAR	20		

TABLE Collector

Item	Datatype	Size	Constraints	Comments
socialSecurityNumber	CHAR	9	PRIMARY KEY	
firstName	VARCHAR2	15	NOT NULL	
lastName	VARCHAR2	20	NOT NULL	
interviewDate	DATE			
interviewerName	VARCHAR2	35		
areaCode	CHAR	3		
telephoneNumber	CHAR	7		
street	VARCHAR2	50		
zip	CHAR	5	FOREIGN KEY Ref Zips	
salesLastYear	NUMBER	8,2		
salesYearToDate	NUMBER	8,2		
collectionArtistId	NUMBER	6	FOREIGN KEY REF Artist	
collectionMedium	VARCHAR	15		
collectionStyle	VARCHAR	15		
collectionType	VARCHAR	20		

TABLE PotentialCustomer

Item	Datatype	Size	Constraints	Comments
potentialCustomerId	NUMBER	6	PRIMARY KEY	
firstname	VARCHAR2	15	NOT NULL	
lastName	VARCHAR2	20	NOT NULL	
areaCode	CHAR	3		
telephoneNumber	CHAR	7		
street	VARCHAR2	50		
zip	CHAR	5	FOREIGN KEY REF Zips	
dateFilledIn	DATE			
preferredArtistId	NUMBER	6	FOREIGN KEY REF Artist	
preferredMedium	VARCHAR2	15		
preferredStyle	VARCHAR2	15		
preferredType	VARCHAR2	20		

TABLE Artwork

Item	Datatype	Size	Constraints	Comments
artworkId	NUMBER	6	PRIMARY KEY	
artistId	NUMBER	6	FOREIGN KEY REF Artist; NOT NULL; (artistId, workTitle) UNIQUE	
workTitle	VARCHAR2	50	NOT NULL; (artistId, workTitle) UNIQUE	
askingPrice	NUMBER	8,2		
dateListed	DATE			
dateReturned	DATE			
dateShown	DATE			
status	VARCHAR2	15		
workMedium	VARCHAR2	15		
workSize	VARCHAR2	15		
workStyle	VARCHAR2	15		
workType	VARCHAR2	20		
workYearCompleted	CHAR	4		
collectorSocialSecurityNumber	CHAR	9	FOREIGN KEY REF Collector	

TABLE Show

Item	Datatype	Size	Constraints	Comments
showTitle	VARCHAR2	50	PRIMARY KEY	
showFeaturedArtistId	NUMBER	6	FOREIGN KEY REF Arist	
showClosingDate	DATE			
showTheme	VARCHAR2	50		
showOpeningDate	DATE			

TABLE ShownIn

Item	Datatype	Size	Constraints	Comments
artworkId	NUMBER	6	PRIMARY KEY(artworkId, showTitle); FOREIGN KEY REF Artwork	
showTitle	VARCHAR2	50	PRIMARY KEY(artworkId, showTitle); FOREIGN KEY REF Show	

TABLE Buyer

Item	Datatype	Size	Constraints	Comments
buyerId	NUMBER	6	PRIMARY KEY	
firstName	VARCHAR2	15	NOT NULL	
lastName	VARCHAR2	20	NOT NULL	
street	VARCHAR2	50		
zip	CHAR	5	FOREIGN KEY REF Zips	
areaCode	CHAR	3		
telephoneNumber	CHAR	7		
purchasesLastYear	NUMBER	8,2		
purchasesYearToDate	NUMBER	8,2		

TABLE Salesperson				
Item	**Datatype**	**Size**	**Constraints**	**Comments**
socialSecurityNumber	CHAR	9	PRIMARY KEY	
firstName	VARCHAR2	15	NOT NULL; (firstName,lastName) UNIQUE	
lastName	VARCHAR2	20	NOT NULL; (firstName,lastName) UNIQUE	
street	VARCHAR2	50		
zip	CHAR	5	FOREIGN KEY REF Zips	

TABLE Sale				
Item	**Datatype**	**Size**	**Constraints**	**Comments**
invoiceNumber	NUMBER	6	PRIMARY KEY	
artworkId	NUMBER	6	NOT NULL; UNIQUE; FOREIGN KEY REF Artwork	
amountRemittedToOwner	NUMBER	8,2	DEFAULT 0.00	
saleDate	DATE			
salePrice	NUMBER	8,2		
saleTax	NUMBER	6,2		
buyerId	NUMBER	6	NOT NULL; FOREIGN KEY REF Buyer	
salespersonSocialSecurityNumber	CHAR	9		

- Paso 6.2. Escriba y ejecute enunciados SQL con el fin de crear todas las tablas necesarias para implementar el diseño.

Puesto que se quiere especificar claves externas conforme se creen las tablas, debe tener cuidado con el orden en el que se crean, porque la "tabla origen" tiene que existir antes de crear la tabla que contiene la clave externa. Por tanto, se usará el siguiente orden: Zips, Artist, Collector, Potential Customer, Artwork, Show, ShowIn, Buyer, Salesperson, Sale. Los enunciados DDL para crear las tablas se muestran en la figura 6.9. Se usa la sintaxis Oracle, pero los enunciados DDL deben funcionar, con modificaciones menores, para cualquier DBMS relacional.

- Paso 6.3. Cree índices para claves externas y cualquier otra columna que se usará con más frecuencia para las consultas.

Los enunciados DDL para crear los índices se muestran en la figura 6.10.

- Paso 6.4. Inserte aproximadamente cinco registros en cada tabla, que preserven todas las restricciones. Ponga suficientes datos para demostrar cómo funcionará la base de datos.

La figura 6.11 muestra el enunciado INSERT. Puesto que se quiere usar los valores generados por el sistema de Oracle para claves subrogadas, se crean secuencias para cada uno de `artistId`, `potentialCustomerId`, `artworkId` y `buyerId`, usando este comando:

```
CREATE SEQUENCE  nombre_secuencia
[START WITH valor_inicial]
[INCREMENT BY paso] . . .;
```

Se elige comenzar con uno e incrementar por uno, los defectos. Para generar cada nuevo valor, use el comando <*nombre_secuencia*>.NEXTVAL, como se muestra en los comandos INSERT. Se supone que `invoiceNumber` es un número preimpreso en la forma de factura, no generada por el sistema, así que no se necesita una secuencia para dicho número.

```
CREATE TABLE Zips (
        zip CHAR (5),
        city VARCHAR2 (15) NOT NULL,
        state CHAR (2) NOT NULL,
        CONSTRAINT Zips_zip_pk PRIMARY KEY (zip));

CREATE TABLE Artist (
        ArtistId NUMBER (6),
        firstName VARCHAR2 (15) NOT NULL,
        lastName VARCHAR2 (20) NOT NULL,
        interviewDate DATE,
        interviewerName VARCHAR2 (35),
        areaCode  CHAR (3),
        telephoneNumber  CHAR (7),
        street VARCHAR2 (50),
        zip  CHAR (5),
        salesLastYear NUMBER (8, 2),
        salesYearToDate   NUMBER (8, 2),
        socialSecurityNumber CHAR (9),
        usualMedium VARCHAR (15),
        usualStyle VARCHAR (15),
        usualType VARCHAR (20),
        CONSTRAINT Artist_ArtistId_pk PRIMARY KEY (ArtistId),
        CONSTRAINT Artist_SSN_uk UNIQUE (socialSecurityNumber),
        CONSTRAINT Artist_fName_lName_uk UNIQUE (firstName,   lastName),
        CONSTRAINT Artist_zip_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE Collector (
        socialSecurityNumber CHAR (9),
        firstName VARCHAR2 (15) NOT NULL,
        lastName VARCHAR2 (20) NOT NULL,
        interviewDate DATE,
        interviewerName VARCHAR2 (35),
        areaCode CHAR (3),
        telephoneNumber   CHAR (7),
        street VARCHAR2 (50),
        zip CHAR (5),
        salesLastYear NUMBER (8, 2),
        salesYearToDate   NUMBER (8, 2),
        collectionArtistId NUMBER (6),
        collectionMedium VARCHAR (15),
        collectionStyle VARCHAR (15),
        collectionType VARCHAR (20),
        CONSTRAINT Collector_SSN_pk PRIMARY KEY
       (socialSecurityNumber).
        CONSTRAINT Collector_collArtistid_fk FOREIGN KEY (collectionArtistId)

REFERENCES Artist (artistId)
        CONSTRAINT Collector_zip_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE PotentialCustomer (
        potentialCustomerId NUMBER (6),
        firstname VARCHAR2 (15) NOT NULL,
        lastName VARCHAR2 (20) NOT NULL,
```

(*continúa*)

FIGURA 6.9

Enunciados DDL Oracle para las tablas de la Galería de Arte

FIGURA 6.9
Continuación

```
        areaCode CHAR (3),
        telephoneNumber CHAR (7),
        street VARCHAR2 (50),
        zip CHAR (5),
        dateFilledIn DATE,
        preferredArtistId NUMBER (6),
        preferredMedium VARCHAR2 (15),
        preferredStyle VARCHAR2 (15),
        preferredType VARCHAR2 (20),
        CONSTRAINT PotentialCustomer_potCusId_pk PRIMARY KEY
        (potentialCustomerId),
        CONSTRAINT PotentialCustomer_zip_fk FOREIGN KEY (zip)
        REFERENCES Zips (zip)
        CONSTRAINT Potential Customer_prefAId_fk FOREIGN KEY
        (preferredArtistId) REFERENCES Artist (artist_Id)) ;

CREATE TABLE Artwork (
        artworkId NUMBER (6),
        artistId NUMBER (6) NOT NULL,
        workTitle VARCHAR2 (50) NOT NULL,
        askingPrice NUMBER (8, 2),
        dateListed DATE,
        dateReturned DATE,
        dateShown DATE,
        status VARCHAR2 (15),
        workMedium VARCHAR2 (15),
        workSize VARCHAR2 (15),
        workStyle VARCHAR2 (15),
        workType VARCHAR2 (20),
        workYearCompeted CHAR (4),
        collectorSocialSecurityNumber CHAR (9),
        CONSTRAINT Artwork_artworkId_pk PRIMARY KEY (artworkId).
        CONSTRAINT Artwork_artId_wTitle_uk UNIQUE (artistId, workTitle),
        CONSTRAINT Artwork_artId_fk FOREIGN KEY (artistId) REFERENCES Artist (artistId)
        CONSTRAINT Artwork_collSSN_fk FOREIGN KEY
(collectorSocialSecurityNumber)   REFERENCES Collector (socialSecurityNumber);

CREATE TABLE Show (
        showTitle VARCHAR2 (50),
        showFeaturedArtistId NUMBER (6),
        showClosingDate DATE,
        showTheme VARCHAR2 (50),
        showOpeningDate DATE,
        CONSTRAINT Show_showTitle_pk PRIMARY KEY (showTitle),
        CONSTRAINT Show_showFeaturedArtId_fk FOREIGN KEY (showFeaturedArtistId)

REFERENCES Artist (artistId);
```

(*continúa*)

FIGURA 6.9
Continuación

```
CREATE TABLE ShownIn (
        artworkId NUMBER (6),
        showTitle VARCHAR2 (50),
        CONSTRAINT ShownIn_artId_showTitle_pk PRIMARY KEY (artworkId, showTitle),
        CONSTRAINT ShownIn_artId_fk FOREIGN KEY (artworkId) REFERENCES Artwork
        (artworkId)
        CONSTRAINT ShownIn_showTitle_fk FOREIGN KEY (showTitle) REFERENCES Show
        (showTitle));

CREATE TABLE Buyer (
        buyerId NUMBER (6),
        firstName VARCHAR2 (15) NOT NULL,
        lastName VARCHAR2 (20) NOT NULL,
        street VARCHAR2 (50),
        zip CHAR (5),
        areaCode CHAR (3),
        telephoneNumber CHAR (7),
        purchasesLastYear NUMBER (8, 2),
        purchasesYearToDate NUMBER (8, 2),
        CONSTRAINT Buyer_buyerId_pk PRIMARY KEY (buyerId),
        CONSTRAINT Buyer_zip_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE Salesperson (
        socialSecurityNumber CHAR (9),
        firstName VARCHAR2 (15) NOT NULL,
        lastName VARCHAR2 (20) NOT NULL,
        street VARCHAR2 (50),
        zip CHAR (5),
        CONSTRAINT Salesperson_SSN_pk PRIMARY KEY
        socialSecurityNumber),
        CONSTRAINT Salesperson_fName_lName_uk UNIQUE (firstName, lastName),
        CONSTRAINT Salesperson_zip_fk FOREIGN KEY (zip) REFERENCES Zips (zip));

CREATE TABLE Sale (
        invoiceNumber NUMBER (6),
        artworkId NUMBER (6) NOT NULL,
        amountRemittedToOwner NUMBER (8, 2) DEFAULT 0.00,
        saleDate DATE,
        salePrice NUMBER (8, 2),
        saleTax NUMBER (6, 2),
        buyerId NUMBER (6) NOT NULL,
        salespersonSocialSecurityNumber CHAR (9),
        CONSTRAINT Sale_invoiceNumber_pk PRIMARY KEY (invoiceNumber),
        CONSTRAINT Sale_artworkId_uk UNIQUE (artworkId),
        CONSTRAINT Sale_artworkId_fk FOREIGN KEY (artworkId) REFERENCES Artwork (artworkId),
        CONSTRAINT Sale_buyerId_fk FOREIGN KEY (buyerId) REFERENCES Buyer (buyerId));
```

FIGURA 6.10

Enunciados DDL Oracle para los índices de la Galería de Arte

```
CREATE UNIQUE INDEX Artist_lastName_firstName ON Artist(lastName, firstName);
CREATE INDEX Artist_zip ON Artist(zip);

CREATE INDEX Collector_collectionArtistId On Collector(collectionArtistId);
CREATE INDEX Collector_zip ON Collector(zip);
CREATE INDEX Collector_lastName_firstName ON Collector(lastName, firstName);

CREATE INDEX PotentialCustomer_zip ON PotentialCustomer(zip);
CREATE INDEX PotentialCustomer_lastName_firstName ON PotentialCustomer(lastName, firstName);

CREATE UNIQUE INDEX Artwork_artistId_workTitle ON Artwork (artistId, workTitle);
CREATE INDEX Artwork_artistId ON Artwork(artistId);
CREATE INDEX Artwork_collectorSocialSecurityNumber ON Artwork
(collectorSocialSecurityNumber);

CREATE INDEX Show_showFeaturedArtistId On Show (showFeaturedArtistId);

CREATE INDEX Shownin_artworkId ON Shownin (artworkId);
CREATE INDEX Shownin_show Title ON ShownIn (showTitle);

CREATE INDEX Buyer_zip ON Buyer(zip);
CREATE INDEX Buyer_lastName_firstName ON Buyer (lastName, firstName);

CREATE UNIQUE INDEX Salesperson_lastName_firstName ON Salesperson (lastName, firstName);
CREATE INDEX Salesperson_zip ON Salespeson (zip);

CREATE INDEX Sale_buyerId ON Sale (buyerId);
```

FIGURA 6.11

Enunciados INSERT para poblar las tablas de la Galería de Arte

```
INSERT INTO Zips VALUES ('10101', 'New York', 'NY');
INSERT INTO Zips VALUES ('10801', 'New Rochelle', 'NY');
INSERT INTO Zips VALUES ('92101', 'San Diego', 'CA');
INSERT INTO Zips VALUES ('33010', 'Miami', 'FL');
INSERT INTO Zips VALUES ('60601', 'Chicago', 'IL');

CREATE SEQUENCE artistId_sequence;
INSERT INTO Artist VALUES(artistId_sequence.NEXTVAL, 'Leonardo', 'Vincenti', '10-Oct-1999', 'Hughes,''212', '5559999', '10 Main
Street', '10101', 9000, 4500, '099999876', 'oil', 'realism', 'painting');
INSERT INTO Artist VALUES(artistId_sequence.NEXTVAL, 'Vincent', 'Gogh', '15-Jun-2004', 'Hughes', '914,''5551234,''55
West 18 Street', '10801', 9500, 5500, '099999877', 'oil', 'impressionism', 'painting');
INSERT INTO Artist VALUES(artistId_sequence.NEXTVAL, 'Winslow', 'Homes', '05-Jan-2004', 'Hughes', '619', '1234567', '100
Water Street', '92101', 14000, 4000, '083999876', 'watercolor', 'realism', 'painting');
INSERT INTO Artist VALUES(artistId_sequence.NEXTVAL, 'Alexander', 'Calderone', '10-Feb-1999', 'Hughes', '212', '5559999',
'10 Main Street', '10101', 20000, 20000, '123999876', 'steel', 'cubism', 'sculpture');
INSERT INTO Artist VALUES(artistId_sequence.NEXTVAL, 'Georgia', 'Keefe', '05-Oct-2004', 'Hughes', '305', '1239999', '5
Chestnut Street', '33010', 19000, 14500, '987999876', 'oil', 'realism', 'painting');

INSERT INTO Collector VALUES('102345678', 'John', 'Jackson', '01-Feb-2004', 'Hughes', '917', '7771234', '24 Pine Avenue',
'10101', 4000, 3000, 1, 'oil', 'realism', 'collage');
INSERT INTO Collector VALUES ('987654321', 'Mary', 'Lee', '01-Mar-2003', 'Jones', '305', '5551234', '10 Ash Street', 33010,
'2000', 3000, 2, 'watercolor', 'realism', 'painting');
INSERT INTO Collector VALUES('034345678', 'Ramon', 'Perez', '15-Apr-2003', 'Hughes', '619', '8881234', '15 Poplar Avenue',
'92101', 4500, 3500, 3, 'oil', 'realism', 'painting');
INSERT INTO Collector VALUES('888881234', 'Rick', 'Lee', '20-Jun-2004', 'Hughes', '212', '9991234', '24 Pine Avenue', '10101',
4000, 3000, 3, 'oil', 'realism', 'sculpture');
```

(*continúa*)

FIGURA 6.11
Continuación

```
INSERT INTO Collector VALUES('777345678', 'Samantha', 'Torno', '05-May-2004', 'Jones', '305', '5551234', '10 Ash Street',
'33010', 40000, 30000, 1, 'acrylic', 'realism', 'painting');

CREATE SEQUENCE potentialCustomerId-sequence;
INSERT INTO PotentialCustomer VALUES(potentialCustomerId_sequence.NEXTVAL, 'Adam', 'Burns', '917', '3456789', '1
Spruce Street', '10101', '12-Dec-2003', 1, 'watercolor', 'impressionism', 'painting');
INSERT INTO PotentialCustomer VALUES(potentialCustomerId sequence.NEXTVAL, 'Carole', 'Burns', '917', '3456789', '1 Spruce
Street', '10101', '12-Dec-2003', 2, 'watercolor', 'realism', sculpture');
INSERT INTO PotentialCustomer VALUES(potentialCustomerId_sequence.NEXTVAL, 'David', 'Engel', '914', '7777777', '715
North Avenue', '10801', '08-Aug-2003', 3, 'watercolor', 'realism', 'painting');
INSERT INTO PotentialCustomer VALUES(potentialCustomerId_sequence.NEXTVAL, 'Frances', 'Hughes', '619', '3216789', '10
Pacific Avenue', '92101', '05-Jan-2004', 2, 'oil', 'impressionism', 'painting');
INSERT INTO PotentialCustomer VALUES(potentialCustomerId_sequence.NEXTVAL, 'Irene', 'Jacobs', '312', '1239876', '1
Windswept Place', '60601', '21-Sep-2003', 5, 'watercolor', 'abstract expressionism', 'painting');

CREATE SEQUENCE artworkId_sequence;
INSERT INTO Artwork VALUES(artworkid_sequence.NEXTVAL, 1, 'Flight', 15000.00, '08-Sep-2003',NULL ,NULL, 'for
sale',oil', '36 in X 48 in', 'realism', 'painting',2001',NULL );
INSERT INTO Artwork VALUES(artworkid_sequence.NEXTVAL, 3, 'Bermuda Sunset', 8000.00, '15-Mar-2004',NULL
,01-Apr-2004' , 'sold','watercolor', '22 in X 28 in', 'realism', 'painting', 2003',NULL );
INSERT INTO Artwork VALUES(artworkid_sequence.NEXTVAL, 3, 'Mediterranean Coast', 4000.00, '18-Oct-2003',NULL
,01-Apr-2004', 'for sale',watercolor', '22 in X 28 in', 'realism', 'painting','2000',102345678');
INSERT INTO Artwork VALUES(artworkid_sequence.NEXTVAL, 5, 'Ghost orchid', 18000.00',
05-Jun-2003',NULL ,NULL , 'sold',oil', '36 in X 48 in', 'realism', 'painting','2001',034345678' );
INSERT INTO Artwork VALUES(artworkid_sequence.NEXTVAL, 4, 'Five Planes', 15000.00,
'10-Jan-2004',NULL ,'10-Mar-2004' , 'for sale',steel', '36inX30inX60in', 'cubism''sculpture' '2003',034345678' );

INSERT INTO Show VALUES('The Sea in Watercolor',3, '30-Apr-2004', 'seascapes', '01-Apr-2004');
INSERT INTO Show VALUES('Calderone: Mastery of Space',4,'20-Mar-2004',mobiles', '10-Mar-2004');

INSERT INTO ShownIn VALUES(2,'The Sea in Watercolor');
INSERT INTO ShownIn VALUES(3, 'The Sea in Watercolor');
INSERT INTO ShownIn VALUES(5, 'Calderone: Mastery of Space');

CREATE SEQUENCE buyerId_sequence;
INSERT INTO Buyer VALUES (BuyerId_sequence.NEXTVAL, 'Valerie', 'Smiley', '15 Hudson Street', '10101', '718','5551234',
5000, 7500);
INSERT INTO Buyer VALUES (BuyerId_sequence.NEXTVAL, 'Winston', 'Lee', '20 Liffey Avenue', '60601', '312','7654321',
3000, 0);
INSERT INTO Buyer VALUES (BuyerId_sequence.NEXTVAL, 'Samantha', 'Babson', '25 Thames Lane', '92101', '619','4329876',
15000, 0);
INSERT INTO Buyer VALUES (BuyerId_sequence.NEXTVAL, 'John', 'Flagg', '22 Amazon Street', '10101', '212','7659876',
3000, 0);
INSERT INTO Buyer VALUES (BuyerId_sequence.NEXTVAL, 'Terrence', 'Smallshaw', '5 Nile Street', '33010', '305','2323456',
15000, 17000);

INSERT INTO Salesperson VALUES('102445566', 'John','Smith', '10 Sapphire Row', '10801');
INSERT INTO Salesperson VALUES('121344321', 'Alan','Hughes', '10 Diamond Street', '10101');
INSERT INTO Salesperson VALUES('101889988', 'Mary','Brady', '10 Pearl Avenue', '10801');
INSERT INTO Salesperson VALUES('111223344', 'Jill','Fleming', '10 Ruby Row', '10101');
INSERT INTO Salesperson VALUES('123123123', 'Terrence','DeSimone', '10 Emerald Lane', '10101');

INSERT INTO Sale VALUES(1234, 2,NULL ,'05-Apr-2004',7500,600, 1, '102445566');
INSERT INTO Sale VALUES(1235, 6,NULL ,'06-Apr-2004',17000,1360, 5, '121344321');
```

- Paso 6.5. Escriba enunciados SQL que procesarán cinco solicitudes no rutinarias para información de la base de datos recién creada. Para cada una, escriba la solicitud en inglés, seguida por el correspondiente comando SQL.

1. Encuentre los nombres de todos los artistas que se entrevistaron después del 1 de enero de 2004, pero que no tengan obras de arte en lista.

```
SELECT firstName, lastName
FROM Artist
WHERE interviewDate > '01-Jan-2004' AND NOT EXISTS
  (SELECT *
  FROM Artwork
  WHERE artistId =Artist.artistId);
```

2. Encuentre la comisión total para el vendedor John Smith obtenida entre las fechas 1 de abril de 2004 y 15 de abril de 2004. Recuerde que la galería carga 10% de comisión y el vendedor recibe la mitad de eso, que es 5% del precio de venta.

```
SELECT .05 * SUM(salePrice)
FROM Sale
WHERE saleDate > = '01-Apr-2004' AND
  saleDate < = '15-Apr-2004' AND
  salespersonSocialSecurityNumber = (SELECT socialSecurityNumber
                                     FROM Salesperson
                                     WHERE firstName= 'John' AND lastName
                                       = 'Smith');
```

3. Encuentre los nombres de coleccionista, nombres de artista y títulos de todas las obras de arte que sean propiedad de coleccionistas (collectors), no de los propios artistas, en orden del apellido del coleccionista.

```
SELECT Collector.firstName, Collector.lastName, Artist.firstName, Artist.
lastName, workTitle
FROM Artist, Artwork, Collector
WHERE   Artist.artistId = Artwork.artistId AND
        Artwork.collectorSocialSecurityNumber =
        Collector.socialSecurityNumber AND
        collectorSocialSecurityNumber IS NOT NULL
ORDER BY Collector.lastName, Collector.firstName;
```

4. Para cada comprador potencial, encuentre información acerca de las exposiciones que presenten a su artista preferido.

```
SELECT firstName, lastName, showTitle, showOpeningDate, showClosingDate,
potentialCustomerId
FROM Show, PotentialCustomer
WHERE showFeaturedArtistId = PotentialCustomer.preferredArtistId
ORDER BY potentialCustomerId;
```

5. Encuentre el precio de venta promedio de las obras de la artista Georgia Keefe.

```
SELECT AVG(salePrice)
FROM Sale
WHERE artworkId IN (SELECT artworkId
  FROM Artwork
  WHERE artistId = (SELECT ArtistId
                    FROM Artist
                    WHERE lastName = 'Keefe' AND firstName ='Georgia'));
```

- Paso 6.6. Cree al menos un disparador y escriba el código para él. Este disparador actualizará la cantidad de compras del comprador del año a la fecha siempre que una venta se complete.

```
CREATE TRIGGER UPDATEBUYERYTD
AFTER INSERT ON Sale
FOR EACH ROW
BEGIN
  UPDATE Buyer
  SET purchasesYearToDate = purchasesYearToDate + :NEW.salePrice
  WHERE Buyer.buyerId = :NEW.buyerId;
END;
```

PROYECTOS ESTUDIANTILES: CREACIÓN Y USO DE UNA BASE DE DATOS RELACIONAL PARA LOS PROYECTOS ESTUDIANTILES

Para las tablas normalizadas que desarrolló al final del capítulo 5 para el proyecto que eligió, realice los siguientes pasos para implementar el diseño usando un sistema de gestión de base de datos relacional como Oracle, SQLServer o MySQL.

- Paso 6.1. Actualice el diccionario de datos y lista de suposiciones según se requiera. Para cada tabla, escriba el nombre de tabla y los nombres, tipos de datos y tamaños de todos los ítems de datos, e identifique cualquier restricción, usando las convenciones del DBMS que usará para la implementación.
- Paso 6.2. Escriba y ejecute enunciados SQL para crear todas las tablas necesarias para implementar el diseño. El website para este libro tiene instrucciones para usar SQL con el fin de crear una base de datos Access.
- Paso 6.3. Cree índices para claves externas y para cualquier otra columna que necesite.
- Paso 6.4. Inserte al menos cinco registros en cada tabla, que preserve todas las restricciones. Ponga suficientes datos para demostrar cómo funcionará la base de datos.
- Paso 6.5. Escriba enunciados SQL que procesarán cinco solicitudes no rutinarias para información de la base de datos recién creada. Para cada una, escriba la solicitud en inglés, seguido por el correspondiente comando SQL.
- Paso 6.6. Cree al menos un disparador y escriba el código para él.

CAPÍTULO 7

El modelo entidad-relación extendido y el modelo objeto-relacional

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Por qué el modelo E-R se extendió al modelo EE-R
- El significado de generalización y especialización
- Cómo representar una jerarquía de generalización en un diagrama EE-R
- Cómo representar restricciones de generalización
- El significado de unión
- Cómo usar notación (*mín..máx*) para restricciones de relación
- Cómo mapear un modelo EE-R a un modelo estrictamente relacional
- Cómo usar las extensiones orientadas a objeto para el modelo relacional con SQL
- Cómo mapear un modelo EE-R a un modelo objeto-relacional
- Cómo Oracle implementa el modelo objeto-relacional

7.1 Razones para la extensión del modelo E-R

Aunque el modelo E-R es suficiente para representar las necesidades de datos en las aplicaciones empresariales tradicionales, no es a nivel semántico suficientemente rico para modelar los entornos más complejos utilizados en aplicaciones más avanzadas. Las bases de datos utilizadas por los sistemas de información geográfica, motores de búsqueda, minado de datos, multimedia, diseño y fabricación asistida por computadora (CAD/CAM), desarrollo de software, diseño de ingeniería y muchas otras aplicaciones sofisticadas deben ser capaces de representar información más semántica de lo que el modelo E-R estándar puede representar. El modelo **entidad-relación extendido (EE-R)** extiende el modelo E-R para permitir la inclusión de varios tipos de abstracción, y para expresar restricciones más claramente. A los diagramas E-R estándar se agregan símbolos adicionales para crear diagramas EE-R que expresen estos conceptos.

7.2 Generalización y especialización

El proceso de generalización y su inverso, la especialización, son dos abstracciones que se usan para incorporar diagramas EE-R más significativos.

7.2.1 Especialización

Con frecuencia, un conjunto de entidades contiene uno o más subconjuntos que tienen atributos especiales o que participan en relaciones que otros miembros del mismo conjunto de entidades no tiene. Por ejemplo, en el ejemplo University cuyo diagrama E-R se muestra en la figura 3.12, el conjunto de entidades `Faculty` se puede descomponer en subconjuntos, `AdjunctFac` y `FullTimeFac`. Los miembros adjuntos del personal docente imparten clases tiempo parcial y por lo general se citan con una base temporal, mientras que los docentes de tiempo completo son empleados regulares que trabajan sobre una base continua. A los docentes adjuntos se les paga por curso, mientras que a los docentes de tiempo completo se les paga un salario anual. Todos los docentes tienen atributos `facId`, `lastName`, `firstName` y `rank`, pero los miembros `AdjunctFac` tienen un atributo local llamado `coursePayRate` (tasa de pago por curso), mientras que los `FullTimeFac` tienen un atributo local llamado `annualSalary` (salario anual).

El método de identificar subconjuntos de conjuntos de entidades existentes, llamado **especialización**, corresponde a la noción de herencia de subclase y clase en el diseño orientado a objetos, donde se representa mediante jerarquías de clase, como se discute en el capítulo 8. La figura 7.1(a) muestra cómo esta relación superclase-subclase se representa en los diagramas EE-R. En este diagrama, la superclase, `Faculty`, se especializó en dos subclases, `AdjunctFac` y `FullTimeFac`. El círculo debajo de `Faculty` se llama **círculo de especialización**, y se conecta mediante una línea a la superclase. Cada subclase se conecta al círculo mediante una línea que tiene un **símbolo de herencia**, un símbolo de subconjunto o copa, con el lado abierto de frente a la superclase. Las subclases heredan los atributos de la superclase, y opcionalmente pueden tener atributos locales distintos, como `coursePayRate` para `AdjunctFac` y `annualSalary` para `FullTimeFac`. Dado que cada miembro de una subclase **es** miembro de la superclase, al círculo de especialización a veces se le conoce como **relación *isa***.

En ocasiones una entidad tiene sólo un subconjunto con propiedades o relaciones especiales de las que quiere tener información. Sólo contiene una subclase para una especialización. En este caso, en el diagrama EE-R se omite el círculo y simplemente se muestra la subclase conectada mediante una línea de subconjunto a la superclase. Por ejemplo, si algu-

nas instancias de entidad `Class` universitarias tienen un componente laboratorio para el que se quiera tener información acerca del laboratorio usado y el horario del laboratorio, se podría diseñar una subclase únicamente de dichas entidades, como se muestra en la figura 7.1(b). Note que `LabClass` hereda todos los atributos de `Class` (es decir, `LabClass` *isa* `Class`), además de tener sus atributos locales de `labNumber` y `labSched`.

Las subclases también pueden participar en relaciones locales que no se apliquen a la superclase o a otras subclases en la misma jerarquía. Por ejemplo, sólo los miembros docentes regulares pueden suscribirse a un plan de pensión. Se supone que cada docente de tiempo completo puede elegir una de muchas compañías, y tiene un número de contrato único para su plan de pensión con dicha compañía. Cada compañía tiene varias personas de contacto, de quienes una se asigna para corresponder con el miembro del personal docente. La figura 7.1(c) ilustra esta posibilidad.

7.2.2 Generalización

Mediante el proceso de especialización se desarrolló la jerarquía de clase `Faculty` al descomponer una clase existente en subclases. También se pueden crear jerarquías de clase al reconocer que dos o más clases tienen propiedades comunes e identificar una superclase común para ellas, un proceso llamado **generalización**. Estos dos procesos son inversos uno de otro, pero ambos resultan en el mismo tipo de diagrama jerárquico. Por ejemplo, dado que los estudiantes y docentes tienen ambas ID y nombres como atributos, estos dos conjuntos de entidades se podrían generalizar en una nueva superclase, `Person`, que tenga los atributos comunes. Las subclases `Student` y `Faculty` conservarían cada una sus atributos especiales, como se muestra en la figura 7.1(d). Al usar la generalización, se desarrolló este diagrama de abajo arriba. Note que se tendría el mismo diagrama si comenzara con `Person` y se especializara en `Student` y `Faculty`, pero habría trabajado de arriba abajo.

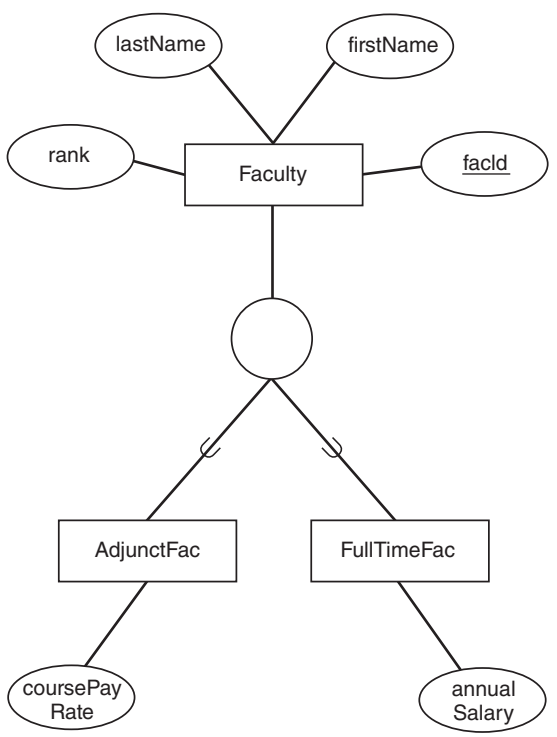


FIGURA 7.1(a)

Especialización con dos subclases

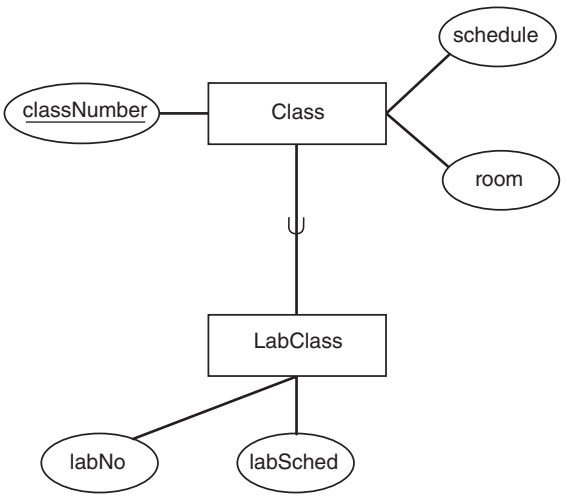


FIGURA 7.1(b)

Especialización con una subclase

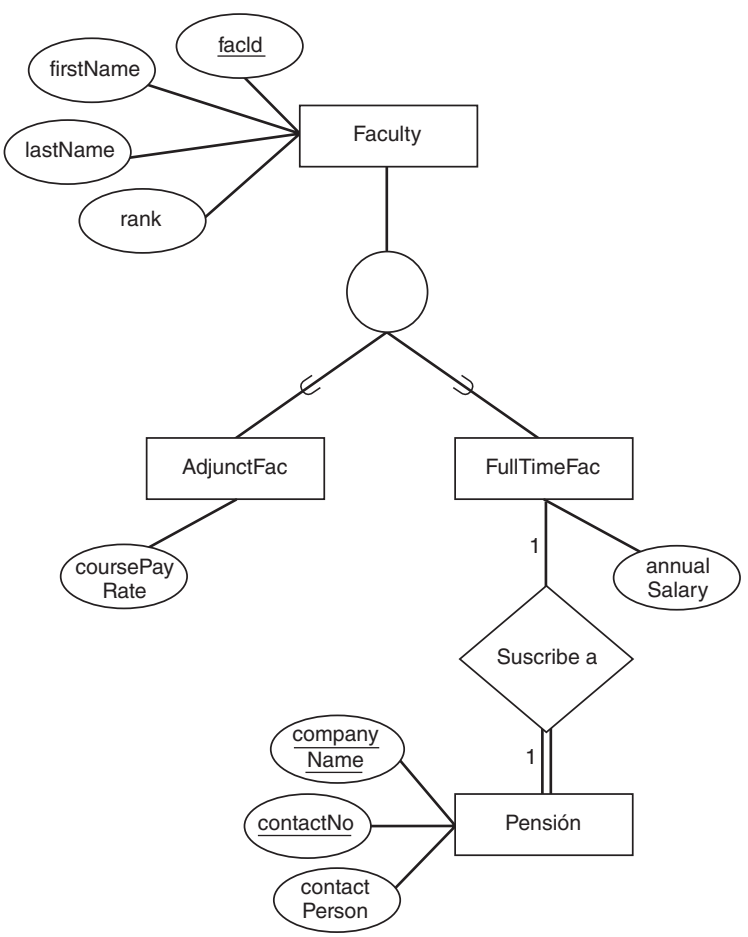


FIGURA 7.1(c)

Subclase con relación

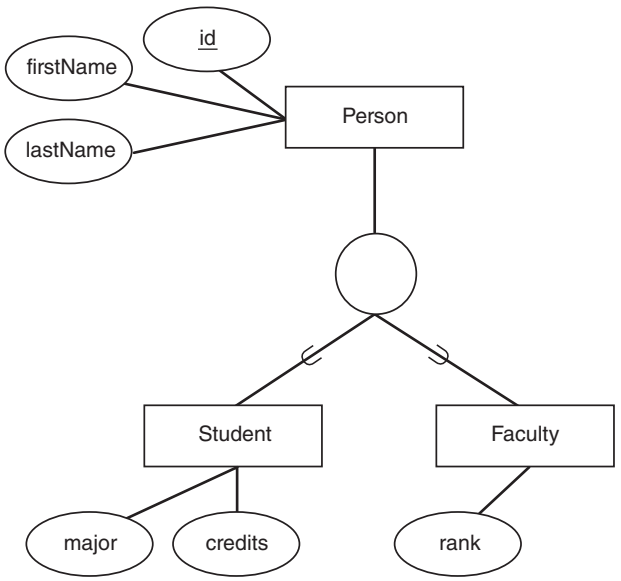


FIGURA 7.1(d)

Generalización con superclase Person

7.2.3 Restricciones de generalización: desarticulación, completud, método de definición

Las subclases pueden ser **traslapantes** (overlapping), lo que significa que la misma instancia de entidad puede pertenecer a más de una de las subclases, o **desarticuladas** (disjoint), lo que significa que no tienen miembros en común. A esto se le refiere como restricción de **desarticulación** (disjointness) y se expresa al colocar una letra adecuada, `d` u `o`, en el círculo de especialización. Una `d` indica subclases de desarticulación, y una `o` indica subclases de traslapamiento. Por ejemplo, dado que un docente no puede ser tanto adjunto como de tiempo completo, se coloca una `d` en el círculo para dicha especialización, como se muestra en la esquina inferior izquierda de la figura 7.2(a). En el círculo de especialización de `Person` se colocó una `o` en `Faculty` y `Student` para indicar que una persona puede ser tanto miembro del personal docente como estudiante. Se supone, sólo para este diagrama, que un docente puede ser capaz de tomar cursos de posgrado, por ejemplo, y en consecuencia podría ser un miembro del conjunto de entidades `Student`, así como del conjunto `Faculty`.

Los estudiantes también se especializan desarticuladamente en este diagrama, en pregrado y graduado. Ambos tipos de estudiantes tienen atributos `id`, `lastName` y `firstName` heredados de `Person`, ambos tienen el atributo `credits`, pero los de pregrado tienen `major` como atributo, mientras que los estudiantes graduados tienen `program`. Los estudiantes de pregrado se especializan desarticuladamente todavía más en años de clase, como se muestra mediante la `d` en el círculo para dicha especialización. Aunque no se les mencionó, se supondrá que cada año de clase tiene algún atributo distintivo. Por ejemplo, los estudiantes de primer año (freshmen) podrían tener un mentor par, los de cuarto año (seniors) tener un supervisor de tesis, y así por el estilo.

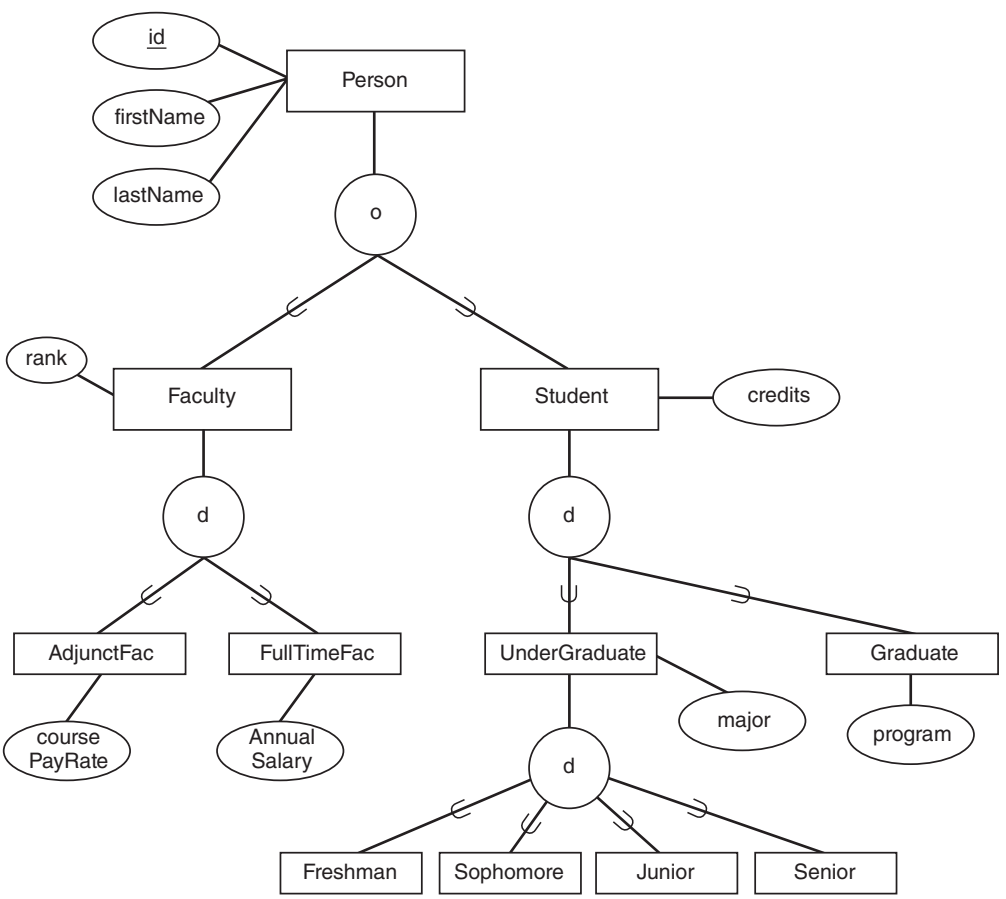
Una especialización también tiene una restricción de **completud** (completeness), que muestra si todo miembro del conjunto de entidades debe participar en ella. Si todo miembro de la superclase debe pertenecer a alguna subclase se tiene una especialización **total**. Si a algunos miembros de la superclase no se les puede permitir pertenecer a alguna subclase, la

especialización es **parcial**. Por ejemplo, dado que todo miembro del personal docente debe ser o `AdjunctFac` o `FullTimeFac`, se tiene especialización total. En la figura 7.2(b) esto se indica mediante la línea doble que conecta la superclase, `Faculty`, al círculo de especialización en la esquina inferior izquierda del diagrama. La línea sencilla que conecta `Person` con su círculo de especialización indica que la especialización en `Faculty` y `Student` es parcial. Puede haber algunas entidades `Person` que no son docentes ni estudiantes. La línea doble abajo de `Student` indica una especialización total en graduado o pregrado. La línea doble debajo de pregrado muestra una especialización total en años clase, lo que implica que todo estudiante de pregrado debe pertenecer a uno de los años clase.

En algunas jerarquías de especialización es posible identificar la subclase a la que pertenece una entidad al examinar una condición o predicado específico para cada subclase. En la especialización de `UnderGraduates`, si el predicado "credits <30" es verdadero, el estudiante pertenece a la subclase freshman, mientras que si el predicado "credits > = 30 AND credits <60" es verdadero, el estudiante es un sophomore, y así para las otras dos subclases. Éste es un ejemplo de una especialización **definida por predicado**, pues la membresía a la subclase está determinada por un predicado. En un diagrama EE-R se puede escribir el predicado definitorio en la línea desde el círculo de especialización hasta la subclase, como se muestra en la figura 7.2(c).

Algunas especializaciones definidas por predicado, incluida ésta, usan el valor del mismo atributo en el predicado definitorio para todas las subclases. Éstas se llaman especializaciones **definidas por atributo**. El atributo definitorio se indica en la línea desde el círculo hasta la superclase, y el valor distintivo para cada subclase en la línea desde la subclase hasta

FIGURA 7.2(a)
Subclases desarticulación frente a traslapamiento



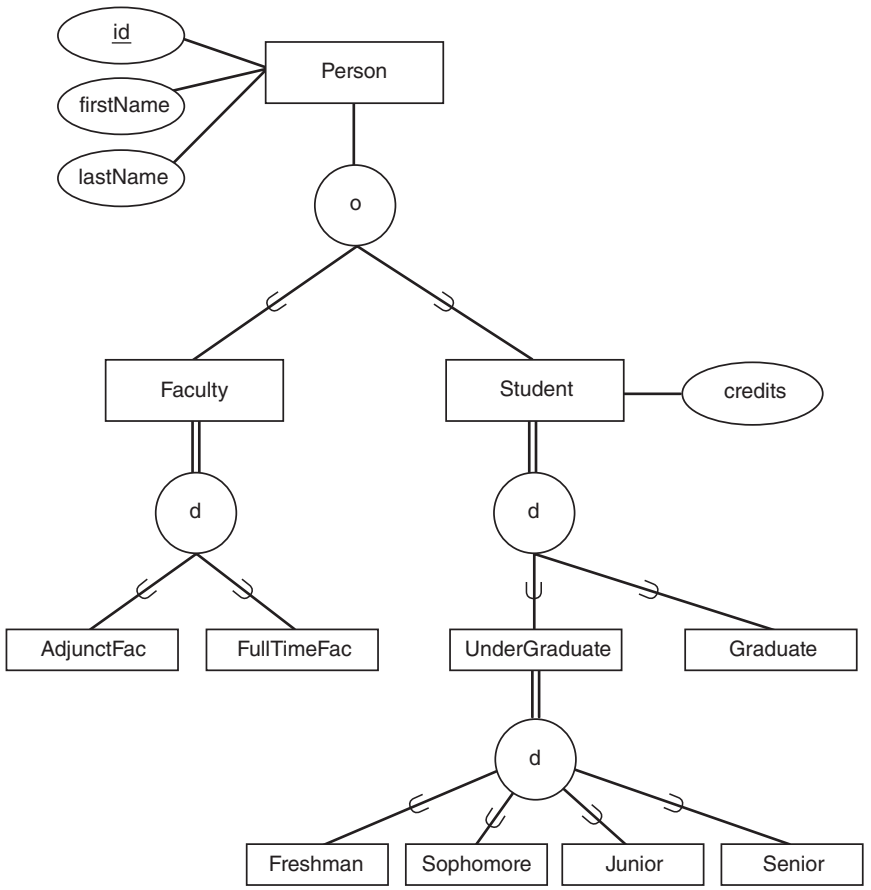


FIGURA 7.2(b)
Participación total y parcial en especialización

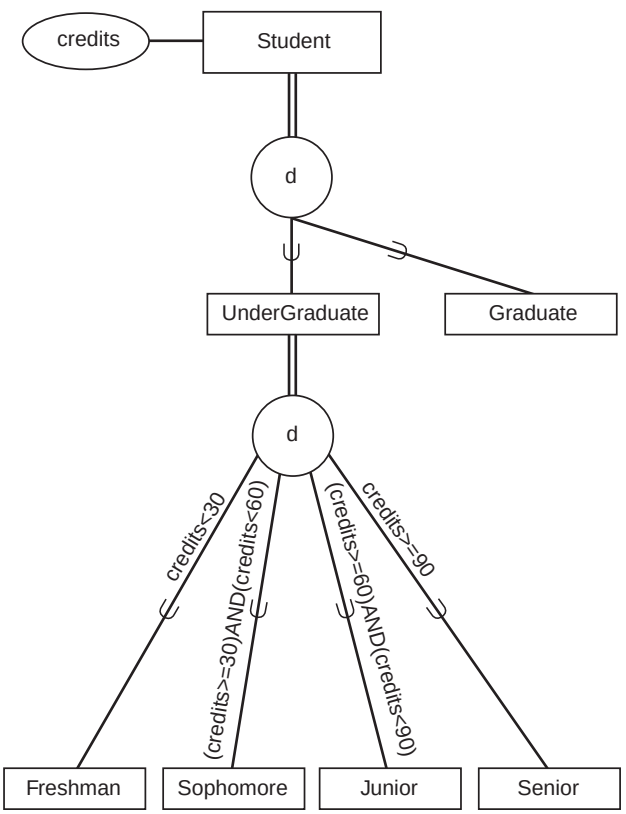
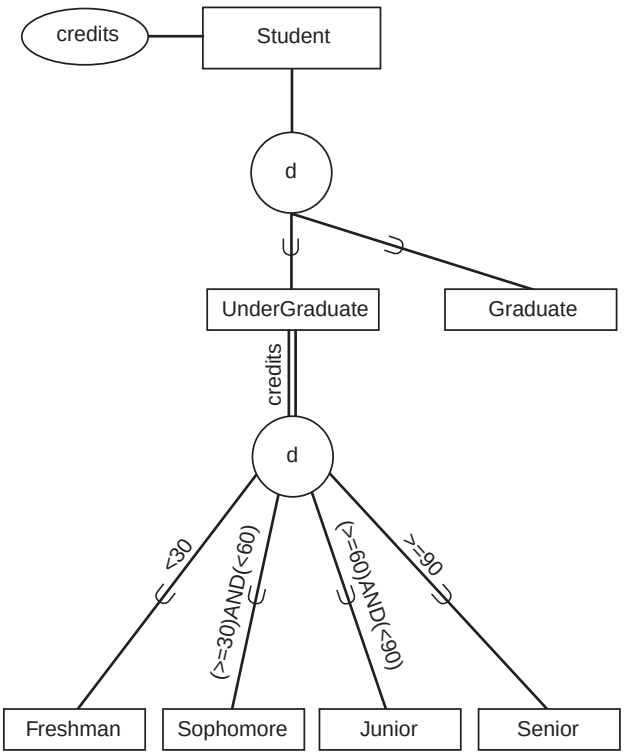


FIGURA 7.2(c)
Especialización definida por predicado

FIGURA 7.2(d)
Especialización definida por atributo



el círculo, como se muestra para la especialización año de clase en la figura 7.2(d). Las especializaciones que no están definidas por predicado se dice que son **definidas por el usuario**, pues el usuario es el responsable de colocar la instancia de entidad en la subclase correcta. En el ejemplo, la especialización de `Person` en `Faculty` y `Student`; de `Faculty` en `Adjunct` y `FullTime`, y `Student` en `UnderGraduate` y `Graduate` son todas definidas por el usuario.

7.2.4 Jerarquías múltiples y herencia

Además de especializar a los estudiantes de pregrado de acuerdo con el año de clase, un diseñador de bases de datos podría decidir que es importante especializarlos por su estatus de residencia. La figura 7.3(a) muestra esta mayor especialización de `UnderGraduate` por residencia de acuerdo con el hecho de si un estudiante vive en un dormitorio dentro del campus, una propiedad universitaria afuera del campus o en casa. La especialización en año de clase y la especialización en residencia son mutuamente independientes, y ambas pueden aparecer en el mismo diagrama. De esta forma, el modelo EE-R permite dar a la misma superclase más de una especialización.

A veces el mismo conjunto de entidades puede ser una subclase de dos o más superclases. Se dice que tal clase es una **subclase compartida** y tiene **herencia múltiple** de sus superclases. Por ejemplo, adicional al ejemplo University, como se muestra en la figura 7.3(b), algunos estudiantes graduados pueden ser asistentes de profesor (teaching assistants). Los asistentes pueden recibir financiamiento para sus matrículas de un `fundingSource` (fuente de financiación) y auxiliar en algunas clases introductorias en la universidad, por lo cual reciben un salario. Los miembros de la subclase `TeachingAssistant` heredan todos los atributos de `Faculty` y de `Graduate`, y de las superclases de dichas clases, `Student` y `Person`, además de tener su propio atributo distintivo de `fundingSource`. Como con todas las subclases, las subclases compartidas pueden participar en relaciones,

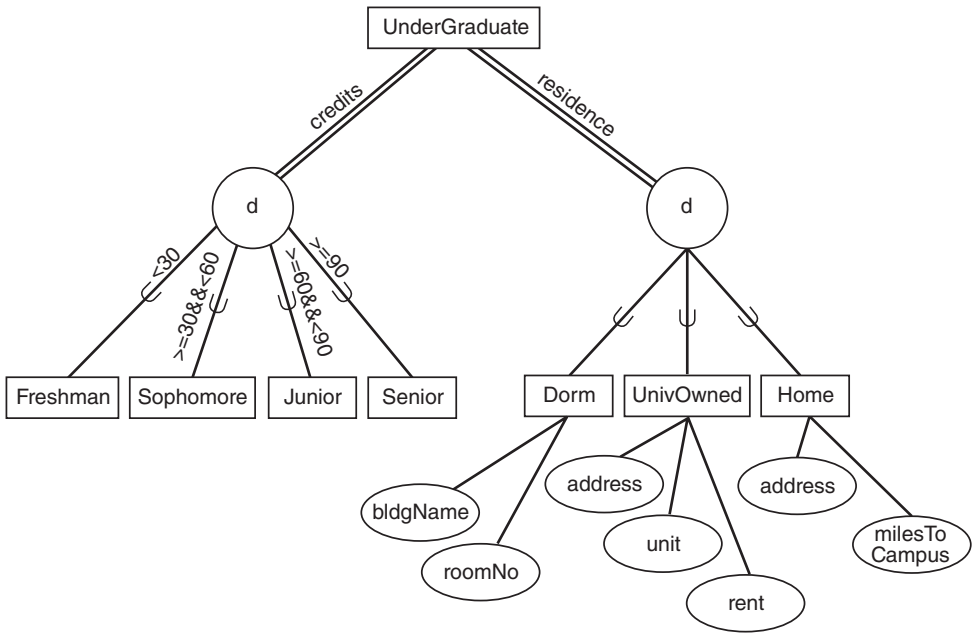


FIGURA 7.3(a)

Especializaciones múltiples, por año de clase y por residencia

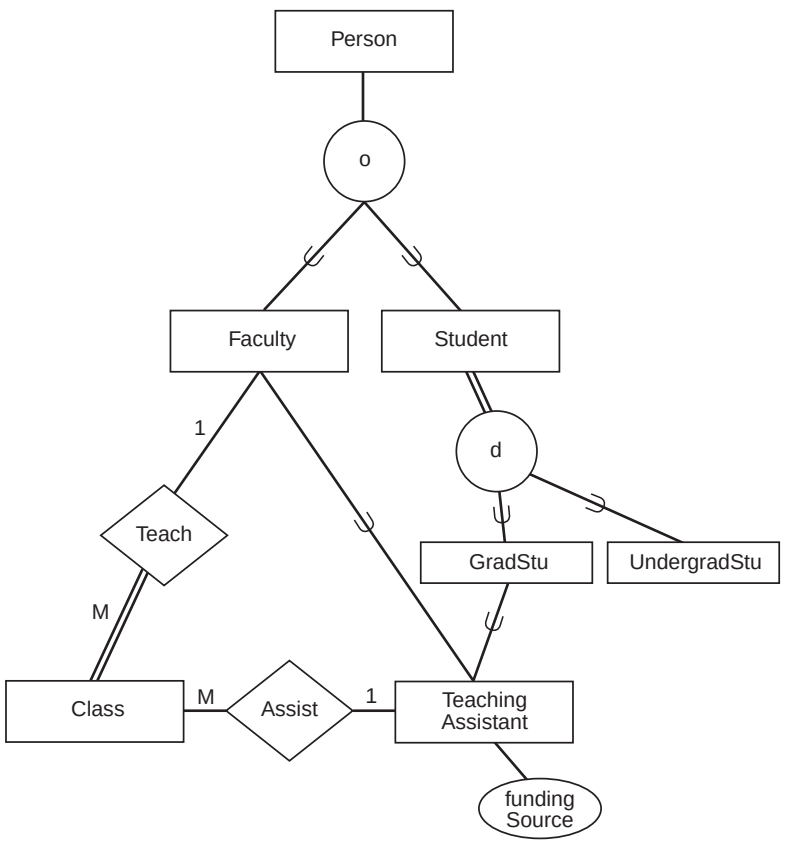


FIGURA 7.3(b)

Herencia múltiple

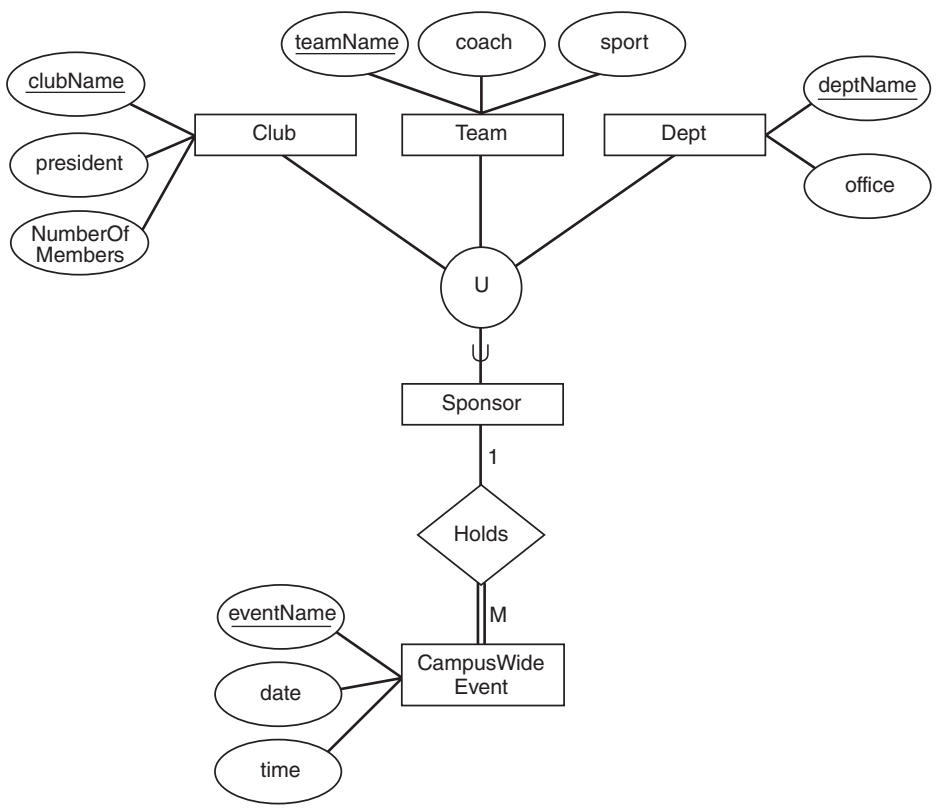
como se ve en `TeachingAssistant` que participa en la relación `Assist` con el conjunto de entidades `Class`.

7.3 Unión

Mientras que una subclase compartida, como se describió en la sección previa, representa un miembro de todas sus superclases y hereda atributos de todas ellas, una subclase se puede relacionar con la de una colección, llamada **unión** o **categoría**, de superclases, en vez de pertenecer a todas ellas. En este caso, una instancia de la subclase hereda sólo los atributos de una de las superclases, dependiendo de a cuál miembro de la unión pertenece. Al expandir el ejemplo University, suponga que se quiere representar patrocinadores de eventos en el campus, que pueden ser patrocinados por equipos, departamentos o clubes. Aquí se supone que un evento en el campus debe tener exactamente un patrocinador, pero puede ser un equipo, un departamento o un club. Se podría formar una unión de `Team` (equipo), `Department` y `Club`, que entonces es el conjunto de todas estas entidades. `Sponsor` (patrocinador) es una subclase de la unión. Los atributos de un patrocinador particular dependerán de cuál tipo es. La figura 7.4(a) ilustra la unión de `Club`, `Team` y `Dept`, representada por el círculo de unión debajo de ellos. La unión tiene subclase `Sponsor`, que se relaciona con `CampusWideEvent`. Las tres clases están conectadas al círculo con un símbolo de unión de conjunto en él. El círculo se conecta a la subclase `Sponsor` mediante una línea con el símbolo de subconjunto (taza) en ella, de frente al círculo.

Las categorías pueden ser parciales o totales, dependiendo de si cada miembro de los conjuntos que constituyen la unión participan en ella. La línea que conecta `Sponsor` a su círculo de unión en la figura 7.4(a) es sencilla, lo que indica una **categoría parcial**. Esto

FIGURA 7.4(a)
Ejemplo de unión



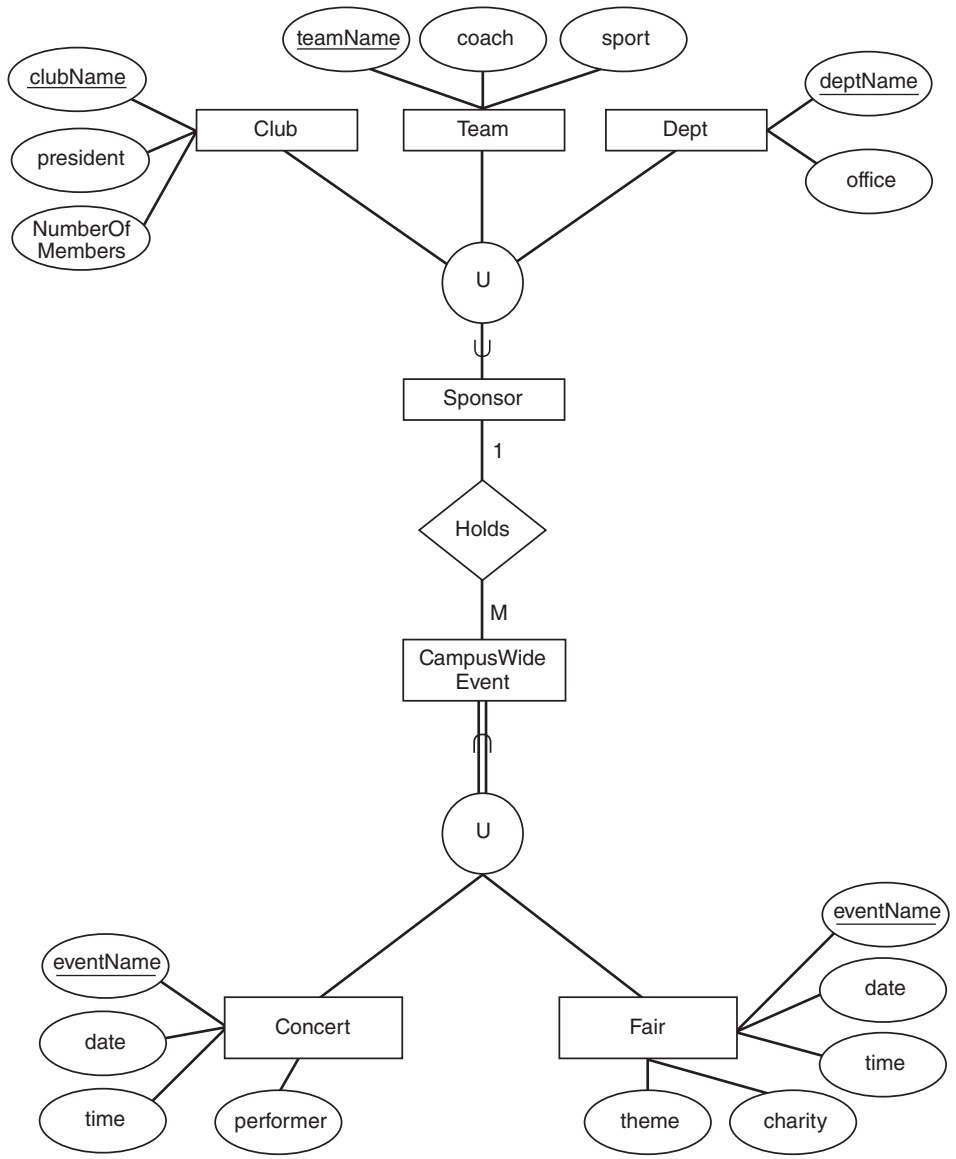


FIGURA 7.4(b)
Unión parcial frente a total

indica que no todo `Club`, `Team` o `Dept` tiene que ser un `Sponsor`. La elaboración de este ejemplo en la figura 7.4(b) muestra que `CampusWideEvent` es una unión de `Concert` (concierto) o `Fair` (feria). Esto indica que todo evento en el campus es o un concierto o una feria, y un evento particular tendrá los atributos de una de estas superclases.

La línea que conecta `CampusWideEvent` con el círculo de unión es doble, lo que indica una **categoría total**, lo que significa que todo `Concert` y `Fair` debe ser miembro de `CampusWideEvent`.

Diferentes diseñadores pueden hacer distintas elecciones acerca de cuándo usar una unión o una abstracción de especialización. Si una unión es total, y las superclases comparten muchos atributos, puede ser preferible usar especialización. La figura 7.5 muestra una especialización de eventos en el campus en dos subclases, en lugar de usar la unión de conciertos y ferias en eventos en el campus, como se hizo en la figura 7.4(b). Aquí se muestra una especialización definida por atributo, con `type` (tipo) como el atributo definitorio. Note que, como especialización/generalización, todo miembro de una subclase es automáticamente

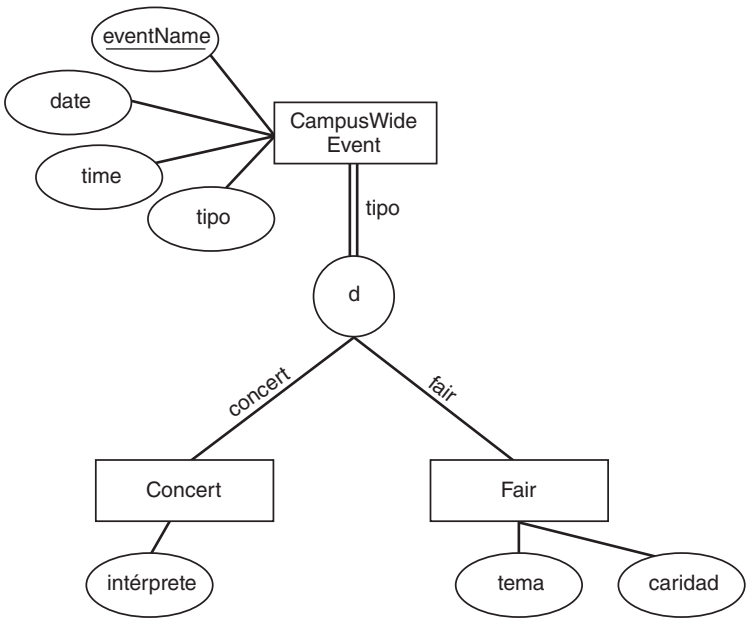


FIGURA 7.5

Unión total sustituida por especialización/generalización

miembro de la superclase. Es una especialización total, como se muestra mediante la doble línea que conecta `CampusWideEvent` al círculo de especialización, lo que indica que todos los eventos en el campus son o ferias o conciertos.

7.4 Uso de notación (*mín..máx*) para cardinalidad y participación

En los diagramas E-R y EE-R hasta el momento se usaron líneas sencillas o dobles con el fin de mostrar restricciones de participación para relaciones. Las relaciones de cardinalidad se mostraron con "1" para "uno" y "M" (o "N") para "muchos" en las líneas adecuadas. Una representación alternativa que muestra tanto restricciones de participación como de cardinalidad es la notación (*mín, máx*) que se ilustra en el diagrama E-R de la figura 7.6 para parte del ejemplo University. Se puede usar con diagramas E-R o EE-R. Cada línea que conecta un rectángulo de entidad a un diamante de relación tiene un par de enteros (*mín, máx*), escrito en los paréntesis de arriba. El primer entero del par, *mín*, especifica el mínimo número de instancias de relación en el que debe participar una instancia de entidad. Si *mín* es 0, significa que puede haber instancias de entidad que no participen en instancias de la relación, de modo que la participación es parcial. Si *mín* es 1 o más, la participación es total. El valor de *máx* es el mayor número de instancias de relación en las que puede participar una entidad. Este valor podría ser 1, N (para representar *muchos*), o algún entero constante, como 10. Algunas notaciones usan * en lugar de N para representar *muchos*. En la figura 7.6, la línea que conecta `Department` al diamante de relación `HasMajor` tiene un mín de 0, lo que indica que un departamento no requiere tener un estudiante con especialidad en ella, y un máx de M, que indica que un departamento puede tener muchos estudiantes con especialidad. La línea de `Student` al diamante de relación `HasMajor` tiene (0,1), lo cual significa que un estudiante no debe tener especialidad alguna, y cuando mucho tiene una especialidad. El (1,M) en la línea que conecta `Department` con la relación `Employs` (emplea) tiene un mín de 1, lo cual significa que un departamento debe tener al menos un

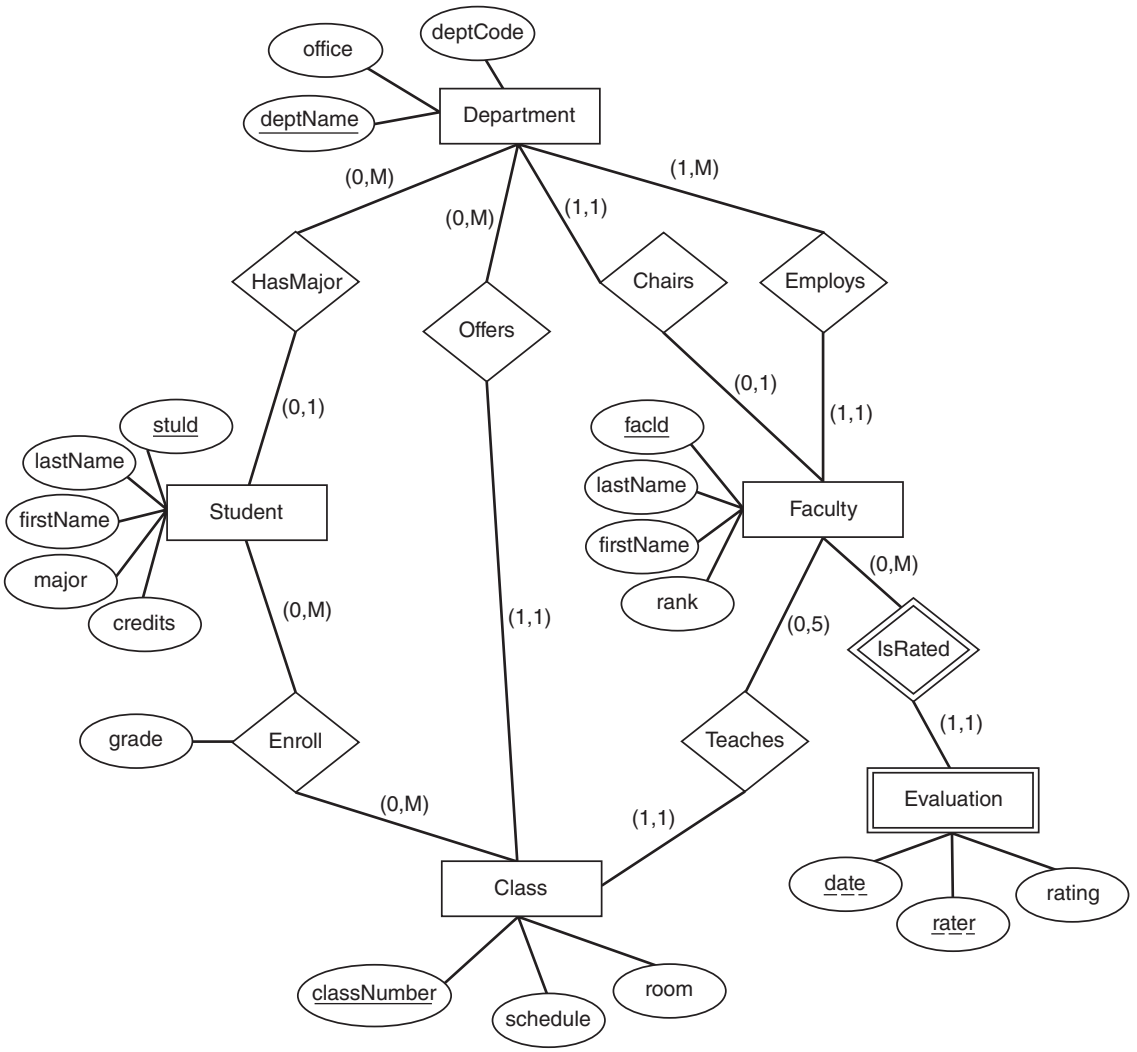


FIGURA 7.6

Parte del diagrama E-R University con notación (*mín, máx*)

miembro del personal docente, y un máx de M, pues un departamento puede tener muchos docentes. La línea de `Faculty` a la relación tiene (1,1), que indica que un docente pertenece exactamente a un departamento. En el diagrama se indican otras restricciones de participación y de cardinalidad. Note que todas estas restricciones reflejan suposiciones acerca de esta universidad en particular y que las restricciones pueden ser diferentes en otra institución.

7.5 Un diagrama de muestra EE-R

Ahora modifique el diagrama E-R University para mostrar algunas de las generalizaciones discutidas en secciones previas. La figura 7.7 muestra un diagrama EE-R para el ejemplo University, modificado para ilustrar el uso de generalización/especialización, unión y notación (*mín..máx*).

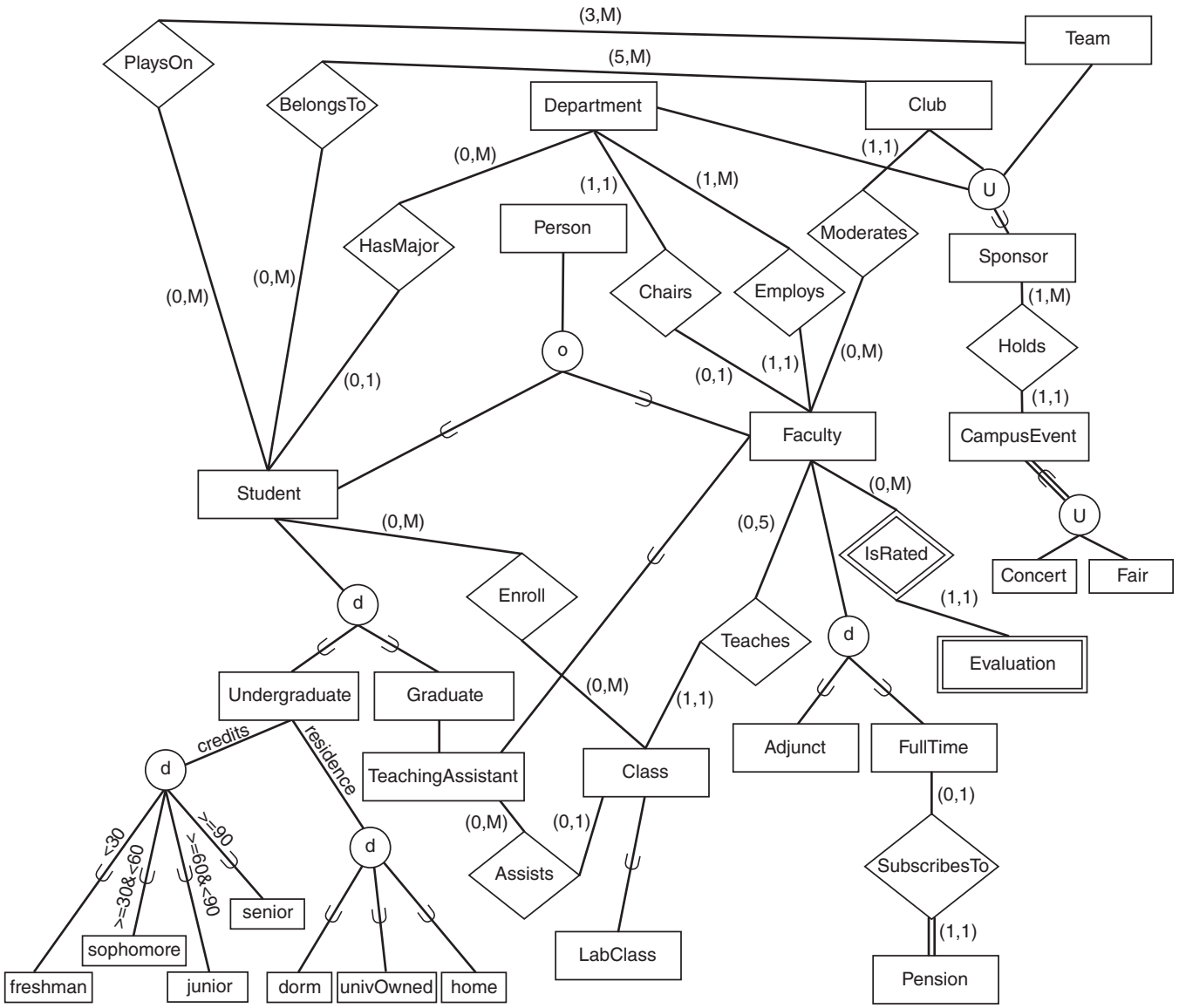


FIGURA 7.7

Diagrama EE-R para el ejemplo University

7.6 Mapeo de un modelo EE-R a un modelo relacional

Las técnicas discutidas en la sección 4.8 para mapear un modelo E-R a un modelo relacional se deben extender para tomar en cuenta las características adicionales del modelo EE-R, a saber, especialización/generalización, y unión.

7.6.1 Resumen de conceptos de mapeo de E-R a relacional

Para conjuntos de entidad y relaciones que no son parte de generalizaciones o uniones, el mapeo procede como para el modelo E-R.

- Los conjuntos de entidad fuerte se mapean a tablas que tengan una columna para cada atributo no compuesto de valor único.
  - Para atributos compuestos, se crea una columna para cada componente y no se representa el compuesto. Alternativamente, puede crear una sola columna para el compuesto e ignorar los componentes.
  - Para cada atributo multivaluado se crea una tabla por separado que tenga la clave primaria de la entidad, junto con una columna para el atributo multivaluado. Alternativamente, si conoce el número máximo de valores para el atributo, puede crear columnas múltiples para estos valores. Otro método es usar el atributo multivaluado de la clave primaria, lo que por tanto "aplana" la tabla.
- Para conjuntos de entidad débiles, cree tablas que incluyan la clave primaria de la entidad propietaria junto con los atributos del conjunto de entidad débil.
- Para relaciones binarias uno a muchos, coloque la clave primaria del lado uno en la tabla del lado muchos, lo que crea una clave externa.
- Para relaciones binarias uno a uno, coloque cualquiera de las claves primarias en la tabla para la otra entidad.
- Para relaciones binarias muchos a muchos, y para todas las relaciones de nivel superior, cree una tabla de relación que consista en las claves primarias de las entidades relacionadas, junto con cualquier atributo descriptivo de la relación.

7.6.2 Mapeo de jerarquías de clase EE-R a tablas relacionales

Para ilustrar la representación de subclases se usará la figura 7.1(a), que muestra `Faculty` con subclases `AdjunctFac` y `FullTimeFac`. Puede elegir uno de los siguientes métodos.

- **Método 1.** Cree una tabla para la superclase y una para cada una de las subclases, y coloque la clave primaria de la superclase en cada tabla de subclase. Para la figura 7.1(a) cree tres tablas, `Faculty`, `AdjunctFac` y `FullTimeFac`. Faculty tiene los atributos que se muestran en el diagrama, incluida una clave primaria, `facId`. Ambas tablas, `AdjunctFac` y `FullTimeFac` tienen columnas de sus propios atributos, más columnas para la clave primaria de la superclase, `facId`, que funciona tanto de clave primaria como de clave externa en dichas tablas. Las tablas son:

```
Faculty(facId, lastName, firstName, rank)
AdjunctFac(facId, coursePayRate)
FullTimeFac(facId, annualSalary)
```

- **Método 2.** Cree tablas para cada una de las subclases, y ninguna tabla para la superclase. Coloque todos los atributos de la superclase en cada una de las tablas subclase. Para este ejemplo, tendría tablas para `AdjunctFac` y `FullTimeFac`, y ninguna para `Faculty`. Las tablas son:

```
AdjunctFac(facId, lastName, firstName, rank, coursePayRate)
FullTimeFac(facId, lastName, firstName, rank, annualSalary)
```

- **Método 3.** Cree una sola tabla que contenga todos los atributos de la superclase, junto con todos los atributos de todas las subclases. Para el ejemplo, podría crear la tabla:

```
AllFac(facId, lastName, firstName, rank, annualSalary, coursePayRate)
```

Una variación del método 3 es agregar un "campo tipo" al registro, que indique a cuál subclase pertenece la entidad. Para especializaciones desarticuladas definidas por atributo, éste es simplemente el atributo definitorio. Para especializaciones de traslapamiento podría agregar un campo membresía para cada especialización, que indica si la entidad pertenece o no a cada una de las subclases.

Existen negociaciones a considerar para cada uno de estos métodos. La primera funciona para todos los tipos de subclases, sin importar si la especialización es desarticulada o traslapante; parcial o total, y definida por atributo o definida por usuario. Para el ejemplo, las consultas acerca del personal docente que involucra sólo los atributos comunes se responden fácilmente con el uso de la tabla superclase. Sin embargo, las consultas acerca de los miembros de una subclase, en general, requerirán una combinación con la tabla superclase. El segundo método también maneja las consultas acerca de las subclases, pues una tabla tiene toda la información acerca de cada subclase. Sin embargo, no se debe usar si las subclases se traslapan, porque los datos acerca de las instancias que pertenecen a más de una subclase se duplicarán. No se puede usar si la especialización es parcial, porque las instancias de entidad que no pertenezcan a subclase alguna no se pueden representar en absoluto usando este método. El tercer método permite responder consultas acerca de todas las subclases y la superclase sin hacer combinaciones, pero resultará en almacenar muchos valores nulos en la base de datos.

7.6.3 Mapeo de uniones

Para representar uniones (categorías) cree una tabla para la unión en sí, y tablas individuales para cada una de las superclases, usando claves externas para conectarlas. Un problema que surge con frecuencia con el mapeo de uniones es que las superclases pueden tener diferentes claves primarias. La figura 7.4(a) muestra `Sponsor` como la unión de `Club`, `Team` y `Department`, que tiene claves primarias `clubName`, `teamName` y `deptName`, respectivamente.

Aunque todos éstos son nombres pueden no ser comparables debido a diferencias en longitud. Otras uniones pueden involucrar se agrupen conjuntos de entidades con claves radicalmente distintas. La solución es crear una clave subrogada que será la clave primaria de la unión. Será una clave externa en las tablas para cada una de las superclases. Para el ejemplo `Sponsor` cree una clave subrogada para patrocinador, que se llamará `sponsorId`, e inserte dicho atributo como una clave externa en las tablas para `Club`, `Team` y `Department`. Agregue también `SponsorType` para indicar si el patrocinador es un club, equipo o departamento. El esquema será:

```
Sponsor(sponsorId, sponsorType)
Club(clubName, president, numberOfMembers, sponsorId)
Team(teamName, coach, sport, sponsorId)
Department(deptName, office, sponsorId)
```

Si las claves primarias de las superclases son las mismas, no es necesario crear una clave subrogada. La tabla para la unión consistirá en el campo clave primaria común y un campo tipo. El campo clave primaria común también funcionará como una clave externa para las tablas superclase. Para el ejemplo `CampusWideEvent` que se muestra en la figura 7.4(b), el esquema será:

```
CampusWideEvent(eventName, eventType)
Concert(eventName, date, time, performer)
Fair(eventName, date, time, theme, charity)
```

7.7 Extensión del modelo relacional

El modelo relacional se diseñó para representar datos que consiste en atributos de un solo valor de una pequeña colección de tipos de datos, relacionados en formas relativamente simples, por lo general utilizadas en procesamiento de datos tradicional. Como se discute en la sección 7.1, el modelo carece de algunas características necesarias para representar los tipos y relaciones más complejos necesarios para aplicaciones avanzadas. La industria respondió a este desafío inicialmente al proponer un modelo de datos por completo nuevo, el modelo de datos objeto, que se discute en el capítulo 8. Los proveedores de sistemas de gestión de bases de datos relacionales, incluidos Oracle, IBM, Sybase, Informix y Microsoft han satisfecho esta competencia al capturar algunas de las características del modelo de datos objeto y extendieron sus productos relacionales para incorporarlos. El estándar SQL:1999 también extendió el lenguaje SQL para usar estas nuevas características. El modelo objeto-relacional se puede ver como un compromiso entre el modelo estrictamente relacional y el modelo orientado a objetos.

Algunas de las características adicionales necesarias son:

- Tipos de datos fundamentalmente más ricos, incluidos tipos para multimedia: texto, imágenes, video, audio, etcétera
- Tipos de colección que pueden contener atributos múltiples
- Tipos de datos definidos por el usuario, incluidas operaciones escritas por el usuario para manipularlos
- Representación de jerarquías de clase, con herencia de estructuras de datos y métodos
- Tipos referencia o puntero para objetos, a fin de permitirles referirse a objetos grandes como archivos multimedia que se almacenan en cualquier parte

7.7.1 Nuevos tipos de datos fundamentales

Para abordar el primer requisito, SQL:1999 agregó dos nuevos tipos de datos fundamentales, LARGE OBJECT (LOB) (objeto grande) y BOOLEAN (booleano), así como dos tipos de constructores, ARRAY (arreglo) y ROW (fila), para el estándar SQL.

El tipo **LOB** se puede usar para almacenar texto, audio, video y otros objetos multimedia. Además de imágenes y clips de audio o video, un atributo como la firma de un estudiante puede almacenarse en formato LOB. El tipo LOB tiene variantes **BLOB** (BINARY LARGE OBJECT, objeto binario grande) y **CLOB** (CHARACTER LARGE OBJECT, objeto carácter grande). Los tipos LOB tienen funciones muy restringidas, como operaciones subcadena. Permiten comparaciones sólo para igualdad o desigualdad, y no se pueden usar para ordenar o en cláusulas GROUP BY u ORDER BY. Dado que son archivos grandes, los LOB normalmente se almacenan y rara vez se recuperan de nuevo. Se pueden manipular usando un **localizador LOB**, un subrogado binario generado por el sistema para el valor real. Si quiere agregar la firma de un estudiante al registro `Student`, agregaría el atributo `signature` en el comando CREATE TABLE para la tabla `Student`, con la declaración de atributo:

```
signature CLOB REF IS sigid SYSTEM GENERATED,
```

El `sigid` es un puntero al archivo que contiene una imagen de la firma. Su uso se discutirá en la sección 7.7.5.

El tipo de datos **BOOLEAN** en SQL:1999 se puede usar para atributos con posibles valores de verdadero, falso o desconocido. En la tabla para estudiantes se podrían agregar un atri-

buto booleano, `matriculated`, para indicar si el estudiante está matriculado o no, con una declaración como la siguiente dentro del comando CREATE TABLE:

```
matriculated BOOLEAN DEFAULT false,
```

7.7.2 Tipos de colección

SQL:1999 incluye dos tipos de colección estructurados, **array** y **row**. La versión actual no soporta algunos tipos de colección no estructurados, aunque se espera que futuras versiones las incluirán. Ambos conjuntos (colecciones no ordenadas de elementos distintos) y multiconjuntos (colecciones no ordenadas que permiten repetición de elementos) se consideraron para el estándar, pero su inclusión se pospuso.

ARRAY[n] es un tipo estructurado que permite el almacenamiento como un solo atributo de una colección ordenada de valores del mismo tipo de datos base. Se debe especificar el tipo base para los n valores a almacenar en el array, usando la forma:

nombre-atributo tipo_base ARRAY [*n*]

Por ejemplo, si quiere permitir que los estudiantes tengan doble especialidad, podría declarar el atributo `majors` en el comando CREATE TABLE como VARCHAR (10) ARRAY[2]. Con el resto del comando que aparece como se hizo en la figura 6.2.

```
CREATE TABLE Stu (
   stuId         CHAR(6),
   lastName      CHAR(20) NOT NULL,
   firstName     CHAR(20) NOT NULL,
   matriculated  BOOLEAN DEFAULT false;
   majors        VARCHAR(10) ARRAY[2],
   credits       SMALLINT DEFAULT 0,
CONSTRAINT Student_stuId_pk PRIMARY KEY (stuId),
CONSTRAINT Student_credits_cc CHECK ((credits>=0) AND (credits < 150)));
```

La notación estándar de corchetes se usa en SQL:1999 para referirse a elementos individuales de un array. Por ejemplo, si supone que la indexación comienza en 1 puede acceder a la primera especialidad mencionada para un estudiante mediante:

```
SELECT Stu.majors[1]
FROM Stu
WHERE stuId = 'S999';
```

Para insertar un registro en una tabla con un tipo array, utilice el constructor arreglo (ARRAY), y mencione los valores a insertar en el array, como en:

```
INSERT INTO Stu VALUES ('S555', 'Quirk','Sean', true,
ARRAY ['French','Psychology'], 30);
```

En SQL:1999 no se soportan arrays multidimensionales ni arrays de arrays.

La declaración de los componentes del **tipo ROW** consiste de una secuencia de pares de nombres de campo y tipos de datos, como la lista de atributos y sus tipos de datos en una definición de tabla tradicional. De hecho, puede usar la declaración del tipo ROW para crear un tipo de datos y luego usar el tipo para crear una tabla, como se ilustra a continuación:

```
CREATE ROW TYPE team_row_type(
   teamName    CHAR(20),
   coach       CHAR(30),
   sport       CHAR(15));

CREATE TABLE Team OF TYPE team_row_type;
```

Todos los tipos que se definen usando constructores tipo de SQL tienen métodos internos que están disponibles para los usuarios, como se vio ilustrado para los arrays. Para el tipo ROW se puede hacer referencia a un campo individual usando la notación punto. Por ejemplo, para la tabla `Team` se puede referir al entrenador de un registro particular con el uso de la notación `Team.coach`, como en

```
SELECT Team.coach
FROM Team
WHERE Team.sport= 'soccer';
```

Un atributo de una tabla puede él mismo tener un tipo row, lo que permite tablas anidadas. Por ejemplo, puede tener un atributo de una tabla `NewStu` para representar un equipo al que pertenezca el estudiante, del modo siguiente:

```
CREATE TABLE NewStu (
   stuId     CHAR(6),
   . . .
   team      team_row_type,
   . . .);
```

El constructor ROW puede usar cualquier tipo de datos para sus campos, incluidos ROW y ARRAY. Por ejemplo, podría poner varios equipos en el registro del estudiante al crear la tabla con un array del tipo row team, al que se llamará `teams`, del modo siguiente:

```
CREATE TABLE NewStu2 (
   stuId          CHAR(6),
   lastName       CHAR(20) NOT NULL,
   firstName      CHAR(20) NOT NULL,
   matriculated   BOOLEAN DEFAULT false;
   teams          team_row_type ARRAY[3],
   majors         CHAR(10) ARRAY[2],
   credits        SMALLINT DEFAULT 0,
CONSTRAINT Student_stuId_pk PRIMARY KEY (stuId),
CONSTRAINT Student_credits_cc CHECK ((credits>=0) AND (credits < 150)));
```

Ahora cada fila de la tabla `NewStu2` contiene una tabla de hasta tres equipos (con sus nombres, entrenadores y deportes) a los que pertenece el estudiante, así que tiene tablas anidadas. Dado que las filas de una tabla ahora se pueden definir como un tipo de datos, puede pasar filas como parámetros a módulos (procedimientos o funciones), o devolver filas de funciones.

Si tiene una fila anidada en una tabla, como en la tabla `NewStu2` descrita anteriormente, puede referirse a los atributos individuales usando notación punto anidada, como en:

```
SELECT NewStu2.teams.coach
FROM NewStu2
WHERE stuId = ''S999';
```

Si el estudiante pertenece a muchos equipos, esto dará muchos entrenadores. Si sólo quiere el entrenador del primer equipo habría usado `NewStu2.teams[1].coach` en la línea SELECT. Puede usar la misma notación para realizar las otras operaciones DML en los atributos row. Para insertar un nuevo registro `NewStu2` use constructores, como se muestra en este ejemplo:

```
INSERT INTO NewStu2 VALUES('S999', 'Smith', 'Michael', true, ARRAY[ROW('Angels', 'Jones',
'soccer'), ROW('Devils', 'Chin', 'swimming'), ROW('Tigers', 'Walters', 'basketball')],
ARRAY['Math', 'Biology'], 60);
```

7.7.3 Tipos de datos definidos por el usuario (UDT)

SQL:1999 también permite un tipo de datos definido por el usuario, **DISTINCT**, a construir a partir de un tipo base. Por ejemplo, si escribe,

```
CREATE DISTINCT TYPE studentAgeType AS INTEGER;
```

y también

```
CREATE DISTINCT TYPE numberOfCreditsType AS INTEGER;
```

entonces, aun cuando ambos sean enteros, no puede comparar un atributo `studentAgeType` con un atributo `numberOfCreditsType`.

SQL:1999 también permite a los usuarios crear **tipos de datos estructurados**, que pueden tener muchos atributos. Los atributos pueden ser cualquier tipo SQL, incluidos los tipos internos tradicionales, los tipos LOB, tipos ARRAY o ROW, u otros tipos estructurados, lo que permite anidado de tipos estructurados. Por ende, un UDT estructurado puede ser o el tipo usado para un atributo, o el tipo usado para una tabla. Los atributos pueden ser ítems almacenados o bien ser **virtuales**, lo que significa que se derivan usando una función definida por el tipo. Los usuarios pueden elegir hacer el nuevo tipo **instanciable**, que permite la creación de instancias del tipo. Si el tipo es instanciable, SQL proporciona una **función constructor** por defecto para permitir la creación de nuevas instancias del tipo. Este constructor siempre tiene el mismo nombre y tipo de datos que el tipo de datos definido por el usuario, de modo que se le puede invocar con el uso del nombre del tipo. El constructor por defecto no toma parámetros y simplemente asigna valores por defecto a los atributos instancia. Los usuarios también pueden crear sus propios constructores cuando definan el nuevo tipo. Los tipos definidos por el usuario también pueden ser **no instanciables**, lo que significa que no se puede crear ninguna instancia de dicho tipo.

En el capítulo 6 se vio cuán útiles son las funciones internas SQL, como SUM y COUNT, para una variedad de consultas. Estas funciones se pueden usar con muchos tipos de datos diferentes. Sin embargo, cuando los usuarios definen sus propios tipos de datos, las funciones internas no se definen por dichos tipos, y sólo algunas operaciones muy básicas. Por tanto, los usuarios también deben definir operaciones en los nuevos tipos que crean y construir sobre las operaciones básicas definidas por los tipos fuente. La definición de un nuevo tipo incluye una lista de sus **atributos** y sus **métodos**, que son rutinas que se definen para el tipo. Los métodos se definen únicamente para un solo tipo definido por el usuario, y no se aplican a otros tipos que crea el usuario. Los tipos estructurados se pueden manipular usando operaciones definidas por el usuario como métodos, funciones y procedimientos. Para proporcionar comparación de objetos del mismo tipo definido por el usuario, éste bien puede definir relaciones de igualdad y ordenamiento para el tipo. SQL:1999 proporciona algunas funciones automáticas para tipos definidos por el usuario. Además del constructor por defecto descrito anteriormente, existen métodos **observador** automáticos (en ocasiones llamados GET) que devuelven los valores de atributos, y métodos **mutador** (en ocasiones llamados SET) que permiten la asignación de valores a los atributos. Aunque el usuario puede invalidar estos métodos observador y mutador al redefinirlos, no se pueden sobrecargar, a diferencia de los correspondientes métodos en algunos lenguajes orientados a objeto.

Como ejemplo de una definición de tipo estructurada se definirá un nuevo `StudentType` del modo siguiente:

```
CREATE TYPE StudentType AS (
stuId          VARCHAR(6),
lastName       VARCHAR(15),
firstName      VARCHAR(12),
```

```
advisorId      VARCHAR2(6),
credits        SMALLINT,
dateOfBirth    DATE)
METHOD         addCredits(smallint);
```

Opcionalmente puede agregar

```
INSTANTIABLE
NOT FINAL
```

Como ilustra este ejemplo, una definición de tipo contiene el nombre del tipo (aquí, el nuevo `StudentType`), una lista de atributos con sus tipos de datos dentro de paréntesis, y una lista de métodos (si hay alguno) para el tipo. Note que no se indica una clave primaria para el tipo, porque se define un tipo, no una tabla. Los atributos pueden ellos mismos ser UDT definidos con antelación. Para cada método se indican los tipos argument y return (si hay alguno). Cuando se invoca, un método toma una instancia del tipo como un parámetro implícito. Los métodos se invocan usando notación punto porque, como los atributos, son miembros del tipo. Por ejemplo, si `firstStudent` (primer estudiante) es una instancia del `StudentType` puede escribir:

```
firstStudent.addCredits(3);
```

El código para el método `addCredits( )` se escribe por separado. Dentro del método, al parámetro implícito (`firstStudent` en el ejemplo) se le refiere como ***self***. El código para el método `addCredits` puede ser:

```
CREATE METHOD addCredits (numberOfCredits smallint) FOR StudentType
   BEGIN
      SET self.credits = self.credits + numberOfCredits;
   END;
```

Con este nuevo `StudentType` puede crear una nueva versión de la tabla `Student`, que difiere ligeramente de la versión usada en capítulos anteriores, para propósitos de ilustración.

```
CREATE TABLE Student of StudentType
(CONSTRAINT Student_stuId_pk PRIMARY KEY(stuId),
CONSTRAINT Student_advisorId_fk FOREIGN KEY (advisorId) REFERENCES
Faculty (facId));
```

Cualquiera otra restricción, incluidas claves externas, se especifican para la tabla, dado que se aplican a la tabla, no al tipo. Puede crear tantas tablas como quiera usando el mismo UDT. Aunque puede insertar valores en la forma usual es preferible usar el constructor de tipo `StudentType`, como en:

```
INSERT INTO Student
VALUES(StudentType('S999', 'Fernandes', 'Luis', 'F101', 0,
'25-Jan-1985'));
```

Además del constructor `StudentType`, utilizado aquí en el INSERT, el nuevo tipo tiene el método `addCredits( )` que se escribió, así como los métodos observador y mutador internos para cada uno de sus atributos. Por ejemplo, `firstName( )` es un método que devuelve el valor del atributo `firstName` de una tupla, y `credits(30)` asigna un valor de 30 al atributo `credits`. Éstos se invocan para una instancia del tipo, usando notación punto, como en

```
firstStudent.firstName( );
```

o

```
secondStudent.credits(30);
```

7.7.4 Jerarquías tipo

Los tipos estructurados pueden participar en jerarquías tipo, en las que los subtipos heredan todos los atributos y operaciones de sus supertipos, pero pueden tener atributos y operaciones adicionales propios. La figura 7.7 muestra una jerarquía de clase bajo `Student`. Cuando se define un nuevo tipo, el usuario puede declararlo como FINAL, lo que significa que ningún subtipo se puede definir para él, o NOT FINAL, que permite la creación de subtipos. Como se discutió en la sección 7.7.3, si una clase se define como NOT INSTANTIABLE, no se pueden crear instancias de dicho tipo, pero si tiene subtipos se pueden crear instancias de los subtipos, siempre que los subtipos mismos se hagan instanciables. Con el tipo definido anteriormente, `StudentType`, que fue NOT FINAL, puede definir un subtipo, `UndergraduateType`, al escribir:

```
CREATE TYPE UndergraduateType UNDER StudentType AS (
   major varchar(10) ARRAY[2])
INSTANTIABLE,
NOT FINAL;
```

Puede definir una nueva función, `hasDoubleMajor`, para el subtipo,

```
CREATE FUNCTION hasDoubleMajor (u UndergraduateType) RETURNS BOOLEAN
   BEGIN
      IF (u.major[2] IS NOT NULL)
         THEN
            RETURN TRUE;
         ELSE
            RETURN FALSE;
      END IF;
   END;
```

Para crear una tabla para el subtipo, cree una subtabla bajo la tabla `Student` que se creó antes para el `StudentType`.

```
CREATE TABLE Undergraduate OF UndergraduateType UNDER Student;
```

Las tuplas pregrado heredan todos los atributos de las tuplas `Student` y tienen un atributo adicional, que es un array de especialidades. También heredan cualquier función, procedimiento o método definido por `StudentType`; sus propios métodos constructor, observador y mutador para especialidades; y su propia definición definida por el usuario, `hasDoubleMajor( )`. Note que `hasDoubleMajor( )` se podría definir como un método en vez de como una función, pero se eligió una función para propósitos de ilustración. Para invocar la función no se puede usar la forma *self*, `IF (thridStudent.hasDoubleMayor())THEN...,`, que se aplica a métodos. La forma *self* proporciona una instancia (*self*-thirdStudent, en este ejemplo) sobre el cual se realiza el método. En vez de ello, para una función, se debe proporcionar un parámetro real de `UndergraduateType`, como en:

```
IF (hasDoubleMajor(thirdStudent)) THEN . . .
```

Además de métodos y funciones, los tipos definidos por el usuario pueden tener **procedimientos**, que son similares a funciones, pero no tienen tipo return. Pueden tener parámetros de entrada, identificados por la palabra clave **IN**; parámetros de salida, identificados por **OUT**, y parámetros de dos vías, identificados mediante **IN/OUT**, antes del nombre del parámetro formal en el encabezado de procedimiento. Por ejemplo, podría crear un procedimiento para contar el número de especialidades que tiene un estudiante de pregrado. Aquí se elige un procedimiento en lugar de un método o función sólo para propósitos de ilustración.

```
CREATE PROCEDURE countMajors(IN u UndergraduteStudent, OUT
numberOfMajors SMALLINT)
BEGIN
   numberOfMajors =0;
   IF (u.majors[1] is NOT NULL) THEN numberOfMajors =
   numberOfMajors +1;
   IF (u.majors[2] IS NOT NULL) THEN numberOfMajors =
   numberOfMajors +1;
END;
```

Invocaría el procedimiento y el parámetro de salida utilizado en enunciados como

```
countMajors(fourthStudent, count);
IF (count = 2) THEN . . .
```

Podría continuar de esta forma y crear un tipo llamado `FreshmanType` bajo `UndergraduateType`. Para los estudiantes de primer año (freshmen), agregaría un atributo para `peerMentor` y cualesquier método, función o procedimiento que sólo se aplicara a dichos estudiantes. Dado que en la figura 7.7 no se muestran subtipos para `FreshmanType`, dicho tipo se haría INSTANTIABLE FINAL.

```
CREATE TYPE FreshmanType UNDER UndergraduateType AS (
   peerMentor varchar(25))
INSTANTIABLE,
FINAL;
```

Entonces podría crear una subtabla `Freshman` con sintaxis SQL:1999:

```
CREATE TABLE Freshmen OF FreshmanType UNDER Undergraduate;
```

Para estudiantes graduados, escribiría una definición para `GraduateStudentType` bajo `StudentType`, similar a la del `UndergraduateType`, esta vez especificando como atributo el programa de graduado en el que está inscrito el estudiante. También podría crear un subtipo, `TeachingAssistantType`, bajo `GraduateStudentType`. Con referencia a la figura 7.7 se ve que `TeachingAssistant` también aparece como un subtipo de `Faculty`. Sin embargo, en el estándar SQL:1999 no se permite herencia múltiple. Si elige convertirla en un subtipo de `GraduateStudentType`, no podría convertirla también en subtipo de `FacultyType`.

7.7.5 Tipos de referencia

En las bases de datos estrictamente relacionales, se vio que las claves externas se usan para establecer relaciones entre tablas. Para entidades relacionadas, el valor de un atributo de una tupla en una tabla es el valor de clave primaria de otra tupla, usualmente de una tabla diferente. Los sistemas objeto-relacional usan un mecanismo similar, que permite atributos de un tipo que en realidad son **referencias** o punteros a otro tipo. Cuando se usan tipos de referencia, el usuario debe encontrar explícitamente y establecer el valor de referencia cuando se inserta una tupla. Por ejemplo, en el modelo relacional, si supone que cada estudiante tiene un consejero que es miembro del personal docente, la correspondiente `facId` se identifica como clave externa cuando se crea la tabla `Student`. En el modelo objeto-relacional, la conexión se representa mediante un tipo referencia. Se supondrá que anteriormente se definió `FacultyType` y que dicho tipo se usó para crear la tabla `Faculty`. Ahora se creará una nueva `StudentType` con una referencia, `aId`, para `FacultyType`. Se usa la forma "CREATE OR REPLACE" para manejar la posibilidad de que el tipo ya exista.

```
CREATE OR REPLACE TYPE StudentType AS (
(stuId       VARCHAR(6),
lastName     VARCHAR(15),
firstName    VARCHAR(12),
aId          REF(FacultyType) SCOPE Faculty,
credits      SMALLINT,
dateOfBirth  DATE)
METHOD       addCredits(smallint));
```

Entonces puede crear una nueva tabla `Student`:

```
CREATE TABLE Student OF StudentType;
```

El atributo referencia, `aId`, que es una referencia a un `FacultyType`, se refiere a tuplas en una tabla existente, `Faculty`, como se indica mediante la especificación scope (ámbito) para ella. Note que la referencia especifica el tipo de objeto al que se alude, mientras que el scope especifica la tabla que contiene las tuplas de dicho tipo. Se usarán consultas para establecer los valores de referencia. SQL:1999 requiere que una tabla que tendrá una referencia a ella contenga un atributo para el identificador de cada tupla, similar a una clave subrogada. Por tanto, cuando se crea la tabla `Faculty` debe agregarse el atributo referencia, cuyo valor puede generar el sistema,

```
CREATE TABLE Faculty of FacultyType
REF IS aid SYSTEM GENERATED;
```

El sistema genera un valor para `aid` cuando se inserta una tupla `Faculty`, y puede recuperar dicho valor cuando sea necesario, en respuesta a una consulta. Note que el valor es uno interno que no aparecería como parte de la tupla a la que se refiere. Con sintaxis SQL: 1999, cuando quiera insertar una tupla `Student` debe obtener el valor de `aid` del consejero, con una consulta. Primero inserte una tupla `Student` con una referencia nula para el `aid`.

```
INSERT INTO Student VALUES('S555', 'Hughes','John', null, 0, '04-Jan-1988');
```

Luego establezca la referencia mediante una actualización de dicha tupla, con una subconsulta sobre la tabla `Faculty` para encontrar el valor `aid` correcto.

```
UPDATE Student
SET aid = (SELECT f.aid
   FROM Faculty AS f
   WHERE facId = 'F101')
WHERE f.stuId ='S555';
```

Para encontrar la tupla referenciada en una consulta tiene que quitar la referencia de `aid`. La forma `DEREF(aid)` se puede usar para encontrar toda la tupla referenciada. Un ejemplo es:

```
SELECT s.lastName, s.firstName, DEREF(s.aid), s.dateOfBirth
FROM Student s
WHERE s.lastName ='Smith';
```

Esta consulta devuelve el apellido y nombre de cada estudiante llamado Smith, junto con el `facId`, `name`, `department` y `rank` del consejero del estudiante (si supone que ésos son los atributos de `FacultyType`), y la fecha de nacimiento del estudiante. En lugar del operador DEREF, puede usar el operador `->` para recuperar el valor de un atributo de la tupla referenciada. Por ejemplo,

```
aid -> lastName
```

recupera el valor del atributo `lastName` de la tupla `Faculty` referenciada.

Aunque el valor `aid` funcione como una clave externa, existen algunas diferencias importantes. Las referencias no soportan las mismas restricciones estrictas que las claves externas. Si se borra el objeto referenciado puede ocurrir un gran problema, porque uno puede quedar con **punteros colgantes** (dangling, punteros cuyos objetos referenciados ya no existen) en la base de datos. Algunos sistemas permiten especificar **REFERENCES ARE CHECKED** (comprobación de referencias) y agregar cláusulas ON DELETE para referencias. Además de referencias generadas por el sistema, el estándar permite referencias generadas por el usuario (para las cuales el usuario debe proporcionar y rastrear los identificadores únicos) y referencias derivadas (para las que el usuario debe especificar el atributo del cual el atributo referencia deriva su valor).

En la sección 7.7.1 se vio que los tipos LOB tienen un **localizador LOB**, que también es una referencia. Las firmas de los estudiantes se declararon usando un tipo referencia.

```
signature CLOB REF IS sigid SYSTEM GENERATED;
```

Aquí, el `sigid` es una referencia al archivo que contiene una imagen de la firma del estudiante. La base de datos almacena la referencia en el registro `Student`. Puede usar un enunciado SELECT para encontrar el valor de esta referencia, como en:

```
SELECT s.sigid
FROM Student AS s
WHERE s.stuId ='S999';
```

Esto proporciona el valor de la referencia en sí. Para recuperar realmente una firma, en el raro caso donde se requiera, tendría que quitar la referencia a dicho puntero dentro de un comando SELECT. Entonces un programa de aplicación o métodos, funciones o procedimientos escritos por el usuario manipularían los datos según se necesite. Para la firma, puede tener un método para desplegarla sobre una pantalla. Si el objeto LOB fuese un archivo de audio en lugar de una firma se puede definir un método para reproducir una porción del archivo, con puntos de inicio y fin identificados mediante parámetros por el método. De igual modo, si el objeto LOB fuese un videoclip, podría localizar el clip al quitar la referencia a un localizador LOB y manipular una porción del video usando métodos.

7.8 Conversión de un diagrama EE-R a un modelo de base de datos objeto-relacional

El modelo objeto-relacional proporciona mucho más flexibilidad en el mapeo de un diagrama EE-R de lo que lo hace el modelo estrictamente relacional. Como se ilustra en los ejemplos puede usar nuevos tipos de datos, tipos de colección, tipos estructurados definidos por el usuario, herencia y tipos referencia en la creación de las tablas. Para incorporar las extensiones se modifica el algoritmo de mapeo que se presentó en la sección 7.6.2.

Para conjuntos de entidades y relaciones que no son parte de generalizaciones o uniones, el mapeo puede proceder en una forma similar al modelo relacional. Los conjuntos de entidades se pueden mapear a tablas, o con sintaxis SQL2 o con creación de un tipo y luego usando dicho tipo para la creación de tablas. Para atributos compuestos, puede definir un tipo estructurado para el compuesto, con los componentes como atributos del tipo. Los atributos multivaluados se pueden expresar como tipos colección, como ARRAY. Para representación de relaciones puede usar tipos referencia en lugar del mecanismo de clave externa. Para jerarquías de especialización, puede definir tipos y subtipos según requiera. Entonces se pueden crear tablas y subtablas que correspondan a los tipos. Sin embargo, dado que ningún subtipo puede tener herencia múltiple, el diseñador debe elegir entre los supertipos potenciales un subtipo compartido. La relación del subtipo con el otro supertipo debe expresarse usando referencias o claves externas. No hay representación directa de uniones

(categorías) en el modelo objeto-relacional, pero puede usar el mismo método como se hizo para el modelo relacional. Se crea una tabla para la unión en sí, y tablas individuales para cada una de las superclases, con claves externas o referencias para conectarlas.

7.9 Representación de objetos en Oracle

La implementación de Oracle de las características objeto-relacional difieren del estándar SQL:1999. Oracle proporciona tipos de datos BLOB y CLOB que pueden soportar hasta cuatro gigabytes de datos, pero no soportan booleano como un tipo de datos interno. Proporciona **tipos array** y **tipos objeto definidos por el usuario**. Los tipos array se declaran con la siguiente forma:

CREATE OR REPLACE TYPE *nombre_tipo_array* AS VARRAY (*tamaño*) OF *tipo_base;*

El tipo base de un array puede ser cualquier tipo de datos válido excepto LOB, incluidos los definidos por el usuario, siempre que ninguno de sus atributos sea LOB. Oracle permite arrays de arrays. Note el uso de paréntesis en lugar de corchetes para el tamaño. Un ejemplo de un tipo array es:

```
CREATE OR REPLACE TYPE MajorsType AS VARRAY(2) OF VARCHAR2(10);
```

El tipo array se puede usar entonces como un tipo atributo para una tabla, como un componente de un tipo de datos definido por el usuario, o como un tipo para un parámetro o variable en PL/SQL. Oracle soporta UDT estructurados en la forma de **tipos objeto**. Un tipo objeto Oracle es un tipo estructurado con atributos, que puede ser cualquier tipo válido, incluidos arrays u otros objetos, y métodos, como se vio en el estándar SQL:1999. La definición de objeto tiene esta forma:

CREATE OR REPLACE TYPE *nombre_tipo_objeto* AS OBJECT(
definiciones atributo
definiciones método);

Por ejemplo:

```
CREATE OR REPLACE TYPE StudentType AS OBJECT (
stuId       VARCHAR2(5),
lastName    VARCHAR2(15),
firstName   VARCHAR2(12),
majors      MajorsType;
credits     NUMBER (3),
dateOfBirth DATE,
MEMBER FUNCTION findAge RETURN INTEGER)
INSTANTIABLE
NOT FINAL;
```

Los tipos que se crearon como NOT INSTANTIABLE pueden cambiarse más tarde mediante este enunciado:

ALTER TYPE *nombre_tipo* INSTANTIABLE;

Sin embargo, alterar un tipo de INSTANTIABLE a NOT INSTANTIABLE es posible sólo si el tipo no se usó para crear columnas, tablas o cualquier otro ítem al que se haga referencia.

El tipo objeto se puede usar como un atributo de otro tipo estructurado (como `MajorsType` se usó para `StudentType`), como el tipo base de un array o como el único tipo en una tabla. A tales tablas se les refiere como **tablas objeto**. Un ejemplo de una tabla objeto es la tabla `Student` que ahora se crea a partir del `StudentType` recién definido:

```
CREATE TABLE Student of StudentType (
CONSTRAINT Student_stuId_pk PRIMARY KEY(stuId),
CONSTRAINT Student_credits_cc CHECK ((credits>=0) AND
(credits < 150)));
```

Las filas de una tabla objeto se describen como **objetos fila**.

Los métodos para un tipo se pueden clasificar como **métodos miembro** o **métodos estáticos**. Los métodos miembro tienen el parámetro implícito, *self*, como se describió en la sección 7.7.3. Se pueden implementar usando PL/SQL, Java o C. Por ejemplo, la función miembro `findAge` (encontrar edad) se puede codificar del modo siguiente:

```
CREATE OR REPLACE TYPE BODY StudentType AS
MEMBER FUNCTION findAge RETURN INTEGER IS
BEGIN
   RETURN TRUNC((SYSDATE - dateOfBirth)/12);
END;
```

Los **métodos miembro** tienen el *parámetro self* implícito y pueden ser o **funciones**, que tienen un solo tipo return, o **procedimientos**, que no tienen return pero pueden tener parámetros de entrada y/o salida. También puede crear **métodos estáticos** o no miembros, que se definen por el tipo, y que no tienen un parámetro *self* implícito. El sistema proporciona un constructor por defecto para el tipo, pero puede escribir constructores propios o funciones o procedimientos adicionales usando métodos estáticos. Se invocan con el nombre del tipo, seguido por un punto, luego el nombre del método estático. Como en el estándar SQL, el constructor automático de Oracle para cada tipo se invoca al escribir el nombre del tipo. Sus parámetros tienen los nombres y tipos de los atributos. Por ejemplo, para insertar un nuevo registro **`Student`**, invoque el constructor para `StudentType` y `MajorsType` al escribir:

```
INSERT INTO Student VALUES StudentType ('S999', 'Smith', 'John',
MajorsType('Math','Accounting'), 0,'05-Jan-1980');
```

Los tipos definidos por el usuario no tienen algún ordenamiento automático de sus elementos, pero los usuarios pueden imponer un orden al identificar o un **método mapa** o un **método orden** para el tipo. Un **método mapa** ordena a Oracle aplicar el método de comparación que normalmente utiliza para uno de sus tipos internos (por lo general, uno de los atributos) como el método de comparación para todo el tipo. Por ejemplo, si la función miembro se hubiese definido de la siguiente forma,

```
MAP MEMBER FUNCTION getstuId( ) RETURN VARCHAR2
```

y luego esta función se codifica para devolver el valor de `stuId`, del modo siguiente,

```
CREATE OR REPLACE TYPE BODY StudentType AS
MEMBER FUNCTION getstuId( ) RETURN VARCHAR2 IS
BEGIN
   RETURN stuId;
END;
```

entonces Oracle usaría su método interno de comparación de valores VARCHAR2 aplicado a los valores `stuId` siempre que compare objetos `StudentType`, pues este método se mencionó como MAP. En vez de ello, los usuarios pueden escribir su propio método de comparación, llamado **método orden**, para decirle a Oracle cómo comparar dos objetos de un tipo definido por el usuario. Si elige escribir una función `compareAge( )` y hacerla método orden al declarar

```
ORDER MEMBER FUNCTION compareAge(s StudentType) RETURN INTEGER
```

podría escribir código para la función del modo siguiente:

```
CREATE OR REPLACE TYPE BODY StudentType AS
MEMBER FUNCTION compareAge(s StudentType) RETURN INTEGER IS
BEGIN
   IF (self.dateOfBirth < s.dateOfBirth) THEN RETURN 1
   ELSE IF(self.dateOfBirth > s.dateOfBirth)THEN RETURN -1
   ELSE RETURN 0;
END;
```

Entonces Oracle siempre compararía dos objetos `StudentType` sobre la base de la edad, no de `stuId`. Un UDT puede tener cuando mucho un método mapa o método orden para todo el tipo. Un subtipo puede crear su propio método mapa si su supertipo raíz tiene uno, pero no su propio método orden.

Cuando se crean tablas sólo con un tipo objeto se llaman tablas objeto, y sus filas se llaman objetos fila, como se describió anteriormente. A cada objeto fila en una tabla objeto Oracle se le da un identificador único, llamado identificador de objeto lógico (**OID**, por sus siglas en inglés). El identificador puede o ser generado por el sistema (el defecto) o el usuario puede especificar que la clave primaria se usará como OID. Un OID generado por el sistema es una clave de 16 bytes al que el usuario no tiene acceso, aunque se puede usar para encontrar y leer objetos. Oracle crea un índice en la columna OID de las tablas objeto. En un entorno no duplicado no distribuido, el usuario puede elegir la clave primaria como el OID. La selección se especifica en el comando CREATE TABLE, como en:

```
CREATE TABLE Student of StudentType (
CONSTRAINT Student_stuId_pk PRIMARY KEY(stuId),
CONSTRAINT Student_credits_cc CHECK ((credits>=0) AND (credits < 150))
OBJECT IDENTIFIER PRIMARY KEY);
```

La última línea se podría sustituir con OBJECT IDENTIFIER SYSTEM GENERATED (identificador de objeto generado por sistema).

Oracle también usa OID para ayudar a construir un **tipo referencia** para referirse a objetos fila. Puede definir un atributo REF, con una cláusula opcional SCOPE que restrinja la columna a la tabla especificada como el ámbito, como se vio para SQL:1999. La cláusula scope se requiere si el identificador de objeto del objeto referenciado es la clave primaria. Si no se especifica un ámbito, la referencia puede ser a cualquier objeto fila del tipo especificado, sin importar si aparece. Dado que los punteros colgantes pueden resultar del borrado o actualización de objetos fila, Oracle proporciona un mecanismo de restricción referencial, RESTRICT REFERENCES, para objetos referencia. También existe un método prueba, IS DANGLING, para determinar si el puntero es colgante, y un operador para quitar referencias DEREF para referencias.

Oracle también permite **tablas anidadas**. Una tabla anidada se crea a partir de un solo tipo interno u objeto tipo, y luego se coloca en otra tabla. Por ejemplo, podría crear una lista de contactos anidada para contener nombres y números telefónicos de los padres, cónyuge, tutor u otros contactos de un estudiante, como:

```
CREATE OR REPLACE TYPE ContactType AS OBJECT(
   lastName       VARCHAR2(15),
   firstName      VARCHAR2(12),
   relationship   VARCHAR2(15),
   phoneNumber    VARCHAR2(10));
CREATE OR REPLACE TYPE ContactListType AS TABLE OF ContactType;
```

Entonces, cuando se define `StudentType` se agrega como un atributo

```
contactList contactListType;
```

inmediatamente después del atributo `dateOfBirth`, y cuando se crea la tabla `Student`, se especifica:

```
CREATE TABLE Student OF StudentType(
CONSTRAINT Student_stuId_pk PRIMARY KEY stuId)
OBJECT IDENTIFIER PRIMARY KEY
NESTED TABLE contactList STORE AS contactListStorageTable(
   (PRIMARY KEY (NESTED_TABLE_ID, lastName, relationship, phoneNumber))
ORGANIZATION INDEX COMPRESS);
```

Note que debe identificar un nombre para una tabla de almacenamiento para la tabla anidada, que aquí se llamará `contactListStorageTable`. Ésta es una tabla separada de `Student`. La clave primaria de la tabla anidada se especifica para incluir `NESTED_TABLE_ID`, que es una columna oculta creada por Oracle que asocia las filas de esta tabla de contactos con la correspondiente fila `Student`. Todas las filas para contactos de un estudiante particular tienen el mismo `NESTED_TABLE_ID`, de modo que se agrega `lastName`, `relationship` y `phoneNumber` a la clave primaria para garantizar exclusividad. La organización indexada comprimida se especifica de modo que las filas que pertenecen al mismo estudiante se agruparán.

Para insertar datos en la subtabla, primero cree una fila para la tabla propietaria, `Student`, incluida una llamada sin parámetro al constructor para la subtabla, del modo siguiente:

```
INSERT INTO Student VALUES StudentType ('S999', 'Smith', 'John', MajorsType('Math',
'Accounting'), 0, '05-Jan-1980', ContactListType( ));
```

El constructor crea una instancia vacía en la subtabla, en la que ahora se pueden insertar datos usando un enunciado INSERT INTO TABLE del modo siguiente:

```
INSERT INTO TABLE(SELECT s.contactList
   FROM Student s
   WHERE s.stuId = 'S999')
VALUES('Smith', 'Marjorie', 'mother', '2017771234');
```

Note que aquí se creó un alias en la línea FROM, y luego se le usó para hacer referencia a una columna, que es la práctica Oracle recomendada para todos los enunciados `UPDATE`, `DELETE` y `SELECT`, y subconsultas. Puede continuar insertando tantos contactos como desee para este estudiante, o crear otras tuplas `Student` y sus listas de contactos.

Las tablas anidadas se pueden desanidar usando la palabra clave **TABLE** para aplanar la tabla como en:

```
SELECT s.lastName, s.firstName, c.*
FROM Student s, TABLE(s.contactList) c
WHERE s.credits <= 60
ORDER BY VALUE(s);
```

El resultado será un multiconjunto con una lista aplanada donde los nombres de estudiante se repetirán para cada uno de sus contactos (un contacto por línea). El orden será cualquier orden que se especifique cuando se cree un método mapa o un método orden para `StudentType`.

Oracle soporta representación de jerarquías de objeto. Los subtipos se pueden crear a partir de tipos definidos por el usuario usando casi la misma sintaxis que SQL:1999. Por ejemplo, suponga que se creó el `StudentType` como:

```
CREATE OR REPLACE TYPE StudentType AS OBJECT (
stuId       VARCHAR2(5),
lastName    VARCHAR2(15),
firstName   VARCHAR2(12),
credits     NUMBER(3),
```

```
dateOfBirth  DATE,
MEMBER FUNCTION findAge RETURN INTEGER)
INSTANTIABLE
NOT FINAL;
```

Para crear `UndergraduateType` como un subtipo `StudentType`, escriba

```
CREATE OR REPLACE TYPE UndergraduateType UNDER StudentType (
majors VARCHAR2(10) VARRAY[2])
INSTANTIABLE,
NOT FINAL;
```

Puede avanzar y crear `FreshmanType` como un subtipo de `UndergraduateType` al escribir

```
CREATE OR REPLACE TYPE FreshmanType UNDER StudentType (
   peerMentor VARCHAR2(25) )
INSTANTIABLE,
FINAL;
```

El subtipo hereda todos los atributos y métodos de su supertipo y puede agregar sus propios atributos y métodos, incluso sobrecargar o invalidar algunos métodos del supertipo. En Oracle, los subtipos se pueden sustituir por supertipos como columnas o filas objeto, en tablas o vistas objeto. Por tanto, puede insertar registros Undergraduate y Freshman en la tabla `Student`. Si crea la tabla `Student`,

```
CREATE TABLE Student OF StudentType;
```

luego puede insertar registros Student, Undergraduate o Freshman en la tabla `Student` usando el constructor adecuado, del modo siguiente:

```
INSERT INTO Student VALUES
(StudentType('S444', 'Klein', 'Susan', 36, '14-Mar-1976'));

INSERT INTO Student VALUES
(UndergraduateType('S555', 'Logan', 'Randolph', 12, '23-May-1982', VARRAY('Spanish',
'French');

INSERT INTO Student VALUES
(FreshmanType('S777', 'Miller', 'Terrence', 30, '30-Jun-1985', VARRAY('Math',NULL), 'Tom
Smith')
```

Desde luego, cada tupla en la tabla es una tupla `Student`, pero algunas también pertenecen a otros subtipos. Toda tupla en una jerarquía tiene un **tipo más específico**, que es el subtipo "más cercano" al que pertenece, el subtipo usado cuando se crea.

Para recuperar datos de una tabla jerárquica puede usar la función **VALUE** (valor), que ordinariamente devuelve los valores atributo de una o más tuplas, incluidos todos los subtipos en el resultado. Por ejemplo, puede aplicar una consulta VALUE a todos los estudiantes, incluidos subtipos, mientras busca estudiantes cuyo apellido es Smith, al escribir:

```
SELECT VALUE(s)
FROM Student s
WHERE s.lastName = 'Smith';
```

Puede limitar la consulta al supertipo y excluir los subtipos al escribir la palabra clave **ONLY** (sólo), como en:

```
SELECT VALUE(s)
FROM ONLY(Student s)
WHERE s.lastName = 'Smith';
```

Esta consulta encontrará sólo aquellos estudiantes que no son subtipos undergraduate o freshmen, o cualquier otro subtipo de `Student`, si se crearon otros. Sólo selecciona aque-

llos cuyo subtipo más específico es `StudentType`. Para limitar la consulta a estudiantes de pregrado puede especificar que las instancias de tupla deban ser del tipo `UndergraduateType`, usando **IS OF (*tipo*)** (es de tipo) del modo siguiente:

```
SELECT VALUE(s)
FROM Student s
WHERE s.lastName = 'Smith' AND VALUE(s) IS OF
(UndergraduateType);
```

Los resultados incluyen todas las tuplas `UndergraduateType` que satisfagan la condición, incluidas tuplas `FreshmanType`. Para eliminar las tuplas Freshman podría escribir:

```
SELECT VALUE(s)
FROM Student s
WHERE s.lastName = 'Smith' AND VALUE(s) IS OF (ONLY
UndergraduateType);
```

La cláusula ONLY limita la devolución a tuplas cuyo tipo más específico sea `UndergraduateType`.

Si quiere recuperar columnas específicas puede citarlas en la línea SELECT, como es usual. Sin embargo, para columnas que sólo aparecen en un subtipo, no puede simplemente citar los nombres de columna en la línea SELECT, pues la columna no aparece en la declaración del tipo de la tabla citada en la línea FROM. Por ejemplo, `peerMentor` no aparece como columna en la tabla `Student`. Existe una función **TREAT( )** (tratar) que permite tratar cada tupla `Student` como si fuese una tupla Freshman para examinar el atributo `peerMentor`.

```
SELECT lastName, firstName, TREAT(VALUE(s) AS
FreshmanType).peerMentor
FROM Student s
WHERE VALUE(s) IS OF (FreshmanType);
```

Si un tipo o subtipo Oracle tiene un atributo REF, el operador DEREF puede usarse como en SQL estándar. Si un tipo o subtipo tiene una tabla anidada, también se puede usar la palabra clave TABLE, como se vio anteriormente, para desanidar la tabla. Las cláusulas ORDER BY y GROUP BY se pueden usar para consultas sobre subtipos tal como se usaron para otros tipos.

Oracle soporta otras características objeto-relacional además de las básicas descritas aquí. Incluso permite a los usuarios de las bases de datos Oracle estrictamente relacionales verlas como las objeto-relacional al usar un mecanismo llamado **vista objeto**. Una vista objeto se construye al definir un tipo objeto en el que cada atributo corresponde a una columna de una tabla relacional existente. La vista objeto se crea usando un enunciado SELECT, en la misma forma que se crearía una vista relacional, excepto que se debe especificar un identificador de objeto, para decir cuál atributo(s) se usará(n) para identificar cada fila de la vista. Este identificador, que por lo general es la clave primaria, se puede usar para crear atributos de referencia, si se desea. Para garantizar seguridad de actualizaciones en la vista objeto, los usuarios pueden escribir disparadores INSTEAD OF. Entonces la vista objeto se puede tratar como si fuese una tabla objeto. Si se tuviera una tabla `Faculty` meramente relacional, definida como

```
CREATE TABLE Faculty(
facId        NUMBER(4) PRIMARY KEY,
lastName     VARCHAR2(15),
firstName    VARCHAR2(10),
dept         VARCHAR2(10),
rank         VARCHAR2(10),
salary       NUMBER(8,2));
```

se podría definir un tipo, `FacultyType`, al escribir

```
CREATE OR REPLACE TYPE FacultyType (
facId        NUMBER(4),
lName        VARCHAR2(15),
fName        VARCHAR2(10),
department   VARCHAR2(10),
rank        VARCHAR2(10));
```

Podría crear una vista objeto, `FacultyView`, al escribir

```
CREATE VIEW FacultyView OF FacultyType WITH OBJECT
IDENTIFIER (facId) AS SELECT f.facId, f.lastName,
f.firstName, f.dept, f.rank
FROM Faculty f;
```

Entonces los usuarios pueden referirse a los atributos del objeto usando notación punto, escribir métodos para el tipo y usar otras características objeto-relacional como si fuese un tipo estructurado definido por el usuario.

7.10 Resumen del capítulo

El modelo E-R no es a nivel semántico suficientemente rico para representar los datos y relaciones complejos necesarios para aplicaciones avanzadas. El modelo EE-R es un modelo **E-R extendido** en el que se pueden representar **especialización**, **generalización**, **unión,** restricciones adicionales y otras abstracciones. **Especialización** significa descomponer un conjunto de entidades, o clase, en los varios tipos de subconjuntos que contiene. Su proceso inverso, **generalización**, significa considerar objetos de diferentes tipos, que tengan algunas características comunes, como subclases de una clase de nivel superior, llamada superclase. En un diagrama EE-R, generalización/especialización, también llamada una relación ***isa***, se representa mediante una línea con un **círculo de especialización** que conecta la clase de nivel superior con sus subclases. Un **símbolo de herencia** (símbolo subconjunto) aparece en la línea desde la subclase hasta el círculo. Si sólo existe una subclase, no se usa círculo, sino que el símbolo de subconjunto aparece en la línea que conecta la subclase a la superclase. Las subclases heredan los atributos y relaciones de sus superclases, pero pueden tener atributos adicionales o participar en relaciones adicionales. Las subclases pueden ser **desarticulación** (disjoint) o **traslapamiento** (overlapping), indicadas al colocar d u o en el círculo de especialización. La especialización puede ser **definida por predicado** (incluso **definida por atributo**) o **definida por el usuario**. Puede ser **parcial**, indicada mediante una sola línea, o **total**, indicada mediante una línea doble desde la superclase hasta el círculo de especialización. Una subclase puede ser **compartida**, que aparece como subclase de más de una superclase, y tener de ellas **herencia múltiple**.

Una subclase puede pertenecer a una **categoría** o **unión** de superclases. Una unión se muestra mediante un círculo con el símbolo de unión de conjuntos, al que la superclase se conecta mediante líneas. La subclase se conecta mediante una línea al círculo de unión. Estas subclases heredan los atributos y relaciones de sólo una de las superclases. Una categoría puede ser **parcial** o **total**, indicada mediante una línea sola o doble, respectivamente, desde la subclase al círculo de unión. En ocasiones, una categoría total se puede sustituir mediante una especialización. Para los diagramas E-R y EE-R, una notación alterna tanto para participación de relación como para cardinalidad es la notación (***mín..máx***), que indica los números mínimo y máximo de instancias de relación en las que participa una instancia de entidad.

Un diagrama EE-R se puede convertir en un modelo relacional en gran medida como un diagrama E-R, excepto por la jerarquía de generalización. Una jerarquía se puede convertir a tablas en tres formas diferentes. En la primera, hay una tabla para la superclase que contiene los atributos comunes, y tablas individuales para las subclases, y cada una contiene una columna o columnas para la clave de la superclase y columna para sus propios atributos. En el segundo método, no hay tabla para la superclase, sino que cada tabla de subclase contiene columnas para todos los atributos de la superclase además de columnas para sus propios atributos. En la tercera, una sola gran tabla contiene todos los atributos de la superclase y todas sus subclases. Puede haber un **campo tipo** para indicar a cuál subclase pertenece una entidad, o un **campo membresía** para cada especialización, en especial en el caso de subclases traslapantes. Las categorías se representan con la creación de una tabla para la subclase de la unión y tablas individuales para cada una de las superclases, usando su clave primaria común como una clave externa en la tabla subclase. Sin embargo, si las superclases tienen diferentes claves primarias, puede ser necesario crear una clave subrogada como la clave primaria de la tabla subclase unión, y convertirla en clave externa para cada una de las superclases.

El modelo relacional extendido tiene tipos de datos fundamentales adicionales, tipos de colección, tipos de datos definidos por el usuario (**UDT**), jerarquías de clase y tipos referencia. **SQL:1999** agregó tipos fundamentales **LOB** y **booleano**. Las colecciones incluyen tipos **array** (arreglo) y **row** (fila). Las tablas pueden estar anidadas. Los usuarios pueden definir tipos **distintos** y **tipos de datos estructurados**. Las definiciones de UDT incluyen especificación de atributos y métodos. Pueden o no ser **finales**, **instanciables** o no. Los métodos internos incluyen **constructores**, **observadores** y **mutadores**. Los **métodos miembro** adicionales, que pueden ser funciones o procedimientos, se pueden definir usando el parámetro ***self*** implícito. Las **funciones** devuelven sólo un tipo, mientras que los **procedimientos** pueden tener parámetros IN, OUT e IN/OUT, que permiten pasar de vuelta varios valores. Para la clase se pueden definir métodos **estáticos**, que no tienen el parámetro *self*. Se pueden crear tablas que usen el tipo como la única columna, o el tipo puede aparecer como una de muchas columnas. Si el tipo no es final se pueden crear subtipos que hereden todos los atributos y métodos, y puede agregar atributos y métodos adicionales para el subtipo. La jerarquía se puede extender al crear subtipos de subtipos. Sin embargo, no se permite la herencia múltiple. Bajo tablas que consistan en el supertipo se pueden crear subtablas que consistan en el subtipo.

Los tipos de **referencia** son punteros hacia otro tipo. Se pueden usar como atributos para representar relaciones, similares a claves externas. Es posible especificar un **scope** (ámbito) que identifique la tabla referenciada. Las referencias pueden ser **generadas por el sistema**, **generadas por el usuario** o **derivadas**. Un operador **DEREF** recupera la tupla a la que se hace referencia, o se puede usar el operador `->` para recuperar el valor de un atributo en la tupla referida por una referencia. Los **punteros colgantes** (dangling) pueden resultar cuando los objetos referenciados se borran, pero una cláusula **REFERENCES ARE CHECKED** proporciona alguna protección de la integridad. Un **localizador LOB** es una referencia a un objeto LOB.

Para convertir un diagrama EE-R a un modelo objeto-relacional puede modificar el mecanismo estándar de conversión relacional, al usar nuevos tipos de datos y tipos de colección, que son útiles para atributos multivaluados y compuestos. Si se desea las relaciones se pueden representar mediante tipos referencia en lugar de por claves externas. Las jerarquías se pueden representar usando supertipos y subtipos.

La **sintaxis Oracle** difiere un poco del estándar SQL:1999. Las UDT pueden tener un solo método, o un **método mapa** o un **método orden**, que define el orden de los miembros. A las **tablas objeto**, que consisten sólo en un UDT, se les dan **OID** únicos para sus **filas**. El OID puede ser generado por el sistema o puede ser la clave primaria, si es globalmente

única. Los OID se usan para crear referencias a objetos fila. Se pueden crear tablas anidadas y jerarquías de objeto. Es posible crear una sola tabla para contener todos los objetos en una jerarquía, y pueden usarse constructores para insertar instancias de todas las diferentes clases en la jerarquía. Cada tupla tiene un **tipo más específico** identificado por el constructor utilizado cuando se insertó. La función **VALUE** es útil para recuperar datos de una tabla jerárquica. La palabra clave **ONLY** y la cláusula **IS OF *tipo*** se pueden usar para restringir el tipo del resultado, y la palabra clave **TREAT** proporciona flexibilidad para recuperar subtipos.

Una **vista objeto** es un mecanismo para tratar una base de datos estrictamente relacional como si fuese una objeto-relacional. Proporciona a los usuarios una vista objeto-relacional.

Ejercicios

7.1 Defina los siguientes términos:

a. especialización
b. generalización
c. unión
d. superclase
e. subclase
f. atributo local
g. relación *isa*
h. relación local
i. restricción de desarticulación
j. restricción de completud
k. especialización definida por atributo
l. herencia múltiple
m. categoría total
n. LOB, BLOB
o. tipo de colección
p. UDT
q. método
r. tipo referencia

7.2 a. Suponga que el conjunto de entidades Empleados está especializado en Oficinistas, Vendedores y Administradores. Muestre cómo se representa la especialización en un diagrama E-R, construya los atributos que necesite y realice las suposiciones que requiera.

b. Para la especialización del inciso a) agregue relaciones que muestren:

i) Todos los empleados pueden tener un plan de pensión
ii) Los vendedores tienen clientes
iii) Los administradores tienen proyectos

c. Con la notación (mín, máx) agregue restricciones para las relaciones y establezca cualquier suposición adicional que necesite.

7.3 a. Convierta el diagrama EE-R que desarrolló para el ejercicio 7.2 en un modelo estrictamente relacional, y escriba el esquema.

b. Convierta el diagrama EE-R que desarrolló para el ejercicio 7.2 en un modelo objeto-relacional, y explique qué cambios haría al esquema relacional del inciso a).

7.4 Desarrolle un diagrama EE-R para la siguiente aplicación, que es una versión extendida del ejercicio 3.5.

Un grupo dental necesita mantener la información de pacientes, las visitas que hacen al consultorio, el trabajo que se debe realizar, los procedimientos realizados durante las visitas, los cargos y pagos por el tratamiento, y los suministros y servicios de laboratorio. Suponga que existen varios dentistas en el grupo. Los pacientes realizan muchas visitas y durante cada visita se les pueden practicar muchos servicios o procedimientos. Aunque los pacientes usualmente hacen citas para ver al mismo dentista, pueden ver a otro en caso de urgencia o por otras razones. La base de datos debe almacenar información acerca de los servicios y procedimientos, incluidos un código, descripción y cargos por cada uno. Algunos servicios pueden abarcar varias visitas. La oficina usa tres laboratorios dentales, uno para suministros regulares, uno para fabricación de dentaduras y uno para otros suministros y servicios.

7.5 a. Mapee el diagrama EE-R que desarrolló para el ejercicio 7.4 en un modelo estrictamente relacional.

b. Convierta el diagrama EE-R que desarrolló para el ejercicio 7.4 en un modelo objeto-relacional, y explique qué cambios haría al esquema relacional del inciso a).

7.6 Desarrolle un diagrama EE-R para la siguiente aplicación, que es una versión extendida del ejercicio 3.6.

Una firma de diseño de interiores quiere tener una base de datos para representar sus operaciones. Un cliente solicita que la firma realice servicios como decorar una casa nueva, redecorar habitaciones, localizar y comprar mobiliario, y cosas por el estilo. Los clientes pueden ser individuos privados o corporaciones. El cliente se reúne con uno de los decoradores de la firma, quien trabaja con el cliente hasta que el trabajo está completo. Los clientes pueden solicitar un decorador particular y también cambiar de decoradores para trabajos futuros. Para cada trabajo, la firma proporciona un estimado de la cantidad de tiempo y dinero requerido para completar el trabajo. Un trabajo puede incluir muchas actividades, como pintar, instalar recubrimiento de pisos, fabricar e instalar cortinajes, papel tapiz, construir e instalar gabinetes, etc. Estas actividades se realizan mediante contratistas contratados por la firma para trabajar sobre una base diaria u horaria a una tasa negociada por ambas partes. El contratista proporciona una estimación del tiempo requerido para cada actividad. Algunas actividades las realiza el decorador de la firma. Cada actividad requiere materiales como pinturas o madera, y la firma debe seguir la pista del costo de los materiales, así como de la mano de obra por cada actividad, para cobrar al cliente.

7.7 a. Mapee el diagrama EE-R que desarrolló para el ejercicio 7.6 a un modelo estrictamente relacional.

b. Convierta el diagrama EE-R que desarrolló para el ejercicio 7.6 en un modelo objeto-relacional, y explique qué cambios haría al esquema relacional del inciso a).

Ejercicio de laboratorio: Creación de un diagrama EE-R

Con software de herramienta de dibujo como Visio, SmartDraw o la herramienta de dibujo de Word, dibuje un diagrama EE-R para el ejemplo descrito en el ejercicio 7.2.

PROYECTO DE MUESTRA: DIBUJO DE UN DIAGRAMA EE-R Y CREACIÓN DE UNA BASE DE DATOS RELACIONAL PARA LA GALERÍA DE ARTE

- Paso 7.1. Modifique el diagrama E-R y dibuje un diagrama EE-R para representar la empresa. Asegúrese de identificar las restricciones de participación de relación y cardinalidad usando notación (mín,máx). Identifique cualesquier conjuntos de entidad débiles. Use generalización y unión, según sea necesario, para expresar relaciones de clase, y agregue notación de restricción adecuada.

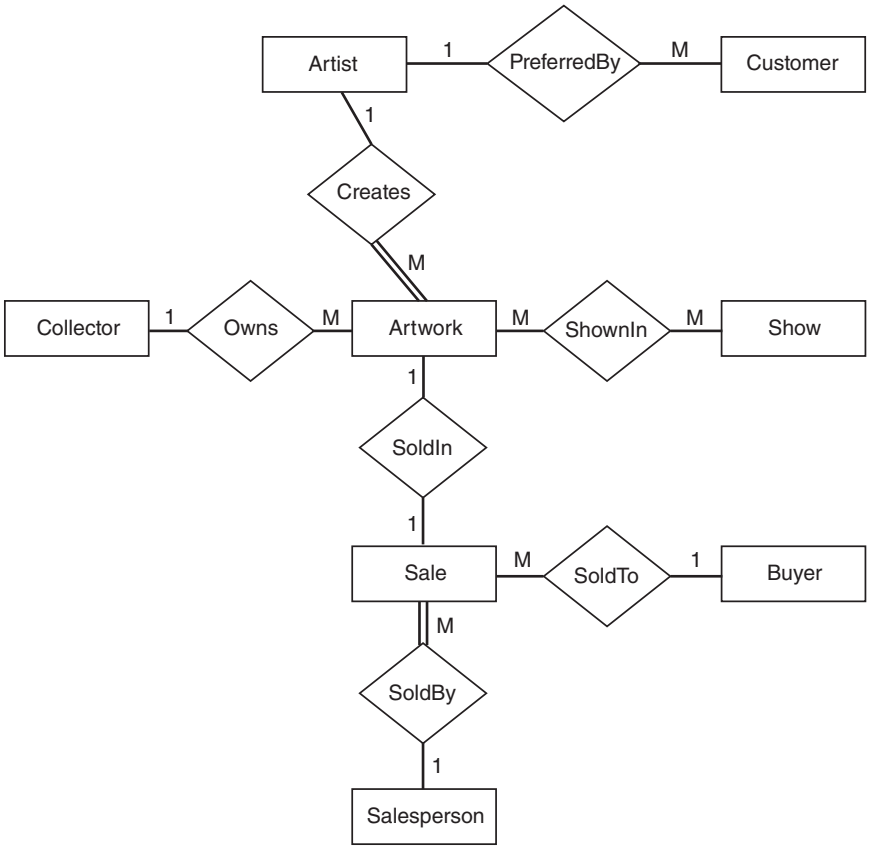
En la figura 7.8 se muestra un diagrama E-R. Note que el diagrama inicial carece de algunas características importantes y tiene algunas entidades y relaciones confusas. Por ejemplo, ¿`Buyer` también es un `PotentialCustomer`? ¿Un `Artist` es un `Collector` de su propia obra? Note también que muchos atributos de `Customer`, `Buyer`, `Artist` y `Collector` son esencialmente iguales. Un análisis más cuidadoso está a la orden. Los resultados se muestran en el diagrama EE-R de la figura 7.9, que debe comparar con el diagrama E-R de la figura 7.8.

En el diagrama E-R, note que `Buyer` y `PotentialCustomer` tienen atributos comunes, con la única diferencia de que las preferencias se conservan para los clientes pero no para los compradores. Tiene gran sentido empresarial agregar la información de preferencia para `Buyer`. Puesto que esto hace sus atributos idénticos, a ambos se les llamará `Customer`. Ahora puede preguntar si `Buyer` y `Customer` son el mismo. Se concluye que un comprador es un cliente que en realidad hizo una compra, es decir, `Buyer` es alguien que tiene una relación con una entidad `Sale`. Por tanto, se puede crear una especialización dentro de `Customer`, con `Buyer` como un subconjunto que tenga la relación con `Sale`.

La entidad `Artwork` se bosqueja como teniendo una relación con `Sale`, pero de hecho sólo aquellas obras que se vendieron tienen esta relación. El atributo `status` determina cuáles entidades obra de arte participan en la relación. Si se crea una especialización definida por atributo sobre estatus, es posible identificar tres subclases desarticulación de `Artwork`, a saber `SoldWork` (obra vendida), `WorkForSale` (obra en venta) y `ReturnedWork` (obra devuelta), como se muestra en la figura 7.9. De éstas, sólo `SoldWork` se relaciona con `Sale`. Se agrega un atributo, `location` (ubicación), para `WorkForSale`. Proporcionará la ubicación actual de la obra de arte.

Los atributos `Artist` y `Collector` son muy similares, y cada entidad se relaciona con `Artwork`. Al examinar estas entidades más de cerca, se ve que una obra de arte a veces es propiedad del artista y a veces de un coleccionista. Dado que el propietario de una obra particular es sólo uno de ellos, no ambos, se usa una unión, con `Owner` (propietario) como una subclase de la unión de `Artist` y `Collector`. El propietario participa en la relación Owns con `Artwork`, como antes. Sin embargo, `Artist` siempre participa en la relación `Creates` con `Artwork`, ya sea que el artista sea o no el propietario. La entidad `Salesperson` y la relación `SoldBy` permanecen como antes, así como `Show` y la relación `ShownIn`. Podría considerar crear una entidad Person con subclases `Customer`, `Salesperson`, `Owner`, pues todos éstos son personas, pero se elige no hacerlo y se detiene el proceso de abstracción.

Todavía es necesario agregar información de restricción. Sería preferible usar la notación (*mín..máx*) para restricciones de participación de relación y cardinalidad, pues es más expresivo que la notación anterior. En la relación `Creates`, anteriormente se decidió que `Artist` tiene participación parcial, pues se pueden conservar datos acerca de artistas entrevistados que todavía no tengan obras de arte aceptadas por la galería, así que el mín es 0. El máx es n, porque un artista puede haber creado muchas obras de arte aceptadas, así



Artist Attributes: artistId, name(firstName, lastName), interviewDate, interviewerName, phone(areaCode, telephoneNumber), address(street, city, state, zip), salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType

PotentialCustomer Attributes: potentialCustomerId, name(firstName, lastName), phone(areaCode, telephoneNumber), address(street, city, state, zip), dateFilledIn, preferredArtistId, preferredMedium, preferredStyle, preferredType

Artwork Attributes: artworkId, workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workyearCompleted

Collector Attributes: socialSecurityNumber, name(firstName, lastName), address(street, city, state, zip), interviewDate, interviewerName, phone(areaCode, telephonenumber), salesLastYear, salesYearToDate, collectionArtistId, collectionMedium, collectionStyle, collectionType, salesLastYear, salesYearToDate

Show Attributes: showTitle, showFeatureArtistId, showClosingDate, showTheme, showOpeningDate

Buyer Attributes: buyerId, name(firstName), address(street, city, state, zip), phone(areaCode, telephoneNumber), purchasesLastYear, purchasesYearToDate

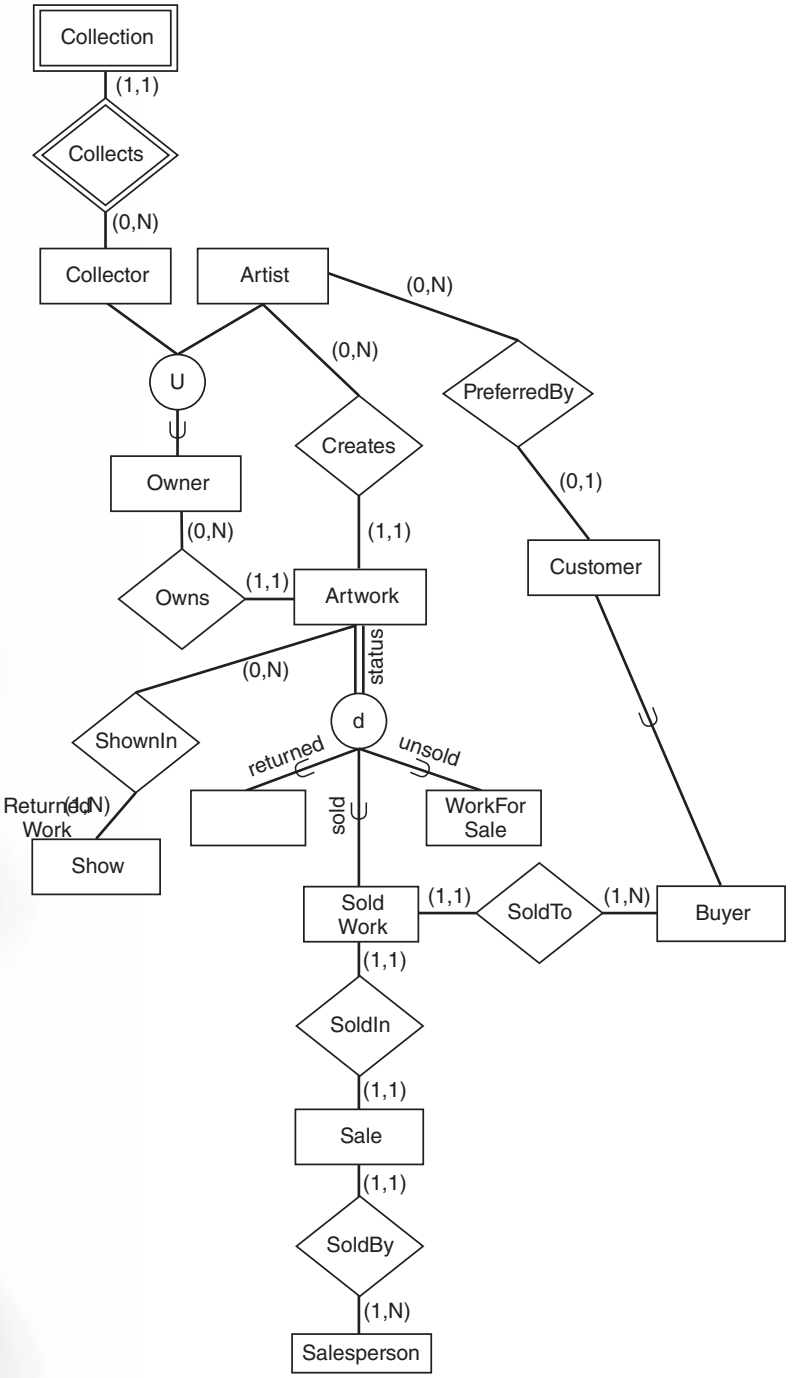
Sale Attributes: InvoiceNumber, amountRemittedToOwner, saleDate, salePrice, saleTax

Salesperson Attributes: socialSecurityNumber, name(firstName, lastName), address(street, city, state, zip)

FIGURA 7.8

Diagrama E-R para la Galería de Arte

FIGURA 7.9
Diagrama EE-R para la Galería de Arte



Artist Attributes: artistId, name(firstName, lastName), interviewerName, InterviewDate, phone(areaCode,telephoneNumber), address(street,city,state,zip),salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType
Collector Attributes:socialSecurityNumber,name(firstName,lastName),address(street,city,state,zip), interviewDate, interviewerName, phone(areaCode, telephoneNumber), salesLastYear,salesYearToDate)
Collection Attributes:collectorSocialSecurityNumber,collectionArtistId,collectionMedium, collectionStyle,collectionType
Owner: union of Artist and Collector
Artwork Attributes: artworkId, workTitle, askingPrice, dateListed, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted
SoldWork subtype of Artwork. Additional attribute: dateSold
ReturnedWork subtype of Artwork: Additional attribute: dateReturned
WorkForSale subtype of Artwork: Additional attribute: location
Customer Attributes: CustomerId, name(firstName, lastName), phone(areaCode, telephoneNumber), address(street, city, state, zip), dateFilledIn, preferredArtistId, preferredMedium, preferredStyle, preferredType
Buyer Subtype of Customer. Additional Attributes: purchasesLastYear, purchasesYearToDate
Show Attributes: showTitle, showFeaturedArtistId, showClosingDate, showTheme, showOpeningDate
Sale Attributes: invoiceNumber, amountRemittedToOwner, saleDate, salePrice, saleTax
Salesperson Attributes: socialSecurityNumber, name(firstName, lastName), address(street, city, state, zip)

que se usa (0..n) para `Artist` a `Creates`. La `Artwork` siempre participa en la relación `Creates` con exactamente un artista, así que se usa (1..1) para `Artwork` a `Creates`.

Para `SoldWork` a `SoldIn` (puesto que cada obra de arte vendida se vende exactamente una vez) se usa (1..1) para `SoldWork` a `SoldIn`. Para `Sale` a `SoldIn`, tanto mín como máx también son 1, así que en esta línea se escribe (1..1).

Para `Sale` a `SoldTo`, cada venta tiene exactamente un comprador, así que se escribe (1..1). Para `Buyer` a `SoldTo`, puesto que un `Buyer` es por definición alguien quien se involucró en una `Sale`, el mín es 1. Un comprador puede comprar muchas obras de arte, por tanto el máx es n, así que se usa (1..n).

Para `Sale` a `SoldBy`, cada venta se realiza por exactamente un vendedor, así que se usa (1..1). Para `Salesperson` a `SoldBy`, puesto que un vendedor puede no haber hecho alguna venta todavía, el mín es 0. Un vendedor puede hacer muchas ventas, así que el máx es n, lo que da (0..n).

Para `Owner` a `Owns` se acuerda que se puede almacenar información `Collector` o `Artist` acerca de personas que todavía no tengan una obra de arte aceptada, así que se supondrá que se puede almacenar información `Owner` sin una obra de arte, lo que hace el mín 0. Un `Owner` puede tener muchas obras de arte, así que el máx es n, lo que da (0..n). Para `Artwork` a `Owns`, cada obra de arte tiene exactamente un propietario, lo que la hace (1..1).

Para `Artwork` a `ShownIn`, el mín es 0, porque no todas las obras se exhiben. El máx es n, porque una obra de arte puede aparecer en muchas exposiciones. Para `Show` a `ShownIn` se hará el mín 1, si se supone que podría tener una exposición de una sola pieza de arte importante. El máx será n, porque una exposición normalmente tendría muchas obras de arte.

Ahora se debe decidir si la unión `Owner` es parcial o total. Para ser total, significaría que cada `Collector` y cada `Artist` debe ser un `Owner`. Dado que se puede tener información acerca de artistas cuyas obras sean propiedad de coleccionistas, esta unión es parcial. Una ampliación al diagrama que surge naturalmente, ahora que `Collector` está más claro, es que es posible que se quiera conservar la información `Collection` separada del `Collector`. Esto también permitiría almacenar datos acerca de múltiples colecciones que pertenecen al mismo coleccionista. Por tanto, se agrega una entidad `Collection`, con atributos `collectionArtist`, `collectionStyle`, `collectionMedium`, `collectionType`. Ésta es una entidad débil, que depende `Collector` para su clave primaria, con `Collects` como la relación. Se usará `collectionArtist` como la clave parcial. El `Collector` a `Collects` tiene mín de 0, porque un coleccionista puede no tener alguna colección distinguida, sino simplemente piezas individuales. El máx es n, pues ahora se supone que un coleccionista puede tener muchas colecciones diferentes, por varios artistas, así que se usa (0..n) para esta línea. Para `Collects` a `Collector`, cada `Collection` pertenece exactamente a un `Collector`, así que se usa (1..1).

Se puede considerar tratar `Customer` en una forma similar, y pedir a los clientes que mencionen varios artistas preferidos, estilos, etc., pero esto haría tediosa la forma de inscripción, y la información resultante podría no ser útil, así que se elige dejar el diagrama EE-R como existe en la figura 7.9.

- Paso 7.2. Mapee el diagrama EE-R a un modelo objeto-relacional. Se definirán tipos estructurados para direcciones, nombres y números telefónicos. Se usarán UDT tan frecuentemente como sea posible: definir tipos y luego crear tablas de dichos tipos. Para ayudar a distinguir entre el tipo y la tabla se usará la forma plural para el nombre de la tabla. Se usarán tipos REF en lugar de claves externas para representar relaciones, cuando sea posible.

Para traducir la unión se creará una tabla para que **Owners** tenga OwnerId y OwnerType como atributos. Luego se crearán tablas para **Artists** y **Collectors**, y cada una contendrá referencias a Owners. Se creará una tabla para **Customers**, que tenga una referencia al artista preferido. **Buyers** será una subtabla de Customers. Para **Artworks** se sustituirá la clave externa por Artist que se tenía en el modelo estrictamente relacional, por una referencia a Artists, y se agregará una referencia Owners en lugar de la clave externa para Collector. Se crearán subtablas para **ReturnedWork**, **WorksForSale** y **SoldWorks,** que tendrán una referencia a Buyers. La tabla **Shows** tendrá una referencia a Artists, para representar al artista presentado. **ShownIn** todavía se necesita para la relación muchos a muchos, pero ambas claves externas se sustituirán por referencias. Se necesita una tabla para **Salespeople**. Entonces se creará la tabla **Sales** con referencias a SoldWorks y Salespeople. Finalmente, se agregará una nueva tabla para **Collections**, con referencias tanto a Collectors como a Artists.

- Paso 7.3. Escriba el DDL para crear la base de datos objeto-relacional usando sintaxis Oracle 9i, y ejecute los comandos.

```
CREATE OR REPLACE TYPE AddressType AS OBJECT(
street      VARCHAR2(50),
city        VARCHAR2(15),
state       CHAR(2),
zip         CHAR(10));

CREATE OR REPLACE PhoneType AS OBJECT(
areaCode           CHAR(3),
telephoneNumber    CHAR(7));

CREATE OR REPLACE NameType AS OBJECT(
firstName   VARCHAR2(15),
lastName    VARCHAR2(20));

CREATE OR REPLACE OwnerType AS OBJECT (
ownerId     NUMBER(6),
ownerType   VARCHAR2(9));

CREATE TABLE Owners OF OwnerType(
CONSTRAINT Owners_ownerId_pk PRIMARY KEY (ownerId))
OBJECT IDENTIFIER PRIMARY KEY;

CREATE OR REPLACE TYPE ArtistType AS OBJECT (
artistId             NUMBER(6),
name                 NameType,
interviewDate        DATE,
interviewerName      NameType,
phone                PhoneType,
address              AddressType,
salesLastYear        NUMBER(8,2),
salesYearToDate      NUMBER(8,2),
socialSecurityNumber CHAR(9),
usualMedium          VARCHAR2(15),
usualStyle           VARCHAR2(15),
usualType            VARCHAR2(20);

CREATE TABLE Artists OF ArtistType(
CONSTRAINT Artists_artistId_pk PRIMARY KEY ArtistId,
CONSTRAINT Artists_SSN_uk UNIQUE socialSecurityNumber))
OBJECT IDENTIFIER PRIMARY KEY;
```

```
CREATE OR REPLACE CollectorType AS OBJECT (
socialSecurityNumber CHAR(9),
name                 NameType,
interviewDate        DATE,
interviewerName      NameType,
phone                Phonetype,
address              Addresstype,
salesLastYear        NUMBER (8,2),
salesYearToDate      NUMBER(8,2));

CREATE TABLE Collectors OF CollectorType(
CONSTRAINT Collectors_SSN_pk PRIMARY KEY (socialSecurityNumber));
OBJECT IDENTIFIER PRIMARY KEY;

CREATE OR REPLACE TYPE CustomerType AS OBJECT (
customerId           NUMBER(6),
Name                 NameType,
Phone                phoneType,
Address              ddressType,
dateFilledIn         DATE,
aid                  REFArtistType,
preferredMedium      VARCHAR2(15),
preferredStyle       VARCHAR2(15),
preferredType        VARCHAR2(20))
NOT FINAL

CREATE TABLE Customers OF CustomerType(
CONSTRAINT Customers_customerId_pk PRIMARY KEY (customerId))
OBJECT IDENTIFIER PRIMARY KEY;

CREATE OR REPLACE TYPE BuyerType UNDER CustomerType(
purchasesLastYear   NUMBER (8,2),
purchasesYearToDate NUMBER(8,2));

CREATE TABLE Buyers OF BuyerType;

CREATE OR REPLACE TYPE ArtworkType AS OBJECT (
artworkId   NUMBER(6),
aId         REF(ArtistType),
workTitle   VARCHAR2(50),
askingPrice NUMBER(8,2),
dateListed  DATE,
dateShown   DATE,
status      VARCHAR2(15),
workMedium  VARCHAR2(15),
workSize    VARCHAR2(15),
workStyle      VARCHAR2(15),
workType       VARCHAR2(20),
workYearCompleted CHAR(4),
ownId          REF OwnerType
NOT FINAL

CREATE OR REPLACE TABLE Artworks OF ArtworkType(
CONSTRAINT Artwork_artworkId_pk PRIMARY KEY (artworkId))
OBJECT IDENTIFIER PRIMARY KEY;

CREATE OR REPLACE TYPE ReturnedWorkType UNDER ArtworkType (
dateReturned DATE);
```

```
CREATE TABLE ReturnedWorks OF ReturnedWorkType (

CREATE OR REPLACE TYPE WorkForSaleType UNDER ArtworkType;
location       VARCHAR2(50));

CREATE OR REPLACE TABLE WorksForSale OF WorkForSaleType;

CREATE OR REPLACE TYPE SoldWorkType UNDER ArtworkType (
dateSold    DATE,
cid         REFCustomerType;

CREATE TABLE SoldWorks OF SoldWorkType;

CREATE OR REPLACE TYPE ShowType AS OBJECT (
ShowTitle       VARCHAR2(50),
aid             REFartistType,
showClosingDate DATE,
showTheme       VARCHAR2(50),
showOpeningDate DATE);

CREATE TABLE Shows of ShowType (
CONSTRAINT Shows_ShowTitle_pk PRIMARY KEY (ShowTitle))
OBJECT IDENTIFIER PRIMARY KEY;

CREATE TABLE ShownIn
(workId  REFArtworkType,
showId   REF ShowType;

CREATE OR REPLACE TYPE SalespersonType AS OBJECT (
socialSecurityNumber CHAR(9),
name                 NameType,
address              AddressType);

CREATE TABLE Salespeople OF SalespersonType (
CONSTRAINT Salespeople_SSN_pk PRIMARY KEY (socialSecurityNumber))
OBJECT IDENTIFIER PRIMARY KEY;

CREATE OR REPLACE TYPE SaleType AS OBJECT (
invoiceNumber          NUMBER(6),
workId                 REFArtworkType,
amountRemittedToOwner  NUMBER(8,2),
saleDate               DATE,
salePrice              NUMBER(8,2),
saleTax                NUMBER(6,2),
sid                    REFSalespersonType;

CREATE TABLE Sales OF SaleType
CONSTRAINT Sales_invoiceNumber_pk PRIMARY KEY (invoiceNumber));

CREATE OR REPLACE TYPE CollectionType AS OBJECT (
cid                    REFCollectorType,
aid                    REFArtistType,
collectionMedium       VARCHAR2(15),
collectionStyle        VARCHAR2(15),
collectionType         VARCHAR2(20));

CREATE TABLE Collections OF CollectionType;
```

- Paso 7.4. Escriba cinco comandos DML para la nueva base de datos, que ilustren diferencias del DML estrictamente relacional.

  Nota: Cuando se usa un alias de tabla, Oracle permite quitar las referencias al usar notación punto, que se usará en las consultas.

1. Encuentre los nombres de todos los artistas que se entrevistaron después del 1 de enero de 2004, pero que no tengan obras de arte en listas.

```
SELECT a.name.firstName, a.name.lastName
FROM Artists a
WHERE a.interviewDate > '01-Jan-2004' AND NOT EXISTS
   (SELECT *
   FROM Artworks w
   WHERE w.aId.artistId=a.artistId);
```

2. Encuentre la comisión total que el vendedor John Smith ganó entre el 1 de abril de 2004 y el 15 de abril de 2004. Recuerde que la galería carga 10% de comisión y que el vendedor recibe la mitad de ello, que es 5% del precio de venta.

```
SELECT .05 * SUM(s.salePrice)
FROM Sales s
WHERE s.saleDate >='01-Apr-2004' AND s.saleDate <='15-Apr-2004' AND
   s.sid.socialSecurityNumber = (SELECT p.socialSecurityNumber
            FROM Salesperson p
            WHERE p.name.firstName= 'John' AND p.name.lastName ='Smith');
```

3. Encuentre los nombres de coleccionista, nombres de artista y títulos de todas las obras de arte que sean propiedad de coleccionistas, no de los mismos artistas, en orden de apellido del coleccionista.

```
SELECT c.name, a,name, w.workTitle
FROM Collectors c, Artists a, Artworks w
WHERE a.artistId = w.aid.artistId AND w.ownid.ownerType = 'collector')
ORDER BY c.name.lastName;
```

4. Para cada comprador potencial, encuentre información acerca de las exposiciones que presentan a su artista preferido. Cite en orden de Id de cliente.

```
SELECT c.name, s.showTitle, s.showOpeningDate, s.showClosingDate
FROM Shows s , Customers c
WHERE s.aid.ArtistId = c.aid.artistId
ORDER BY c.customerId;
```

5. Encuentre el precio de venta promedio de las obras de la artista Georgia Keefe.

```
SELECT AVG(s.salePrice)
FROM Sales s
WHERE s.workId.artworkId IN (SELECT w.artworkid
   FROM Artworks w
   WHERE w.aid.artistId = (SELECT a.artistId
      FROM Artists a
      WHERE a.name.lastName = 'Keefe' AND a.name.firstName =
      'Georgia'));
```

PROYECTOS ESTUDIANTILES: DIBUJO DE UN DIAGRAMA EE-R Y CREACIÓN DE UNA BASE DE DATOS OBJETO-RELACIONAL PARA LOS PROYECTOS ESTUDIANTILES

- Paso 7.1. Modifique el diagrama E-R y dibuje un diagrama EE-R para representar la empresa. Asegúrese de identificar las restricciones de participación de relación y cardinalidad usando notación (mín, máx). Identifique cualquier conjunto de entidades débil. Use generalización y unión, según necesite, para expresar relaciones de clase, y agregue notación de restricción adecuada.
- Paso 7.2. Mapee el diagrama EE-R a un modelo objeto-relacional.
- Paso 7.3. Escriba el DDL para crear la base de datos objeto-relacional, usando sintaxis Oracle 9i o la sintaxis del sistema de gestión de base de datos objeto-relacional que use, y ejecute los comandos.
- Paso 7.4. Escriba cinco comandos DML para la nueva base de datos, e ilustre las diferencias del DML estrictamente relacional.

CAPÍTULO 8

El modelo orientado a objetos

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Los orígenes del modelo de datos orientado a objetos
- Conceptos fundamentales del modelo de datos orientado a objetos: objeto, clase, herencia, identidad de objeto
- Cómo construir un diagrama de clase UML
- Los conceptos básicos del modelo ODMG
- Cómo escribir ODL para un esquema orientado a objetos usando un diagrama de clase UML
- Cómo escribir consultas simples usando OQL
- Cómo se desarrolla y usa una base de datos orientada a objetos

8.1 Razones para el modelo de datos orientado a objetos

El modelo relacional tradicional, en el cual los datos se representan como tablas que tienen filas de atributos de un solo valor, es limitado en su capacidad para representar los datos y relaciones complejos necesarios para aplicaciones avanzadas. Así como se encontró que el modelo E-R carecía de capacidad para representar un modelo conceptual en aplicaciones como desarrollo de software, diseño asistido por computadora, sistemas de información geográfica, diseño de ingeniería y muchas otras aplicaciones avanzadas, el modelo relacional tradicional carece de las estructuras de datos para soportar los requisitos de información en dichas aplicaciones. Los **lenguajes de programación orientados a objetos**, que comenzaron con Simula y Smalltalk, presentaron una forma alternativa de los programas de diseño, en los que las estructuras de datos y sus operaciones tienen importancia primaria. La programación orientada a objetos se reconoce ampliamente como un método para producir código reutilizable muy confiable. La promesa de los primeros lenguajes orientados a objetos, y la popularidad de los lenguajes orientados a objetos C++ y Java, también influyeron en el modelado de las bases de datos. El reto de extender estos lenguajes a las bases de datos es moverse de los objetos temporales creados y manipulados por programas a **objetos persistentes** que se pueden almacenar en una base de datos. Los sistemas de gestión de bases de datos orientadas a objetos permiten al diseñador de bases de datos crear objetos interrelacionados enormemente complejos y darles persistencia. Como se discutió en el capítulo 7, los proveedores de sistemas de bases de datos del modelo relacional respondieron a los retos de los sistemas orientados a objetos al extender el modelo relacional para incorporar algunos conceptos orientados a objetos. Puesto que los proveedores abordaron algunas de las limitaciones de las bases de datos relacionales, y dado que los usuarios que tienen una fuerte inversión en los sistemas relacionales no están inclinados a migrar a un paradigma completamente nuevo, la popularidad de las bases de datos orientadas a objetos ha sido limitada. Sin embargo, están en uso algunos sistemas de gestión de bases de datos estrictamente orientadas a objetos (**OODBMS**, por sus siglas en inglés), como Objectivity, GemStone, ObjectStore, Ontos y Versant.

8.2 Conceptos de datos orientados a objetos

8.2.1 Objetos y literales

El concepto de **objeto** es fundamental en el modelo orientado a objetos. Un objeto tiene un estado (valor) y un identificador único. Un objeto es similar a una entidad en la terminología previa, excepto que un objeto no sólo tiene **elementos de datos**, sino también un conjunto de **métodos** (funciones o procedimientos) que se pueden realizar sobre él. En el capítulo 7 se presentaron muchos ejemplos de objetos, en la discusión acerca de cómo el modelo relacional se extendió para incluir objetos. Además de los tipos de datos fundamentales y los tipos estructurados definidos por el usuario que corresponden a registros, los sistemas orientados a objetos soportan tipos colecciones como arrays, conjuntos y otros, y tipos referencia, que son similares a punteros en programación. Se pueden crear nuevos tipos que incorporen los tipos definidos con antelación. Los objetos están **encapsulados**, lo que significa que sus datos y métodos forman una unidad, y que el acceso a los datos está restringido. Sólo los métodos propios del objeto, creados y programados por el diseñador del objeto, tienen permiso de acceder a los datos, y los protegen de cambios por código externo. La apariencia externa de un objeto, visible al mundo exterior, se llama **interfaz**. El mundo exterior interactúa con el objeto a través de su interfaz, y llama a los métodos propios del objeto para realizar cualquier operación sobre sus elementos de datos. La figura 8.1 muestra dos objetos de una clase llamada `Person`. Cada instancia objeto `Person` tiene su

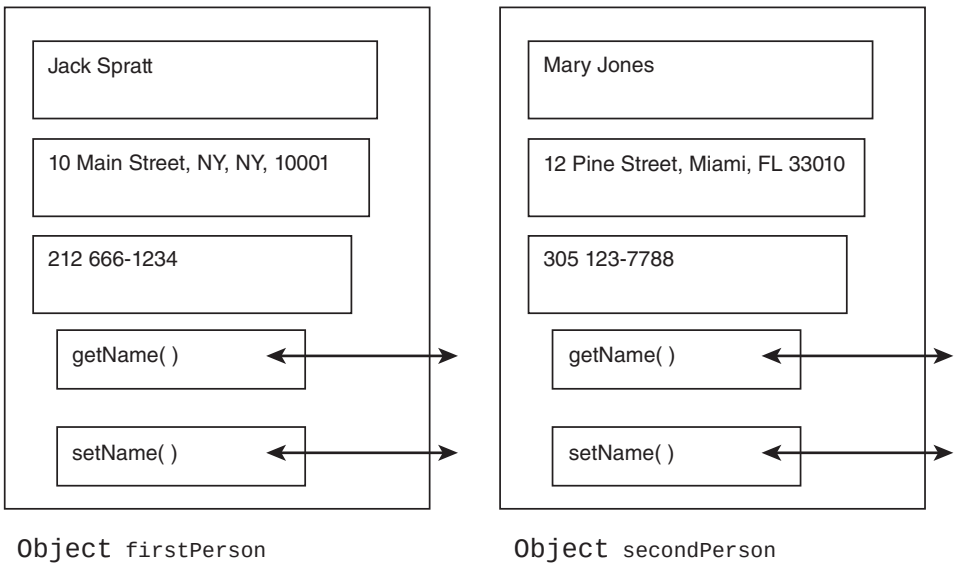


FIGURA 8.1
Dos objetos Person

propio identificador (`firstPerson` y `secondPerson`, en este ejemplo) y atributos `name`, `address` y `telephone`, con sus propios valores para dichos atributos. Cada objeto está encapsulado y los valores de sus atributos son accesibles al mundo exterior sólo a través de sus métodos, `getName( )` [obtener nombre] y `setName( )` [establecer nombre]. A partir de los nombres de estos métodos se puede suponer que el programador que diseñó estos objetos escribió código para una función que devuelve el valor del atributo `name` (`getName( )`) y uno que permite al usuario proporcionar un nuevo valor para el atributo `name` (`setName( )`). Para también hacer accesibles los otros dos atributos, el programador que diseñó estos objetos a su vez necesita agregar métodos `get` y `set` para ellos.

Una **literal** difiere de un objeto en que tiene un estado (valor) pero no identificador de objeto. Las literales atómicas son los valores de los tipos internos básicos, pero también se pueden tener literales estructuradas.

8.2.2 Clases

Una **clase** es un conjunto de objetos que tiene la misma estructura, que incluye las mismas variables con los mismos tipos de datos, y los mismos métodos y relaciones. Una clase es aproximadamente equivalente a un tipo entidad, excepto que incluye operaciones y relaciones, no sólo elementos de datos. Las clases se definen por citar sus componentes, que consiste en **miembros datos** (atributos, variables de instancia), **métodos** miembro (funciones o procedimientos que pertenecen a la clase) y las **relaciones** en las que participa la clase. La forma general que se usará para una definición de clase es:

```
class nombre_clase {
   lista de atributos, métodos, relaciones
}
```

Por ejemplo, se puede definir una clase `Person` como:

```
class Person {
   attribute string name;
   attribute string address;
   attribute string phone;
   void setName(string newName); // method
   string getName( ); //method
}
```

En esta definición de clase, los tres atributos se identifican mediante sus tipos de datos y nombres, y se mencionan dos métodos. El primero, `setName( )`, tiene un solo parámetro cadena, pero no tiene valor return, como se indica mediante `void`. Como el nombre implica, tiene la intención de establecer el valor del atributo `name` de un objeto. El código para este método lo escribe el diseñador de la clase. Se puede definir del modo siguiente:

```
void setName(string newName)
{
   self.name = newName;
}
```

El método se llama desde un programa usuario para establecer el nombre de un objeto `Person`. Por ejemplo, si `firstPerson` se crea como un objeto `Person` dentro del programa usuario, al usar una declaración como

```
Person          firstPerson;
```

o

```
Person firstPerson = new Person();
```

el usuario no tiene capacidad de establecer el nombre de la persona al referirse directamente al atributo `name` de `firstPerson`, debido al encapsulado de los objetos. En vez de ello, el programa usuario debe contener un comando como:

```
firstPerson.setName('Jack Spratt');
```

Aquí, `firstPerson` es el "objeto que llama", el objeto utilizado para llamar al método, y se convierte en el parámetro implícito para el método. Dentro del método se le refiere como el parámetro *self*. A él se hace referencia mediante el nombre en la línea de asignación del método,

```
self.name = newName;
```

pero uno no lo necesita, y sólo se podría escribir

```
name = newName;
```

pues se entiende que los atributos no calificados referidos dentro del método son los del objeto que llama, *self*.

Para el método `getName( )`, el código escrito por el diseñador de la clase puede ser:

```
string getName( )
{
return name;
}
```

Dentro del programa usuario se llamaría este método usando un objeto como el parámetro implícito. Continuando con el ejemplo, si, como parte del programa usuario, se escribe

```
string employeeName = firstPerson.getName( );
```

el objeto `firstPerson` es el parámetro implícito para el método, y se devuelve el nombre de dicho objeto, lo que resulta en la asignación del valor 'Jack Spratt' a la variable de programa `employeeName`.

El conjunto de objetos que pertenece a una clase se llama **extensión** de la clase. Un conjunto de entidades es aproximadamente equivalente a una extensión de clase. Los tipos de datos en una clase pueden ser tipos atómicos predefinidos, incluidos enteros, reales, carácter y booleano, y tipos más complejos también, como tipos definidos por el usuario. Los objetos individuales en la clase se llaman **instancias**, **instancias objeto** o simplemente **objetos**, y son similares a instancias entidad.

8.2.3 Jerarquías de clase y herencia

Las clases se organizan en **jerarquías de clase**, que consisten en **superclases** y **subclases**, en las que cada subclase tiene una relación ***isa*** con su superclase. Esto es similar al concepto de especialización y generalización que se vio en el modelo EE-R. Las subclases heredan los miembros de datos y métodos de sus superclases, y pueden tener miembros datos adicionales y métodos propios. La figura 8.2 muestra un ejemplo de una jerarquía de clase. Note que éste no es un diagrama EE-R, como se ve por la presencia del triángulo *isa*, en oposición al círculo de especialización que se vio en los diagramas EE-R. La figura 8.3 muestra un conjunto simplificado de declaraciones de clase para corresponder a la jerarquía de clase de la figura 8.2. Todos los objetos en la superclase, `Person`, tienen miembros datos `name`, `address` y `phone` que son comunes a todas las personas. Los que están en la `subclase Student` heredan `name`, `address` y `phone`, y agregan miembros datos propios, `stuId` y `credits`. La subclase `Faculty` también hereda los atributos `Person`, pero tiene sus propios miembros datos, `facId`, `dept` y `rank`. A su vez, las subclases `Undergraduate` y `Graduate` heredan los atributos de ambas, `Person` y `Student`. Las subclases se indicaron mediante *isa* y se incluyeron algunos métodos para cada clase. Junto con los atributos, la subclase también hereda automáticamente los métodos de cada superclase, de modo que, por ejemplo, las subclases `Student` y `Faculty`, así como las subclases `Undergraduate` y `Graduate` de `Student`, tienen los métodos `getName( )` y `setName( )` de `Person`. Las subclases también pueden tener métodos adicionales, como se ve para la subclase `Student`, que tiene métodos `getCredits( )` y `addCredits( )`. Las subclases `Undergraduate` y `Graduate` también heredan estos métodos, y cada una tiene sus propios métodos: `setMajor( )` y `getMajor( )` para `Undergraduate`, y `getProgram( )` para `Graduate`. Se podría **sobrescribir** (override) un método al definir para la subclase un método que tenga la misma **firma** (signature) como método de la superclase, esto es, el mismo nombre y el mismo número y tipos de parámetros y tipo return. En este caso, el nuevo método se usará sobre objetos de la subclase.

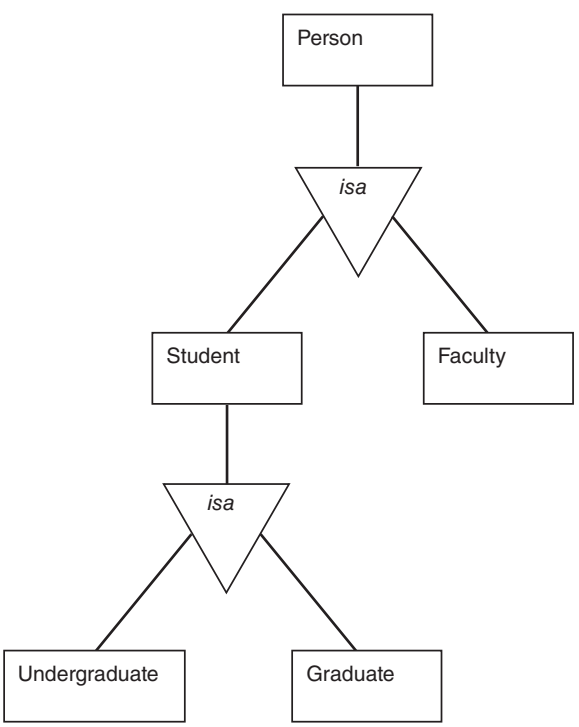


FIGURA 8.2
Una jerarquía de clases

FIGURA 8.3
Definición simplificada de una jerarquía de clase

```
class Person {
    attribute name string;
    attribute Struct addr(string street, string city, string state,
        string zip) address;
    attribute phone string;
    string getName( ); //método que devuelve una cadena
    void setName(string newName); // método sin devolución
};
class Student isa Person {
    attribute stuId string;
    attribute credits int;
    int getCredits( );
    void addCredits(int numCredits);
};
class Faculty isa Person {
    attribute facId string;
    attribute dept string;
    attribute enum FacultyRank{instructor, assistant, associate,
        professor} rank ;
    string getRank( );
};
class Undergraduate isa Student{
    attribute major string;
    string getMajor( );
    void setMajor(string newMajor);
};
class Graduate isa Student {
    attribute program string;
    string getProgram( );
};
```

Herencia múltiple significa que una subclase pertenece a más de una superclase, y que hereda atributos y métodos de las superclases múltiples. En la figura 8.4 se muestra un ejemplo simple, que muestra una clase para asistentes de profesor graduados como una subclase tanto de `Student` como de `Faculty`. Cada objeto `TeachingAssistant` tiene todos los atributos y métodos de `Person`, `Faculty`, `Student` y `Graduate`, así como cualesquier atributos y métodos propios. Se podría agregar su definición de clase, que se muestra en la figura 8.4, a la de la figura 8.3. La definición de clase para `TeachingAssistant` incluye un atributo referencia `assignedClassSection`, que es una referencia a otra clase, `ClassSection`, cuya definición no se muestra.

8.2.4 Identidad de objeto

La **identidad de objeto** es otro concepto fundamental para los modelos orientados a objetos. A cada objeto en la base de datos se le asigna su propio identificador de objeto único, **OID**, que permanece invariable durante toda la vida del objeto. A diferencia del modelo relacional, donde se debe identificar una clave primaria para cada relación, el modelo orientado a objetos proporciona automáticamente identificadores únicos. El valor del OID no depende del valor de algún atributo del objeto. En el modelo relacional podría actualizarse el valor de la clave primaria. Sin embargo, en el modelo orientado a objetos, el OID nunca

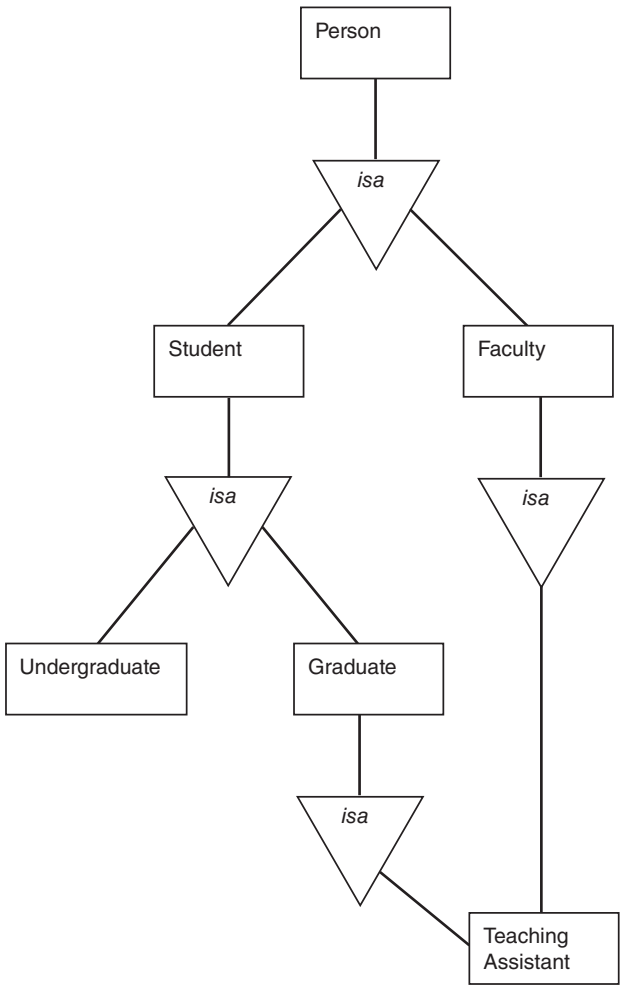


FIGURA 8.4
Herencia múltiple

```
...
class TeachingAssistant isa Graduate isa Faculty{
      attribute fundingSource string;
      attribute annualStipend real(7,2);
      attribute assignedClassSection REF ClassSection;
};
```

cambia. Dado que siempre es posible separar los objetos por sus OID, también es posible que dos objetos diferentes tengan exactamente los mismos valores para todos sus ítems de datos, una situación prohibida en el modelo relacional. Las referencias usan los OID de los objetos referenciados.

8.3 Modelado de datos orientados a objetos usando UML

Los diagramas E-R y EE-R no son muy adecuados para la representación de objetos, pues no permiten la representación de métodos. Una técnica de diagramación ampliamente usada llamada diagramas de clase **UML** (Unified Modeling Language, lenguaje de modelado unificado) permite expresar conceptos orientados a objetos de manera más natural que cualquiera de las técnicas anteriores. La figura 8.5 muestra un diagrama de clase UML para parte de una base de datos University, en la que se muestran muchas de las características

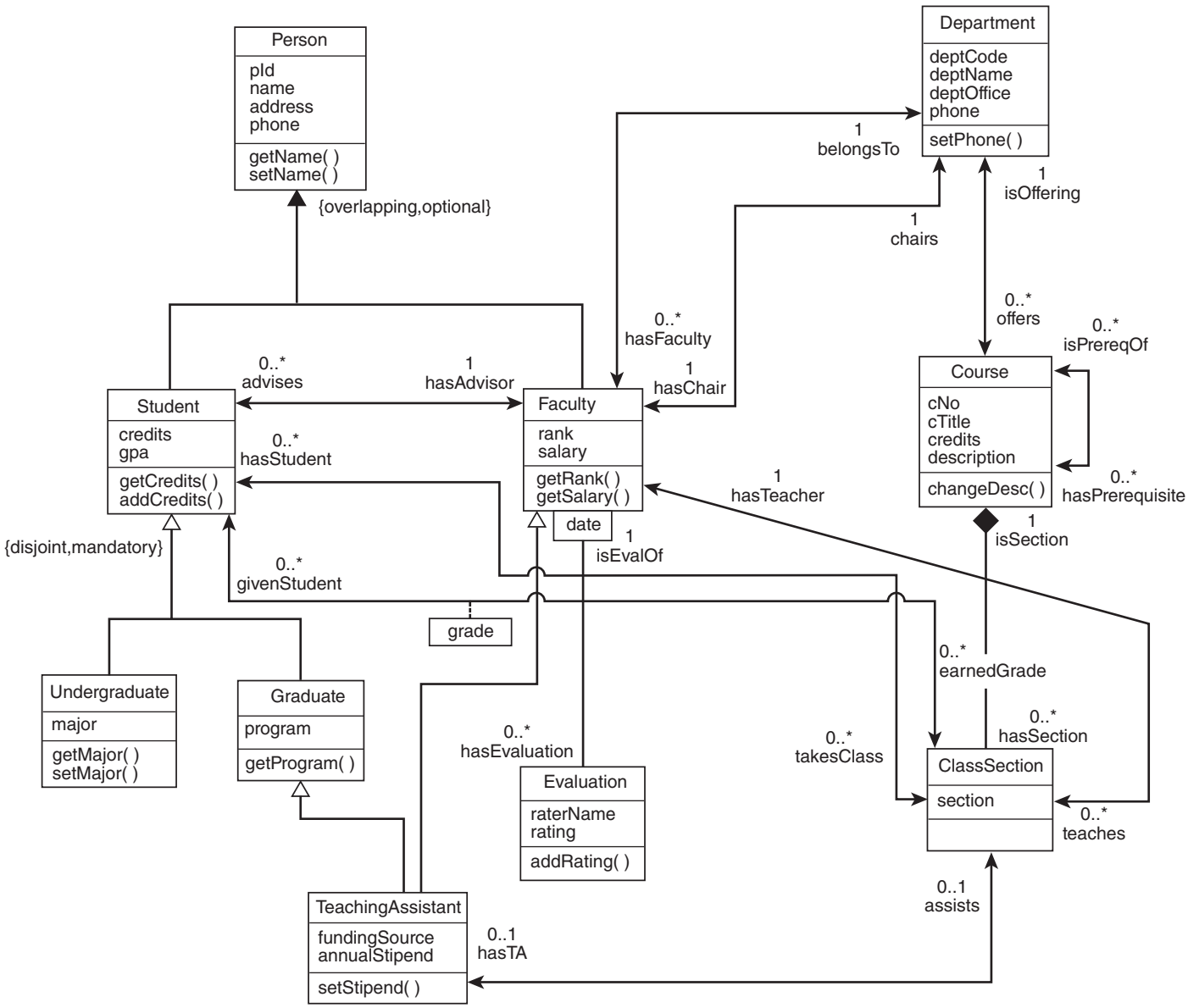


FIGURA 8.5

Un diagrama de clase UML

de tales diagramas. Las elecciones hechas en la construcción de este diagrama tienen la intención de demostrar diferentes cualidades de la notación de diagramación, y no representan el único modelo posible para el ejemplo. En un diagrama UML, los **rectángulos** representan clases. Los rectángulos de clase se subdividen en tres secciones, la parte superior para el nombre de la clase, la parte media para los atributos y la parte inferior para los métodos. En UML se distingue entre **asociaciones**, que son relaciones unidireccionales o bidireccionales entre clases, y **agregación**, que es un tipo de relación que conecta un todo a sus partes. Las **asociaciones**, o conexiones binarias entre clases, se representan mediante líneas que conectan los rectángulos de clase. El nombre de la asociación opcionalmente puede aparecer en la línea. Cualquier atributo descriptivo de la relación se muestra en un recuadro

conectado a la línea de asociación mediante una línea punteada, como se muestra en el atributo `grade` de la relación `Grading`. En las líneas de asociación también pueden aparecer **nombres de función** (rolenames), si se necesitan por claridad, pero su colocación es opuesta a la de un diagrama E-R. Para una de las asociaciones entre `Dept` y `Faculty` se usaron los nombres de función `has chair` y `chairs` para mostrar que conectan al jefe de departamento con el departamento. La **agregación** es un tipo especial de relación que conecta un todo con sus partes. Si la frase que describe la relación de una clase con otra es la de "es parte del" otro, y la clase está claramente subordinada a la otra, entonces la agregación es la mejor representación de la relación. La relación entre `Course` y `ClassSection` se representó como una agregación. La agregación se representa mediante una línea con un diamante en el lado de la clase que es el agregado. Si a la agregación no se le da un nombre, la línea de agregación se interpreta como "tiene". Las restricciones de participación y la cardinalidad para ambos tipos de relaciones se especifican con el uso de una variación de la notación (*mín..máx*) discutida en la sección 7.4. En UML se llaman **indicadores de multiplicidad**. Se escriben con la forma *mín..máx*, sin paréntesis, y se colocan en el lado opuesto de su ubicación en un diagrama EE-R. El * se usa en lugar de M, y la abreviatura 1 significa exactamente uno, esto es, 1..1. Las relaciones recursivas entre miembros de la misma clase, llamadas **asociaciones reflexivas** o **agregaciones reflexivas**, se muestran mediante una línea de asociación o de agregación que va de una clase hacia ella misma. La relación que conecta los cursos con sus prerrequisitos es una asociación reflexiva. Los nombres de funciones aparecen en la línea, con indicadores de multiplicidad en cada extremo. Se muestra que un curso puede tener cero a muchos prerrequisitos, y que un curso puede ser un prerrequisito para cero a muchos otros cursos. Las **jerarquías de generalización** se representan mediante líneas que conectan las subclases a la superclase, con un triángulo que apunta hacia la superclase. En la figura 8.5, `Person` se especializa en `Student` y `Faculty`, y `Student` se especializa en `Undergraduate` y `Graduate`. `Graduate` tiene una subclase, `TeachingAssistant`, que también es una subclase de `Faculty`, lo que demuestra herencia múltiple. Un triángulo de generalización lleno representa subclases **traslapadas** (overlapping), mientras que un triángulo en blanco (sólo el borde) representa las **desarticuladas** (disjoint). Si cada miembro de la superclase debe pertenecer a una subclase, la especialización se describe como **mandatory** (obligatoria), en oposición a **opcional**. Las restricciones de especialización se pueden escribir sobre la línea junto al triángulo, encerradas entre llaves. Por ejemplo, se escribe {overlapping,optional} para la especialización de `Person` en `Student` y `Faculty`, y {disjoint,mandatory} para `Student` en `Undergraduate` y `Graduate`.

Las **entidades débiles** se pueden representar mediante un recuadro unido a la entidad propietaria fuerte asociada, con el **discriminador** (clave parcial) que aparece en un recuadro abajo del rectángulo de clase del propietario. `Evaluation` se muestra como una entidad débil, dependiente de `Faculty`. Para este ejemplo, se supone que el mismo docente nunca se evalúa dos veces el mismo día, así que la fecha se usa como discriminador.

8.4 El modelo ODMG y ODL

El **Object Database Management Group** (ODMG: grupo de gestión de bases de datos objeto) es un grupo de proveedores que desarrolló y publicó estándares para bases de datos orientadas a objetos. El grupo tiene estándares para el modelo objeto en sí, **lenguaje de definición de objetos** (ODL), **lenguaje de consulta de objetos** (OQL) y ligas de lenguaje para C++, Java y Smalltalk. ODL es un lenguaje estándar para la descripción de esquemas en términos orientados a objetos, que atiende a la misma función que el DDL. La figura 8.6 muestra la definición ODL para el esquema de universidad que se muestra en la figura 8.5.

FIGURA 8.6
Definición ODL para el esquema University

```
class Person(
extent people
key pId)
{    attribute pId int;
     attribute name string;
     attribute Struct Addr(string street, string city, string state, string zip) address;
     attribute phone string;
     string getName( );
     void setName(string newName);
};

class Student extends Person
(extent students)
{    attribute credits int;
     attribute gpa real(3,2);
     int getCredits( );
     void addCredits(int numCredits);
     relationship Set<ClassSection> takesClass Inverse ClassSection::hasStudent;
     relationship Set<Grade> earnedGrade Inverse Grade::givenStudent
     relationship Faculty hasAdvisor Inverse Faculty::advises;
};

class Faculty extends Person
(extent facs)
{    attribute enum FacultyRank{instructor, assistant, associate, professor} rank;
     attribute salary real(8,2);
     string getRank( );
     real get salary( );
     relationship Department belongsTo Inverse Department::hasFaculty;
     relationship Set<ClassSection> teaches Inverse ClassSection::hasTeacher
     relationship Set<Student>advises Inverse Student::hasAdvisor;
     relationship Department chairs Inverse Department::hasChair;
};

class Undergraduate extends Student
(extent undergraduates)
{    attribute major string;
     string getMajor( );
     void setMajor(string newMajor);

};

class Graduate extends Student
(extent graduates)
{    attribute program string;
     string getProgram( );
};

class TeachingAssistant extends Graduate
(extent teachingAssistants)
{    attribute fundingSource string;
     attribute annualStipend real(7,2);
     relationship ClassSection assists Inverse ClassSection::hasTA;
};
```

(*continúa*)

FIGURA 8.6
Continuación

```
class Department
(extent departments
key deptCode, deptName)
{   attribute deptCode string;
    attribute deptName string;
    attribute deptOffice string;
    attribute phone string;
    relationship Set<Faculty> hasFaculty Inverse Faculty::belongsTo;
    relationship Faculty hasChair Inverse Faculty::chairs;
    relationship Set<Courses> offers Inverse Courses::isOffering
};
class Course
(extent courses
key cNo)
{   attribute cNo string;
    attribute cTitle string;
    attribute credits int;
    attribute description string;
    void change Desc(string newDesc);
    relationship Set<Course> hasPrerequisite Inverse isPrereqOf;
    relationship Set<Course> isPrereqOf Inverse hasPrerequisite;
    relationship Department isOffering Inverse Department::offers;
};
class ClassSection extends Course
(extent sections)
{   attribute section string
    relationship Set<Student> hasStudent Inverse Student::takesClass;
    relationship Faculty hasTeacher Inverse Faculty::Teaches
    relationship TeachingAssistant hasTA Inverse TeachingAssistant::assists;
    relationship Set<Grade>givenStudent Inverse Grade::earnedGrade
};
class Evaluation
(extent evaluations
key (date, isEvalOf)
{   attribute Struct DateType(int day, int month, int year) date;
    attribute string raterName;
    attribute int rating;
    relationship Faculty isEvalOf Inverse Faculty::hasEvaluation;
};
class Grade
(extent grades)
{   attribute string grade;
    relationship ClassSection section Inverse ClassSection::givenStudent;
    relationship Student givenStudent Inverse Student::earnedGrade;
};
```

8.4.1 Declaraciones de clase

Cada declaración de clase comienza con la palabra clase `class`, el nombre de la clase, una extensión opcional y declaración de clave entre paréntesis, y una lista de atributos, métodos y relaciones encerrados en llaves.

8.4.2 Extensión

La **extensión** (extent) de una clase se puede considerar como el conjunto de instancias objeto para dicha clase, que se almacenan en la base de datos en un momento dado. Con los términos del capítulo 2, la clase en sí es parte de la intensión de la base de datos, mientras que la extensión es parte de la extensión de la base de datos. La clase corresponde a un esquema de relación en los sistemas relacionales, y la extensión corresponde a la tabla almacenada que existe en un momento dado, esto es, los registros reales. En términos E-R, la clase es como el tipo entidad, y la extensión es el conjunto de instancias de entidad. La extensión se puede considerar como el nombre del archivo donde se almacenan los objetos en una clase. Es buena práctica elegir un nombre para la extensión que sea ligeramente diferente al nombre de la clase. Para la clase `Person`, la extensión se llamará `people`. Los paréntesis también pueden contener el nombre de la clave, que se discute en una sección posterior.

8.4.3 Atributos

Los atributos para una clase se mencionan, junto con sus tipos de datos, dentro de llaves. En ODL se permite una gran variedad de tipos, incluidos tipos simples y tipos más complejos que pueden construirse a partir de ellos. Los **tipos atómicos** incluyen los tipos entero, flotante, carácter, cadena, booleano y enumerado. Los **tipos enumerados** se identifican por la palabra clave enum, el nombre del tipo definido por el usuario, una lista de literales para el tipo dentro de llaves, y el nombre del atributo con dicho tipo. Por ejemplo, al atributo `rank` en `Faculty` se le asignó un tipo enumerado llamado `FacultyRank`. También puede tener **tipos estructurados**, identificados por la palabra clave `Struct`, el nombre del tipo y llaves que contienen cada atributo con su tipo de datos. En la definición de clase `Person`, `Addr` es un tipo estructurado que se usó para el atributo `address`. Dado que se nombró el tipo se le puede usar de nuevo para otras clases, pero se debe identificar la clase en que se definió, usando la forma `Person::Addr`, llamada el **nombre de ámbito** (scoped name), que consiste en el nombre de clase, un par de dos puntos y el nombre de estructura. Esta forma de ámbito se usa para referirse a cualquier propiedad en una clase diferente. Además de `Struct` existen constructores tipo, a saber `Set`, `List`, `Array`, `Bag` y `Dictionary`.

- Un **set** (conjunto) consiste de un número finito de valores de un tipo de datos, que se especifica en paréntesis angulados, con la notación `Set<`*`nombre_tipo`*`>`. Se puede definir un atributo como poseedor de un conjunto de valores, o se pueden usar conjuntos en relaciones. Por ejemplo, en la clase `Student` se identifica una relación, `takesClass`, entre cada instancia `student` y un conjunto de instancias tipo `classSection`, usando `Set<classSection>`. Esta relación conecta a cada estudiante con las clases que toma.
- El constructor **List** se usa para denotar una lista finita de elementos de un solo tipo, escrito List<*tipo_datos*>. Por ejemplo, dado que ya se definió el tipo estructurado `Addr` se puede usar `List<Person::Addr>` para denotar una lista de valores tipo dirección para alguna otra clase. Desde luego, el tipo para una lista también podría ser un tipo atómico, como en List<integer>.

- Un **array** consiste en un conjunto de elementos todos del mismo tipo, con un índice que indica posición de cada elemento. El constructor array requiere la especificación del tipo de datos y el número de elementos, como en Array<float, 5>.
- Una **bag** (bolsa) o multiconjunto es similar a un set, excepto que permite valores duplicados para un solo tipo de datos. Se le denota Bag<*tipo_datos*>, como en Bag<string>.
- El constructor de tipo **dictionary** (diccionario) tiene la forma Dictionary <K,V>, donde K y V son algunos tipos de datos. Se usa para construir pares de valores, <k, v>, donde k es un tipo clave y v es algún tipo rango, y su propósito es proporcionar un medio eficiente para encontrar el valor del tipo rango para un valor dado del tipo clave.

8.4.4 Relaciones

Las relaciones representan conexiones entre instancias objeto. Por ejemplo, los miembros del personal docente se relacionan con las secciones clase porque las imparten, los estudiantes se relacionan con las clases que toman, los departamentos se relacionan con los docentes que están empleados en ellos, y así por el estilo. El modelo orientado a objetos representa estas conexiones usando **referencias**. En ODL, la descripción de una relación implica que el sistema almacena y mantiene tales referencias. Por ejemplo, existe una relación llamada `takesClass` definida por la clase `Student` en esta línea:

```
relationship Set<ClassSection> takesClass Inverse ClassSection::
hasStudent;
```

La primera parte de la declaración, `relationship Set<ClassSection> takesClass`, dice que cada instancia `Student` puede contener un conjunto de referencias, llamada `takesClass`, a instancias `ClassSection`. La palabra clave `Set` muestra que ésta es una relación "muchos" para cada estudiante; esto es: una instancia `Student` puede relacionarse con un conjunto de instancias `ClassSection`, no sólo a una. La relación `takesClass` se puede ver como un camino, de modo que, dada una instancia `Student`, se puede encontrar cuáles clases toma el estudiante. También se espera que sea capaz de voltear la consulta y, dada una instancia `ClassSection`, preguntar cuáles estudiantes están en la clase. Se ve que existe tal relación inversa, `hasStudent`, definida por `ClassSection`. La frase

```
Inverse ClassSection::hasStudent;
```

significa que cada objeto `ClassSection` contiene referencias a un conjunto de las correspondientes instancias `Student`. Note que, además de especificar en la clase `Student` que existe el inverso, también especifica esta relación en la clase `ClassSection`, en esta línea:

```
relationship Set<Student> hasStudent Inverse Student::takesClass;
```

La conexión entre `takesClass` y `hasStudent` es que, si una clase aparece en el conjunto de referencias `takesClass` para un estudiante, dicho estudiante aparecería en el conjunto de referencias `hasStudent` para la clase. En consecuencia, estas dos relaciones son inversas una de otra, como se indicó. Note que, cuando se identifica la relación inversa, normalmente se usa la forma de ámbito para su nombre

```
Inverse className::relationshipName,
```

pues la relación inversa por lo general está en otra definición de clase. ODL permite que las relaciones tengan o no inversos. Las que no tienen inversos se describen como **unidireccionales**, mientras que aquellas con inversos son **bidireccionales**. Las relaciones también tienen **cardinalidad**, que puede ser uno a uno, uno a muchos, muchos a uno o muchos a muchos. La relación bidireccional `Student-classSection` es muchos a muchos, y ambas especificaciones de relación incluyen un **set** de referencias. Si una relación es uno a muchos, como `Department-Faculty`, la relación en el lado "uno", `Department`, es un

conjunto. En este caso, se tiene `Set<Faculty>,` pues cada instancia `Department` tiene referencias a muchas instancias `Faculty`, como se muestra en la línea.

```
relationship Set <Faculty>hasFaculty Inverse Faculty::belongsTo;
```

Sin embargo, la relación en el lado `Faculty`,

```
relationship Department belongsTo Inverse Department::hasFaculty
```

especifica sólo `Department` (no `Set<Department>`), pues un registro de personal docente se refiere sólo a un departamento.

8.4.5 Métodos

Las definiciones de clase ODL también pueden contener declaraciones para métodos en la clase. Los ejemplos son los métodos `getName( )` y `setName( )` mencionados para la clase `Person`. Un método es una función o procedimiento que se puede realizar sobre los miembros de la clase. Las declaraciones de método en ODL simplemente especifican la **firma** (signature), que es el nombre del método, el tipo de devolución y el número y tipo de parámetros, que se pueden identificar como IN, OUT o IN/OUT, como se describió antes. Los métodos miembro de clase se aplican a una instancia de la clase, que se refiere como *self* en el código para el método. El código real para el método no es parte del ODL, sino que se escribe en uno de los lenguajes huésped. Los métodos se pueden **sobrescribir**, lo que significa que el mismo método de nombre se puede usar para diferentes clases, y tendrá distinto código para ellos. Dos métodos para la misma clase también pueden tener el mismo nombre, pero si sus firmas son diferentes, se considerarán métodos distintos.

8.4.6 Clases y herencia

Una subclase se identifica mediante la palabra clave `extends` y el nombre de su superclase después del nombre de subclase. La subclase hereda todos los atributos, relaciones y métodos de la superclase, y puede tener algunas propiedades adicionales propias que aparecen en la definición. Por ejemplo, en la figura 8.6, `Faculty` extiende a `Person`, así que hereda

```
name, address, phone, getName( ) and setName( )
```

de `Person`, pero agrega algunas propiedades adicionales. Si la subclase tiene más de una superclase, se agrega dos puntos y el nombre de la segunda superclase inmediatamente después del nombre de la primera superclase. La segunda superclase debe ser una **interfaz**, una definición de clase sin una extensión asociada.

8.4.7 Relaciones *n*-arias y relaciones M:N con atributos

Las relaciones en ODL son binarias, pero si debe modelar una relación ternaria o de orden superior, se puede hacer al crear una clase para la relación misma. La definición de clase incluiría tres o más relaciones que conectan la nueva clase a las clases originalmente relacionadas. Por ejemplo, en la sección 3.5.1 se describió una relación ternaria que conectó una clase, instructor y texto, que se muestra en la figura 3.12 como `Faculty-Class-Textbook`. En el modelo orientado a objetos se definiría la clase para la relación, que se llamará `BookOrder`, como se muestra en el siguiente bosquejo:

```
Class BookOrder{
relationship Faculty teacher Inverse . . .
relationship Book bookFor Inverse . . .
relationship ClassSection uses Inverse . . .
}
```

Si la relación tiene algún atributo descriptivo, también se mencionarían. Note que esta técnica es similar a la forma como se trataron las relaciones de orden superior en el modelo relacional, donde se crearon tablas para tales relaciones.

Las relaciones binarias muchos a muchos con atributos descriptivos no se pueden manejar con la solución usual de hacer a la relación un conjunto en ambas direcciones, pues esto no deja lugar para atributos descriptivos. Con referencia al diagrama UML de la figura 8.5 se ve que `grade` es un ejemplo de atributo descriptivo. Se usa la misma solución que para relaciones *n*-arias, al establecer una clase para la relación. Los atributos descriptivos se colocan como atributos de la nueva clase, y se definen dos relaciones uno a muchos entre la nueva clase y las dos clases originales. Esto se hizo en la figura 8.6, al definir la clase `Grade`. Note que también se mantiene la relación muchos a muchos entre `Student` y `Class-Section` que representa inscripción en la sección al definir relaciones set en ambos.

8.4.8 Claves

Las claves son opcionales en ODL, porque el identificador de objeto único (OID) que automáticamente se da a cada instancia permite al sistema separar las instancias. Sin embargo, el diseñador puede elegir identificar también cualquier clave candidata. Esto se hace al principio de la declaración de clase dentro de los mismos paréntesis que la declaración de extensión. Una clave puede ser un atributo sencillo o uno compuesto, que se identifica al colocar paréntesis alrededor de los nombres de los atributos componentes. Por ejemplo, se podría escribir:

```
class Faculty
(extent Fac
key facId, socSecNo, (name, department)
```

Esto significa que `facId` es una clave, `socSecNo` es una clave, y la combinación de `name` y `department` es una clave. Las claves no están restringidas a atributos. Se puede mencionar una relación o incluso un método como una clave, siempre que proporcione valores únicos. Los conjuntos de entidades débiles con frecuencia se representan como subclases en las que la relación a la superclase es parte de la clave, con el discriminador como otra parte. Por ejemplo, para el conjunto de entidades débil `Evaluation` se escribió

```
class Evaluation
(extent Evaluations
key (date, IsEvalOf))
{. . .
relationship Faculty isEvalOf Inverse Faculty::hasEvaluation;
. . .
}
```

Esto significa que la clave compuesta consiste en la fecha de la evaluación y el docente relacionado.

8.5 Lenguaje de consulta de objetos

El lenguaje de consulta de objetos (object query language, OQL) usa una sintaxis que es similar a SQL, pero opera sobre objetos, no tablas. La forma para las consultas es:

SELECT *lista_expresiones*
FROM *lista_variables*
WHERE *condición*;

Aunque la sintaxis es similar a SQL, existen algunas diferencias importantes. La lista de expresiones puede contener los nombres de atributos de objetos, identificados con el uso de la notación punto, como en:

```
SELECT s.stuId, s.credits
FROM students s;
```

El resultado será los valores de `stuId` y `credits` para cada una de las instancias en la extensión estudiantes. Además de atributos es posible usar métodos en la lista de expresiones. Esto recupera el resultado de aplicar el método. Por ejemplo, podría escribir

```
SELECT p.getName( )
FROM people p;
```

que devolverá el valor del atributo `name`, siempre que el método `getName( )` se haya escrito correctamente para hacer eso. También se puede usar una relación en la lista de expresiones. Ésta recupera el objeto o conjunto de objetos relacionados con el objeto "que llama" a través de la relación. Por ejemplo

```
SELECT s.stuId, s.takesClass
FROM students s
WHERE s.stuId = 'S999';
```

recupera el conjunto de clases que toma el estudiante S999, así como el `stuId`, S999.

La lista de variables en la línea FROM es similar a definir un alias en SQL. Por lo general se menciona el nombre de una extensión, como estudiantes o personas, y un identificador para el nombre de la variable, como s o p, como se vio líneas antes. La variable es en realidad una **variable iterador**, que varía sobre la extensión. Existen formas alternativas para declarar una variable iterador, como

```
FROM students s
FROM s in students
FROM students as s
```

En la cláusula WHERE, está restringido a una expresión booleana que tenga constantes y variables definidas en la cláusula FROM. Puede usar <, <=, >, >=, !=, AND, OR y NOT en la expresión. Como en SQL, OQL no elimina duplicados, así que devuelve una bag (multiconjunto) en lugar de un set. Si quiere eliminar duplicados puede agregar la palabra clave DISTINCT como en SQL, lo que produce un set. Opcionalmente puede agregar una cláusula ORDER BY, como en SQL.

- **Ejemplo 1.** Encuentre el ID, nombre y especialidad de todos los estudiantes de pregrado que tengan entre 60 y 90 créditos, en orden por nombre.

```
OQL:
SELECT s.stuId, s.name, s.major
FROM students as s
WHERE s.credits >=60 AND s.credits<=90
ORDER BY s.name;
```

- **Ejemplo 2.** Puede usar una relación para encontrar registros a través de referencias. Por ejemplo, para encontrar los nombres de todos los docentes en el departamento de biología, escriba la siguiente consulta:

```
OQL:
SELECT f.name
FROM departments as d, d.hasFaculty as f
WHERE d.deptName = 'Biology';
```

La primera parte de la cláusula FROM dice que d es una variable iterador que varía sobre los objetos en la extensión `departments`, y la segunda parte dice que, para cada uno de

los objetos `department`, el conjunto de docentes identificados por la relación `hasFaculty` para dicho departamento se identificará con la variable iterador f. Esto es en esencia establecer un bucle anidado que toma d en cada departamento a la vez, y f toma cada docente en dicho departamento. La línea WHERE restringe los resultados al departamento de biología, y la línea SELECT despliega sólo el nombre del docente.

- **Ejemplo 3.** En OQL también se pueden realizar subconsultas. Por ejemplo, la consulta anterior se puede descomponer. Primero puede encontrar el objeto departamento de biología al escribir:

```
OQL:
SELECT d
FROM d in departments
WHERE d.deptName = 'Biology';
```

Puede usar esta subconsulta para proporcionar el valor para una variable iterador, b. Luego se declara una variable iterador, f, para el conjunto relación `hasFaculty` para b. Toda la consulta se expresa del modo siguiente:

```
OQL:
SELECT f.name
FROM (SELECT d
   FROM d in departments
   WHERE d.deptName = 'Biology') as b,
     b.hasFaculty as f;
```

- **Ejemplo 4.** Es posible definir estructuras para resultados devueltos justo en una consulta. Por ejemplo, puede encontrar el nombre, categoría y salario de todos los docentes en el departamento de inglés y poner los resultados en una estructura, en orden descendente por salario.

```
OQL:
SELECT struct(name: f.name, rank: f.rank, salary: f.salary)
FROM Faculty as f
WHERE f.belongsTo = 'English'
ORDER BY salary DESC;
```

- **Ejemplo 5.** Como SQL, OQL incluye los operadores COUNT, SUM, AVG, MAX y MIN, pero se usan de manera diferente. En OQL, el operador se aplica a una colección en lugar de a una columna. COUNT regresa un entero, mientras que la devolución de los otros operadores tiene el mismo tipo que la colección. Los operadores se pueden usar en los resultados de una consulta. Por ejemplo, para encontrar el salario promedio de los docentes en el departamento de historia, la consulta OQL se escribe del modo siguiente:

```
OQL:
AVG(SELECT f.salary
   FROM fac as f
   WHERE f.belongsTo = 'History');
```

- **Ejemplo 6.** Para encontrar el número de estudiantes de pregrado con especialidad en ciencias de la computación, puede usar la función COUNT.

```
OQL:
COUNT((SELECT u
   FROM students as u
   WHERE u.majorsIn = 'Computer Science');
```

Para encontrar todas las secciones de clase que tengan más de 30 estudiantes en ellas se puede poner el COUNT en la línea WHERE.

```
OQL:
SELECT  s
FROM    sections as s
WHERE COUNT(s.hasStudents >30);
```

- **Ejemplo 7.** Las operaciones de conjuntos UNION, INTERSECTION y EXCEPT (diferencia) se usan como en SQL. Por ejemplo, para encontrar a todos los estudiantes cuya especialidad o programa es ciencias de la computación, se escribe una unión.

```
OQL:
(SELECT u
FROM undergraduates as u
WHERE u.major = 'Computer Science')
UNION
(SELECT g
FROM graduates as g
WHERE g.program = 'Computer Science');
```

Se eligieron ejemplos que son similares a SQL. OQL tiene muchas formas y capacidades adicionales que no se ilustraron aquí.

8.6 Desarrollo de una base de datos OO

El proceso de crear una base de datos orientada a objetos con el uso de un OODBMS como Objectivity tiene la intención de ser una extensión natural del desarrollo de aplicaciones en un entorno de programación orientada a objetos. Como se discutió anteriormente, existen vinculaciones de lenguaje especificadas en el estándar ODMG para C++, Java y Samlltalk. La diferencia entre usar objetos en programas escritos en estos lenguajes y objetos de base de datos es la persistencia de los objetos de la base de datos. Los proveedores que soportan el estándar ODMG proporcionan facilidades para hacer persistentes los objetos de programa, y para ofrecer acceso a objetos de base de datos para manipulación dentro de programas. En la figura 8.7 se muestra un proceso de desarrollo típico que usa, por ejemplo, C++ como el lenguaje huésped. El diseñador de la base de datos define el esquema usando un lenguaje de definición de datos como ODL o el mismo C++. Las definiciones de clase en el esquema son esencialmente estándar C++ que se extendieron para proporcionar persistencia y soportar relaciones entre objetos, así como herencia. La persistencia se proporciona al hacer que todos los objetos que serán persistentes heredan de una clase proporcionada por el OODBMS sólo para dicho propósito. Por ejemplo, Objectivity proporciona la clase ooOdb para C++. Para hacer permanente una clase, el programador debe especificar que extiende ooOdb. Para la clase `Person` se escribiría en C++.

```
class Person: public ooOjb
```

y luego continuaría con la definición de clase. La persistencia la heredaría `Faculty`, `Student` y también sus subclases. Si el esquema se escribe en C++, cualquier puntero utilizado se debe sustituir mediante referencias, y el archivo encabezado (header) se debe guardar con la extensión .ddl.

Luego se usa el procesador DDL para procesar los archivos de esquema. Esto crea el esquema de base de datos en disco y genera archivos fuente de modelo de datos. El código de aplicación para los objetos que pueblan la base de datos se puede escribir en un programa C++ separado. En C++, cada clase tiene al menos un método, llamado **constructor**, que genera nuevas instancias objeto. Los objetos a almacenar en la base de datos se crean dentro de la aplicación C++ con el uso de constructores. Los valores de los atributos de las instan-

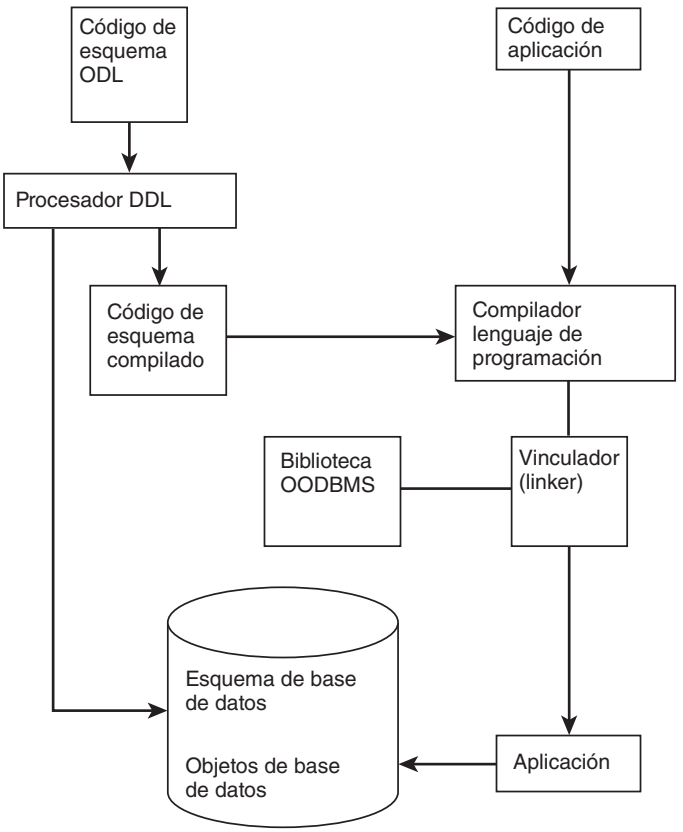


FIGURA 8.7
Proceso de desarrollo típico de una base de datos orientada a objetos

cias objeto se pueden establecer o mediante los constructores usando parámetros, o mediante métodos como `setName( )`. El programa también puede manipular objetos recuperados usando enunciados OQL SELECT para actualizar atributos u otras propiedades. Las clases C++ también pueden tener **destructores** cuya función es destruir objetos que ya no se necesiten más. Éstos se pueden usar para borrar objetos de base de datos. Cuando el programa se compila, el compilador C++ usa los archivos de esquema creados por el procesador DDL. Entonces el resultado se liga con la biblioteca de tiempo de corrido de la base de datos para producir el código ejecutable. El proceso es similar en Java y Smalltalk.

8.7 Resumen del capítulo

Las bases de datos orientadas a objetos permiten a los diseñadores de bases de datos y desarrolladores expresar estructuras de datos, operaciones y relaciones complejas de manera más natural que las bases de datos relacionales tradicionales. Los lenguajes de programación orientados a objetos, incluidos C++, Java y Samlltalk se extendieron para permitir almacenamiento **persistente** de los objetos que crean y manipulan. Los conceptos fundamentales del paradigma orientado a objetos involucra una comprensión de **objetos**, **métodos**, **encapsulado**, **clases** y **herencia**. En una base de datos orientada a objetos, las definiciones de clase corresponden a definiciones de clase para un lenguaje de programación orientado a objetos. El conjunto de instancias objeto en una clase se llama **extensión** de la clase. A cada instancia objeto se le da un identificador de objeto único (**OID**) que es independiente de cualquiera de los valores del objeto, y que es invariable. Los OID se usan para establecer relaciones entre instancias objeto. Los **atributos** en una clase pueden ser

tipos atómicos, tipos **enumerados**, tipos **estructurados** o tipos **colección**, que incluyen tipos **set**, **bag**, **array**, **list** y **dictionary**.

Los diagramas UML son un método útil para representar clases y relaciones, y coinciden bien con el modelo de datos orientados a objetos. A partir de un diagrama UML es relativamente fácil traducir el diseño en definiciones de clase que correspondan de manera directa a los ítems en el diagrama, incluidas las relaciones entre clases. Una clase se representa mediante un rectángulo que tiene tres partes: el nombre de la clase, los atributos y los métodos. Las relaciones se muestran mediante líneas, con indicadores de multiplicidad de la forma *mín..máx*, pero colocados en el lado del opuesto al que se utiliza en los diagramas EE-R. Las relaciones entre clases, llamadas **asociaciones**, pueden ser unidireccionales o bidireccionales, que se representan mediante la definición de un inverso para la relación. Pueden especificar una relación "uno" con un miembro de una clase, o una "muchos", que se representa al especificar Set antes del nombre de la clase relacionada. La **agregación** es un tipo especial de relación que conecta un todo a sus partes. La **generalización** se indica mediante líneas que conectan las subclases a la superclase, con un triángulo sobre la línea en la superclase.

El Object Data Management Group (**ODMG**) estableció estándares para bases de datos orientadas a objetos, incluidos estándares para Lenguaje de definición de objetos (**ODL**) y Lenguaje de consulta de objetos (**OQL**). En ODL se proporcionan declaraciones de clase, incluidos el nombre de la clase, cualquier herencia, extensión y claves, y descripciones de los atributos, relaciones y métodos. OQL usa una sintaxis que es similar a SQL, pero tiene mayor flexibilidad para lidiar con una variedad de estructuras. Debido a los estándares ODMG para extensiones de lenguaje orientado a objetos, el proceso de desarrollar una base de datos usando un OODBMS está estrechamente vinculado con aplicaciones en dichos lenguajes.

Ejercicios

8.1 Defina los siguientes términos:

a. persistencia
b. encapsulado
c. interfaz
d. método
e. extensión
f. jerarquía de clase
g. OID
h. asociación
i. agregación
j. indicador de multiplicidad
k. asociación reflexiva
l. especialización obligatoria
m. relación inversa
n. firma de un método
o. sobrecarga
p. interfaz

q. método constructor

r. método destructor

s. iterador

8.2 Desarrolle un diagrama de clase UML para la aplicación del grupo dental descrito en el ejercicio 7.4.

8.3 Escriba un esquema ODL que corresponda al diagrama de clase UML que desarrolló en el ejercicio 8.2.

8.4 Desarrolle un diagrama de clase UML para la aplicación de la firma de diseño de interiores descrito en el ejercicio 7.6.

8.5 Escriba un esquema ODL que corresponda con el diagrama de clase UML que desarrolló en el ejercicio 8.4.

8.6 a. Modifique el diagrama UML que se muestra en la figura 8.5 al agregar métodos adicionales.

b. Modifique el esquema ODL que se muestra en la figura 8.6 para incluir los nuevos métodos que agregó.

Ejercicios de laboratorio

Creación de diagramas UML usando una herramienta de diagramación Si tiene una herramienta de diagramación, como Visio, que soporte diagramas UML, haga los ejercicios 8.2 y 8.4 usando la herramienta de diagramación.

PROYECTO DE MUESTRA: CREACIÓN DE UN DIAGRAMA UML PARA LA GALERÍA DE ARTE Y CONVERSIÓN DEL DIAGRAMA A UN ESQUEMA DE BASE DE DATOS ORIENTADO A OBJETOS

- Paso 8.1. Cree un diagrama UML para la Galería de Arte.

La figura 8.8 muestra un bosquejo de un diagrama UML para el proyecto de muestra. Para mantener simple el diagrama, no se muestran los atributos de clase ni los métodos. Se creó una clase `Person`, con `Collector`, `Artist`, `Customer` y `Salesperson` como subclases. Debido a la facilidad con la que se pueden definir las relaciones, se agregaron relaciones entre `Artist` y `Collection`, y entre `Artist` y `Show`, dado que una colección o una exposición pueden presentar a un artista particular. Se eligió hacer todas las relaciones bidireccionales, al proporcionar inversos para cada una.

- Paso 8.2. Convierta el diagrama UML a un esquema de base de datos orientada a objetos.

La figura 8.9 da el ODL para una base de datos que corresponde al diagrama UML. Se incluyen algunos métodos para la clase `Person`. Se deben agregar métodos para las clases restantes. Note que no se representa la unión de `Collector` y `Artist` en `Owner`, pues las uniones son difíciles de representar. En vez de ello se agregó un método a la clase `Artwork`, llamado `OwnerIsArtist`, que devolverá un valor booleano. Prueba si la referencia a `Collector` es nula y, si lo es, devuelve verdadero, lo que indica que el artista posee la obra. Sin embargo, si se quiere representar la unión podría usar la misma solución que se utilizó para el caso objeto-relacional.

FIGURA 8.8

Diagrama UML para la Galería de Arte

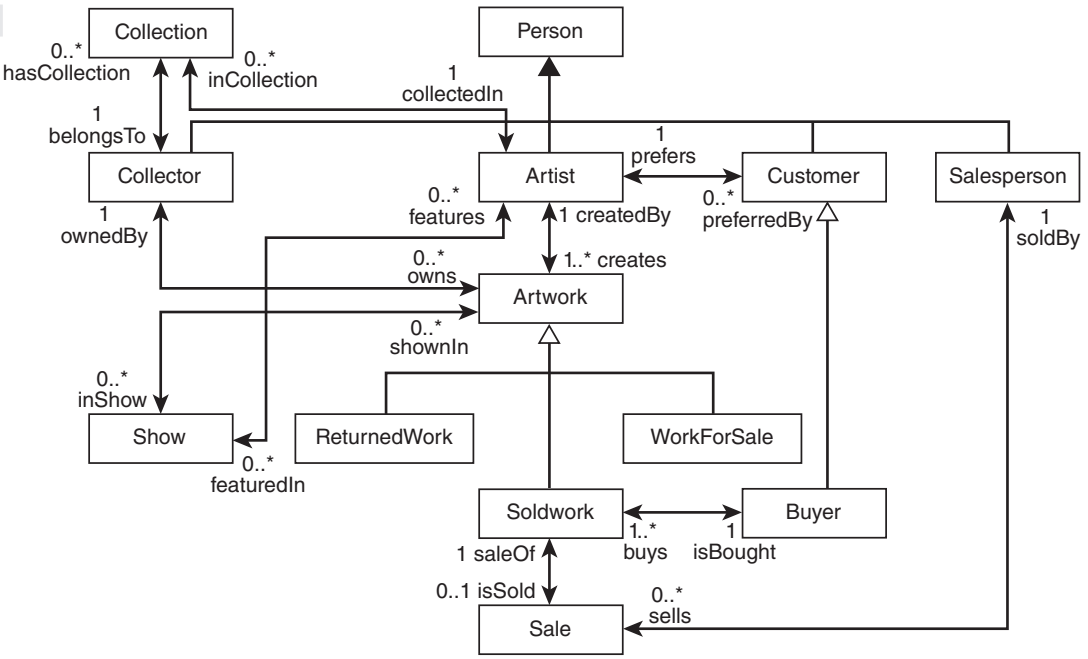


FIGURA 8.9

ODL para el esquema de la Galería de Arte

```
class Person(
extent people
key pId)
{   attribute pId int;
    attribute Struct NameType(string first, string last) name;
    attribute Struct AddressType(string street, sting city, string state, string zip) address;
    attribute Struct PhoneType (string areaCode, string telephoneNumber) phone;
    int getPId( );
    void setPId(int newId);
    string getName( );
    void setName(string newName);
    string getAddress( );
    void setAddress(AddressType newAddress);
    string getPhone( );
    void setPhone(PhoneType newPhone);
};
class Artist extends Person
{   attribute Struct DateType(int day, int month, int year) interviewDate;
    attribute NameType interviewerName;
    attribute real(10,2) salesLastYear;
    attribute real(10,2) salesYearToDate;
    attribute string socialSecurityNumber;
    attribute string usualMedium;
    attribute string usualStyle;
    attribute string usualType;
    DateType getinterviewDate();
    void setinterviewDate(DateType newartistinterviewDate);
    NameType getinterviewerName();
    void setinterviewerName(NameType newartistinterviewerName);
    real getsalesLastYear();
```

(*continúa*)

```
    void setsalesLastYear(real artistlastyearsalesAmount);
    real get salesYearToDate();
    void setsalesYearToDate(real artistsalesYTD);
    void updatesalesYearToDate(real artistNewSalesAmount);
    string getsocialSecurityNumber();
    void setsocialSecurityNumber(string artistnewSSN);
    string getusualMedium();
    void setusualMedium(string artistusualMedium);
    string getusualStyle();
    void setusualStyle(string artistusualStyle);
    string getusualType();
    void setusualType(string artistusualType);
    relationship Set<Artwork> creates Inverse Artwork::createdBy;
    relationship Set<Customer> preferredBy Inverse Customer::prefers;
    relationship Set<Show> featuredIn Inverse Show::features;
    relationship Set<Collection> inCollection Inverse Collection::collectedIn;
};

class Collector extends Person
{   attribute string socialSecurityNumber;
    attribute Artist::DateType interviewDate;
    attribute NameType interviewerName;
    attribute real(10,2) salesLastYear;
    attribute real(10,2) salesYearToDate;
    string getsocialSecurityNumber();
    void setsocialSecurityNumber(string collectornewSSN);
    Artist::DateType getinterviewDate();
    void setinterviewDate(Artist::DateType newcollectorinterviewDate);
    NameType getinterviewerName();
    void setinterviewerName(NameType newcollectorinterviewerName);
    real getsalesLastYear();
    void setsalesLastYear(real collectorsalesLastYearAmount);
    real getsalesYearToDate();
    void setsalesYearToDate(real collectorsalesYTDAmount);
    void updatesalesYearToDate(real collectornewSalesAmount);
    relationship Set<Collection> hasCollection Inverse Collection::belongsTo;
    relationship Set<Artwork> owns Inverse Artwork::ownedBy;
};

class Collection
{   attribute string collectionMedium;
    attribute string collectionStyle;
    attribute string collectionType;
    string getcollectionMedium();
    void setcollectionMedium(string medium);
    string getcollectionStyle();
    void setcollectionStyle(string style);
    string getcollectionType();
    void setcollectionType(string type);
    relationship Collector belongsTo Inverse Collector::hasCollection;
    relationship Artist collectedIn Inverse Artist::inCollection;
};
```

(*continúa*)

FIGURA 8.9
Continuación

FIGURA 8.9
Continuación

```
class Artwork
{   attribute string artworkId;
    attribute string workTitle;
    attribute real(6,2) askingPrice;
    attribute Artist::DateType dateListed;
    attribute Artist::DateType dateShown;
    attribute string status;
    attribute string workMedium;
    attribute string workSize;
    attribute string workStyle;
    attribute string workType;
    attribute int workYearCompleted;
    string getartworkId();
    void setartworkId(string newartworkId);
    string getartworkTitle();
    void setartworkTitle(string newartworkTitle);
    real getaskingPrice();
    void setaskingPrice(real newaskingPrice);
    Artist::DateType getdateListed();
    void setdateListed(Artist::DateType newdateListed);
    Artist::DateType getdateShown();
    void setdateShown(Artist::DateType newdateShown);
    string getstatus();
    void setstatus(string newstatus);
    string getworkMedium();
    void setstworkMedium(string newmedium);
    string getworkSize();
    void setstworkSize(string newsize);
    string getworkStyle();
    void setstworkStyle(string newStyle);
    string getworkType();
    void setstworkType(string newType);
    int getworkYearCompleted();
    void setworkYearCompleted(int newworkYearCompleted);
    boolean artistIsOwner( )//method to determine if the work is owned by the artist;
    relationship Artist createdBy Inverse Artist::creates;
    relationship Set<Show> inShow Inverse Show::shownIn;
    relationship Collector ownedBy Inverse Collector::owns;
};

class SoldWork extends Artwork
{   attribute Artist::DateType dateSold;
    Artist::DateType getdateSold();
    void setdateSold(DateType newdateSold);
    relationship Sale isSold Inverse Sale::saleOf;
    relationship Buyer isBought Inverse Buyer::buys;
};

class ReturnedWork extends Artwork
{   attribute Artist::DateType dateReturned;
    Artist::DateType getdateReturned();
    void setdateReturned(Artist::DateType newdateReturned);
};
```

(*continúa*)

```
class WorkForSale extends Artwork
{   attribute string location;
    string getlocation();
    void setlocation(string newlocation);
};
class Customer extends Person
{   attribute Artist::DateType dateFilledIn;
    attribute string preferredMedium;
    attribute string preferredStyle;
    attribute string preferredType;
    Artist::DateType getdateFilledIn();
    void setdateFilledIn(Artist::DateType newdateFilledIn);
    string getpreferredMedium();
    void setpreferredMedium(string newpreferredMedium);
    string getpreferredStyle();
    void setpreferredStyle(string newpreferredStyle);
    string getpreferredType();
    void setpreferredType(string newpreferredType);
    relationship Artist prefers Inverse Artist::preferredBy;
};
class Buyer extends Customer
{   attribute real(8,2) purchasesLastYear
    attribute real(8,2) purchasesYearToDate;
    real getpurchasesLastYear();
    void setpurchasesLastYear(real purchasesLastYearAmount);
    real getpurchasesYearToDate();
    void setpurchasesYearToDate(real purchasesYearToDateAmount);
    void updatepurchasesYearToDate(real amountNewPurchases);
    relationship SoldWork buys Inverse SoldWork::isBought;
};
class Show
{   attribute string showTitle;
    attribute string showClosingDate;
    attribute string showTheme;
    attribute string showOpeningDate;
    string getshowTitle();
    void setshowTitle(string newTitle);
    string getshowclosingDate();
    void setshowclosingDate(string newTitle);
    string getshowTheme();
    void setshowTheme(string newTitle);
    string getshowopeningDate();
    void setshowopeningDate(string newTitle);
    relationship Artist features Inverse Artist::featuredIn;
    relationship Set<Artwork> shownIn Inverse Artwork::inShow;
};
```

(*continúa*)

FIGURA 8.9
Continuación

FIGURA 8.9
Continuación

```
class Sale
{   attribute string invoiceNumber;
    attribute real(8,2) amountRemittedToOwner;
    attribute Artist::DateType saleDate;
    attribute real(8,2) salePrice;
    attribute real(6,2) saleTax;
    string getinvoiceNumber();
    void setinvoiceNumber(string newinvoiceNumber);
    real get amountremittedToOwner();
    void setamountremittedToOwner(real newamountRemitted);
    Artist::DateType getsaleDate();
    void setsaleDate(Artist::DateType newsaleDate);
    real getsalePrice();
    void setsalePrice(real newsalePrice);
    real getsaleTax();
    void setsaleTax(real newtaxAmount);
    relationship SoldWork saleOf Inverse SoldWork::isSold;
    relationship Salesperson soldBy Inverse Salesperson::sells;
};

class Salesperson extends Person
{   attribute string socialSecurityNumber;
    string getsocialSecurityNumber();
    void setsocialSecurityNumber(string newSSN);
    relationship Set<Sale> sells Inverse Sale::soldBy;
};
```

PROYECTOS ESTUDIANTILES: DIBUJE UN DIAGRAMA UML Y DISEÑE UN MODELO DE BASE DE DATOS ORIENTADO A OBJETOS

- **8.1.** Cree un diagrama UML para el proyecto estudiantil.
- **8.2.** Convierta el diagrama UML en un modelo de base de datos orientado a objetos y escriba el ODL para el esquema.

CAPÍTULO 9

Introducción a la seguridad de las bases de datos

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- El concepto de la seguridad de las bases de datos
- La relación entre privacidad y seguridad
- Ejemplos de amenazas accidentales o deliberadas para la seguridad
- Algunas medidas físicas de seguridad
- El significado de la autentificación del usuario
- El significado de la autorización del usuario
- Cómo puede representarse el control del acceso
- Cómo funcionan las vistas como mecanismo de seguridad
- El propósito del registro de seguridad y procedimiento de auditoría
- Cómo y por qué se ejecuta el encriptado de datos
- Cómo se refuerza la seguridad en ciertos sistemas

9.1 Temas de la seguridad en las bases de datos

Seguridad en las bases de datos significa proteger las bases de datos del acceso, modificación o destrucción no autorizados. Como la base de datos representa un recurso corporativo esencial, la seguridad es una meta importante. Además de la necesidad de preservar y proteger los datos para el funcionamiento continuo de la organización, los diseñadores de las bases de datos tienen la responsabilidad de proteger la privacidad de los individuos de quienes se guardan datos. **Privacidad** es el derecho que tienen los individuos de tener control sobre la información acerca de ellos. Muchos países tienen leyes diseñadas para proteger la privacidad, y toda organización que recabe y almacene información sobre las personas tiene la obligación legal de adoptar políticas que se ajusten a las leyes locales sobre la privacidad. El diseño de la base de datos debe reflejar el compromiso de la organización para proteger el derecho a la privacidad de los individuos, a través de la inclusión sólo de aquellos ítems que la organización tiene derecho a saber. Además, debe garantizarse la privacidad para proteger información almacenada cuya naturaleza sea delicada. Las amenazas a la seguridad ocurren en forma **accidental** o **deliberada**.

9.1.1 Amenazas accidentales para la seguridad

Los siguientes son algunos ejemplos de violaciones accidentales de la seguridad.

- El usuario solicita sin malicia un objeto u operación para los que no debería estar autorizado, y la solicitud se aprueba debido a un fallo en los procedimientos de autorización o porque hay un error en el sistema de administración de la base de datos o sistema operativo.
- Una persona envía por accidente un mensaje que debería enviarse a otro usuario, lo que resulta en el acceso no autorizado al contenido de la base de datos.
- Un error en el sistema de comunicaciones conecta a un usuario a una sesión que pertenece a otro que tiene diferentes privilegios de acceso.
- Por accidente, el sistema operativo sobrescribe, erróneamente, en archivos y destruye parte de la base de datos, busca en los archivos incorrectos, y luego los envía inadvertidamente al usuario, o no borra archivos que deben ser destruidos.

9.1.2 Amenazas deliberadas para la seguridad

Las violaciones deliberadas a la seguridad ocurren cuando un usuario logra de manera intencional acceso no autorizado a la base de datos. Un empleado disgustado que conozca el sistema de cómputo de la organización representa una amenaza enorme a la seguridad. Los espías industriales que buscan información para la competencia también amenazan la seguridad. Hay muchos modos de hacer violaciones deliberadas en la seguridad, entre las que están las siguientes:

- Intervención de líneas de comunicación para interceptar mensajes hacia y desde la base de datos
- Escucha furtiva electrónica para obtener señales de las estaciones de trabajo, impresoras y otros dispositivos dentro de un edificio
- Lectura o copiado de pantallas e impresiones dejadas con descuido por usuarios autorizados
- Suplantación de un usuario autorizado, o con mayor acceso, por medio del uso de su registro y clave

- Escritura de programas con código ilegal para esquivar el sistema de administración de la base de datos y su mecanismo de autorización, con el fin de acceder a los datos directamente desde el sistema operativo
- Escribir programas de aplicaciones con código que efectúa operaciones no autorizadas
- Obtención de información sobre datos ocultos, al consultarlos con inteligencia a la base de datos
- Retirar dispositivos de almacenamiento físico de los equipos de cómputo
- Hacer copias físicas de archivos almacenados sin pasar por el sistema de administración de la base de datos, con lo que se esquivan sus mecanismos de seguridad
- Sobornar, amenazar o influir en los usuarios autorizados, a fin de utilizarlos como agentes para obtener información o dañar la base de datos

9.2 Seguridad física y autentificación del usuario

La seguridad de las bases de datos se implementa mejor si sólo es una parte de un plan más amplio para controlar la seguridad. Este plan debe comenzar con medidas físicas para proteger el edificio, con precauciones especiales para las instalaciones de cómputo. El diseño de un edificio seguro en su planta física está claramente fuera del dominio del diseñador de la base de datos. Sin embargo, el ABD o administrador de base de datos debe tener la capacidad de sugerir medidas para controlar el acceso a las instalaciones de la base de datos. Es frecuente que éstas comiencen en la entrada principal, donde todos los empleados deben ser identificados visualmente por guardias, o con el uso de gafetes, huellas digitales, firmas u otros mecanismos. Para ingresar a las instalaciones de cómputo debe requerirse una identificación adicional. Las medidas físicas de la seguridad deben ampliarse para que cubran cualquier ubicación en la que datos fuera de línea, como los respaldos, también estén almacenados en forma segura.

Como la seguridad física de las estaciones de trabajo puede ser difícil de implementar, los controles de la seguridad de aquéllas requieren la **autentificación** de los usuarios. Autentificación significa la verificación de la identidad del usuario (hacer una comprobación para garantizar que el usuario real es quien dice ser). Por lo general se implementa al nivel del sistema operativo. Cuando el usuario se registra, él o ella ingresa una ID de usuario, cuya validez se revisa. El sistema tiene un perfil del usuario para esa ID, lo que da información sobre el usuario. El perfil normalmente incluye una clave (password) que se supone sólo conoce el usuario. Las claves deben conservarse en secreto y cambiadas con frecuencia. Una precaución elemental de seguridad es que el sistema requiera que las claves se cambien cada mes. Es obvio que el sistema nunca debe desplegar las claves al registrarse, y los perfiles almacenados deben mantenerse seguros, tal vez en forma encriptada. Aunque las claves son el método de autentificación que se usa con más amplitud, no son muy seguras, ya que los usuarios en ocasiones las escriben, escogen palabras que son fáciles de adivinar o las comparten con otros usuarios. En ciertas organizaciones, los usuarios deben insertar tarjetas o llaves cuando se registran. En otras, se examina la voz, huellas digitales, retina, u otras características físicas del usuario. Algunas utilizan un procedimiento de autentificación más allá de una sola clave. Un procedimiento puede consistir en responder una serie de preguntas y requerir de más tiempo y ser más difícil de reproducir que una clave. Aunque la autentificación puede hacerse sólo a nivel del sistema operativo, es posible que se pida otra vez al nivel de la base de datos. Como mínimo, debe pedirse al usuario que proporcione una clave adicional para acceder a la base de datos.

9.3 Autorización

Además de la autentificación, la mayor parte de los sistemas de administración de bases de datos para multiusuarios tiene sus propios subsistemas de seguridad, los cuales proveen la **autorización de usuario**, método por el que se asigna a los usuarios derechos para usar objetos de la base de datos. La mayoría de los sistemas multiusuarios tiene un **lenguaje de autorización** que forma parte del sublenguaje de datos. Por ejemplo, SQL provee comandos de autorización estándar para asegurar privilegios a los usuarios, como se ve en la sección 9.8. El ABD usa el lenguaje de autorización para especificar los derechos del usuario por medio de **reglas de autorización**, que son enunciados que especifican cuáles usuarios tienen acceso a cuál información, y cuáles operaciones les está permitido utilizar sobre qué datos. El mecanismo de autorización está diseñado para proteger la base de datos al impedir a las personas la lectura, actualización o eliminación no autorizada del contenido de la base de datos. Estas restricciones se agregan a los mecanismos de seguridad que provee el sistema operativo. Sin embargo, en un número sorprendentemente grande de casos, los subsistemas de seguridad de la base de datos son mínimos o no se utilizan en forma completa. El diseñador debe reconocer que los datos constituyen un recurso corporativo valioso, e incluir los mecanismos de seguridad disponibles como factor importante en la evaluación alternativa de sistemas de administración de bases de datos, así como desarrollar políticas de seguridad efectivas que utilicen los controles de que dispone el sistema elegido.

9.4 Control del acceso

El **control del acceso** es el medio por el que se implementan las autorizaciones. Controlar el acceso significa asegurarse de que a los datos u otros recursos sólo se accede en las formas autorizadas. Al planear el acceso, el ABD puede usar una **matriz de control del acceso** a la base de datos, como se ilustra en la figura 9.1. Los encabezados de las columnas representan objetos de la base de datos, que pueden ser los nombres de las tablas, vistas, ítems de datos, objetos, módulos u otras categorías, en función del modelo de la base de datos y sistema de administración que se utilice. Las etiquetas de las filas representan individuos, roles, grupos de usuarios o aplicaciones.

El contenido de las celdas especifica el tipo de acceso permitido. Los valores de las entradas también dependen del sistema particular que se utilice, pero la elección por lo general incluye READ, INSERT, UPDATE, DELETE y sus combinaciones. Una vez que la matriz de control de acceso está completa, el ABD debe usar el lenguaje de autorización apropiado para implementarla. Por supuesto, al ABD le está permitido crear y cambiar la estructura de la base de datos y usar el lenguaje de autorización para garantizar o revocar el acceso a otros. Algunos sistemas permiten que el ABD delegue parte de su poder de autorización, en

FIGURA 9.1
Matriz de control del acceso

	OBJECT					
SUBJECT	`Student` table	`StuView1`	`WrapUp` Procedure	`Faculty` table	`Enroll` table	. . .
User U101	read, update	read	execute	read		. . .
User U102		read				. . .
Role Advisor	read	read			read, insert, update, delete	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .

cuyo caso, ciertos usuarios tienen permiso de modificar las estructuras existentes de la base de datos o crear estructuras nuevas y actualizar el estado de los datos. En un ambiente multiusuario, tales cambios tienen consecuencias para los demás usuarios. Como el ABD es con frecuencia el único que tiene un punto de vista amplio de las necesidades de todos los usuarios, es frecuente que no sea conveniente dar esa clase de autorización. En ocasiones, el ABD da a ciertos usuarios el poder de autorizar a otros la ejecución de operaciones sobre la base de datos. Sin embargo, tener muchos de tales "autorizadores" es peligroso en extremo, pues los autorizadores generan a otros que también autorizan, lo que lleva a que la situación se salga de control muy rápido y que el ABD tenga dificultades para revocar las autorizaciones.

9.5 Uso de las vistas para el control del acceso

La vista es un método muy utilizado para implementar el control del acceso. El mecanismo de la vista tiene dos propósitos principales. Es una facilidad para el usuario, pues simplifica y personaliza el modelo externo con el que éste maneja la base de datos, lo que lo libera de las complejidades del modelo subyacente. También es una medida de seguridad, ya que oculta estructuras y datos que el usuario no debería ver. El modelo relacional es un modelo que consiste por completo en vistas o alguna combinación de tablas básicas y vistas relacionales. Una vista relacional se obtiene de tablas básicas por medio de la operación SELECT con el fin de obtener columnas o filas, o mediante el uso de otras operaciones para obtener datos calculados o materializados. Al especificar restricciones en la línea WHERE del enunciado SELECT que se usa para crear vistas, éstas se hacen **dependientes del valor**. La figura 9.2(a) da un ejemplo de vista creada a partir de la tabla `Student` incluyendo sólo los datos de los estudiantes cuya especialidad es CSC. Las vistas **independientes del valor** se crean mediante la especificación de columnas de las tablas básicas y omitir la línea WHERE del enunciado SELECT. En la figura 9.2(b) se presenta un ejemplo de vista de la tabla `Student` que sólo muestra las columnas `stuID`, `stuName` y `major`.

9.6 Registros de seguridad y procedimientos de auditoría

Otra herramienta de seguridad importante es el **registro de seguridad (bitácora)**, que es un diario que registra todos los intentos de violar la seguridad. La violación puede sólo registrarse o enviar un mensaje inmediato al operador o al ABD. Saber de la existencia del registro es un disuasivo en sí mismo. Si el ABD sospecha que los datos están siendo atacados sin que se disparen los registros de la seguridad, es posible que ordene un **procedi-**

```
CREATE VIEW CSCMAJ AS
    SELECT stuId, lastName, firstName, credits
    FROM Student
    WHERE major = 'CSC';
```

FIGURA 9.2(a)
Vista dependiente del valor

```
CREATE VIEW StuView1 AS
    SELECT stuId, lastName, firstName, major
    FROM Student;
```

FIGURA 9.2(b)
Vista independiente del valor

FIGURA 9.3

Procedimiento de auditoría que usa un disparador

```
CREATE OR REPLACE TRIGGER EnrollAuditTrail
    BEFORE UPDATE OF grade ON Enroll
    FOR EACH ROW
    BEGIN
        INSERT INTO EnrollAudit
            VALUES(SYSDATE, USER, :OLD.stuId, :OLD.courseNo, :OLD.grade, :NEW.grade);
    END;
```

miento de auditoría. Este sistema de auditoría registra todos los accesos a la base de datos, mantiene información acerca del usuario que solicitó el acceso, la operación que ejecutó, la estación de trabajo desde la que lo hizo, la hora exacta en que ocurrió, el tipo de datos, su valor anterior y su valor actual, si lo hubiera. De esa manera, la auditoría tiene la capacidad de descubrir el origen de las operaciones sospechosas ejecutadas sobre la base de datos, aun si las hubieran realizado usuarios autorizados, por ejemplo empleados a disgusto. También es posible usar **disparadores** para que inicien un procedimiento de auditoría para una tabla, con el registro de todos los cambios, la hora en que se hicieron, y la identidad del usuario que los ejecutó. Por ejemplo, en Oracle, si se desea vigilar los cambios realizados a las calificaciones en la tabla `Enroll`, primero se debe hacer una tabla que mantenga los registros de auditoría. Este esquema sería el siguiente para dicha tabla:

```
EnrollAudit(dateOfUpdate, userId, oldStuid, oldCourseNo, oldGrade, newGrade)
```

El disparador debe insertar un registro en la tabla `EnrollAudit` cuando un usuario trate de actualizar una calificación en la tabla `Enroll`. El código para ello se muestra en la figura 9.3. Utiliza SYSDATE y USER, a las que se puede hacer referencia como seudocolumnas en Oracle. Ambas actúan como funciones que devuelven valores apropiados. SYSDATE devuelve datos actuales, en tanto que USER devuelve la ID del usuario actual.

9.7 Encriptado

Como réplica a la posibilidad de que haya archivos a los que se acceda directamente desde el sistema operativo, o que sean robados, es posible guardar los datos de la base en forma encriptada. Sólo el sistema de gestión de bases de datos (DBMS) puede descifrarlos, de modo que cualquier persona que los obtenga por otros medios recibirá datos sin sentido. Cuando los usuarios autorizados acceden a la información de manera adecuada, el DBMS recupera los datos y los decodifica en forma automática. El encriptado también debe usarse siempre que los datos se envíen a otros sitios, a fin de que quien intervenga las líneas también los reciba en forma cifrada. El encriptado requiere un **sistema de encriptado**, que consiste en los componentes que se mencionan a continuación:

- Un **algoritmo de encriptado**, que toma al texto normal (**texto plano**) como entrada, hace algunas operaciones con éste y produce como salida al texto en clave (**texto encriptado**)
- Una **clave de encriptado**, que es parte de entrada para el algoritmo de encriptado, y se elige de un conjunto muy grande de claves posibles
- Un **algoritmo de desencriptado**, que opera sobre el texto encriptado como entrada y produce al texto plano como salida

- Una **clave de desencriptado**, que es parte de entrada para el algoritmo de desencriptado y se escoge de un conjunto muy grande de posibles claves.

Un esquema muy usado para encriptar datos es el **Data Encryption Standard** (DES, Estándar de encriptado de datos), inventado por la National Bureau of Standards y adoptado en 1977. En el esquema DES, el algoritmo en sí es público, mientras que la clave es privada. Usa encriptado **simétrico**, en el que la clave de encriptado es la misma que la de desencriptado, y el algoritmo de desencriptado es el inverso del de encriptado. La figura 9.4 presenta la panorámica del proceso DES. Como el algoritmo es un estándar, es posible implementar el hardware para él en un solo chip, de modo que el encriptado y desencriptado sean muy rápidos y baratos, en comparación con la implementación en software del algoritmo. El algoritmo DES utiliza una clave de 56 bits en bloques de 64 bits sobre el texto plano, y produce bloques de 64 bits de texto encriptado. Cuando los datos se codifican, se separan en bloques de 64 bits. Dentro de cada bloque los caracteres se sustituyen y reacomodan de acuerdo con el valor de la clave. El algoritmo de decodificación usa la misma clave para volver a los caracteres originales y restaurarlos en sus posiciones originales en cada bloque. Hay dos desafíos importantes con el sistema DES, la seguridad de la clave y la facilidad de violar el código. La clave debe mantenerse segura o el encriptado no servirá de nada, ya que cualquiera que poseyera la clave tendría acceso a los datos. Por tanto, la seguridad depende del secreto con que se guarde la clave, pero ésta debe comunicarse a todos los usuarios autorizados. Entre más personas la conozcan, más probable es que llegue a usuarios no autorizados. Asimismo, es frecuente que sea necesario distribuir la clave a los receptores de los

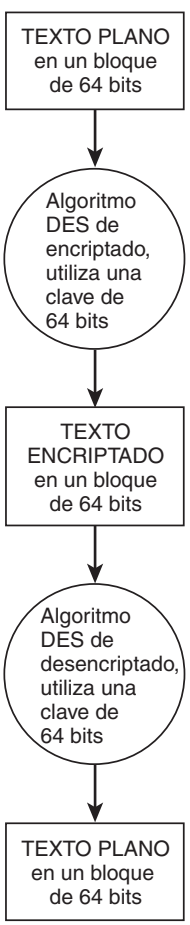


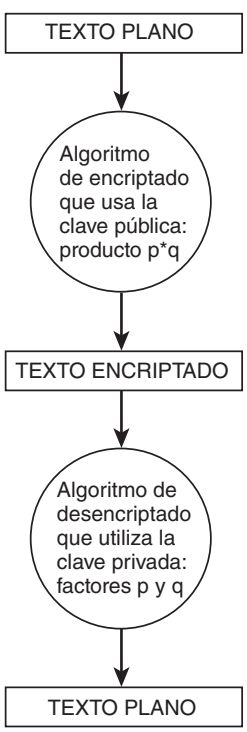
FIGURA 9.4
Panorámica del encriptado DES

mensajes encriptados. Si se emplean líneas de telecomunicación, la transmisión de la clave permitirá a quienes intervengan las líneas acceder con facilidad a los mensajes encriptados. Con frecuencia se utilizan líneas más seguras para enviar la clave, o se hace por correo o mensajería. Aunque el estándar DES aún se usa con amplitud, no es muy seguro porque puede violarse en una cantidad de tiempo razonable. En el año 2000 se desarrolló y adoptó como estándar nuevo una versión mejorada llamada **Advanced Encryption Standard** (AES, Estándar de encriptado avanzado). Utiliza un esquema simétrico más sofisticado que el DES, y soporta tres tamaños posibles de clave: 128, 192 o 256 bits, lo que depende del nivel de seguridad necesario. Entre más grande sea el tamaño de la clave, más difícil es violar el esquema.

Un segundo enfoque es el **encriptado de clave pública**, que utiliza parejas de números primos. En la figura 9.5 se presenta la panorámica del encriptado de clave pública. Se escoge para cada usuario una pareja de números primos muy grandes (p, q), como clave **privada** del usuario, y el producto de la pareja p*q se convierte en la clave **pública** de ese usuario. Las claves públicas se comparten con libertad, de modo que cualquier persona que desee enviar un mensaje al usuario encuentre su clave pública con facilidad. Después, la clave pública se emplea como entrada de un algoritmo de encriptado, que produce el texto encriptado para ese usuario. Cuando éste recibe un mensaje encriptado debe producir los factores primos de la clave pública para decodificarlo. Como no existe un método fácil o rápido de encontrar los factores primos de un número grande, es difícil en extremo que un intruso los obtenga. Sin embargo, un intruso con la determinación de violar la clave lo hará, siempre y cuando esté decidido a dedicar recursos sustanciales a la tarea. Este método sólo es tan seguro como la clave privada, por lo que debe darse a los usuarios sus claves privadas en alguna forma segura, y deben protegerse éstas contra su divulgación.

Un método bien conocido de encriptado de clave pública es el **RSA**, llamado así en honor a sus desarrolladores, Rivest, Shamir y Adleman.

FIGURA 9.5
Panorámica del encriptado de clave pública



9.8 Lenguaje de autorización en SQL

SQL tiene un sublenguaje de autorización que incluye enunciados para conceder y revocar privilegios de los usuarios. Un **privilegio** es una acción como leer, actualizar o eliminar, que se permite ejecutar a un usuario sobre los objetos de una base de datos. En el SQL estándar, se dan al creador de un esquema todos los privilegios sobre todos los objetos (tablas, vistas, roles, módulos) pertenecientes al esquema, y puede otorgar esos privilegios a otros usuarios. Lo común es que sólo el creador del esquema lo pueda modificar (agregar tablas, columnas, etc.). El enunciado para dar privilegios tiene la forma siguiente:

```
GRANT {ALL PRIVILEGES | lista de privilegios}
ON {table-name|nombre de vista}
TO {PUBLIC | lista de usuarios | lista de roles} [WITH GRANT OPTION];
```

Los posibles privilegios para las tablas de la base de datos son SELECT, DELETE, INSERT, UPDATE o REFERENCES (*nombre de columna*). Si una tabla se especifica en la cláusula ON, entonces ALL PRIVILEGES incluye todas estas operaciones. Si una vista se especifica en la cláusula ON y se construyó en forma tal que sea actualizable, en esa vista pueden asegurarse los privilegios SELECT, DELETE, INSERT y UPDATE. Para vistas no actualizables, sólo puede garantizarse SELECT. El privilegio UPDATE se hace más restrictivo al especificar una lista de columnas en los paréntesis después de la palabra UPDATE, lo que restringe al usuario a actualizar sólo ciertas columnas, por ejemplo así:

```
GRANT UPDATE ON Student(major) TO U101;
```

El privilegio REFERENCES se aplica a columnas que se usan como claves externas. Este privilegio permite al usuario hacer referencia a éstas en la creación de restricciones de integridad de claves externas. Por ejemplo, para permitir a un usuario capaz de actualizar la tabla `Enroll` que pueda hacer referencia a `stuId` en la tabla `Student` con objeto de comparar sus valores para la tabla `Enroll`, se escribe:

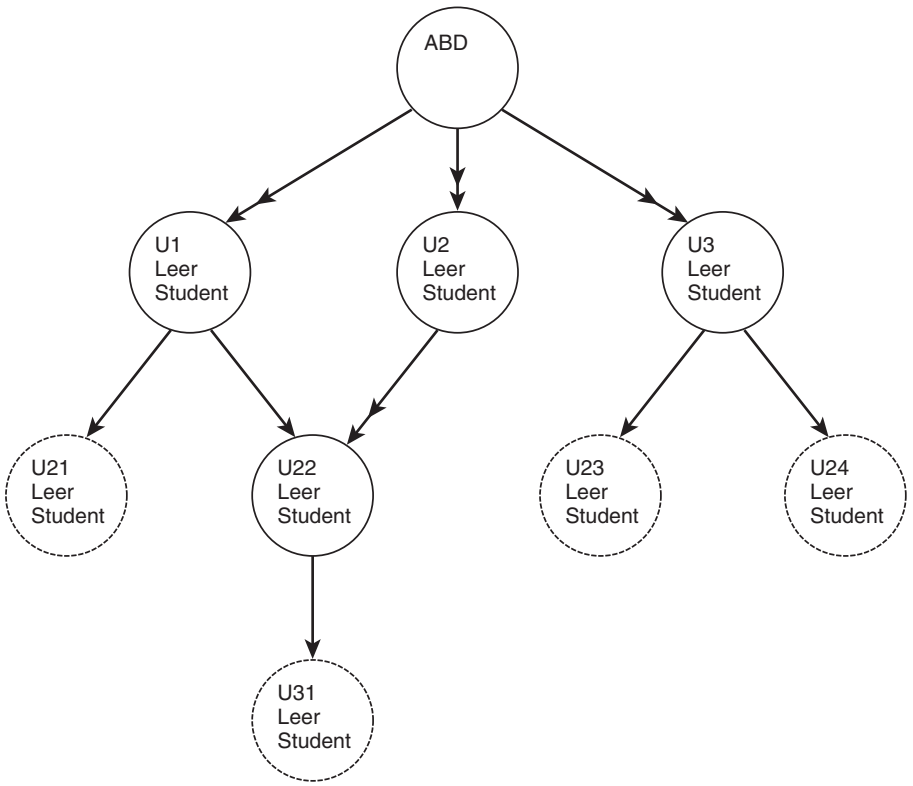
```
GRANT REFERENCES (stuId) ON Student TO U101;
```

La lista del usuario en la cláusula TO puede incluir a uno o varios usuarios. SQL:1999 incluye la capacidad de crear roles de usuario. Un **rol** puede considerarse como un conjunto de operaciones que deben ser ejecutadas por un individuo o grupo de individuos como parte de un trabajo. Por ejemplo, en una universidad, los asesores tal vez necesiten leer los expedientes de estudiantes seleccionados, por lo que debería haber un rol `Advisor` que lo permitiera. En función de las políticas de la universidad, el rol `Advisor` también incluye el privilegio de insertar registros de inscripción para los estudiantes en el momento de darse de alta. Se permite a los estudiantes ejecutar la operación SELECT pero no UPDATE en sus datos personales, por lo que tiene que haber un rol `Student` que permita ese acceso. Una vez que el ABD ha identificado un rol, garantiza para éste un conjunto de privilegios y luego le asigna las cuentas de usuario. Algunas de dichas cuentas tienen varios roles. Si se especifica PUBLIC en la cláusula TO, se da a todos los usuarios los privilegios especificados en el enunciado GRANT. La cláusula opcional WITH GRANT OPTION permite a los usuarios recientemente autorizados otorgar los mismos privilegios a otros. Por ejemplo, se podría escribir:

```
GRANT SELECT, INSERT, UPDATE ON Student TO U101, U102, U103 WITH GRANT OPTION;
```

Entonces, se permitiría a los usuarios U101, U102 y U103 escribir enunciados SQL, SELECT, INSERT y UPDATE para la tabla `Student` y pasar el permiso a otros usuarios. Debido a la capacidad que la opción GRANT da a los usuarios para autorizar a otros, el sistema debe dar seguimiento de las autorizaciones por medio de un **diagrama de garantías**, también llamado **grafo de autorización**.

FIGURA 9.6
Grafo de autorización



La figura 9.6 muestra un grafo de autorización. Aquí, el ABD, que se supone es el creador del esquema, dio un privilegio específico (por ejemplo, para leer la tabla `Student`) WITH GRANT OPTION a los usuarios U1, U2 y U3. Se usará una flecha con dos cabezas para denotar WITH GRANT OPTION, y con una sola cabeza para especificar sin ella. Un nodo con contorno continuo significa que el nodo recibió la opción GRANT, y punteado quiere decir que no. U1 pasó el privilegio a U21 y a U22, ambos sin la opción GRANT. U2 también pasó el privilegio a U22, esta vez con la opción GRANT, y U22 lo pasó a U31 sin la opción GRANT. U3 autorizó U23 y U24, ambos sin la opción GRANT. Observe que si se diera un privilegio diferente a uno de estos usuarios se necesitaría un nodo nuevo que representara al privilegio nuevo. Cada nodo del grafo representa una combinación de privilegio y usuario.

Para crear un rol se escribe un enunciado como el que sigue:

```
CREATE ROLE AdvisorRole;
CREATE ROLE FacultyRole;
```

Después se garantizan privilegios al rol del mismo modo que se hizo con los individuos, al escribir enunciados como los siguientes:

```
GRANT SELECT ON Student TO AdvisorRole;
GRANT UPDATE ON Enroll TO AdvisorRole;
GRANT SELECT ON Enroll TO FacultyRole;
```

Para asignar un rol a un usuario se escribe un enunciado así:

```
GRANT AdvisorRole TO U999;
```

Incluso se puede asignar un rol a otro rol si se escribe, por ejemplo:

```
GRANT FacultyRole TO AdvisorRole;
```

Esto da un medio de heredar privilegios a través de roles.

El enunciado de SQL para quitar privilegios tiene la forma siguiente:

REVOKE {ALL PRIVILEGES | *lista de privilegios*}
ON *lista de objetos*
FROM {PUBLIC| *lista de usuarios* | *lista de roles*};
[CASCADE | RESTRICT];

Por ejemplo, para U101, al cual previamente se había garantizado SELECT, INSERT y UPDATE, para `Student` con la opción GRANT, podrían retirársele algunos privilegios si se escribiera esto:

```
REVOKE INSERT ON Student FROM U101;
```

Esto revoca la capacidad de U101 tanto para insertar registros en `Student` como para autorizar a otros a hacerlo. Puede revocarse sólo la opción GRANT sin revocar INSERT, si se escribe lo siguiente:

```
REVOKE GRANT OPTION FOR INSERT ON Student FROM U101;
```

Si un individuo tiene la opción GRANT para cierto privilegio, y si después se revoca éste o la opción, todos los usuarios que hubieran recibido el privilegio de dicho individuo también verían revocado su privilegio. De esta manera, las revocaciones actúan en **cascada**, es decir, disparan otras revocaciones. Si un usuario obtuvo el mismo privilegio de dos autorizadores, y a uno de los cuales se revocó la autorización, el usuario aún conserva el privilegio por parte del otro autorizador. Así, si en la figura 9.6 el ABD revocó la autorización del usuario U1, U21 perdería todos sus privilegios, pero U22 conservaría los que hubiera recibido de U22. Como U22 tiene la opción GRANT, el usuario U21 podría volver a obtener privilegios de U22. De esta manera, los usuarios poco escrupulosos podrían estar en posibilidad de conspirar para conservar privilegios a pesar de los intentos del ABD para anularlos. Por esta razón, el ABD debe ser muy cuidadoso respecto de pasar la opción GRANT a otras personas. Si se especifica la opción **RESTRICT**, el sistema comprueba si hay algunas revocaciones en cascada, y si existen devuelve un mensaje de error sin ejecutar el enunciado de revocación. El valor de antemano es CASCADE. Cuando se anula un privilegio, el grafo de autorización se modifica con la eliminación del (los) nodo(s) que perdió (perdieron) sus privilegios.

9.9 La seguridad en Oracle

Oracle proporciona un sistema amplio de seguridad que va mucho más allá de los comandos del lenguaje de autorización de SQL. Además de los métodos que se han analizado aquí, hay muchas formas distintas de estructurar y administrar la seguridad de una base de datos en Oracle. Lo normal es que el ABD maneje el proceso de garantizar privilegios a los usuarios. Una forma de hacer esto es por medio de SQL*Plus. El ABD escribe el comando CREATE USER que tiene esta forma:

CREATE USER *nombre del usuario* IDENTIFIED BY *clave;*

Por ejemplo:

```
CREATE USER U999 IDENTIFIED BY SESAME;
```

Sin embargo, este comando no concede ningún privilegio al usuario, por lo que el U999 no podrá abrir una sesión a menos que el ABD también escriba lo siguiente:

```
GRANT CREATE SESSION TO U999;
```

Para probar esta cuenta, el ABD puede conectarse con el uso del nuevo nombre, si escribe, por ejemplo:

CONNECT U999/SESAME@*connectstring*;

El administrador conoce *@connectstring*, que se necesita sólo si Oracle está instalado en red. Es costumbre pedir que el usuario cambie su clave la primera vez que se registre en la realidad. Esto lo hace el ABD con el siguiente comando:

```
ALTER USER nombre del usuario
PASSWORD EXPIRE;
```

Cuando el usuario trate de conectarse, recibirá un mensaje que diga que su clave ha expirado y lo invitará a que establezca una nueva antes de conectarse. Una vez conectado, el usuario también puede cambiar su clave en cualquier momento con la escritura de los comandos SQL*Plus siguientes:

```
ALTER USER nombre del usuario
IDENTIFIED BY nueva clave;
```

Aunque el usuario se conectará, no podrá acceder a ningún dato porque el único privilegio que tiene es el de abrir una sesión. Para hacer uso de las funcionalidades de Oracle, el usuario necesita que le den privilegios adicionales, que pueden ser privilegios de sistema o de objeto.

9.9.1 Privilegios de objeto

En Oracle, un **privilegio de objeto** es el derecho a ejecutar un comando del DML sobre una tabla, vista, procedimiento, función, secuencia o paquete. El creador de un esquema tiene automáticamente todos los privilegios de objeto sobre todos los objetos que haya en el esquema, y puede garantizar los mismos privilegios de objeto a otros usuarios. Para las tablas, los privilegios incluyen SELECT, INSERT, UPDATE, DELETE y REFERENCES, según se describió en la sección 9.8, pero también ALTER (el derecho de usar el comando ALTER TABLE) e INDEX (derecho de usar el comando CREATE INDEX). Para vistas actualizables, los privilegios son SELECT, INSERT, UPDATE y DELETE. La sintaxis para éstos es la misma que el del estándar del lenguaje de autorización SQL que aparece en la sección 9.8. Por ejemplo, el ABD puede conceder al U999 privilegios amplios sobre la tabla `Student` con la escritura de lo siguiente:

```
GRANT ALL PRIVILEGES ON Student TO U999 WITH GRANT OPTION;
```

Si hay almacenado un procedimiento llamado `WrapUp`, el ABD puede dar permiso a U999 para correrlo con la escritura de este comando:

```
GRANT EXECUTE ON WrapUp TO U999;
```

9.9.2 Privilegios de sistema

En Oracle, los **privilegios de sistema** incluyen el derecho a ejecutar acciones con el uso de comandos DDL sobre datos, esquemas, espacios para tablas u otros recursos de la base de datos Oracle, así como el derecho de crear cuentas de usuario. Hay alrededor de 140 posibles diferentes privilegios de sistema. La lista de ellos se obtiene con el siguiente comando de SQL:

```
SELECT name
FROM SYSTEM_PRIVILEGE_MAP;
```

Los privilegios de sistema se dan a través de SQL*Plus con el uso del comando GRANT, en esta forma:

```
GRANT privilegio de sistema
TO nombre de usuario
[WITH ADMIN OPTION];
```

Por ejemplo, se permitiría al U999 que creara tablas si se escribe:

```
GRANT CREATE TABLE TO U999 WITH ADMIN OPTION;
```

Adicionalmente, los privilegios que son privilegios de objeto en tablas únicas pueden extenderse para convertirlos en privilegios de sistema que se extiendan a cualquier tabla, con el uso de la palabra clave ANY, así:

```
GRANT SELECT ANY TABLE TO U999;
```

La cláusula WITH ADMIN OPTION permite al usuario pasar el privilegio a otros.

9.9.3 Roles

Igual que en el estándar SQL:1999, Oracle permite que se concedan privilegios a un rol, así como a individuos o grupos de usuarios. Un rol consiste en un grupo de privilegios. Puede garantizarse cualquier número de roles a un usuario, y también a otros roles, lo que permite privilegios de herencia. Los roles se crean con el empleo de los comandos del lenguaje de autorización que se estudiaron en la sección 9.8.

9.9.4 Uso del administrador de la seguridad de Oracle

Oracle Enterprise Manager (Administrador de la Empresa de Oracle) contiene un módulo llamado Security Manager (Administrador de la seguridad) que ofrece opciones para garantizar y anular privilegios. El ABD tiene que registrarse para acceder a Security Manager. A partir de ahí, el ABD puede escoger *`User`* del menú, y después *`Create`* del submenú User. El ABD llena el nombre del nuevo usuario, escoge un perfil (que puede ser el *`default`*), y elige un método de autentificación, que por lo general es con una *`clave`*. Después ingresa una clave temporal y puede elegir que expire de inmediato. Esto hará que el nuevo usuario sea invitado a crear una nueva clave la primera vez que utilice la cuenta. También es posible especificar que la autentificación se haga por medio del sistema operativo en lugar de con una clave, si se elige *`External`* como mecanismo de autentificación. Al crear una o más cuentas de usuario, el ABD queda en posibilidad de usar los menúes de Security Manager para garantizar privilegios tanto de sistema como de objeto, así como definir roles.

9.9.5 Control del acceso para una sola base de datos

Una forma de obtener privilegios de ABD sobre una sola base de datos es crear una nueva base de datos Oracle 9i. Una forma fácil de hacerlo es con el Asistente de configuración de base de datos de Oracle, que se localiza en un directorio que se crea al instalar Oracle. Esta herramienta es una ayuda que guía a los usuarios a través del proceso de crear una nueva base de datos. Se invita al usuario a dar nombre a la base de datos y a elegir otras opciones. Entonces, el sistema creará una base de datos con el nombre elegido, arrancará cuentas SYS y SYSTEM para el usuario, y pedirá a éste las claves para dichas cuentas. De ese modo, el usuario tiene todos los privilegios en la nueva base de datos, inclusive la capacidad de crear usuarios nuevos de ella y de garantizar las autorizaciones que ya se estudiaron. Para usar la base de datos a través de SQL*Plus, el usuario comienza éste y de inmediato aparece una ventana de registro. El usuario puede registrarse usando SYS o SYSTEM, junto con la clave apropiada creada antes. También debe ingresarse la "cadena huésped", que es el nombre escogido para la base de datos. Una vez en SQL*Plus, el usuario puede crear el esquema de la base de datos con el empleo de SQL DDL. El usuario que crea una base de datos recibe una cuenta SYSTEM de manera automática, que en esencia es una cuenta de usuario con todos los privilegios para esa base de datos. Él o ella también pueden crear cuentas para otras personas, con el método SQL*Plus que ya se estudió, o con Security Manager.

9.10 Seguridad de una base de datos estadística

Las bases de datos estadísticas están diseñadas para proveer datos de apoyo al análisis estadístico de poblaciones. Los datos contienen hechos sobre los individuos, pero no significan nada si se recuperan sobre una base individual. Se garantiza a los usuarios permiso para acceder a información estadística como totales, cuentas o promedios, pero no a información sobre un individuo en particular. Por ejemplo, si se permite a un usuario acceso estadístico a una base de datos de los empleados, podría escribir peticiones como éstas:

```
SELECT SUM(Salary)
FROM Employee
WHERE Dept = 10;
```

pero no como éstas:

```
SELECT Salary
FROM Employee
WHERE empId = 'E101';
```

Cuando se permita que los usuarios tengan acceso a datos estadísticos se deben tomar precauciones especiales para garantizar que no puedan deducir datos sobre un individuo en particular. Para el ejemplo anterior, si no hubiera restricciones excepto que todas las consultas deben involucrar contar, sumar u obtener promedios, alguien que quisiera encontrar al empleado E101 podría hacerlo si estableciera las condiciones en la línea de WHERE para acotar la población a contener a ese individuo, así:

```
SELECT SUM (Salary)
FROM EMPLOYEE
WHERE Dept = 10 AND jobTitle = 'Programmer' AND dateHired >
'01-Jan-2004';
```

El sistema puede modificarse a fin de que rechace cualquier consulta para la que sólo haya un registro que satisfaga el predicado. Sin embargo, esta restricción se burla con facilidad si el usuario solicita el total de los salarios del departamento y luego lo vuelve a pedir sin el de E101. Ninguna de estas dos consultas se limita a un registro, pero a partir de ellas el usuario puede deducir fácilmente el salario del empleado E101. Para impedir que los usuarios deduzcan información sobre los individuos, el sistema puede restringir las consultas requiriendo que el número de registros que satisfagan el predicado esté por arriba de cierta cantidad, pero al mismo tiempo que el número de registros que satisfagan simultáneamente un par de consultas no supere cierto límite. También se pueden impedir conjuntos de consultas que repitan la participación de los mismos registros.

9.11 La seguridad de las bases de datos en Internet

A menos que se utilice software de seguridad, todos los mensajes que se envían por Internet se trasmiten en texto plano y son susceptibles de detección por parte de intrusos que usen software de "olfateo de paquetes". Es obvio que los clientes que quieren comprar productos necesitan tener cierta seguridad de que la información de su tarjeta de crédito va a permanecer privada cuando la envíen por Internet. Las compañías que permiten conexiones de Web a sus redes internas para tener acceso a sus bases de datos necesitan ser capaces de protegerlas de ataques. Tanto los que reciben como los que envían mensajes necesitan tener maneras de garantizar que el sitio con el que se comunican es genuino y confiable. Existen varias técnicas para lograr esto.

9.11.1 Cortafuegos

Un **cortafuegos** es una barrera de hardware y/o software que se usa para proteger la red interna de una organización (intranet) del acceso no autorizado. Para asegurar que los mensajes que entran o salen de la intranet cumplen con los estándares de la organización, se utilizan diversas técnicas. Por ejemplo, para ocultar la dirección real de la red se utiliza un **servidor próximo (*proxy server*)**, que es una computadora que intercepta todos los mensajes en ambas direcciones. Otra técnica es un **filtro de paquetes**, que examina cada paquete de información antes de que entre o salga de la intranet para asegurarse de que cumple con un conjunto de reglas. Son varias las técnicas de acceso con mecanismos de seguridad que pueden aplicarse a las aplicaciones o conexiones.

9.11.2 Autoridades de certificación

Los clientes que desean comprar artículos de un sitio Web de comercio electrónico necesitan tener confianza de que el sitio al que se comunican es genuino y que la información de sus órdenes se trasmitirá de manera privada. Un método que se emplea mucho para verificar que un sitio es genuino es por medio de **autoridades de certificación** tales como Verisign. El proceso utiliza encriptado de llave pública. El sitio comienza el proceso de certificación por medio de generar una llave pública y otra privada, y envía una solicitud a Verisign, junto con la llave pública del sitio. Verisign envía un certificado encriptado al sitio. Cuando el cliente desea colocar una orden usando una conexión segura al sitio, su navegador pregunta al sitio por su certificado de Verisign, el cual lo envía al navegador en forma encriptada. El navegador desencripta el certificado con el uso de la llave pública de Verisign, y comprueba que en verdad se trata de un certificado Verisign y que la URL del sitio es la correcta. El certificado también contiene la llave pública del sitio. El navegador crea una clave para la sesión, que encripta por medio de la llave pública del sitio a partir del certificado, y envía al sitio la clave de la sesión, que al estar encriptada sólo puede ser desencriptada por el sitio real con el empleo de su llave privada. Como tanto el navegador como el sitio son los únicos poseedores de la clave de la sesión, ahora pueden intercambiar mensajes encriptados con la clave de la sesión y el uso de un protocolo sencillo como DES o AES. El proceso descrito aquí es el que se usa en el protocolo **Secure Sockets Layer** (SSL). Un protocolo similar, **Secure HTTP** (S-HTTP), garantiza la seguridad de los mensajes individuales en lugar de la de toda la sesión. Una medida adicional de seguridad para la transmisión de números de tarjetas de crédito la da el protocolo **Secure Electronic Transaction** (SET). Cuando el cliente se encuentra listo para transmitir esta información al final del proceso de su orden, el navegador envía al sitio la mayor parte de ella encriptada con su clave pública, pero la información de la tarjeta de crédito está encriptada con la clave pública de la compañía de la tarjeta, por lo que el sitio no la puede desencriptar directamente. En vez de ello, el sitio tiene que enviar la información de la tarjeta de crédito en forma directa a la empresa emisora para que apruebe el pago y luego lo haga efectivo.

9.11.3 Firmas digitales

Las firmas digitales utilizan una forma doble de encriptado con llave pública para crear comunicaciones seguras en dos sentidos que no sea posible rechazar. Permiten que los usuarios verifiquen la autenticidad de la persona con que se comunican y dan un medio de probar que cierto mensaje debe provenir de esa persona. El uso de una clave pública no es demostración suficiente de que el emisor es auténtico, ya que un impostor puede encontrar con facilidad la llave pública de otra persona y usarla para elaborar mensajes. Un método para usar firmas digitales es que el emisor primero codifique el mensaje con su propia clave

privada, y luego con la clave pública del receptor. Éste desencripta primero el mensaje con el uso de su clave privada, y luego con la clave pública del emisor. El doble encriptado garantiza que las dos partes son auténticas, ya que ninguna podría encriptar o desencriptar el mensaje sin su clave privada. Una variación de esta técnica involucra la utilización de una Autoridad de certificación con un proceso similar al utilizado en SSL.

9.12 Resumen del capítulo

La **seguridad** en las bases de datos significa proteger éstas del acceso, modificación o destrucción no autorizados. La **privacidad** es el derecho que tienen los individuos a tener cierto control de la información sobre sí mismos, y en muchos países está protegida por las leyes. La privacidad de los individuos se protege con la seguridad de la base de datos. Las violaciones a la seguridad son accidentales o deliberadas, y las violaciones en ella ocurren en varias formas. Un plan de control de la seguridad comienza con medidas para la seguridad física del edificio, en especial de las instalaciones de computación. El control de la seguridad de las estaciones de trabajo involucra el uso de la **autentificación**, es decir, la verificación de la identidad de los usuarios. Normalmente, el sistema operativo tiene ciertos medios de establecer la identidad del usuario, con el empleo de sus perfiles, ID del usuario, contraseñas, procedimientos de autentificación, gafetes, claves, o características físicas del usuario. Para acceder a la base de datos se puede requerir autentificación adicional.

La mayoría de los sistemas de administración de bases de datos que están diseñados para usuarios múltiples tienen un subsistema de seguridad, que proveen la **autorización**, con la que se asigna a los usuarios derechos para usar los objetos de la base de datos. La mayor parte tiene un **lenguaje de autorización** que permite al ABD escribir **reglas de autorización** que especifican cuáles usuarios tienen qué tipo de acceso a los objetos de la base de datos. El **control del acceso** cubre los mecanismos para implementar las autorizaciones. Se utiliza una **matriz de control del acceso** para identificar qué tipos de operaciones se permite ejecutar a los distintos usuarios en diversos objetos de la base de datos. En ocasiones, el ABD delega en otros su poder de autorización.

Las **vistas** son un método sencillo para implementar el control del acceso. Un **registro de seguridad** es un diario que almacena los intentos de violación de la seguridad. Un **procedimiento de auditoría** registra todos los accesos a la base de datos, mantiene información sobre el solicitante, la operación ejecutada, la estación de trabajo desde la que se hizo, la hora, características de los datos y valores involucrados. Los **disparadores** se usan para comenzar un procedimiento de auditoría. El **encriptado** usa un sistema de encriptado que consiste en un **algoritmo de encriptado** que convierte el **texto plano** en **texto encriptado**, una **clave de encriptado**, un **algoritmo de desencriptado** que reproduce el texto plano a partir del texto encriptado y una **clave de desencriptado**. Algunos esquemas de amplio uso para el encriptado son el **Data Encryption Standard (DES, Estándar de encriptado de datos)**, el **Advanced Encryption Standard (AES, Estándar de encriptado avanzado)** y el **encriptado de clave pública**. El DES y el AES utilizan un algoritmo estándar que con frecuencia se implementa en el hardware. El encriptado de clave pública usa una multiplicación de números primos como clave pública, y los factores primos de este producto como clave privada.

SQL tiene un **lenguaje de autorización** que da seguridad. El enunciado GRANT se utiliza para la autorización, y el enunciado REVOKE para quitar la autorización. Los privilegios se otorgan a individuos o a un rol, y luego se da el rol a individuos. En Oracle, los privilegios incluyen **privilegios de objeto** y **privilegios de sistema**. Se garantizan con el uso del sublenguaje de autorización o a través de Oracle Security Manager (Administrador de la seguridad de Oracle).

Cuando se accede a la base de datos a través de Internet, son necesarias técnicas de seguridad especiales, entre las que se encuentran cortafuegos, autoridades de certificación, como Verisign, que emiten certificados digitales con el uso de SSL o S-HTTP, SET para la información financiera, y las firmas digitales.

Ejercicios

9.1 Para cada uno de los siguientes, escriba enunciados SQL a fin de crear vistas donde sea necesario para garantizar los privilegios indicados en la base de datos Universtiy con este esquema:

```
Student(stuId, lastName, firstName, major, credits)
Faculty(facId, name, department, rank)
Class(classNumber, facId, schedule, room)
Enroll(stuId, classNumber, grade)
```

a. Dé permiso al usuario 201 para leer las tablas `Student` y `Class`.

b. Cree una vista de `Enroll` que no incluya el atributo del grado, y conceda permiso al usuario 201 para leer y actualizar la vista.

c. Cree un rol que incluya la lectura de `Student`, `Class` y la vista creada en el inciso b). Asigne a ese rol a todos los funcionarios de la oficina del rector, que son los usuarios 202, 203, 204 y 205.

d. Dé permiso al usuario 206, auxiliar del rector, para leer y modificar (insert, delete, update) las tablas `Faculty` y `Class`. Este usuario puede autorizar a otros para que lean y modifiquen `Class`, pero no `Faculty`.

e. El usuario 206 autoriza al 300 para que lea `Class` y `Faculty`. Escriba el comando para hacer esto.

f. Elabore un grafo de autorización que muestre todos los privilegios dados hasta este momento. Necesitará un nodo separado para cada combinación de privilegio y usuario.

g. Revoque el privilegio de autorización que se dio al asistente del rector en el inciso d), pero mantenga los de lectura y modificación. ¿Cómo indicaría este cambio en el grafo de autorización?

h. Dé permiso al archivista, usuario 500, de leer y modificar `Student`, `Class` y `Enroll`, y garantice esos derechos a otros.

i. Para todos los asesores académicos, conceda permiso de leer todos los registros `Class`. Al asesor del departamento de matemáticas permítale leer los registros `Student` de quienes cursan la carrera de Matemáticas, y modificar los registros `Enroll` de estos estudiantes.

9.2 Suponga que tiene una base de datos estadística con el esquema siguiente. Las únicas consultas legales son las que involucran COUNT, SUM y AVERAGE.

```
newFaculty(facId, lastName, firstName, department, salary, rank,
dateHired)
```

a. Escriba una consulta legal en SQL para encontrar el salario del único miembro de la facultad que es profesor en el departamento de arte.

b. Suponga que el sistema rechazará responder las consultas para las que sólo haya un registro que satisfaga el predicado, como en a). Escriba un conjunto legal de consultas que permitan al usuario deducir el salario del profesor de arte.

c. Suponga que hay 10 profesores en el Departamento de Arte. El sistema rechaza responder consultas en las que el número de registros sea menor de seis. También rechazará responder parejas de consultas para las que el número de registros que las satisfaga simultáneamente exceda de tres. ¿Estas restricciones, harían que las consultas de los incisos a) o b) fueran ilegales? Si así fuera, ¿existe otro conjunto de consultas legales que permitan deducir el salario del profesor de Arte?

9.3 a. Con el esquema de University que se ilustra en el ejercicio 9.1 escriba un enunciado SQL para crear una vista dependiente del valor de `Student`, que incluya sólo seniors.

b. Escriba un enunciado SQL para una vista independiente del valor de `Faculty`. No incluya toda la tabla.

c. Escriba un enunciado que autorice al usuario 125 a leer ambas vistas.

9.4 Escriba un disparador que genere un procedimiento de auditoría que dé seguimiento a todas las actualizaciones en el campo del salario de la tabla `newFaculty` que se ilustra en el ejercicio 9.2.

9.5 Ingrese a un sitio Web de comercio electrónico, como una librería grande. Localice y lea la información que se da acerca de la seguridad de las transacciones en línea. Determine si se utiliza el protocolo seguro SSL o algún otro. Si es posible, despliegue e imprima la información sobre el certificado Verisign del sitio.

Ejercicios de laboratorio: Exploración del sistema de autorización de Oracle

Si aún no lo ha hecho, cree la base de datos Worker-Department-Project-Assign, según las instrucciones en el ejercicio de laboratorio que está al final del capítulo 6. Conforme haga cada uno de los ejercicios siguientes, elabore un grafo de autorización que muestre los privilegios que se den.

a. Cree cinco usuarios, U100, U101, U102, U103 y U104.

b. Dé al usuario U100 todos los privilegios sobre todas las tablas, con la opción GRANT.

c. Conéctese como U100 y pase el privilegio de leer las cuatro tablas a U101 y U102, con la opción de GRANT.

d. Todavía como U100, transfiera el privilegio de leer `Worker` a U103 y U104, sin la opción GRANT.

e. Conéctese como U101 y transmita el privilegio de leer `Worker` a U103 y U104, sin la opción GRANT.

f. Conéctese como U100 y revoque todos los privilegios garantizados a U101.

g. A partir del grafo, determine cuáles privilegios, si hay alguno, todavía conservarán los usuarios restantes.

h. Pruebe estos privilegios con el uso apropiado de comandos SQL.

PROYECTO DE MUESTRA: IMPLANTACIÓN DE MEDIDAS DE SEGURIDAD PARA LA BASE DE DATOS DE LA GALERÍA DE ARTE

Se trabajará con la base de datos puramente relacional creada al final del capítulo 6.

- Paso 9.1. Cree una vista de valor independiente que oculte cierta información privada.

La vista será la de la tabla `Artist`, pero sin el número de seguridad social, ventas o información de la entrevista.

```
CREATE VIEW ArtistView1 AS
   SELECT artistId, firstName, lastName, areaCode, telephoneNumber,
     street, zip, usualMedium, usualStyle, usualType
   FROM Artist;
```

- Paso 9.2. Cree un usuario y autorícelo a leer la vista. Comience un grafo de autorización.

```
CREATE USER U1 IDENTIFIED BY SESAME;
GRANT SELECT ON VIEW ArtistView1 TO UI;
```

- Paso 9.3. Autorice a otros cuatro usuarios a acceder y/o modificar varias partes de la base de datos, y actualice el grafo de autorización.

```
GRANT SELECT ON Collector TO U2;
GRANT ALL PRIVILEGES ON Collector TO U3;
GRANT SELECT, INSERT ON Sale TO U4 WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON Artwork TO U5;
```

El grafo de autorización se presenta en la figura 9.7.

- Paso 9.4. Escriba un disparador de procedimiento de auditoría para las actualizaciones de campos delicados susceptibles de ser actualizados por los usuarios.

Este disparador vigilará los cambios al precio de una obra de arte.

```
CREATE OR REPLACE TRIGGER ArtworkPriceAuditTrail
   BEFORE UPDATE OF askingPrice ON Artwork
   FOR EACH ROW
   BEGIN
      INSERT INTO ArtworkPriceAudit
         VALUES(SYSDATE, USER, :OLD.artworkId,:OLD.askingPrice,
            :NEW.askingPrice);
   END;
```

PROYECTOS ESTUDIANTILES: IMPLANTACIÓN DE MEDIDAS DE SEGURIDAD PARA LOS PROYECTOS ESTUDIANTILES

- Paso 9.1. Cree una vista con valor independiente que oculte cierta información privada.

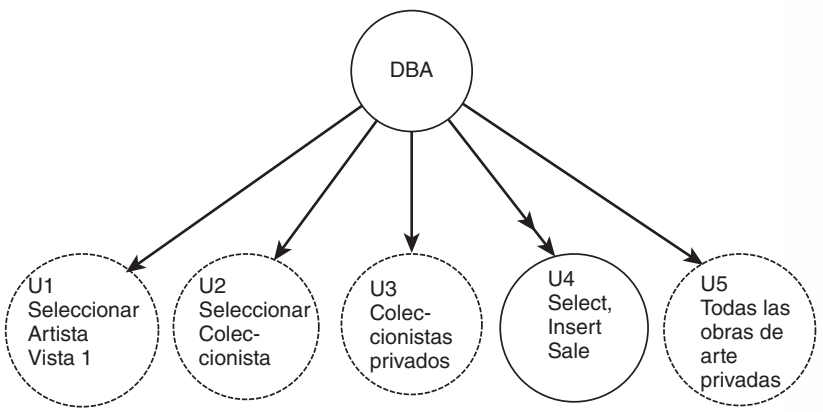


FIGURA 9.7
Grafo de autorización para la Galería de Arte

- Paso 9.2. Cree un usuario y autorícelo a leer la vista. Comience un grafo de autorización.
- Paso 9.3. Autorice a cuatro usuarios a acceder y/o modificar varias partes de la base de datos, y actualice el grafo de autorización.
- Paso 9.4. Haga un procedimiento de auditoría para las actualizaciones a un campo delicado que los usuarios tengan la capacidad de actualizar, y pruébelo por medio de la actualización.

CAPÍTULO 10

Administración de transacciones

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Las características de una transacción
- El significado del control de la concurrencia
- Por qué es necesario el control de la concurrencia
- El significado de la serialización
- Por qué y cómo se hace el bloqueo
- Cómo se detecta el candado mortal
- Cómo funciona el protocolo de bloqueo de dos fases
- Qué niveles de bloqueo se pueden usar
- Cómo se usan las estampas de tiempo para la serialización
- Cómo operan las técnicas optimistas de control de concurrencia
- El significado de la recuperación de la base de datos
- Algunas causas de falla de una base de datos
- Naturaleza y propósito del registro de la transacción de la base de datos
- Por qué y cómo se llevan a cabo los puntos de verificación

10.1 Propiedades de las transacciones

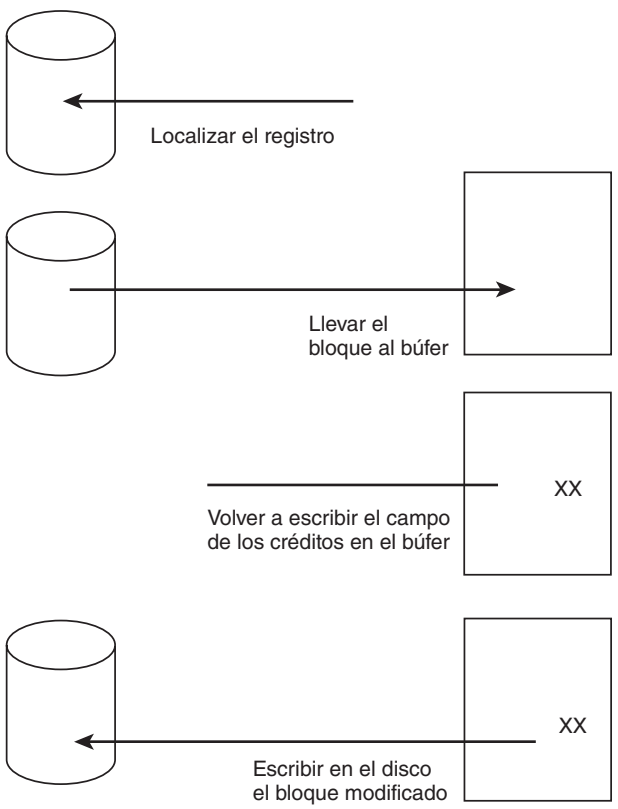
No importa el cuidado con que se diseñe y cree una base de datos, es fácil que se dañe o destruya, a menos que haya controles apropiados de la concurrencia y técnicas de recuperación. **Recuperación** de la base de datos es el proceso de restaurarla a su estado correcto en caso de falla. El **control de la concurrencia** es la capacidad de administrar procesos simultáneos que acceden a la base de datos sin que interfieran entre sí. Ambas medidas son necesarias para impedir que los datos sean inconsistentes o se pierdan.

El concepto de **transacción** es fundamental para entender tanto la recuperación como el control de la concurrencia. Una transacción se puede considerar una unidad lógica de trabajo en la base de datos. Puede ser un programa completo, fragmento de programa o comando único. Involucra cualquier número de operaciones en la base de datos. Para la base de datos University se usará el esquema relacional siguiente:

```
Student (stuId, lastName, firstName, major, credits)
Faculty (facId, facname, department, rank)
Class (classNumber, facid, schedule, room)
Enroll (classNumber, stuId, grade)
```

Una transacción sencilla ejecutada en esta base de datos podría actualizar el número de créditos en un registro `Student`, dado el `stuId`. Esta tarea implica localizar en el disco el bloque que contiene el registro `Student` deseado, llevar el bloque al búfer, volver a escribir el valor del campo `credits` en el registro del búfer y, finalmente, escribir el bloque actualizado en el disco. La figura 10.1 resume los pasos de esta transacción. Una transacción más complicada sería cambiar el `stuId` asignado a un estudiante en particular. Es obvio que se necesita localizar el registro `Student` apropiado, para llevar su bloque al búfer para actua-

FIGURA 10.1
Etapas en una transacción simple



lizar el `stuId` en el búfer y escribir la actualización en el disco, igual que antes, pero también es necesario encontrar todos los registros `Enroll` que tengan el antiguo `stuId` para actualizarlos. Si estas actualizaciones no se hicieran se tendría un estado inconsistente de la base de datos, estado en que los datos son contradictorios. Una transacción siempre debe llevar a la base de datos de un estado consistente a otro consistente. Mientras la transacción esté en progreso, es permisible tener un estado temporal inconsistente. Por ejemplo, durante la actualización de `stuId` habrá un momento en que una ocurrencia de `stuId` contenga el valor nuevo y otra el viejo. Sin embargo, al final de la transacción, todas las ocurrencias coincidirán. Una transacción es el conjunto de operaciones necesarias para llevar a cabo una unidad lógica de trabajo, a fin de llevar a la base de datos a un nuevo estado consistente. La transacción es un proceso atómico, una sola unidad del "todo o nada". No se permite ejecutar sólo una parte de una transacción, se ejecutan todas las operaciones de la transacción o ninguna, ya que una transacción parcial dejaría la base de datos en un estado inconsistente.

Hay dos formas de finalizar o terminar una transacción. Si se ejecuta hasta el final con éxito, se dice que la transacción fue **comprometida (commit)**, y la base de datos es llevada a un nuevo estado consistente. La otra posibilidad es que la transacción no se ejecute con éxito. En este caso, la transacción es **abortada**. Si se aborta una transacción, es esencial que la base de datos se restaure al estado consistente en que estaba antes de comenzar la transacción. Es decir, dicha transacción fue **deshecha**, esto quiere decir que se retrocede y al mismo tiempo se deshace **(rollback)** cada una de las operaciones realizadas por la transacción hasta su inicio. Una transacción comprometida no se puede abortar. Si se decide que la transacción comprometida fue un error, se debe ejecutar otra **transacción compensadora** para revertir sus efectos. Sin embargo, una transacción abortada que haya sido deshecha puede ser reiniciada en algún momento futuro y, en función de cuál haya sido la causa de la falla, podría ejecutarse exitosamente y ser comprometida. La figura 10.2 ilustra los estados posibles de una transacción.

Por lo general, es responsabilidad del programador identificar el comienzo y final de cada transacción, toda vez que el sistema de administración de la base de datos no es capaz de determinar cuáles actualizaciones constituyen una transacción única. En ciertos DML existen las palabras BEGIN TRANSACTION, END TRANSACTION, COMMIT, ABORT y ROLLBACK, para delimitar transacciones. Si no se utilizan estos delimitadores, por lo general se considera a todo el programa como una transacción única, y el sistema ejecuta de manera automática la instrucción COMMIT cuando el programa finaliza correctamente, y ROLLBACK si no lo hace. El estado activo de la transacción comienza con el enunciado BEGIN TRANSACTION y continúa hasta que el programa de aplicación aborta o concluye con éxito, y en este último caso se llega a END TRANSACTION, y el DBMS (sistema de gestión de base de datos) se prepara para comprobar que la transacción fue comprometida,

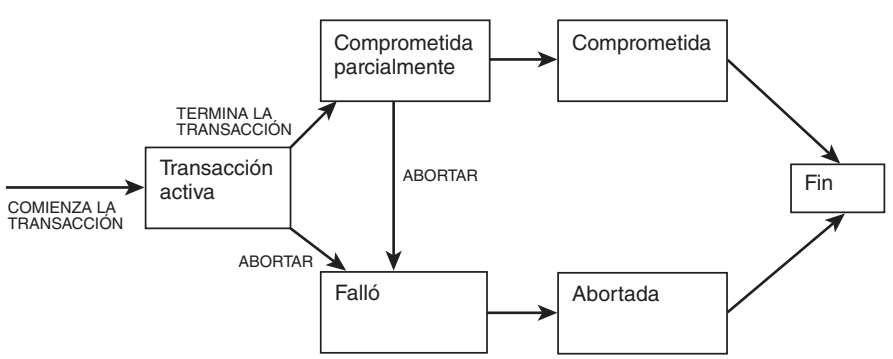


FIGURA 10.2
Diagrama del estado de la transacción

en cuyo caso se ejecuta COMMIT. Durante esta etapa de comprometimiento parcial, el DBMS verifica que la transacción no viole los protocolos de control de concurrencia que se estudian en la sección 10.4, o alguna restricción, y que el sistema sea capaz de hacer los cambios requeridos en la base de datos. Si no surgen problemas, la transacción se compromete. Por otro lado, si ocurriera un error fatal mientras la transacción está activa, se etiqueta como fracaso y se aborta. Si se hubieran hecho cualesquiera actualizaciones a la base de datos, se deshacen, es decir se realiza un ROLLBACK a la transacción. Aun después de que la transacción haya terminado con éxito, si hubiera algún problema con el control de concurrencia, integridad o falla del sistema mientras se está en el estado de comprometimiento parcial, es posible que la transacción se etiquete como fracaso y se aborte.

Para garantizar que la base de datos mantiene un estado correcto a pesar de una falla de concurrencia o del sistema, todas las transacciones deben mostrar cuatro propiedades importantes, por lo general llamadas **ACID**, que es un acrónimo en inglés de las propiedades siguientes:

- **Atomicidad.** La transacción es una sola unidad de "todo o nada". Se ejecuta todo el conjunto de acciones o ninguna. Para garantizar esta propiedad, el DBMS debe ser capaz de cancelar transacciones que no terminen con éxito, y anular sus efectos sobre la base de datos. El subsistema de recuperación del DBMS mantiene un **registro**, llamado **bitácora**, de todas las transacciones escritas en la base de datos, el cual se utiliza al deshacer la transacción.
- **Consistencia.** El usuario es responsable de asegurar que su transacción, si se ejecutara por sí sola, deje a la base de datos en un estado consistente. Es trabajo del subsistema de control de concurrencia del DBMS garantizar la consistencia cuando se ejecutan al mismo tiempo transacciones múltiples.
- **Aislamiento.** Es posible que varias transacciones se ejecuten al mismo tiempo con sus operaciones intercaladas. La propiedad de aislamiento requiere que el efecto final sea como si las transacciones se hubieran ejecutado una después de otra en vez de ejecutarse en forma concurrente. Como cada transacción de manera aislada deja a la base de datos en un estado consistente, el resultado en conjunto de todas las transacciones también será un estado consistente. El sistema de control de concurrencia tiene que garantizar el aislamiento.
- **Durabilidad.** Si una transacción ha sido comprometida, el DBMS debe asegurar que sus efectos se registren de manera permanente en la base de datos, aun si el sistema fallara antes de que se hubieran escrito en la base de datos los registros involucrados por la transacción. El subsistema de recuperación es responsable de garantizar la persistencia de los datos, para lo que utiliza la bitácora de las transacciones.

10.2 Necesidad del control de la concurrencia

Uno de los objetivos principales al desarrollar una base de datos es crear un recurso de información que compartan muchos usuarios. Si las transacciones se ejecutan una a la vez, es decir, **en forma serial**, de modo que cada transacción se comprometa antes de que comience la siguiente, no hay ningún problema de interferencia entre ellas. Sin embargo, es frecuente que los usuarios necesiten acceder a los datos en forma concurrente. Si todos los usuarios sólo van a leer datos, no hay forma de que interfieran uno con otro, por lo que no hay necesidad del control de concurrencia. Si los usuarios acceden a diferentes partes de la base de datos, sus transacciones pueden correr de manera concurrente sin ningún problema. Sin embargo, cuando dos usuarios tratan de hacer actualizaciones simultáneas a los mismos datos, o uno los actualiza mientras el otro los lee, es posible que haya conflictos.

Multiprogramación significa tener dos o más programas o transacciones en proceso al mismo tiempo. Por ejemplo, los sistemas de control de entrada y salida manejan estas operaciones en forma independiente, mientras el procesador principal ejecuta otras operaciones. Tales sistemas permiten que dos o más transacciones se ejecuten de manera simultánea. El sistema comienza a ejecutar la primera transacción hasta que alcanza una operación de entrada o salida. Mientras se ejecutan éstas, el procesador principal, que de otra manera estaría ocioso durante la entrada o salida, cambia a la segunda transacción y realiza cualesquiera operaciones que pueda ejecutar de esta transacción. Después, el control regresa a la primera transacción y sus operaciones se ejecutan hasta que otra vez llegue a una operación de entrada o salida. De esta manera, las operaciones de las dos transacciones son **intercaladas**, de modo que se llevan a cabo algunas de la primera y luego otras de la segunda, y así sucesivamente, hasta terminar todas las operaciones de las dos transacciones. Pero aun cuando ambas sean perfectamente correctas en sí mismas, la intercalación de sus operaciones puede producir un resultado incorrecto. Algunos ejemplos de los problemas potenciales ocasionados por la concurrencia son la **actualización perdida**, **actualización no comprometida**, **análisis inconsistente**, **lectura irrepetible** y **datos fantasmas**. A continuación se ilustrarán los tres primeros problemas para mostrar el orden cronológico en que se realizan las operaciones. Una **secuencia** es una lista de las acciones que realiza un conjunto de transacciones y muestran el orden en que se llevan a cabo. En dicha secuencia se tiene una columna para cada transacción, donde se indican las operaciones de esa transacción. Se preserva el orden de las operaciones de cada transacción, pero pueden ejecutarse simultáneamente las operaciones de otra transacción durante la ejecución de la primera, lo que resulta en la intercalación de las operaciones.

Problema de la actualización perdida

Suponga que Jack y Jill tienen una cuenta de ahorros mancomunada con saldo de 1 000 dólares. Jill recibe su salario y decide depositar 100 dólares en la cuenta, en una sucursal cerca de su trabajo. Entre tanto, Jack necesita gastar algo de dinero y decide retirar 50 dólares de la cuenta, en una sucursal diferente. Si estas transacciones se ejecutaran de manera serial, una después de otra sin intercalación de sus operaciones, el balance final sería 1 050 dólares sin que importara cuál se hubiera hecho primero. Sin embargo, si se realizan de manera concurrente el saldo final puede ser incorrecto. La figura 10.3 muestra la programación de la ejecución concurrente que da como resultado un saldo de 1 100 dólares. En la secuencia se denota como t1, t2, etc., las unidades secuenciales de tiempo a fin de mostrar el orden cronológico de las operaciones. Las únicas operaciones que involucran a la base de datos se indican explícitamente como *read* <*atributo*> o *write* <*atributo*>. Como se aprecia en la figura 10.1, estas operaciones involucran ya sea llevar un valor de la base de datos al búfer (leer) o escribir un nuevo valor en el bloque de la base de datos en el búfer (escribir) para luego copiar en la base de datos el bloque modificado. Las variables de la transacción, que por sencillez supondremos tienen el mismo nombre que los atributos de la base de datos a que corresponden, están separados de los atributos de ésta. La actualización de una variable de la transacción no tiene efecto en la base de datos hasta que se encuentra un comando *write* <*atributo*>. El valor que una transacción lee para una variable es el que utiliza para los cálculos. Cada transacción ignora cualquier cambio hecho a la base de datos, a menos que los haya realizado ella misma. Como se aprecia en la secuencia, la transacción de Jack lee un valor de 1 000 dólares para la variable BAL en el momento t2. La transacción de Jill hace lo mismo un poco más tarde, en t3. La transacción de Jack resta 50 dólares de BAL y luego, en t5, manda que se escriba en la base de datos, como se refleja en la columna de BAL. En el momento t5, la transacción de Jill todavía usa el valor de BAL que leyó, no el nuevo, que no ha leído. Su transacción suma 100 dólares y luego escribe un valor nuevo de 1 100 dólares en la base de datos, sobre la actualización de Jack. De acuerdo con la secuen-

FIGURA 10.3
Problema de la actualización perdida

HORA	Transacción de Jack	Transacción de Jill	SALDO
t1	BEGIN TRANSACTION		
t2	read BAL*(lee 1000)*	BEGIN TRANSACTION	1000
t3	. . .	read BAL*(lee 1000)*	1000
t4	BAL = BAL – 50*(950)*	. . .	1000
t5	write BAL*(950)*	BAL = BAL + 100*(1100)*	950
t6	COMMIT	. . .	950
t7		write BAL *(1100)*	1100
t8		COMMMIT	1100

cia, la actualización de Jack se pierde. Si se hubiera retrasado la lectura de Jill hasta que hubiera terminado la actualización de Jack, se habría evitado este problema.

El problema de la actualización no comprometida

Este problema ocurre con transacciones concurrentes cuando se permite que la primera modifique un valor que luego es leído por la segunda, y después la primera deshace la actualización, lo que invalida el dato que está usando la segunda. Éste también se conoce como problema de la **lectura sucia**, ya que se genera por leer **datos sucios**, que son resultados intermedios de una transacción no comprometida. La figura 10.4 muestra la secuencia de un ejemplo de actualización no comprometida que genera un error. Aquí, la transacción INTERÉS calcula el interés para una cuenta de ahorro, con tasa de 5%, mientras que la transacción DEPÓSITO aporta 1 000 dólares a la cuenta, pero ésta es deshecha. Suponga que el saldo inicial en la cuenta es de 1 000 dólares. Si DEPÓSITO actualizara en verdad el valor del saldo para que fuera 2 000 dólares antes de que INTERÉS lo leyera, la cantidad que almacenaría INTERÉS sería 2 100 dólares. Sin embargo, DEPÓSITO se deshace, lo que significa que el estado de la base de datos no debe mostrar ningún efecto debido a DEPÓSITO, por lo que en el momento t6 se restaura el valor original de 1 000 dólares como resultado de deshacer DEPÓSITO. Observe que DEPÓSITO ignora que INTERÉS ha leído lo que escribió, e INTERÉS también ignora que se deshizo DEPÓSITO. La razón de deshacer DEPÓSITO pudo ser que la transacción era errónea, o era el resultado de una caída del sistema. En t8, se escribe el valor calculado por INTERÉS. Como INTERÉS se calculó con el saldo incorrecto, el valor final de éste también será incorrecto.

El problema del análisis inconsistente

El problema del análisis inconsistente ocurre cuando una transacción lee varios valores pero una segunda transacción actualiza algunos de ellos durante la ejecución de la primera. Suponga que se desea encontrar el saldo total de todas las cuentas de ahorro pero se permite que durante esa transacción se realicen depósitos, retiros y transferencias. La figura 10.5 contiene la secuencia que muestra el resultado incorrecto que pudiera producirse. La transacción SUMBAL calcula la suma del saldo de todas las cuentas. Para que el ejemplo sea pequeño se supondrá que sólo hay tres cuentas. Observe que esta transacción no modifica las cuentas sino sólo las lee. Al mismo tiempo, la transacción TRANSFER transfiere 1 000 dólares de la cuenta A a la C; esta transacción se ejecuta y compromete correctamente. Sin embargo, interfiere con la transacción SUMBAL, que suma dos veces los 1 000 dólares, una cuando está en la cuenta A y otra cuando llega a la cuenta C.

HORA	DEPÓSITO	INTERÉS	SALDO
	Transacción	Transacción	
t1	BEGIN TRANSACTION		1000
t2	read BAL *(1000)*	. . .	1000
t3	BAL = BAL + 1000*(2000)*	. . .	1000
t4	write BAL *(2000)*	BEGIN TRANSACTION	2000
t5	. . .	read BAL *(2000)*	2000
t6	. . .	BAL = BAL * 1.05*(2100)*	2000
t7	ROLLBACK	. . .	1000
t8	. . .	write BAL *(2100)*	2100
t9		COMMIT	2100

FIGURA 10.4

Problema de la actualización no comprometida

Existen otros dos problemas famosos que pueden surgir debido a la concurrencia, y son:

a. **problema de la lectura irrepetible,** que ocurre cuando una primera transacción lee un registro, después otra escribe un valor nuevo para éste, y luego la primera transacción vuelve a leer el registro y obtiene un valor diferente.

b. **problema de los datos fantasma,** que sucede cuando una primera transacción lee un conjunto de renglones (por decir, las tuplas en el resultado de una consulta), otra transacción inserta un renglón y después la primera lee otra vez los renglones y detecta el renglón nuevo.

Hora	SUMBAL	TRANSFER	SALDO DE A	SALDO DE B	SALDO DE C	SUMA
t1	BEGIN TRANSACTION		5000	5000	5000	–
t2	SUM = 0;	BEGIN TRANSACTION	5000	5000	5000	–
t3	read BAL_A *(5000)*	. . .	5000	5000	5000	–
t4	SUM = SUM + BAL_A *(5000)*	read BAL_A *(5000)*	5000	5000	5000	–
t5	read BAL_B *(5000)*	BAL_A = BAL_A –1000*(4000)*	5000	5000	5000	–
t6	SUM = SUM + BAL_B *(10000)*	write BAL_A *(4000)*	4000	5000	5000	–
t7	. . .	read BAL_C *(5000)*	4000	5000	5000	–
t8		BAL_C = BAL_C +1000 *(6000)*	4000	5000	5000	–
t9		write BAL_C *(6000)*	4000	5000	6000	–
t10	Read BAL_C *(6000)*	COMMIT	4000	5000	6000	–
t11	SUM = SUM + BAL_C *(16000)*		4000	5000	6000	–
t12	write SUM *(16000)*		4000	5000	6000	16000
t13	COMMIT		4000	5000	6000	16000

FIGURA 10.5

Problema del análisis inconsistente

10.3 Serialización

Una **ejecución serial** de las transacciones significa que éstas se ejecutan una después de otra, sin cualquier intercalación de operaciones. Si A y B son dos transacciones, sólo hay dos posibilidades de ejecución seriales: primero se termina la transacción A y después se hace la B, o bien se concluye la B para luego ejecutar la A. En una ejecución serial no hay interferencia entre las transacciones, puesto que en cualquier momento dado sólo una se está ejecutando. Si las transacciones que aparecen en la figura 10.3 se hubieran ejecutado de manera serial, sin que importara cuál se hubiera realizado primero, la del depósito o la del retiro, los resultados habrían sido correctos. Si las transacciones en la figura 10.4 se hubieran realizado de manera serial, la transacción DEPÓSITO se habría deshecho antes de que comenzara la transacción INTERÉS, o ésta habría concluido antes de que comenzara DEPÓSITO, por lo que tampoco habrían interferido entre sí. En la figura 10.5, si la transacción TRANSFER se hubiera pospuesto hasta que terminara la transacción SUMINT, o si TRANSFER hubiera finalizado antes de que comenzara SUMINT, los resultados habrían sido correctos. Los tres problemas descritos se originaron al ejecutarse concurrentemente y dejaron a la base de datos en un estado inconsistente. Una ejecución serial no habría originado que surgieran estos problemas. Sin embargo, no hay garantía de que los resultados de todas las posibles ejecuciones seriales de un conjunto dado de transacciones sean idénticos. Por ejemplo, en la banca importa si INTERÉS de una cuenta se calcula *antes* o *después* de que se haga un depósito grande. En la mayoría de los casos, a los usuarios no les importa el orden en que se ejecutan sus transacciones, siempre y cuando los resultados sean correctos y no tengan que esperar demasiado. Para n transacciones hay n! posibles secuencias seriales. No importa la secuencia serial que se elija, ésta nunca producirá un estado inconsistente de la base de datos, por lo que se considera que toda ejecución serial es correcta, aun cuando puedan producirse resultados diferentes. El objetivo es encontrar formas de permitir que las transacciones se ejecuten de manera concurrente mientras se asegura que no interfieren una con otra, y producir un estado de la base de datos que pudiera haber sido generado por una verdadera ejecución serial.

Si un conjunto de transacciones se ejecuta de manera concurrente, se dice que la secuencia es **seriable** si produce los mismos resultados que una ejecución serial. Es esencial garantizar la seriabilidad de las transacciones concurrentes a fin de garantizar la consistencia e integridad de la base de datos. Al tener ejecuciones seriales, son importantes los factores siguientes:

- Si dos transacciones únicamente leen los ítem de datos, ellas no entran en conflicto y el orden no es importante
- Si dos transacciones operan (ya sea lectura o escritura) sobre los ítem de datos separados completamente, no hay conflicto y no es importante el orden
- Si una transacción escribe sobre un ítem de datos y otro lee o escribe sobre este mismo ítem, entonces el orden de ejecución sí es importante

Entonces, hay un **conflicto** entre dos operaciones si son verdaderos todos los enunciados siguientes:

- Pertenecen a transacciones diferentes
- Acceden al mismo ítem de datos
- Al menos una escribe en el mismo ítem

Una ejecución seriable de transacciones establece una secuencia de operaciones conflictivas en la misma manera que lo haría una ejecución serial, por lo que los resultados de la ejecución concurrente son los mismos que los de al menos una de las posibles ejecuciones seria-

les. Este tipo de seriabilidad se denomina **conflicto seriable**. La figura 10.6 ilustra dos secuencias, una serial y otra seriable para dos transacciones S y T. La figura 10.6(a) muestra la secuencia serial S, T, mientras que la figura 10.6(b) presenta la secuencia serial T, S. La figura 10.6(c) presenta una secuencia seriable que es equivalente en cuanto al conflicto a la primera secuencia serial, S, T. La lectura y escritura conflictivas de los ítem de datos ocurren en el mismo orden en la secuencia seriable que en la secuencia serial S, T. Observe que en (a), S lee y escribe *x* antes de que T la lea y escriba, y S lee y escribe *y* antes de que lo haga T. En la secuencia seriable en (c) se preserva el mismo orden de lecturas y escrituras. También note que aquí no son equivalentes las dos secuencias seriales, pero ambas se consideran correctas y dejarían la base de datos en un estado consistente. Puede verificar que las dos secuencias seriales dan resultados diferentes, por medio de la asignación de valores iniciales a *x* y *y*, y observar los valores finales que arrojan.

Es posible determinar si una secuencia es del tipo de conflicto seriable por medio de examinar el orden de todas sus lecturas y escrituras conflictivas, y compararlas con el orden de las mismas lecturas y escrituras en las secuencias seriales de las transacciones, a fin de determinar si el orden coincide con al menos una. Sin embargo, existe una técnica gráfica llamada **grafo de precedencias**, que permite determinar si una secuencia es de conflicto seriable. Para construir dicho grafo se dibuja un nodo para cada transacción, T1, T2, . . . , Tn. Los arcos dirigidos se dibujan como sigue:

- *Si Ti escribe X, y después Tj lee X, el arco se dibuja de Ti a Tj*
- *Si Ti lee X, y después Tj escribe X, el arco se dibuja de Ti a Tj*
- *Si Ti escribe X, y después Tj escribe X, el arco se dibuja de Ti a Tj*

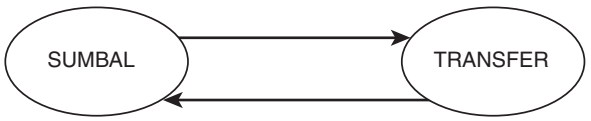
S es seriable si el grafo de precedencia **no tiene ciclos**. Un ciclo es una trayectoria que comienza en un nodo particular y termina en este mismo. La figura 10.7 muestra un grafo de precedencias para la secuencia en la figura 10.5. Como SUMBAL lee BAL_A y luego

Hora	S	T	S	T	S	T
t1	BEGIN TRANS			BEGIN TRANS	BEGIN TRANS	
t2	read x			read x	read x	BEGIN TRANS
t3	x = x*3			x=x+2	x=x*3	
t4	write x			write x	write x	
t5	read y			read y		read x
t6	y=y+2			y=y+5	read y	x=x+2
t7	write y			write y	y=y+2	write x
t8	COMMIT			COMMIT	write y	
t9		BEGIN TRANS	BEGIN TRANS		COMMIT	read y
t10		read x	read x			y=y+5
t11		x=x+2	x=x*3			write y
t12		write x	write x			COMMIT
t13		read y	read y			
t14		y=y+5	y=y+2			
t15		write y	write y			
t16		COMMIT	COMMIT			
	(a) En serie: S, T		**(b) En serie: T, S**		**(c) seriable**	

FIGURA 10.6

Dos secuencias en serie y otras seriable para las transacciones S y T

FIGURA 10.7
Grafo de precedencias para la figura 10.5



TRANSFER escribe BAL_A, se dibuja un arco de SUMBAL a TRANSFER. Como TRANSFER escribe BAL_C y después SUMBAL la lee, se dibuja un arco de TRANSFER a SUMBAL, lo que da como resultado un ciclo que demuestra que la secuencia no es seriable.

Si una secuencia es seriable, es posible utilizar el grafo de precedencias para encontrar una secuencia serial equivalente con el análisis de los arcos del grafo. Siempre que haya un arco de Ti a Tj, escriba Ti antes de Tj en la secuencia serial. Si varios nodos aparecen en el grafo, lo usual es obtener una secuencia parcial del grafo. Puede haber varias secuencias seriales. Si en el grafo hay un ciclo, no es posible ninguna secuencia serial, ya que a la larga se necesitará que un nodo se preceda a sí mismo.

La seriabilidad se logra de varias maneras, pero la mayor parte de los sistemas utilizan bloqueo o estampas de tiempo para lograrla, dos técnicas que se describen más adelante. En general, el DBMS tiene un subsistema de control de concurrencia que es parte del software y no lo controla directamente ni el usuario ni el ABD. Se utiliza una herramienta llamada **programador de actividades** para que las operaciones se ejecuten de inmediato, se retrasen o sean rechazadas. Si se rechaza la operación de una transacción, ésta se aborta, para reiniciarla después de que la transacción en conflicto finalice.

10.4 Candados

Los candados (también llamados bloqueos) y las estampas de tiempo son las dos técnicas que generalmente se usan para garantizar la seriabilidad de transacciones concurrentes. De éstas, el candado es el que se utiliza con más frecuencia. Una transacción coloca un candado a un objeto de la base de datos para impedir que otra transacción lo modifique. Es posible colocar candados a objetos de varios tamaños, desde toda la base hasta un solo ítem de datos. El tamaño del objeto determina la finura, o **granularidad**, del candado. El candado se implementa con la inserción de una bandera en el ítem de datos, registro, página o archivo, para indicar que esa porción de la base de datos se encuentra bloqueada, manteniendo una lista de los objetos de la base de datos que tienen candado o por otros medios. Con frecuencia existen dos tipos de candados: **compartidos** y **exclusivos**. Si una transacción tiene un candado compartido sobre un ítem, éste se puede leer pero no actualizar. De este modo, muchas transacciones pueden tener candados compartidos sobre el mismo objeto al mismo tiempo. Si una transacción tiene un candado exclusivo sobre un ítem, esa transacción es la única que puede leer y actualizar al ítem, esto con la finalidad de impedir que otras transacciones interfieran con ella. La figura 10.8 ilustra una **matriz de compatibilidad de canda-**

FIGURA 10.8
Matriz de compatibilidad de candados

	La transacción 2 solicita un candado compartido	La transacción 2 solicita un candado exclusivo
La transacción 1 no tiene candado	Sí	Sí
La transacción 1 tiene un candado compartido	Sí	No
La transacción 1 tiene un candado exclusivo	No	No

dos, que indica qué tipo de solicitudes de candado se pueden solicitar simultáneamente. Si la transacción 1 tiene el tipo de candado indicado a la izquierda, y la transacción 2 solicita el tipo de candado indicado en la parte superior de la columna, la matriz muestra si se puede garantizar la solicitud del candado. Como se ve, si la primera transacción no tiene candado, en la segunda se garantiza un candado compartido o exclusivo. Si la primera transacción tiene un candado compartido, en la segunda se garantiza sólo un candado compartido, no uno exclusivo. Todas las demás solicitudes serán denegadas.

Si un sistema utiliza candados, cualquier transacción que necesite acceder a un ítem de datos primero debe colocar un candado sobre éste, con la solicitud de un candado compartido para permitir sólo lectura o uno exclusivo para acceso de escritura. Si al ítem todavía no se le ha puesto candado mediante otra transacción, se garantiza el candado. Si el ítem ya tiene candado, el sistema determina si la solicitud es compatible con el candado existente. Si se solicita un candado compartido sobre un ítem que ya tiene un candado compartido, la solicitud se garantiza; de otro modo la transacción tendrá que esperar hasta que se libere el candado existente. Los candados de una transacción se liberan en forma automática cuando ésta termina. Además, una transacción puede liberar explícitamente los candados que tenga antes de que concluya.

10.4.1 Candado mortal

El conjunto de los ítem que lee una transacción se llama **conjunto de lectura**, y el que escribe una transacción se denomina **conjunto de escritura**. En el procesamiento de una base de datos, con frecuencia resulta difícil identificar todo el conjunto de lectura y escritura de una transacción antes de ejecutar ésta. No es raro que se acceda a registros debido a la relación que tienen con otros registros. Por lo general es imposible decir por adelantado exactamente cuáles registros se relacionarán. Por ejemplo, en la base de datos Universtiy, si se piden el nombre y departamento de todos los profesores de un estudiante en particular, no es posible predecir con exactitud cuáles registros de `Enroll`, `Class` y `Faculty` se necesitarán.

Como no siempre se pueden especificar de antemano los ítem que necesita bloquear una transacción, durante la ejecución de ésta se hacen solicitudes de candado. Éstas llegan a originar una situación llamada **candado mortal**, que sucede cuando dos transacciones quedan a la espera de que se liberen candados controlados por la otra. La figura 10.9 muestra dos transacciones, S y T, que tienen un candado mortal; la razón es que cada una espera que la otra libere un candado de un ítem que maneja. En el momento t1, la transacción S pide y obtiene un candado exclusivo (bloqueo X) del ítem a, y en el momento t2, la transacción T obtiene un candado exclusivo del ítem b. Entonces, en t3, S solicita un candado del ítem

Hora	Transacción S	Transacción T
t1	Bloqueo X de a	. . .
t2	. . .	bloqueo X de b
t3	solicitud de bloqueo X de b	. . .
t4	en espera	solicitud de bloqueo S de a
t5	en espera	en espera
t6	en espera	en espera
t7	en espera	en espera
. . .	. . .	. . .

FIGURA 10.9
Candado mortal con dos transacciones

FIGURA 10.10
Candado mortal con cuatro transacciones

Hora	Trans Q	Trans R	Trans S	Trans T
t1	bloqueo X por Q1	. . .	. . .	. . .
t2	. . .	bloqueo X por R1	. . .	. . .
t3	. . .	. . .	bloqueo X por S1	. . .
t4	. . .	. . .	. . .	bloqueo X por T1
t5	solicitud de bloqueo R1	. . .	. . .	. . .
t6	en espera	solicitud de bloqueo S1	. . .	. . .
t7	en espera	en espera	solicitud de bloqueo T1	. . .
t8	en espera	en espera	en espera	solicitud de bloqueo Q1
t9	en espera	en espera	en espera	en espera
. . .	. . .	. . .	. . .	. . .

b. Como T tiene un candado exclusivo de b, la transacción S queda en espera. Luego, en t4, T solicita un candado compartido (bloqueo S) del ítem a, que es mantenido con un candado exclusivo por medio de la transacción S. Ninguna transacción puede terminar porque cada una espera un candado que no puede obtener hasta que la otra termine. También es posible que ocurra un candado mortal que involucre varias transacciones. Por ejemplo, en la figura 10.10 se ilustran cuatro transacciones, Q, R, S y T, que tienen un candado mortal porque Q espera un ítem bloqueado por R, R espera un ítem de datos bloqueado por S, S espera un ítem bloqueado por T, y T espera un ítem bloqueado por Q. Una vez que ocurre un candado mortal, las aplicaciones involucradas no son capaces de resolver el problema. En vez de ello, el sistema tiene que reconocer la existencia del candado mortal y romperlo de algún modo.

FIGURA 10.11
Grafos de espera

FIGURA 10.11(a)
U espera a V

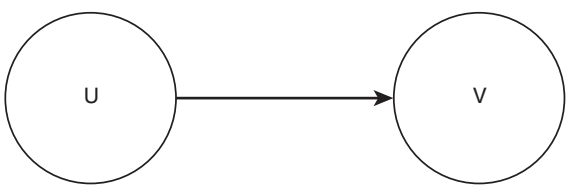


FIGURA 10.11(b)
S y T tienen un candado mortal

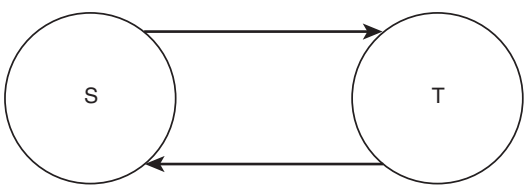
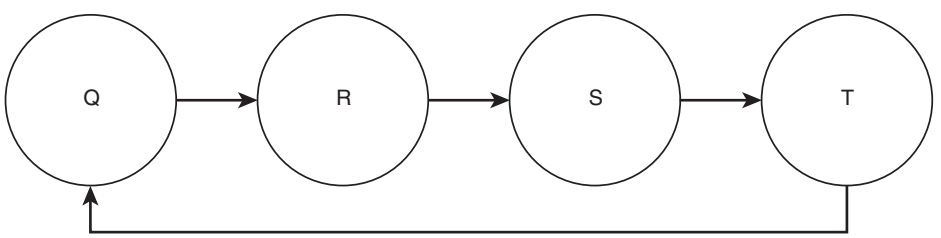


FIGURA 10.11(c)
Cuatro transacciones con candado mortal



Hay dos técnicas generales para manejar un candado mortal: **impedirlos**, que es cuando el sistema busca hacia delante si una transacción ocasionaría un candado mortal y nunca permitiría que éste ocurriera, y la **detección y recuperación**, que permite que suceda el candado mortal pero, una vez ocurrido, lo detecta y lo rompe. Como es más fácil determinar si hay un candado mortal para romperlo, que evitar que ocurra, muchos sistemas emplean el método de detección y recuperación. Los esquemas de detección de candado mortal utilizan un **grafo de solicitud de recursos** o **grafo de espera** para mostrar cuáles son las transacciones a la espera de los recursos bloqueados por otras. En dicho grafo, cada nodo representa una transacción. Un arco indica que una transacción está a la espera de que otra libere un candado. Por ejemplo, la figura 10.11(a) muestra dos transacciones U y V, con el arco dirigido de U a V que indica que U espera un ítem de datos bloqueado por V. El grafo de espera es mantenido por el sistema. Nuevos nodos se agregan al grafo a medida que comienzan nuevas transacciones. Cuando una transacción hace la solicitud de un candado sobre cierto recurso bloqueado, se agrega un arco al grafo. Si el recurso es desbloqueado y obtenido por la transacción, el arco se borra. Cuando se presenta un candado mortal, el grafo contendrá un **ciclo**. La figura 10.11(b) muestra un grafo de espera en el que las transacciones S y T están en candado mortal, ya que existe un ciclo que comienza en S, se dirige a T y regresa a S. Este ciclo es de longitud 2 porque contiene 2 arcos o trayectorias dirigidas, S a T y T a S (por supuesto, también hay un ciclo que comienza en T). La figura 10.11(c) muestra un grafo de espera que tiene un ciclo de longitud 4. Éste ilustra las transacciones Q, R, S y T, de la figura 10.10.

El sistema detecta candados mortales por medio de la revisión periódica de su grafo de espera, en busca de ciclos. Observe que debe revisar ciclos de cualquier longitud, ya que uno puede involucrar muchas transacciones. Una vez que se detecta un ciclo, el sistema debe resolver el candado mortal. Lo hace por medio de escoger a una de las transacciones del ciclo como **víctima**, la cual tendrá que ser deshecha (roll back), liberando de esta manera los candados que tenía y permitiendo que las demás transacciones que están en el ciclo puedan continuar. La víctima puede ser la transacción más nueva, que no ha realizado mucho trabajo, o la que haya hecho menos actualizaciones, entre otras opciones. Debe tenerse el cuidado de evitar elegir siempre a la misma transacción como la víctima, situación que se llama **inanición**, ya que esa transacción nunca terminará. Si en la figura 10.11(c) se escogiera a la transacción S como la víctima, entonces R obtendría sus ítem de datos solicitados, podría terminar y liberaría sus candados, de modo que Q también concluiría y, por último, T obtendría sus ítem de datos. Entonces, la transacción S, la víctima, estaría en posibilidad de volver a iniciar y obtendría sus candados en un momento posterior.

10.4.2 Bloqueo de dos fases

Un esquema de bloqueo que garantiza la seriabilidad es el **protocolo de bloqueo de dos fases**. De acuerdo con las reglas de este protocolo, cada transacción se desarrolla en dos fases: primero una **fase de crecimiento** en la que adquiere todos los candados necesarios para la transacción, y luego una **fase de encogimiento** en la que libera sus candados. No hay ningún requerimiento de que todos los candados se obtengan de manera simultánea. Normalmente, la transacción adquiere algunos candados, hace cierto procesamiento y adquiere candados adicionales que necesite. Sin embargo, nunca libera ningún candado hasta que ha alcanzado una etapa en la que no se necesitarán nuevos candados. Las reglas son:

- Una transacción debe adquirir un candado sobre un ítem antes de operar en éste. Para el acceso de sólo lectura basta un candado compartido. Para tener acceso a la escritura se requiere un candado exclusivo.
- Una vez que la transacción libera uno de los candados, ya no adquiere nuevos candados.

FIGURA 10.12
Aplicación del protocolo del bloqueo de dos fases a la figura 10.3

HORA	Transacción de Jack	Transacción de Jill	BAL
t1	COMIENZA TRANSACCIÓN		
t2	Xlock BAL		
t3	read BAL *(1000)*	COMIENZA TRANSACCIÓN	1000
t4	BAL = BAL – 50 *(950)*	request Xlock Bal	1000
t5	write BAL *(950)*	WAIT	950
t6	COMPROMETIDA/DESBLOQUEA BAL	WAIT	950
t7		grant XlockBAL read BAL (950)	950
t8		BAL = BAL +100	950
t9		write BAL *(1050)*	1050
t10		COMPROMETIDA/DESBLOQUEA BAL	1050

En la figura 10.12 se ilustra la manera en que se aplica a la figura 10.3 el protocolo de bloqueo de dos fases. Observe que ambas transacciones necesitaron un candado exclusivo sobre BAL, puesto que las dos planeaban actualizarla. La transacción de Jack solicitó y obtuvo primero el candado, y lo mantuvo hasta que terminó la modificación de BAL. Después liberó el candado y la transacción de Jill, que había estado suspendida (en estado de espera) mientras esperaba que el candado se liberara, pudo lograr éste y concluir.

En el protocolo **estándar** de bloqueo de dos fases, descrito por las dos reglas mencionadas, los candados pueden ser liberados antes de comprometerse la transacción (COMMIT), siempre y cuando no se soliciten nuevos candados después de que se ha liberado cualquiera. Un problema que puede surgir es que una transacción no comprometida que haya liberado sus candados sea deshecha. Si a una segunda transacción se le permitió leer un valor escrito por la transacción deshecha, también debe ser deshecha ya que leyó datos sucios. Este problema se llama **anulación en cascada**. La figura 10.13 muestra un caso de anulaciones en cascada. La transacción R escribió un valor de 4 para a, después liberó su candado debido a que ya no usaría a. La transacción S leyó el valor de 4 que había escrito R, y escribió un valor nuevo, 8, para a. La transacción T leyó el valor de 8 y calculó otro valor más, 18. Cuando R se deshizo, el valor de 4 que había escrito quedó invalidado, lo que hizo que también S se deshiciera. Entonces, lo que escribió, 8, se invalidó, por lo que T tuvo que deshacerse también.

Una variación, llamada bloqueo de dos fases **estricto**, requiere que las transacciones conserven sus candados exclusivos hasta que queden comprometidas (COMMIT), lo que impide las anulaciones en cascada. Un protocolo aún más estricto es el bloqueo de dos fases **riguroso**, que requiere que las transacciones mantengan todos sus candados, tanto compartidos como exclusivos, hasta que se comprometan.

El protocolo de bloqueo de dos fases se refina un poco con el uso del **escalamiento de bloqueo** para maximizar la concurrencia. En esta versión del bloqueo de dos fases, puede solicitar al principio candados compartidos que permitirán a otras transacciones concurrentes de sólo lectura acceder a los ítem. Después, cuando la transacción esté lista para una actualización, solicita que el candado compartido se **escale (aumente)**, o convierta en un candado exclusivo. El escalamiento puede tener lugar sólo durante la fase de crecimiento y tal vez requiera que la transacción espere hasta que otra transacción libere un candado compartido sobre el ítem. Una vez que el ítem ha sido actualizado, su candado puede ser disminuido

Hora	R	S	T	a
t1	COMIENZA TRANS			1
t2	Bloqueo X de a	COMIENZA TRANS		
t3	read a *(1)*	solicitud de bloqueo X de a	COMIENZA TRANS	
t4	a=a+3 *(4)*	en espera	solicitud de bloqueo X de a	
t5	write a *(4)*	en espera	en espera	4
t6	unlock a	en espera	en espera	
t7		solicitud de bloqueo X de a read a*(4)*	en espera	
t8		a = a*2 *(8)*	en espera	
t9		write a*(8)*	en espera	8
t10		desbloquear a	en espera garantizado el	
t11			bloqueo X de a *(8)*	
t12			a=a+10 *(18)*	
t13			write a *(18)*	18
t14	DESHECHA			
t15		DESHECHA		
t16			DESHECHA	

FIGURA 10.13
Anulaciones en cascada

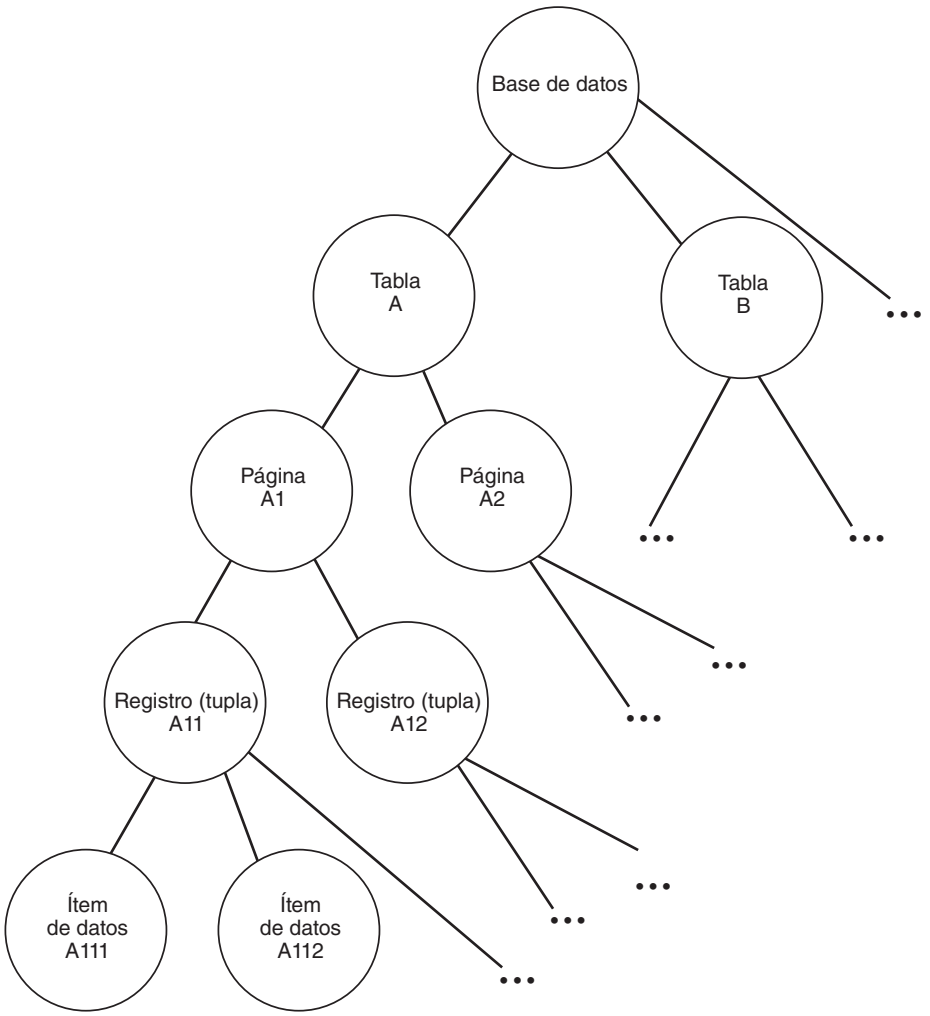
(convertido de uno exclusivo a uno compartido). La disminución puede tener lugar sólo durante la fase de encogimiento.

Aunque todas las formas del protocolo de dos fases garantizan seriabilidad, llegan a ocasionar candados mortales, ya que las transacciones esperan a poner candados sobre los ítem de datos. Si dos transacciones esperan bloquear ítem de datos ya bloqueados por la otra, ocurrirá un candado mortal y se necesitará el esquema de detección y recuperación ya descrito.

10.4.3 Niveles de bloqueo

Los candados se pueden aplicar a nivel de ítem de datos, de un registro, de un archivo, de un grupo de archivos o de toda la base de datos. Pocos sistemas implementan candados a nivel de ítem de datos debido a lo indirecto del procedimiento. Algunos bloquean páginas al mismo tiempo. La finura o granularidad de los bloqueos se expresa por medio de una estructura jerárquica en que los nodos representan objetos de datos de diferentes tamaños, como se ilustra en la figura 10.14. Aquí, el nodo raíz representa toda la base de datos, los nodos del nivel 1 indican tablas, los del nivel 2 son páginas de tablas, los del nivel 3 son registros, y los restantes representan ítem de datos. Siempre que un nodo esté bloqueado, todos sus descendientes también lo estarán. Por ejemplo, el bloqueo de la página A1 de la figura 10.14 bloquea los registros A11 y A12, así como todos sus ítem de datos. Si una segunda transacción solicita un candado incompatible sobre el mismo nodo, el sistema sabe sin duda que el bloqueo no puede garantizarse. Por ejemplo, una solicitud de candado exclusivo sobre la página A1 sería denegada. Si la segunda transacción solicita un candado incompatible sobre cualquiera de los descendientes del nodo bloqueado, el sistema debe revisar la estructura jerárquica a partir de la raíz hasta el nodo solicitado para ver si cualquiera de sus antecesores está bloqueado, antes de decidir si garantiza el candado. Así, si se solicita un candado exclusivo del registro A11, el sistema debe revisar a su padre, la página A1, a su abuelo, la tabla A, y la propia base de datos para ver si cualquiera de ellos está bloqueado. Cuando encuentre que la página A1 ya ha sido bloqueada, niega la solicitud. ¿Qué pasa si una transacción solicita el bloqueo de un nodo cuando uno de sus descendientes ya

FIGURA 10.14
Niveles de bloqueo



ha sido bloqueado? Un ejemplo es, si se solicita un bloqueo de la tabla A, el sistema tendría que comprobar cada página de la tabla, cada registro en esas páginas, y cada conjunto de datos en esos registros, para ver si cualquiera de ellos está bloqueado. Para reducir la búsqueda involucrada en la localización de bloqueos de los descendientes, el sistema usa otro tipo de bloqueo llamado bloqueo de **intención**. Cuando se bloquea algún nodo, se coloca un bloqueo de intención en todos sus antecesores. Así, si algún descendiente de la tabla A (verbigracia: la página A1) está bloqueado, y se hace una solicitud de bloquear la tabla A, la presencia de un bloqueo de intención en ésta indicaría que alguno de los descendientes de ese nodo ya está bloqueado. Los bloqueos de intención usan el modo exclusivo o compartido. Para garantizar la seriabilidad con niveles de bloqueo, se utiliza el protocolo de dos fases, lo que quiere decir que ningún bloqueo se garantiza una vez que se haya desbloqueado cualquiera; ningún nodo se bloquea hasta que su padre queda bloqueado por un nodo de intención, y ningún nodo se desbloquea hasta que todos sus descendientes hayan sido desbloqueados. El efecto de estas reglas es aplicar el bloqueo desde la raíz y hacia abajo, con el empleo de nodos de intención hasta que se alcanza el nodo que requiere un candado compartido o exclusivo, y liberar bloqueos de los restantes hacia arriba. Sin embargo, aún son posibles los candados mortales, y deben detectarse y resolverse con los métodos ya analizados.

10.5 Estampas de tiempo

Un mecanismo de control de concurrencia alternativo que elimina el problema de los candados mortales son las **estampas de tiempo**. En este esquema no se utilizan candados, por lo que no puede presentarse el candado mortal. Una estampa de tiempo para una transacción es un identificador único que indica el momento relativo de inicio de la transacción. Puede ser la lectura del reloj interno en el momento en que comienza la transacción, o el valor de un contador lógico que se incrementa cada vez que principia una transacción nueva. En cualquier caso, el valor de la estampa de tiempo de cada transacción T, que se denotará con TS(T), es único e indica qué tan antigua es la transacción. El efecto de una estampa de tiempo es asignar un orden serial a las transacciones. Entonces, el protocolo asegura que el orden se cumpla, por medio de no permitir ninguna lectura o escritura que viole dicho orden. Además de las estampas de tiempo para las transacciones, las hay para ítem de datos. Cada ítem de datos, P, contiene dos estampas de tiempo.

- Estampa de tiempo de lectura (P), que da la estampa de tiempo de la última transacción que leyó los ítem. Se actualiza cuando una transacción realiza una operación de lectura (P).
- Estampa de tiempo de escritura (P), que da la estampa de tiempo de la última transacción que escribió (actualizó) los ítem. Se actualiza cuando se efectúa una operación de escritura (P).

10.5.1 Protocolo básico de las estampas de tiempo

Hay dos problemas que pueden surgir cuando se utilizan estampas de tiempo:

1. Una transacción, T, pide leer en un ítem que ya ha sido actualizado por una transacción más reciente (posterior a ella)
2. T pide escribir en un ítem cuyo valor actual ya fue leído o escrito por una transacción más reciente

El primer problema es de **lectura tardía**. El valor que T necesita leer ya no se encuentra disponible debido a que fue sobrescrito, por lo que T debe deshacerse y volver a comenzar con una estampa de tiempo posterior. El segundo problema es el de **escritura tardía**, en el que una transacción más reciente ya leyó el antiguo valor o escribió uno nuevo. Otra vez, la solución está en deshacer T y reiniciarla con una estampa de tiempo posterior. Las reglas siguientes resumen este **protocolo básico de las estampas de tiempo.** Siempre se prueba la estampa de tiempo de la transacción, TS(T), contra la estampa de tiempo de escritura(P) y/o la estampa de tiempo de lectura(P) que identifica la(s) transacción(es) que escribieron o leyeron los ítem de datos por última vez.

1. Si T pide **leer** un ítem de datos P, se compara su TS(T) con la estampa de tiempo de escritura(P).
   a. Si la estampa de tiempo de escritura(P) <= TS(T), entonces se procede a usar el valor actual de los datos y se establece el valor de la estampa de tiempo de lectura(P) con TS(T). Sin embargo, si la estampa de tiempo de lectura aún tiene el valor más grande que TS(T) sólo la lee y no cambia la estampa de tiempo de lectura(P).
   b. Si la estampa de tiempo de escritura(P) > TS(T), entonces T llega tarde para hacer su lectura, y el valor de P que necesita ya fue sobrescrito, por lo que se deshace T.
2. Si T pide **escribir** un ítem de datos P, se compara TS(T) tanto con la estampa de tiempo de escritura(P) como con la estampa de tiempo de lectura (P).
   a. Si la estampa de tiempo de escritura(P) <= TS(T) y la estampa de tiempo de lectura(P) <= TS(T), entonces se escribe y reemplaza la estampa de tiempo de escritura(P) con TS(T).
   b. En otro caso, se deshace T, se asigna una nueva estampa de tiempo y se reinicia T.

En el caso de una escritura tardía, 2(b), la razón de deshacer si la estampa de tiempo de lectura (P) es posterior a TS(T), es que una transacción más reciente ya leyó el valor actual, y sustituir éste ahora introduciría un error. Si la estampa de tiempo de escritura (P) es posterior a TS(T), el valor que T intenta escribir es obsoleto, por lo que también introduciría un error.

Este esquema garantiza que las transacciones sean seriables y los resultados son equivalentes a una secuencia de ejecución serial en la que las transacciones son ejecutadas en orden cronológico por las estampas de tiempo. La figura 10.15 ilustra lo que ocurre cuando se aplica el protocolo estándar de las estampas de tiempo a la secuencia que se presenta en la figura 10.5.

10.5.2 Regla de escritura de Thomas

Una variación del protocolo básico de estampas de tiempo que permite una concurrencia mayor es aquel en que se ignoran en forma segura algunas escrituras obsoletas sin deshacer la transacción. Se aplica en el caso en que T intenta escribir en un conjunto de datos P, pero una transacción más reciente ya escribió un nuevo valor para P. Si una transacción más reciente ya leyó P, entonces necesitó el valor que T intenta escribir, por lo que es necesario deshacer T y reiniciarla. De otro modo la transacción puede continuar, con o sin hacer una

Hora	SUMBAL	TRANSFER	BAL_A,R,W	BAL_B,R,W	BAL_C,R,W	SUM
t1	BEGIN TRANSACTION- Assign timestamp t1		5000,t0,t0	5000,t0,t0	5000,t0,t0	–
t2	SUM = 0;	BEGIN TRANSACTION- Assign timestamp t2	5000,t0,t0	5000,t0,t0	5000,t0,t0	–
t3	read BAL_A *(5000)*	. . .	5000,t1,t0	5000,t0,t0	5000,t0,t0	–
t4	SUM = SUM + BAL_A *(5000)*	read BAL_A *(5000)*	5000,t2,t0	5000,t0,t0	5000,t0,t0	–
t5	read BAL_B *(5000)*	BAL_A = BAL_A – 1000 *(4000)*	5000,t2,t0	5000,t1,t0	5000,t0,t0	–
t6	SUM = SUM + BAL_B *(10000)*	write BAL_A *(4000)*	4000,t2,t2	5000,t1,t0	5000,t0,t0	–
t7		read BAL_C *(5000)*	4000,t2,t2	5000,t1,t0	5000,t2,t0	–
t8		BAL_C = BAL_C +1000 *(6000)*	4000,t2,t2	5000,t1,t0	5000,t2,t0	–
t9		write BAL_C *(6000)*	4000,t2,t2	5000,t1,t0	6000,t2,t2	–
t10	Read BAL_C –lectura tardía *DESHACER*		4000,t2,t2	5000,t0,t0	6000,t2,t2	–
t11		COMMIT				
t12	restart with timestamp t12					
. . .						

FIGURA 10.15

Estampas de tiempo aplicadas a la figura 10.5

escritura, de acuerdo con el siguiente protocolo de escritura, llamado **regla de escritura de Thomas**, que se aplica a la transacción T que intenta escribir P.

1. Si la estampa de tiempo de lectura(P) > TS(T), entonces una transacción más reciente necesitó el valor nuevo, por lo que T tiene que deshacerse.
2. Si la estampa de tiempo de escritura(P) > TS(T) pero la estampa de tiempo de lectura(P) <= TS(T), la operación de escritura se puede ignorar ya que es obsoleta, pero T puede continuar porque no se necesitó su escritura.
3. En otro caso, se realiza la operación de escritura y se reemplaza la estampa de tiempo de escritura(P) por TS(T).

10.5.3 Estampas de tiempo de versión múltiple

Los protocolos de estampas de tiempo analizados hasta este momento suponen que sólo hay una versión de cada conjunto de datos en la base. Es posible aumentar la concurrencia si se permite almacenar versiones múltiples, de modo que las transacciones accedan a la versión que sea consistente con ellas. Cada conjunto de datos P tiene una secuencia de versiones $<P_1, P_2, \ldots, P_n>$, cada una de las cuales tiene,

- El campo de contenido, un valor para P_i
- Una estampa de tiempo de escritura(P_i), que es la estampa de tiempo de la transacción que escribió el valor
- Una estampa de tiempo de lectura(P_i), que es la estampa de tiempo de la transacción más reciente que ha leído la versión P_i.

Siempre que se realiza una escritura(P), se crea una versión nueva de P, con estampa de tiempo de escritura apropiada. Cuando se efectúa una lectura(P), el sistema selecciona la versión apropiada de P. El protocolo de la estampa de tiempo de versión múltiple es,

1. Cuando la transacción T hace una lectura(P), el valor que se devuelve es el del campo de contenido asociado con la estampa de tiempo de escritura más tardía que sea menor o igual que TS(T). El tiempo de lectura es establecido al valor más tardío de TS(T) o su valor actual.
2. Cuando T hace una escritura(P), la versión que debe usarse es aquella cuya estampa de tiempo de escritura es la mayor que sea menor o igual a TS(T). Para esa versión,
   a. Si la estampa de tiempo de lectura(P) > TS(T), indica que P ya ha sido leída por una transacción más reciente, y se debe deshacer T porque sería una escritura tardía.
   b. En otro caso, se crea una versión nueva de P, con estampas de tiempo de lectura y escritura iguales a TS(T).

Por ejemplo, en la figura 10.16(a), la transacción 10 solicita un conjunto de datos que ya ha sido actualizado por la transacción 11. Sin embargo, existe una versión anterior de los datos con un valor de contenido de 100 y una estampa de tiempo de escritura de 9. Como ésta es la última estampa de tiempo de escritura menor o igual que la estampa de tiempo de la transacción (10), se usa esa versión para la transacción 10. Entonces, la estampa de tiempo de lectura de esa versión cambia a 10. El efecto de este proceso es asegurar que las dos transacciones leen conjuntos de datos como si en realidad se ejecutaran en forma serial. En la figura 10.16(b), la transacción 10 hace una escritura en un ítem cuya versión actual fue escrita por una transacción posterior. Sin embargo, se analiza la versión cuya estampa de tiempo de escritura es menor o igual que 10 (9, en este caso). Para esa versión, como la estampa de tiempo de lectura es menor o igual a 10, ésta no es una escritura tardía, por lo que se crea una versión nueva con ambas estampas de tiempo de 10.

FIGURA 10.16(a)

La transacción lee un ítem ya actualizado por una transacción más reciente

	Conjunto de ítem de datos - VAL		
	Contenido	Estampa de tiempo de lectura	Estampa de tiempo de escritura
(versión anterior)	100	9 (←cambia a 10)	9
(versión nueva)	150	11	11
La transacción con estampa de tiempo 10: lee VAL (obtiene 100, con el uso de la versión anterior, y cambia a 10 su estampa de tiempo de lectura)			

FIGURA 10.16(b)

La transacción intenta escribir un ítem ya actualizado por una transacción más reciente

	Data Item - VAL		
	Valor	Estampa de tiempo de lectura	Estampa de tiempo de escritura
(versión anterior)	100	10	9
(versión actual)	150	11	11
La transacción con estampa de tiempo 10: lleva VAL (obtiene 100) VAL = VAL –10 escribe VAL() (crea una versión nueva de VAL con valor de 90 y estampas de tiempo de lectura y escritura de 10)			
(versión nueva de VAL)	90	10	10

FIGURA 10.16

Ejemplos de estampas de tiempo de versión múltiple

10.6 Técnicas de validación

En muchos entornos, los conflictos entre transacciones son relativamente raros, y para la mayoría de éstas es innecesario el procesamiento adicional requerido para aplicar esquemas de candado. Hay una clase de técnicas que elimina esta carga. Las técnicas de **validación**, también llamadas **optimistas**, funcionan con la suposición de que normalmente no habrá un conflicto. Permiten que las transacciones se efectúen como si no existieran problemas de concurrencia, pero justo antes de que una transacción se comprometa, se revisa para determinar si ha ocurrido un conflicto. Si lo hubiera, la transacción debe deshacerse. Como la suposición es que es raro que el conflicto suceda, el deshacer una transacción será poco común. El deshacer de modo ocasional es el precio que se paga por eliminar los candados. Estas técnicas permiten más concurrencia que los esquemas tradicionales porque no se realizan candados.

En los esquemas optimistas una transacción pasa por dos o tres fases en orden, lo que depende de si es de sólo lectura o de actualización.

1. Fase de **lectura**, que se extiende del principio de la transacción hasta apenas antes de que se comprometa. Durante esta fase la transacción lee los valores de todas las variables que necesita, y los guarda en variables locales. Si la transacción hace cualesquiera escrituras, éstas se hacen a la copia local de los datos, no a la base de datos en sí.
2. Fase de **validación**, que sigue a la de lectura. En este momento, la transacción prueba a fin de determinar si hay alguna interferencia. Para una transacción de sólo lectura, esto consiste en observar que no haya errores causados por otra transacción que hubiera estado activa cuando se leyeron los valores de los datos. Si no hubo error, la

transacción se compromete. Si ocurrió una interferencia, la transacción se aborta y reinicia. Para una transacción que hace actualizaciones, la validación consiste en determinar si la transacción actual dejará la base de datos en un estado consistente, mediante seriabilidad. Si no es así, se aborta la transacción. Si la transacción de actualizar aprueba la validación, pasa a la tercera fase.

3. Fase de **escritura** que sigue a la fase de validación exitosa para las transacciones de actualización. En esta fase las actualizaciones hechas a la copia local se aplican a la base de datos.

La fase de validación se lleva a cabo por medio del examen de las lecturas y escrituras de las transacciones que pudieran causar interferencia. Se da a cada transacción, T, tres estampas de tiempo. Inicio (T) indica el tiempo de inicio relativo de la transacción. Al final de su fase de lectura y a medida que entra a la de validación, se da otra estampa de tiempo, Validación(T). Además, en el momento de finalizar, que es el momento en que concluye (inclusive su fase de escritura, si la hubiera), se da una estampa de tiempo Fin(T). Para aprobar la prueba de validación debe ser verdadero uno de los casos siguientes:

1. Todas las transacciones con estampas de tiempo anteriores deben haber terminado (incluso sus escrituras) antes de que comience la transacción actual. O bien,
2. Si la transacción actual comienza antes de que termine una anterior, entonces los dos enunciados siguientes deben ser verdaderos:
   a. los ítem escritos por la transacción anterior no son los que leyó la transacción actual, y
   b. la transacción anterior termina su fase de escritura antes de que la actual entre a su fase de validación.

La regla 2(a) garantiza que las escrituras de la transacción anterior no son leídas por la actual, en tanto que la regla 2(b) asegura que las escrituras se realicen en forma serial.

Aunque las técnicas optimistas son muy eficientes cuando hay pocos conflictos, pueden originar el deshacer transacciones individuales. Observe que la deshacer involucra sólo a una copia local de los datos, por lo que no hay anulaciones en cascada, ya que las escrituras no han alcanzado en realidad a la base de datos. Sin embargo, si la transacción abortada es larga, se perderá un tiempo valioso de procesamiento debido a que debe ser reiniciada. Un indicador de que el método optimista es una mala elección para el control de concurrencias en un entorno particular es que se deshacen transacciones con frecuencia.

10.7 Necesidad de la recuperación

La recuperación después de una falla consiste en restaurar la base de datos a un estado correcto. Hay muchos tipos distintos de fallas que llegan a afectar el procesamiento de una base de datos. Algunas sólo afectan la memoria principal, mientras que otras involucran el almacenamiento en disco o los dispositivos de almacenamiento de respaldo. Entre las causas de falla están las siguientes:

- Desastres físicos naturales, como incendios, inundaciones, terremotos o fallas de energía
- Sabotaje, contaminación o destrucción intencional de los datos, hardware o software
- Descuido, destrucción o contaminación no intencional de los datos o equipos por parte de los operadores o usuarios

- Descomposturas de discos, como aterrizaje de cabezas, discos defectuosos o sectores ilegibles, que causan la pérdida de datos almacenados
- Caídas del sistema debido a descomposturas del hardware, que ocasionan la pérdida de la memoria principal y caché
- Errores del sistema de software, que originan la terminación anormal o un daño en el sistema de administración de la base de datos
- Errores en el software de aplicaciones, como fallas de lógica en el programa que accede a la base de datos

Sin importar la causa de la falla, es necesario estudiar los posibles efectos de modo que se protejan o limiten los daños a la base de datos. Los efectos principales que hay que atender son la pérdida de la memoria principal, que incluye los búferes de la base de datos, y la de la copia del disco de la base de datos. Por fortuna, existen técnicas que minimizan estos efectos.

10.8 Técnicas de recuperación

Las transacciones se usan como la unidad básica de recuperación de una base de datos. El DBMS tiene un **administrador de recuperación** encargado de asegurar la **atomicidad** y **durabilidad** de las transacciones en caso de falla. La atomicidad requiere que se ejecute la totalidad de una transacción o nada, de modo que el administrador de recuperación tiene que asegurar que todos los efectos de las transacciones comprometidas se ejerzan sobre la base de datos, y que los efectos de las no comprometidas se deshagan como parte de la recuperación. La durabilidad requiere que los efectos de una transacción comprometida sean permanentes, por lo que deben sobrevivir tanto a la pérdida de la memoria principal como a la del almacenamiento en disco.

La posible pérdida en un disco se maneja haciendo respaldos frecuentes, o copias de la base de datos. En caso de falla, el respaldo se actualiza utilizando la bítacora del sistema, que se describirá en seguida. Si ocurre una falla del sistema y los búferes de la base de datos se pierden, la copia en disco de ésta sobrevive, pero tal vez sea incorrecta. Una transacción se compromete una vez que sus escritos se hacen en los búferes de la base de datos, por lo que puede haber un retraso entre el compromiso y la escritura real en disco. Debido a que las actualizaciones de los bloques del búfer no se escriben en forma automática en el disco (aun para transacciones comprometidas), si durante este retraso ocurre una falla del sistema, éste debe ser capaz de garantizar que las actualizaciones lleguen a la copia en disco de la base de datos. Para ciertos protocolos, los escritos realizados mientras una transacción está activa, aun antes de comprometerse, pueden alcanzar la copia en disco de la base de datos. Si sobreviene una falla del sistema antes de que se complete la transacción, los escritos necesitan deshacerse. Para dar seguimiento a las transacciones de la base datos, el sistema mantiene un archivo especial llamado **bitácora** que contiene información sobre todas las actualizaciones hechas a la base. La bitácora contiene registros de las transacciones, el identificador dada a la transacción y cierta información sobre ella. Por ejemplo, suponga que T es el identificador de la transacción, habría lo siguiente:

- Un registro de la forma <T comienza> que muestra el inicio de una transacción
- Registro de la forma <T,X,n> para cada operación de escritura que muestre el identificador de la transacción, nombre del ítem de datos y el valor nuevo
- Registro de la forma <T comprometida> que indica el fin de la transacción comprometida **o** un registro de la forma <T aborta> que señala que abortó una transacción
- Registros de los puntos de verificación, que se describirán en breve

10.8.1 Protocolo de actualización diferida

Con el uso del protocolo de recuperación diferida de actualizaciones, el DBMS registra todos los escritos de la base de datos en la bitácora, y no escribe en ella hasta que la transacción está lista para comprometerse.

La bitácora se utiliza como protección contra fallas del sistema, del modo siguiente:

- Cuando comienza una transacción, escribe un registro de la forma <T comienza> en la bitácora
- Al realizarse una operación de escritura, en realidad no escribe la actualización en los búferes de la base de datos o en ésta, sino que escribe un registro en la bitácora de la forma <T,X,n>
- Si una transacción está por comprometerse, escribe un registro en la bitácora de la forma <T comprometida>, que escribe en el disco todos los registros que hay en la bitácora para esa transacción, y luego se compromete la transacción. Usa los registros de la bitácora para realizar las actualizaciones a la base de datos
- Si la transacción aborta, simplemente ignora los registros que hubiera en la bitácora para la transacción y no ejecuta los escritos.

Observe que los registros de la bitácora se escriben en el disco antes de que la transacción se comprometa, por lo que si ocurre una falla del sistema mientras están en marcha las actualizaciones a la base de datos, los registros de la bitácora sobrevivirán y las actualizaciones se aplicarán más tarde.

La figura 10.17 ilustra una transacción corta, sus entradas correspondientes en la bitácora, y las partes de la base de datos involucradas. Note que la base de datos no se actualiza hasta

FIGURA 10.17

Una transacción y su bitácora

Figura 10.17(a)

Transacción T

```
T starts;
    read (X);
    X = X + 10;
    write (X);
    read (Y);
    Y = 2 * Y;
    write (Y);
    T commits;
```

(Suponga que al comenzar tanto X como Y tienen un valor de 100.)

Figura 10.17(b)

Entradas de la bitácora e ítem de datos de la base de datos

Bitácora	Ítem de datos de la base de datos	
	X	Y
<T comienza>	100	100
<T,X,110>	100	100
<T,Y,200>	100	100
<T se compromete>	100	100
	110	200

después de que la transacción se compromete. En caso de una falla del sistema, se examina la bitácora para identificar las transacciones que hubieran sido afectadas por la falla. Cualesquier transacción con registros en la bitácora <T comienza> y <T comprometida> debe **rehacerse** en el orden en que aparezca en la bitácora. No tiene que ejecutarse otra vez la transacción misma. En vez de ello, el **procedimiento de rehacer** realizará todos los escritos en la base de datos, con el uso de los registros de escritura de la bitácora para las transacciones. Recuerde que todos éstos tendrán la forma <T,X,n>, lo que significa que el procedimiento escribirá el valor n en el ítem de datos X. Esta escritura no tendrá efecto en el ítem de datos si ya se hubiera hecho antes de la falla, por lo que no habrá ningún daño si se escribe de manera innecesaria. Sin embargo, este método garantiza que será actualizado cualquier ítem de datos que no lo hubiera sido antes de la falla. Para cualquier transacción con registros en la bitácora de <S comienza> y <S aborta>, o cualquier transacción con <S comienza> pero sin compromiso ni aborto, no se hace nada, ya que no se hizo ninguna escritura real en la base de datos con el empleo de la técnica de las actualizaciones diferidas de la bitácora incremental, por lo que las transacciones no tienen que deshacerse. Este protocolo se denomina método de **rehacer/no hacer**, puesto que las transacciones comprometidas se rehacen, y no se deshace ninguna transacción. En caso de que ocurriera una segunda caída del sistema durante la recuperación, los registros de la bitácora se utilizan otra vez la segunda vez que se restablece el sistema. Debido a la forma de los registros de escritura en la bitácora, no importa cuántas veces se vuelvan a hacer los escritos.

10.8.2 Puntos de verificación

Una dificultad con este esquema es que cuando ocurre una falla tal vez no sepamos qué tan atrás buscar en la bitácora. Por ejemplo, tal vez se vuelvan a hacer las transacciones de todo un día, aun cuando la mayoría de ellas se hubieran escrito con seguridad en la base de datos. Para establecer un límite a la búsqueda que se necesita hacer en la bitácora, se utiliza una técnica llamada **punto de verificación**. Los puntos de verificación están programados a intervalos predeterminados e involucran las operaciones siguientes:

- Escritura de los bloques modificados en los búferes de la base de datos hacia el disco
- Escritura de todos los registros de la bitácora en la memoria principal hacia el disco
- Escritura de un registro de punto de verificación en la bitácora. Este registro contiene los nombres de todas las transacciones que estaban activas en el momento del punto de verificación.

Los puntos de verificación permiten limitar la búsqueda en la bitácora. Cuando ocurre una falla, se revisa ésta para encontrar la última transacción que haya comenzado antes del último punto de verificación. Si se supone que las transacciones se llevan a cabo en forma serial, cualquier transacción anterior se habría comprometido previamente y habría sido escrita en la base de datos en el punto de verificación. Por tanto, sólo se necesita volver a hacer aquella que estaba activa en el punto de verificación y las posteriores para las que aparezcan en la bitácora registros de comienzo y compromiso. Si las transacciones se realizan en forma concurrente, el esquema sólo se complica un poco más. Recuerde que el registro de los puntos de verificación contiene los nombres de todas las transacciones que estaban activas en el momento de establecerlos. Entonces, se rehacen todas esas transacciones y las posteriores.

10.8.3 Protocolo de actualización inmediata

Una técnica de recuperación un poco distinta utiliza una bitácora con actualizaciones inmediatas. En este esquema, las actualizaciones se aplican a los búferes de la base de datos a medida que ocurren y se escriben en ésta cuando es conveniente. Sin embargo, primero se

escribe un registro en la bitácora, ya que éste es un **protocolo de escritura adelantada**. El esquema para las transacciones exitosas es el siguiente:

- Al comenzar una transacción, se escribe en la bitácora un registro de la forma <T comienza>.
- Cuando se realiza una operación de escritura, se escribe un registro en la bitácora que contiene el nombre de la transacción, el del campo, el valor anterior del campo y el nuevo. Éste tiene la forma <T,X,o,n>.
- Una vez escrito el registro en la bitácora, se escribe la actualización en los búferes de la base de datos.
- Cuando sea conveniente se escriben en el disco los registros de la bitácora y luego las actualizaciones en la base de datos en sí.
- Cuando la transacción se compromete, se escribe en la bitácora un registro de la forma <T comprometida>.

Si una transacción aborta, la bitácora se usa para deshacerla porque contiene todos los valores anteriores de los campos modificados. Como una transacción tal vez haya hecho varios cambios en un ítem, los escritos se deshacen en orden inverso. Sin importar que los escritos de la transacción se hayan aplicado a la base de datos, la escritura de los valores anteriores garantiza que la base de datos se restablecerá al estado que tenía al principio de la transacción.

Si el sistema falla, la recuperación involucra el uso de la bitácora para **deshacer** o **rehacer** las transacciones, lo que constituye un protocolo **rehacer/deshacer**. Para cualquier transacción T para la que aparezcan en la bitácora registros <T comienza> y <T comprometida>, se **rehacen** con el uso de los registros de la bitácora para escribir los nuevos valores de los campos actualizados. Obsérvese que si ya hubieran sido escritos en la base de datos, aunque innecesarios, no tendrán efecto. Sin embargo, ahora se ejecutaría cualquier escrito que no hubiera llegado a la base de datos. Será necesario **deshacer** cualquier transacción, S, para la que la bitácora contenga un registro <S comienza> pero no <S comprometida>. Esta vez se utilizan los registros de la bitácora para escribir los valores antiguos de los campos afectados y así restablecer la base de datos al estado que tenía antes del comienzo de la transacción.

10.8.4 Paginación sombra

Una alternativa a las técnicas de recuperación basadas en la bitácora es la paginación sombra. El DBMS mantiene una **tabla de páginas** que contiene apuntadores a todas las páginas de la base de datos actual en el disco. Con la paginación sombra se mantienen dos tablas de páginas, una **tabla de páginas actual** y otra **tabla de páginas sombra**. Al principio las tablas son idénticas. Todas las modificaciones se realizan en la tabla de páginas actual y la sombra se deja sin cambio. Cuando una transacción necesita hacer un cambio a una página de la base de datos el sistema encuentra en el disco una página no usada, copia la página vieja de la base de datos a la nueva, y hace los cambios a la nueva página. También actualiza la tabla de páginas actual para apuntar a la página nueva. Si la transacción termina con éxito, la tabla de páginas actual se convierte en la tabla de páginas sombra. Si la transacción fracasa, se ignoran las páginas nuevas y la tabla de páginas sombra se vuelve la tabla de páginas actual. La operación de compromiso requiere que el sistema haga lo siguiente:

- Escribir todas las páginas modificadas del búfer de la base de datos al disco
- Copiar al disco la tabla de páginas actual
- En la ubicación del disco donde está registrada la dirección de la tabla de páginas sombra, escribir la dirección de la tabla de páginas actual y hacer de ésta la nueva tabla de páginas sombra

10.8.5 Panorama del algoritmo de recuperación ARIES

El algoritmo de recuperación ARIES es un método muy flexible y de concepto sencillo para la recuperación de bases de datos. Usa una bitácora, en la que se da a cada registro de bitácora un único **número de secuencia de bitácora** (LSN, log sequence number), que se asigna en orden creciente. Cada registro en la bitácora contiene el LSN del registro anterior de bitácora perteneciente a la misma transacción, lo que forma una lista vinculada. Cada página de la base de datos también tiene un **LSN de la página**, que es el LSN del último registro en la bitácora que actualizó la página. El administrador de recuperación mantiene una **tabla de transacciones** con una entrada para cada transacción activa, lo que da el identificador de la transacción, el estado (activa, comprometida o abortada) y el **último LSN**, que es el LSN del último registro en la bitácora para la transacción. También mantiene una **tabla de páginas sucias**, con una entrada para cada página en el búfer que se haya actualizado pero no escrito en el disco, así como el **recLSN**, que es el LSN del registro más antiguo en la bitácora que representó una actualización en la página en el búfer. El protocolo a utilizar es el de escribir por adelantado, el cual requiere que el registro de la bitácora se escriba a disco antes de actualizar el registro de la base de datos. También realiza puntos de verificación para limitar la búsqueda en la bitácora al hacer una recuperación. A diferencia de otros protocolos, ARIES trata de repetir la historia durante la recuperación, al intentar repetir todas las acciones de la base de datos realizadas antes de la falla, incluso aquellas de transacciones incompletas. Después emprende acciones de rehacer y deshacer, según sea necesario. La recuperación después de una caída se realiza en tres fases, como sigue:

1. **Análisis.** El DBMS comienza con el registro más reciente de punto de verificación y lee la bitácora hacia delante para identificar cuáles transacciones estaban activas en el momento de la falla. También usa la tabla de transacciones y la de páginas sucias con objeto de determinar cuáles páginas en el búfer contienen actualizaciones que todavía no han sido escritas en el disco. Asimismo, determina qué tan atrás es necesario ir en la bitácora para hacer la recuperación, con el uso de las listas vinculadas de LSN.
2. **Rehacer.** A partir del punto de inicio en la bitácora identificado durante el análisis, que es el más pequeño recLSN encontrado en la tabla de páginas sucias durante la fase de análisis, el administrador de recuperaciones va hacia delante en la bitácora, y aplica todas las actualizaciones no aplicadas a partir de los registros de la bitácora.
3. **Deshacer.** Al ir hacia atrás desde el final de la bitácora, el administrador de recuperaciones deshace las actualizaciones realizadas por transacciones no comprometidas, y termina en el registro más antiguo en la bitácora de cualquier transacción que hubiera estado activa en el momento de la falla.

Este algoritmo es más eficiente que los analizados previamente porque evita las operaciones innecesarias de rehacer y deshacer, ya que sabe cuáles actualizaciones han sido realizadas.

10.9 Administración de transacciones en Oracle

Oracle utiliza un mecanismo de control de concurrencia de versión múltiple, sin candados de lectura. Si una transacción es de sólo lectura, funciona con una vista consistente de la base de datos en el tiempo en que comenzó, que incluye sólo aquellas actualizaciones comprometidas en ese momento. El DBMS crea **segmentos de deshacer** que contienen las versiones antiguas de los ítem de datos utilizados tanto para la consistencia de la lectura como para cualesquiera operaciones de deshacer que sean necesarias. Utiliza un tipo de estampas de tiempo llamado **número de cambio de sistema** (NCS) asociado con cada transacción en su inicio. El sistema proporciona de manera automática candados al nivel del renglón para

enunciados SQL que requieren candados. Se dispone de varios tipos de candados, incluyendo candados tanto de DML como de DDL; los últimos se aplican al nivel de tabla. El candado mortal se maneja con el uso de un esquema de detección y deshaciendo alguna de las transacciones involucradas. Oracle tiene dos **niveles de aislamiento**, que son grados de protección de otras transacciones. Son los siguientes:

- **Lectura comprometida.** Ésta es la consistencia al nivel de enunciado, garantizando que cada enunciado en una transacción sólo lee datos comprometidos antes de que el enunciado inicie. Sin embargo, como los datos pueden cambiar durante la transacción que contiene al enunciado, puede haber lecturas irrepetibles y datos fantasmas. Éste es el nivel por default.
- **Seriable.** Ésta es la consistencia a nivel de transacción, lo que asegura que una transacción vea sólo datos comprometidos antes de que inicie ésta.

Para la recuperación, Oracle tiene un **administrador de recuperaciones** (RMAN por sus siglas en inglés), herramienta GUI que usa el ABD para controlar las operaciones de respaldo y recuperación. El RMAN se emplea para hacer respaldos de la base de datos o partes de ella, para hacer respaldos de las bitácoras, para restaurar datos desde los respaldos, y para realizar operaciones de recuperación que incluyen las de rehacer y deshacer, según se necesite. Mantiene el control de archivos, segmentos rehechos, bitácoras de lo que hay que rehacer y bitácoras archivadas de la actividad de rehacer. Cuando se llena una bitácora de rehacer, se archiva en forma automática. Oracle también tiene una herramienta de respaldo para ambientes en que es importante tener mucha disponibilidad en la forma de una base de datos administrada en espera. Ésta es una copia de la base de datos operativa que por lo general está en una ubicación distinta y a la que se puede recurrir si falla la base de datos regular. Se mantiene actualizada casi al instante por medio del envío de bitácoras de rehacer archivados y aplicando las actualizaciones a la base de datos en espera.

10.10 Resumen del capítulo

La **recuperación** de la base de datos es el proceso para regresar ésta a un estado correcto después de que ha ocurrido una falla. El **control de concurrencia** es la capacidad de hacer uso simultáneo de la base de datos sin que los usuarios interfieran unos con otros. Ambos son mecanismos de protección de la base de datos.

Una **transacción** es la unidad lógica de trabajo que garantiza llevar a la base de datos de un estado consistente a otro. Las transacciones se **comprometen** si terminan con éxito, y se **abortan** si concluyen en fracaso. Las transacciones abortadas deben deshacerse o **anularse**. Las transacciones comprometidas no pueden deshacerse.

El control de concurrencia es necesario cuando se permite el proceso simultáneo de las transacciones. Sin éste, pueden surgir problemas tales como el de la **actualización perdida**, el de la **actualización no comprometida** y el del **análisis inconsistente**. **Ejecución serial** significa hacer una transacción a la vez, sin intercalación de operaciones. Para dos o más transacciones habrá más de una posible ejecución serial, y quizá no todas produzcan los mismos resultados. Se emplea una **secuencia** para mostrar el momento en que ocurren las operaciones de una o más transacciones. Una secuencia es **seriable** si produce los mismos resultados que se obtendrían al ejecutar en forma serial las transacciones.

Dos métodos para garantizar la seriabilidad son los **candados** y las **estampas de tiempo**. Los candados son **exclusivos** o **compartidos**. Los candados compartidos son suficientes si una transacción sólo necesita acceso de lectura, pero son necesarios candados exclusivos para acceso de escritura. Una matriz de compatibilidad de candados muestra los tipos de candados que se han solicitado y que se garantizan de manera simultánea. Las transacciones

tal vez requieran esperar hasta que se liberen ciertos candados a fin de que puedan garantizarse sus solicitudes de colocar otros candados.

Como las transacciones a veces esperan candados, es posible que ocurra una situación de **candado mortal** en la que dos o más transacciones esperan que se liberen candados activados por cada una de ellas. Los esquemas de detección de candado mortal usan un grafo de **espera** para identificarlos. Tales grafos representan a las transacciones como nodos unidos por arcos que indican que una transacción está en espera de que otra libere un candado. Un **ciclo** en el grafo indica que ha ocurrido un candado mortal. El sistema detecta los candados mortales por medio de la revisión periódica de su grafo de espera en busca de ciclos. Un candado mortal se resuelve con la selección de una **víctima**, es decir, de la transacción que se ha de deshacer.

El **bloqueo de dos fases** es un protocolo que se utiliza mucho. Cada transacción adquiere todos sus candados antes de liberar alguno. Las variaciones del bloqueo estándar de dos fases incluyen el **estricto** y el **riguroso**. Otra variación es la que permite que las transacciones comiencen por medio de adquirir candados compartidos y **escalarlos** (mejorarlos) a candados exclusivos justo antes de hacer una escritura. El escalamiento sólo se puede hacer durante la primera fase (de crecimiento). En la segunda fase (encogimiento), una transacción **disminuye** un candado exclusivo a otro compartido. El propósito de este refinamiento es maximizar el acceso concurrente.

Se utiliza un árbol para representar la finura de los bloqueos en un sistema que permite bloquear objetos de diferente tamaño. Cuando se bloquea un ítem, todos sus descendientes en el árbol también se bloquean. Cuando una transacción nueva solicita un candado, es fácil revisar todos los antecedentes del objeto para ver si ya están bloqueados. Sin embargo, la revisión para ver si se encuentra bloqueado cualquiera de los descendientes del nodo consume tiempo. Para reducir el tiempo de búsqueda se coloca una intención de candado en todos los antecedentes de cualquier nodo bloqueado. Por tanto, si un nodo no tiene intención de bloqueo, ninguno de sus descendientes estará bloqueado. Se asegura la seriabilidad por medio del empleo de un protocolo de bloqueo de dos fases.

En un protocolo de **estampas de tiempo**, se asigna a cada transacción una estampa de tiempo, que es un identificador único que da el orden relativo de la transacción, y cada conjunto de datos tiene una **estampa de tiempo de lectura** y una **estampa de tiempo de escritura**. Los problemas surgen cuando una transacción intenta leer un ítem ya actualizado por una transacción más reciente, o cuando una transacción trata de escribir un ítem cuyo valor ya fue cambiado por una transacción posterior. El protocolo evita estos problemas por medio de deshacer las transacciones que no se pueden ejecutar de manera correcta. Si se mantienen versiones múltiples de conjuntos de datos, las lecturas tardías son susceptibles de hacerse.

Las técnicas de **validación** se usan en situaciones en las que los conflictos son raros. Cada transacción comienza con una **fase de lectura**, durante la cual lee todas las variables que necesita, las guarda como variables locales y sólo escribe a la copia local. Después de la fase de lectura, la transacción pasa a la **fase de validación**, durante la que el sistema determina si hubo alguna interferencia. Si no la hubo, avanza a la **fase de escritura**, en la que se aplican a la base de datos las actualizaciones a la copia local de la transacción. Si hubo alguna interferencia, la transacción se aborta y ya no avanza a la fase de escritura.

Para facilitar la recuperación, el sistema mantiene una **bitácora**, que contiene registros de transacciones que identifican el inicio de éstas, dan información acerca de las operaciones de escritura e identifican su final. Con el empleo de una bitácora incremental con **actualizaciones diferidas**, los escritos se hacen inicialmente sólo a la bitácora, y los registros de ésta se emplean para realizar actualizaciones reales a la base de datos. Si el sistema falla, examina la bitácora para determinar cuáles transacciones necesita **rehacer**, sobre todo

aquellas comprometidas pero tal vez no escritas en la base de datos. No hay necesidad de deshacer la escritura. En un **punto de verificación** se escriben en el disco todos los bloques modificados en los búferes de la base de datos, todos los registros de la bitácora, y un registro de puntos de verificación que identifican a todas las transacciones activas en ese momento. Si ocurre una falla, los registros de los puntos de verificación identifican a las transacciones que es necesario rehacer. Con el empleo de una bitácora incremental con **actualizaciones inmediatas**, éstas se hacen en la base de datos en cualquier momento después de que se escribió un registro en la bitácora para la actualización. La bitácora se usa para **deshacer** así como **rehacer** transacciones en caso de una falla. La **paginación sombra** es una técnica de recuperación que no utiliza bitácora, sino que las actualizaciones se hacen a una copia nueva de cada página. Una **tabla de páginas actuales** apunta a la página nueva. Una **tabla de páginas sombra** continúa el apuntamiento a la página anterior hasta que la transacción se compromete, momento en el que la tabla de páginas actual se convierte en la tabla de páginas sombra. El algoritmo de recuperación ARIES es muy sensible y flexible, y lleva a cabo la recuperación por medio de tratar de recrear el estado exacto que tenía la base de datos en el momento en que ocurrió la falla, para luego aplicar las operaciones de rehacer y deshacer según se necesite.

Oracle maneja el control de concurrencia con el uso de varias técnicas de candados y estampas de tiempo. Utiliza segmentos de deshacer tanto para la concurrencia como para la recuperación. El administrador de recuperación de Oracle mantiene archivos de control, segmentos de deshacer, bitácoras de rehacer y archivos de lo que se rehizo para recuperación.

Ejercicios

10.1 Suponga que un DBMS que usa actualizaciones inmediatas tiene los siguientes registros en la bitácora:

<R inicia>

<R,X,1,5>

<R,Y, – 1,0>

<R comprometida>

<S inicia>

<S,Z,8,12>

<registro de punto de verificación>

<S,X,5,10>

<T inicia>

<T,Y,0,15>

<S comprometida>

————————caída del sistema————————

Suponga que ocurre una caída del sistema como se indica, inmediatamente después del registro en la bitácora <S comprometida>,

a. ¿Cuáles transacciones, si hay alguna, necesitan rehacerse?

b. ¿Cuáles transacciones, si las hubiera, es necesario deshacer?

c. ¿Cuáles transacciones, si las hay, no se ven afectadas por la caída?

10.2 Suponga que las mismas transacciones y operaciones del ejercicio 10.1 las está realizando un sistema que utiliza el protocolo de actualización diferida.

a. Vuelva a escribir las entradas a la bitácora para las transacciones del ejercicio 10.1 para este método de registro.

b. ¿Cuáles transacciones, si las hay, necesitan rehacerse?

c. ¿Cuáles transacciones, si las hubiera, necesitan deshacerse?

d. ¿Cuáles transacciones, si hay alguna, no resultan afectadas por la falla?

10.3 (a) Suponga que la bitácora del ejercicio 10.1 contenía la entrada <S aborta> en lugar de <S comprometida>.

a. Si el sistema utiliza actualizaciones inmediatas y la caída ocurrió antes de que tuviera lugar la anulación, ¿qué cambios, si los hay, se harían en el proceso de recuperación?

b. Si el sistema estuviera utilizando actualizaciones diferidas, ¿qué cambios haría en el proceso de recuperación del ejercicio 10.2?

10.4 Suponga que deben realizarse las transacciones siguientes:

Transacción S:

```
read(a);
a = a + 10;
write(a);
read(b);
b=b*5;
write(b);
```

Transacción T:

```
read(a);
a = a*2;
write a;
```

a. Si los valores iniciales de a y b son 10 y 20, respectivamente, ¿cuáles son sus valores finales si las transacciones se ejecutan en forma serial, en el orden S, T?

b. Con el uso de los mismos valores iniciales, ¿qué valores finales tendrían a y b si el orden de ejecución fuera T, S?

10.5 Escriba una secuencia concurrente para las transacciones S y T en el ejercicio 10.4, que ilustre el problema de la actualización perdida.

10.6. Aplique el protocolo estándar de dos fases a la secuencia que se escribió en el ejercicio 10.5. ¿El protocolo permitirá la ejecución de esa secuencia? ¿Ocurre un candado mortal?

10.7 Aplique el protocolo estándar de estampas de tiempo a la secuencia que escribió en el ejercicio 10.5. En las columnas para los ítem de datos agregue estampas de tiempo de lectura y escritura, suponiendo que ambas fueron establecidas por una transacción con estampa de tiempo t0, según se ilustra en la figura 10.15. ¿Permitirá el protocolo la ejecución de esa secuencia? ¿Hay anulaciones?

10.8 Aplique el protocolo estándar de estampas de tiempo a la secuencia que se presenta en la figura 10.3. ¿El protocolo permitirá la ejecución de esa secuencia? ¿Hay algunas anulaciones?

10.9 Aplique el protocolo estándar de dos fases a la secuencia ilustrada en la figura 10.4. ¿Ocurre algún bloqueo muerto?

10.10 Aplique el protocolo estándar de estampas de tiempo a la secuencia que se ilustra en la figura 10.4. ¿Permitirá el protocolo que se ejecute esa secuencia? ¿Hay anulaciones?

10.11 Aplique el protocolo estándar de dos fases a la secuencia que se muestra en la figura 10.5. ¿Hay un candado mortal?

10.12 Aplique el protocolo estándar de estampas de tiempo a la secuencia de la figura 10.5. ¿El protocolo permitirá esa secuencia? ¿Hay anulaciones?

10.13 Sean T_1, T_2 y T_3 transacciones que operan sobre los mismos ítem de la base de datos, A, B y C. Sea que $r_1(A)$ significa que T_1 lee A, $w_1(A)$ significa que T_1 escribe A, y así sucesivamente para T_2 y T_3. Cada una de las siguientes muestra el orden de las lecturas y escrituras en una secuencia para T_1, T_2 y T_3. En cada caso, dibuje un grafo de precedencia con nodos para T_1, T_2 y T_3, y dibuje arcos para los conflictos. Determine si cada secuencia es seriable por medio de examinar el grafo de precedencia. Si es seriable, dé una secuencia en serie que sea equivalente.

a. $r_1(A)$; $r_2(A)$; $w_1(A)$; $r_3(A)$; $w_3(A)$; $w_2(A)$

b. $r_1(A)$; $r_1(B)$; $r_2(A)$; $r_3(B)$; $r_3(C)$; $w_2(A)$; $w_2(B)$

c. $r_1(A)$; $r_3(C)$; $r_2(C)$; $w_3(A)$; $r_2(B)$; $r_3(B)$; $w_2(B)$; $w_2(A)$

CAPÍTULO 11

Optimización de consultas relacionales

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

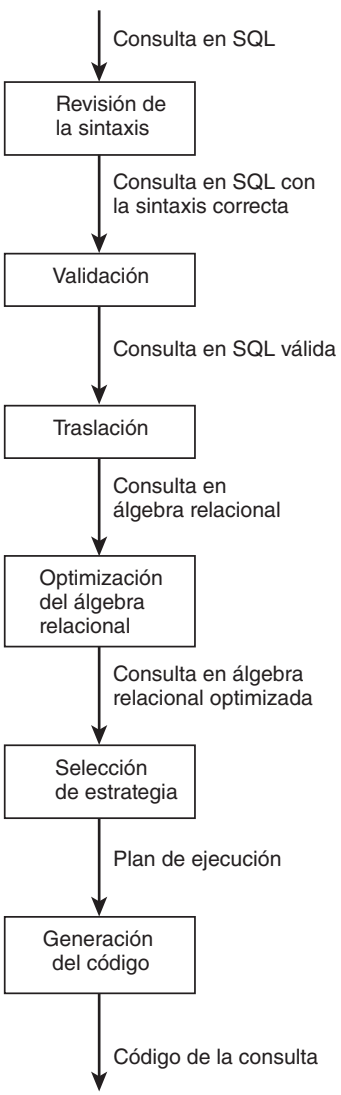
- Cómo se interpreta una consulta relacional
- Forma en que un árbol de consulta representa expresiones del álgebra relacional
- Por qué las operaciones SELECT deben realizarse primero
- Modo en que las condiciones de conjunción se pueden ejecutar en forma eficiente
- Cuándo deben realizarse las operaciones PROJECT
- Algunas reglas para las equivalencias de las operaciones algebraicas
- La heurística de la optimización de consultas
- Factores que determinan el costo de una consulta
- Cómo estimar el costo de los distintos métodos de realización de las operaciones SELECT
- Cómo estimar el costo de varios métodos para efectuar las operaciones JOIN
- Métodos de procesamiento de las operaciones PROJECT
- Métodos de procesamiento de las operaciones de conjuntos
- Cómo utilizar los ductos para hacer consultas

11.1 Interpretación y optimización de consultas

En el capítulo 6 se vio que ciertas consultas en SQL se podían expresar de diferentes maneras. Por ejemplo, en un enunciado SELECT que involucrara dos o más tablas, en ocasiones es posible elegir entre hacer una combinación (join) o una subconsulta. De manera similar, en el capítulo 4 se consideraron diferentes secuencias de comandos del álgebra relacional que producían resultados equivalentes. Se observó que ciertas formulaciones de dichas consultas eran más eficientes que otras. En este capítulo se estudiará con más detalle la eficiencia relativa y se analizará ciertas técnicas para mejorar la eficiencia de las consultas.

Cuando un sistema de administración de base de datos relacional recibe una consulta de alto nivel en un lenguaje como SQL, primero verifica la sintaxis para asegurar que el lenguaje se usa con toda corrección. Después se valida la consulta sintácticamente correcta por medio de revisar el diccionario de datos para comprobar que los atributos, tablas o vistas a que se hace referencia son objetos reales de la base y que las operaciones que se solicitan son válidas para dichos objetos. Entonces, la consulta tiene que traducirse a alguna forma que el sistema sea capaz de usar. Por ejemplo, aun cuando una consulta se exprese externa-

FIGURA 11.1
Proceso de interpretación y optimización



mente en SQL, el sistema debe transformarla a una representación interna de bajo nivel susceptible de usarse para procesarla. El álgebra relacional se utiliza para esta representación interna gracias a que sus operaciones se transforman con facilidad a operaciones del sistema. En lugar de álgebra relacional puede utilizarse cálculo relacional, pero para estos ejemplos se supondrá que se emplea álgebra relacional. Luego, el sistema examina la representación interna y utiliza su conocimiento de las operaciones del álgebra relacional para determinar si las operaciones se pueden reacomodar o cambiar para producir una representación equivalente pero más eficiente. Una vez que la representación interna es tan eficiente como sea posible, el sistema utiliza su conocimiento del tamaño de las tablas, índices, orden de las tuplas, y distribución de los valores, para determinar con toda exactitud cómo se procesará la consulta. Al hacerlo, estima el "costo" de los planes alternativos de ejecución y elige aquel que tenga asociado el menor costo. Para estimar el costo se considera el número y tipo de accesos requeridos al disco, cantidad de memoria interna y externa necesaria, tiempo de procesamiento que se necesita, y costos de comunicación si los hubiera. Después se codifica y lleva a cabo el plan de ejecución en el momento apropiado. La figura 11.1 resume este proceso para una consulta en SQL. Aunque la optimización de este tipo consume tiempo, son enormes los ahorros potenciales en tiempo de ejecución de la búsqueda y tamaño de los resultados intermedios, por lo que es rentable dedicar tiempo a la optimización antes de ejecutar la consulta. Por supuesto, la optimización en sí tiene un costo asociado. Como se verá en las secciones posteriores, para que sea posible optimizar, el DBMS debe mantener metadatos acerca de la base de datos, con seguimiento del número de registros en las tablas, número de valores para los atributos, tamaño de los índices, y otras estadísticas por usar en las estimaciones. Éste es el costo de ejecución de los algoritmos utilizados en la optimización. Algunos optimizadores "inteligentes" cambian en forma dinámica las estrategias de ejecución de la búsqueda si la que se eligió no resulta tan eficiente como se esperaba cuando se ha ejecutado parcialmente la consulta, proceso por el que se incurre en costos adicionales. El resultado en la eficiencia de la consulta debe compensar estos costos agregados.

11.2 Técnicas algebraicas para la transformación de una consulta

Se usará el esquema de la base de datos University para ilustrar la manipulación de las consultas del álgebra relacional.

```
Student (stuId, lastName, firstName, major, credits)
Class (classNumber, facId, schedule, room)
Faculty (facId, name, department, rank)
Enroll (stuId, classNumber, grade)
```

11.2.1 El árbol de consulta

Una técnica que se utiliza con frecuencia para representar expresiones del álgebra relacional es un **árbol de consulta** o **árbol de análisis sintáctico** para una expresión. Esta clase de árbol es una representación gráfica de las operaciones y operandos en una expresión de álgebra relacional. Las relaciones se representan por medio de nodos hojas y las operaciones unarias o binarias se indican por nodos internos que no son nodos hojas del árbol. Al ejecutar un árbol de consulta, un nodo interno se ejecuta cuando sus operandos se encuentran disponibles. Después el nodo es reemplazado por el resultado de la operación que representa. El nodo raíz es el último en ejecutarse y es sustituido por el resultado de todo el árbol. Si en vez de lo anterior se usa cálculo relacional, las operaciones se representan por un **grafo de consulta** o **hipergrafo de conexión**. El grafo tiene nodos para representar cada constante que aparezca en una consulta y cada atributo en una relación, lo que permite repeticiones para atributos que aparezcan en más de una relación. Los arcos que conectan nodos de atri-

butos representan condiciones de combinación entre tuplas variables. Los arcos que conectan nodos constantes con nodos de atributos representan condiciones de selección.

11.2.2 Una consulta en SQL y su traslación al álgebra relacional

La consulta "Encontrar los horarios y salones de todas las materias de cualquier especialidad de Matemáticas" podría expresarse en SQL como sigue:

```
SELECT   schedule, room
FROM     Class, Enroll, Student
WHERE    major = 'Math' AND Class.classNumber = Enroll.classNumber AND
         Enroll.stuId = Student.stuId;
```

Una traslación de álgebra relacional para esta consulta en SQL es,

```
JOIN Student, Enroll GIVING Temp1
JOIN Temp1, Class GIVING Temp2
SELECT Temp2 WHERE major = 'Math' GIVING Temp3
PROJECT Temp3 OVER schedule, room
```

o en forma simbólica,

$$\Pi_{\texttt{schedule, room}}(\sigma_{\texttt{major='Math'}}\ (\texttt{(Student |×| Enroll) |×| Class}))$$

El árbol de consulta que corresponde a esta expresión se presenta en la figura 11.2(a). Si se fuera a ejecutar esta consulta en álgebra relacional como está escrita y se muestra en el árbol, se comenzaría por formar la combinación natural (natural join) de `Student` y `Enroll` sobre su columna común, `stuId`. Como ésta es una clave para `Student`, el número de tuplas en la combinación es simplemente la cardinalidad de `Enroll`, ya que habrá exactamente una ocurrencia en `Student` para cada `stuId` en `Enroll`. Esta operación produce una tabla intermedia que se conoce como `Temp1`. Ahora se combina este resultado con `Class`, y otra vez se produce una tabla intermedia que contiene una tupla por cada tupla en `Enroll`, puesto que se está haciendo la combinación sobre `classNumber`, la clave de `Class`. La tabla intermedia tiene atributos `stuId`, `lastName`, `firstName`, `major`, `credits`, `classNumber`, `grade`, `facId`, `schedule` y `room`. Después se efectúa una selección sobre la tabla intermedia, eligiendo sólo aquellos renglones en que el valor de `major` es 'Math'. Por último, se proyecta sobre dos columnas, `schedule` y `room`, y se eliminan los duplicados si los hubiera.

11.2.3 Realizar al principio las operaciones SELECT

Si se supone que `Student` tiene 10 000 registros, `Class` tiene 2 500, y `Enroll` tiene 50 000, la ejecución que se acaba de describir involucraría la producción de dos tablas intermedias con tamaño de unos 50 000 registros completamente largos. Obsérvese que, para la primera combinación, cada registro de `Enroll` se combinará con exactamente un registro de estudiante, por lo que habrá 50 000 tuplas en `Temp1`. De manera similar, cada uno de estos registros se combinará con exactamente un registro de `Class`, lo que producirá 50 000 tuplas para `Temp2`. De éstas, sólo un número pequeño, aquellas para las que `major` = 'Math', se eligen en la operación SELECT. Se podría reducir con facilidad el tamaño de las tablas intermedias si la operación SELECT se realiza pronto y luego sólo se considera aquellos estudiantes cuya especialidad sea matemáticas. La consulta se rescribe, entonces, como sigue:

```
SELECT Student WHERE major = 'Math' GIVING T1
JOIN T1, Enroll GIVING T2
JOIN T2, Class GIVING T3
PROJECT T3 OVER schedule, room
```

o, en forma simbólica,

$\Pi_{schedule,room}$ (((σ$_{major = 'Math'}$ (Student)) |×| Enroll) |×| Class)

Al reordenar las operaciones en esta forma, se reduce el tamaño de las tablas intermedias. Si se supone que hay 400 estudiantes de matemáticas, cada uno de los cuales toma cinco cursos, se esperaría que resultaran sólo 2 000 registros de la combinación, reducción significativa en comparación con los 50 000 que se esperaban. Este ejemplo ilustra los posibles ahorros si se ejecutan al principio las operaciones SELECT. La figura 11.2(b) muestra el árbol de consulta de la expresión revisada. Observe que se ha colocado a SELECT abajo, cerca de una hoja del árbol.

11.2.4 Evaluación de condiciones de conjunción

Si la consulta involucrara un criterio de selección adicional, como encontrar los horarios y salones de los cursos que tomaran los estudiantes de matemáticas que tuvieran más de 100 créditos, se querría aplicar ese criterio tan pronto como fuera posible. En ese caso habría una condición de **conjunción**, llamada así porque involucra una conjunción, es decir un "and" en las condiciones. La consulta en SQL es:

```
SELECT schedule, room
FROM Student, Class, Enroll
WHERE major='Math' AND credits >100 AND Class.classNumber = Enroll.classNumber AND Enroll.
stuId = Student.stuId;
```

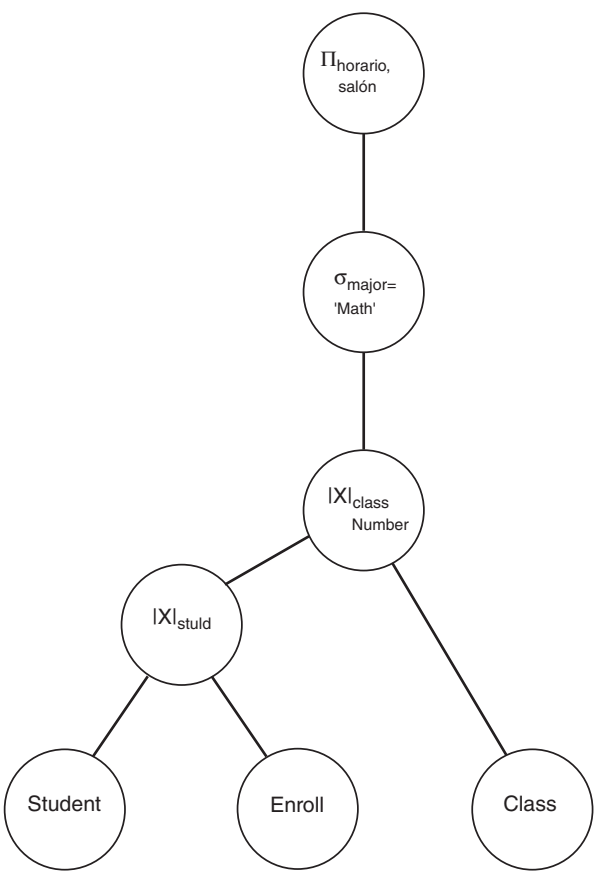
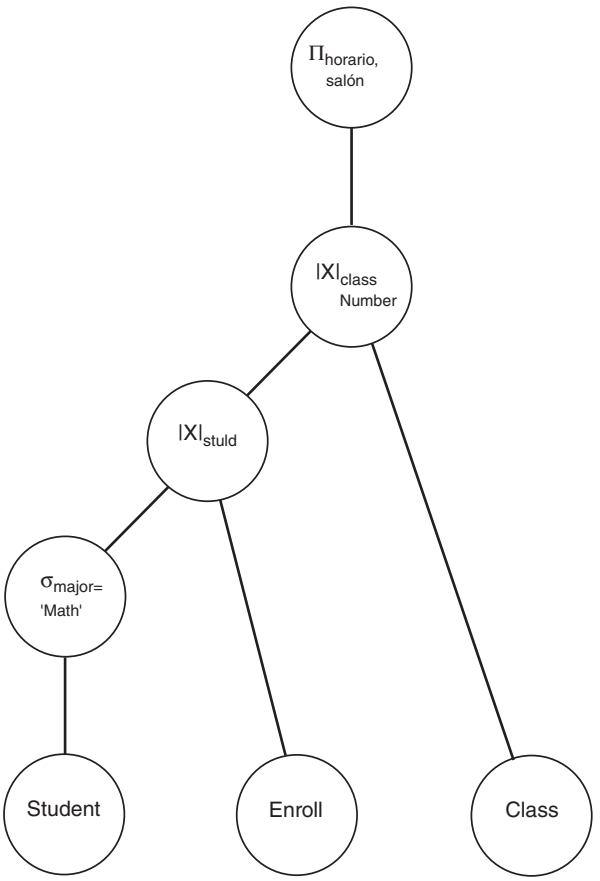


FIGURA 11.2(a)
Árbol de consulta inicial

FIGURA 11.2(b)
Árbol de consulta revisada



Para la expresión del álgebra relacional, se podrían aplicar ambos criterios de selección directamente a la relación `Student`, antes de hacer la primera combinación,

```
SELECT Student WHERE major = 'Math' AND credits > 100 GIVING T1
JOIN T1, Enroll GIVING T2
JOIN T2, Class GIVING T3
PROJECT T3 OVER schedule, room
```

O, en forma simbólica,

$$\Pi_{\texttt{schedule,room}}(((\sigma_{\texttt{major='Math' \& credits>100}}(\texttt{Student})) \mid\times\mid \texttt{Enroll}) \mid\times\mid \texttt{Class})$$

Esto reduce aún más el tamaño de la combinación. El árbol de consulta es esencialmente el que se presenta en la figura 11.2 (b), modificado con la adición de la condición de los créditos en el predicado para la selección.

Aun si el segundo criterio de selección en una selección de conjunción hiciera referencia a otra tabla, sería mejor hacerla en una etapa temprana. Por ejemplo, si se hubiera pedido encontrar los horarios y salones para todos los estudiantes de matemáticas que hubieran obtenido una calificación de F, no se podría aplicar la condición de conjunción,

```
major = 'Math' AND grade = 'F'
```

a la tabla `Student`, ya que la calificación aparece en la tabla `Enroll`. Se puede formar primero la combinación y luego aplicar la condición de conjunción y evaluar el resto de la consulta,

```
JOIN Student, Enroll GIVING T1
SELECT T1 WHERE major = 'Math' AND grade ='F' GIVING T2
JOIN T2, Class GIVING T3
PROJECT T3 OVER schedule, room
```

que es:

$$\Pi_{\text{schedule,room}}\ ((\sigma_{\text{major='Math'\&grade='F'}}(\text{Student} \mid\times\mid \text{Enroll})) \mid\times\mid \text{Class})$$

La figura 11.3(a) muestra el árbol de consulta para esta expresión. Desafortunadamente, esto nos regresa a una tabla intermedia con tamaño de 50 000 registros.

Sin embargo, si ahora se aplican las dos selecciones a sus tablas correspondientes antes de la combinación, se reduce el tamaño de las tablas intermedias,

```
SELECT Student WHERE major = 'Math' GIVING T1
SELECT Enroll WHERE grade = 'F' GIVING T2
JOIN T1, T2 GIVING T3
JOIN T3, Class GIVING T4
PROJECT T4 OVER schedule, room
```

o, en forma simbólica,

$$\Pi_{\text{schedule,room}}\ (((\sigma_{\text{major='Math'}}(\text{Student})) \mid\times\mid (\sigma_{\text{grade='F'}}(\text{Enroll}))) \mid\times\mid \text{Class})$$

La figura 11.3(b) da el árbol de consulta para esta versión. Eso es mucho más eficiente que la versión anterior, ya que reduce las tuplas seleccionadas tanto de la tabla `Student` como de la `Enroll` antes de hacer la combinación, y mantiene las tuplas de la combinación al mínimo. Observe que cuando la condición de la calificación se aplica a la tabla `Enroll`, todos los registros `Enroll` que tengan una calificación de F serán devueltos, y no sólo aquellos que pertenezcan a los estudiantes de matemáticas. La combinación natural de estas tuplas seleccionadas de `Enroll` con las tuplas seleccionadas de `Student` elimina las tuplas de `Enroll` para estudiantes de otras áreas.

11.2.5 Primero realizar PROJECT

Ya se dijo que las tablas intermedias que resultan de dos combinaciones consistían en registros muy largos. Sin importar el momento en que se hagan las selecciones o combinaciones, la tabla intermedia siempre tenía todos los atributos de las tres tablas originales. Para reducir el tamaño de las tuplas en las tablas intermedias se podría usar la proyección, pero al hacerlo se debe tener cuidado de no eliminar ninguno de los atributos que se necesitarán más adelante. Al emprender la consulta "Encontrar los horarios y salones de todas las materias tomadas por los estudiantes de matemáticas que tengan una calificación de F en cualquier materia", se podrían agregar proyecciones para reducir el tamaño de las tablas intermedias,

```
SELECT Student WHERE major = 'Math' GIVING T1
PROJECT T1 OVER stuId GIVING T2
SELECT Enroll WHERE grade = 'F' GIVING T3
JOIN T2,T3 GIVING T4
PROJECT T4 OVER classNumber GIVING T5
JOIN T5, Class GIVING T6
PROJECT T6 OVER schedule, room
```

o, en forma simbólica,

$$\Pi_{\text{schedule,room}}((\Pi_{\text{classNumber}}((\Pi_{\text{stuId}}(\sigma_{\text{major='Math'}}\ (\text{Student}))) \mid\times\mid (\sigma_{\text{grade='F'}}\ (\text{Enroll})))) \mid\times\mid \text{Class})$$

FIGURA 11.3(a)
Árbol de consulta inicial que utiliza una conjunción

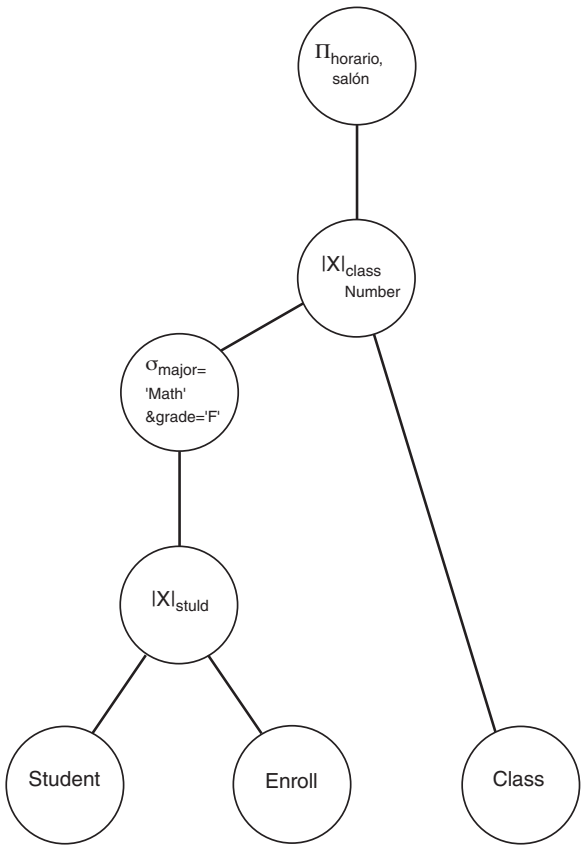
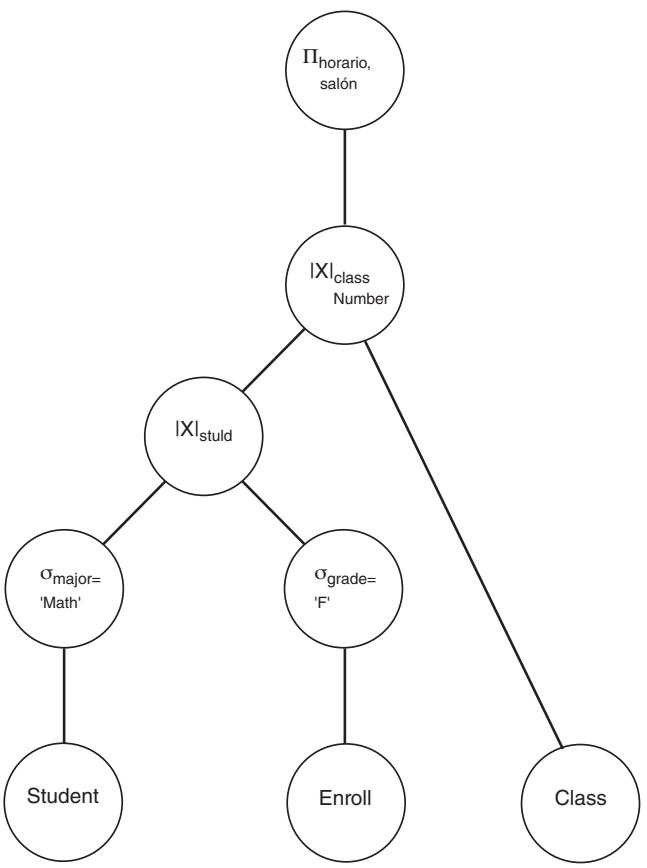


FIGURA 11.3(b)
Árbol de consulta final que utiliza conjunción



El árbol de consulta para esta expresión se presenta en la figura 11.4. El que sea conveniente realizar todas las proyecciones depende de la cantidad de espacio ahorrado y los costos en que se incurre al procesar una proyección. En cualquier caso, si se decide hacer proyecciones se deben hacer en una etapa temprana, antes de las combinaciones, cuando sea posible.

Como ilustran estos ejemplos, el orden en que se efectúan las operaciones de álgebra relacional al ejecutar una consulta afecta mucho su eficiencia. Es necesario examinar con más detalle las propiedades de estas operaciones a fin de determinar el modo en que pueden reordenarse.

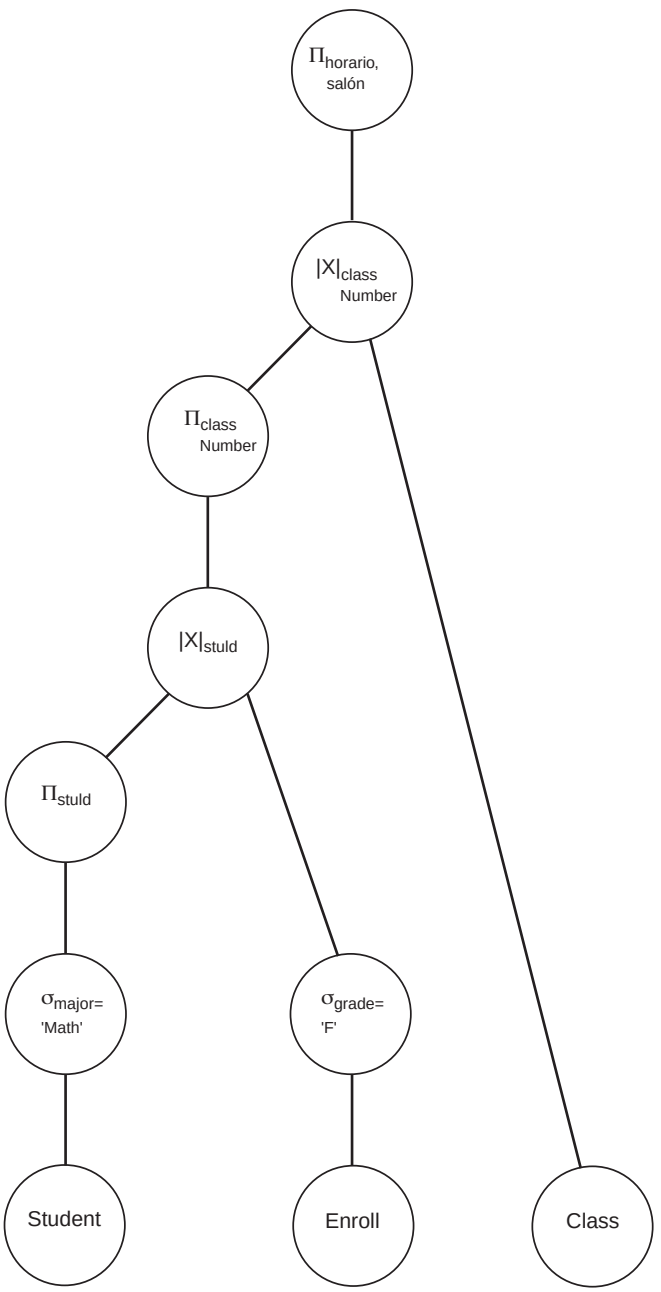


FIGURA 11.4
Árbol de consulta que muestra una proyección

11.2.6 Propiedades de la combinación natural

En el ejemplo que se ilustra en la figura 11.2(a) se ve que:

```
(Student |×| Enroll) |×| Class)
```

Las combinaciones naturales son asociativas, por lo que esta expresión se puede procesar como

```
Student |×| (Enroll |×| Class)
```

Así, se podría haber encontrado la combinación de `Enroll` y `Class` y formado una tabla intermedia grande que hubiera podido combinarse con las tuplas de `Student`. Esto habría sido menos eficiente que el método que se escogió en la figura 11.2(b). Sin embargo, hay muchas instancias en las que es posible aprovechar la asociatividad para reducir el tamaño de los resultados.

La combinación natural también es conmutativa, si se ignora el orden de las columnas. Por tanto, `Enroll |×| Class = Class |×| Enroll`. Al usar tanto la conmutatividad como la asociatividad, podríamos haber ejecutado la consulta así:

```
(Student |×| Class) |×| Enroll
```

Ésta habría sido una elección ineficiente para este ejemplo, ya que `Student` y `Class` no tienen atributos comunes, por lo que su combinación natural es un producto cartesiano. Si en `Student` se tuvieran 10 000 registros y en `Class` 2 500, como se supuso antes, la tabla intermedia tendría 25 000 000 de registros. Es obvio que para este ejemplo no hay ninguna ventaja en reacomodar las combinación. Sin embargo, si se hubiera comenzado con esta expresión se habrían podido usar las propiedades de la combinación natural para desarrollar la forma más eficiente que se vio en la figura 11.2.

11.2.7 Equivalencia de operaciones algebraicas

Los ejemplos mostraron algunas de las formas en que las expresiones del álgebra relacional se transforman en otras equivalentes de mayor eficiencia. Se considera que dos relaciones son equivalentes si tienen los mismos atributos, aun cuando el orden de los atributos sea diferente, y si los valores asociados con dichos atributos son idénticos para las tuplas correspondientes de las dos relaciones. La lista que sigue presenta algunas leyes que gobiernan las operaciones del álgebra relacional, donde R, S y T representan relaciones o expresiones que involucran relaciones. Es posible probar todas las reglas listadas, pero no se darán las demostraciones.

1. *Las combinaciones y productos son conmutativos.* La ley conmutativa se cumple para todos los tipos de combinaciones y productos. Recuerde que se utiliza la letra θ (se lee "theta") para denotar cualquier condición para una combinación. Entonces, se tiene que:

   $R \times S = S \times R$ y
   $R \mid\times\mid_{\theta} S = S \mid\times\mid_{\theta} R$ y
   $R \mid\times\mid S = S \mid\times\mid R$

2. *Las combinaciones y productos son asociativos.* Las combinaciones naturales, combinaciones theta y productos, son todas asociativas.

   $(R \times S) \times T = R \times (S \times T)$
   $(R \mid\times\mid_{\theta} S) \mid\times\mid_{\theta} T = R \mid\times\mid_{\theta} (S \mid\times\mid_{\theta} T)$
   $(R \mid\times\mid S) \mid\times\mid T = R \mid\times\mid (S \mid\times\mid T)$

3. *La selección es conmutativa.* Si p y q son condiciones de selección, entonces

   $\sigma_{p} (\sigma_{q} (R)) = \sigma_{q} (\sigma_{p} (R))$

4. *Las selecciones conjuntivas se descomponen en selecciones individuales.* Si se tiene una serie de condiciones de selección conectadas por un "and", es posible descomponer la conjunción en una secuencia de selecciones individuales.

   $\sigma_{p\&q\&...\&z}(R) = (\sigma_p(\sigma_q(...(\sigma_z(R))...)))$

   Obviamente, se puede usar la equivalencia en la otra dirección para combinar las selecciones desagregadas en una selección conjuntiva, cuando sea conveniente.

5. *Proyectos sucesivos se reducen al proyecto final.* Si $list_1$, $list_2$,..., $list_n$, son listas de nombres de atributos tales que cada lista $list_i$ contiene a la lista $list_{i-1}$, entonces,

   $\Pi_{list1}(\Pi_{list2}(...\Pi_{listn}(R)...)) = \Pi_{list1}(R)$

   Esto significa que sólo tiene que ejecutarse el último proyecto.

6. *Select y Project a veces son conmutativas.* Si la condición de selección involucra sólo los atributos de la lista de proyección, projlist, entonces select y project conmutan:

   $\Pi_{projlist}(\sigma_p(R)) = \sigma_p(\Pi_{projlist}(R))$

7. *Select y combinación (o producto) en ocasiones conmutan.* Si la condición de selección involucra sólo atributos de una de las tablas que se combinan, entonces select y combinación(o select y producto) conmutan. Si se supone que el predicado p involucra sólo atributos de R, entonces se cumple lo siguiente:

   $\sigma_p(R \bowtie S) = (\sigma_p(R)) \bowtie S$

   Esta regla también se aplica a la combinación theta y producto.

8. *A veces, Select es distributiva sobre combinación (o producto).* Si la condición de selección es conjuntiva que tiene la forma p AND q, donde p sólo involucra los atributos de la relación R y q sólo los de la relación S, entonces Select se distribuye sobre la combinación:

   $\sigma_{p\ ANDq}(R \bowtie S) = (\sigma_p(R)) \bowtie (\sigma_q(S))$

   Otra vez, la regla se aplica a la combinación theta y al producto.

9. *Project en ocasiones se distribuye sobre combinación(o producto).* Si la lista de proyección, projlist, se puede descomponer en listas separadas, list1 y list2, de modo que list1 contenga sólo atributos de R y list2 contenga sólo atributos de S, entonces:

   $\Pi_{projlist}(R \bowtie S) = (\Pi_{list1}(R)) \bowtie (\Pi_{list2}(S))$

   Para una combinación theta, la ley distributiva se cumple si la condición de combinación sólo involucra atributos de projlist. Si involucra atributos adicionales, primero deben agregarse a la lista de proyección a la izquierda del signo de equivalencia. Después se colocan a la derecha con list1 o list2, dependiendo de con cuál relación o expresión relacional están asociadas. Después de realizar la combinación, debe hacerse una proyección final sobre la proyección original.

10. *Las operaciones de conjuntos unión e intersección son conmutativas.*

    $R \cup S = S \cup R$
    $R \cap S = S \cap R$

    Sin embargo, la diferencia de conjuntos no es conmutativa.

11. *Las operaciones de conjuntos de unión e intersección son individualmente asociativas.*

    $(R \cup S) \cup T = R \cup (S \cup T)$
    $(R \cap S) \cap T = R \cap (S \cap T)$

    Aunque la unión e intersección son asociativas cuando se utilizan en forma individual, en una expresión no se pueden mezclar. La diferencia de conjuntos no es asociativa.

12. *Select se distribuye sobre la unión, intersección y diferencia de conjuntos.*

$$\sigma_p\ (R \cup S) = \sigma_p\ (R) \cup \sigma_p\ (S)$$
$$\sigma_p\ (R \cap S) = \sigma_p\ (R) \cap \sigma_p\ (S)$$
$$\sigma_p\ (R - S) = \sigma_p\ (R) - \sigma_p\ (S)$$

13. *Project se distribuye sobre la unión, intersección y diferencia de conjuntos.*

$$\Pi_{projlist}\ (R \cup S) = (\Pi_{projlist}\ (R)) \cup (\Pi_{projlist}\ (S))$$
$$\Pi_{projlist}\ (R \cap S) = (\Pi_{projlist}\ (R)) \cap (\Pi_{projlist}\ (S))$$
$$\Pi_{projlist}\ (R - S) = (\Pi_{projlist}\ (R)) - (\Pi_{projlist}\ (S))$$

14. *Project es idempotente, lo que significa repetir éste produce el mismo resultado.*

$$\Pi_{projlist}\ (\Pi_{projlist}\ (R)) = \Pi_{projlist}\ (R)$$

15. *Select es idempotente.*

$$\sigma_p(\sigma_p\ (R)) = \sigma_p\ (R)$$

Esta lista de transformaciones no es exhaustiva, pero es suficiente para ilustrar posibles sustituciones que el optimizador de consulta pudiera considerar para expresiones algebraicas dadas.

11.2.8 Heurística para la optimización de consultas

La heurística es un conjunto de "reglas prácticas" que ayudan a resolver problemas. No está garantizado que funcionen siempre para una situación particular, pero son técnicas que pueden aplicarse para ver si producen un buen resultado. A continuación se estudian algunas heurísticas que se han desarrollado para optimizar expresiones en álgebra relacional. Una regla fundamental es tratar de reducir el tamaño de los resultados intermedios por medio de hacer operaciones Select y Project tan pronto como sea posible, y ejecutar primero aquellas que reduzcan los tamaños de las tablas tanto como se pueda. Las reglas heurísticas son:

1. Realizar la operación de selección tan pronto como sea posible. Si es apropiado, hay que usar descomposiciones de la selección, la conmutatividad de la selección con Project, combinaciones y productos, y distributividad de la selección sobre la unión de conjuntos, intersección de conjuntos y diferencia de conjuntos para mover la selección tan abajo del árbol de consulta como se pueda.
2. Usar la asociatividad de la combinación, producto, unión e intersección para reacomodar las relaciones en el árbol de consulta de modo que se ejecute primero la operación de selección que produzca la tabla más pequeña.
3. Si un producto aparece como argumento de una selección, donde ésta involucra atributos de las tablas en el producto, transforme el producto en una combinación. Si la selección involucra atributos de sólo una de las tablas en el producto, primero aplique la selección a esa tabla.
4. Haga pronto la proyección. Si es apropiado utilice la descomposición de proyectos, distributividad de la proyección sobre el producto y la unión, intersección y diferencia de conjuntos, y la conmutatividad de la selección y proyección para mover la proyección tan abajo del árbol de consulta como sea posible. Examine todas las proyecciones para ver si algunas son innecesarias.
5. Si hay una secuencia de selecciones y/o proyecciones con el mismo argumento, utilice la conmutatividad o descomposición para combinarlas en una selección, una proyección o selección seguida por una proyección.
6. Si una subexpresión aparece más de una vez en el árbol de consulta, y el resultado que produce no es demasiado largo, calcúlelo una vez y guárdelo. Esta técnica es útil

en especial cuando se consultan vistas, ya que se emplea la misma subexpresión para construir la vista cada vez.

Estas reglas heurísticas se incorporan en un algoritmo que un optimizador de consultas podría usar para producir la consulta optimizada con álgebra relacional, como se vio para el ejemplo de las consultas para University en las figuras 11.2(b), 11.3(b) y 11.4. En realidad, se utilizaron estas reglas para optimizar las consultas en esos ejemplos. El paso siguiente es que el selector de estrategia elija un plan de ejecución.

11.3 Técnicas de procesamiento y estimación del costo

Para desarrollar un plan de ejecución, el selector de estrategia debe considerar las distintas maneras en que podría ejecutarse cada una de las operaciones del árbol de consulta. Evalúa cada método por medio de considerar los costos de los procesos involucrados.

11.3.1 Factores del costo

El costo de ejecución de una consulta está determinado por el costo de leer archivos, los costos del procesamiento una vez que los datos están en la memoria principal, el costo de escribir y almacenar cualesquiera resultados intermedios, el costo de comunicación y el costo de escribir los resultados finales al almacenamiento. De éstos, el factor más significativo es el costo del número de veces que se acceda al disco, es decir los costos de lectura y escritura. Al calcular estos costos, el sistema debe usar las estadísticas almacenadas en el diccionario de datos y el conocimiento acerca del tamaño, estructura y métodos de acceso para cada archivo.

Una tabla se puede almacenar en **forma compacta**, en la que los bloques contienen sólo tuplas de dicha tabla, o en **forma no compacta**, en la que tuplas de la tabla se alternan con tuplas de otras tablas. Si la forma de almacenamiento es el no compacto, es difícil calcular el número de bloques necesarios. En el peor de los casos, se tendría que suponer que cada tupla de la relación está en un bloque diferente. Si la forma del almacenamiento es el compacto, se podría estimar el número de bloques que se necesitan para conservar la tabla si se conociera el tamaño de las tuplas, número de éstas y capacidad de los bloques. Normalmente se supondrá que todas las tablas están en forma compacta.

Para el ejemplo de University, si la relación `Student` se almacena en forma compacta en bloques de 4K bytes, y cada uno de los 10 000 registros de estudiantes tiene una longitud de 200 bytes, entonces caben 20 registros por bloque (4 096/200), si se ignora cualquier almacenamiento indirecto para los encabezados de los bloques, u otros. Entonces, para mantener este archivo en forma compacta son necesarios 10 000/20 o 500 bloques. El costo del acceso es el número de bloques que deben llevarse a la memoria principal para leer, o que deben escribirse en almacenamiento secundario para la escritura. Los valores asociados con el costo del acceso son:

- t(R), el número de tuplas en la relación R
- b(R), el número de bloques necesarios para guardar la relación R
- bf(R), el número de tuplas de R por bloque, también llamado **factor de bloqueo** de R

Si R está en forma compacta, entonces:

- b(R) = t(R)/bf(R)

Las tuplas en los bloques que contienen R se pueden ordenar, con frecuencia por medio de incrementar el valor de una clave primaria, pero en ocasiones por el valor de una clave secundaria. De manera alternativa, pueden estar en orden aleatorio, o utilizar una tabla de disper-

sión mediante el valor de una clave primaria. El método de acceso que se utiliza determina la estructura del almacenamiento. Se puede acceder al archivo con un índice de la clave primaria, índices secundarios sobre atributos de claves no primarias, o una función de dispersión. Para cada tabla se puede tener un **índice agrupado**, lo que significa que las tuplas con el mismo valor del índice aparecen en el mismo bloque. Entonces, habrá otros índices **no agrupados**. Un índice puede ser **denso**, si tiene una entrada para cada tupla de la relación, o **no denso**. El índice normalmente es una estructura de niveles múltiples como un **árbol B+** u organización similar, como se describe en el apéndice A. El propósito del índice es acelerar el acceso al archivo de datos, pero primero se debe acceder al índice mismo, por lo que en el costo debe considerarse el costo indirecto adicional de tener acceso al índice. Sin embargo, por lo general es bajo en comparación con el costo de acceder a los registros de datos. Cuando se utilizan índices se considera lo siguiente:

- l(*nombre del índice*), número de niveles en un índice multiniveles o número promedio de accesos al índice que se necesitan para encontrar una entrada.

El propósito de un índice es que diga cuáles registros del archivo tienen un valor particular para un atributo. Para estimar el costo es frecuente que se necesite una estimación del número de tales registros. El diccionario de datos puede almacenar este estadístico:

- n(A,R), el número de valores distintos del atributo A en la relación R

A partir de este estadístico es posible aproximar el número de tuplas que tienen un valor particular para A. Si se supone que los valores de A están **distribuidos uniformemente** en R, entonces el número de tuplas que se espera tengan un valor particular, c, para A, que se llamará **tamaño de la selección**, o s(A = c,R), es:

- s(A=c,R) = t(R)/n(A,R)

Si el atributo, A, es una clave candidata, entonces cada tupla tendrá un valor único para ese atributo, por lo que n(A,R) = t(R) y el tamaño de la selección es 1.

Para ilustrar el empleo de la fórmula, suponga que se trata de estimar el número de estudiantes en la universidad que estudian matemáticas. Se sabe que hay 10 000 estudiantes, por lo que t(`Student`) = 10 000. Suponga que hay 25 especialidades posibles, por lo que n(`major`, `Student`) = 25. Entonces, se puede estimar el número de estudiantes de matemáticas como:

```
s(major='Math',Student) = t(Student)/n(major,Student) = 10000/25 = 400
```

Observe que se tiene que suponer que las especialidades están distribuidas uniformemente, es decir, que el número de estudiantes que eligen cada carrera es igual. Algunos sistemas guardan más información sobre la distribución de los valores, con **histogramas**, que son gráficas que muestran las frecuencias de los diferentes valores de los atributos. Se emplearía el histograma para hacer una estimación más exacta del tamaño de la selección para un valor particular. Muchos sistemas también guardan los valores máximo y mínimo de cada atributo. En los ejemplos se seguirá suponiendo una distribución uniforme.

11.3.2 Costo del procesamiento de selecciones

Estos factores se utilizan para estimar el costo de lectura de varias técnicas para realizar una selección de la forma $\sigma_{A=c}(R)$. La selección del método depende mucho de las trayectorias de acceso que existan, si se accede al archivo por una función de dispersión utilizando el (los) atributo(s) de selección, si el archivo tiene un índice en el atributo(s), y si es así, si el índice está agrupado, o si el archivo está en orden según el(los) atributo(s) de selección.

1. *Exploración de toda la tabla.* Comenzaremos con el método del "peor caso", también llamado **exploración de toda la tabla**, que se usa cuando no hay una trayectoria de acceso para el atributo, como un índice, clave de dispersión u ordenamiento sobre el

atributo, y el atributo no es una clave candidata. Como éste es el método predeterminado, siempre se comparará con éste el costo de otros métodos, y se escogerá el de menor costo. El costo es el número de bloques por tabla, puesto que se tiene que examinar cada tupla de la tabla para ver si califica para la selección, es decir:

- b(R)

Por ejemplo, si se quiere encontrar a todos los estudiantes cuyo primer nombre sea "Tom", es necesario acceder a cada bloque de `Student`. Previamente se calculó que el número de bloques de `Student` es 10 000/20, o 500, por lo que:

Costo de lectura $(\sigma_{\texttt{firstName='Tom'}}\texttt{(Student))} = \texttt{b(Student)} = 500$

2. *Uso de una clave de dispersión para recuperar un solo registro.* Si A es una clave de dispersión que tiene valores únicos, entonces se aplica el algoritmo de dispersión para calcular la dirección del registro buscado. Si no hay desbordamiento, el número de accesos esperados es 1. Si hay desbordamiento se puede obtener una estimación del número promedio de accesos requerido para llegar a un registro, lo que depende de la cantidad del desbordamiento y el método que se utilice para manejar éste. Este estadístico, al cual se llamará h, se encuentra disponible para el optimizador. El costo es:

   - h

   Por ejemplo, si se informa que el archivo `Faculty` utiliza una función de dispersión utilizando `facId` y h = 2, entonces:

   Costo de lectura $(\sigma_{\texttt{facId='F101'}}\texttt{(Faculty))} = 2$

3. *Selección de igualdad sobre una clave con el uso de un índice.* Cuando se tiene un índice sobre un campo clave, se recuperan cualesquiera de los bloques de índice que se necesiten y luego ir directamente al registro del índice. El sistema puede almacenar el número de niveles de los índices. Por tanto, si se usa l para representar el número de niveles del índice, el costo es:

   - l(nombre del índice) + 1.

   Considere la selección simple $\sigma_{\texttt{stuId='S1001'}}\ \texttt{Student}$. Como `stuId` es la clave primaria, si se tiene un índice sobre `stuId`, llamado `Student_stuId_ndx`, con 3 niveles, se tiene:

   Costo de lectura $(\sigma_{\texttt{stuId='S1001'}}\texttt{(Student))} = \texttt{l(Student\_stuId\_ndx)} + 1 = 3 + 1 = 4$

4. *Selección de igualdad en un índice no agrupado sobre un atributo de clave secundaria.* Para el predicado A = c con índice no agrupado sobre la clave secundaria A, el número de tuplas que satisfacen la condición es el tamaño de selección del atributo indizado, s(A = c, R), y se debe suponer que las tuplas están en bloques diferentes, a los que se tiene acceso de manera individual. Se supondrá que el índice apunta a todas las tuplas con valor A = c, tal vez con el empleo de una lista enlazada o un arreglo de apuntadores. El costo es el número de accesos al índice más el número de bloques para las tuplas que satisfacen la condición, o:

   - l(nombre del índice) + s(A = c, R)

   Por ejemplo, suponga que se tiene un índice no agrupado sobre el atributo especialidad en `Student`, y se desea hacer la selección $\sigma_{\texttt{major='CSC'}}\texttt{(Student)}$. Se encontrarían los registros con valor de especialidad `'CSC'` por medio de leer el nodo índice para `'CSC'` e ir de ahí a cada tupla al que apunte. Si el índice tiene 2 niveles, el costo será:

   Costo de lectura $(\sigma_{\texttt{major='CSC'}}\ \texttt{(Student))} = \texttt{1(Student\_major\_ndx)} + \texttt{s(major='CSC', Student)} = 2 + (10000/25) = 2 + 400 = 402$

   Observe que éste es sólo un poco menor que el costo del peor caso, que es de 500.

5. *Selección de igualdad con el uso de un índice agrupado sobre un atributo.* Si la selección involucra una clave secundaria, A, y se tiene un índice agrupado sobre el atributo A, se usa el tamaño de la selección para el atributo indizado dividido entre el factor de bloqueo para estimar el número de bloques que se tiene que recuperar. Observe que se supone que las tuplas de R que tienen un valor de A = c residen en bloques contiguos, por lo que este cálculo estima el número de bloques que se necesitan para almacenar estas tuplas, el cual se suma al número de bloques indizados requeridos. Entonces, el costo es:

   - l(nombre del índice) + (s(A = c, R))/bf(R)

   Por ejemplo, si el índice sobre la especialidad en el archivo `Student` fuera un índice agrupado, entonces para seleccionar a los estudiantes con especialidad de CSC se supondría que los 400 registros que se espera que tengan dicho valor para la especialidad se guardarían en bloques contiguos y el índice apuntaría al primer bloque. Después, simplemente se recuperarían los bloques siguientes para encontrar los 400 registros. El costo es:

   Costo de lectura $(\sigma_{\texttt{major='CSC'}}\texttt{(Student))} = \texttt{l(Student\_major\_ndx)} + \texttt{s(major ='CSC', Student)/bf(Student)} = 2 + (400/20) = 22$

6. *Selección sobre un archivo ordenado.* Si el predicado tiene la forma A = c, donde A es una clave con valores únicos y los registros son arreglados en orden por A, se usa una búsqueda binaria para acceder al registro con el valor de c para A. Con el uso del costo de la búsqueda binaria, el costo de este método con las condiciones mencionadas es, aproximadamente:

   - $\log_2$ b(R)

   Por ejemplo, suponga que se desea encontrar un registro de clase para un `classNumber` dado, y que se dijo que el archivo `Class` está en orden por `classNumber`. Primero se calcula el número de bloques en la tabla. Si hay 2 500 registros en `Class`, cada uno con longitud de 100 bytes, guardados en bloques de tamaño 4K, el factor de bloqueo es 4 096/100, o 40, por lo que el número de bloques es 2 500/40 o 63 (observe que se redondea hacia arriba si se necesita parte de un bloque). Luego se usa la fórmula para obtener la estimación del costo:

   $\sigma_{\texttt{classNumber='Eng201A'}}\texttt{(Class)} = \log_2(63) \approx 6$

   Si A no es un atributo clave, puede haber varios registros con un valor de c para A, y la estimación debe ajustarse para considerar el tamaño de la selección, s(A = c,R) dividido entre el número de registros por bloque. En ese caso, se obtiene una estimación de:

   - $\log_2$ b(R) + s(A = c,R)/bf(R)

7. *Selección conjuntiva con un índice compuesto.* Si el predicado es una conjunción y existe un índice compuesto para los atributos en el predicado, este caso se reduce a alguno de los anteriores, en función de si los atributos representan una clave compuesta, y de si el índice es agrupado o un árbol B+.

8. *Selección conjuntiva sin un índice compuesto.* Si una de las condiciones involucra un atributo que se usa para ordenar los registros del archivo, o tiene un índice o clave de dispersión, entonces se utiliza el método apropiado de los descritos para recuperar registros que satisfagan parte del predicado, con el uso de las estimaciones de costo dadas previamente. Una vez que se recuperan los registros, se revisa para ver si satisfacen las demás condiciones. Si no puede usarse ningún atributo para hacer una recuperación eficiente, se explora la tabla completa y se verifican todas las condiciones simultáneamente para cada tupla.

11.3.3 Proceso de combinaciones

La combinación es por lo general la operación más cara de realizar en un sistema relacional, y como es frecuente que se utilice en consultas, es importante la estimación de su costo. El costo de acceso depende del método de procesamiento y del tamaño de los resultados.

11.3.3.1 Estimación del tamaño del resultado

Un factor importante del costo que se debe considerar al procesar combinaciones es el tamaño del resultado. Si R y S son relaciones de tamaño t(R) y t(S), respectivamente, entonces para calcular el tamaño de su combinación se necesita estimar el número de tuplas de R que coincidirán con las tuplas de S en los atributos correspondientes. Existen dos casos especiales:

- Si las tablas no tienen atributos comunes, entonces la combinación se vuelve un producto, y el número de tuplas en el resultado es **t(R)*t(S)**. Por ejemplo, si se combinan `Class` y `Student`, que no tienen atributos comunes, en el resultado hay 10 000*2 500 tuplas.
- Si el conjunto de atributos comunes es una clave para una de las relaciones, el número de tuplas en la combinación no es mayor que el número de tuplas en la otra relación, ya que cada uno de éstos no coincide con más de uno de los valores clave. Para R|×|S, si los atributos comunes son una clave para R, entonces el tamaño de la combinación es **menor o igual que t(S)**. Por ejemplo, si se forma la combinación natural de `Student` y `Enroll`, como `stuId` es la clave primaria de `Student`, el número de tuplas en el resultado será el mismo que el número de tuplas en `Enroll`, o 50 000, ya que cada tupla de `Enroll` tiene exactamente una tupla que coincide con `Student`.

El caso difícil es el general en el que los atributos comunes no forman una clave de ninguna relación. Se debe estimar el número de coincidencias. Suponga que hay un atributo común, A, y que sus valores tienen distribución uniforme en ambas relaciones. Para un valor particular, c, de A en R, se esperaría que el número de tuplas en S que tienen un valor que coincide de c para A fuera el tamaño de la selección de A en S, o s(A = c,S). Ya se vio que una estimación del tamaño de la selección es el número de tuplas en S dividido entre el número de valores diferentes para A en S, o t(S)/n(A,S). Esto da el número de tuplas en S para una tupla particular en R. Sin embargo, como hay t(R) tuplas en R, cada una de las cuales tiene este número de coincidencias, el número total esperado de éstas en la combinación está dado por

- $t(R \mid\times\mid S) = t(R)*t(S) / n(A,S)$

Si se hubiera comenzado por la consideración tuplas en S para buscar coincidencias en R, se habría obtenido una fórmula un poco diferente.

- $t(R \mid\times\mid S) = t(S)*t(R) / n(A,R)$

Normalmente, se utiliza la fórmula que dé el resultado más pequeño.

El número de bloques que se necesita para escribir el resultado de una combinación depende del factor de bloqueo. Si se conoce el número de bytes para cada tupla de R y S, es posible estimar que el número de bytes en las tuplas de la combinación será aproximadamente su suma, y se divide el tamaño del bloque entre ese número para obtener el factor de bloqueo del resultado. Después, se calcula el número de bloques por medio de dividir el número esperado de tuplas entre el factor de bloqueo.

11.3.3.2 Métodos de ejecución de combinaciones

Ahora se considerará los costos de leer para diferentes métodos de ejecución de una combinación. La elección del método depende del tamaño de los archivos, si éstos se encuentran ordenados por el (los) atributo(s) de la combinación, o si los índices o claves de dispersión existan para el (los) atributo(s) de la combinación.

1. *Lazos anidados.* Éste es el método predeterminado, que debe usarse cuando no existan trayectorias especiales de acceso. Si se supone que tanto R como S son relaciones compactas, que tienen b(R) y b(S) bloques respectivamente, y se tienen dos búferes para leer, más uno para escribir el resultado, se lleva el primer bloque de R al primer búfer, y luego se lleva cada bloque de S, uno a la vez, al segundo búfer. Se compara cada tupla del bloque R con cada tupla del bloque S antes de pasar al siguiente bloque de S. Cuando se ha terminado con todos los bloques S, se lleva el siguiente bloque de R al primer búfer, y se pasan de nuevo todos los bloques de S. Este proceso se repite hasta que se hayan comparado todos los bloques de S. La figura 11.5 ilustra el proceso. El algoritmo es el que sigue:

```
para cada bloque de R
    para cada bloque de S
        para cada tupla en el bloque R
            para cada tupla en el bloque S
                si las tuplas satisfacen la condición entonces se agregan a la combinación
            end
        end
    end
end
```

El número de accesos para leer los datos con este método está dado por

- Costo de lectura (R |×| S) = b(R) + (b(R)*b(S))

ya que tiene que leerse cada bloque de R, y cada bloque de S tiene que leerse una vez para cada bloque de R. Para el ejemplo de la figura 11.5, hay 4 bloques de R y 3 bloques de S, lo que da 4 + 4*3 = 16 bloques leídos. Se leen en el orden que se muestra en la parte inferior de la figura. Observe que si R y S no fueran relaciones compactas, no se podría utilizar el método de bloque por bloque, y se habría tenido que aceptar que cada tupla se encontraba en un bloque separado. Entonces, el costo de la lectura habría sido mucho mayor, es decir,

- t(R) + (t(R)*t(S))

El método del lazo anidado se utiliza para calcular el costo de lectura para ejecutar la combinación `Student` |×| `Enroll`, si se supone que ambos son compactos. Supongamos que `Student` tiene 10 000 tuplas, que ocupan 500 bloques, y `Enroll` tiene 50 000 tuplas, cada una de 100 bytes. Si, como antes, se supone que los bloques son de 4K, el factor de bloqueo es 4 096/100, o 40, por lo que el archivo `Enroll` ocupa 50 000/40 o 1 250 bloques. Se supone que `Student` tiene 500 bloques, como se calculó previamente. Con el empleo de `Student` para el lazo exterior, el costo de lectura para este ejemplo es:

b(`Student`) + (b(`Student`) * b(`Enroll`)) = 500 + (500*1 250) = 625 500

Si los archivos no fueran compactos, entonces el costo de lectura para `Student` |×| `Enroll` sería:

t(`Student`) + (t(`Student`) * t(`Enroll`) o 10 000 + (10 000*50 000) = 500 010 000.

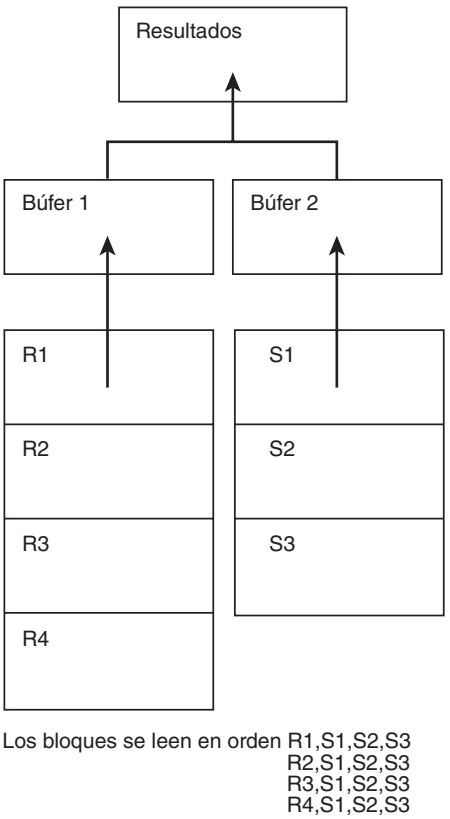


FIGURA 11.5
Combinación de lazo anidado con dos búferes para lectura

Es importante notar que el tamaño del archivo elegido para el lazo exterior tiene un efecto significativo en el costo de lectura, ya que al número de tuplas (para el caso no compacto) o el número de bloques (para el caso compacto) en el archivo del lazo exterior debe agregarse al producto de los tamaños de los dos archivos. Por tanto, para el lazo exterior debe elegirse el archivo más pequeño. Si se hubiera escogido `Enroll` para el lazo exterior, el resultado para el caso compacto habría sido 626 250, y para el caso no compacto, 500 050 000.

Si el búfer tiene capacidad para más de tres bloques, la mejor estrategia es leer en el búfer tantos bloques como sea posible del archivo en el lazo exterior, y ahorrar espacio para un solo bloque del archivo en el lazo interior, más el espacio para escribir el resultado. Por ejemplo, si b(B) es el número de bloques que el búfer puede contener, entonces, con el empleo de R como el archivo para el lazo exterior y S como el archivo para el lazo interior, se debe leer b(B)-2 bloques de R en el búfer a la vez, y sólo 1 bloque de S. El número total de bloques de R al que se acceden es b(R), pero el número total de bloques de S a los que se necesita acceder se reduce aproximadamente a b(S)*(b(R)/(b(B) – 2)). Entonces, el costo de acceder a los archivos se convierte en

- b(R) + ((b(S)*(b(R))/(b((B) – 2))

Sin embargo, si hay suficiente espacio en el búfer a fin de permitir que una de las relaciones se ajuste con espacio para un bloque más del otro archivo, más el bloque para guardar los resultados, entonces se debe elegir ése para el lazo interior. Por ejemplo, suponga que S se ajusta en la memoria principal. Entonces S tiene que leerse una sola vez, y se debe guardar en el búfer mientras se cambia entre los bloques de R uno a la vez. Entonces, el costo de leer los dos archivos compactos se reduce al costo más eficiente posible, que es:

- b(R) + b(S)

2. *Combinación mediante ordenamiento por fusión (Sort-Merge).* Una estrategia muy eficiente para hacer la combinación es aquella en que ambos archivos se ordenan según el (los) atributo(s) por combinar. En este caso, el algoritmo de combinación es una variación del algoritmo para fusionar dos archivos ordenados. Cuando los archivos se ordenan, se esperaría que todas las tuplas en R que tienen un valor específico, por decir, c, para el atributo de combinación, A, estarían en un solo bloque en cada relación. La combinación se comienza por medio de traer el primer bloque de cada relación al búfer y encontrar todos los registros en R con el primer valor para A, y luego todos los registros en S con ese mismo valor. Después éstos se unen y escriben en el archivo del resultado. Luego se avanza al siguiente valor en cada uno de los archivos, y así sucesivamente. Cada bloque de cada archivo se leerá una sola vez, a menos que haya algunos registros con el mismo valor de A en diferentes bloques, lo que es relativamente raro. Por tanto, el costo de acceder a los dos archivos es:

   - b(R) + b(S)

   Como esta combinación es tan eficiente, tal vez sea benéfico ordenar los archivos antes de hacer una combinación. En ese caso, el costo de ordenar, que depende del método que se use para hacerlo, tendría que agregarse al costo de acceder a los archivos.

3. *Uso de un índice o una clave de dispersión.* Si uno de los archivos, S, tiene un índice en el atributo común A, o si A es una clave de dispersión para S, entonces cada tupla de R se recuperaría en la forma usual por medio de leer en los bloques de R, y se emplearía el índice o algoritmo de dispersión para encontrar todos los registros de S que coincidieran. El costo de este método depende del tipo de índice. Por ejemplo, si A es la clave primaria de S, se tiene un índice primario en S y el costo de acceso es el costo de acceder a todos los bloques de R más el costo de leer el índice y acceder a un registro de S para cada una de las tuplas en R:

   - b(R) + (t(R) * (l(indexname) + 1))

   Esto se podría utilizar para encontrar Student |×| Enroll si se acepta que `Student` tiene un índice en su clave primaria `stuId`. Se accede a cada bloque `Enroll` en secuencia, y luego, para cada tupla `Enroll` en cada bloque, se utiliza el índice en `stuId` para encontrar cada registro de `Student` que coincida. Si el índice tiene 2 niveles, el costo de lectura es,

   b(`Enroll`) +(t(`Enroll`) *(2+1)) = 1 250 + (50 000*3) = 151 250

   que es alrededor de cuatro veces más eficiente que el método de lazo anidado cuyo costo se calculó antes.

   Si el índice en un atributo no es una clave primaria, se tiene que considerar el número de coincidencias en S por tupla de R, lo que incrementa el costo de lectura a:

   - b(R) + (t(R)*(1(nombredelíndice) + s(A = c,S)))

   Sin embargo, si el índice es agrupado, el estimador se reduce por medio de dividir entre el factor de bloqueo, ya que varios registros de S en cada bloque tendrán el mismo valor para A.

   - b(R) + (t(R)*(1(nombredelíndice) + s(A = c,S)/bf(S))

   Si se tiene una función hash (dispersión) en S en lugar de un índice, entonces el costo es

   - b(R) + (t(R)*h)

   donde h es el número promedio de accesos por hacer a un bloque desde su clave en S.

11.3.4 Procesamiento de otras operaciones

11.3.4.1 Proyección

Una operación de proyección involucra encontrar los valores de los atributos en la lista de proyección para cada tupla de la relación, y eliminar los duplicados, si es que existen. Si una proyección tiene una lista de proyección que contiene una clave de la relación, entonces la proyección se puede ejecutar por medio de leer cada tupla de la relación y copiar al archivo de resultados sólo los valores de los atributos en la lista de proyección. No habrá duplicados debido a la presencia de la clave. El costo de lectura es el número de bloques en la relación, que es,

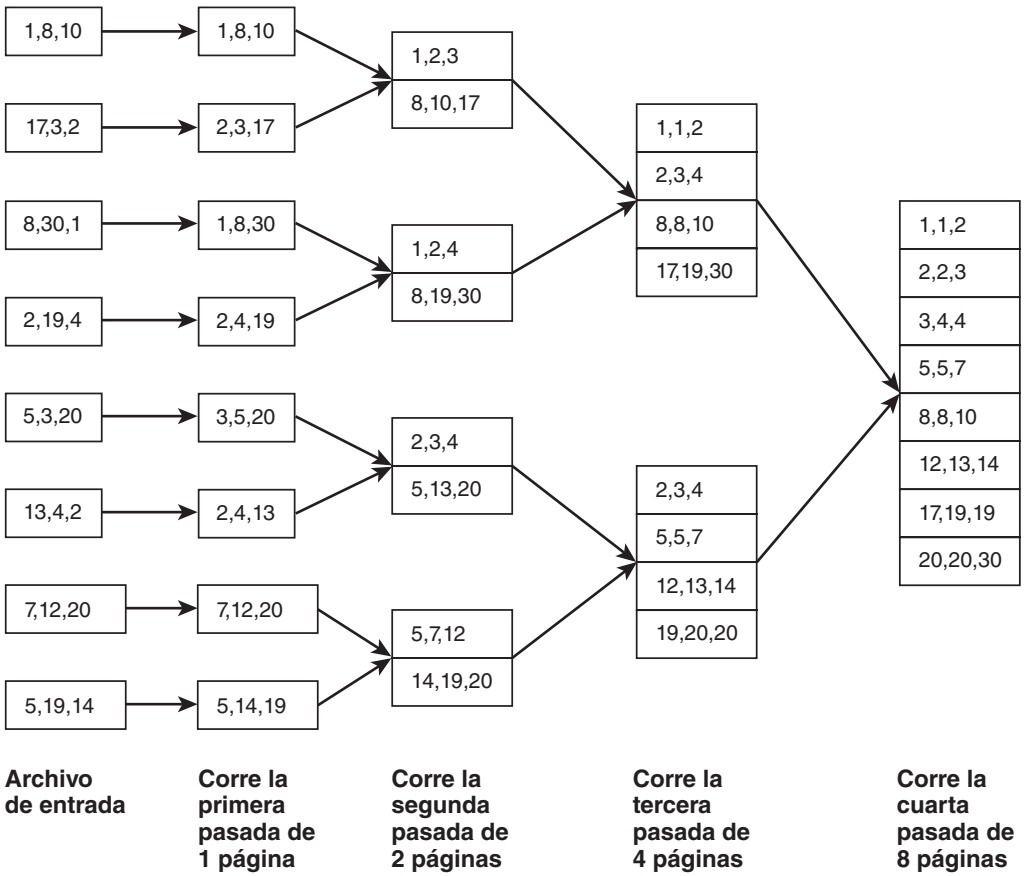
- b(R)

y el número de tuplas en el resultado será el número de tuplas en la relación, t(R). Sin embargo, en función del tamaño de la lista de proyección, las tuplas resultantes serán considerablemente más pequeñas que las tuplas de R, por lo que el número de bloques necesarios para escribir el resultado tal vez sean mucho más chicos que los que se necesitan para R.

Si la lista de proyección consiste sólo en atributos que no son clave, entonces se deben eliminar los duplicados. Un método consiste en ordenar los resultados de modo que las duplicaciones aparezcan una junto a otra, y luego se elimina cualquier tupla que sea un duplicado de la anterior hasta que ya no haya duplicaciones. Para calcular el costo, se encuentra la suma del costo de acceder a todos los bloques de la relación para crear un archivo temporal con sólo los atributos de la lista de proyección, más el costo de escribir el archivo temporal, el de ordenar el archivo temporal, el de acceder al archivo temporal ordenado desde el almacenamiento secundario a fin de eliminar las duplicaciones, y el costo de escribir el archivo final de resultados. La operación de ordenamiento es la más cara de todas éstas. En general, se requiere ordenamiento externo en un ambiente de base de datos, ya que los archivos son demasiado grandes para que quepan en la memoria principal. Si se dispone de tres búferes es posible utilizar un ordenamiento y fusión sencillo de dos vías, el cual se ilustra en la figura 11.6(a). El proceso involucra la creación en cada etapa de subarchivos ordenados llamados **corridas**. Después, las corridas se fusionan en la etapa siguiente. En la primera etapa que se muestra a la izquierda de la figura 11.6(b), cada página del archivo de entrada se lee y ordena internamente con el uso de un algoritmo estándar dentro de la memoria, como ordenamiento rápido (Quicksort). La segunda etapa se realiza con la lectura por pares de corridas a partir de los resultados anteriores, fusionándolas para producir corridas con longitud de dos páginas cada una. Se leen dos páginas a la vez, cada una en su propio búfer. El primer valor del primer búfer se compara con el primer valor del segundo búfer, y el más pequeño de los dos se escribe en el búfer de salida y se elimina de los valores por comparar. Después se compara el más grande de ambos con el siguiente valor en el búfer del que se eliminó el primero. El proceso continúa hasta que todos los valores de ambas páginas han sido escritos en las dos páginas de salida. Después se comparan las dos páginas siguientes, y así sucesivamente hasta que hayan sido leídas todas las páginas. En la tercera etapa se comparan las corridas previas con longitud de dos páginas, lo que crea corridas de cuatro páginas. En la cuarta etapa se crea una sola corrida ordenada. El número de accesos al disco para este ejemplo es 64, porque en cada etapa debe leerse y/o escribirse cada una de las ocho páginas, lo que resulta en 16 operaciones de entrada/salida para cada etapa. Para las cuatro etapas son necesarias 64 operaciones. En general, si hay n páginas en el archivo, el número de pasadas necesarias será $(\log_2 n) + 1$, y el número de accesos al disco requeridos sólo para la fase de ordenamiento será:

- $2n((\log_2 n) + 1)$

FIGURA 11.6(a)
Ordenamiento por fusión externa de dos vías



En el ejemplo se tenían ocho páginas, y se necesitaron $(\log_2 8) + 1$, que es igual a 3 + 1, o 4 pasadas. Se necesitaron $2*8((\log_2 8) + 1)$, que es 16(4) o 64 accesos al disco. El costo se reduce si se usan más de tres búferes. El costo inicial de leer todo el archivo de entrada para escoger sólo atributos en la lista de proyección se elimina si, además de ordenar, se usa la primera pasada del ordenamiento para eliminar atributos no deseados. También se pueden eliminar duplicaciones en cada etapa de la fusión, lo que resulta en la mejoría que se aprecia en la figura 11.6(b) del ejemplo, que requiere sólo 59 accesos al disco.

Es posible usar otro método si hay varios búferes disponibles. Usa dispersión, que requiere dos fases, una de partición y otra de eliminación de duplicados. En la fase de partición, se utiliza una página del búfer para leer la tabla, una página a la vez. Los búferes restantes forman las páginas de salida para las particiones de dispersión. La figura 11.7 muestra la fase de partición. Cada tupla en la página de entrada se reduce a su lista de atributos de proyección, y luego se usa una función de dispersión sobre la combinación de estos atributos. El valor de la salida determina en cuál página del búfer se colocará la tupla reducida, y en cuál partición terminará. Ya que dos tuplas con el mismo valor en los atributos de la lista de proyección pudieran dispersarse en la misma partición, se colocarán cualesquier duplicación en la misma partición. Por tanto, para cada una se ejecuta la eliminación de duplicados. Esta fase se lleva a cabo por medio de usar una nueva función de dispersión en las páginas de la partición. Se lee cada página de la partición, una a la vez, y se crea una tabla de dispersión en la memoria. Si dos tuplas son transformadas al mismo valor, se comparan y se elimina cualquier duplicado. Cuando se ha leído toda la partición, se escribe la tabla de dispersión en el disco, se despejan los búferes y se procesa la siguiente partición.

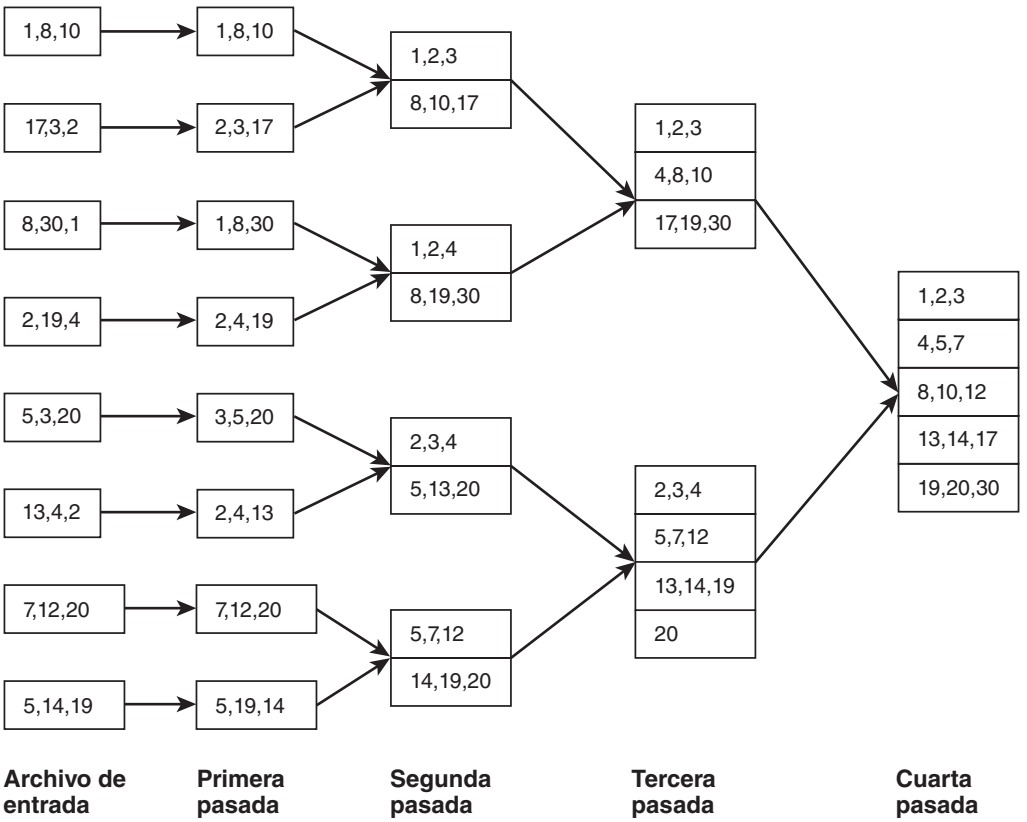


FIGURA 11.6(b)

Ordenamiento por fusión externa revisado de dos vías

11.3.4.2 Operaciones de conjuntos

Las operaciones de unión, intersección y diferencia de conjuntos sólo se pueden realizar en archivos compatibles para la unión, es decir, con estructuras idénticas. Si se ordenan ambos archivos en los mismos atributos, entonces es posible modificar el algoritmo de ordenamiento por fusión para ejecutar la unión y colocar en el archivo de resultados cualquier tupla que aparezca en alguno de los archivos originales, pero eliminando los duplicados. Para hacer la intersección se usa el mismo algoritmo básico, pero en el archivo de resultados sólo se colocan las tuplas que aparezcan en los dos archivos originales. Para la diferencia de conjuntos, `R – S`, se examina cada tupla de R y se coloca en el archivo de resultados si no tiene coincidencias en S. En cada caso, el costo es la suma del costo de acceder a todos los

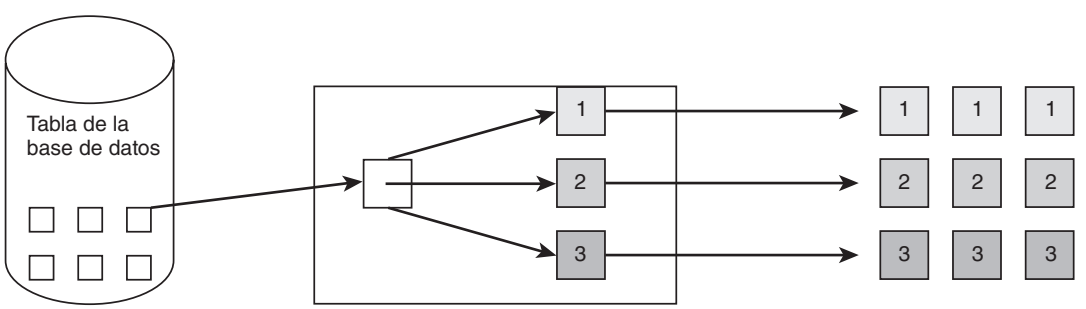


FIGURA 11.7

Proyección que usa dispersión

bloques de ambos archivos, con el ordenamiento y escritura de los archivos ordenados, el acceso a los archivos temporales para hacer la fusión, y la escritura del archivo final.

11.4 Establecimiento de ductos

En el análisis hecho hasta este momento se ha supuesto que una consulta que involucre varias operaciones de álgebra relacional es ejecutada realizando cada operación sobre la relación o relaciones involucradas, construyendo una tabla de resultados temporales y usando ésta como entrada para la operación siguiente. Este proceso, conocido como **materialización**, puede ser muy caro, ya que se tiene que considerar no sólo la suma de todos los costos de las operaciones, sino el de escribir las tablas intermedias. Una alternativa a la materialización es el **establecimiento de ductos**, en la que las tuplas "pasan a través" de una operación a la siguiente en el ducto, sin crear un archivo temporal. Por ejemplo, si se fueran a ejecutar operaciones de álgebra relacional de combinación seguidas por un proyecto, como en

$$\Pi_{\texttt{firstName,lastName}}\ \texttt{(Student |×| Enroll)}$$

usando ductos, se ejecutaría la combinación como un proceso, y se colocarían las tuplas de la combinación en un búfer, del cual cada tupla se tomaría como entrada para el proceso de proyección, y de esta forma no hay necesidad de escribir la combinación en el disco. Aunque el establecimiento de ductos es más eficiente que la materialización, no se puede utilizar en algoritmos que requieran que toda la relación esté disponible como entrada, como en el caso de la combinación mediante ordenamiento por fusión.

11.5 Optimización de las consultas en Oracle

Oracle proporciona dos tipos de optimización para bases de datos relacionales: el **basado en el costo**, que es el predeterminado, y el **basado en la regla**. Su optimizador basado en la regla brinda calificaciones de eficiencia para diferentes tipos de trayectorias de acceso. Se examina el predicado en cada enunciado SQL para determinar cuáles de las trayectorias de acceso se aplican a la consulta, y el sistema escoge aquella con la mejor calificación de entre las que se hallan disponibles. Por ejemplo, la calificación más alta (1) se da al uso de una clave primaria para recuperar una sola tupla de la tabla, una calificación media (7) se da a una clave agrupada indizada, y la más baja (15) es para el caso de explorar en una tabla completa. Si la consulta es,

```
SELECT stuId, firstName, lastName
FROM Student
WHERE major = 'CSC';
```

el optimizador no puede usar la regla de la clave primaria, ya que ésta no se da en el predicado. Si la tabla tiene un índice agrupado en la especialidad, el optimizador escogerá esa clave agrupada indizada, con calificación de 7, en lugar de explorar en toda la tabla, con calificación de 15.

El optimizador de Oracle basado en el costo utiliza estadísticos para generar estimaciones del costo de varios métodos de ejecución de consultas. El usuario es responsable de solicitar que el sistema reúna estadísticos para los objetos específicos tales como columnas, tablas o el esquema, por medio de ejecutar el paquete DBMS_STATS. También hay un comando COMPUTE STATISTICS que genera algunos estadísticos. Las frecuencias de los valores en las columnas se guardan por medio de histogramas. El sistema evalúa varios planes de ejecución para hacer la consulta y elige aquel que usa el mínimo de recursos. Sin embargo, el usuario influye en la selección al hacer **sugerencias** por medio de comentarios formateados

incluidos en la consulta. Por ejemplo, el usuario puede obligar al sistema a usar un índice particular si modifica el enunciado SQL anterior para que incluya una sugerencia +INDEX comentada, así:

```
SELECT /*+ INDEX(Student_major_ndx)*/ stuId, firstName, lastName
FROM Student
WHERE major = 'CSC';
```

Los usuarios examinan los planes de ejecución de un enunciado SQL con el comando EXPLAIN PLAN. Oracle también permite a los usuarios extender la optimización a tipos definidos por él con el uso de su opción de optimización extensible, que da al usuario el control de muchos aspectos de los planes de consulta.

11.6 Resumen del capítulo

Los sistemas de administración de bases de datos relacionales revisan cada consulta en busca de errores de sintaxis, la trasladan a una representación interna, reacomodan o cambian las operaciones para producir un orden eficiente de las operaciones, estiman el costo de ejecución de la consulta con el empleo de varios planes y escogen el más eficiente. Se utiliza un **árbol de consulta** para representar las operaciones de álgebra relacional. Con el uso de propiedades tales como conmutatividad, asociatividad y distributividad, las operaciones se reacomodan para producir un árbol de consulta equivalente pero más eficiente. La heurística guía la selección de las propiedades por aplicar. Reglas sencillas del tipo "Hacer la selección primero" dan como resultado ahorros enormes en el costo. Para escoger el plan de ejecución real, el sistema considera factores tales como el costo de acceder a un archivo, procesamiento, escritura y almacenamiento de los resultados intermedios, comunicación y escritura de los resultados finales. Debe considerarse el tamaño, estructura y trayectorias de acceso a los archivos. El sistema almacena estadísticas tales como el número de tuplas en cada relación, número de bloques, factor de bloqueo, número de niveles en cada índice, y número de valores distintos para cada atributo. Cuando se elige un plan que involucra un índice, el tipo de éste es muy significativo.

Hay varios métodos para procesar las selecciones y combinaciones, y el sistema utiliza sus estadísticas para estimar el costo de los métodos antes de elegir uno. Otras operaciones tales como la proyección y las de conjuntos tienen planes de acceso más sencillos. Cuando una consulta involucra varias operaciones, debe considerarse el costo de materializar las tablas intermedias y de acceder a ellas, además del de las operaciones. Una alternativa a la materialización es el establecimiento de ductos, en la que las tuplas producidas por una operación son pasadas a la siguiente en el ducto. Oracle usa dos optimizadores, con base en la regla y en el costo, y este último es el método predeterminado. Los usuarios tienen la responsabilidad de decir al sistema que obtenga las estadísticas apropiadas ya sea con el paquete DBMS_STATS o la opción COMPUTE STATISTICS.

Ejercicios

Esquema para los ejercicios **11.1** a **11.5**

```
Student (stuId, lastName, firstName, major, credits)
Faculty (facId, facName, department, rank)
Class (className, facId, schedule, room)
Enroll (className, stuId, grade)
```

11.1 Escriba el árbol de consulta para la siguiente expresión de álgebra relacional:

```
SELECT Class WHERE room = 'A205' GIVING T1
JOIN T1,Faculty GIVING T2
PROJECT T2 OVER facName,dept
```

o, en forma simbólica,

$\Pi_{facName,dept}\ ((\sigma_{room='A205'}\ (Class))\ |\times|\ Faculty)$

11.2 Considere la consulta en SQL siguiente para el ejemplo University:

```
SELECT  facName, schedule
FROM    Faculty, Class, Enroll, Student
WHERE      lastName = 'Burns' AND firstName='Edward'
    AND Class.classNumber =
        Enroll.classNumber AND Faculty.facId = Class.facId AND
        Student.stuId = Enroll.stuId;
```

a. Escriba una expresión de álgebra relacional para esta consulta.

b. Con el uso de equivalencias de expresiones de álgebra relacional, vuelva a escribir su expresión de álgebra relacional en una forma más eficiente, si fuera posible. Explique por qué la nueva expresión es eficiente.

11.3 Escriba una expresión eficiente de álgebra relacional para la consulta en SQL que sigue:

```
SELECT  lastName, firstName, grade
FROM    Student, Enroll, Class, Faculty
WHERE      facName = 'Tanaka' AND schedule= 'MTHF12' AND
        Class.classNumber = Enroll.classNumber AND Faculty.facId =
        Class.facId AND Student.stuId = Enroll.stuId;
```

Explique por qué es eficiente su expresión de álgebra relacional.

11.4 Escriba un árbol de consulta inicial para la siguiente expresión de álgebra relacional:

```
JOIN Class, Enroll GIVING T1
JOIN T1, Student GIVING T2
SELECT T2 WHERE facId = 'F101' GIVING T3
SELECT T3 WHERE major = 'Art' GIVING T4
PROJECT T4 OVER lastName, firstName
```

o, en forma simbólica,

$\Pi_{lastName,\ firstName}(\sigma_{major='Art'}\ (\sigma_{facId='F101'}\ ((Class\ |\times|\ Enroll)\ |\times|\ Student)))$

Con el uso de la heurística dada en la sección 11.2.3, optimice el árbol de consulta.

11.5 Considere el árbol de consulta que se ilustra en la figura 11.8. Con el empleo de la heurística dada en la sección 11.2.3, escriba dos árboles de consulta diferentes y optimizados para esta expresión.

Información para los ejercicios 11.6 a 11.12: Suponga que se tiene la siguiente información acerca de la base de datos University:

Todas las tablas están guardadas en forma compacta en bloques de 4 096 bytes de longitud.

`Student` tiene 10 000 tuplas, cada una de 200 bytes de largo. Está dispersa en `stuId`, la clave primaria, y tiene un índice secundario sobre `lastName`, con 3 niveles. Hay 8 000 valores para `lastName`, 2 000 valores para `firstName`, 25 valores para `major` y 150 valores para `credits`.

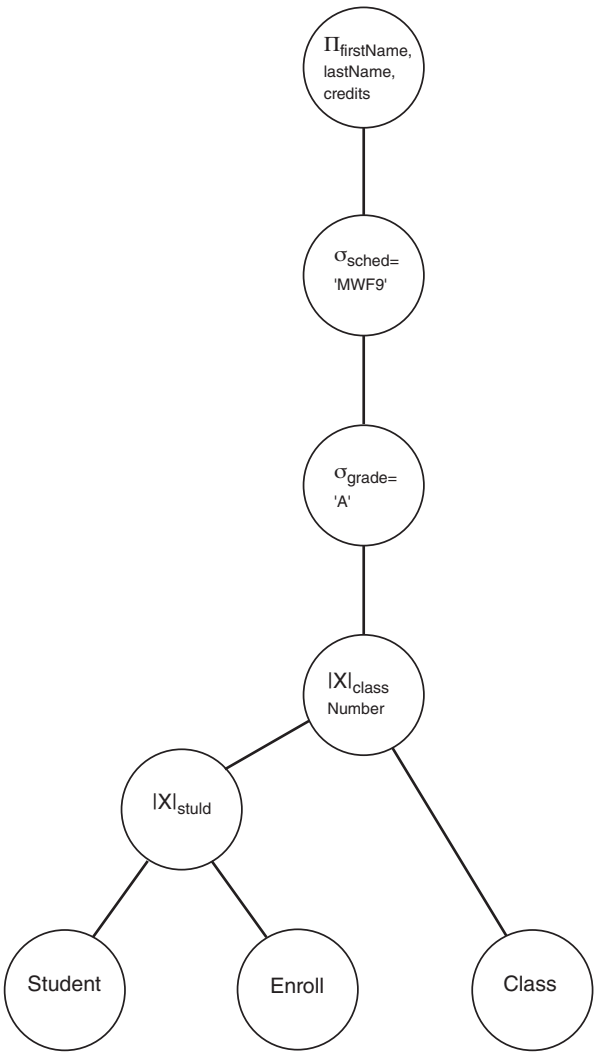


FIGURA 11.8
Árbol de consulta para el ejercicio 11.5

`Faculty` tiene 800 tuplas con 100 bytes de largo. Tiene un índice sobre `facId`, la clave primaria, con 2 niveles. Hay un índice secundario sobre `department`, con un nivel. `facName` tiene 650 valores, `department` tiene 25 valores, y `rank` tiene 4 valores.

`Class` tiene 2 500 tuplas con 100 bytes de largo. Está disperso en `classNumber`, la clave primaria, y no tiene índices secundarios, `facId` tiene 700 valores aquí, `schedule` tiene 35 valores y `room` tiene 350 valores.

`Enroll` tiene 50 000 tuplas con 100 bytes de largo. Tiene un índice compuesto sobre la clave primaria, {`classNumber`, `stuId`}, con 4 niveles, y ningún otro índice. El atributo `grade` tiene 10 valores.

11.6 a. Calcule el factor de bloqueo y el número total de bloques necesarios para cada una de las cuatro relaciones. Use esta información para las preguntas que siguen.

b. Calcule los siguientes tamaños de selección:

```
s(major='Art',Student)
s(rank='Professor',Faculty)
s(grade='A',Enroll)
  s(room='A205',Class)
```

11.7 a. Estime el número de tuplas en el resultado de `Class |×| Enroll`. ¿Aproximadamente cuántos bloques se necesitan para guardar el resultado?

b. Estime el número de tuplas y el de bloques en el resultado de `Student |×| Faculty`, donde se compara `major` en `Student` con `department` en `Faculty`.

c. Estime el número de tuplas y bloques para `Faculty |×| Class`.

d. Haga una estimación del número de tuplas y bloques para `Faculty |×| Enroll`.

11.8 Encuentre el costo de sólo leer (no el de escribir) de `Class |×| Enroll`, con el empleo de lazos anidados con:

a. `Class` en el lazo exterior y el tamaño de búfer de sólo dos bloques

b. `Enroll` como el lazo exterior y tamaño de búfer de dos bloques

c. `Class` como el lazo exterior y tamaño de búfer de 10 bloques

d. `Enroll` como el lazo exterior y tamaño de búfer de 10 bloques

11.9 Encuentre el costo total del acceso para `Faculty |×| Student`, con el uso de `major` y `department` como campos combinados y la suposición de que el búfer sólo tiene capacidad de dos bloques, con el empleo de:

a. Lazos anidados, donde se escoge la mejor colocación de las relaciones en los lazos interior y exterior. Incluya el costo de escribir los resultados.

b. El índice secundario (a B + árbol con un nivel) sobre `department` en `Faculty`. Incluya el costo de escribir los resultados, como en el inciso (a).

11.10 Determine el costo de acceso para ejecutar `Enroll |×| Student`, con el empleo de la clave de dispersión sobre `stuId`. Suponga que el número promedio de accesos para recuperar un registro de `Student` dada `stuId` es igual a 2.

11.11 Dé una estrategia eficiente para cada una de las operaciones SELECT que siguen y diga el costo de cada una:

a. `SELECT Student WHERE stuId = 'S1001'`

b. `SELECT Student WHERE lastName = 'Smith'`

c. `SELECT Student WHERE major = 'Art'`

d. `SELECT Student WHERE credits >= 100`

e. `SELECT Faculty WHERE department = 'History' and rank = 'Professor'`

f. `SELECT Class WHERE facId = 'F101'`

g. `SELECT Enroll WHERE grade = 'A'`

11.12 Suponga que se desea formar la combinación de tres vías:

```
Class |×| Enroll |×| Student
```

a. Estime el costo de (`Class |×| Enroll`) `|×| Student`, con el uso de las estrategias más eficientes para cada una de las dos combinaciones.

b. Estime el costo de `Class |×|` (`Enroll |×| Student`), con el empleo de las estrategias más eficientes para las dos combinaciones.

c. Considere el método siguiente:

```
para cada bloque de Enroll
    para cada tupla en el bloque Enroll
        usar el valor de studId de la tupla Enroll como clave de dispersión para Student
            y obtener la tupla que coincida en Student
        usar el valor de classNumber de la tupla Enroll como clave de dispersión para Class y
            obtener la tupla que coincida en Class
        escribir la combinación de las tres tuplas
    fin
fin
```

Encuentre el costo estimado de este método. Suponga que el número promedio de accesos para un valor de clave de dispersión es igual a 2.

CAPÍTULO 12

Bases de datos distribuidas

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- La definición de un sistema de bases de datos distribuidas
- Las ventajas de la distribución
- Los factores por considerar en el diseño de un sistema de bases de datos distribuidas
- Arquitecturas alternativas para un sistema distribuido
- La identificación y funciones de los componentes de un sistema de bases de datos distribuidas
- Factores para decidir la colocación de los datos
- Alternativas de colocación de los datos
- Cómo se colocan los componentes de un sistema de base de datos distribuidas
- Actualización de técnicas de sincronización
- Cómo procesa el DDBMS las solicitudes de procesamiento

12.1 Racionalidad de la distribución

Un **sistema de bases de datos distribuidas** es aquel en el que hay múltiples sitios de bases de datos unidos por un sistema de comunicaciones, en forma tal que los datos en cualquier sitio son accesibles para los usuarios de otros sitios. Normalmente, cada sitio o nodo tiene un sistema completo de procesamiento de información, con su propia función de administración de datos, personal, usuarios, hardware y software, inclusive una base de datos local, sistema de administración de base de datos y software de comunicaciones. Lo mínimo que debe tener un sitio es memoria y procesador de comunicaciones. Los sitios por lo general están separados geográficamente y están unidos por un sistema de telecomunicaciones, aunque es posible tener un sistema distribuido y comunicado por medio de una red de área local dentro de un solo edificio o área pequeña. Se pretende que los usuarios no necesiten conocer la verdadera localización de los datos a que acceden, y para ellos el sistema parece ser una base de datos local. En función de las necesidades de la organización, las bases de datos distribuidas tienen las siguientes ventajas sobre un sistema único y centralizado que dé acceso remoto a los usuarios.

1. *Autonomía local.* Un objetivo importante de cualquier sistema distribuido es permitir que el usuario tenga un control más directo sobre el sistema que utiliza. Si cada sitio tiene su propio sistema, se pueden hacer localmente más funciones básicas de procesamiento de información, como análisis de sistemas, programación de aplicaciones, operaciones y entrada de datos, lo que resulta en un mayor control local y satisfacción de los usuarios que si estas funciones se efectuaran en un sitio central, lejos de las ocupaciones y actividades de los usuarios. Al mismo tiempo, los diseñadores de un sistema distribuido deben insistir en la planeación y coordinación a fin de que se desarrollen estándares de uso obligatorio, y los sistemas individuales sean compatibles.
2. *Confiabilidad mejorada.* Un sistema distribuido es más confiable que uno centralizado, porque el procesamiento se lleva a cabo en varios sitios, por lo que la falla de un nodo no detiene a todo el sistema. Los sistemas distribuidos se diseñan para que continúen funcionando a pesar de que falle un nodo o vínculo de comunicaciones. Si falla un nodo, los usuarios de ese sitio no podrán usar el sistema, o sus solicitudes se envían a otro sitio. Los usuarios de otros sitios no se ven afectados a menos que soliciten datos almacenados en el nodo inhabilitado, o cierto procesamiento que sólo en éste se lleva a cabo. Si falla un vínculo, el nodo se puede aislar, pero el resto del sistema continúa en operación. En ciertos sistemas, el nodo tiene otras líneas que se utilizan en lugar de la que está fuera de uso.
3. *Mejor disponibilidad de datos.* Los sistemas de bases de datos distribuidas con frecuencia brindan la duplicación de los datos, de modo que si un nodo falla o el único vínculo que comunica con éste se cae, sus datos siguen disponibles, siempre y cuando se mantenga una copia en algún lugar del sistema.
4. *Mejor rendimiento.* A medida que aumentan los requerimientos de procesamiento de la información, el sistema existente tal vez sea incapaz de manejar la carga de procesamiento. En un sistema centralizado llega a ser necesario actualizarlo con cambios en el hardware y software, o cambio a un sistema nuevo, con mayor conversión de software, a fin de que satisfaga el incremento de desempeño requerido. Un sistema distribuido es básicamente modular, lo que permite que se agreguen nuevos procesadores a medida que se necesitan. En función de la topología de la red, o planta física, es fácil integrar sitios nuevos.
5. *Tiempo menor de respuesta.* Un sistema distribuido debe diseñarse de modo que los datos se almacenen en la ubicación en que se usan con mayor frecuencia. Esto per-

mite un acceso más rápido a los datos locales que con un sistema centralizado que atienda sitios remotos. Sin embargo, en un sistema distribuido mal diseñado, el sistema de comunicaciones quizá se utilice mucho y dé como resultado un tiempo de respuesta más largo.

6. *Costos menores de comunicaciones*. Si los datos que se usan localmente se guardan en forma local, los costos de las comunicaciones serán menores, ya que no se empleará la red para la mayor parte de solicitudes. En un sistema centralizado, la red de comunicaciones es necesaria para todas las solicitudes remotas. Sin embargo, se debe considerar el costo adicional del software de la base de datos, los costos adicionales de almacenamiento para múltiples copias de ítems de datos y software, y mayores costos de hardware y de distribución.

12.2 Arquitecturas para un sistema distribuido

Los factores que debe considerar el diseñador de un sistema de base de datos distribuidas al elegir una arquitectura incluyen la colocación de los datos, tipo de sistemas de comunicaciones, modelos de datos que soporta y tipos de aplicaciones. Las alternativas de colocación de los datos se estudiarán en detalle en la sección 12.5. Difieren en la cantidad de replicación que permiten de los datos. Cada alternativa impone un tipo diferente de sistema y utiliza procedimientos distintos de actualización y descomposición de solicitudes. Si el sistema de comunicaciones es lento y caro de usar se favorece un almacenamiento y procesamiento locales. Un sistema rápido y barato de comunicaciones, como una red de área local, favorece el almacenamiento y procesamiento centralizados. Los sistemas distribuidos aceptan varios modelos de datos y lenguajes de manipulación para ellos, igual que en los sistemas centralizados. En general, un diseñador debe evitar los modelos que usen una recuperación de un registro a la vez, y a cambio de eso elegir aquellos que permitan operaciones a nivel de conjunto, debido al número de mensajes que se requieren para la recuperación con navegación del programador. Ésta es una de las razones por las que el modelo relacional es el que se usa con más frecuencia en las bases de datos distribuidas. Al considerar los tipos de aplicaciones por realizar contra la base de datos escogida, el diseñador necesita estimar el tamaño de ésta, número de transacciones, cantidad de datos que requiere la transacción, complejidad de las transacciones, número de recuperaciones en relación con el número de actualizaciones, y número de transacciones que hacen referencia a datos locales en comparación con los remotos. Entre las alternativas están las siguientes:

12.2.1 Procesamiento distribuido con el uso de una base de datos centralizada

En esta arquitectura, que se ilustra en la figura 12.1, la base de datos en sí no está distribuida, pero los usuarios acceden a ella a través de una red de cómputo. El procesamiento se lleva a cabo en sitios múltiples, con el empleo de datos del sitio de la base de datos central. Éste también lleva a cabo procesamiento, con frecuencia tanto para sus aplicaciones locales como para las centralizadas. Cuando acceden a datos desde la base de datos para su procesamiento, los sitios locales se comunican sólo con el sitio central, no entre sí.

12.2.2 Sistemas cliente-servidor

Como se ilustra en la figura 12-2, en los sistemas cliente-servidor, la base de datos reside en una máquina en el extremo terminal, llamada **servidor**, y es común que los usuarios accedan a los datos a través de sus estaciones de trabajo, que funcionan como **clientes**. Las fun-

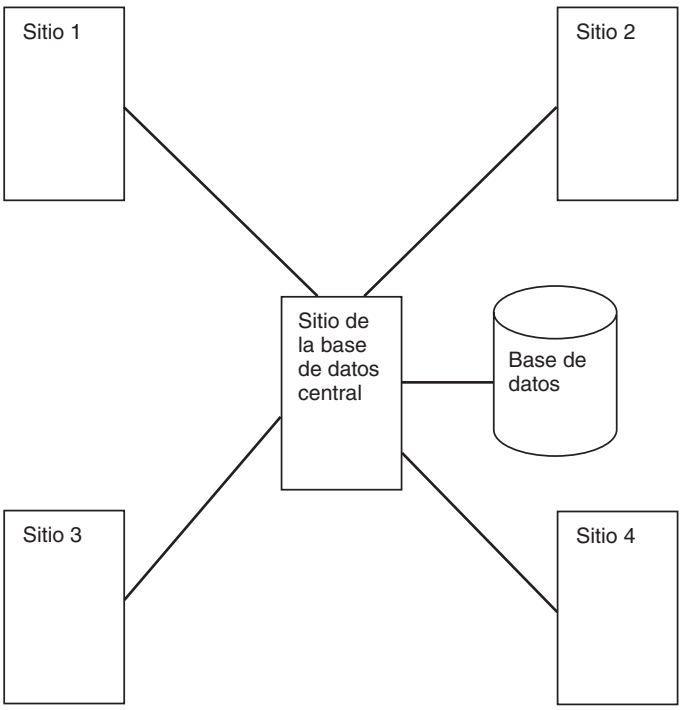


FIGURA 12.1

Sistema de procesamiento distribuido

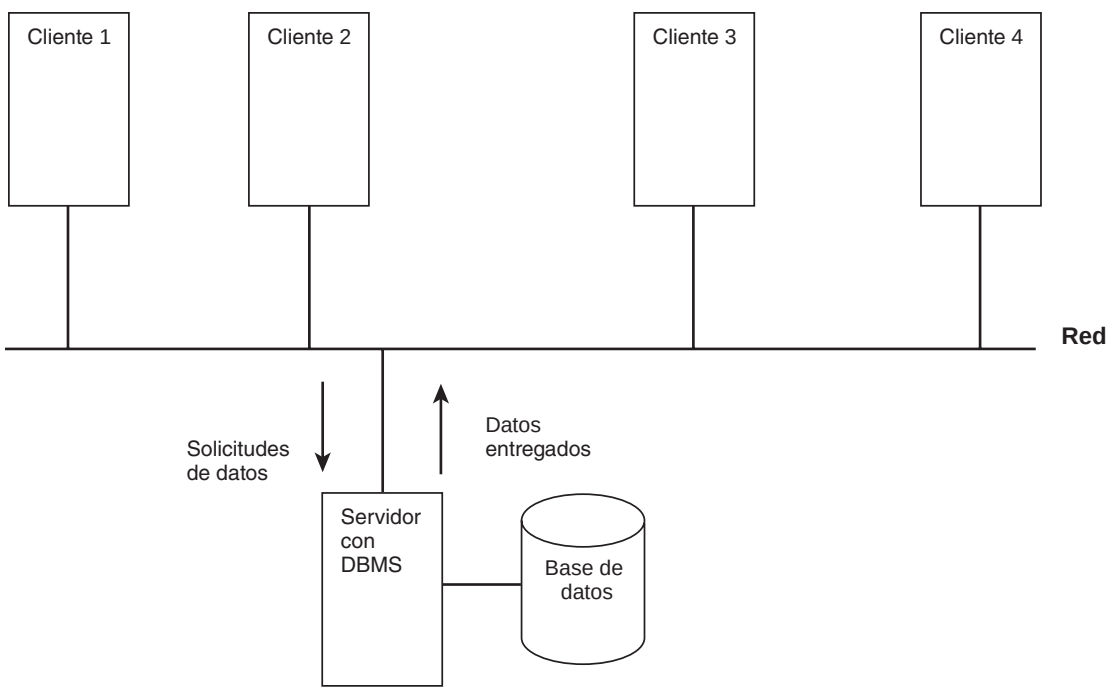


FIGURA 12.2

Sistema cliente-servidor

ciones de la base de datos se dividen entre el cliente y el servidor. El cliente proporciona al usuario la interfaz y ejecuta la aplicación lógica, mientras que el servidor administra los datos y procesa las solicitudes de éstos. En una transacción interactiva común, el usuario interactúa con la estación de trabajo cliente por medio de una interfaz gráfica provista ya sea por el sistema de base de datos o por un tercer proveedor. Además de manejar la aplicación lógica, el cliente realiza la edición inicial de las solicitudes de datos (por lo general, SQL), revisa la sintaxis de la solicitud y genera otra para la base de datos, que es enviada al servidor a través de la red. Éste valida la solicitud por medio de la revisión del diccionario de datos, autorización y las restricciones de integridad; optimiza la consulta; aplica controles de concurrencia y técnicas de recuperación; recupera los datos y los envía de regreso al cliente, que los presenta al usuario. En el cliente también se ejecutan programas de aplicación, en los que las solicitudes de datos pasan por una **interfaz de programa de aplicación (IPA)** hacia el servidor en una forma similar a la descrita. Si el cliente se apega a un estándar como el ODBC o el JDBC, se comunica con cualquier servidor que provea una interfaz estándar. A diferencia del ambiente de base de datos centralizada, el servidor no procesa aplicaciones por sí mismo.

12.2.3 Bases de datos paralelas

En la arquitectura de bases de datos paralelas hay procesadores múltiples que controlan unidades de disco múltiples que contienen a la base de datos, que puede estar particionada en los discos, o tal vez duplicada. Si la tolerancia a las fallas tiene gran prioridad, el sistema se prepara de modo que cada componente sirva como respaldo para los demás componentes del mismo tipo y se haga cargo de las funciones de cualquier componente similar que falle. Las arquitecturas de sistemas de bases de datos paralelas son de **memoria compartida**, **disco compartido**, **nada compartido**, o **jerárquicas**, que también se llaman **cluster**.

- En un sistema de **memoria compartida**, todos los procesadores tienen acceso a la misma memoria y a los discos compartidos, como se ilustra en la figura 12.3(a). La base de datos reside en los discos, ya sea que esté duplicada en ellos o particionada entre todos. Cuando un procesador hace una solicitud de datos, éstos son enviados desde cualquiera de los discos hacia los búferes de la memoria compartidos por todos los procesadores. El DBMS informa al procesador cuál página de la memoria contiene la página de datos solicitada.
- En el diseño de **disco compartido**, que se muestra en la figura 12.3(b), cada procesador tiene acceso exclusivo a su propia memoria, pero todos los procesadores tienen acceso a las unidades de disco compartidas. Cuando un procesador solicita datos, las páginas respectivas son llevadas a la memoria del procesador.
- En los sistemas de **nada compartido**, cada procesador tiene control exclusivo de su propia unidad o unidades de disco y de su memoria, como se aprecia en la figura 12.3(c), pero los procesadores se comunican uno con otro.
- En las arquitecturas **jerárquica** o **cluster**, los sistemas constituidos por nodos de memoria compartida están conectados por medio de una red, como se ve en la figura 12.3(d). Los sistemas sólo comparten comunicaciones entre sí, y la arquitectura general entre sistemas no comparte nada.

El propósito de las bases de datos paralelas es mejorar el rendimiento por medio de ejecutar las operaciones en forma paralela en los distintos dispositivos. Es esencial la partición cuidadosa de los datos de modo que sea posible la evaluación en paralelo de las consultas. La partición de los datos se hace con **partición de rango**, que significa que los registros se colocan en discos diseñados de acuerdo con un rango de valores para cierto atributo. Otros métodos son por **dispersión (hashing)** de algunos atributos, o con la colocación de los

registros nuevos en un formato **round robin** sobre discos sucesivos. Cuando se procesa una consulta, como los datos requeridos tal vez residan en discos diferentes, la consulta se descompone en subconsultas que luego se procesan en paralelo con el uso de la partición apropiada de la base de datos.

Las bases de datos paralelas que usan una arquitectura de no compartir nada proveen **velocidad** lineal, lo que significa que conforme se incrementa el número de procesadores y discos, aumenta en forma lineal la velocidad de las operaciones. También brindan **escalamiento** lineal, es decir son escalables, de modo que si se agregan más procesadores y discos el nivel de rendimiento se mantiene. Esto permite incrementar la cantidad de datos almacenados y procesados sin sacrificar el rendimiento.

12.2.4 Bases de datos distribuidas

En esta arquitectura, la base de datos está distribuida, tal vez con duplicación, entre varios sitios relativamente autónomos. La distribución es transparente para el usuario, que no necesita especificar el lugar en que se localizan los datos (**transparencia de la ubicación**). Un sistema distribuido es **homogéneo** o **heterogéneo**. En un sistema homogéneo todos los nodos usan el mismo hardware y software para el sistema de la base de datos. En un sistema heterogéneo los nodos tienen diferente hardware y/o software. Como un sistema homogéneo es mucho más fácil de diseñar y administrar, es normal que los diseñadores lo escojan. Los sistemas heterogéneos por lo general son el resultado de las situaciones en que los sitios individuales toman sus propias decisiones sobre el hardware y software, y las comunicacio-

FIGURA 12.3

Arquitecturas para bases de datos paralelas

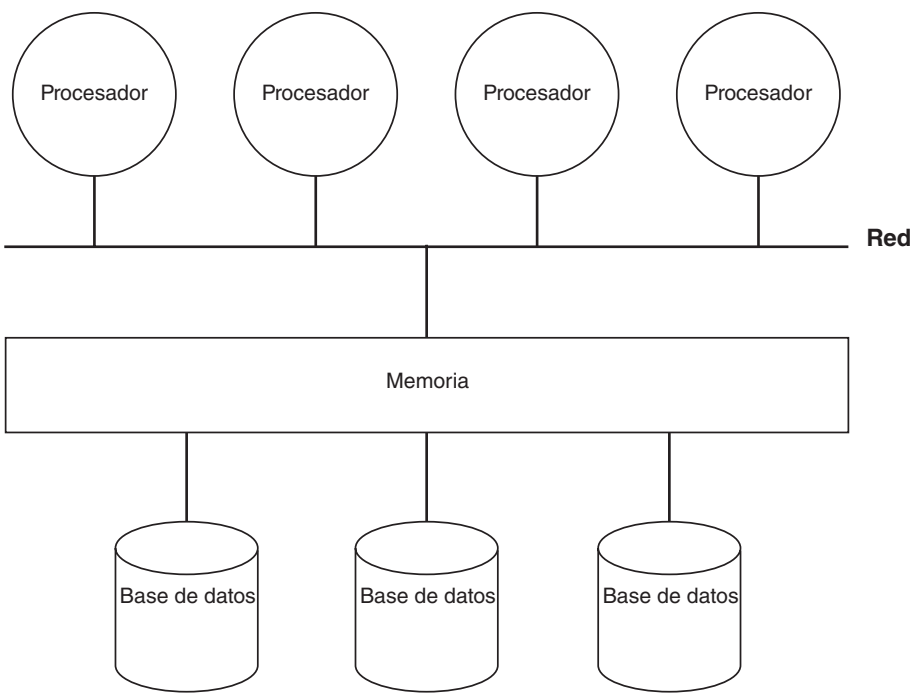


Figura 12.3(a)

Memoria compartida

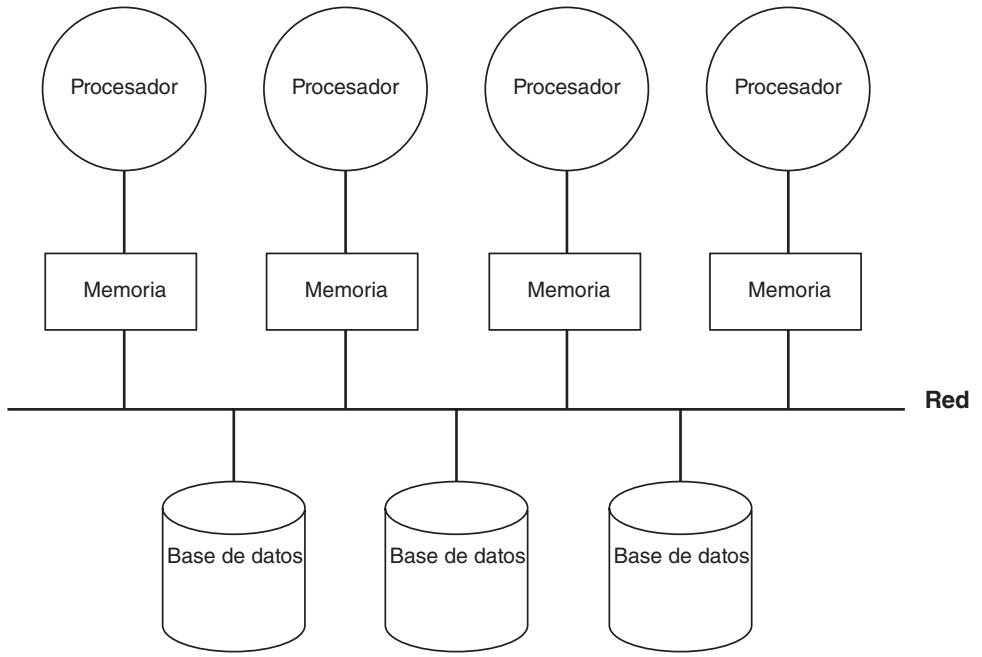


FIGURA 12.3(b)

Disco compartido

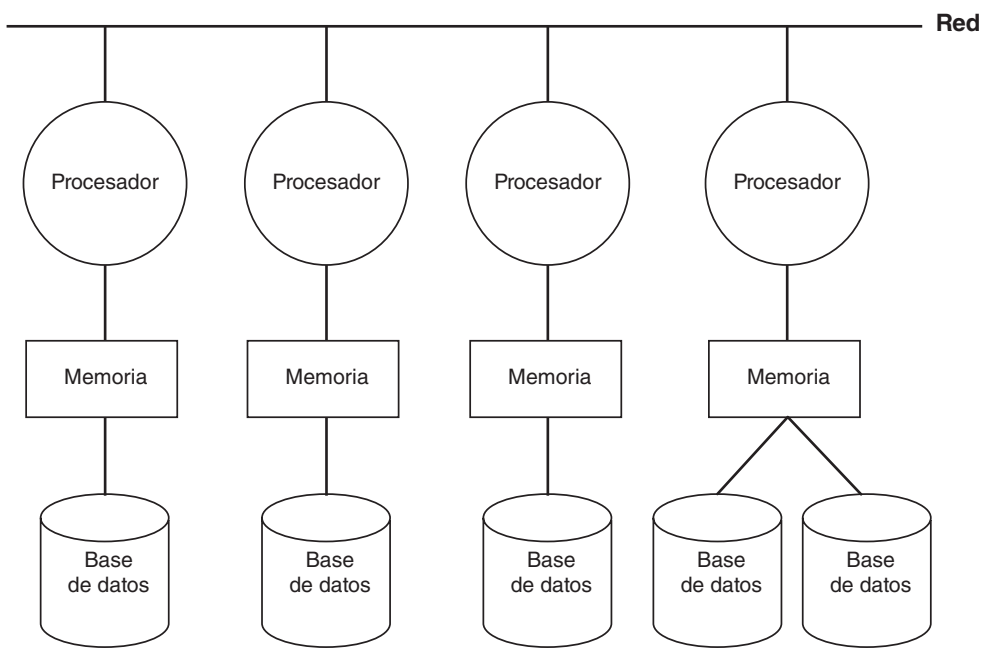


FIGURA 12.3(c)

Nada compartido

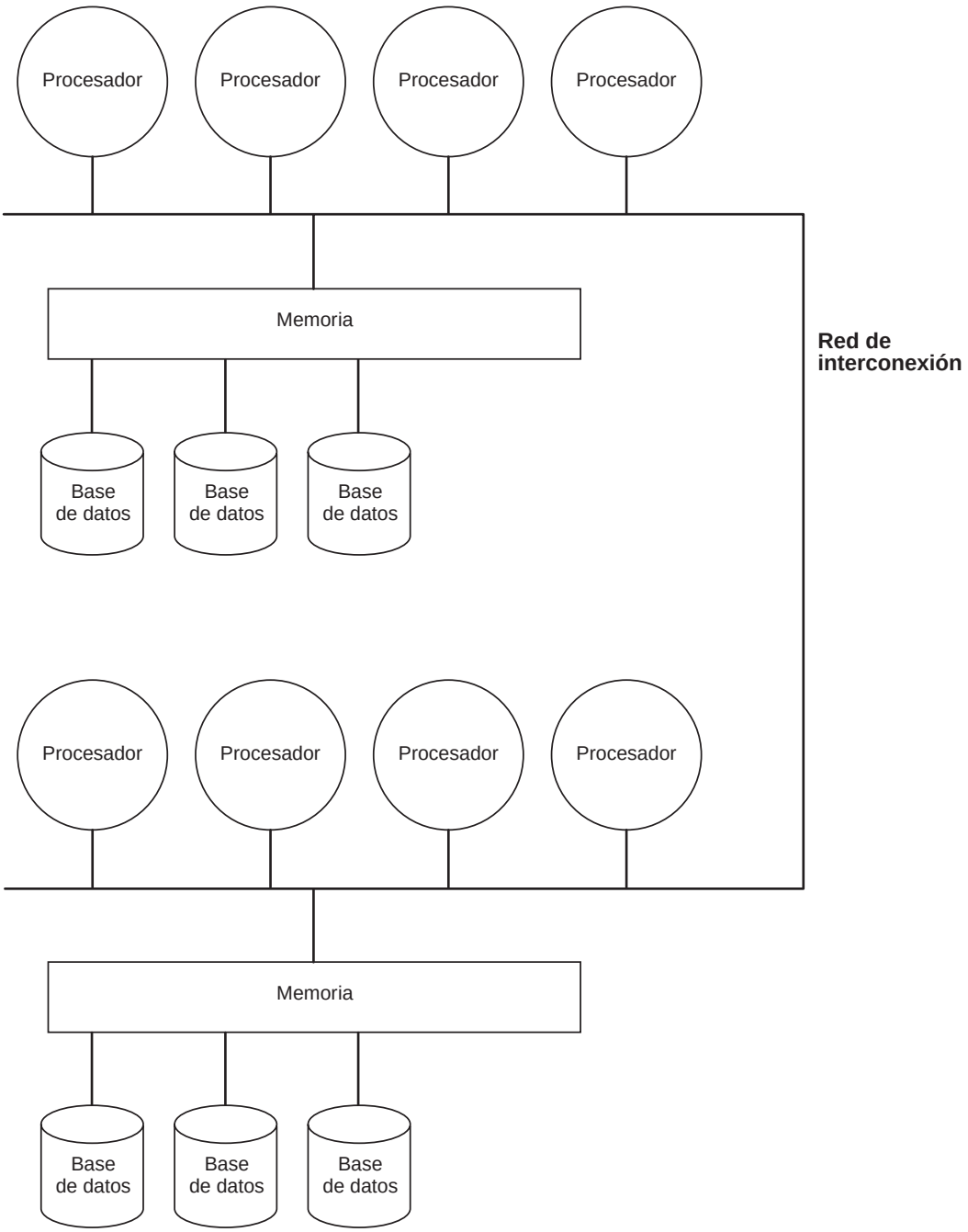


FIGURA 12.3(d)

Jerárquica

nes se agregan en una etapa posterior, lo cual es la norma. Entonces, el diseñador tiene la tarea de unificar sistemas dispares. En un sistema heterogéneo es necesario hacer traducciones que permitan que las bases de datos se comuniquen. Como la transparencia es un objetivo importante, el usuario en un sitio particular hace solicitudes en el lenguaje del sistema de administración de la base de datos en ese sitio. Entonces, es el sistema el que tiene la responsabilidad de localizar los datos, que pueden estar en otro sitio con diferente hardware y software. La solicitud incluso podría requerir la coordinación de datos almacenados en

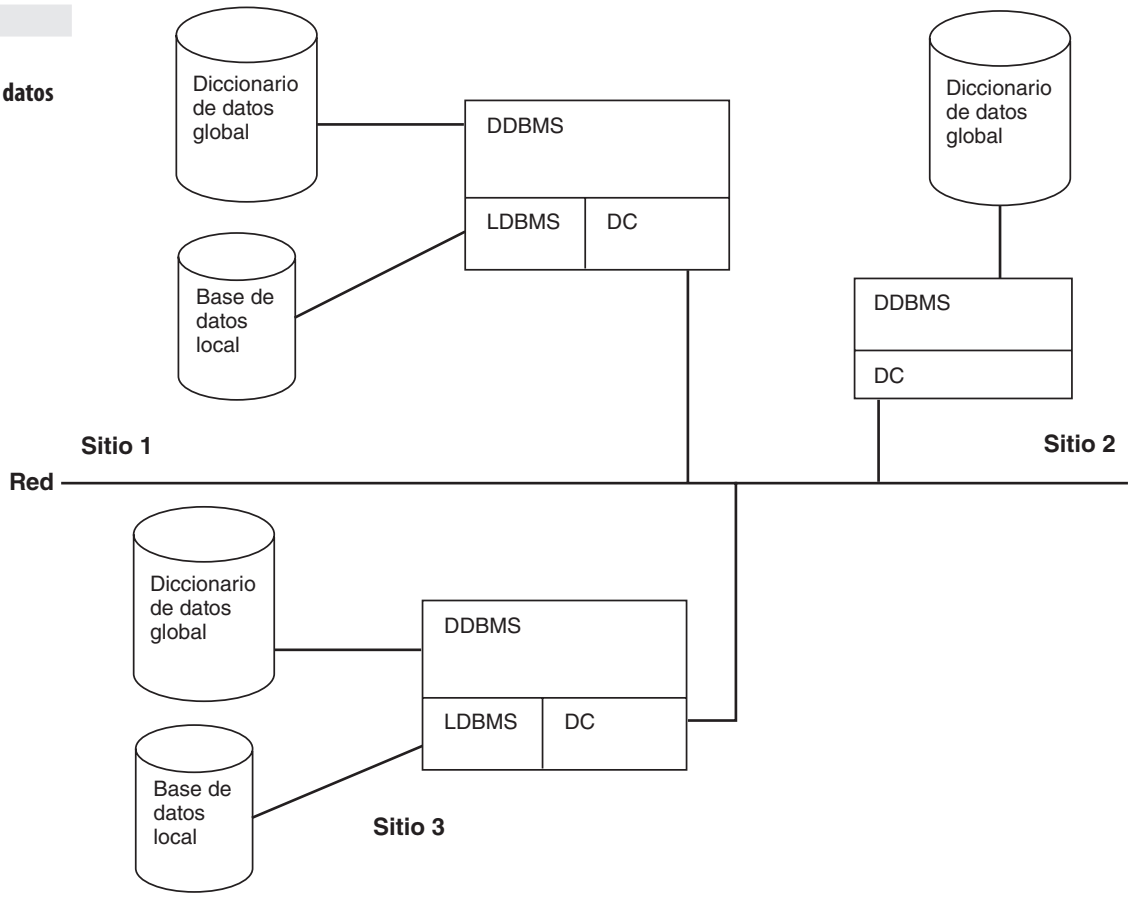
varios sitios, todos con distintos equipamientos. Si dos sitios tienen hardware diferente pero iguales sistemas de administración de la base de datos, la traducción no es demasiado difícil, y consiste sobre todo en el cambio de códigos y tamaños de la palabra. Si tienen el mismo hardware pero diferente software, la traducción sí es difícil, y requiere que el modelo y estructura de los datos de un sistema se exprese en términos de los modelos y estructuras de otro. Por ejemplo, si una base de datos es relacional y la otra utiliza el modelo orientado a objetos, éstos deben presentarse en forma de tabla a los usuarios de la base de datos relacional. Además, deben traducirse los lenguajes de las consultas. El ambiente más difícil involucra hardware y software distintos. Esto requiere la traducción de los modelos de los datos, estructuras de éstos, lenguaje de la consulta, tamaños de las palabras y códigos.

12.3 Componentes de un sistema de bases de datos distribuidas

Un sistema de bases de datos distribuidas normalmente tiene los siguientes componentes de software, que se ilustran en la figura 12.4. Observe que ciertos sitios contienen a todos, mientras que otros no.

- **Componente de comunicaciones de los datos** (**DC**). El componente de comunicaciones de los datos es el software en cada nodo que lo liga con la red. Este componente incluye la descripción completa de los nodos y líneas de la red. Identifica para cada nodo el procesamiento realizado, la capacidad de almacenamiento, la potencia de procesamiento y el estado actual. Identifica para cada vínculo los nodos que conecta, el tipo de vínculo, ancho de banda, protocolos requeridos y estado presente del vínculo.
- **Componente del sistema de administración de la base de datos local** (**LDBMS,** por sus siglas en inglés)**.** El componente del sistema de administración de la base de datos local funciona como un sistema estándar de administración de base de datos, responsable de controlar los datos locales en cada sitio que tenga una base de datos. Tiene su propio diccionario de datos locales, así como los subsistemas usuales para el control de concurrencia, recuperación, optimización de la consulta, etcétera.
- **Diccionario de datos global** (**GDD**, por sus siglas en inglés). El diccionario de datos global es una recopilación de información acerca de la base de datos distribuida. Incluye una lista de todos los aspectos de los datos, su ubicación y otra clase de información sobre cualquier dato almacenado en cualquier parte del sistema distribuido.
- **Componente del sistema de administración de la base de datos distribuida** (**DDBMS**, por sus siglas en inglés). El componente de la base de datos distribuida es el sistema de administración para la base de datos global. Tiene muchas funciones, entre las que se tienen las siguientes:
  1. *Proporciona la interfaz de usuario.* La **transparencia de la ubicación** es uno de los principales objetivos de las bases de datos distribuidas. Idealmente, el usuario no necesita especificar el nodo en que se localiza cada dato, sino que actúa como si todos los datos estuvieran almacenados localmente y se accediera a ellos por medio del DBMS local. Sin embargo, éste ignora la distribución, por lo que al DBMS local sólo deben enviarse solicitudes que se puedan satisfacer en forma local. Entonces, el DDBMS intercepta todas las solicitudes de datos y las dirige al (los) sitio(s) apropiado(s).
  2. *Localiza los datos.* Después de recibir una solicitud de datos, el DDBMS consulta al diccionario de datos global para encontrar el nodo o nodos en los que se almacenan. Si la solicitud se puede satisfacer por completo en el nodo local, pasa la

FIGURA 12.4
Componentes de un sistema de bases de datos distribuidas



consulta al DBMS local y éste la procesa. De otro modo, debe desarrollarse y ejecutarse un plan para obtener los datos.

3. *Proceso de consultas.* Las consultas se clasifican en **locales**, **remotas** o **compuestas**. Una consulta local es aquella susceptible de ser satisfecha por el DBMS local. Las solicitudes locales son simplemente enviadas al DBMS local, que localiza los datos y los envía de regreso al DDBMS, que a su vez los pasa al usuario (recuerde que el DBMS local no tiene interfaz de usuario). Una solicitud remota es la que puede cumplirse por completo en otro nodo. En este caso, el DDBMS pasa la solicitud al DBMS de ese nodo y espera la respuesta, que luego presenta al usuario. Una solicitud compuesta, también llamada **global**, es la que requiere información de varios nodos. Para procesar una solicitud compuesta, el DDBMS tiene que descomponer la consulta en varias solicitudes remotas y locales que juntas proporcionarán la información necesaria. Después dirige cada consulta al DBMS apropiado, cada uno de los cuales la procesa y envía los datos al DDBMS. Luego, el DDBMS coordina los datos recibidos a fin de formular una respuesta para el usuario.
4. *Proporciona control de concurrencia y procedimientos de recuperación en todo el sistema.* Aunque cada DBMS local es responsable de manejar los cambios y recuperación de sus propios datos, sólo el DDBMS está a cargo de problemas que involucren a todo el sistema. El control de concurrencia en todo el sistema es necesario para impedir que usuarios simultáneos interfieran uno con otro. Los procedimientos de recuperación en todo el sistema son necesarios porque si un nodo falla, aunque el DBMS local devuelva los datos a la condición que tenían en

el momento de la falla, únicamente el DDBMS puede darles seguimiento y aplicar los cambios realizados mientras el nodo estaba inactivo.

5. *Proporciona la traducción de consultas y datos en sistemas heterogéneos.* En sistemas heterogéneos con diferente hardware pero el mismo DBMS local, son necesarias traducciones menores de códigos y tamaños de la palabra, y modificaciones ligeras debidas a las diferencias en la implementación. Si los DBMS locales son diferentes, se necesitan traducciones más complejas. Esto incluye el cambio del lenguaje de las consultas de un DBMS al del otro, y de los modelos y estructuras de los datos. Si tanto el hardware como el DBMS son diferentes, se requieren ambos tipos de traducción.

12.4 Colocación de los datos

Una de las decisiones más importantes que un diseñador de bases de datos tiene que tomar se refiere a la ubicación de los datos. Esto es un factor crucial para determinar el éxito de un sistema de base de datos distribuida. Hay cuatro alternativas básicas: **centralizada**, **replicada, particionada** o **híbrida**. Algunas de éstas requieren un análisis adicional para afinar la ubicación de los datos. A fin de decidir entre las alternativas para colocar los datos es necesario considerar los factores siguientes:

1. *Localidad de los datos a que se hace referencia.* Los datos deben colocarse en el sitio en que se utilizan con más frecuencia. El diseñador estudia las aplicaciones para identificar los sitios en que se ejecutan y trata de situar los datos en forma tal que la mayoría de accesos sean locales.
2. *Confiabilidad de los datos.* Al guardar múltiples copias de los datos en sitios geográficamente remotos, el diseñador maximiza la probabilidad de que los datos sean recuperables en caso de daño físico a cualquier sitio.
3. *Disponibilidad de los datos.* Igual que con la confiabilidad, guardar muchas copias garantiza a los usuarios que los datos serán asequibles, aun si el sitio desde el que se acceden normalmente no se encuentra disponible debido a la falla del nodo o su único vínculo de comunicación.
4. *Capacidades y costos de almacenamiento.* Aunque el costo del almacenamiento de los datos por lo general es más bajo que el de su transmisión, los nodos pueden tener distintas capacidades y costos de almacenamiento. Éstos deben tomarse en cuenta al decidir dónde guardarlos. Los costos de almacenamiento se minimizan si se conserva una sola copia de cada dato.
5. *Distribución de la carga de procesamiento.* Una de las razones para escoger un sistema distribuido es repartir la carga de trabajo de modo que la potencia de procesamiento se utilice con más eficacia. Este objetivo debe balancearse con el de la ubicación de los datos a que se hace referencia.
6. *Costos de comunicación*. El diseñador debe considerar el costo de usar la red de comunicaciones para la recuperación de datos. Los costos y tiempos de recuperación son mínimos si cada sitio tiene su propia copia de todos los datos. Sin embargo, cuando los datos se actualizan, los cambios deben enviarse a todos los sitios. Si los datos son muy volátiles, esto da como resultado costos de comunicación elevados para actualizar la sincronización.

Como se aprecia en la figura 12.5, las cuatro alternativas de ubicación son las siguientes:

1. *Centralizada.* Esta alternativa consiste en una sola base de datos y DBMS guardados en una ubicación, con los usuarios distribuidos según se ilustra en la figura 12.1. No

	Alternativa			
Criterio	**Centralizada**	**Replicada**	**Particionada**	**Híbrida**
LOCALIDAD DE REFERENCIA	la más baja	la más alta	*debe ser alta	*debe ser alta
CONFIABILIDAD	la más baja	la más alta	alta para el sistema, baja para el ítem	*
DISPONIBILIDAD	la más baja	la más alta	alta para el sistema, baja para el ítem	*
COSTOS DE ALMACENAMIENTO	los más bajos	la más alta	los más bajos	*debe ser el promedio
DISTRIBUCIÓN DE LA CARGA	mala	la mejor	buena	*debe ser buena
COSTOS DE COMUNICACIÓN	los más altos	bajos, excepto para las actualizaciones	*deben ser bajos	*deben ser bajos
* Depende de datos exactos				

FIGURA 12.5

Evaluación de las alternativas de ubicación de los datos

hay necesidad de un DDBMS o diccionario de datos global, porque no hay distribución real de los datos, sólo del procesamiento. Los costos de recuperación son altos porque todos los usuarios, excepto aquellos en el sitio central, usan la red para todos los accesos. Los costos de almacenamiento son bajos, ya que se conserva sólo una copia de cada registro. No hay necesidad de actualizar la sincronización, es suficiente con el mecanismo estándar de control de concurrencia. La confiabilidad es baja y la disponibilidad mala, ya que una falla en el nodo central da como resultado la pérdida de todo el sistema. La carga de trabajo puede ser distribuida pero los nodos remotos necesitan acceder a la base de datos para ejecutar las aplicaciones, de modo que la ubicación de los datos a que se hace referencia es baja. Esta alternativa no es un verdadero sistema de base de datos distribuida.

2. *Replicada.* Con esta alternativa se mantiene en cada nodo una copia completa de la base de datos. Las ventajas son máxima referencia de ubicación, confiabilidad, disponibilidad de los datos y distribución de la carga de procesamiento. Con esta alternativa los costos de almacenamiento son los más elevados, y los de comunicación para hacer recuperaciones son bajos, pero los de las actualizaciones son altos, ya que cada sitio debe recibir cada una de éstas. Si es muy raro hacer actualizaciones, esta alternativa es buena.

3. *Particionada.* Aquí únicamente hay una copia de cada uno de los ítems de datos, pero están distribuidos en los nodos. Para permitir esto la base de datos se descompone en **fragmentos** desarticulados. Si la base de datos es relacional, los fragmentos pueden ser subconjuntos verticales en una tabla (formados por proyección) u horizontales (formados por selección) de relaciones globales. En cualquier **esquema de fragmentación horizontal**, cada tupla de toda relación debe asignarse a uno o más fragmentos de modo que al tomar la unión de éstos el resultado sea la relación original; para el caso de la partición horizontal, se asigna una tupla exactamente a un fragmento. En un **esquema de fragmentación vertical**, las proyecciones deben ser sin pérdida a fin de que las relaciones originales se reconstruyan con la combinación de los fragmentos. El método más fácil de asegurar que las proyecciones son sin pérdida es, por supuesto, incluir la clave en cada fragmento; sin embargo, esto viola la condición de desarticulación porque entonces se duplicarían los atributos clave. El diseñador puede escoger entre aceptar la duplicación de la clave, o que el sistema agregue una ID de tupla, identificación única para cada tupla, invisible para el usuario. El sistema incluiría entonces la ID de la tupla en cada fragmento vertical que incluya la tupla, y

FIGURA 12.6

Fragmentación

Student

Estudiante	lastName	firstName	major	credits
S1001	Smith	Tom	Historia	90
S1002	Chin	Ann	Matemáticas	36
S1005	Lee	Perry	Historia	3
S1010	Burns	Edward	Arte	63
S1013	McCarthy	Owen	Matemáticas	0
S1015	Jones	Mary	Matemáticas	42
S1020	Rivera	Jane	CSC	15

FIGURA 12.6(a)

Fragmento horizontal: $\sigma_{major='Math'}$ **(Estudiante)**

Student

Estudiante	lastName	firstName	major	credits
S1001	Smith	Tom	Historia	90
S1002	Chin	Ann	Matemáticas	36
S1005	Lee	Perry	Historia	3
S1010	Burns	Edward	Arte	63
S1013	McCarthy	Owen	Matemáticas	0
S1015	Jones	Mary	Matemáticas	42
S1020	Rivera	Jane	CSC	15

FIGURA 12.6(b)

Fragmento vertical: $\Pi_{stuId, major}$ **(Estudiante)**

Student

Estudiante	lastName	firstName	major	credits
S1001	Smith	Tom	Historia	90
S1002	Chin	Ann	Matemáticas	36
S1005	Lee	Perry	Historia	3
S1010	Burns	Edward	Arte	63
S1013	McCarthy	Owen	Matemáticas	0
S1015	Jones	Mary	Matemáticas	42
S1020	Rivera	Jane	CSC	15

FIGURA 12.6(c)

Fragmento mixto: $\Pi_{stuId, major}(\sigma_{major='Math'}$ **(Estudiante))**

usaría ese identificador para hacer las combinaciones. Además de los fragmentos verticales y horizontales hay **fragmentos mixtos**, obtenidos por medio de aplicaciones sucesivas de las operaciones de seleccionar y proyectar. Esta alternativa requiere un análisis cuidadoso para asegurar que los registros de los ítems de datos se asignen al sitio apropiado. La figura 12.6 proporciona ejemplos de fragmentación. Si los ítems de los datos se asignan al sitio en que se utilizan con más frecuencia, la ubicación de referencia de los datos será alta con esta alternativa. Debido a que sólo se almacena una copia de cada ítem, serán bajas la confiabilidad y disponibilidad de los datos para

un ítem específico. Sin embargo, la falla de un nodo da como resultado la pérdida exclusivamente de los datos de ese nodo, por lo que la confiabilidad y disponibilidad de todo el sistema son más altas que en el caso centralizado. Los costos de almacenamiento y comunicaciones de un sistema bien diseñado deben ser bajos. La carga de procesamiento también debe estar bien distribuida si los datos están distribuidos en forma apropiada.

4. *Híbrida.* En esta alternativa, porciones distintas de la base de datos están distribuidas de manera diferente. Por ejemplo, aquellos registros o tablas que son muy referenciados localmente estarán fragmentados, mientras que aquellos que es común utilizar en todos los nodos estarán duplicados, si las actualizaciones no son frecuentes. Aquellos que se necesitan en todos los nodos, pero se actualizan con tanta frecuencia que la sincronización es un problema, deberían estar centralizados. Esta alternativa está diseñada para optimizar la ubicación de los datos a fin de posibilitar todas las ventajas y ninguna de las desventajas de los otros métodos. Sin embargo, con este plan se requiere un análisis y un procesamiento muy cuidadosos de los datos.

Como ejemplo, considere cómo se distribuiría el esquema de la base de datos University. Se supuso que la universidad tiene un campus principal y cuatro sucursales (Norte, Sur, Este y Oeste), y que los estudiantes tienen un campus sede en el que normalmente se inscriben y toman clases. Sin embargo, se pueden inscribir en cualquier materia de cualquier campus. Los profesores enseñan en un solo campus. Cada clase se ofrece en un solo campus. Se modificará un poco el esquema para que el nuevo esquema global incluya información sobre el campus:

```
Student(stuId, lastName, firstName, major, credits, homecampus)
Faculty(facId, name, department, rank, teachcampus)
Class(classNumber, campusoffered, facId, schedule, room)
Enroll(stuid, classNumber, grade)
```

Es razonable suponer que los registros `Student` son los que se emplean con más frecuencia en el campus sede de un estudiante, por lo que se hará la partición de la relación `Student` colocando el registro de cada estudiante en su campus sede. Lo mismo se aplica para los registros `Faculty`. También podría hacerse la partición de la relación `Class` de acuerdo con el campus en que se ofrezca la clase. Sin embargo, esto significaría que las consultas sobre las ofertas en otros campos siempre requerirían el uso de la red. Si tales consultas fueran frecuentes, una alternativa sería duplicar toda la relación `Class` en todos los sitios. Como las actualizaciones a esta tabla son relativamente raras, y la mayor parte de accesos son de sólo lectura, la duplicación reduciría los costos de comunicación, por lo que se escogerá esta alternativa. La tabla `Enroll` no sería una buena candidata a ser duplicada porque durante la inscripción las modificaciones deben hacerse rápido y modificar las múltiples copias consumiría tiempo. Es deseable tener los registros `Enroll` en el mismo sitio que los registros `Student` y `Class`, pero éstos pueden no estar en la misma ubicación. Por tanto, se debería escoger centralizar los registros `Enroll` en el sitio principal, posiblemente con copias en los sitios de `Student` y `Class`. El sitio principal se puede diseñar como la copia principal de `Enroll`. El esquema de fragmentación se expresaría a través de la especificación de las condiciones para los registros de selección, como en:

$$\text{Student}_{\text{Main}} = \sigma_{\text{homecampus='Main'}}(\text{Student})$$
$$\text{Student}_{\text{North}} = \sigma_{\text{homecampus='North'}}(\text{Student})$$
. . . .
$$\text{Faculty}_{\text{Main}} = \sigma_{\text{teachcampus='Main'}}(\text{Faculty})$$
$$\text{Faculty}_{\text{North}} = \sigma_{\text{teachcampus='North'}}(\text{Faculty})$$
. . . .
$$\text{Enroll}_{\text{North}} = \Pi_{\text{stuId,classNumber,grade}}(\sigma_{\text{Student.homecampus='North'}}(\text{Enroll} \mid\times\mid \text{Student})) \cup \Pi_{\text{stuId,classNumber,grade}}(\sigma_{\text{Class.campusoffered='North'}}(\text{Enroll} \mid\times\mid \text{Class}))$$

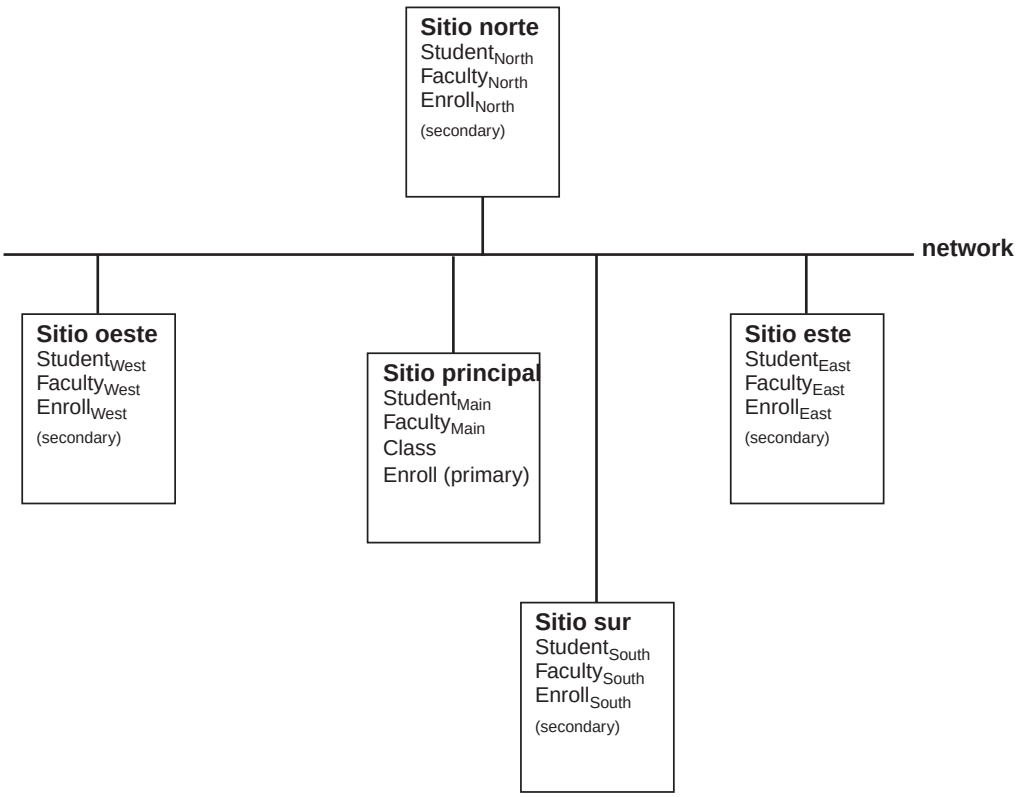


FIGURA 12.7
Ubicación de los datos para el esquema distribuido de University

No se está fragmentando la relación `Class`. Como ya se han tomado diferentes decisiones para distintas partes de la base de datos, éste es un esquema de fragmentación híbrido. La figura 12.7 ilustra las decisiones de ubicación de los datos para este ejemplo.

12.5 Transparencia

Lo ideal es que la distribución en un sistema de base de datos distribuida sea transparente para el usuario, cuya interacción con la base de datos debe ser similar a la que proporciona una base de datos centralizada y única. Las formas de transparencia que son deseables son las siguientes:

- **Transparencia de la distribución de los datos.** Este tipo de transparencia adopta diferentes formas. El usuario no necesita conocer la forma en que los datos están fragmentados, propiedad que se llama **transparencia de la fragmentación**. Los usuarios deberían desconocer la ubicación real de los registros de los datos a que acceden, propiedad denominada **transparencia de la ubicación**. Si los datos se duplicaran, los usuarios deberían ignorar el hecho de que existen copias múltiples, propiedad que recibe el nombre de **transparencia de la replicación.** Para apoyar estas propiedades, es esencial que los nombres de los ítems de datos sean exclusivos. En un sistema centralizado es fácil verificar la exclusividad. Sin embargo, en un sistema distribuido es difícil garantizar que no haya dos sitios con el mismo nombre para distintos datos. Es posible garantizar la exclusividad de todo el sistema de nombres si cada nombre de ítem de datos tiene un prefijo que sea el identificador del sitio en que se originó, llamado **sitio de nacimiento**. Es frecuente que se use la dirección del sitio de Internet. Sin embargo, esta técnica compromete la transparencia de la ubicación. El problema se resuelve si se utilizan alias para los ítems de datos, los cua-

les el sistema puede mapear para formar el nombre compuesto. Otro enfoque es tener un **servidor de nombres** con el que se revise la exclusividad de todos los nombres y éstos se registren. No obstante, el servidor podría convertirse en un cuello de botella, lo que comprometería su desempeño, y si fallara todo el sistema se vería afectado.

- **Transparencia de la heterogeneidad del DBMS.** Los usuarios que necesiten acceder a datos de un sitio remoto con un DBMS diferente al de ellos, no debieran darse cuenta de que están usando uno distinto. Sus consultas debieran enviarse en el lenguaje que utilizan normalmente. Por ejemplo, si son usuarios de Access y el sitio remoto usa Oracle, debieran poder usar el formato de consultas de Access para hacerlas. El sistema debiera encargarse de cualesquiera traducciones necesarias. El problema de dar dicha transparencia se hace más difícil si los dos sistemas utilizan diferentes modelos de datos.
- **Transparencia de la transacción.** Las propiedades ACID de las transacciones, estudiadas en el capítulo 10 para las bases de datos centralizadas, también deben estar garantizadas para las transacciones distribuidas. Por tanto, el sistema debe garantizar la **transparencia de la concurrencia**, que asegura que las transacciones concurrentes no interfieren una con otra. También debe proporcionar **transparencia de la recuperación**, que maneje la recuperación de la base de datos de todo el sistema. Estos aspectos se estudiarán con detalle en la sección 12.6.
- **Transparencia del desempeño.** Un sistema de base de datos distribuida debería tener un rendimiento comparable con otro que utilizara una arquitectura centralizada. Por ejemplo, debe emplearse un optimizador de consultas para minimizar el tiempo de respuesta de las consultas, de manera que los usuarios no esperen mucho tiempo los resultados. Esto se analizará en la sección 12.7.

12.6 Control de transacciones para bases de datos distribuidas

Cada sistema administrador de bases de datos local en un sistema de base de datos distribuido tiene un **administrador de transacciones** que incluye un controlador de concurrencia y un administrador de recuperaciones que funciona en la forma usual. Cada sitio que inicia transacciones también tiene un **coordinador de transacciones** cuya función es administrar todas las que se originen en el sitio, sean locales, remotas o globales. Para transacciones locales o remotas, el administrador de transacciones en el sitio de los datos toma el control. Para las transacciones globales, cuando una llega al sitio una secuencia normal de eventos requiere que el coordinador de transacciones en ese sitio particular:

- Comience la ejecución de la transacción
- Consulte el diccionario de datos global para dividir la transacción en subtransacciones e identifique el lugar en que se ejecutarán
- Envíe las subtransacciones a los sitos de destino por medio del sistema de comunicaciones
- Reciba los resultados de las subtransacciones desde los sitios de destino
- Administre la terminación de la transacción, con su compromiso en todos los sitios o su aborto en todos ellos.

El coordinador de transacciones es el responsable de asegurar el control de concurrencia distribuido y la recuperación distribuida. Sigue un protocolo de control de concurrencia global para garantizar la seriabilidad de las transacciones globales y mantener una bitácora para la recuperación.

12.6.1 Control de concurrencia

En el capítulo 10 se consideran los problemas del control de concurrencia en un sistema centralizado, que son los de la **actualización perdida**, el de la **actualización no comprometida**, el del **análisis inconsistente**, el de **lectura irrepetible** y el de **datos fantasma**, y también ocurren en una base de datos distribuida. Se estudió la seriabilidad de las transacciones, concepto que ahora se extenderá al ambiente distribuido. El bloqueo fue la primera solución que se vio, y se examinó el problema del candado mortal, situación en la que dos transacciones esperan ítems de datos bloqueados por la otra. Se analizaron métodos para detectar candados mortales, con el uso de un grafo de espera. También se estudiaron las estampas de tiempo como método de control de concurrencia. En un sistema de base de datos distribuido, si la base de datos es centralizada o particionada de manera que sólo haya una copia de cada ítem, y todas las solicitudes son locales (se cumplen en un sitio local) o remotas (se satisfacen por completo en otro sitio), entonces son suficientes los mecanismos usuales de bloqueo y estampas de tiempo. Surgen problemas de concurrencia más difíciles cuando hay copias múltiples de ítems de datos dispersos en todo el sistema. Para resolver este **problema de la consistencia de copias múltiples**, debe asegurarse de que cada ubicación recibe y ejecuta las actualizaciones a los ítems de datos en común. También tiene que administrar transacciones compuestas que requieran modificar diferentes registros de datos localizados en sitios distintos. Estos aspectos se abordan con el uso de protocolos que emplean el bloqueo o las estampas de tiempo de manera más global.

12.6.1.1 Protocolos de bloqueo

En el capítulo 10 se analizó el **protocolo de bloqueo de dos fases**, que garantiza la serialización para las transacciones en una base de datos centralizada. Al usar este protocolo cada transacción adquiere todos los bloqueos (candados) requeridos durante su fase de crecimiento y libera sus bloqueos durante la fase de encogimiento. La transacción no puede adquirir ningún bloqueo nuevo una vez que libera alguno. La solución para un ambiente distribuido se puede extender por medio de administrar los bloqueos en diferentes maneras.

- **Administrador de bloqueos en un solo sitio.** En este diseño hay un sitio central de administración del bloqueo al que se envían todas las solicitudes para ejecutar bloqueos. Sin importar en cuál sitio se origine una transacción, el sitio de bloqueo determina si es posible garantizar los bloqueos necesarios, con el uso de las reglas del protocolo de bloqueo de las dos fases, como se describió en el capítulo 10. El sitio que solicita lee datos desde cualquier sitio que tenga una réplica del ítem. Sin embargo, las actualizaciones deben realizarse en todos aquellos en que el ítem esté duplicado. Esto se conoce como manejo de réplicas del tipo **lee uno y escribe todos**. El esquema de administrador de bloqueo en un solo sitio es relativamente sencillo y fácil de implementar. La detección de candados mortales y la recuperación correspondiente también son sencillas y sólo involucran las técnicas analizadas en el capítulo 10, puesto que únicamente un sitio administra la información del bloqueo. Sin embargo, éste se puede convertir en un cuello de botella debido a que todas las solicitudes de bloqueo deben pasar por él. Otra desventaja es que la falla del administrador de bloqueos detiene todo el procesamiento. Esto se evita si se tiene un sitio de respaldo que se haga cargo de la administración del bloqueo en caso de falla.
- **Administrador de bloqueos distribuido**. En este esquema son varios los sitios (posiblemente todos) que tienen un administrador de bloqueos local que los maneja para los ítems de los datos en el sitio. Si una transacción solicita un bloqueo sobre un ítem de datos que reside sólo en un sitio dado, se envía al administrador de bloqueos de ese sitio una solicitud apropiada de bloqueo, y el sitio de éste procede como en el

bloqueo estándar de dos fases. La situación es más complicada si el ítem está repetido, ya que son varios los sitios en que hay copias. Con el uso de la regla lee uno y escribe todos, el sitio al que se solicita únicamente requiere un bloqueo compartido en un sitio para su lectura, pero necesita bloqueos exclusivos en todos los sitios en los que se guarda una réplica para las actualizaciones. Hay varias técnicas para determinar si los bloqueos quedan garantizados, la más simple de las cuales es que el sitio que solicita espere que los demás informen si los bloqueos están garantizados. Otro esquema requiere que el sitio que solicita espere hasta que la mayoría de los demás sitios garantice los bloqueos. El esquema de administrador distribuido elimina el problema del cuello de botella, pero el candado mortal es más difícil de determinar debido a que involucra sitios múltiples. Si los ítems de datos por actualizar están muy repetidos, los costos de la comunicación son más altos que para el método anterior.

- **Copia primaria.** Este método se utiliza con una distribución de datos repetida o híbrida en la que la referencia a datos locales tiene una evaluación alta, las actualizaciones no son frecuentes y los nodos no siempre necesitan la versión más reciente de los datos. Para cada registro repetido se escoge una copia como **copia primaria**, y el nodo en que se almacena es el **nodo dominante**. Lo normal es que para partes diferentes de los datos sean dominantes distintos nodos. El diccionario de datos global debe mostrar cuál copia es la primaria de cada registro de los datos. Cuando una actualización entra a cualquier nodo, el DDBMS determina dónde está la copia primaria y primero envía ahí la actualización, aun antes de hacer cualquier actualización local. Para comenzar la actualización, el nodo dominante bloquea su copia local del(os) ítem(s) de datos, pero ningún otro nodo lo hace. Si el nodo dominante no puede obtener un bloqueo, entonces otra transacción ya bloqueó el registro, por lo que la modificación deberá esperar a que la transacción termine. Esto garantiza la serialización de cada registro. Una vez obtenido el bloqueo, el nodo dominante realiza su modificación y luego controla los cambios de todas las demás copias, por lo general comienza en el nodo por el que entró la solicitud. El usuario en ese nodo recibe la notificación cuando se modifica la copia legal, aun cuando puede haber algunos nodos remotos en los que los nuevos valores todavía no hayan sido ingresados. En forma alternativa, la notificación podría no hacerse hasta que todas las copias hubieran sido actualizadas. Para mejorar la confiabilidad en caso de falla del nodo dominante, se elige un nodo de respaldo para el nodo dominante. En ese caso, antes de que el nodo dominante efectúe una actualización, debe enviar una copia de la solicitud para respaldarla. Si el nodo dominante no envía la notificación de cumplimiento de la actualización poco después de este mensaje, el nodo de respaldo usa un tiempo límite para determinar si el dominante ha fallado. Si ocurrió una falla, el nodo de respaldo actúa como el dominante, lo que notifica a todos los demás nodos, envía una copia de la solicitud a su propio nodo de respaldo, bloquea los datos y realiza su propia modificación y supervisa los cambios en todos los demás nodos. También es responsable de suministrar los datos necesarios cuando el nodo dominante original se recupere. Si ningún nodo dominante falla, el sistema prosigue su funcionamiento y el nodo dominante es responsable de mantener el registro de los cambios realizados mientras estuvo fuera, a fin de usarlos en la recuperación. El enfoque de la copia primaria tiene costos más bajos de comunicación y mejor desempeño que el bloqueo distribuido.
- **Protocolo de bloqueo de la mayoría.** En este esquema, en todos los sitios donde se almacenan ítems de datos existen administradores de bloqueo. Si un ítem de datos se repite en varios sitios, una transacción que requiera un bloqueo sobre el registro envía su solicitud al menos a la mitad de los sitios que tienen una repetición. Cada uno de estos sitios garantiza el bloqueo local si es posible, o retrasa la garantía si es

necesario. La transacción procede cuando ha recibido garantías de al menos la mitad de los sitios con repetición. Note que aunque varias transacciones pueden manejar la mayoría de los bloqueos compartidos sobre un registro, sólo una tiene la mayoría de los bloqueos exclusivos sobre un ítem particular. Este protocolo es más complejo que los anteriores, y requiere más mensajes. El candado mortal entre sitios también es difícil de determinar.

Sin importar cómo se administren los bloqueos, se aplica el protocolo de bloqueo de dos fases. La transacción debe obtener todos sus bloqueos antes de liberar cualquiera. Una vez que se libera alguno, no se pueden obtener más bloqueos.

12.6.1.2 Detección de candado mortal global

Los mecanismos de bloqueo pueden dar como resultado un candado mortal, situación en la que hay transacciones que esperan una a otra, como se vio en el capítulo 10. La detección de candados mortales en una base de datos centralizada involucra la construcción de un grafo de **espera**, en el que los nodos representan las transacciones y cada arista es la unión entre el nodo T1 y el T2, lo que indica que la transacción T1 espera un recurso suministrado por la transacción T2. En el grafo, un **ciclo** indica que ha ocurrido un candado mortal. En un ambiente de distribución, cada sitio construye su **grafo de espera local**. Los nodos en el grafo representan transacciones tanto locales que usan datos locales, como transacciones no locales que emplean los datos en ese sitio. Por tanto, cualquier transacción que emplee datos de un sitio está representada por un nodo en el grafo, aun si la transacción se está ejecutando en otro sitio. Si ocurre un ciclo en un grafo de espera local, en ese sitio ha ocurrido un candado mortal, y se detecta y rompe del modo usual: con la elección de una víctima y deshaciéndola. Sin embargo, puede ocurrir un **candado mortal distribuido** sin que aparezca un ciclo en algún grafo de espera local. La figura 12.8(a) muestra una programación global con cuatro sitios que ejecutan transacciones de manera concurrente. En el sitio 1, la transacción T2 espera un bloqueo compartido sobre el registro de datos A, bloqueada en forma concurrente exclusivamente por T1. En la figura 12.8(b), el grafo de espera local para el sitio 1 muestra una arista que va de T2 a T1, pero no hay ciclo. En forma similar, los sitios 2 a 4 tienen grafos de espera, cada uno de los cuales no tiene ciclo, lo que indica que en ese sitio no hay ningún candado mortal.

FIGURA 12.8

Grafos de espera para la detección de candado mortal

Tiempo	Sitio1	Sitio2	Sitio3	Sitio4
t1	T1: Xbloqueo a	T2: Sbloqueo g	T3: Sbloqueo m	T4: Xbloqueo q
t2	T1: Xbloqueo b	T2: Xbloqueo h	T3: Xbloqueo n	T4: Sbloqueo r
t3	T2: solicita Sbloqueo a	T3: solicita Xbloqueo g	T4: solicita Sbloqueo n	T1: solicita Sbloqueo q
t4	. . . T2espera . . .	. . . T3espera . . .	. . . T4espera . . .	. . . T1espera . . .

FIGURA 12.8(a)

Programación global de las transacciones T1, T2, T3 y T4

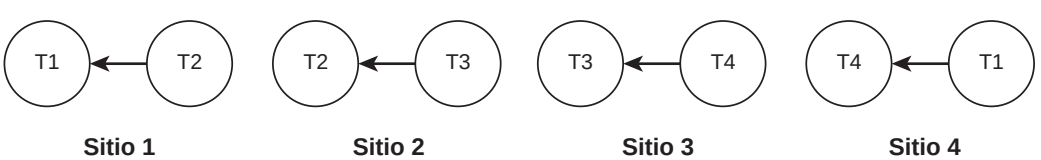


FIGURA 12.8(b)

Grafos de espera locales sin ciclos

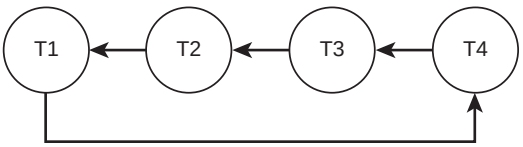


FIGURA 12.8(c)
Grafo de espera global con ciclo

Sin embargo, cuando se toma la unión de los grafos, como se ve en la figura 12.8(c), se observa que el grafo de espera global tiene un ciclo. En el sitio 4, T1 espera a T4, que espera en el sitio 3 a T3, que a su vez espera en el sitio 2 a T2, que, finalmente, espera en el sitio 1 a T1. Por tanto, se tiene un ciclo que indica un candado mortal distribuido. Ningún sitio por sí mismo podría detectar el candado mortal con exclusivamente su grafo local. Entonces, el sistema necesita mantener un **grafo de espera global** además de los locales. Puede haber un sitio único identificado como **coordinador de detección de candados mortales** que mantenga al grafo, o la responsabilidad podría compartirse en varios sitios a fin de evitar los cuellos de botella y mejorar la confiabilidad.

El coordinador de detección de candados mortales es responsable por construir un grafo global que use información de los administradores de bloqueo en todos los sitios. Estos administradores transmiten mensajes que informan al coordinador siempre que se agrega o elimina un arco de sus grafos locales. El coordinador busca continuamente ciclos en el grafo global, y si detecta uno es responsable de seleccionar una víctima e informar a todos los sitios participantes de que deben anular esa transacción.

12.6.1.3 Protocolos de estampas de tiempo

En el capítulo 10 se analizaron las estampas de tiempo en un sistema centralizado, y se estudiaron los protocolos que garantizan que las transacciones se ejecutaran en el orden de las estampas de tiempo. En esencia, se aplican los mismos protocolos a los sistemas distribuidos, si se garantizan estampas de tiempo únicas. Esto se hace ya sea por emitirlas desde un solo sitio o especificando que tendrán dos partes, la primera de las cuales es la estampa de tiempo usual generada por el reloj local o contador lógico, y la segunda parte es el identificador del nodo. Como el reloj o contador avanza en cada nodo para cada transacción, esto garantiza que cada transacción tendrá una estampa de tiempo única, aunque dos estampas de tiempo para una transacción pueden tener la misma primera parte. Sería deseable tener algún tipo de reloj global de modo que fuera posible determinar cuál de las dos transacciones que entraron a diferentes sitios fue en realidad la primera, pero es virtualmente imposible sincronizar por completo todos los relojes locales involucrados. En vez de ello, hay un esquema usual para impedir que las estampas de tiempo locales diverjan demasiado. Por ejemplo, puede haber una regla para que si un nodo recibe una estampa de tiempo con la primera parte, t, mayor que su lectura actual de reloj, lo avance en forma automática a t + 1. Esto impide que los nodos que tengan transacciones frecuentes se adelanten demasiado de los nodos menos activos. Como en el caso centralizado, el protocolo incluye procedimientos de resolución para seguirlos en caso de transacciones conflictivas. El protocolo básico de estampas de tiempo, la regla de escritura de Thomas, y las estampas de tiempo de versión múltiple se aplican directamente al ambiente distribuido con el empleo de estampas de tiempo únicas.

12.6.2 Recuperación

Igual que con una base de datos centralizada, un sistema de administración de bases de datos distribuidas debe garantizar que sus transacciones son atómicas y durables, aun cuando hubiera fallas en el sistema. Si la transacción aborta sus efectos deben deshacerse por

completo en todos los sitios afectados, y si se comprometen deben persistir en todos los sitios donde debiera.

12.6.2.1 Fallas y recuperación

Además de los tipos de fallas que pueden ocurrir en una base de datos centralizada, como errores de disco o caídas de sistema, las bases de datos distribuidas adolecen de fallas por pérdida de mensajes, de un sitio o línea de comunicaciones. También deben ser capaces de manejar una situación llamada **partición de la red**. Ésta ocurre cuando falla una línea o nodo que unía partes de la red, lo que da como resultado la separación de ésta en grupos de nodos, de tal forma que los que quedan dentro de un grupo sí se pueden comunicar, pero no hay comunicación entre grupos.

Es difícil decir si lo que ha fallado es uno de los nodos o uno de los vínculos. Si un sitio es incapaz de comunicarse con un nodo puede deberse a que éste se encuentre ocupado o haya fallado, que el vínculo de comunicaciones tenga una falla o que la red esté particionada debido a la falla de un nodo o vínculo diferentes. En el caso de una falla de vínculo se vuelve a enrutar y el sistema sigue como antes. Si no existe otro vínculo, se aísla el nodo y la falla del vínculo se trata como falla de un nodo. El sistema reconoce la falla de un nodo ya sea por un reporte de éste o, si esto no es posible, por el método de tiempo límite. En ese caso, si un nodo no responde un mensaje dentro de un tiempo preestablecido se supone que falló y comienzan los procedimientos de recuperación. Es esencial que el tiempo límite sea suficientemente largo para que éstos no arranquen de manera innecesaria. Los siguientes pasos son para permitir que el sistema continúe después de que falla un nodo.

1. El sistema debe marcar al nodo o vínculo que falló, para impedir que cualquier otro nodo trate de usarlo y genere un segundo procedimiento de recuperación.
2. El sistema debe abortar y deshacer cualesquiera transacciones afectadas por la falla.
3. El sistema debe revisar en forma periódica si el nodo se ha recuperado, o bien el nodo reportar cuando lo haya hecho.
4. Una vez que reinicia, el nodo fallido debe hacer una recuperación local, deshaciendo cualesquiera transacciones parciales que hubieran estado activas en el momento de la falla.
5. Después de una recuperación local, el nodo fallido debe actualizar su copia de los datos, de modo que coincida con el estado actual de la base de datos, para lo cual el sistema lleva un registro de los cambios ocurridos durante la falla del nodo.

12.6.2.2 Protocolos de compromiso

El coordinador de transacciones es responsable de determinar si una transacción se compromete o aborta. Si la transacción involucra varios sitios como participantes debe comprometerse en todos o abortar en todos. El coordinador debe usar un protocolo a fin de determinar el resultado de una transacción y manejar cualquier falla. El protocolo más sencillo es el de dos fases para el compromiso. El de tres fases es una versión mejorada para enfrentar algunos problemas adicionales.

1. **Protocolo de compromiso de dos fases.** Este protocolo relativamente sencillo consiste en una **fase de votación** y otra de **resolución**. Al final de la ejecución de una transacción, el coordinador de la transacción pregunta a los sitios participantes si pueden comprometer la transacción T. Si todos están de acuerdo en comprometerla, el coordinador la compromete; de otro modo la aborta. La fase consiste en las siguientes actividades.

- **Fase 1: Votación.** El coordinador escribe un mensaje <begin commit T> en su bitácora, y la fuerza a escribirla en el disco. El coordinador envía un mensaje <prepare T> a todos los sitios participantes. Cada sitio determina si puede comprometer su parte de T. Si puede, agrega un mensaje <ready T> en su propia bitácora y la obliga a registrarla en el disco, y devuelve al coordinador el mensaje <ready T>. Si un sitio no la puede comprometer agrega <abort T> en su bitácora y la fuerza a escribirla en disco, y devuelve al coordinador su voto por <abort T>.
- **Fase 2: Resolución.** El coordinador resuelve la suerte de la transacción de acuerdo con las respuestas que recibe.
  a. Si el coordinador recibe un mensaje <abort T> desde cualquier sitio, escribe otro <abort T> en su bitácora y fuerza a escribirla en disco. Envía un mensaje <abort T> a todos los sitios participantes. Cada sitio registra el mensaje en su propia bitácora y aborta la transacción.
  b. Si un sitio no vota dentro de un tiempo especificado (el periodo límite), el coordinador supone un voto por <abort T> desde ese sitio y procede a abortar la transacción como se dijo.
  c. Si el coordinador recibe de todos los sitios participantes el mensaje <ready T>, escribe un mensaje <commit T> en su bitácora, fuerza a escribirla y envía a todos el mensaje <commit T>. Cada sitio lo registra en su propia bitácora y responde al coordinador. Cuando ha recibido todas las confirmaciones, el coordinador escribe un mensaje <end T> en su bitácora. Si durante el tiempo límite no se ha recibido alguna confirmación, el coordinador supone que el sitio ha fallado. En caso de falla del coordinador o de un sitio participante, el protocolo determina las acciones por tomar.
     (1) *Falla de un sitio participante.* Como ya se dijo, si dentro del tiempo límite el coordinador no recibe un voto de algún sitio, supone un voto <abort T> y aborta la transacción. Si el sitio vota por aceptar el compromiso y después falla, el coordinador continúa con la ejecución del protocolo para los demás sitios. Cuando el sitio fallido se recupera, consulta su bitácora para determinar qué debe hacer con cada transacción que estaba activa en el momento de la falla.
        i. Si la bitácora tiene un mensaje <commit T>, rehace T, es decir, efectúa un redo (rehacer)(T).
        ii. Si la bitácora tiene un mensaje <abort T>, efectúa un undo (rehacer) (T).
        iii. Si la bitácora no tiene mensajes ready, abort o commit, entonces debe haber fallado antes de responder con un voto, por lo que sabe que el coordinador tuvo que haber abortado la transacción. Por tanto, hace realizar undo (deshacer)(T).
        iv. Si el registro tiene un mensaje <ready T>, el sitio trata de determinar el destino de la transacción. Consulta al coordinador, y si éste responde que la transacción se comprometió ejecuta redo (rehacer) (T). Si el coordinador responde que la transacción fue abortada, el sitio efectúa undo (deshacer) (T). Si el coordinador no responde, el sitio supone que ha fallado y trata de determinar el destino de la transacción con la consulta de otros sitios participantes. Si no puede determinar el destino, el sitio es **bloqueado** y debe esperar a que el coordinador se recupere.

(2) *Falla del coordinador.* Si el coordinador falla durante el protocolo commit, los sitios participantes intentan determinar el destino de la transacción. Existen varias posibilidades:

i. Si uno de los sitios tiene <abort T> en su bitácora, se aborta T.

ii. Si uno de los sitios tiene <commit T> en su registro, T se compromete.

iii. Si existe un sitio sin <ready T> en su bitácora, ese sitio no puede haber votado que se comprometiera T, por lo que el coordinador no puede haber decidido su compromiso. Entonces, T puede ser abortada.

iv. En todos los demás casos (todos los sitios activos tienen <ready T> pero ningún sitio tiene <abort T> o <commit T>) es imposible decir si el coordinador ha tomado una decisión acerca del destino de T, por lo que ésta es **bloqueada** hasta que el coordinador se recupere.

2. **Protocolo de compromiso de tres fases.** Como ya se dijo, en ciertas condiciones el protocolo de compromiso de dos fases puede originar un bloqueo. El protocolo de compromiso de tres fases no bloquea, a menos que todos los sitios fallen. Las condiciones para que el protocolo funcione son las siguientes:

- Al menos un sitio no ha fallado
- No hay partición de la red
- Para cierto número predeterminado k, no pueden fallar más de k sitios simultáneamente.

El protocolo agrega una tercera fase después de la de votación y antes de la de resolución. El coordinador primero notifica al menos a k sitios que trata de dar el compromiso, aun antes de que fuerce a escribir su propio registro de bitácora de compromiso. Si el coordinador falla antes de emitir el compromiso global, los sitios restantes eligen un nuevo coordinador, el cual comprueba con ellos para determinar el estatus de T. Como no pueden fallar más de k sitios y son k los que tienen el mensaje de que "trata de dar el compromiso", esos sitios garantizan que T se comprometa. Por tanto, no ocurre ningún bloqueo a menos que fallen k sitios. Sin embargo, la partición de la red hace que parezca que han fallado más de k sitios, y a menos que se administre con cuidado el protocolo, es posible que la transacción parezca como comprometida en una partición y abortada en otro.

12.7 Procesamiento distribuido de consultas

En el capítulo 11 se estudiaron técnicas para transformar consultas en SQL en representaciones de bajo nivel tales como álgebra relacional y métodos para transformar expresiones de álgebra relacional en otras equivalentes pero más eficientes. También se presentaron varios métodos para estimar el costo del procesamiento de consultas para varias operaciones de álgebra relacional. El factor dominante en las estimaciones de costo fue el número de accesos de disco requeridos. Un ambiente distribuido necesita que también se tome en cuenta el costo de transmitir los datos por la red. Si ésta es relativamente lenta, este factor será el dominante en el costo. Si los datos están repetidos, se tiene que considerar el aspecto de elegir el mejor de varios sitios posibles para ejecutar una consulta o parte de ella. La partición de los datos complica las consultas que requieren el empleo de más de una de las particiones. Por tanto, en un sistema de bases de datos distribuidas se deben considerar procedimientos adicionales de optimización de la consulta.

Como ya se dijo, las consultas de la base de datos se clasifican en **locales**, **remotas** y **compuestas** (también llamadas **globales**). Una solicitud local es aquella que puede satisfacerse en el nodo en que entró. Una solicitud remota puede ser satisfecha en un solo nodo remoto.

Una solicitud compuesta es la que requiere acceder a más de un nodo. Al satisfacer una solicitud compuesta, el DDBMS es responsable de descomponer la consulta en varias subconsultas, y cada una es dirigida a un nodo específico para su procesamiento. Luego, el DDBMS debe recabar y coordinar las respuestas para obtener un resultado conjunto. Idealmente, el usuario no debe conocer la distribución de los datos y tampoco necesitar especificar dónde se localizan éstos.

12.7.1 Etapas del procesamiento distribuido de consultas

En un DDBMS, para responder una consulta deben efectuarse algunos o todos los pasos siguientes:

1. *Aceptar la solicitud del usuario.* La interfaz de usuario es responsabilidad del DDBMS, y no del DBMS local en cada nodo. Como el DBMS no está al tanto de la distribución, es incapaz de manejar consultas sobre datos que no estén almacenados localmente.
2. *Comprobar la validez de la solicitud.* Esto requiere comprobar que se utiliza la sintaxis correcta en el lenguaje de la solicitud, y que los datos a que se hace referencia existen en realidad. Ésta es una comprobación de validez de nivel externo que requiere que el DDBMS tenga acceso al subesquema del usuario.
3. *Comprobar la autorización.* Información del control de acceso debe estar disponible para el DDBMS, de modo que pueda revisar si el usuario está autorizado para efectuar las operaciones solicitadas sobre los ítems de datos especificados. Esta revisión debe ser efectuada tanto por el nodo del usuario como en cada nodo en que se procesa la solicitud. Ésta es una comprobación de nivel lógico, y requiere que el esquema se encuentre disponible para el DDBMS.
4. *Mapeo externo a nivel lógico.* El DDBMS mapea los nombres de los datos definidos por el usuario y ve hacia los objetos del nivel lógico correspondiente.
5. *Determinar una estrategia de procesamiento de la solicitud.* El DDBMS debe consultar el diccionario de datos global para determinar la ubicación de aquellos que se requieren. Si la solicitud es local, la envía al DBMS local para su procesamiento. Si la solicitud es remota pasa a la etapa siguiente. Si la solicitud es compuesta, debe descomponerla en subconsultas y dirigir cada una hacia un nodo. Puede haber varias formas de descomponer una consulta, por lo que el DDBMS normalmente usa alguna forma de optimización con objeto de determinar cuál descomposición utilizar. La optimización en un sistema centralizado involucra varias técnicas. Por ejemplo, si hay condiciones múltiples, primero se hacen pruebas sobre la condición que elimine al mayor número de registros, o aquella que requiera las menores operaciones de entrada y salida. Para acceder a los registros se eligen las rutas más cortas. En un sistema distribuido la optimización también incluye el uso del sistema de comunicaciones. Un objetivo de la optimización debe ser minimizar el tiempo de respuesta, lo que impondría la descomposición de la solicitud de tal manera que pueda efectuarse procesamiento paralelo en varios nodos, en especial si el sistema de comunicaciones es rápido y la carga de procesamiento ligera. En ocasiones son varios los nodos que realizan el mismo proceso, y los resultados se obtienen de aquel que termina primero. Si el sistema se utiliza mucho, son más apropiados otros objetivos. Éstos incluyen la minimización del tiempo total de procesamiento, que consiste en la suma de los tiempos de procesamiento (posiblemente simultáneo) de los nodos más el tiempo usado para ejecutar el software de comunicaciones y consolidar los resultados. Al seleccionar un algoritmo la verdadera optimización requiere que el DDBMS encuentre todas las formas de procesar la solicitud y luego "costee" cada una, con el uso de

fórmulas matemáticas que incluyen costos de procesamiento, comunicaciones y almacenamiento. Debido a la dificultad de encontrar todos los métodos posibles de procesamiento es frecuente que se emplee el método "subir la colina", que consiste en encontrar una solución inicial y costearla, luego se evalúa una alternativa un poco diferente y se escoge la mejor de las dos, que se convierte en la base de una comparación con otra alternativa, y el proceso continúa hasta que no se encuentra ningún otro método, o con cierto número de iteraciones no se produce una mejora sustancial.

6. *Si el sistema es heterogéneo, se traduce cada consulta al DML del nodo de los datos.* En las solicitudes remotas o compuestas, el nodo en que se localizan los datos tal vez utilice un DBMS diferente de aquel del nodo del usuario. Por tanto, el DDBMS debe brindar traducción debido a las diferencias en el hardware y software. En el peor de los casos están involucrados diferentes modelos y estructuras de datos, así como distintos códigos y tamaños de palabra.
7. *Encriptar la solicitud.* La seguridad mejora si cada solicitud y respuesta se encripta antes de que pasen al sistema de comunicaciones. El nodo que las recibe debe descodificar el mensaje antes de procesarlo.
8. *En función del sistema, el componente local de comunicaciones de los datos determina el enrutamiento.* El nodo que solicita debe proveer identificadores lógicos para cada nodo que vaya a recibir una subsolicitud. La función de traducir un identificador lógico a uno físico y escoger una ruta requiere una descripción de la red, que está disponible para el componente DC. El enrutamiento podría ser directo o indirecto y requiere que el mensaje pase a través de nodos intermedios. Si es indirecto, el enrutamiento lo determina por adelantado el nodo de origen, o es determinado en forma dinámica por los nodos intermedios.
9. *El componente DC transmite el mensaje a través del sistema de comunicaciones.* Como las comunicaciones entre las bases de datos son de proceso a proceso, el mensaje debe pasar a través de todas las capas de comunicación de los datos en el nodo fuente, transmitirse al destino y regresar por todas las capas de comunicación en el nodo destino.
10. *Descodificar la solicitud.* Si el mensaje se encriptó en la fuente, ahora debe descodificarse en el nodo de destino antes de que comience el procesamiento.
11. *Efectuar la sincronización de la actualización, si fuera necesario.* Si la solicitud es una actualización, el DDBMS en el nodo receptor debe efectuar procedimientos de sincronización, como se dijo en la sección 12.6.
12. *El DBMS hace su procesamiento.* El DBMS local en el nodo de datos efectúa la combinación lógica y física, determina la estrategia de procesamiento local y recupera los datos que se requieren de aquél.
13. *Si el sistema es heterogéneo, se traducen los datos.* El nodo que solicita, nodo de datos o alguno intermedio, efectúa cualquier traducción que sea necesaria.
14. *Enviar los resultados al nodo de destino.* Normalmente, los resultados de la consulta son devueltos al nodo que solicita, pero también es posible especificar como destino a otros nodos. Los resultados tal vez se encripten antes de enviarlos.
15. *Consolidar los resultados, editar y dar formato a la respuesta para el usuario.* El nodo que solicita por lo general coordina todo el proceso de descomposición de la consulta, y normalmente es el sitio donde se efectúa la consolidación final de los resultados parciales. Además, el DDBMS en ese nodo consulta el modelo externo del usuario para presentar los datos en la forma que espera el usuario. Sin embargo, otros nodos podrían realizar algunas de estas funciones.

16. *Devolver los resultados al usuario.* Como la interfaz de usuario la proporciona el DDBMS, es responsable de desplegar los resultados.

12.7.2 Estimación de los costos de las comunicaciones de datos

Suponga que tiene el esquema estudiado en la sección 12.4, con el número de bytes para cada atributo indicado como sigue:

```
Student(stuId(5), lastName(15), firstName(15), major(12), credits(4),
homecampus(10))
Faculty(facId(5), name(15), department(15), rank(20), teachcampus(10))
Class(classNumber(10), campusoffered(10), facId(5), schedule(10), room(5))
Enroll(stuId(5), classNumber(10), grade(3))
```

Ejemplo 1

Se desea encontrar información sobre el horario de todas las especialidades CSC, con el uso de la distribución de datos que se indica en la figura 12.7. La consulta es: encontrar `stuId`, `lastName`, `firstName`, `classNumber`, `campusoffered` (campus ofrecido), `schedule` (horario) y `room` (salón) de todas las especialidades CSC, sin importar el campus. En SQL, se escribe:

```
SELECT stuId, lastName, firstName, classNumber, campusoffered, schedule, room
FROM Student, Enroll, Class
WHERE major='CSC' AND Student.stuId = Enroll.stuId AND
Enroll.classNumber=Class.classNumber;
```

Estrategia 1.

1. En cada sitio, seleccionar las especialidades CSC ($\sigma_{major='CSC'}$(Student)).
2. En cada sitio, combinar aquellas tablas con `Enroll` sobre `stuId`.

   ($\sigma_{major='CSC'}$(Student))|×|Enroll
3. Proyectar los resultados sobre `classNumber`, `stuId`, `firstName`, `lastName`, en cada sitio.

   $\Pi_{classNumber, stuId, firstName, lastName}$(($\sigma_{major='CSC'}$(Student))|×|Enroll)
4. Traer los resultados de 4 sitios remotos al sitio Main.
5. En el sitio Main, combinar con `Class`.

   ($\Pi_{classNumber, stuId, firstName, lastName}$(($\sigma_{major='CSC'}$(Student))|×| Enroll))|×|Class
6. En el sitio Main, proyectar sobre `stuId`, `lastName`, `firstName`, `classNumber`, `campusoffered`, `schedule`, `room`.

En el paso 4 se necesita determinar el costo de enviar los resultados desde cada uno de los sitios remotos. Suponga que hay 25 campos principales disponibles en los cinco campus, y que cada campus tiene 2 000 estudiantes. Debe haber 2 000/25, u 80 especialidades CSC en cada campus. Cuando se combinan éstas con `Enroll`, si se supone que los estudiantes cursan en promedio cinco materias cada uno, se obtienen 5*80 o 400 registros. Se proyecta sobre los atributos deseados, que totalizan 45 bytes, para obtener un total de 45*400 o18 000 bytes, de cada uno de los 4 sitios, o **72 000** bytes. El resto de la consulta se ejecuta en el sitio Main, como se indica en los pasos 5 y 6. El costo está determinado por los métodos analizados en el capítulo 11. Si la consulta entra en el sitio Main, no hay costos de envío adicionales. Sin embargo, si entra en otro sitio se tendrían que agregar los costos de enviar los resultados de regreso a ese sitio.

Estrategia 2.

1. En cada sitio, seleccionar las especialidades CSC y proyectar los resultados sobre `stuId`, `lastName`, `firstName`. Las operaciones de álgebra relacional son:

   $\Pi_{\texttt{stuId,lastName,firstName}}\ (\sigma_{\texttt{major='CSC'}}(\texttt{Student}))$

2. Enviar las cuatro tablas intermedias de Norte (North), Sur (South), Este (East) y Oeste (West) al sitio Principal (Main).
3. Combinar las cinco tablas intermedias con `Enroll` en Main.

   $(\Pi_{\texttt{stuId,lastName,firstName}}\ (\sigma_{\texttt{major='CSC'}}(\texttt{Student})))|\times|\texttt{Enroll}$

4. Combinar la tabla resultado con `Class` en el sitio Main.

   $((\Pi_{\texttt{classNumber,stuId,firstName, lastName}}(\sigma_{\texttt{major='CSC'}}(\texttt{Student})))|\times|\texttt{Enroll})\ |\times|\texttt{Class}$

5. Todavía en el sitio Main, proyectar los resultados sobre `stuId`, `lastName`, `firstName`, `classNumber`, `campusoffered`, `Schedule`, `room`.

Los únicos costos de envío son los de enviar cada una de las cuatro tablas intermedias (de North, South, East y West) a Main. El resultado de SELECT tiene 80 registros, cada uno de 61 bytes. Cuando se efectúa la proyección, el tamaño es de 35 bytes, por lo que se necesita enviar 80*35, 2 800 bytes, desde cada uno de los cuatro sitios, para un total de **11 200** bytes enviados. El resto de la consulta se ejecuta en el sitio Main, según se indica en los pasos 3, 4 y 5.

Al comparar las dos estrategias, se observa que los costos de envío de la primera son alrededor de seis veces los de la segunda.

Ejemplo 2

Para cada profesor de todos los campus, encuentre el nombre (`name`) de la facultad y `classNumber`, `campus` y `schedule` de todas las materias que imparta el profesor. Se supone que hay 2 500 tuplas `Class` y 800 tuplas `Faculty`. La consulta entra en el campus West. En SQL es:

```
SELECT name, classNumber, campusoffered, schedule
FROM Faculty, Class
WHERE Faculty.facId = Class.facId;
```

Estrategia 1. Enviar todo el archivo `Class` al campus West. También enviar las subtablas `Faculty` desde los demás campus al campus West. Terminar toda la consulta en el campus West. Los costos de comunicaciones son:

Archivo `Class`: 2 500 tuplas, cada una con 40 bytes, para un total de 100 000 bytes enviados.

Cuatro subtablas `Faculty`: 640 tuplas en total, cada una con 65 bytes, para un total de 41 600 bytes enviados.

El gran total es de **141 600** bytes enviados.

Estrategia 2. Enviar cada subtabla `Faculty` al sitio Main, y ahí combinar con `Class`. Proyectar sobre `name`, `classNumber`, `campusoffered`, `schedule` y enviar los resultados al campus West.

Subtablas `Faculty`: 640 tuplas en total, cada una con 65 bytes, para un total de 41 600 bytes enviados.

Subtabla `Results`: 2 500 tuplas, cada una de 45 bytes, para un total de 112 500 bytes enviados.

El gran total es de **154 100** bytes enviados.

Para este ejemplo, la primera estrategia es más eficiente debido a que no hay necesidad de enviar el archivo de resultados. Observe que si la consulta entrara en el campus Main, la segunda estrategia habría sido más eficiente.

12.7.3 La operación semicombinación (semijoin)

Para muchas bases de datos distribuidas hay una operación muy eficaz en cuanto a costo, se llama **semicombinación** (semijoin) y se describió en el capítulo 4. Si A y B son tablas, entonces la semicombinación izquierda `A |× B` se encuentra con la combinación natural de A y B para luego proyectar el resultado sobre los atributos de A. El resultado serán aquellas tuplas de A que participan en la combinación. La semicombinación derecha se define en forma similar. Como ejemplo del uso de semicombinación en un ambiente distribuido se usará el esquema siguiente:

`Department(deptCode(5), deptName(20), chairpersonId(5), telephone(10), office(5)`

almacenados en el sitio A. Suponga que hay 100 registros de 45 bytes cada uno.

`Faculty(facId(5), deptCode(5), firstName(15), lastName(15), birthDate(12), rank(10), socialSecurityNumber(9))`

almacenados en el sitio B. Suponga que hay 1 000 registros de 71 bytes cada uno.

Considere la consulta "Encontrar el nombre de cada departamento y el nombre del director de ese departamento", que entra en el sitio A. En SQL se escribe:

```
SELECT firstName, lastName, deptName
FROM Faculty, Department
WHERE Department.chairpersonId = Faculty.facId;
```

Estrategia 1

Enviar todo el archivo `Faculty` al sitio A y hacer la combinación ahí. El costo de envío es 1000*71 o **71 000** bytes. Después, proyectar los resultados sobre `firstName`, `lastName`, `deptName`, y presentarlos al usuario en el sitio A, sin más envío.

Estrategia 2

Enviar todo el archivo `Department` al sitio B y ahí hacer la combinación. El costo de envío es 100*45 o 4 500 bytes. Después enviar los resultados al sito A. El costo de envío es 50*1 000 o 50 000 bytes, para un total de **54 500** bytes.

Estrategia 3

Proyectar `Department` sobre `chairpersonId` y enviar únicamente `chairpersonId` al sitio B. Como Id es de 5 bytes, se envían 5*100, o 500 bytes, hasta este momento.

En el sitio B, combinar con `Faculty` sobre `facId`. Proyectar los resultados y transferir sólo `firstName`, `lastName` y `deptCode` de regreso al sitio A. El costo es 35*100 o 3 500 bytes.

En el sitio A, combinar el resultado con `Department` sobre `DeptCode`, y proyectar el resultado sobre `firstName`, `lastName` y `departmentName`. No están involucrados más costos de envío.

El costo total de envío es **4 000** bytes.

Como puede verse, la proyección inicial identificó sólo las Id de los directores, lo que redujo sustancialmente la cantidad de datos por enviar. La semicombinación se hizo en el sitio B, donde se realizó una combinación para luego proyectar los resultados sobre los atributos de `Faculty (firstName, lastName` y `deptCode`) que estaban en la combinación.

La semicombinación se utiliza con frecuencia de este modo en los sistemas distribuidos, especialmente consultas en las que pocas tuplas participan en la combinación. Cuando las tablas residen en diferentes sitios, el optimizador considera una semicombinación en lugar de una combinación.

12.8 Resumen del capítulo

Un **sistema de bases de datos distribuidas** tiene sitios múltiples conectados por un sistema de comunicaciones, de modo que los datos en cualquier sitio se hallan disponibles para los usuarios en otros sitios. El sistema consiste en sitios geográficamente separados pero unidos por telecomunicaciones, o los sitios tal vez estén cerca unos de otros y conectados por una **red de área local**. Las ventajas de la distribución incluyen la **autonomía local**, **más confiabilidad**, **mejor disponibilidad de los datos**, **desempeño incrementado**, **menor tiempo de respuesta** y **costos más bajos de comunicaciones**.

El diseñador de un sistema de bases de datos distribuidas debe considerar el **tipo de sistema de comunicaciones**, **modelos de datos soportados**, **tipos de aplicaciones** y **alternativas de colocación de los datos**. La selección del diseño final incluye el **procesamiento distribuido con el uso de una base de datos centralizada**, **sistemas cliente-servidor**, **bases de datos paralelas**, o **bases de datos distribuidas**. Una base de datos distribuida puede ser **homogénea**, donde todos los nodos utilizan el mismo hardware y software, o **heterogénea**, donde los nodos tienen distinto hardware o software. Los sistemas heterogéneos requieren la traducción de códigos y tamaños de palabra debido a las diferencias en el hardware, o de los modelos y estructuras de los datos debido a las diferencias en el software.

Un sistema de base de datos distribuidas tiene los siguientes componentes de software: **componente de comunicaciones de los datos (DC)**, **sistema de administración de la base de datos local(DBMS)**, **diccionario global de los datos (GDD)** y **componente del sistema de administración de la base de datos distribuida (DDBMS)**. Las responsabilidades del sistema de administración de la base de datos distribuida incluyen proveer la interfaz de usuario, localización de los datos, consultas de procesamiento, proveer el control de concurrencia y procedimientos de recuperación con alcance en toda la red, y hacer la traducción en los sistemas heterogéneos. Las **alternativas de colocación de los datos** son **centralizados**, **repetidos**, **particionados** e **híbridos**. Los factores por considerar para tomar la decisión de colocación de la base de datos son referencias locales, confiabilidad de los datos, disponibilidad de los datos, capacidades y cosos de almacenamiento, distribución de la carga de procesamiento y costos de comunicación. Los otros componentes de un sistema de base de datos distribuidas se colocan con el uso de cualquiera de las cuatro alternativas.

Las formas de transparencia deseables en una base de datos distribuida incluyen la **transparencia en la distribución de los datos** (que incluye la **transparencia de la fragmentación**, **transparencia de la ubicación** y **transparencia de la réplica**), **transparencia de la heterogeneidad del DBMS, transparencia en la transacción** (que incluye **transparencia de la concurrencia** y **transparencia de la recuperación**) y **transparencia del desempeño**.

Una de las tareas más difíciles de un sistema de administración de base de datos distribuida es la **administración de la transacción**. Cada sitio que inicia transacciones también tiene un **coordinador de transacción**, cuya función es administrar todas las transacciones, sean locales, remotas o globales, que se originen en ese sitio. Los problemas de **control de concurrencia** que pueden surgir incluyen el **problema de la actualización perdida**, el de la **actualización no comprometida**, el del **análisis inconsistente**, el de **lectura no repetible** y el de los **datos fantasma**, así como el problema de la **inconsistencia de la copia múltiple**, que es exclusivo de las bases de datos distribuidas. Igual que para el caso centralizado, las soluciones para el caso distribuido incluyen técnicas tales como el **bloqueo (candado)** y las **estampas de tiempo**. El protocolo de bloqueo de dos fases tiene variaciones en el ambiente distribuido que incluye el **administrador del bloqueo en un solo sitio**, **administradores de bloqueo distribuido**, **copia primaria** o **bloqueo de mayoría**. La regla **lee uno y escribe todos** se usa con frecuencia para datos repetidos. El candado mortal distribuido se detecta con el uso de un **grafo de espera global**. Para garantizar la seriabilidad también se emplean las estampas de tiempo, que constan de dos partes, la estampa de tiempo normal y un identificador de nodo. Para impedir que las estampas de tiempo de diferentes sitios diverjan demasiado, cada sitio adelanta en forma automática sus estampas de tiempo siempre y cuando reciba una después de su tiempo actual. Los protocolos de recuperación incluyen el **compromiso de dos fases** y el **compromiso de tres fases**.

Otra tarea importante del sistema de administración de bases de datos distribuidas es el **procesamiento distribuido de la consulta**, que incluye varias etapas, la más difícil de las cuales es **determinar una estrategia de procesamiento de la solicitud** y supervisar su ejecución. Las técnicas estándares de optimización de la consulta deben extenderse para que tomen en cuenta el costo de transferir los datos entre los sitios. En ocasiones se utiliza la operación semicombinación cuando se requiere la combinación de datos en diferentes sitios, lo que da como resultado ahorros sustanciales cuando se requiere hacer una combinación.

Ejercicios

12.1 Con el uso del esquema de distribución de los datos mostrado en la figura 12.7 para el ejemplo de University distribuido, describa al menos dos estrategias para cada una de las consultas siguientes. Estime en cada caso el costo de transferencia de los datos.

a. Encuentre `classNumber` y los nombres de todos los estudiantes inscritos en todas las materias con horario 'MWF9'.

b. Encuentre `classNumber`, `facId` del profesor y campus de todas las materias que toma Rosemary Hughes, estudiante del campus North (Norte).

12.2 Considere la consulta "Encontrar `classNumber`, `schedule` y `lastName` de todos los estudiantes inscritos para todas las materias que imparte el profesor Smith del campus North (Norte)". La consulta entra en el campus North.

a. Describa una estrategia que no use la semcombinación y estime el costo de transferencia de datos de dicha estrategia.

b. Describa una estrategia de semcombinación para la misma consulta y estime el costo de transferir los datos.

12.3 Para el esquema distribuido descrito en la sección 12.7.3 de `Department` y `Faculty`:

a. Considere la consulta "Para cada profesor, imprima nombre (`first name`), apellido (last name) y nombre del departamento (`department name`)".

i. Diga dos estrategias diferentes para procesar esta consulta sin usar una semicombinación, y el costo de cada una.

ii. Dé una estrategia que utilice semicombinación, y su costo.

b. Considere la consulta "Para cada profesor, imprima nombre (`first name`), apellido (`last name`), nombre del departamento (`department name`) y nombre del director (`chairperson`).

i. Explique por qué es más compleja esta consulta que la anterior.

ii. Plantee una estrategia eficiente para ejecutar esta consulta y diga su costo.

12.4 Una base de datos va a almacenar información para una agencia de bienes raíces que usa un servicio de listado múltiple, el cual es un servicio de cooperación en el que distintas agencias de bienes raíces acuerdan listar y mostrar las propiedades que cada una tiene en venta, y dividir la comisión por ésta. Las personas que quieren vender sus casas contactan a una de las agencias y se registran con ella. Las propiedades ofrecidas a la venta son visitadas por un agente (el agente de lista), que recaba información. Una vez listada, la propiedad puede ser vendida por un agente de cualquier agencia (el agente de ventas). Sin embargo, cuando hay un compromiso (declaración de la intención de compra que hace una persona) y se hace un depósito por la propiedad, ningún agente puede mostrar la propiedad a otro comprador potencial. Si dentro del plazo de un mes no hay un contrato o si el comprador se retracta de su oferta, la propiedad queda disponible de nuevo. Si la venta avanza y todas las partes firman un contrato, el estatus de compromiso y el de lista cambian al de vendida, y se registra la fecha del contrato en la tabla de compromiso. Al final de cada mes, la información de lista y compromiso de todas las casas vendidas ese mes (i.e., para las que se firmó contrato) se elimina de sus tablas correspondientes y se coloca en la tabla `Sold` (Vendida). Los compradores en potencia se registran sólo con un agente. Cuando la propiedad se vende, la comisión se divide entre el agente de lista y el agente de ventas, que pueden ser la misma persona. Suponga que se usa el siguiente esquema:

```
AgencySalesOffice(name, address, numberOfAgents, generalTelephone)
Agent(agentName, salesOfficeName, agentTelephone)
ActiveListing(address, type, style, size, askingPrice, dateListed,
ownerName, ownerTelephone, lotSize, houseSize, numBedrooms, numBaths,
numFloors, features, status, listingAgent)
Sold(address, type, style, size, askingPrice, dateListed, ownerName,
ownerTelephone, lotSize, houseSize, numBedrooms, numBaths, numFloors,
features, status, listingAgent, sellingAgent, buyerName, sellingPrice,
dateOfContract)
ProspectiveBuyer(name, address, telephone, typeWanted, styleWanted,
sizeWanted, bedroomsWanted, highestPrice, dateRegistered,
specialRequests, agentName)
Binder(buyerName, listingAddress, sellingPrice, dateBid, dateAccepted,
amountDeposit, status, dateCancelled, dateOfContract)
```

Por sencillez, suponga que todos los nombres son únicos. Las características de los atributos en `Listing` es una descripción de la propiedad que contiene a lo sumo 100 caracteres. El atributo `specialRequests` en las listas `ProspectiveBuyer` lista cualesquiera necesidades especiales del cliente.

a. Escoja un plan de distribución para estos datos y justifique su elección, con el empleo de los criterios listados en la sección 12.5.

b. Escriba un esquema de fragmentación como sigue: decida cómo deben fragmentarse los datos para que se ajusten al plan de distribución. Use álgebra relacional o

comandos SQL para crear los fragmentos, y para cada uno identifique la(s) ubicación(es) donde se almacenará.

c. Explique cómo se respondería la consulta siguiente: encontrar los nombres y direcciones de todos los compradores potenciales y sus agentes para hacer un nuevo listado. Se trata de una residencia (p. ej., `type` = 'residence') estilo colonial con cuatro recámaras y precio de venta de 800 000 dólares.

d. Explique cómo se respondería la consulta siguiente: encontrar todas las casas adecuadas para mostrar a un comprador en potencia interesado en una residencia de dos plantas con tres recámaras y dos baños que cuesten menos de 500 000 dólares. Para cada casa, mostrar toda la información de la lista más el nombre y teléfono del agente de ventas.

e. Explique cómo se contestaría la siguiente consulta: para todas las casas vendidas el último mes en las que Jane Hayward fue la agente de lista, encontrar el nombre del agente de ventas.

12.5. Suponga que una cadena de tiendas de cómputo que vende computadoras, componentes y partes utiliza el siguiente esquema global para una base de datos relacional que mantiene información sobre los artículos en cada tienda:

```
Item(itemNo, itemName, supplier, unitCost)
Store (storeName, address, manager, telephone)
Stock (itemNo, store, qtyOnHand, qtyOnOrder, reorderPoint)
```

La compañía tiene 20 tiendas que almacenan 15 000 artículos distintos, cada uno de los cuales proviene de un solo proveedor, pero hay varios proveedores para artículos diferentes. La base de datos da seguimiento a cuáles artículos están en cuál tienda, cuáles han sido reordenados en cada tienda y el punto de reorden, que es el número mínimo de artículos que cada tienda desea tener en inventario. Cuando la cantidad disponible es inferior al punto de reorden, se hace un nuevo pedido de artículos, a menos que el artículo ya haya sido reordenado. Cada tienda sirve al menos a los clientes de su propia área geográfica, pero si un artículo no está en su inventario el gerente lo obtiene de otra tienda. Actualmente, toda la información está almacenada en una sola base de datos central. Se desea diseñar un sistema distribuido para hacer más eficientes las operaciones.

a. Escoja un plan de distribución de datos y justifique la elección.

b. Escriba un esquema de fragmentación con el empleo de álgebra relacional o SQL para crear los fragmentos. Para cada fragmento identifique la(s) ubicación(es) donde se almacena.

c. Explique cómo se respondería la consulta siguiente: encontrar la cantidad total del artículo número 1001 que esté en todas las tiendas.

d. Si el número de bytes en cada registro de datos es 10, encontrar el costo de las comunicaciones involucradas en la estrategia que empleó para hacer la consulta en el inciso c).

e. Suponga que un cliente de la tienda A pidió una gran cantidad de un artículo que no está en el almacén de dicha tienda. Explique cómo permitiría la base de datos que el gerente de ella localizara otras tiendas que tuvieran suficientes artículos para cumplir la orden, si se supone que ninguna tienda por sí sola tiene las cantidades necesarias para satisfacer toda la orden. Escriba en álgebra relacional o SQL la consulta necesaria.

f. Suponga que cada ítem de datos ocupa 10 bytes, estime el costo de las comunicaciones involucrado en la estrategia usada para hacer la consulta del inciso e), haga las suposiciones necesarias.

PROYECTO DE MUESTRA: PLANEACIÓN DE LA DISTRIBUCIÓN DE LA BASE DE DATOS RELACIONAL PARA LA GALERÍA DE ARTE

Suponga que la Galería de Arte se expandió a tres localidades: Midtown (Periferia), que es la galería original, más Uptown (Suburbios) y Downtown (Centro). Se desea distribuir la base de datos entre las tres localidades. Como esquema global se usará el conjunto de relaciones desarrolladas en el capítulo 6, que es el siguiente:

```
(1) Artist (artistId, firstName, lastName, interviewDate, interviewerName,
areaCode, telephoneNumber, street, zip, salesLastYear, salesYearToDate,
socialSecurityNumber, usualMedium, usualStyle, usualType)
(2) Zips (zip, city, state)
(3) PotentialCustomer (potentialCustomerId, firstName, lastName, areaCode,
telephoneNumber, street, zip, dateFilledIn, preferredArtistId,
preferredMedium, preferredStyle, preferredType)
(4) Artwork (artworkId, artistId, workTitle, askingPrice, dateListed,
dateReturned, dateShown, status, workMedium, workSize, workStyle, workType,
workYearCompleted, collectorSocialSecurityNumber)
(5) ShownIn (artworkId,showTitle)
(6) Collector (socialSecurityNumber, firstName, lastName, street, zip,
interviewDate, interviewerName, areaCode, telephonenumber, salesLastYear,
salesYearToDate, collectionArtistId, collectionMedium, collectionStyle,
collectionType, SalesLastYear, SalesYearToDate)
(7) Show (showTitle, showFeaturedArtistId, showClosingDate, showTheme,
showOpeningDate)
(8) Buyer (buyerId, firstName, lastName, street, zip, areaCode,
telephoneNumber, purchasesLastYear, purchasesYearToDate)
(9) Sale (InvoiceNumber, artworkId, amountRemittedToOwner, saleDate,
salePrice, saleTax, buyerId, salespersonSocialSecurityNumber)
(10) Salesperson (socialSecurityNumber, firstName, lastName, street, zip)
```

- Etapa 12.1. Escriba un conjunto de ubicaciones de usuario final y las aplicaciones que se ejecutan en cada una. Las tres ubicaciones son Midtown (sitio principal), Uptown y Downtown. Las aplicaciones realizadas en cada sucursal por los datos propios de ella son:
  1. Mantener registros de las obras de arte
  2. Producir facturas de las ventas
  3. Mantener registros de las ventas
  4. Mantener los registros de los clientes potenciales
  5. Producir el reporte Works for Sale (Obras a la venta)
  6. Generar el reporte Sales This Week (Ventas de esta semana)
  7. Producir el reporte Buyer Sales (Ventas por comprador)
  8. Generar el reporte Preferred Customer (Cliente preferente)
  9. Generar el reporte Salesperson Performance (Desempeño de vendedores)
  10. Producir el reporte Aged Artworks (Obras de arte antiguas)
  11. Generar el Owner Payment Stub (Talón de pago al propietario)
  12. Producir el reporte Art Show Details (Detalles de la exposición de arte)

Además, se realizan las aplicaciones siguientes, sólo en Midtown:

13. Mantener un registro de artistas
14. Llevar registros de coleccionistas
15. Producir el reporte Active Artists Summary (Resumen de artistas activos)
16. Generar el reporte Individual Artist Sales (Ventas por artista individual)
17. Producir el reporte Collectors Summary (Resumen de coleccionistas)
18. Generar el reporte Individual Collector Sales (Ventas por coleccionista individual)

- Etapa 12.2. Para cada aplicación, decida cuáles tablas se requieren.
  1. Mantener registros de obras de arte: tablas `Artwork`, `Artist` y `Collector`
  2. Producir facturas de las ventas: tablas `Sale`, `Buyer`, `Salesperson`, `Artist`, `Collector` y `Zip`
  3. Mantener un registro de las ventas: `Sale`, `Buyer`, `Salesperson`, `Zip` y `Artwork`
  4. Mantener registros de los clientes potenciales: tablas `PotentialCustomer`, `Zip` y `Artist`
  5. Generar el reporte Works for Sale (Obras a la Venta): tablas `Artwork`, `Artist` y `Collector`
  6. Producir el reporte Sales This Week: tablas `Sale`, `Salesperson`, `Buyer`, `Artist`, `Artwork`, `Zip` y `Collector`
  7. Producir el Buyer Sales Report: tablas `Buyer`, `Zip`, `Sale`, `Artwork` y `Artist`
  8. Generar el reporte Preferred Customer: tablas `PotentialCustomer`, `Zip`, `Buyer`, `Artwork` y `Artist`
  9. Producir el reporte Salesperson Performance Report: tablas `Sale`, `Artwork`, `Artist`, `Salesperson` y `Zip`
  10. Producir el reporte Aged Artworks: tablas `Artwork`, `Collector` y `Artist`
  11. Generar el Owner Payment Stub: tablas `Sale`, `Collector`, `Zip` y `Artist`
  12. Producir el reporte Art Show Details: tablas `Show`, `ShownIn`, `Artwork` y `Artist`
  13. Mantener registros de los artistas: tablas `Artist` y `Zip`
  14. Conservar registros de los coleccionistas: tablas `Collector`, `Zip` y `Artist`
  15. Generar el reporte Active Artists Summary: tablas `Artist`, `Zip`, `Sale` y `Artwork`
  16. Producir el reporte Individual Artist Sales: tablas `Artist`, `Zip`, `Sale` y `Artwork`
  17. Generar el reporte Collectors Summary: tablas `Collector`, `Artist`, `Zip`, `Sale` y `Artwork`
  18. Producir el reporte Individual Collector Sales: tablas `Collector`, `Artist`, `Zip`, `Sale` y `Artwork`.
- Etapa 12.3. Con el uso de las relaciones normalizadas, realizar las operaciones de selección y proyección, a fin de crear el conjunto de fragmentos vertical, horizontal y mixto de los datos para cada aplicación.

  Artista. En todos los datos se utilizan partes de esta tabla para todas sus aplicaciones (1 a 12). Para la mayor parte de ellas sólo es necesaria `artistId`, pero para unas

cuantas se requiere el nombre. Los datos requeridos se producen con la proyección de la tabla `Artist` sobre las columnas requeridas, lo que crea un fragmento que se llamará `ArtistFragment1`.

$$ArtistFragment1=\Pi_{artistId,firstName,lastName}(Artist)$$

En cada sucursal, el Owner Payment Stub (talón de pago al propietario) también requiere el nombre del artista, su dirección y número de seguridad social, si el artista es el propietario. Para esta proyección se forma otro fragmento.

$$ArtistFragment2=\Pi_{artistId,firstName,lastName,street,zip,\ socialSecurityNumber}(Artist)$$

El sitio Midtown necesita toda la tabla para mantener los registros de artistas (13) y de coleccionistas (14), y para todos sus reportes, 15 a 18.

Zips. Como rara vez se actualiza la tabla y se necesita en todos los sitios, en cada sucursal se repetirá completa.

PotentialCustomer. Cada sucursal tendrá sus propios registros de clientes potenciales. La ID comenzará con un código que indique la sucursal.

$$PotentialCustomerDowntown=\sigma_{ID\ LIKE\ 'D\%'}(PotentialCustomer)$$
$$PotentialCustomerMidtown=\sigma_{ID\ LIKE\ 'M\%'}(PotentialCustomer)$$
$$PotentialCustomerUptown=\sigma_{ID\ LIKE\ 'U\%'}(PotentialCustomer)$$

Artwork. Cada sucursal necesita esta tabla para mantener registros de las obras de arte en esa sucursal (1) y registros de ventas (3). Los fragmentos se forman con el uso de una operación de selección, si se supone que `artworkId` contiene un código que indica la sucursal, como se hizo para `PotentialCustomer`. Los fragmentos se identifican usando:

$$ArtworkDowntown=\sigma_{ID\ LIKE\ 'D\%'}(Artwork)$$
$$ArtworkMidtown=\sigma_{ID\ LIKE\ 'M\%'}(Artwork)$$
$$ArtworkUptown=\sigma_{ID\ LIKE\ 'U\%'}(Artwork)$$

La sucursal también utiliza estos mismos fragmentos de tabla para producir sus propios reportes Works for Sale (5), Buyer Sales (7), Salesperson Performance (9), Aged Artworks (10) y Art Show Details (12).

El sitio Midtown también usa toda la tabla para los reportes 15, 16, 17 y 18.

ShownIn. Cada sucursal usa esta tabla para su reporte Art Show Details (12). Sin embargo, sólo necesita el fragmento horizontal para sus propias obras y exposiciones. Se agregará un atributo, `branch` (sucursal), a la tabla `Show`, para identificar la sucursal en que está cada exposición. Entonces, la tabla `ShownIn` se puede fragmentar horizontalmente con el uso de una subconsulta SQL para identificar la sucursal apropiada. Por ejemplo, para la sucursal del centro se crea el fragmento horizontal, `ShownInDowntown`, como

```
SELECT *
FROM ShownIn
WHERE showTitle IN (SELECT showTitle
                    FROM Show
                    WHERE branch='Downtown');
```

De manera similar, se forman `ShownInMidtown` y `ShownInUptown`.

Collector. Cada sucursal usa esta tabla para mantener registros de las obras de arte (1), producir facturas de las ventas (2) y para los reportes 5, 6, 10 y 11. Para algunos de éstos se necesita `collectorId` y su nombre. Esta tabla se fragmenta con el empleo de una proyección, como se hizo para la tabla `Artist`.

$$CollectorFragment1 = \Pi_{collectorId,\ firstName,\ lastName}(Collector)$$

El Owner Payment Stub requiere el nombre, dirección y número de seguridad social del coleccionista, si éste fuera el propietario. Para esta proyección se forma otro fragmento.

$$\texttt{CollectorFragment2} = \Pi_{\texttt{collectorId,firstName,lastName,street,zip, socialSecurityNumber}}(\texttt{Collector})$$

La sucursal Midtown usa toda la tabla para mantener los registros (14) de coleccionistas y los reportes 17 y 18.

Show. Cada sucursal utiliza esta tabla para su reporte (12) Art Show Details. Se agrega un atributo, `branch`, a la tabla `Show`, y se crea fragmentos como sigue:

$$\texttt{ShowDowntown} = \sigma_{\texttt{branch='Downtown'}}(\texttt{Show})$$
$$\texttt{ShowMidtown} = \sigma_{\texttt{branch='Midtown'}}(\texttt{Show})$$
$$\texttt{ShowUptown} = \sigma_{\texttt{branch='Uptown'}}(\texttt{Show})$$

Comprador. Cada sucursal usa esta tabla para producir facturas de ventas (2), mantener registros de ventas (3) y para los reportes 6, 7 y 8. Si se supone que `buyerId` es una cadena que contiene un código para la sucursal donde el comprador hace una compra, se usa la operación selección para formar fragmentos horizontales.

$$\texttt{BuyerDowntown} = \sigma_{\texttt{ID LIKE 'D\%'}}(\texttt{Buyer})$$
$$\texttt{BuyerMidtown} = \sigma_{\texttt{ID LIKE 'M\%'}}(\texttt{Buyer})$$
$$\texttt{BuyerUptown} = \sigma_{\texttt{ID LIKE 'U\%'}}(\texttt{Buyer})$$

Sale. Cada sucursal utiliza esta tabla para generar facturas de ventas (2), conservar sus registros de ventas (3) y para hacer los reportes 6, 7, 9 y 11. Cada sucursal emplea su propio conjunto de números de factura, cuyo dígito inicial identifica a cada una. Se crean fragmentos horizontales a fin de identificar la sucursal para cada venta, con el empleo de lo siguiente:

$$\texttt{SaleDowntown} = \sigma_{\texttt{invoiceNumber>=0 and invoiceNumer<20000}}(\texttt{Sale})$$
$$\texttt{SaleMidtown} = \sigma_{\texttt{invoiceNumber>=20000 and invoiceNumer<40000}}(\texttt{Sale})$$
$$\texttt{SaleUptown} = \sigma_{\texttt{invoiceNumber>=40000 and invoiceNumer<60000}}(\texttt{Sale})$$

Midtown también usa la tabla completa para los reportes 15, 16, 17 y 18.

Salesperson. Cada sucursal usa esta tabla para producir facturas de ventas (2), mantener registros de ventas (3) y para los reportes 6 y 9. Se agrega un atributo, `branch`, a la tabla para identificar la sucursal a que pertenece un vendedor. Con la operación de selección se forman los subconjuntos horizontales:

$$\texttt{SalespersonDowntown} = \sigma_{\texttt{branch='Downtown'}}(\texttt{Salesperson})$$
$$\texttt{SalespersonMidtown} = \sigma_{\texttt{branch='Midtown'}}(\texttt{Salesperson})$$
$$\texttt{SalespersonUptown} = \sigma_{\texttt{branch='Uptown'}}(\texttt{Salesperson})$$

- Etapa 12.4. Mapear los fragmentos a las aplicaciones y ubicaciones. Para cada fragmento que se requiera en más de una ubicación de aplicación, hay que decidir si el fragmento se repite, tomando en cuenta la frecuencia de uso y actualización.

  Artista. La tabla `Artist` se actualizará con mucha frecuencia, y `artis ID` y el nombre aparecen en muchas aplicaciones en todas las sucursales. Por tanto, se escogerá repetir el fragmento `ArtistFragment1` en todas las sucursales. La sucursal Midtown necesita toda la tabla para sus aplicaciones, por lo que ahí se almacenará toda la tabla. Se observa que $\texttt{ArtistFragment2}\ (\Pi_{\texttt{artistId, firstName, lastName, street, zip, socialSecurityNumber}}(\texttt{Artist}))$ contiene datos delicados —dirección y número de seguridad social del artista— y se necesita en las sucursales sólo para el talón de pago al propietario cuando éste es el artista. Por tanto, se elegirá guardar este fragmento sólo en la sucursal Midtown, y se permitirá que las demás sucursales accedan cuando lo necesiten. Sin embargo, como Midtown ya tiene toda la tabla `Artist`, se creará una vista que reemplace `ArtistFragment2`, que

permite que las sucursales accedan a la vista cuando se hagan pagos a los artistas, y no se creará el fragmento.

Zips. Esta tabla se necesita en todas las ubicaciones, se modifica rara vez y no contiene datos delicados, por lo que se repite en todos los sitios.

PotentialCustomer. Cada sucursal almacena datos sobre sus propios clientes potenciales, con el uso de los fragmentos `PotentialCustomerDowntown`, `PotentialCustomerMidtown` y `PotentialCustomerUptown`.

Artwork. Cada sucursal almacena registros sobre sus propias obras de arte, con el uso de los fragmentos `ArtworkDowntown`, `ArtworkMidtown` y `ArtworkUptown`. La sucursal Midtown guarda una copia de toda la tabla.

ShownIn. Cada sucursal utiliza el fragmento de esta tabla para sus propias exposiciones, es decir `ShownInDowntown`, `ShownInMidtown` y `ShownInUptown`.

Collector. Cada sucursal almacena una copia de `CollectorFragment1`, que tiene `Id` y nombre. La sucursal Midtown guarda toda la tabla y da acceso a `CollectorFragment2` como vista de esa tabla.

Show. Cada sucursal guarda los registros de sus propias exposiciones, es decir `ShowDownton`, `ShowMidtown` y `ShowUptown`.

Buyer. Cada sucursal almacena los registros de sus propios compradores, es decir `BuyerDowntown`, `BuyerMidtown` y `BuyerUptown`.

Sale. Cada sucursal tiene sus propios registros de ventas, identificados por número de factura, como `SaleDowntown`, `SaleMidtown` y `SaleUptown`. La sucursal Midtown también guarda una copia de toda la tabla.

Salesperson. Cada sucursal tiene registros de sus propios vendedores, es decir `SalespersonDowntown`, `SalespersonMidtown` y `SalespersonUptown`.

- Etapa 12.5. Construir una tabla que muestre la red geográfica, con la lista de nodos y aplicaciones y que indique los fragmentos de datos en cada nodo.

  La tabla se muestra en la figura 12.9.

- Etapa 12.6. Para cada aplicación en la red geográfica, determinar si el acceso será local, remoto o compuesto. Construir una tabla que muestre cada sitio y las aplicaciones que requieren acceso local, remoto y compuesto.

  En la figura 12.10 se presenta la tabla.

- Etapa 12.7. Para cada uno de los accesos no locales, identifique la aplicación y ubicación de los datos. Estime el número de accesos requeridos por día. Si es grande, justifique su elección de almacenamiento no local.

  Las únicas aplicaciones que requieren acceso remoto son las aplicaciones de las facturas de ventas y del talón de pago al propietario, que requieren que las sucursales accedan al sitio Midtown para determinar el nombre, dirección y número de seguridad social del propietario de la obra de arte. Se ha decidido mantener estos datos sólo en un sitio por razones de privacidad. El volumen corresponderá al número de ventas en cada sitio. Para las obras de arte originales del tipo ofrecido en The Art Gallery, el número de transacciones por día no será grande.

- Etapa 12.8. Hacer cualesquiera ajustes indicados por el análisis de las aplicaciones y tráfico, y planear una red geográfica final.

  Como la mayor parte de los accesos son locales, no es necesario ajustar la red geográfica que se muestra en la figura 12.10.

PROYECTOS ESTUDIANTILES: PLANEACIÓN PARA LA DISTRIBUCIÓN

Para el proyecto que haya elegido, suponga que el procesamiento ha de distribuirse en al menos cuatro ubicaciones. Identifique las aplicaciones que se efectuarán en cada una y luego siga las etapas mostradas en el caso de estudio para planear la distribución de su base de datos.

Aplicación	Downtown	Midtown	Uptown
1 Mantener Artwork Records (Registros de obras de arte)	ArtworkDowntown ArtistFragment1 CollectorFragment1	Artwork Artist Collector	ArtworkUptown ArtistFragment1 CollectorFragment1
2 Producir Sales Invoice (Facturas de ventas)	SaleDowntown BuyerDowntown SalespersonDowntown ArtistFragment1 CollectorFragment1 Zip	Sale BuyerMidtown SalespersonMidtown Artist Collector Zip	SaleUptown BuyerUptown SalespersonUptown ArtistFragment1 CollectorFragment1 Zip
3 Mantener Sales Records (Registros de ventas)	Sale Downtown BuyerDowntown SalespersonDowntown ArtworkDowntown Zip	Sale BuyerMidtown SalespersonMidtown Artwork Zip	SaleUptown BuyerUptown SalespersonUptown ArtworkUptown Zip
4 Mantener Potential Customer Record (Registro de clientes potenciales)	PotentialCustomerDowntown ArtistFragment1 Zip	PotentialCustomerMidtown Artist Zip	PotentialCustomerUptown ArtistFragment1 Zip
5 Works for Sale Report (Obras para reporte de ventas)	ArtworkDowntown ArtistFragment1 CollectorFragment1	Artwork Artist Collector	ArtworkUptown ArtistFragment1 CollectorFragment1
6 Sales This Week (Ventas esta semana)	SaleDowntown BuyerDowntown SalespersonDowntown ArtworkDowntown ArtistFragment1 CollectorFragment1 Zip	Sale BuyerMidtown SalespersonMidtown ArtworkMidtown Artist Collector Zip	SaleUptown BuyerUptown SalespersonUptown ArtworkUptown ArtistFragment1 CollectorFragment1 Zip
7 Buyers Sales Report (Reporte de ventas por compradores)	BuyerDowntown Zip SaleDowntown ArtworkDowntown ArtistFragment1	BuyerMidtown Zip SaleMidtown ArtworkMidtown Artist	BuyerUptown Zip SaleUptown ArtworkUpown ArtistFragment1
8 Preferred Customer Report (Reporte de clientes preferenciales)	Potential CustomerDowntown Zip BuyerDowntown ArtworkDowntown ArtistFragment1	PotentialCustomerMidtown Zip BuyerMidtown ArtworkMidtown Artist	PotentialCustomerUptown Zip BuyerUptown ArtworkUptown ArtistFragment

Aplicación	Downtown	Midtown	Uptown
9 Salesperson Performance Report (Reporte de desempeño del vendedor)	SalespersonDowntown SaleDowntown ArtworkDowntown ArtistFragment1 Zip	SalespersonMidtown SaleMidtown ArtworkMidtown Artist Zip	SalespersonUptown SaleUptown ArtworkUptown ArtistFragment1 Zip
10 Aged Artworks Report (Reporte de obras de arte antiguas)	ArtworkDowntown CollectorFragment1 ArtistFragment1	ArtworkMidtown Collector Artist	ArtworkUptown CollectorFragment1 ArtistFragment1
11 Owner Payment Stub (Talón de pago del propietario)	SaleDowntown CollectorFragment1 Zip ArtistFragment1	SaleMidtown Collector Zip Artist	SaleUptown CollectorFragment1 Zip ArtistFragment1
12 Art Show Details Report (Reporte de detalles de exposiciones)	ShowDowntown ShownInDowntown ArtworkDowntown ArtistFragment1	ShowMidtown ShownInMidtown ArtworkMidtown Artist	ShowUptown ShownInUptown ArtworkUptown ArtistFragment1
13 Maintaining Artist Records (Registros de mantenimiento del artista)		Artist, Zip	
14 Maintaining Collector Records (Registros de mantenimiento del coleccionista)		Collector, Zip, Artist	
15 Active Artists Summary Report (Reporte resumido de artistas activos)		Artist, Zip, Sale, Artwork	
16 Individual Artist Sales Report (Reporte de ventas de artistas individuales)		Artist, Zip, Sale, Artwork	
17 Collectors Summary Report (Reporte resumido de coleccionistas)		Collector, Artist, Zip, Sale, Artwork	
18 Individual Collector Sales (Ventas por coleccionistas individuales)		Collector, Artist, Zip, Sale, Artwork	

FIGURA 12.9

Red geográfica para The Art Gallery

Aplicación	Downtown	Midtown	Uptown
1 Mantener Artwork Records (Registros de obras de arte); todos los accesos locales	local	local	local
2 Producir Sales Invoice (Facturas de ventas); acceso remoto desde las sucursales para dirección, teléfono y número de seguridad social del propietario	remoto	local	remoto
3 Mantener Sales Records (Registros de ventas); todos los accesos locales	local	local	local
4 Mantener Potential Customer Record (Registro de clientes potenciales); todos los accesos locales	local	local	local
5 Works for Sale Report (Obras para reporte de ventas); todos los accesos locales	local	local	local
6 Sales This Week-all (Ventas esta semana); todos los accesos locales	local	local	local
7 Buyers Sales Report (Reporte de ventas por compradores); todos los accesos locales	local	local	local
8 Preferred Customer Report (Reporte de clientes preferenciales); todos los accesos locales	local	local	local
9 Salesperson Performance Report (Reporte de desempeño del vendedor); todos los accesos locales	local	local	local
10 Aged Artworks Report (Reporte de Obras de Arte Antiguas); todos los accesos locales	local	local	local
11 Owner Payment Stub (Talón de Pago del Propietario); acceso remoto desde las sucursales para la dirección, teléfono y número de seguridad social del propietario	remoto	local	remoto

Aplicación	Downtown	Midtown	Uptown
12 Art Show Details Report (Reporte de detalles de Exposiciones); todos los accesos locales	local	local	local
13 Maintaining Artist Records (Registros de mantenimiento del artista); acceso local	local	local	local
14 Maintaining Collector Records (Registros de mantenimiento del coleccionista); acceso local	local	local	local
15 Active Artists Summary Report (Reporte resumido de artistas activos); acceso local	local	local	local
16 Individual Artist Sales Report (Reporte de ventas de artistas individuales); acceso local	local	local	local
17 Collectors Summary Report (Reporte resumido de coleccionistas); acceso local	local	local	local
18 Individual Collector Sales (Ventas por coleccionistas individuales); acceso local	local	local	local

FIGURA 12.10

Aplicaciones con accesos local, remoto y compuesto

CAPÍTULO 13

Bases de datos e Internet

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Por qué las organizaciones dan acceso por Internet a sus bases de datos
- Terminología y conceptos fundamentales para entender la World Wide Web
- Los orígenes de Internet y la World Wide Web
- Características de http
- Fundamentos de HTML y XML
- Características de los documentos XML
- Estructura de una DTD
- Estructura de un esquema XML
- ¿Qué es lo que hace válido un documento de instancia XML?
- Características de la arquitectura multicapa
- Funciones y tecnología usadas en la capa de presentación
- Funciones y tecnología usados en la capa intermedia
- Funciones de la capa de datos
- Características y representación gráfica del modelo de datos semiestructurados
- Algunas operaciones fundamentales de consultas XML

- Algunos métodos usados para conversión de formatos XML y de bases de datos relacionales

Ejercicios

Ejercicios de laboratorio

13.1 Navegación en un sitio Web de comercio electrónico que utilice una base de datos

13.2 Creación de páginas Web que usen Access

PROYECTO DE MUESTRA: Creación de un sitio Web que use Access, para la Galería de Arte

PROYECTOS ESTUDIANTILES: Creación de un sitio Web para los proyectos estudiantiles

13.1 Introducción

El uso tan extendido de la World Wide Web brinda nuevas oportunidades a las organizaciones para que capturen y usen datos en formas innovadoras, y plantea retos significativos a los diseñadores y administradores de bases de datos. La Web puede ser vista como un recurso de información poco organizada cuyo potencial es enorme. Actualmente, muchos sitios Web almacenan recursos en archivos HTML (o en otro formato) separados, estáticos y vinculados. Si los archivos y vínculos son creados y actualizados por distintos autores, su administración se vuelve un problema. Si contienen datos tomados de una base de datos, los archivos estáticos HTML se podrían hacer inconsistentes cuando se actualice la base de datos. Por esto, muchas organizaciones escogen dar un acceso dinámico a las bases de datos directamente desde Web.

El comercio electrónico (e-comercio) está revolucionando la forma en que interactúan las empresas con sus clientes, proveedores y contratistas. Las organizaciones están desarrollando aplicaciones basadas en Web a fin de crear mercados mundiales para sus productos, distribuir información, dar a sus clientes un servicio mejor y más barato, comunicarse con sus proveedores, capacitar a los empleados, expandir el sitio de trabajo e implementar muchas otras actividades innovadoras.

El desarrollo y administración de este recurso requiere una combinación de tecnología de las comunicaciones, de recuperación de información y bases de datos. Ha habido una carrera por el desarrollo de estándares y son muchos los modelos propuestos. En el momento presente, XML ha surgido como un estándar prometedor para el almacenamiento, intercambio y recuperación de documentos que tal vez lleve a la integración de tecnologías en competencia. El estándar XML que describe el contenido de documentos facilita el desarrollo de aplicaciones en línea, inclusive las de comercio electrónico.

13.2 Conceptos fundamentales de Internet y la World Wide Web

Con objeto de entender las tecnologías y estándares usados en la World Wide Web es necesario tener algo de información sobre los antecedentes de Internet.

13.2.1 Orígenes de la Web

Los términos **Internet** y **World Wide Web** se utilizan de manera indistinta, pero en realidad tienen significados diferentes. Internet precede a Web en varias décadas, y consiste en una red grande compuesta por otras más pequeñas (internetwork, trabajo entre redes). Fue

desarrollado por **Arpanet**, red de comunicaciones creada en la década de 1960 con fondos de la U.S. Department of Defense Advanced Research Project Agency (DARPA), para comunicar instituciones del gobierno y de investigación académica. La red usaba un protocolo común, **TCP/IP** (Transmission Control Protocol/Internet Protocol), para facilitar las comunicaciones entre distintos sitios. Después, la National Science Foundation tomó la responsabilidad de administrar la red, que hoy se conoce como Internet. Aunque Internet permitía el acceso a los recursos de lugares alejados entre sí, requería conocimientos considerables por parte del usuario, quien tenía que encontrar los objetos de interés, entrar al sistema remoto que los contenía, navegar por los directorios de ese sistema para encontrar los archivos que deseaba, copiar éstos a un directorio local y luego desplegarlos correctamente.

En 1989, Tim Berners-Lee propuso un método para simplificar el acceso a los recursos que llevó al desarrollo de la World Wide Web. Su propuesta incluía:

- Un método para identificar la ubicación de los recursos, llamado **Uniform Resource Locator**, **URL** (localizador uniforme de recursos).
- Un protocolo estándar para transferir documentos por Internet, el Hypertext Transfer Protocol, **HTTP** (protocolo de transferencia de hipertexto).
- Un lenguaje llamado Hypertext Markup Language, **HTML** (lenguaje de marcado de hipertexto) que integra las instrucciones para desplegar documentos dentro de los documentos en sí.
- **Hipertexto**, técnica que se usa para incrustar vínculos en los documentos para hacer referencia a otros recursos.
- Un **navegador gráfico** que muestra contenido en una ventana y que contiene vínculos incrustados.

La propuesta de Berners-Lee hizo posible automatizar el complicado proceso de encontrar, descargar y mostrar archivos en Internet.

13.2.2 Navegadores, vínculos y los URL

Los usuarios acceden a Internet con un navegador Web como Microsoft Explorer o Netscape. Al usar el navegador para visitar un sitio pueden escribir el URL que identifica al servidor Web del sitio, el que contiene cierto número de páginas Web, cada una de las cuales tiene un documento raíz correspondiente que dice cómo ha de mostrarse la página. Las páginas pueden tener vínculos que contienen los URL de otros recursos. Al hacer clic en los vínculos se permite al usuario visitar los sitios que contienen esos recursos para que se muestre la información en ellos sin tener que escribir su URL.

Un URL es un tipo específico de **identificador uniforme de recursos** (**URI**, **Uniform Resource Identifier**), que es una cadena que identifica la ubicación de cualquier tipo de recurso en Internet, inclusive páginas Web, buzones, archivos descargables, etc. Un ejemplo de URI es *`http://www.iona.edu/about.htm`*. La primera parte identifica al protocolo, http, que va a usarse para acceder al recurso. La siguiente parte, `www.iona.edu`, identifica al servidor en que se localiza el recurso. La última parte, el nombre de la ruta, identifica al documento, `about.htm`, al que va a accederse.

13.2.3 HTTP

El protocolo de transferencia de hipertexto (HTTP) es el más utilizado en Internet. Un **protocolo de comunicaciones** es un conjunto de estándares para la estructura de mensajes entre las partes. Con el empleo de este protocolo, un navegador Web, que funciona como

cliente HTTP, primero envía una solicitud a un servidor HTTP, el cual tiene unas cuantas líneas de texto, con una línea vacía al final. Un ejemplo es el siguiente:

```
GET about.htm HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
```

La primera línea, la de solicitud, tiene tres campos:

- El campo del método HTTP, que puede ser GET o POST; en este caso es GET
- El campo URI, que da el nombre del objeto para get o post; aquí es `about.htm`
- El campo de versión HTTP, que dice cuál es la versión del protocolo que está usando el cliente; en el ejemplo es 1.1

La segunda línea, la del agente usuario, muestra el tipo del cliente. La tercera línea dice los tipos de archivos que aceptará el cliente.

Después de recibir este mensaje de solicitud, el servidor envía un mensaje de respuesta HTTP. El servidor recupera el objeto solicitado (`about.htm`, en este ejemplo), usa la página o recurso para ensamblar el mensaje de respuesta HTTP y envía el mensaje de regreso al cliente. Tiene varias líneas, inclusive una de estado, algunas para el encabezado y el cuerpo del mensaje, que por lo general contiene el objeto solicitado. La línea de estado es la primera, y proporciona la versión de HTTP, un código de estado y un mensaje del servidor. Por ejemplo, una línea de estado como ésta:

```
HTTP/1.1 200 OK
```

Significa que la solicitud tuvo éxito (indicado por el código de estado 200 y el mensaje OK), y el objeto solicitado está en el cuerpo del mensaje que sigue. Otros valores de código de estado y los mensajes correspondientes indican que el recurso no se encontró, o que ocurrió un error. Las líneas de encabezado incluyen información sobre la fecha y hora de la respuesta, la fecha de modificación del objeto, el número de bytes en el contenido del objeto, y el tipo de contenido. El cuerpo del mensaje, que contiene al objeto real, va después de las líneas de encabezado. Por lo general contiene información de forma HTML y también contenido.

Cuando el navegador del cliente obtiene el mensaje de respuesta que contiene el objeto solicitado, usa HTML en el archivo para mostrar el objeto en forma apropiada. Si el objeto contiene vínculos usa el protocolo HTTP y los URI para establecer una conexión con cada recurso a fin de recuperar los objetos asociados.

HTTP es un protocolo **sin estado**. Cada mensaje de solicitud y su respuesta son autocontenidos, sin facilidades para recordar interacciones anteriores. Esto crea un problema en el contexto del comercio electrónico, que requiere una sesión continua con el usuario. La aplicación debe codificar cualquier información de estado ya sea en el cliente o el servidor, no en el protocolo.

13.2.4 HTML

HTML, el lenguaje de marcado de hipertexto es un formato de datos usado para presentar contenido en Internet. Se llama lenguaje de **marcado** porque los documentos HTML contienen marcas o etiquetas que proveen información del formato para el texto. Los lenguajes de marcado se han utilizado por muchos años para especificar el formato de los documentos; por ejemplo, especifican qué parte de un documento debe subrayarse, dónde insertar espacios, en qué lugar debe comenzar un párrafo nuevo, etc. Las etiquetas HTML están encerradas entre paréntesis de pico, < >. La mayor parte de las etiquetas ocurre en parejas, y tienen la forma <*tag*> y </*tag*>. La primera de las parejas especifica dónde comienza el for-

mato especial, y la segunda dónde termina. Por ejemplo, <**B**> significa *negritas*, de modo que en

```
La última palabra en esta oración está en <B> boldface</B>.
```

se interpreta que la <B> significa "cambiar a negritas" y </B> significa "quitar negritas". Un documento HTML comienza con la etiqueta <**HTML**> y termina con <**/HTML**>. El encabezado de un documento se encierra en las etiquetas <**HEAD**> y <**/HEAD**>, y el cuerpo se encierra en las etiquetas <**BODY**> y <**/BODY**>. Hay muchas otras etiquetas, pero algunas de las más útiles son las siguientes:

- <UL> para una lista no ordenada, utilizada para identificar una lista de ítems que siguen
- <LI> para una lista de ítems, usada para cada ítem en la lista
- <H1> para el encabezado 1, que especifica que debe usarse el formato usual para un encabezado de nivel 1
- <H2> a <H6>, para encabezados de los niveles 2 a 6
- <I> para itálicas
- <U> para subrayar

Todas estas etiquetas requieren una etiqueta de terminación, pero algunas, como la que se utiliza para un párrafo nuevo, <P> no siempre requieren una etiqueta de terminación.

Además del texto, un documento HTML contiene una variedad amplia de objetos tales como applets de Java, archivos de audio, imágenes, archivos de video y otros. Cuando el documento es recuperado por el navegador del usuario, el texto se muestra de acuerdo con los comandos de formato, las applets se ejecutan, los archivos de audio suenan, el video se muestra, etc., en la estación de trabajo del usuario. La figura 13.1 presenta un ejemplo de documento HTML.

13.2.5 XML

Aunque HTML funciona bien para mostrar texto, utiliza un conjunto limitado de etiquetas con significados fijos, lo que lo hace relativamente inflexible. En el ejemplo que aparece en la figura 13.1 es posible inferir los significados de los objetos, pero el documento no los describe. HTML es una aplicación de un lenguaje más poderoso y general, el lenguaje estándar generalizado de marcado (**SGML**, Standard Generalized Markup Language). SGML en realidad es un metalenguaje que permite a los usuarios definir sus propios lenguajes de marcado. Sin embargo, SGML es muy complejo. En 1996 World Wide Web Consortium XML Special Interest Group creó un lenguaje más sencillo basado en SGML y HTML, llamado **XML**. El objetivo era retener la flexibilidad de SGML y la sencillez de HTML. XML también permite que los usuarios definan su propio lenguaje de marcado. Por ejemplo, los usuarios tienen la posibilidad de crear etiquetas que describan los ítems de datos en un documento. La comunidad que trabaja con bases de datos tiene mucho interés en XML como medio para describir documentos de todos los tipos, inclusive bases de datos. Con el uso de XML es posible definir la estructura de las bases de datos heterogéneas, lo que facilita la traducción de datos entre bases de datos diferentes.

13.2.5.1 Documentos XML independientes

La figura 13.2 muestra un ejemplo de documento XML. Un documento de este tipo por lo general comienza con una **declaración** opcional que identifica la versión XML usada para el documento y, de manera opcional, el sistema de codificación utilizado, y/o si el docu-

FIGURA 13.1
Documento HTML que muestra una lista de clientes

```
<HTML>
<HEAD>
<TITLE>Customer List</TITLE>
</HEAD>
<BODY>
<H2>Corporate Customer List</H2>
<UL>
        <LI> WorldWide Travel Agency</LI>
        <LI> 10 Main Street, New York, NY 10001</LI>
        <LI> 212 123 4567</LI>
</UL>
<H2>Individual Customer List</H2>
<UL>
        <LI> Mary Jones</LI>
        <LI> 25 Spruce Street, San Diego, CA 92101</LI>
        <LI> 619 555 6789</LI>
</UL>
<UL>
        <LI> Alice Adams</LI>
        <LI> 25 Orange Blossom Street, Miami, FL 60601</LI>
        <LI> 305 987 6543</LI>
</UL>
</BODY>
</HTML>
```

mento tiene o no una definición de tipo documento (DTD) externa. El sistema de codificación más común para un documento XML es Unicode, identificado con UTF-8. Un documento sin DTD externa se describe como "independiente". Un ejemplo de declaración es la siguiente:

```
<?xml version=''1.0'' encoding=''UTF-8'' standalone=''yes''?>
```

Si hay una DTD externa, el nombre de los archivos que contienen la DTD aparece en la segunda línea, como en:

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<!DOCTYPE CUSTOMERLIST SYSTEM ''C:\BOOK\EXAMPLES\CUSTOMERLIST.DTD''>
```

La declaración va seguida por uno o más **elementos** XML, cada uno de los cuales tiene una **etiqueta de inicio** que muestra el nombre del elemento, algunos **datos caracteres** y una **etiqueta final**. XML es sensible a mayúsculas. Un elemento es el componente básico de un documento XML. Los elementos pueden ser subelementos de otros elementos, por lo que deben estar anidados en forma apropiada a fin de que si la etiqueta de inicio de un elemento aparece dentro del cuerpo de otro elemento, su etiqueta final también aparezca antes de la etiqueta final del elemento que lo contiene. El primer elemento en la DTD debe ser un único elemento raíz en el que estén anidados todos los demás elementos. La figura 13.2 presenta el elemento raíz CUSTOMERLIST, que contiene toda la información acerca de los clientes. Dentro de este elemento, el elemento CUSTOMER da toda la información sobre un cliente. Un elemento también puede estar vacío, lo que se indica con el empleo de la etiqueta de autoterminación:

```
<EMPTYELEMENT/>.
```

```
<?xml version="1.0" encoding="UTF–8" standalone="yes"?>
<CUSTOMERLIST>
<CUSTOMER type="Corporate" status="Active">
        <NAME>WorldWide Travel Agency</NAME>
        <STREET>10 Main Street</STREET>
        <CITY>New York</CITY>
        <STATE>NY</STATE>
        <ZIP>10001</ZIP>
        <AREACODE>212</AREACODE>
        <PHONE>123 4567</PHONE>
</CUSTOMER>
<CUSTOMER type="Individual"> <!--start of individual customers-->
        <NAME>Mary Jones</NAME>
        <STREET>25 Spruce Street</STREET>
        <CITY>San Diego</CITY>
        <STATE>CA</STATE>
        <ZIP>92101</ZIP>
        <AREACODE>619</AREACODE>
        <PHONE>555 6789</PHONE>
</CUSTOMER>
<CUSTOMER type="Individual" status="Inactive">
        <NAME>Alice Adams</NAME>
        <STREET>25 Orange Blossom Street</STREET>
        <CITY>Miami</CITY>
        <STATE>FL</STATE>
        <ZIP>60601</ZIP>
        <AREACODE>305</AREACODE>
        <PHONE>987 6543</PHONE>
</CUSTOMER>
</CUSTOMERLIST>
```

FIGURA 13-2
Documento de instancia XML que muestra una lista de clientes

Los elementos tienen **atributos** cuyos nombres y valores aparecen dentro de la etiqueta de inicio del elemento. Note que CUSTOMER tiene un atributo llamado type cuyos valores aparecen como "Corporate" o "Individual", y otro atributo de nombre `status` desde sus valores aparecen como "Active" o "Inactive". Los valores de los atributos se muestran entre comillas. El diseñador tiene la opción de representar los datos con el uso de un elemento o de un atributo. Los atributos ocurren sólo una vez dentro de cada elemento, mientras que los subelementos pueden ocurrir cualquier número de veces.

Es posible insertar **comentarios** en cualquier lugar que se desee del documento. Un comentario comienza con <!-- y termina con -->, y puede contener cualquier texto explicativo, excepto la cadena --, como se ilustra en la figura 13.2 para el segundo cliente.

Se hace referencia a los archivos externos, texto común, caracteres Unicode, o algunos símbolos reservados, con el empleo de una **referencia de entidad**. La referencia empieza con el carácter *ampersand*, **&**, y termina con un punto y coma, **;**. Cuando el documento se despliega en un navegador, la referencia se sustituirá por su contenido. Si se desea incluir símbolos

reservados que tengan un significado especial en XML, es necesario usar **&. . .;** como secuencia de escape. Estas entidades predefinidas son &, <, >, ' y '', y nos referiremos a ellas por sus nombres nemónicos `amp`, `1t`, `gt`, `apos` y `quot`. Por ejemplo, para indicar la cadena,

```
value>'a'&value<'z'
```

se escribiría:

```
value&gt;&apos;a&apos;&amp;value&lt;&apos;z&apos;
```

Las referencias también se pueden usar para insertar caracteres Unicode en el texto. Por ejemplo, para insertar el símbolo © se escribe `&00A9`; ya que el carácter Unicode para © es 00A9.

Se dice que un documento XML, como el que aparece en la figura 13.2, está **bien formado** si obedece las reglas de XML. Debe apegarse a los siguientes lineamientos:

- Comienza con una declaración XML como la que hay en la primera línea de la figura 13.2.
- Tiene un elemento raíz que contiene todos los demás elementos, como el elemento CUSTOMERLIST en la figura 13.2.
- Todos los elementos están anidados de manera apropiada. Si la etiqueta de inicio de un elemento ocurre dentro de otro, su etiqueta final ocurre también dentro del elemento que lo contiene. Por ejemplo, cada elemento NAME se encuentra anidado dentro del elemento CUSTOMER.

13.2.5.2 Las DTD

Los usuarios pueden definir su propio lenguaje de marcado por medio de escribir una **definición de tipo documento (DTD)** o con la escritura de un **esquema XML**. Una DTD es una especificación para un conjunto de reglas para los elementos, atributos y entidades de un documento. Se dice que un documento, al que ahora se hará referencia como un documento de instancia en oposición a uno de la DTD, que obedece las reglas de su DTD asociada es de **tipo válido**. Los procesadores pueden validar un documento de instancia XML por medio de revisar sus DTD. Varios documentos de instancia pueden compartir la misma DTD. La figura 13.3 muestra un ejemplo de DTD que describe la estructura del documento de instancia CUSTOMERLIST. Una DTD debe obedecer las reglas siguientes:

- La DTD está contenida en <!DOCTYPE *`nombre[DTDdeclaración]`*>
- *`nombre`* es el nombre de la etiqueta más exterior que lo contiene, que en este ejemplo es CUSTOMERLIST
- La *`DTDdeclaración`*, que está encerrada entre paréntesis cuadrados, da las reglas para los documentos que utilizan esta DTD
- Cada **elemento** se declara con el uso de una declaración de tipo con la estructura <!ELEMENT (*content type*)>, donde el tipo de contenido puede ser:
- Otros elementos que sean subelementos del elemento declarado. En la figura 13.3, el elemento CUSTLIST tiene un subelemento CUSTOMER, el elemento CUSTOMER tiene subelementos NAME, ADDRESS y TELEPHONE, y tanto ADDRESS como TELEPHONE a su vez tienen subelementos.
- #PCDATA, que significa que el valor del elemento consiste en datos de caracteres parseados. En la figura 13.3, el elemento NAME tiene este tipo de contenido, igual que STREET, CITY, STATE, ZIP, AREACODE y PHONE.
- El elemento vacío, indicado por <EMPTYELEMENT/>

```
<!DOCTYPE CUSTOMERLIST[
<!ELEMENT CUSTLIST (CUSTOMER)*>
<!ELEMENT CUSTOMER (NAME,ADDRESS+,TELEPHONE?)>
        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT ADDRESS (STREET,CITY,STATE,ZIP)>
                <!ELEMENT STREET (#PCDATA)>
                <!ELEMENT CITY (#PCDATA)>
                <!ELEMENT STATE (#PCDATA)>
                <!ELEMENT ZIP (#PCDATA)>
        <!ELEMENT TELEPHONE (AREACODE,PHONE)>
                <!ELEMENT AREACODE (#PCDATA)>
                <!ELEMENT PHONE (#PCDATA)>
<!ATTLIST CUSTOMER TYPE (Corporate|Individual) #REQUIRED>
<!ATTLIST CUSTOMER STATUS (Active|Inactive) "Active">
]>
```

FIGURA 13.3
DTD posible para el documento CUSTOMERLIST

- El símbolo ANY, que significa que se permite cualquier contenido.
- Una expresión regular que se construya a partir de estas cuatro opciones.
- En una declaración de elemento, el nombre de cualquier subelemento tiene la opción de ser seguido por uno de los símbolos *, + o ?, para indicar el número de veces que ocurre el subelemento dentro de su elemento que lo contiene. Los significados son los siguientes:
  - * El elemento ocurre cero o más veces. En la figura 13.3, los elementos CUSTOMER ocurren cero o más veces, lo que indica que CUSTOMERLIST podría no tener clientes o tener muchos.
  - + El elemento ocurre una o más veces. En la figura 13.3, ADDRESS ocurre una o más veces para cada cliente.
  - ? El elemento ocurre cero o una vez. En la figura 13.3, para cada cliente, TELEPHONE ocurre cero o una vez.
- **Atributos** lista las declaraciones para los elementos que son declarados fuera del elemento. Especifican cuáles elementos tienen atributos, el nombre de cada uno, el tipo de datos, los valores posibles (opcional), si se requiere el atributo, y el valor por default si lo hubiera. Tienen la forma <!ATTLIST elementoNombre attNombre (attTipo)default>. Existen varios tipos de atributos, inclusive los tipos de cadena y tipos numerados. El tipo de cadena se declara como **CDATA**. Un tipo enumerado se muestra con la lista de sus valores posibles. Una declaración de atributo también puede tener una especificación por default. Un atributo requerido se identifica con la escritura de #REQUIRED. Si el atributo no es requerido puede aparecer el valor por default, que será provisto cuando no se especifique ningún otro valor para el atributo. En la figura 13.3, el elemento CUSTOMER tiene dos atributos, TYPE y STATUS. El atributo TYPE es un tipo enumerado que tiene los posibles valores Corporate e Individual. Es un atributo requerido. El atributo STATUS tiene los posibles valores Active e Inactive, con Active como default. No es requerido.

Las DTD de dominio específico se han creado para cierto número de campos, y es probable que en el futuro se desarrollen más. Permitirán el intercambio terso de datos entre docu-

mentos con la misma DTD. Por ejemplo, MathML (Mathematics Markup Language, lenguaje de marcado de matemáticas) es una DTD que se usa mucho para documentos con especificaciones matemáticas.

13.2.5.3 Esquemas XML

Un **esquema XML** es una forma nueva y más poderosa para describir la estructura de documentos que las DTD. Permite una estructura más compleja, tipos adicionales de datos fundamentales, tipos de datos definidos por el usuario, vocabulario de dominio creado por el usuario (espacios de nombres) y también soporta la unicidad y restricciones de claves extranjeras. Un esquema XML define la organización y tipos de datos de una estructura XML. Se dice que un documento de instancia que obedece las reglas XML está **bien formado**, y un documento de instancia de **tipo válido** es aquel que se apega a una DTD. Si un documento de instancia se apega a un esquema XML se llama de **esquema válido**. Los esquemas XML son en sí mismos documentos XML y pueden validarse con respecto de los estándares provistos por World Wide Web Consortium, W3C, según se especifica en `www.w3.org`. Este sitio contiene un documento de esquema estándar XML para todos los documentos de esquema XML definidos por el usuario, así como vínculos a validadores de esquema, que se encuentran en: `http://www.w3.org/XML/Schema`. `XML` *Esquema* es un tema muy extenso y su tratamiento exhaustivo está más allá del alcance de este libro, pero a continuación se hace una introducción breve para dar sus elementos fundamentales.

En la figura 13.4 aparece un esquema XML simple. Como una DTD, un esquema XML lista los elementos y atributos. Los elementos pueden ser complejos, lo que significa que tienen subelementos, o simples. En la figura 13.4, el elemento `Customer List` es de tipo complejo y consiste en cualquier número de elementos `Customer`. `Customer` también es un elemento de tipo complejo que consiste en `Name`, `Address` y `Telephone`. `Name` es un elemento simple tipo cadena. `Address` es un tipo complejo que consiste en los elementos simples tipo cadena `Street`, `City`, `State` y `Zip`. `Telephone` es un tipo complejo que consiste en los elementos simples tipo cadena `AreaCode` y `Phone`. `Customer` tiene dos atributos, `Type` y `Status`. `Type` es un atributo requerido del tipo numerado. Su tipo básico es de cadena y sus valores posibles son “Corporate” o “Individual”. `Status` es de tipo cadena con el valor por default “Active”.

Pueden usarse atributos o elementos para almacenar valores de datos. Los atributos se utilizan para valores simples no repetidos, como Type y Status. Los elementos son complejos o simples y pueden ocurrir muchas veces. Type y Status podrían representarse como elementos en lugar de atributos, como se indica en la figura 13.5.

13.3 Arquitecturas multicapas

Para aplicaciones intensivas en datos se identifican tres funciones principales que se requieren en un ambiente de Internet: presentación, lógica de aplicación y administración de datos. La colocación de estas funciones depende de la arquitectura del sistema.

13.3.1 Arquitectura de una sola capa

Inicialmente, las bases de datos se crearon en un ambiente de mainframe centralizado, al que se accedía con procesamiento por lotes y terminales. La base de datos en sí residía en un almacenamiento secundario, y el sistema administrador de ella, la aplicación y la interfaz de usuario residían en una sola computadora, como se ilustra en la figura 13.6(a).

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name ="CustomerList">
 <xsd:sequence>
   <xsd:element name="Customer" minoccurs="0" maxoccurs="unbounded">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Name" type="xsd:string"/>
            <xsd:element name="Address" minoccurs="1" maxoccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="Street" type="xsd:string"/>
                     <xsd:element name="City" type="xsd:string"/>
                     <xsd:element name="State" type="xsd:string"/>
                     <xsd:element name="Zip" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
            <xsd:element name="Telephone" minoccurs="0" maxoccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="AreaCode" type="xsd:string"/>
                     <xsd:element name="Phone" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      <xsd:attribute name="Type" use="required">
         <xsd:simpleType>
            <xsd:restriction base="xsd:string">
               <xsd:enumeration value="Corporate"/>
               <xsd:enumeration value="Individual"/>
            </xsd:restriction>
         </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="Status" type="xsd:string" default="Active"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 </xsd:element>
</xsd:schema>
```

FIGURA 13.4
Esquema XML posible para CustomerList

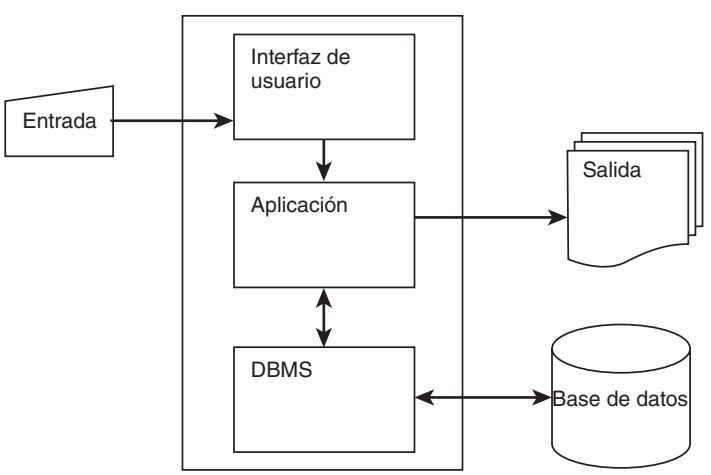
13.3.2 Arquitectura de doble capa

Cuando las PC se hicieron más baratas y se extendía su uso se desarrolló la arquitectura de doble capa. En esta arquitectura, la estación de trabajo del usuario u otros equipos funcionan como clientes, y el sistema administrador de la base de datos reside en un servidor. Los dos sistemas interactúan con el uso de un protocolo de redes. Si el cliente proporciona sólo la interfaz de usuario y el servidor corre la aplicación y maneja el acceso a los datos, el cliente funciona como **cliente ligero**. Esta arquitectura, que se ilustra en la figura 13.6(b), es común cuando se utilizan como clientes equipos sencillos tales como PDA o teléfonos celulares.

FIGURA 13.5
Otro Esquema XML para CustomerList

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name ="CustomerList">
    <xsd:sequence>
      <xsd:element name="Customer" minoccurs="0" maxoccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Name" type="xsd:string"/>
          <xsd:element name="Address" minoccurs="1" maxoccurs="unbounded">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="Street" type="xsd:string"/>
                <xsd:element name="City" type="xsd:string"/>
                <xsd:element name="State" type="xsd:string"/>
                <xsd:element name="Zip" type="xsd:string"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="Telephone" minoccurs="0" maxoccurs="unbounded">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="AreaCode" type="xsd:string"/>
                <xsd:element name="Phone" type="xsd:string"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="Type" type="xsd:string"/>
          <xsd:element name="Status" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
 </xsd:element>
</xsd:schema>
```

FIGURA 13.6(a)
Arquitectura de una sola capa



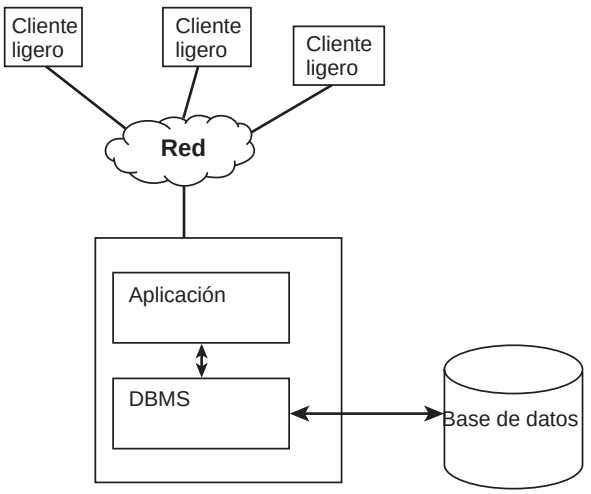


FIGURA 13.6(b)
Arquitectura de doble capa para clientes ligeros

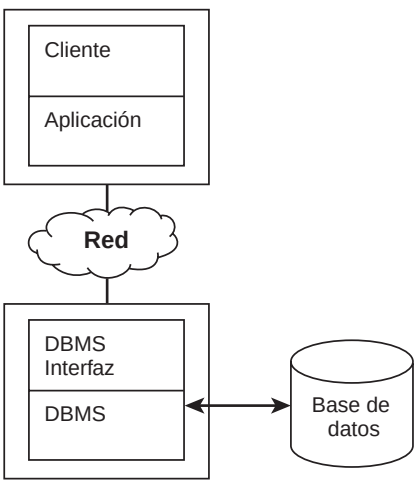


FIGURA 13.6(c)
Arquitectura de doble capa para clientes pesados

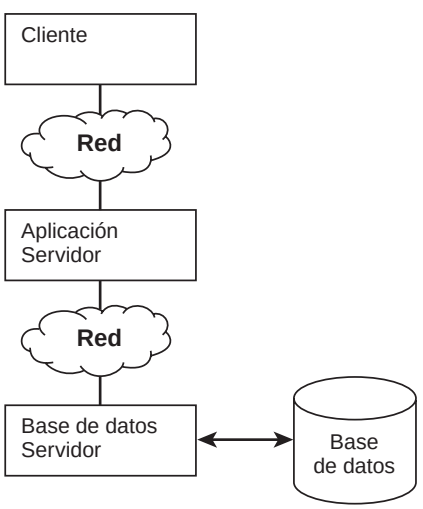


FIGURA 13.6(d)
Arquitectura de triple capa

Si el cliente hace algo del procesamiento de la aplicación así como del manejo de la interfaz de usuario, funciona como **cliente pesado** u **obeso**, como se ilustra en la figura 13.6(c). En la arquitectura de doble capa, las PC y las estaciones de trabajo funcionan como clientes pesados porque proporcionan la potencia de procesamiento y almacenamiento necesarios. El servidor maneja tareas tales como la validación y el acceso a los datos.

En este ambiente surgen varios problemas. El primero es un problema de escalabilidad, porque el servidor no es capaz de manejar un número grande de clientes simultáneamente. El número de conexiones independientes a la base de datos es limitado. El segundo problema es que como la lógica de la aplicación es manejada sobre todo por el cliente, el servidor tiene que confiar en que el código de ella no dañará la base de datos. Las aplicaciones en sí mismas tienen que incluir rutinas para el manejo de transacciones, inclusive para excepciones y deshacer, si fuera necesario. El problema de la confianza se enfrenta con el requerimiento del registro de los programas con el servidor. El servidor entonces rechazará las aplicaciones no registradas.

El tercer problema es que puede haber un número grande de clientes que ejecutan aplicaciones, y no hay un lugar central que mantenga la lógica del negocio.

13.3.3 Arquitectura de triple capa

Las **arquitecturas de triple capa** separan por completo a la lógica y la administración de los datos. El cliente maneja la interfaz de usuario, que se llama **capa de presentación** o la primera capa. La interfaz puede estar basada en la Web. Un **servidor de aplicación** ejecuta la lógica de la aplicación y constituye la **capa intermedia**. El servidor de la base de datos, que ejecuta el DBMS y administra el acceso a los datos, es la tercera capa. La red de comunicaciones conecta cada capa con la siguiente. Esta arquitectura en capas es similar a la formación de capas que se encuentra en los protocolos de comunicaciones y otros campos, y su propósito es dar apoyo a la estandarización entre proveedores y proporcionar flexibilidad. La arquitectura de triple capa permite:

- Atender a clientes ligeros, ya que sólo necesitan manejar la capa de presentación.
- Independencia de las capas. Cada capa utiliza la plataforma o software que esté mejor adaptado a su ambiente. Se usan API bien definidos para estandarizar y controlar las interacciones entre capas. El código de cualquier capa se puede modificar sin afectar a las otras.
- Mantenimiento de aplicaciones más fácil, ya que el servidor de aplicaciones es un lugar central en el que está almacenada la mayor parte de la lógica del negocio.
- Acceso integrado a los datos, porque la capa media maneja datos de varias bases de datos heterogéneas en una forma transparente para el cliente.
- Escalabilidad, porque la capa intermedia puede compartir conexiones de la base de datos con muchos clientes en forma simultánea.

13.3.3.1 Capa de presentación

En la primera capa debe presentarse una interfaz amigable con el usuario que sea apropiada para el equipo. Es común que en esta capa el usuario vea formas con las que hace solicitudes y mire las respuestas para ellas. Para este propósito es frecuente que se utilicen formas HTML en el que está incrustado código script en Perl, JavaScript, JScript, VBScript y otros lenguajes de escritura similares para brindar cierto procesamiento sencillo en el lado del cliente. También se utilizan hojas de estilo que especifican cómo están presentados los datos en equipos específicos.

Para pasar valores de entrada se emplean **formas HTML** del cliente a la capa intermedia. El navegador del usuario despliega una página Web que contiene un formato con campos de entrada, áreas para texto, listas desplegables, y otros objetos en los que el usuario introduce los datos de entrada. Un documento HTML contiene formas múltiples. Cada forma puede tener cualesquiera etiquetas HTML, excepto otra etiqueta FORM. En la figura 13.7 se presenta el código HTML para una forma sencilla de firma. El HTML para una forma comienza con una etiqueta FORM, y tiene el formato general

```
<FORM ACTION="applicationURL" METHOD="GET" NAME="formName">
<INPUT TYPE="inputType" SIZE=size NAME="name" VALUE="value">
. . .
</FORM>
```

Una etiqueta FORM tiene los campos siguientes:

- ACTION: da el URI de la aplicación que procesará la entrada provista en la forma. Es una referencia relativa que especifica el directorio y programa por ejecutar. Si no se especifica directorio, usa el URI de la página actual.
- METHOD: especifica el método usado para enviar al servidor los datos de entrada desde la forma llena. Los valores posibles son GET o POST. Un método GET envía el nombre del programa y la información de entrada desde la forma en un solo paso. Con un método POST, el nombre del programa forma parte del URL, y la información de entrada se transmite en un mensaje separado.
- NAME: permite de manera opcional que la forma reciba un nombre, para que con éste los programas hagan referencia a la forma y a sus campos.

  Las formas incluyen etiquetas INPUT, SELECT y TEXTAREA que especifiquen objetos de entrada para el usuario. La etiqueta TAG no tiene etiqueta de terminación. Los atributos de la etiqueta Input incluyen TYPE, SIZE, NAME y VALUE. El atributo TYPE especifica el tipo del campo de entrada. Si el tipo es texto, el cuadro de entrada aparece como un rectángulo vacío del tamaño especificado en SIZE, donde el usuario escribe texto. Si el tipo es password, los caracteres que se introduzcan aparecerán como asteriscos a manera de protección. Si el tipo es reset, aparecerá un botón con leyenda que al ser presionado por el usuario hará que en todos los campos de entrada de la forma se asignen sus valores por default. Si el tipo es submit, aparecerá un botón etiquetado que al ser presionado por el usuario envía valores de todos los campos de entrada al servidor. NAME proporciona un nombre al campo para identificar su contenido ante el servidor. Debe darse nombre a todos los tipos de entrada excepto a submit y reset. VALUE es un campo opcional que se usa para especificar el valor por default de un campo de entrada. También se emplea para dar una etiqueta al botón para enviar o volver a sus valores de inicio.

```
<FORM ACTION="/cgi-bin/signon.cgi" METHOD="GET" NAME="UserLogin"
      <P>
      User ID:<INPUT TYPE="text" SIZE=20 NAME="userID">
      <P>
      Password:<INPUT TYPE="password" SIZE=20 NAME="password">
      <P>
      <INPUT TYPE="submit" VALUE="Log on">
      <INPUT TYPE="reset" VALUE="Clear">
<FORM>
```

FIGURA 13.7
Forma HTML simple

La entrada del usuario también puede provenir de las etiquetas TEXTAREA y SELECT, en una forma similar a las etiquetas INPUT.

El URI que aparece en el atributo ACTION de la etiqueta FORM debe ser el de una aplicación, página o script. Si el método es GET, la acción y cada nombre de área de entrada y valor se concatenan en una solicitud URI que tiene la forma:

```
action?inputName1=inputValue1&inputName2=inputValue2&inputName
3=inputValue3
```

Por ejemplo, para la forma que aparece en la figura 13.7, si el usuario introdujo el nombre Adam Adams y la clave (password) sesame, la solicitud URI sería:

```
http://localhost/cgi-bin/signon.cgi?userID=Adam+Adams&password=sesame
```

Note que los espacios vacíos están reemplazados con + y los nombres están conectados a sus valores por medio de =. Cualesquiera caracteres especiales están codificados con el uso del formato %xxx, donde xxx es el código ASCII del carácter.

Lenguajes de script: Los lenguajes de script, inclusive Perl, JavaScript, JScript y VBScript, se utilizan para agregar programas sencillos que corren en la capa cliente. Los scripts se identifican con una etiqueta SCRIPT incrustada en un documento HTML. Su forma es:

```
<SCRIPT LANGUAGE="scriptLanguageName" SRC="externalFileName"></SCRIPT>
```

El atributo LANGUAGE asigna el nombre del lenguaje de script, como "JavaScript". El atributo SCR asigna el nombre del archivo externo con el código del script, como "checkForms.js". El código queda incrustado en forma automática en el documento HTML. En el cliente se usan scripts para procesos tales como revisar la validez de los datos que el usuario haya introducido en la forma, identificar el tipo de navegador con que trabaja el cliente y controlar algunas funciones de éste.

Hojas de estilo: Las hojas de estilo **CSS** y **XSL** contienen instrucciones que dicen al navegador Web u otro controlador de presentación cómo desplegar los datos en forma apropiada para el equipo del cliente. Por ejemplo, la misma página Web se mostrará un poco diferente con Netscape que con Explorer, y muy distinto en un teléfono celular o PDA. La hoja de estilo dice cómo se muestran los tipos de letra, colores, márgenes y colocación de las ventanas para el dispositivo específico.

Hojas de estilo en cascada (CCS) pueden ser utilizadas en documentos HTML. Y la misma hoja puede ser utilizada para muchos documentos, lo que permite que los diseñadores de Web apliquen una plantilla a todas las páginas de un sitio dado, si la organización desea tener una vista uniforme de ellas. En el documento HTML se puede incluir un vínculo a la hoja por medio de escribir una etiqueta para esto con la forma:

```
<LINK REL="StyleSheet" HREF="nameOfStyleSheet">
```

La hoja de estilo consiste en un grupo de especificaciones para las propiedades del documento, como colores, fuentes, espaciado y otros. Cada especificación está dada por una línea de la hoja de estilo, que tiene la forma:

```
Atributo {propiedad; valor}
```

Por ejemplo, para establecer un tamaño de fuente y color para los encabezados de nivel uno, se podría escribir en CSS:

```
H1 {FONT-SIZE: 24pt; COLOR: blue}
```

Entonces, un documento HTML que use la hoja de estilo reemplazaría cualesquiera etiquetas H1 con las especificaciones dadas en la hoja de estilo.

Para escribir hojas de estilo para archivos XML se utiliza el lenguaje extensible de hoja de estilo (XSL, Extensible Stylesheet Language). Los archivos XSL contienen especificaciones

de atributos que controlan el despliegue de los documentos XML en una forma similar a la forma en que CSS controla el de los documentos HTML. Es un lenguaje mucho más poderoso que el CSS, permitiendo cambiar la estructura de los documentos con el uso del Lenguaje de transformación XSL (XSLT), a fin de que sea posible hacer referencia a partes de ellos con el uso del lenguaje de trayectoria XML (XPath) y dar formato a objetos para su despliegue.

13.3.3.2 Capa intermedia

El servidor de la aplicación en la capa intermedia es responsable de ejecutar las aplicaciones. Esta capa determina el flujo del control, adquiere datos de entrada de la capa de presentación, hace solicitudes de datos al servidor de la base de datos, acepta resultados de las consultas a partir de la capa de la base de datos y las utiliza para ensamblar páginas HTML generadas dinámicamente. El procesamiento del lado del servidor involucra muchas tecnologías diferentes tales como *Servlets* de Java, páginas de Servidor en Java, etcétera.

Puede ser utilizado **CGI (Interfaz Gateway Común)** para conectar formas HTML con programas de aplicación. Este protocolo define el modo en que las entradas se pasan a los programas que se ejecutan en la capa intermedia. Los programas que utilizan este protocolo se llaman **scripts CGI**, y es frecuente que se escriban en Perl. Un script normal en Perl extrae los argumentos de entrada de la forma HTML y construye en forma dinámica una página Web como salida del programa, la cual envía al navegador.

Es posible que cada página solicite iniciar un proceso nuevo en el servidor, pero esto limita el número de solicitudes simultáneas que se pueden manejar. En vez de ello, un servidor de aplicaciones mantiene un conjunto de hilos o procesos y los usa para ejecutar solicitudes, lo que evita el trabajo indirecto de crear un nuevo proceso para cada solicitud. Permiten el acceso concurrente a varias fuentes de datos. El servidor de aplicaciones también administra las sesiones, detecta cuándo inician éstas, identifica cuáles sesiones pertenecen a cuáles usuarios y cuándo termina una sesión. Para identificar una sesión, el servidor utiliza cookies, los cuales son campos ocultos en las formas HTML y extensiones URI. Una cookie es una pequeña cadena de texto que el servidor envía al navegador del usuario cuando éste se conecta. En esencia es un identificador de sesión que se almacena en la memoria del navegador y permanece ahí después de que la sesión termina. Siempre que el usuario se reconecta al servidor, el navegador devuelve una copia de la cookie al servidor, con lo que establece que el usuario es el mismo que el de la sesión anterior. Como se ve, las cookies se usan para guardar información del usuario entre las sesiones.

Una función importante del servidor de aplicaciones es **mantener el estado**, ya que HTTP es un protocolo carente de éste. Cualquier transacción de etapas múltiples que requiera varias interacciones con un usuario necesita que se mantenga el estado de la transacción. En una transacción común de comercio electrónico, el usuario firmará, verá productos recomendados para su compra con base en sus intereses personales, navegará por las ofertas del sitio, colocará artículos en una canasta que modificará varias veces, y luego pasará a la compra. En ese momento, el comprador tiene que introducir el nombre y domicilio de envío, dar o confirmar su nombre y dirección, proporcionar un número de tarjeta de crédito y fecha de expiración, y solicitar cualesquiera servicios especiales tales como envoltura para regalo, envío urgente, etc. Debe calcularse el importe total, inclusive impuestos y cargos adicionales, y el comprador debe estar de acuerdo con el monto antes de que finalice la compra. Después se envía un mensaje de confirmación al comprador. Estas etapas requieren que se mantenga el estado de la transacción. Como HTTP es un protocolo sin estado, el cliente o el servidor deben ser capaces de mantenerlo, y para este propósito se utilizan cookies por el lado del cliente. Las cookies se generan fácilmente en la capa intermedia con el uso de la clase Cookie de Java y se envían al cliente, donde se almacenan en la memoria caché del navegador. El navegador del cliente envía la cookie con cada solicitud. Un problema de esta

solución es que los usuarios pueden rechazar la aceptación de cookies o limpiar la memoria caché del navegador antes de que expiren, lo que las borraría. El estado también se puede mantener en la base de datos, pero todas las consultas o modificaciones del estado, como una firma o modificación de la canasta de compras, requeriría acceder a la base. Es mejor recurrir a esta solución para información relativamente permanente, como órdenes pasadas, nombre y dirección del cliente, preferencias de éste, etc. También es posible mantener el estado en la capa intermedia, con el uso de archivos locales en ese nivel.

El procesamiento en el lado del servidor usa servlets de Java, páginas JavaServer (JSP) u otras tecnologías. Los servlets de Java son programas que implementa la interfaz Servlet. Usan la clase HttpServlet, que proporciona métodos para recibir entradas desde formas HTML y para construir páginas Web dinámicas a partir de los resultados del programa. Tanto la solicitud como la respuesta se pasan como parámetros para los servlets, que tienen toda la funcionalidad de los programas de Java estándar, lo que les permite implementar transacciones complicadas que involucran la base de datos. Como se trata de programas Java, son plataformas independientes y corren en contenedores. Acceden a todas las API inclusive JDBC. Corren en la capa intermedia, ya sea en un servidor Web o de aplicaciones.

Las páginas JavaServer están escritas en HTML, pero contienen etiquetas especiales que permiten algún código de tipo servlet que use JavaScript, el cual es un lenguaje de programación muy completo que puede incrustarse en HTML. No es usual que las páginas JavaServer soporten aplicaciones completas, sino que se usan para construir interfaces de usuario. Sin embargo, se ejecutan como servlets.

13.4 Modelo de datos semiestructurado

La Web puede verse como una colección masiva de documentos que incluyen vínculos y contenido multimedios como sonido, imágenes o video, así como texto. Las técnicas para buscar documentos de texto se desarrollaron originalmente en el campo de la recuperación de información, que era el dominio de los bibliotecarios más que del público o de los administradores de bases de datos. Sin embargo, con el desarrollo de la Web, la creación de colecciones de documentos y la búsqueda de aquéllos se volvió una actividad que realizan sobre todo los usuarios normales de computadoras y no los bibliotecarios. Debido a la disponibilidad de recursos ricos en información que proporciona la Web, han convergido los campos de su recuperación y de la administración de bases de datos, disciplina ésta en la que los documentos Web son vistos como fuentes de datos similares a otras bases, y el proceso de búsqueda se considera como una consulta a la base de datos. XML proporciona un puente entre las disciplinas de la recuperación de información, que tradicionalmente ha incluido lenguajes de marcado, y la administración de bases de datos, estos últimos fundamentados en modelos de datos estructurados. XML permite que se marquen documentos con metadatos, e incluye etiquetas que les dan cierta estructura. El modelo de datos semiestructurado es un método para describir dicha estructura parcial, el cual contiene un conjunto de nodos, cada uno con datos que posiblemente tengan esquemas diferentes. El nodo en sí contiene información acerca de la estructura de su contenido.

13.4.1 Representación gráfica

El modelo de datos semiestructurado se representa con un grafo que consiste en nodos y arcos. Cada documento se representa como un **árbol** que tiene un solo nodo **raíz** y una secuencia de nodos **hijos**. Cada nodo hijo tiene a su vez otros nodos hijos, que tiene otros hijos, y así sucesivamente. Una regla estricta para los árboles es que cada nodo, excepto la raíz, tenga exactamente un nodo padre que esté en un nivel superior del árbol. Los nodos

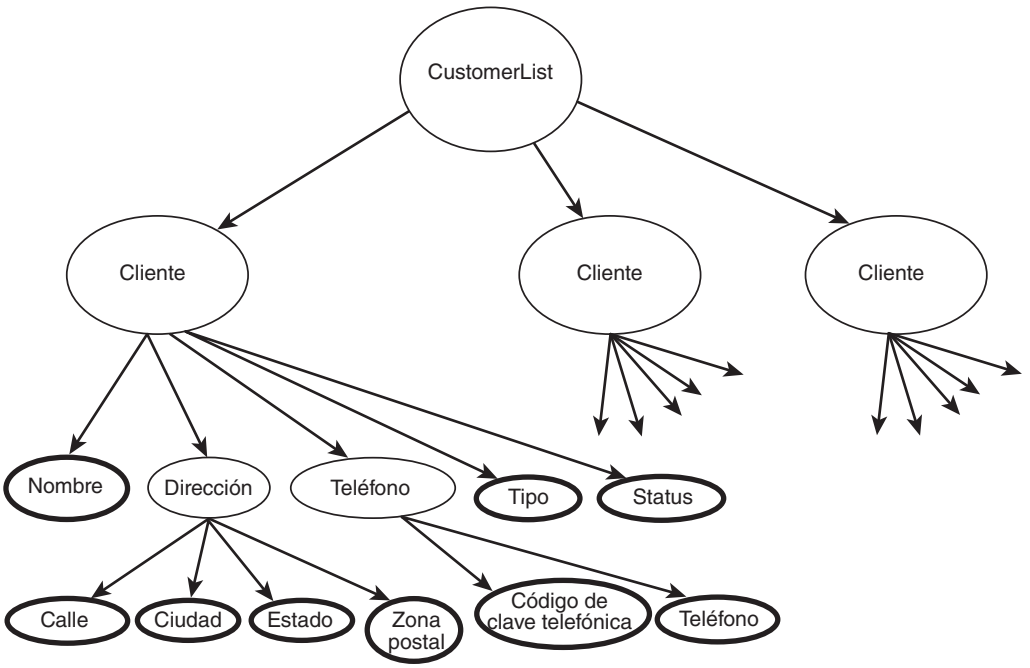


FIGURA 13.8
Representación jerárquica de CustomerList

representan objetos complejos o valores atómicos. Un arco representa ya sea una relación entre un objeto y su subobjeto o entre un objeto y su valor. Los nodos hoja, que no tienen subobjetos, representan valores. No hay un esquema separado, ya que el grafo se autodescribe. Por ejemplo, la figura 13.8 ilustra una representación jerárquica de los datos en `Customer List` que corresponde al esquema XML de la figura 13.5. Todo el documento `CustomerList` se vuelve la raíz del árbol. Cada elemento `Customer` se representa como un hijo de la raíz. Los subelementos de `Customer`, es decir `Name`, `Address`, `Telephone`, `Type` y `Status`, se representan como hijos de `Customer`. Los subelementos de `Address` y `Telephone` están representados como sus hijos. En la figura 13.9 se muestra un documento de instancia para este esquema. El documento de instancia refleja la estructura del esquema XML (o de una DTD, si fuera aplicable), y muestra los valores proporcionados por los nodos hoja del árbol. Todos los datos residen en los nodos hoja. Si existen atributos cualesquiera, ellos también se convierten en nodos hoja.

13.4.2 Manipulación de datos XML

El lenguaje actual de consultas estándar W3C para datos XML, es XQuery. Como está codificado en XML usa la estructura lógica abstracta de un documento XML, posiblemente en una DTD o una definición de esquema XML.

13.4.2.1 Expresiones XPath

Las consultas usan el concepto de **expresión de trayectoria**, que viene de un lenguaje anterior llamado **XPath**. Una expresión de trayectoria por lo general consiste en el nombre del documento y una especificación de los elementos por recuperar, con el uso de una relación de trayectoria. Un ejemplo de esto es

```
Document("CustomerList.xml")//Customer/Name
```

La expresión especifica el nombre del documento que va a buscarse y una o más etapas especificadas por / o por //. El separador // significa que el elemento siguiente puede anidarse en cualquier lugar dentro del precedente (es decir, cualquier descendiente del nodo

FIGURA 13.9

Documento de instancia para el esquema CustomerList de la figura 13.5

```
<CustomerList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="C:\XML\XMLExample1">
  <Customer>
    <Name>WorldWide Travel Agency</Name>
    <Address>
       <Street> 10 Main Street </Street>
       <City> New York </City>
       <State> NY </State>
       <Zip> 10001 </Zip>
    </Address>
    <Telephone>
       <AreaCode> 212 </AreaCode>
       <Phone> 123 4567 </Phone>
    </Telephone>
    <Type>Corporate</Type>
    <Status>Active</Status>
  </Customer>
  <Customer>
    <Name>Mary Jones</Name>
    <Address>
       <Street> 25 Spruce Street </Street>
       <City> San Diego </City>
       <State> CA </State>
       <Zip> 92101 </Zip>
    </Address>
    <Telephone>
       <AreaCode> 619 </AreaCode>
       <Phone> 555 6789 </Phone>
    </Telephone>
    <Type>Individual</Type>
    <Status>Active</Status>
  </Customer>
<Customer>
    <Name>Alice Adams</Name>
    <Address>
       <Street> 25 Orange Blossom Street </Street>
       <City> Miami </City>
       <State> FL </State>
       <Zip> 60601 </Zip>
    </Address>
    <Telephone>
       <AreaCode> 305 </AreaCode>
       <Phone> 987 6543 </Phone>
    </Telephone>
    <Type>Individual</Type>
    <Status>Inactive</Status>
  </Customer>
</CustomerList>
```

precedente), mientras que el separador / significa que el siguiente elemento nombrado debe anidarse inmediatamente bajo el precedente (es decir, en el grafo debe ser un nodo hijo del anterior). Las etapas en la expresión de trayectoria hacen que la lectura avance a través del documento. Los nodos del grafo para un documento XML estructurado son ordenados con el empleo del preorden transversal, que es el que va de arriba hacia abajo, y de izquierda a derecha. Para un documento XML todo el documento es el nodo raíz, que va seguido por nodos de elementos, otros tipos de nodos que no se han estudiado aquí, y nodos de atributos. Cada elemento precede a sus hijos, y todos los hijos de un elemento nodo preceden a los hermanos de dicho nodo. Los nodos de todo el documento son vistos como totalmente ordenados por el procesador del lenguaje. En la figura 13.10, los números en los nodos muestran el orden en que cada nodo es encontrado con el uso de un recorrido transversal normal (hacia delante). Se da a cada nodo un identificador de objeto basado en este orden. En el documento de instancia en la figura 13.9, los nodos se listan en el mismo orden. La expresión de trayectoria normalmente se evalúa por medio de leer hacia delante en el documento hasta encontrar un nodo del tipo especificado. Para la expresión de trayectoria en este ejemplo, se encuentra el primer nodo cliente y el nombre de éste, WorldWide Travel Agency, se convierte en el objetivo. En una expresión de trayectoria se pueden agregar condiciones a cualesquier nodo. Por ejemplo, si se escribe,

```
doc("CustomerList.xml")//Customer/Address[State="CA"]
```

el primer nodo que satisface esta condición es el nodo dirección del segundo cliente. La condición fuerza a la lectura a avanzar a través de los nodos del documento hasta encontrar uno que la satisfaga. XQuery permite direcciones de lectura tanto hacia delante como hacia atrás. El usuario especifica un eje o dirección de búsqueda. Un eje hacia delante permite buscar un hijo, descendiente, a sí mismo, atributo, siguiente hermano, etc. Un eje inverso permite la búsqueda de un padre, ancestro, a sí mismo, hermano precedente, etcétera.

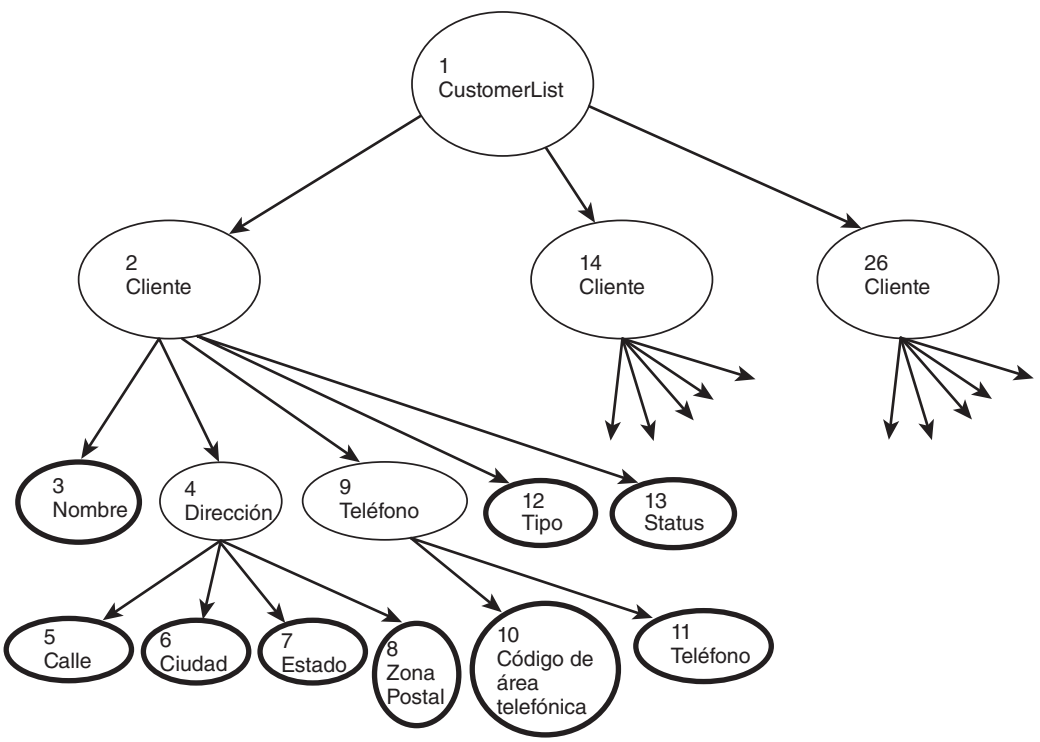


FIGURA 13.10
Preorden transversal de un grafo

13.4.2.2 Expresiones XQuery

Una consulta simple que use una expresión de trayectoria por lo general devuelve un conjunto de nodos que califican para la expresión. Por ejemplo, para la consulta,

```
FOR
   $N IN doc("CustomerList.xml")//Customer/Name
RETURN <Result> $N </Result>
```

liga la variable $N a cada nodo `Name` en cuestión. La ejecución de la consulta comienza con la lectura del nodo raíz, después avanza al primer nodo `Customer`, luego al nodo `Name` que es hijo de cliente. Este nodo se convierte en el **nodo de contexto**, también llamado nodo actual, el cual es ligado a la variable. Desde esta posición, la lectura avanza hacia el nodo siguiente buscado, es decir, el siguiente nodo `Name`, que se vuelve el nuevo nodo de contexto, y así sucesivamente. La consulta construye una lista de nombres y la cláusula RETURN genera un documento XML que contiene la lista de los valores a los que estuvo ligada la variable $N. A cada uno de los nodos `Name` encontrados por la consulta se aplica la etiqueta `Result`. El resultado de esta consulta es el documento XML:

```
<Result><Name> WorldWide Travel Agency </Name></Result>
<Result><Name> Mary Jones </Name></Result>
<Result><Name> Alice Adams </Name></Result>
```

13.4.2.3 Expresiones FLWOR

XQuery usa una forma general llamada expresión FLWOR, la cual permite definir FOR, LET, WHERE, ORDER BY y RETURN, que son las cláusulas encontradas en tales expresiones. No se requieren todas estas cláusulas. Las expresiones permiten asignar a las variables los resultados, y también la iteración a través de los nodos de un documento. También permiten que se realicen combinaciones (joins) y que se reestructuren los datos. Un ejemplo es el siguiente,

```
FOR $C IN
   doc("CustomerList.xml")//Customer)
WHERE $C/Type="Individual"
ORDER BY Name
RETURN <Result> $C/Name, $C/Status </Result>
```

Que produce el documento XML que sigue:

```
<Result><Name>Alice Adams</Name><Status>Inactive</Status></Result>
<Result><Name>Mary Jones</Name><Status>Active</Status></Result>
```

Los resultados están ordenados por `Name`, según se especifica en la consulta. Si no se establece ningún orden los resultados aparecen en el que se encuentran en el documento.

XQuery también permite la construcción de nodos y la realización de combinaciones. Por ejemplo, el esquema que se muestra en la figura 13.11 describe un segundo documento XML que represente las órdenes hechas por los clientes. En la figura 13.12 aparece un documento de instancia. Se usa XQuery para formar una combinación del documento `CustomerList` con el documento `OrdersList`. Por ejemplo, la consulta "Encontrar el nombre y dirección del cliente para cada orden" requiere una combinación de estos documentos. La consulta se expresa así:

```
<Result>
{ FOR $O IN doc("OrdersList.xml")/Order,
   $C IN doc("CustomerList.xml")/Customer[Name=$O/Customer]
ORDER BY Customer
RETURN
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name ="OrdersList">
    <xsd:sequence>
      <xsd:element name="Order" minoccurs="0" maxoccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="OrderNumber" type="xsd:integer"/>
            <xsd:element name="Customer" type="xsd:string" minoccurs="1"/>
            <xsd:element name="OrderDate" type="xsd:date" minoccurs="1"/>
            <xsd:element name="Item" minoccurs="1" maxoccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                 <xsd:element name="ItemName" type="xsd:string"/>
                 <xsd:element name="Price" type="xsd:decimal"/>
                 <xsd:element name="Quantity" type="xsd:integer"/>
               </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:element>
</xsd:schema>
```

FIGURA 13.11
Esquema XML para OrdersList

```
    <OrderCust>
      {$O/OrderNumber, $O/Customer, $C/Address}
    </OrderCust>
  </Result>
```

Esta expresión construye un documento XML que consiste en elementos `OrderCust`, cada uno de los cuales tiene un número de orden, nombre de cliente, dirección de cliente, arreglado por orden del nombre del cliente. La condición `[Name=$O/Customer]` especifica que el nodo `Name` del elemento `Customer` en el documento `CustomerList` coincide con el elemento `Customer` en el documento `Orders`, que es una condición de combinación.

XQuery da muchas funciones predefinidas, inclusive count, avg, max, min y sum (contar, promedio, máximo, mínimo y suma) que se utilizan en expresiones FLOWR. El ejemplo que sigue lista los clientes que tienen dos o más órdenes:

```
<RepeatList>
FOR $C IN DISTINCT(doc("CustomerList.xml"))
LET $O := doc("OrderList.xml")/Customer[Customer=$C.Name]
WHERE count($O)>1
RETURN
<RepeatCustomer>
{$C/Name,$C/Address}
</RepeatCustomer>
<RepeatList>
```

FIGURA 13.12
Documento de instancia OrdersList

```
<OrdersList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="C:\XML\XMLExample2">
  <Order>
    <OrderNumber>1001</OrderNumber>
    <Customer>Mary Jones</Customer>
    <OrderDate>08-Feb-2004</OrderDate>
    <Item>
       <ItemName>pen</ItemName>
       <Price>15.99</Price>
       <Quantity>1</Quantity>
    </Item>
    <Item>
       <ItemName>desk lamp</ItemName>
       <Price>34.98</Price>
       <Quantity>1</Quantity>
    </Item>
  </Order>
<Order>
    <OrderNumber>1002</OrderNumber>
    <Customer>WorldWide Travel Agency</Customer>
    <OrderDate>8-Feb-2004</OrderDate>
    <Item>
       <ItemName>folder</ItemName>
       <Price>1.59</Price>
       <Quantity>12</Quantity>
    </Item>
    <Item>
       <ItemName>printer</ItemName>
       <Price>336.49</Price>
       <Quantity>1</Quantity>
    </Item>
    <Item>
       <ItemName>telephone</ItemName>
       <Price>95.00</Price>
       <Quantity>1</Quantity>
    </Item>
</Order>
<Order>
    <OrderNumber>1003</OrderNumber>
    <Customer>Mary Jones</Customer>
    <OrderDate>10-Feb-2004</OrderDate>
    <Item>
       <ItemName>filing cabinet</ItemName>
       <Price>45.00</Price>
       <Quantity>1</Quantity>
    </Item>
  </Order>
</OrdersList>
```

Para los datos mostrados en los documentos de instancia de las figuras 13.9 y 13.12, el documento xml `RepeatList` será:

```
<RepeatCustomer>
<Name>Mary Jones</Name>
<Address>
<Street>25 Spruce Street</Street>
<City>San Diego</City>
<State>CA</State>
<Zip>92101</Zip>
</Address>
</RepeatCustomer>
```

XQuery es un lenguaje muy poderoso con muchas características adicionales que están más allá del alcance de este libro. Todos los detalles de las especificaciones actuales del lenguaje se encuentran en el sitio Web W3C.

13.5 XML y las bases de datos relacionales

En ocasiones es deseable guardar datos XML en una base de datos relacional en forma que permita su recuperación y modificación. El aspecto fundamental es cómo mapear la estructura jerárquica de los documentos XML en un modelo relacional. Una solución sencilla pero que no es flexible es hacer que todo el documento esté representado como un solo atributo. Una solución más satisfactoria es "dividir" el documento en partes que se representan como atributos. Sin embargo, esto requiere un algoritmo de mapeo inteligente. El esquema XML puede ser utilizado para facilitar este mapeo. También es deseable poder transformar datos relacionales en formato XML y publicarlos de este modo.

13.5.1 Representación y manipulación de datos XML en una base de datos relacional

Muchos proveedores de sistemas de administración de bases de datos relacionales han extendido sus tipos de datos nativos a fin de que permitan el almacenamiento de documentos XML. Por ejemplo, Oracle proporciona el tipo de datos XMLType. Un método para manejar documentos XML es crear una tabla en la que cada documento se almacena en una única tupla. Por ejemplo, el enunciado,

```
CREATE TABLE xml_Customer of XMLTYPE;
```

crea una tabla con una sola columna que contiene documentos. En realidad, la columna es una seudocolumna, XMLDATA, para la que el tipo subyacente es CLOB. Para guardar y recuperar la columna se utilizan las operaciones CLOB habituales. Para llenar la tabla se emplea un enunciado INSERT, como sigue:

```
INSERT INTO xml_Customer VALUES
(xmltype('<?xml version="1.0"?>
<Customer><Name>WorldWide Travel Agency</Name><Address><Street>
10 Main Street </Street><City> New York </City><State> NY </State><Zip> 10001
</Zip></Address><Telephone><AreaCode> 212 </AreaCode><Phone> 123 4567
</Phone></Telephone><Type>Corporate</Type><Status>Active</Status>
</Customer>
<Customer><Name>Mary Jones</Name><Address><Street> 25 Spruce Street
</Street><City> San Diego </City><State> CA </State><Zip> 92101 </Zip>
</Address><Telephone><AreaCode> 619 </AreaCode><Phone> 555 6789</Phone>
</Telephone><Type>Individual</Type><Status>Active</Status>
```

```
</Customer> <Customer><Name>Alice Adams</Name><Address><Street>
25 Orange Blossom Street </Street><City> Miami </City><State>
FL </State><Zip> 60601 </Zip></Address><Telephone><AreaCode>
305 </AreaCode><Phone> 987 6543 </Phone></Telephone><Type> Individual
</Type><Status>Inactive</Status></Customer>'));
```

Es obvio que este proceso de inserción puede tomar datos de un documento de instancia XML, aun sin un esquema XML. Se puede consultar la tabla con el uso de enunciados Select de SQL, tales como:

```
SELECT c.getClobVal() FROM xml_Customer c;
```

Aunque el tratamiento de un documento como un solo atributo es sencillo y posible sin un esquema XML, está restringido debido a las limitaciones de los operadores CLOB. Si se desea acceder a los elementos individuales de los documentos se tiene que recurrir a operadores XPath o XQuery.

Una alternativa es usar un esquema XML y crear una tabla en la que se mapeen los elementos XML a objetos en una base relacional de objetos. Por ejemplo, si se tiene un esquema como el que se ilustra en la figura 13.5 es posible crear una tabla en la que cada elemento `Customer` es una tupla, por medio de escribir lo siguiente:

```
CREATE SCHEMA xml_Orders OF XMLTYPE
XMLSCHEMA "schema location"
ELEMENT "Order";
```

Las columnas de la tabla corresponden a los elementos de `Order` en el documento XML. Igual que antes se pueden insertar registros, pero ahora también es posible hacer referencia a los elementos individuales con el uso de la seudocolumna XMLDATA. Por ejemplo, se puede imponer una restricción de unicidad a `OrderNumber` con la escritura de lo que sigue:

```
ALTER TABLE xml_Orders ADD (UNIQUE(XMLDATA."OrderNumber"));
```

Es posible usar SQL con expresiones XPath para recuperar valores de la base de datos. La función EXTRACTVALUE se utiliza con una instancia XMLType, y usa una expresión XPath para devolver un valor escalar del tipo VARCHAR2. Por ejemplo, la consulta SQL,

```
SELECT EXTRACTVALUE('/Customer/Name') AS "Customer Name"
FROM xml_Customers;
```

produce una tabla que tiene nombres de clientes.

Nombre del cliente
WorldWide Travel Agency
Mary Jones
Alice Adams

Debido a que se crea la tabla `xml_Customer` con una sola columna CLOB, sin usar un esquema XML, hay limitación a una devolución VARCHAR2. Sin embargo, si la base de datos se creó a partir de un documento que usa su esquema XML, como se hizo para `xm1_Orders`, pueden ser devueltos tipos escalares adicionales, con base en los tipos de los nodos de instancia. Para la consulta SQL,

```
SELECT EXTRACTVALUE('/Order/Customer') AS "Customer Name",
EXTRACTVALUE(‘Order/OrderDate') as "Date",
FROM xml_Orders;
```

la tabla resultante tiene una columna VARCHAR2 para `Customer Name` y una columna DATE para la fecha de la orden, con base en los tipos de los datos XML subyacentes.

La función EXTRACT funciona en forma similar, excepto que devuelve un tipo XML. Por ejemplo,

```
SELECT EXTRACT ('/Customer/Name') AS "Customer Name"
FROM xml_Customers;
```

produce una tabla con datos XML.

Customer Name
<Name>WorldWide Travel Agency</Name>
<Name>Mary Jones</Name>
<Name>Alice Adams</Name>

13.5.2 Publicación de bases de datos relacionales en formato XML

Las bases de datos heterogéneas existentes se pueden consultar con el uso de lenguajes estándar tales como SQL, y los resultados de la consulta se colocan en un documento de instancia XML. El lenguaje de la consulta debe tener la posibilidad de etiquetar y estructurar datos relacionales en un formato XML. El esquema de base de datos relacional existente se utiliza para determinar los elementos XML y/o atributos. Por ejemplo, para extraer datos de la tabla Student de la base de datos relacional University, con el empleo de un Servidor Microsoft SQL se escribiría el enunciado SQL que sigue:

```
SELECT *
FROM Student
   FOR XML RAW;
```

El documento XML resultante aparece como se ilustra en la figura 13.13.

La cláusula FOR XML RAW hace que cada columna de la tabla relacional se convierta en un atributo del documento XML (Nota: los atributos XML difieren de los elementos XML y sólo tienen valores únicos). Si se desea mapear a elementos en lugar de atributos se escribe:

```
SELECT *
FROM Student
FOR XML AUTO, ELEMENTS;
```

que produce un documento como el de la figura 13.14. Con el uso de FOR XML EXPLICIT también es posible mapear ciertas columnas a los elementos y otras a los atributos.

```
<Student stuId="S1001"lastName="Smith     "firstname="Tom    "major="History"credits=90/>
<Student stuId="S1002"lastName="Chin      "firstname="Ann    "major="Math   "credits=36/>
<Student stuId="S1005"lastName="Lee       "firstname="Perry  "major="History"credits=3/>
<Student stuId="S1010"lastName="Burns     "firstname="Edward"major="Art     "credits=63/>
<Student stuId="S1013"lastName="McCarthy"firstname="Owen  "major="Math   "credits=0/>
<Student stuId="S1015"lastName="Jones     "firstname="Mary   "major="Math   "credits=42/>
<Student stuId="S1020"lastName="Rivera    "firstname="Jane   "major="CSC    "credits=15/>
```

FIGURA 13.13

Resultado de la consulta SQL SELECT*FROM Student FOR XML RAW;

FIGURA 13.14

Resultado de la consulta SQL SELECT*FROM Student FOR XML AUTO, ELEMENTS;

```
<Student><stuId>S1001</stuId><lastName>Smith</lastName><firstName>Tom</firstName>
<major>History</major><credits>90</credits></Student>
<Student><stuId>S1002</stuId><lastName>Chin</lastName><firstName>Ann</firstName>
<major>Math</major><credits>36</credits></Student>
<Student><stuId>S1005/stuId><lastName>Lee</lastName><firstName>Perry</firstName>
<major>History</major><credits>3</credits></Student>
<Student><stuId>S1010</stuId><lastName>Burns</lastName><firstName>Edward</firstName>
<major>Art</major><credits>63</credits></Student>
<Student><stuId>S1013</stuId><lastName>McCarthy</lastName><firstName>Owen</firstName>
<major>Math</major><credits>0</credits></Student>
<Student><stuId>S1015</stuId><lastName>Jones</lastName><firstName>Mary</firstName>
<major>Math</major><credits>42</credits></Student>
<Student><stuId>S1020</stuId><lastName>Rivera</lastName><firstName>Jane</firstName>
<major>CSC</major><credits>15</credits></Student>
```

13.6 Resumen del capítulo

La World Wide Web puede ser vista como un recurso de información poco organizada. Muchos sitos Web tienen archivos HTML estáticos vinculados, que es fácil que se hagan inconsistentes y caducos. Ciertas organizaciones permiten el acceso dinámico a sus bases de datos directamente desde la Web. La presencia del comercio electrónico ha impulsado a las organizaciones a desarrollar aplicaciones de bases de datos basadas en Web para crear mercados mundiales, distribuir información, dar un mejor servicio al cliente, comunicarse con sus proveedores, capacitar a sus empleados, expandir el sitio de trabajo, y muchas otras actividades innovadoras. XML es un estándar prometedor para almacenar, intercambiar y recuperar documentos, que integra tecnologías que compiten entre sí.

Internet se desarrolló a partir de **Arpanet**, red de comunicaciones creada en la década de 1960 con fondos de la agencia estadounidense DARPA, con el propósito de vincular instituciones gubernamentales y académicas. Utiliza un protocolo común, **TCP/IP**, para facilitar las comunicaciones entre sitios. La US National Science Foundation tomó la administración de la red, que a partir de entonces se conoce como **Internet**. La navegación y el uso de Internet requerían conocimientos considerables de parte del usuario. En 1989, Tim Berners-Lee propuso un método para simplificar el acceso a los recursos de Internet, lo que llevó al desarrollo de la **World Wide Web**. Su propuesta incluía los conceptos de **URL**, **HTTP**, **HTML**, **hipertexto** y **navegadores gráficos** con **vínculos** incrustados. La propuesta de Berners-Lee hizo posible automatizar el complicado proceso de encontrar, descargar y mostrar archivos en Internet.

HTTP es un protocolo **carente de estado** sin capacidad para recordar interacciones previas, lo que complica el comercio electrónico, que requiere una sesión continua con el usuario.

HTML es una forma de datos que se usa para presentar contenido en Internet. Se llama lenguaje de **marcado** porque los documentos HTML contienen marcas o etiquetas que dan información de formato para el texto. Un documento HTML contiene applets de Java, archivos de audio, imágenes, archivos de video y contenido. Cuando el navegador del usuario recupera el documento, el texto se muestra de acuerdo con los comandos de formato, los applets se ejecutan, los archivos de audio suenan, etc., en la estación de trabajo del usuario. **XML**, o lenguaje de marcado extensible, fue creado en 1966 por World Wide Web Consortium (W3) XML Special Interest Group. XML permite a los usuarios definir su propio lenguaje de marcado, inclusive sus etiquetas que describen aspectos de los ítems de datos en

los documentos, incluidas bases de datos. Con XML es posible definir la estructura de bases de datos heterogéneas y apoyar la traducción de datos entre bases diferentes.

Un **elemento** es el componente básico de un documento XML. Un documento contiene uno o más elementos XML, cada uno de los cuales tiene una **etiqueta de inicio** que muestra el nombre del elemento, algunos **datos de caracteres** y una **etiqueta final**. Los elementos pueden ser **subelementos** de otros elementos. Los subelementos deben anidarse en forma apropiada. Los elementos tienen **atributos** cuyos nombres y valores se muestran dentro de la etiqueta de inicio del elemento. Los atributos ocurren sólo una vez dentro de cada elemento, mientras que los subelementos ocurren cualquier número de veces. Un documento contiene **referencias a entidades** para hacer referencia a archivos externos. Se dice que un documento XML está **bien formado** si obedece las reglas de XML. Los usuarios definen su propio lenguaje de marcado con la escritura de una **definición de tipo documento** (DTD) o de un **esquema XML**. Una DTD es una especificación para un conjunto de reglas para los elementos, atributos y entidades de un documento. Se dice que un documento de instancia que obedece las reglas de su DTD asociada es de **tipo válido**. En varios campos se han desarrollado DTD de dominio específico y es probable que en el futuro se desarrollen más. **Esquema XML** es una forma nueva y más poderosa para describir la estructura de documentos diferentes de DTD. Permite una estructura más compleja, tipos adicionales de datos fundamentales, tipos de datos definidos por el usuario, vocabulario de dominio (nombres de espacios) creado por éste, y también apoya la unicidad y restricciones de claves foráneas. Un esquema XML define la organización y tipos de datos de una estructura XML. Si un documento de instancia se apega a un esquema XML, éste se conoce como **válido por esquema**. Un esquema XML lista elementos y atributos. Los elementos son complejos, lo que significa que tienen subelementos, o simples. Los atributos o elementos se utilizan para almacenar valores de los datos. Los atributos se emplean para valores simples que no se repitan, mientras que los elementos son complejos o simples y ocurren múltiples veces.

En un ambiente de Internet se requieren tres funciones principales, que son la presentación, la lógica de la aplicación y la administración de datos. La colocación de estas funciones depende de la arquitectura del sistema. Las **arquitecturas de triple capa** separan por completo la lógica de la aplicación de la administración de los datos. El cliente maneja la interfaz de usuario, que se llama **capa de presentación** o primera capa. Un **servidor de aplicaciones** separado ejecuta la lógica de la aplicación y constituye la **capa intermedia**. El **servidor de la base de datos** es la tercera capa**.** La red de comunicaciones conecta cada capa con la siguiente. La arquitectura de triple capa soporta a **clientes ligeros** que sólo necesiten manejar la capa de presentación, independencia de capas que usen diferentes plataformas, mantenimiento más fácil de la aplicación en el servidor de aplicaciones, acceso transparente a los datos a fuentes de datos heterogéneas y escalabilidad.

Es frecuente que en la capa de presentación se utilicen **formas HTML**. En HTML puede estar incrustado código en Perl, JavaScript, JScript, VBScript y otros lenguajes similares de escritura, a fin de dar cierto procesamiento por parte del cliente. Las **hojas de estilo** especifican la forma en que se presentan los datos en equipos específicos. El servidor de aplicaciones en la capa intermedia es el responsable de ejecutar las aplicaciones. Determina el flujo del control, adquiere datos de entrada desde la capa de presentación, hace solicitudes al servidor de la base de datos, acepta resultados de consultas desde la capa de la base de datos, y las usa para ensamblar páginas HTML generadas dinámicamente. El procesamiento por parte del servidor involucra muchas tecnologías diferentes tales como Servlets de Java, páginas de Servidor en Java, entre otras. La Interfaz Común Gateway, **CGI**, se utiliza para conectar formas HTML con programas de aplicación. A fin de mantener el estado durante una sesión, los servidores usan cookies, campos ocultos en formas HTML y extensiones URI. Las cookies se generan y envían al cliente con facilidad en la capa intermedia con el uso de la clase Cookie de Java, donde se almacenan en la memoria caché del navegador.

XML permite que los documentos se marquen con metadatos, e incluyen etiquetas que dan cierta estructura a los documentos. El **modelo de datos semiestructurado** es un método para describir dicha estructura parcial. Un modelo semiestructurado contiene un conjunto de nodos, cada uno con datos, posiblemente con distintos esquemas. El modelo usa un grafo consistente en nodos y arcos. Cada documento se representa como un **árbol** que tiene un solo nodo **raíz** y una secuencia de nodos hijos. Cada nodo hijo tiene otros nodos **hijos** que a su vez tienen hijos, y así sucesivamente. Una regla estricta para los árboles es que cada nodo, excepto el raíz, tenga exactamente un nodo **padre**, que está en el nivel superior del árbol. Los nodos representan ya sea **objetos complejos** o **valores atómicos**. Un **arco** representa bien la relación entre un objeto y su subobjeto o entre un objeto y su valor. Los nodos **hoja**, que no tienen subobjetos, representan valores. Los nodos del grafo para un documento XML estructurado están ordenados con el uso de un **preorden transversal**, que establece un orden de recorrido de arriba hacia abajo y de izquierda a derecha. No hay un esquema separado, puesto que el grafo es autodescriptivo.

XQuery es el lenguaje de consultas estándar actual W3C para datos de XML. Usa la estructura lógica abstracta de un documento según está codificado en XML, posiblemente en una definición DTD o esquema XML. Las consultas usan el concepto de **expresión de trayectoria** que proviene de un lenguaje anterior, **XPath**. Una expresión de trayectoria consiste por lo general en el nombre del documento y una especificación de los elementos por recuperar, con el uso de una relación de trayectoria. En una expresión de trayectoria es posible agregar condiciones a cualesquiera nodos. La expresión de trayectoria normalmente se evalúa con la lectura hacia delante del documento hasta encontrar un nodo del tipo y condición especificados. El usuario especifica un eje o dirección de búsqueda. XQuery usa un formato general llamado expresión **FLWOR**, para cláusulas FOR, LET, WHERE, ORDER BY y RETURN. Las expresiones permiten fijar las variables a los resultados, y también para iterar a través de los nodos de un documento. También permiten la realización de combinaciones y la restructuración de los datos. XQuery proporciona muchas funciones predefinidas, inclusive count, avg, max, min y sum, que se emplean en expresiones FLOWR.

Muchos proveedores de sistemas de administración de bases de datos relacionales han ampliado sus tipos de datos nativos para que permitan el almacenamiento de documentos XML. También es posible usar SQL con expresiones XPath para recuperar valores de la base de datos. Las bases de datos heterogéneas existentes se consultan con lenguajes estándar tales como SQL, y los resultados de la consulta se colocan en documentos de instancia XML. El lenguaje de la consulta debe tener la capacidad de etiquetar y estructurar datos relacionales en un formato XML.

Ejercicios

13.1 Defina los términos siguientes:

a. comercio electrónico (e-comercio)
b. World Wide Web
c. DARPA
d. hipertexto
e. Arpanet
f. TCP/IP
g. HTML
h. HTTP
i. URI

j. navegador
k. protocolo sin estado
l. SGML
m. XML
n. lenguaje de marcado
o. referencia de entidad
p. documento bien formado
q. documento de tipo válido
r. documento de esquema válido
s. W3C
t. cliente ligero
u. capa de presentación
v. hoja de estilo
w. cookie
x. modelo semiestructurado
y. XQuery
z. XPath

13.2 Explique lo que significa convergencia del procesamiento de la base de datos y procesamiento del documento.

13.3 Si un sitio Web usa archivos HTML estáticos vinculados para que muestren datos, ¿qué problemas pueden surgir?

13.4 ¿Para qué se usan las formas HTML en el comercio electrónico?

13.5 ¿Cómo se mantiene el estado durante una transacción larga de comercio electrónico?

13.6 ¿Cómo se relacionan HTML, SGML y XML?

13.7 ¿Cuál es el propósito de las hojas de estilo?

13.8 ¿Qué es lo que hace que un documento XML esté bien formado?

13.9 ¿Cuándo es de tipo válido un documento XML?

13.10 ¿Qué es lo que hace que un documento XML sea de esquema válido?

13.11 ¿Cómo se determina si un documento de esquema XML es de esquema válido?

13.12 Explique la diferencia entre un elemento y un atributo en un documento XML.

13.13 Cree un documento HTML similar al de la figura 13.1 para una lista de productos y pruébelo con el uso de un navegador.

13.14 Para la lista de productos del ejercicio 13.16 cree un documento XML bien formado, similar al que aparece en la figura 13.2.

13.15 Escriba una DTD para la lista de productos del ejercicio 13.16 y asegúrese de que el documento de instancia es de tipo válido para la DTD.

13.16 Escriba un esquema XML para la misma lista de productos.

13.17 Escriba un nuevo documento de instancia que liste productos y asegúrese de que la nueva lista es de esquema válido.

13.18 Dibuje un diagrama jerárquico para los datos de la figura 13.12.

13.19 Con el empleo de XQuery escriba dos consultas para los datos en la figura 13.12.

Ejercicios de laboratorio

Ejercicio de laboratorio 13.1: Navegación en un sitio Web de comercio electrónico que utilice una base de datos

Navegue en el sitio Web de una librería grande. Observe las características de la página inicial y las páginas vinculadas; conforme avance preste atención a las formas. Busque libros sobre un tema que le interese. Elija un libro que tal vez quiera ordenar y colóquelo en la canasta de compras. Regrese para buscar otros libros, escoja otro para agregarlo a la canasta de compras y observe cómo se modifica ésta. Comience el proceso de ordenar con cuidado de no finalizar la orden. Observe el número de interacciones requerida para el proceso de ordenar. Cancele la orden antes de terminar el proceso de compra.

Ejercicio de laboratorio 13.2: Creación de páginas Web que usen Access

Este ejercicio le dará práctica en la creación de páginas Web y una página inicial para la base de datos que usa Access. (Nota: si todavía no lo ha hecho, descargue la base de datos University desde el sitio Web de este libro, o créela en Access.)

1. Para crear una página Web para la tabla `Class` en Access, siga estos pasos:
   a. Abra la base de datos University y escoja el objeto *Pages* en el panel izquierdo, después *cree la página de acceso de datos con el uso del ayudante.*
   b. Escoja la tabla `Class` y seleccione todos sus campos, después haga clic en *Next*.
   c. No use niveles de agrupamiento (si se muestra un campo para agrupar, haga doble clic sobre el encabezado de su nombre para activar el agrupamiento), después haga clic en *Next*.
   d. No elija ordenar; sólo haga clic en *Next*.
   e. Escriba un nombre para su página: `Class`.
   f. Escoja *Modificar el diseño de la página* y escoja la aplicación de un tema. Haga clic en *Finish*.
   g. Mire los temas y escoja alguno como *Blends*.
   h. Aparecerá una forma en la que debe introducir un título, `Class Form`, en la parte superior. Debe ver cada campo y su etiqueta en el cuerpo del formato.
   i. Examine cada cuadro de texto para cerciorarse de que está asociado con la fuente correcta de datos. Seleccione el cuadro de texto (**no** lo etiquete), haga clic con el botón derecho, elija *Element Properties*, haga clic en el botón *Data*, después en *Control Source* y asegúrese de escoger el nombre del atributo correcto de la lista desplegable. Esto conecta a la página con el atributo correcto en la tabla correcta. Cierre la ventana desplegable.
   j. Si su página Web va a ser usada para recibir entradas, querrá utilizar listas desplegables para cualesquiera campos que tengan una lista conocida de valores posibles, para impedir que los usuarios cometan errores al introducir datos. Para la tabla `Class` ésta incluye `facId`.
      i. Haga clic en el cuadro de texto para `facId`, y oprima la tecla *Eliminar* (*Delete*) para quitarlo de la forma. Si la caja de herramientas no aparece, en el menú elija *View, Toolbox*.
      ii. Haga clic una vez en *Dropdown List* de la caja de herramientas y desplácela sobre el formato en que eliminó el campo. Esto sitúa la caja y una etiqueta por default en la forma y comienza un ayudante nuevo.

iii. Elija *I want the combo box to look up the values from a table or query.* Haga clic en *Next*. Escoja la tabla `Faculty`, *Next*, después seleccione el campo `facid` y después *Next*. Verá los valores `facId` de la tabla `Faculty`. Haga clic en *Next*. Para la etiqueta, que los usuarios verán más tarde, introduzca `Faculty ID`. Haga clic en *Finish*.

iv. Ahora tiene que asociar esta caja con la fuente correcta de los datos en la tabla `Class`, por lo que cada registro tendrá la `facId` correcta. Seleccione la caja desplegable, haga clic derecho, escoja *Element Properties*, haga clic en el botón *Data*, resalte *Control Source* y asegúrese de escoger el nombre del atributo `facId` de la lista desplegable. Cierre la ventana desplegable.

k. Guarde la página: haga clic en el ícono de página Web en la esquina superior izquierda, *Close*, e introduzca el nombre `Class`. Salga de la referencia absoluta cuando se pida y escoja *OK*.

l. Cuando se guarde la página elija *Pages* y vea la página `Class`. Note que los controles le permiten navegar por los registros, agregar otros nuevos, modificarlos o eliminarlos de la página. Agregue una clase nueva de registro, el número de grupo, escoja una `facId` de la lista desplegable y elabore un horario y asigne un aula. Guarde la página `Class` y ciérrela. Abra la tabla `Class` y vea que el nuevo registro aparece insertado.

m. Pruebe la página `Class` con el uso de un navegador. Abra Explorer, escoja *File, Open, Browse*, encuentre la página `Class` en el directorio donde la guardó y haga clic en *OK* para abrirla. Navegue por los registros. Agregue otro grupo nuevo con el navegador y guárdelo. Regrese a Access, vea la tabla y constate que el nuevo registro se encuentre ahí.

2. Cree una página Web `Student`, con los pasos a-i anteriores, esta vez con el uso de la tabla `Student`. Para hacer que el campo `major` sea una lista desplegable introduzca sus propios valores:

i. Haga clic en el cuadro de diálogo `major` y oprima la tecla *Delete*. De la caja de herramientas escoja un cuadro desplegable y arrástrelo.

ii. Haga clic en el cuadro de la lista desplegable de la caja de herramientas y arrástrelo al formato donde eliminó el campo.

iii. Escoja *I will type in the values that I want*, después haga clic en *Next*. Escriba los valores válidos para `major`: `History`, `Math`, `Art`, `CSC`. Haga clic en *Next*. Para la etiqueta (que los usuarios verán después) introduzca `major`. Haga clic en *Finish*.

iv. Ahora tiene que asociar esta caja con la fuente correcta de los datos en la tabla Student. Seleccione la caja desplegable, haga clic con el botón derecho, escoja *Element Properties*, haga clic en el botón *Data* y en *Control Source*, y asegúrese de elegir el nombre del atributo `major` de la lista desplegable.

v. Guarde la página: haga clic en el ícono de la página Web, *Close*, *Save Changes* e ingrese el nombre `Student`, respetando la referencia completa.

3. Cree la página Web `Faculty` de la misma manera, con la creación de una lista desplegable para el departamento en el que se introducirán valores como se hizo para el atributo `Student major`. Antes de guardar la página, asegúrese de conectar la caja combo con el departamento que es la fuente de los datos en la tabla `Faculty`.

4. Cree la página Web `Enroll`, esta vez para elegir valores de la caja combo para `stuId` a partir de la tabla `Student` y valores para `classNumber` de la tabla `Class`. Antes de guardarla, conecte cada caja combo con el atributo correspondiente en la tabla `Enroll` (seleccione la caja desplegable, haga clic con el botón derecho,

escoja *Element Properties*, haga clic en el botón *Data* y en *Control Source*, y escoja el nombre correcto del atributo).

5. Cree una página inicial para vincular las páginas que creó. Encontrará una página inicial de muestra para estos archivos en el archivo de notas llamado `Home.txt` en el sitio Web de este libro. Copie y guarde el archivo como `Home.htm`, con objeto de crear su propia página inicial en el mismo directorio que las páginas Access. Si utilizó nombres diferentes para sus páginas, modifique el archivo para que haga referencia a ellos.
6. Para probar el sistema, abra Explorer, escoja *File*, *Open*, *Browse*, encuentre la página `Home` en el directorio donde lo guardó y ábralo. Haga clic en cada botón para probar las conexiones. Navegue a través de los registros y agregue algunos nuevos a cada tabla a fin de probar las formas.

PROYECTO DE MUESTRA: CREACIÓN DE UN SITIO WEB QUE USE ACCESS, PARA LA GALERÍA DE ARTE

- Etapa 13.1. Para cada una de las tablas diseñadas para el modelo relacional se crea una tabla Access por medio de ejecutar los enunciados SQL DDL. Se introducen los datos con el uso de comandos SQL INSERT. Se crea una página Web para cada tabla, con el empleo de listas desplegables donde sea apropiado, siguiendo la etapa ilustrada en el ejercicio de laboratorio 2. Se diseña una página inicial sencilla que vincula cada una de estas páginas. El sitio Web para este libro contiene los archivos para este proyecto.

PROYECTOS ESTUDIANTILES: CREACIÓN DE UN SITIO WEB PARA LOS PROYECTOS ESTUDIANTILES

- Etapa 13.1. Si no lo ha hecho cree una base de datos Access para su proyecto, con la ejecución de comandos SQL DDL para crear las tablas, o con la interfaz de usuario de Access para crear tablas en vista de diseño, como se ilustró en el ejercicio de laboratorio 1.2 del capítulo 1. Inserte registros con el uso de comandos SQL Insert de la forma de hoja de cálculo en Access. Cree una página Web para cada una de las tablas que haya creado, como se demostró en el ejercicio de laboratorio 13.2. Cree una página inicial para vincular cada una de estas páginas por medio de modificar el archivo de Notas `Home.txt` provisto en el sitio Web de este libro. Pruebe su diseño por medio de desplegar y actualizar las páginas en un navegador.

CAPÍTULO 14

Aspectos sociales y éticos

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- El significado de la propiedad intelectual
- Cuáles formas de protección existen para la propiedad intelectual
- El significado del derecho de autor y cómo funciona
- La historia de las leyes del derecho de autor
- En qué difieren las leyes del derecho de autor de Estados Unidos y las internacionales
- Cómo se usan las patentes
- Cómo se utilizan las patentes en el ámbito internacional
- Qué constituye un secreto comercial
- Cómo están protegidos los secretos comerciales
- Qué constituye una marca registrada o una marca de servicio
- Cómo están protegidas las marcas registradas
- Cómo se puede proteger el software
- Cómo se relacionan la privacidad y la seguridad
- La historia y estado de la legislación sobre privacidad en Estados Unidos
- La historia y estado de la legislación sobre privacidad en Europa

- Cómo afectan al comercio las diferencias entre las leyes de la privacidad de Estados Unidos y Europa
- Cómo se toman en cuenta los factores humanos en el desarrollo de software
- Factores en la interfaz humano-base de datos
- La importancia de la usabilidad
- Algunas normas éticas para los profesionales de la computación

14.1 Computarización y aspectos éticos

El efecto de la computarización en la sociedad se describe con frecuencia como una revolución que sobrepasa a la revolución industrial. Igual que otras tecnologías emergentes, la computarización ha creado nuevas situaciones y posibilidades que desafían los códigos morales de la sociedad. Se requiere tiempo y reflexión para que la sociedad desarrolle valores, principios morales y códigos para relacionarse con aspectos creados por cualquier tecnología nueva. Debido a que la tecnología de la computación avanza demasiado rápido, los productos llegan al mercado antes de que la sociedad tenga tiempo para considerar los aspectos morales que plantea y reformule los valores y principios morales que guíen su uso. Dado el vacío que se genera por ese retraso, los individuos formulan sus propias reglas de comportamiento, con frecuencia sin una consideración adecuada de las implicaciones de sus actos. La tecnología de la computación brinda nuevas tentaciones debido a siete factores, de acuerdo con Richard Rubin ("Moral Distancing and the Use of Information Technology: The Seven Temptations". En Kizza, J. M. *Ethical and Social Issues in the Information Age,* 2a. ed., Nueva York: Springer 2003, pp. 58-59). Los siete factores son los siguientes:

1. **Velocidad.** Como las acciones se ejecutan con mucha rapidez, las personas piensan que las posibilidades de que las atrapen son pocas y que por ello pueden llevar a cabo acciones carentes de ética.
2. **Privacidad y anonimato.** Tener computadoras en lugares con menos visibilidad para el público, tales como los hogares y software que garantiza el anonimato, crean un ambiente tentador para realizar acciones sin ética.
3. **Naturaleza del medio.** El poder copiar datos sin afectar el original genera pocas o ninguna sospecha y facilita las acciones faltas de ética.
4. **Atracción estética.** La tecnología nueva ofrece productos que dan una sensación de realización a la persona que los utiliza. El sentimiento de éxito y logro son incentivos para aquellos que entran a un sistema de cómputo.
5. **Mayor disponibilidad de víctimas potenciales.** Las redes de computadoras hacen posible que un delincuente llegue a un número sin precedente de personas.
6. **Alcance internacional.** Internet hace posible que la gente trate de violar las leyes de su propio país y cometa actos ilegales en otro.
7. **El poder de destrucción.** Las computadoras parecen dar un enorme poder invisible a los usuarios. Lo que hace que sea una tentación para la gente ver cuánto daño puede hacer.

Las reglas de la ley que se aplican en las sociedades bien ordenadas están siendo cuestionadas por Internet. Al tratarse de una entidad global, ningún país puede aplicar sus leyes a todo el Internet. En el ciberespacio se valora mucho la individualidad, y la comunidad de Internet es muy renuente a los intentos de los gobiernos para controlarla. La sociedad debe llegar a algún acuerdo acerca del balance del control entre los estados y los individuos. Internet da un foro para que todas las personas se expresen gratis y alcancen a una audiencia vasta, y ejerzan su derecho humano de libertad de expresión. Sin embargo, si no hay algunos controles sobre el contenido, proliferan abusos tales como la pornografía y la retórica de odio. Otra área de desacuerdo es el modo en que las leyes de la propiedad intelectual y protecciones tales como derechos de autor, patentes, secretos comerciales y marcas registradas, se adaptan al ambiente cambiante de la era digital. Por un lado, están aquellos que creen que todo uso de las obras protegidas por derechos de autor, inclusive su transmisión a las computadoras para uso personal y almacenamiento en archivos temporales de Internet,

está sujeto a restricciones del derecho de autor. Por otro lado, hay quienes creen que toda la información debe ser libre y que la propiedad intelectual no se aplica al ciberespacio. El caso de entrar a otros sitios Web también genera controversias. Por ejemplo, si un vínculo lleva al navegador a una página Web con material protegido por leyes de derecho de autor, ¿se viola la ley de derechos de autor? Si un vínculo salta otra página inicial y lleva al usuario a una página interna, con lo que se evitan los avisos que tal vez haya en la página inicial, ¿se violan los derechos del propietario del sitio, ya que disminuye el valor del espacio para publicidad?

El aspecto de la protección de la propiedad intelectual del software también se debate mucho. Por lo general el software comercial está protegido por derechos de autor y otros medios. La justificación para esa protección se basa en conceptos de justicia y el deseo de estimular el espíritu empresarial. Por otro lado, existe un movimiento de "software libre" muy fuerte que apoya el concepto de que todo el software debe ser gratuito y poder copiarse libremente. La Free Software Foundation and the Open Source son movimientos que desafían la protección del software con bases filosóficas y prácticas. Richard Stallman, presidente de la Free Software Foundation ha escrito y hablado mucho sobre las razones por las que el software no debe compararse con la propiedad, y por qué la propiedad de los programas carece de ética. El libro de Eric Raymond (*The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly and Associates, 2001) hace una analogía que usan los defensores de la fuente abierta. Explica que cuando se dispone del código fuente del software, los programadores independientes y los *hackers* que trabajen con éste pueden desarrollar un software mejor que el que elabora el grupo selecto de programadores bien pagados de una compañía de software. La "catedral" representa una compañía de software como Microsoft que da empleo al equipo profesional, mientras que el "bazar" representa al grupo con poca cohesión de programadores independientes. El argumento es que el enfoque del bazar es mejor porque la inteligencia colectiva de los independientes se usa para corregir errores y mejorar al software con más rapidez. La privacidad es otro tema controvertido. La mayor parte de países tienen leyes que protegen la privacidad de los individuos de la intrusión por parte del gobierno y las empresas privadas, sobre la base de que la privacidad es un derecho humano fundamental. La privacidad individual se ve cuestionada por el amplio uso de tecnología de cómputo para compartir información personal que se almacena en bases de datos.

Los oponentes de la legislación sobre la privacidad alegan que compartir los datos de los clientes entre las empresas es una forma de libertad de expresión que debe protegerse. Lo comparan con el compartimiento tradicional de información entre los negocios de las comunidades. Este compartimiento, que ya existía antes de la computarización, es calificado como chismes. Como los datos de las bases de datos son más exactos y menos personales que los chismes, que no están regulados, afirman que la restricción para compartir los datos de las bases de datos por parte de las empresas no está justificada. También se debate la responsabilidad que tienen los profesionales de la computación para hacer que sus productos sean más sencillos y seguros para los usuarios, y el mejor modo de hacerlo.

14.2 Propiedad intelectual

La cuestión de quién es dueño de la propiedad intelectual y cuáles derechos tiene sobre ella, incluso el software y las bases de datos, es compleja y controvertida. Los derechos y obligaciones de los creadores o dueños de los productos intelectuales son tema de debate en los foros públicos, en especial en Internet. En la mayor parte de los países existen legislaciones que protegen los derechos de propiedad intelectual, y se han celebrado acuerdos internacionales para tratar de garantizar estos derechos en todo el mundo.

14.2.1 Definición de propiedad intelectual

La propiedad privada es un concepto fundamental en la mayoría de sociedades bien ordenadas. Los temas legales dedican mucha atención a los derechos y obligaciones de los propietarios de los dueños de la propiedad. Sin embargo, no es fácil definir la propiedad. El concepto de poseer se relaciona mucho con la noción de propiedad. Sir William Blackstone describió los orígenes y naturaleza de los derechos de los dueños de la propiedad en el siglo XVIII. Escribió lo siguiente:

> No hay nada que por lo general despierte más la imaginación y los afectos de la humanidad, que el derecho a la propiedad; o que el dominio único y déspota que alguien ejerza sobre las cosas materiales del mundo, con la total exclusión del derecho de cualquier otro individuo en el universo (*Commentaries on the Laws of England,* 1765-1769).

Al principio, la propiedad privada se refería a la tierra. Blackstone listó entre los derechos de un propietario de tierra el usarlo como pueda, recibir ingresos por ella, transferir la propiedad y expulsar a los demás de ella, y este uso se llama **paquete de Blackstone**. El concepto de propiedad se amplió hasta llegar a ser el de "activos tangibles sobre los que alguien tiene el control" (McFarland, Michael, "Intellectual Property, Information and the Common Good", *en Proceedings of the Fourth Annual Ethics and Technology Conference*, 1999, pp. 88-95). Una persona que posea un trozo de tierra, una cuenta en el banco, una joya o un carro, lo puede usar, decidir quién lo usa y transferir su propiedad, en tanto no interfiera con la seguridad o propiedades de otras personas.

En comparación con la propiedad tangible, la propiedad intelectual es más difícil de definir y los derechos asociados con ella son más difíciles de identificar. A diferencia de la tierra, la propiedad intelectual por lo general es intangible. La propiedad intelectual se refiere a los productos de la mente (como la literatura, música, arte, arquitectura, inventos, fórmulas para hacer productos y software). Una definición amplia es: "los derechos legales que resulten de la actividad intelectual en los campos industrial, científico, literario y artístico" (World Intellectual Property Organization, *WIPO Intellectual Property Handbook: Policy, Law and Use*, p. 3).

En el caso de la propiedad tangible, como un terreno o automóvil, su uso es exclusivo, puesto que sólo una persona los puede emplear a la vez. La propiedad intelectual difiere porque no es exclusiva en el sentido de que muchas personas la pueden emplear al mismo tiempo y su uso simultáneo no disminuye su valor. El costo de dar acceso adicional por lo general es bajo, aun cuando el costo de producirla originalmente haya sido alto. El concepto de justicia está en el fundamento de los derechos de la propiedad intelectual. Como el creador o inventor de un trabajo original ha invertido su tiempo y recursos en su creación, es justo que reciba las recompensas. Al garantizar un rendimiento por su trabajo, la sociedad estimula a los autores y otras personas creativas a que sigan desarrollando obras nuevas. En el caso de un escritor de software, diseñador o desarrollador de bases de datos, el individuo ha dedicado dinero y esfuerzo tanto para adquirir los conocimientos que le permitan realizar esas tareas como en la creación del producto. El individuo tiene derecho a cierto rendimiento. Si la persona es un empleado que creó el trabajo bajo contrato es su empleador quien debe percibir la utilidad.

14.2.2 Protecciones legales para la propiedad intelectual

Los derechos de los propietarios de la propiedad intelectual están protegidos por varios mecanismos, inclusive los derechos de autor, patentes, secreto comercial, marcas registradas y marcas de servicios.

14.2.2.1 Significado del derecho de autor (copyright)

El derecho de autor se da al creador de una obra literaria o artística. El derecho de autor lo da el gobierno del país de origen por un periodo limitado. Durante ese tiempo, sólo el autor de la obra tiene derechos exclusivos de hacer copias, publicar o vender el trabajo. De acuerdo con la World Intellectual Property Organization (WIPO), los siguientes tipos de obra están amparados por el derecho de autor:

> . . . obras literarias tales como novelas, canciones, poemas, obras de referencia, periódicos y programas de cómputo; bases de datos, películas, composiciones musicales y coreografía; obras artísticas tales como pinturas, dibujos, fotografías y esculturas; arquitectura; y anuncios, mapas y dibujos técnicos (WIPO, http://www.wipo.org/about-ip/en/).

Para ser susceptible de ser protegida por el derecho de autor, la obra debe ser original, tener norma tangible y estar fija en un medio, condiciones que por lo general se conocen como **originalidad**, **expresión** y **fijación**. Quien sea que produzca una obra que pueda ser protegida por el derecho de autor, de manera automática se vuelve titular de este derecho. No es necesario que se publique la obra para usar el símbolo del derecho de autor, ©, en ella, o hacer una solicitud formal para el derecho de autor. Es automático. Por tanto, cada página Web, programa de software, base de datos, papel, etc., que sea original y esté fija en un medio (por ejemplo, papel, disco, servidor) es automáticamente protegida por el derecho de autor. El *copyright* protege el derecho del propietario para hacer copias de la obra y usarla como la base de un trabajo nuevo (llamado trabajo **derivado**), para distribuirla o publicarla, para mostrarla en público y ejecutarla públicamente. La publicación de la obra incluye su colocación en Internet. Distribuir el trabajo incluye la distribución electrónica. Mostrar públicamente la obra incluye su proyección, su aparición en una pantalla de computadora. La realización pública de la obra incluye la ejecución en público de una grabación. El propietario tiene derechos exclusivos en esas áreas, con pequeñas excepciones que se encuentran en una categoría llamada **uso justo**, como son las revisiones de la obra. Cualquiera que desee emplear la obra protegida debe obtener permiso del dueño de los derechos de autor, y quizá pagar una tarifa, a menos que su uso sea justo.

Las obras protegidas por los derechos de autor siempre se pueden usar con libertad para uso personal, como la lectura. El uso justo de la obra protegida también permite el uso limitado no personal de los materiales protegidos sin permiso del autor. Esta área incluye el empleo de una cantidad pequeña del material para fines educativos, citas en revistas, parodias u otros usos limitados. Con frecuencia es difícil determinar si el uso de un trabajo se sale de la categoría de uso justo. Hay cuatro factores que se deben considerar cuando se tome una decisión. El criterio requiere que se cumplan los cuatro factores siguientes para que se trate de uso justo:

1. **El carácter del uso.** Si el uso es para revisión, parodia, comentarios, reportajes o propósitos similares, y es personal, educativo o no lucrativo, es más probable que se considere uso justo. Si el uso es comercial, lo más probable es que no sea uso justo.
2. **La naturaleza del trabajo por usar.** Si el trabajo es real y está publicado, es más probable que se trate de un uso justo, mientras que si es imaginativo y no está publicado, no sería raro que su uso no fuera justo**.**
3. **La cantidad del trabajo por usar.** Es probable que una pequeña cantidad sea de uso justo, mientras que una parte grande del trabajo no lo sea.
4. **El efecto que tendría en el mercado del trabajo original o sus permisos, si la clase de uso fuera muy amplia.** Este factor debe considerarse sólo después de evaluar los tres factores anteriores. Si el uso proviene de ventas del trabajo original o evita el pago de regalías por la obra, no es un uso justo. Sin embargo, si el original no se halla

disponible para su compra o si el propietario del derecho de autor no puede ser identificado, tal vez sea un uso justo.

Algunas obras intelectuales no están protegidas por el derecho de autor y se pueden usar con toda libertad. La lista siguiente incluye trabajos que se consideran utilitarios o triviales.

- Títulos, nombres, frases breves y lemas; símbolos o dibujos conocidos; meras variaciones de ornamentos tipográficos, letras o colores; listas de ingredientes o contenidos.
- Ideas, procedimientos, métodos, sistemas, procesos, conceptos, principios, descubrimientos o dispositivos (distintos de una descripción, explicación o ilustración).
- Trabajos que consisten por completo en información que es de propiedad común y que no tienen un autor original (por ejemplo: calendarios estándares, tablas de estatura y peso, cintas métricas y reglas, y listas o tablas tomadas de documentos públicos o de otras fuentes comunes) (U.S. Copyright Office, *Copyright Basics, Circular 1*).

Los hechos son siempre de dominio público, pero un arreglo y expresión específicos de ellos como el que se encuentra en una base de datos, puede estar protegido por el derecho de autor. Las ideas también son de dominio público, pero una explicación particular de ellas, como en un ensayo, puede tener protección. Las fórmulas matemáticas y químicas son de dominio público, pero las fórmulas de los productos se pueden proteger. Algunas de las demás excepciones, como los símbolos, son susceptibles de protegerse de acuerdo con leyes de la marca registrada, y otros, como los procesos, por medio de patentes o protección del secreto comercial.

Otros trabajos del dominio público incluyen aquellos cuyos derechos de autor han expirado y no se han renovado, en los que está perdido este derecho, o aquellos en que el autor hizo de dominio público intencionalmente. Los trabajos en el dominio público pueden ser usados con libertad por cualquiera sin permiso.

14.2.2.2 Las leyes del derecho de autor internacional y las de Estados Unidos

Los derechos de autor tienen reconocimiento internacional, pero cada país elabora sus propias leyes. Estados Unidos, Inglaterra y Francia tenían originalmente leyes de derechos de autor que se remontaban al siglo XVIII. La mayor parte de ellas se basan en la **Convención de Berna** para la protección de obras literarias y artísticas, que se reunió por primera vez en 1886 a instancias de Víctor Hugo. La Convención de Berna tuvo revisiones en 1896, 1908, 1914, 1928, 1967, 1971 y 1979. El propósito de la convención era garantizar que cada país reconociera derechos de autor de obras publicadas en otros países miembros. La convención no requería que la obra se registrara o que mostrara el símbolo del copyright. Establece la duración del derecho como la vida del autor más 50 años. Aunque la Convención de Berna fija lineamientos, cada país miembro hizo sus propias leyes para apegarse a éstos. La Convención de Berna es administrada por la World Intellectual Property Organization (**WIPO**), que se fundó en 1967.

Estados Unidos originalmente no firmó la Convención de Berna porque sus leyes de protección al derecho de autor difieren de los lineamientos de ésta. En 1952, la UNESCO fundó la Universal Copyright Convention (UCC), Convención Universal de los Derechos de Autor, que fue firmada en Génova por países que no suscribían la Convención de Berna, inclusive Estados Unidos, la Unión Soviética y la mayor parte de los países de América Latina. Con los estándares menos estrictos de la UCC, la protección del derecho de autor no era automática, y países miembros como Estados Unidos requerían que la obra estuviera publicada, tuviera el símbolo del copyright y se registrara a fin de que estuviera protegida. Sin

embargo, Estados Unidos cambió sus leyes de derechos de autor en 1978 a una forma que sigue una norma más estricta y firmó la Convención de Berna en 1989. La mayor parte de países miembros de la UCC también firmaron la Convención de Berna, lo que hizo que la UCC fuera menos importante.

La Organización Mundial de Comercio (WTO) se estableció en Ginebra en 1995 para administrar el Trade Related Aspects of Intellectual Property Rights, Acuerdo de Aspectos Comerciales Relacionados con los Derechos de la Propiedad Intelectual (TRIPPS) al que se llegó en la ronda de Uruguay del GATT (General Agreement on Tariffs and Trade) en su protocolo de 1994. Tiene 146 países miembros. El resumen del acuerdo TRIPPS dice:

> Con respecto del derecho de autor se requiere que las partes cumplan las provisiones principales de la Convención de Berna sobre la protección de obras artísticas y literarias, en su última versión (París, 1971), . . . Garantiza que los programas de cómputo se protegerán como obras literarias de acuerdo con la Convención de Berna y las leyes que surgen de ésta en lo que respecta a las bases de datos, por lo que deben tener protección de los derechos de autor. Otras adiciones importantes a las reglas internacionales existentes en el área de los derechos de autor y otros parecidos son las cláusulas acerca de los derechos de renta. El borrador requiere que los autores de programas de cómputo y productores de grabaciones de sonido tengan el derecho de autorizar o prohibir la renta comercial de sus obras para el público (Organización Mundial de Comercio, *A Summary of the Final Act of the Uruguay Round*).

En el artículo 10, el acuerdo analiza específicamente los programas de cómputo y bases de datos.

1. Los programas de cómputo, ya sea en código fuente o como objeto, deben protegerse como obras literarias según la Convención de Berna (1971).
2. Las compilaciones de datos o de otros materiales, ya sea legibles en una máquina o en otra forma, que por razón de la selección o arreglo de su contenido constituyan creaciones intelectuales deben protegerse como tales. Dicha protección, que no debe extenderse a los datos o materiales en sí mismos debe ser sin perjuicio de cualesquiera derechos de autor existentes en los datos o el material en sí (Organización Mundial de Comercio, *Uruguay Round Agreement: TRIPS Part II: Standards concerning the availability, scope and use of Intellectual Property Rights,* artículo 10).

La protección de los derechos de autor en Estados Unidos tiene sus orígenes en su Constitución, cuyo artículo 1, sección 8, da al Congreso el poder

> De promover el progreso de la ciencia y las artes útiles, asegurando por tiempos limitados a los autores e inventores el derecho exclusivo a sus escritos y descubrimientos respectivos, . . .

La primera Ley de Derechos de Autor de Estados Unidos, aprobada por el Congreso para protegerlos, se aprobó en 1790. La ley tuvo enmiendas en 1831 y 1870. En 1909 fue adoptada el acta de derechos de autor y siguió vigente con pocas modificaciones, hasta 1978. La ley requería la publicación del trabajo y que el símbolo del copyright apareciera en éste para poder reclamar la protección de los derechos de autor. En 1976 el Congreso aprobó una nueva ley de derechos de autor que entró en efecto en 1978.

La oficina del copyright de Estados Unidos lista el tipo de obras que se pueden proteger con la ley actual:

> La protección de los derechos de autor ampara "los trabajos originales de un autor" que estén fijos en una forma tangible de expresión. La fijación no necesita ser perceptible en forma directa en tanto pueda comunicarse con ayuda de una máquina o dispositivo. Los trabajos susceptibles de protección incluyen las categorías siguientes:

1. Obras literarias
2. Obras musicales, inclusive cualesquiera palabras de acompañamiento
3. Obras de teatro, inclusive la música que las acompaña
4. Pantomimas y obras coreográficas
5. Pinturas, artes gráficas y esculturas
6. Películas y otras obras audiovisuales
7. Grabaciones de sonido
8. Obras arquitectónicas

Estas categorías deben verse con amplitud. Por ejemplo, los programas de computación y la mayor parte de "compilaciones" pueden estar registradas como "obras literarias"; los mapas y planos arquitectónicos tal vez se registren como "pinturas, artes gráficas y esculturas" (US Copyright Office, *Copyright Basics Circular 1*, http://www.copyright.gov/circs/circl.html#wwp).

Esta ley requiere de la originalidad, fijación y expresión, pero no necesita la publicación, el uso del símbolo del copyright, o el registro de la obra en la oficina de derechos de autor. Estos cambios ubicaron a las leyes estadounidenses de acuerdo con la Convención de Berna, por lo que Estados Unidos la firmó en 1989. Sin embargo, para el propósito de proteger y determinar los datos de la protección, la oficina del copyright aconseja que los autores registren la obra y que avisen que está protegida, para lo que debe incluir el símbolo o palabra o aviso del copyright, el nombre del autor y la fecha de publicación. Para completar el proceso de registro, el autor debe llenar un formato de registro, pagar una cuota y enviar dos copias del trabajo publicado a la oficina del copyright para su depósito en la biblioteca del Congreso. El registro permite que el autor reclame la protección de sus derechos en un tribunal en caso de violación. La oficina del copyright de Estados Unidos ofrece lineamientos para este registro.

En general, el registro del copyright es una formalidad legal que pretende hacer un registro público de los hechos básicos de un copyright particular. Sin embargo, el registro no es condición para la protección de los derechos. Aun cuando el registro no es requisito para la protección, la ley de derechos de autor provee varias formas o ventajas para estimular a que los dueños del derecho se registren. Entre estas ventajas están las siguientes:

- El registro establece una manifestación pública de la protección.
- Para que pueda perseguirse una violación ante un tribunal es necesario que las obras con origen en Estados Unidos estén registradas.
- Si se hace dentro de los cinco años de su publicación, el registro establece una evidencia *prima facie* en el tribunal de la validez del derecho de autor y de los hechos establecidos en el certificado.
- Si el registro se hace dentro de los tres meses posteriores a la publicación de la obra o antes de la violación de los derechos, los daños y honorarios del abogado establecidos serán para el dueño del derecho en las acciones del tribunal. De otro modo éste sólo podrá reclamar una compensación por los daños y utilidades reales.
- El registro permite que el dueño del derecho lo registre ante el Servicio de Aduanas de Estados Unidos para que impida la importación de las copias que lo perjudiquen (US Copyright Office, *Copyright Basics, Circular 1*).

El trabajo publicado debe depositarse en la Biblioteca del Congreso. En realidad, la ley requería que cualquier obra publicada en Estados Unidos se depositara en ésta dentro de los tres meses siguientes a su publicación, estuviera o no registrada. La Ley de los Derechos

de Autor de 1976 especificaba la duración del derecho. Para trabajos publicados antes de 1978, el derecho expiraba 75 años después de la fecha de emisión de la protección. Para los derechos recibidos después de 1978, la ley permanece vigente durante la vida del autor más 50 años. Las obras para contratar estaban protegidas durante 75 años a partir de la primera publicación del trabajo o 100 años después de la fecha de su creación. La ley tuvo varias enmiendas. Un cambio importante fue la Sonny Bono Copyright Term Extension Act (Ley de Extensión del Término del Derecho de Autor para Sonny Bono), que extendió la duración de todos los derechos de autor por 20 años, de modo que los trabajos creados antes de 1978 estaban protegidos por 95 años, los derechos para las obras posteriores a 1978 lo estaban durante la vida del autor más 70 años, y las obras para contratar de 95 o 120 años. En 1980 se hizo una enmienda para amparar al software, tanto en código fuente como en código objeto.

En 1998 el Congreso aprobó la **Digital Millenium Copyright Act (Ley de Protección de los Derechos de Autor Digitales del Milenio)**. Esta ley implementa el acuerdo TRIPPS aprobado por el GATT, así como dos acuerdos WIPO, el tratado de protección de los derechos de autor WIPO y el tratado de los fonogramas y obras de teatro del WIPO, por el que cada país miembro debe dar protección a las obras de otros países miembros en la misma forma que protege las de sus ciudadanos. Sin embargo, tiene algunos puntos controvertidos en relación con herramientas diseñadas para violar sistemas de protección o encriptado y cataloga como un delito utilizarlas.

> Ninguna persona debe violar la medida tecnológica que controle el acceso a una obra protegida por esta ley (Ley de Protección de los Derechos de Autor Digitales del Milenio, sección 1201, 1a).

También es un delito manufacturar u ofrecer una herramienta diseñada sobre todo con el propósito de violar las tecnologías de protección, tenga un uso limitado distinto de dicha violación, o se comercialice con fines de burlar alguna protección.

> Ninguna persona debe fabricar, importar, ofrecer al público, proveer, o traficar de cualquier modo alguna tecnología, producto, servicio, dispositivo, componente o parte que:
>
> A. Esté diseñada o producida, sobre todo con el propósito de burlar una medida tecnológica que controle efectivamente el acceso a un trabajo protegido por esta ley;
>
> B. Tenga un propósito limitado comercialmente significativo o use otra medida tecnológica que viole el control del acceso a una obra protegida de esta ley o,
>
> C. Sea comercializada por esa persona u otra que actúe con el conocimiento de aquélla para usarla en la violación de una medida tecnológica que controle el acceso de una obra protegida por esta ley (Ley de Protección de los Derechos de Autor Digitales del Milenio, sección 1201, 2a).

Las excepciones específicas se hacen para las bibliotecas, archivos e instituciones educativas, pero sólo con el propósito de evaluar el trabajo a fin de determinar si se adquiere una copia de él. También se hacen excepciones para las instituciones que buscan hacer cumplir las leyes, en circunstancias limitadas para hacer ingeniería inversa que permita la operación entre diferentes productos, para hacer investigaciones sobre encriptado, reparación de computadoras y otros usos similares. La ley también limita la responsabilidad de los proveedores de servicios de Internet por infringir los derechos de autor en línea por parte de suscriptores del servicio.

Infringir los derechos de autor significa que una persona utiliza o vende un artículo protegido por el copyright, o que da intencionalmente apoyo a otros para hacerlo, lo que viola los derechos de propiedad del dueño. El dueño del copyright tiene que demostrar que la persona no hizo un uso justo del ítem. Los tribunales por lo general quieren que el dueño de la

protección demuestre que es el propietario de la obra, que el violador tenía conocimiento de ésta y que la nueva obra producida no es sustancialmente distinta de la que se copió. El castigo es una multa muy elevada.

14.2.2.3 Patentes

Otra forma de protección de la propiedad intelectual son las patentes, que están diseñadas para proteger un invento o descubrimiento. Una patente es la garantía de un derecho de propiedad del inventor. Las patentes son emitidas por los gobiernos de países individuales, aunque hay algunos tratados internacionales al respecto. En Estados Unidos las patentes las emite la Patent and Trademark (Oficina de Patentes y Marcas), que es una oficina del Departamento de Comercio. Una patente da al inventor el derecho de impedir que otros fabriquen, utilicen, ofrezcan a la venta o vendan el invento en Estados Unidos o sus territorios y posesiones, o de importar el invento a dicho país. Corresponde al inventor ejercer su derecho de impedir estas actividades. Por lo general, el término de la patente es de 20 años desde el momento de la solicitud, o a partir de la fecha en que se hizo una solicitud relacionada anterior, aunque se puede extender en casos especiales. Durante este tiempo, el inventor tiene la oportunidad de recuperar los costos de crear el invento.

Las leyes de patentes en Estados Unidos, igual que las de los derechos de autor, tienen su origen en el artículo 2, sección 8 de la Constitución. El Congreso emitió la primera ley de patentes en Estados Unidos en 1790 y la revisó en 1952. En 1999 el Congreso aprobó la American Inventors Protection Act (AIPA), Ley de Protección de los Inventores Estadounidenses. De acuerdo con la ley,

> . . . cualquier persona que invente o descubra cualquier proceso, máquina, manufactura, o composición de la materia que sea nueva y útil, o cualquier mejora nueva y útil al respecto, puede obtener una patente sujeta a las condiciones y requerimientos de la ley. La palabra "proceso" es definida por la ley como un procedimiento, acto o método, e incluye sobre todo a los procesos industriales o técnicos (United States Patent and Trademark Office, Oficina de Patentes y Marcas de los Estados Unidos, *Información General Acerca de las Patentes*).

El invento debe ser **útil**, lo que significa que tiene algún propósito útil y funciona en verdad. No basta una idea o sugerencia de un dispositivo útil. Debe demostrarse que la creación funciona. El invento debe ser **nuevo**, lo que significa que no debe haber sido conocido, publicado o usado previamente. Debe ser **no obvio** para una persona que tenga conocimiento normal del área. El inventor debe llenar un formato que incluya las especificaciones y que describa al invento con todo detalle:

> La especificación debe incluir una descripción escrita del invento y de la manera y proceso de fabricarlo y usarlo, y se requiere que lo haga en términos exhaustivos, claros, concisos y exactos como para permitir que cualquier persona con habilidad en el área tecnológica a que pertenezca el invento, o con la que se relacione lo más posible, lo fabrique y utilice (United States Patent and Trademark Office, *Información General Acerca de las Patentes*).

También se requieren dibujos que describan el invento. La oficina de patentes revisa la solicitud y la envía a uno de sus centros tecnológicos a fin de determinar si el invento es nuevo, útil y no es obvio. Si la patente procede tiene efecto 20 años. Durante ese tiempo, otras personas pueden usar las ideas pero no el proceso de la patente. Al expirar ésta, cualquier persona puede estudiar los detalles y usar el invento.

Las patentes que se dan en un país tienen efecto legal automático en otro. Sin embargo, durante la **Convención de París** para la protección de la propiedad industrial, de la cual Estados Unidos es miembro, se firmó un tratado que establece que cada uno de los países

miembros garantiza dar a los ciudadanos de los otros países el mismo derecho a una patente (y marca registrada) que ofrece a sus propios ciudadanos. Además, si un inventor llena una solicitud regular para una primera patente en uno de los países miembros, él o ella pueden, dentro de un periodo especificado, solicitar la protección de la patente en todos los demás países miembros. Aun si la solicitud hubiera sido hecha en forma tardía se considerará que fue hecha en la misma fecha que la primera solicitud regular, con el fin de determinar el derecho de prioridad si otra persona solicita otra patente para el mismo invento. El tratado de cooperación de patentes de 1970 crea un formato de solicitud estándar y un procedimiento centralizado para la solicitud de patentes en todos los países miembros.

Los dueños de los inventos patentados normalmente inscriben el número de la patente en el producto. Las personas que están en el proceso de solicitud de un invento tienen permiso para escribir las palabras "patente pendiente" o "solicitud de patente" en el producto mientras se revisa la solicitud. Estas dos prácticas son métodos de notificar al público que existe una patente o existirá, y ayudan al inventor a identificar las violaciones al hacer que el público reporte cualquier sospecha de violación. Es responsabilidad del propietario de la patente identificar e investigar cualquier violación, y hacerlo es lento y costoso. Si el dueño de la patente demuestra que se infringió, los castigos son severos.

14.2.2.4 Secretos comerciales

Los secretos comerciales son otra forma de proteger la propiedad intelectual. Un secreto comercial es cualquier información que se use en la operación de un negocio que sea secreto y le dé una ventaja competitiva. Puede ser una fórmula, proceso, patrón, compilación, programa, método, técnica, dispositivo o cualquier otra información. La fórmula de un refresco, los ingredientes secretos de un cosmético y el proceso de manufactura de un producto distintivo son ejemplos de secretos comerciales. A diferencia de los derechos de autor y las patentes no hay manera de registrar un secreto comercial con ninguna ley nacional o internacional. En Estados Unidos, los secretos comerciales están protegidos por los estados, la mayor parte de los cuales han adoptado leyes basadas en el trabajo de la National Conference of Commissioners of Uniform State Laws, que aprobó la Uniform Trade Secrets Act en 1970 y la modificó en 1985. Estas leyes protegen a la compañía de la apropiación indebida del secreto y convierten en delito la adquisición ilegal de un secreto comercial. La protección dura mientras la información permanezca como un secreto. La amenaza más grande para un secreto comercial por lo general proviene del interior de la compañía. Los empleados actuales pueden revelar el secreto por accidente en pláticas informales en reuniones de negocios, conferencias profesionales o en conversaciones personales, o pueden ser sobornados por los competidores de su empresa para que lo revelen. Los empleados antiguos se llevan con ellos el conocimiento de los procesos de negocios de una compañía y otra información exclusiva, y pueden revelar esta información a sus nuevos empleadores. Por lo general las compañías protegen los secretos comerciales limitando el número de trabajadores que tienen acceso a ellos, con la custodia cuidadosa de los documentos que los contienen y a través de exigir a los empleados que acceden a ellos que firmen acuerdos de confidencialidad.

Es frecuente que cuando se revela un secreto comercial se origine un litigio. Los tribunales deben determinar si la información constituye un secreto comercial. De acuerdo con R. Mark Halligan ("The Sorry State of Trade Secret Protection" http://www.rmarkhalligan2.com/trade/default.asp) hay seis factores que los tribunales utilizan para determinar si algo es un secreto comercial.

1. La amplitud con que se conoce la información fuera de la compañía. Si la información es ampliamente conocida en el exterior de la empresa, es menos probable que se trate de un secreto comercial susceptible de ser protegido.

2. La amplitud con que conocen la información los empleados y otras personas en la compañía. Entre más trabajadores la conozcan, menos probable es que se trate de un secreto comercial.
3. El grado de las medidas que se toman para guardar en secreto la información. Entre más cuidado se ponga en guardar la información es más probable que se trate de un secreto comercial.
4. Lo valiosa que sea la información para el propietario y los competidores. A menos que tanto la compañía como sus competidores consideren que la información es valiosa, no será un secreto comercial.
5. El tiempo, esfuerzo y dinero en generar la información. Entre más haya gastado la compañía en el desarrollo de la información, más probable es que sea un secreto comercial.
6. Lo difícil que sea que otras personas adquieran o dupliquen la información. Si es fácil repetir la información es menos probable que sea un secreto comercial.

La apropiación indebida de secretos comerciales se remonta a los empleados actuales o antiguos que han tenido acceso al secreto y que lo revelaron. Si el propietario puede establecer que se trataba de un secreto con el uso de los factores ya descritos, y que el secreto se obtuvo por medios impropios tales como el soborno, entonces puede obtener una compensación. Hay multas muy elevadas por violar las leyes sobre el secreto comercial.

Como la protección con las leyes de patentes requiere la revelación total y la protección con las leyes del secreto comercial no implica que éste se revele, una compañía no puede reclamar un secreto comercial y a la vez solicitar una patente para la misma información.

14.2.2.5 Marcas registradas y marcas de servicio

Las marcas registradas y las marcas de servicio también son elegibles para invocar el amparo de las leyes de protección intelectual. Las marcas incluyen una o más letras, números o palabras. También son dibujos, símbolos, colores, sonidos, olores, o la forma o empaque de los bienes que resulten distintivos. De acuerdo con la United States Patent and Trademark Office,

- Una **marca registrada** es una palabra, frase, símbolo o diseño, o combinación de palabras, frases, símbolos o diseños, que identifique o distinga la fuente de los bienes de una parte de aquéllos de las demás.
- Una **marca de servicio** es lo mismo que una marca registrada, excepto que identifica y distingue a la fuente de un servicio en vez de a la de un producto (US Patent and Trademark Office, *¿Qué son las patentes, marcas registradas, marcas de servicio y derechos de autor*?).

En Estados Unidos las marcas registradas están protegidas tanto por leyes federales de acuerdo con la **Lanham Act** (Ley de Marcas Registradas de 1936) como por leyes estatales. No se requiere el registro de ellas. Una compañía puede establecer su derecho a una marca simplemente por usarla y la define como marca registrada con el uso del símbolo TM, o como una marca de servicio con el símbolo SM. Sin embargo, las marcas usadas en el comercio interestatal o extranjero pueden protegerse por medio del registro en la Patent and Trademak Office. El registro requiere la reproducción clara de la marca, inclusive cualesquiera aspectos tridimensionales, una muestra de ella en uso, la lista de bienes y servicios para las que se usará la marca, así como otros requerimientos. Después se hace una investigación para garantizar que la marca es apropiada y distintiva y no infringe otras marcas que estuvieran ya registradas. Una vez registrada, los dueños utilizan el símbolo de registro federal, ®, para designar la marca. El registro de la marca permite al propietario notificar al

público que la marca tiene dueño, establecer el derecho legal a la propiedad y el uso exclusivo de ella, permite que el propietario emprenda en los tribunales acciones contra la violación de la marca, le permite usarla como base de registro de la marca en países extranjeros, y le permite solicitar el registro en el servicio de aduanas de Estados Unidos para impedir que los bienes que violan su marca sean importados.

Para demostrar la violación, el dueño debe demostrar que es probable que los consumidores se confundan por el uso de la marca ilegal. El registro es válido por 10 años, pero se puede renovar. Muchos países han adoptado ya sea el **Protocolo de Madrid** o el Acuerdo de Madrid sobre el Registro Internacional de Marcas. WIPO administra un sistema de registro internacional de marcas con base en esos dos tratados. Más de 60 países constituyen la Unión de Madrid y han firmado uno o ambos acuerdos. Para registrar la marca, el propietario primero debe llenar una solicitud en la oficina de marcas de uno de los países miembros. Una vez registrada la marca en un país miembro, el dueño la usa como base para obtener un registro internacional válido en los países miembros de la Unión de Madrid. Aunque Estados Unidos no es miembro de ésta, el registro de una marca en la US Patent and Trademark Office también se utiliza como base para el registro internacional.

14.2.3 Protección de la propiedad intelectual del software

El software y su paquetería pueden protegerse por medio de las leyes de derechos de autor, patentes o secreto comercial, por la marca registrada o por alguna combinación de ellos. Aunque los algoritmos que se usan para crear software no pueden ser protegidos por los derechos de autor debido a que se trata de ideas, los programas de software en sí mismos sí pueden recibir protección porque están clasificados como expresiones. Tanto las leyes sobre derechos de autor de Estados Unidos como la Convención de Berna, mencionan específicamente al software como materia de la protección de derechos de autor. El software no tiene que haber sido publicado a fin de recibir protección. Tan pronto como se crea, el programa queda protegido y todos los derechos pertenecen al propietario, que puede ser el programador o un empleador, en el caso de trabajo por contrato. Por tanto, es ilegal para quien sea hacer una copia de cualquier programa, a menos que el dueño dé permiso para ello, excepto para el uso justo limitado ya descrito. La documentación que acompaña al software, inclusive todos los manuales, también está protegida por las leyes de derechos de autor, sin importar si tiene o no el símbolo del copyright. Excepto para el uso justo, también es ilegal copiar los manuales a menos que se cuente con permiso. Es frecuente que también haya una marca registrada asociada con el software. Es ilegal comercializar el software de un competidor usando una marca que sea tan similar a la registrada que los clientes se confundan al pensar que es el producto original. En ocasiones el software contiene un elemento de protección técnica para impedir su copiado. La anulación de dicho elemento o la creación de tecnología que facilite la violación a fin de eliminar la protección, infringen la DMCA en Estados Unidos y viola las normas de la Convención de Berna.

Un cliente que compre software en realidad compra una **licencia para usarlo**, no el software en sí, que continúa siendo propiedad del dueño. El cliente arrienda el software. El término del arrendamiento con frecuencia se enuncia en una pantalla de apertura al instalar el software y el cliente manifiesta su acuerdo con los términos antes de instalar el producto. Los términos usuales son que el cliente tiene el derecho de usar el software y de hacer una copia de respaldo, pero sólo una copia del programa se puede usar a la vez. En ese caso, es ilegal para el cliente ejecutar el respaldo en una segunda computadora al mismo tiempo que utiliza el original. Por supuesto que es ilegal y constituye piratería de software si el cliente da una copia de éste a otra persona para que lo use. No es raro que el registro del producto que usa un número de licencia único sea requerido por el vendedor, pero la protección se aplica aun si no se pide el registro. Un negocio u otra organización tienen la posibilidad de com-

prar una licencia para usar cierto número de copias de software a la vez. Si alguna vez se usa simultáneamente más del número de copias o licencia, la organización estará cometiendo piratería de software. La organización debe usar algún mecanismo para contar el número de copias en uso simultáneo a fin de impedir el abuso de la licencia. Algunas organizaciones compran sitios con licencia que permiten un número ilimitado de copias de software en cierto sitio. En ese caso será piratería de software permitir que éste se utilice en una segunda ubicación, como una sucursal de la empresa.

Cierta clase de software, conocido como de **open source** (**fuente abierta**), es gratuito y se puede copiar. Este tipo de software tiene mejoras, manuales o marcas registradas protegidas por las reglas de la propiedad intelectual. Aunque utilizar este tipo de software es legal, copiar cualquier producto relacionado protegido no lo es, por supuesto.

Otra categoría de software es el **shareware** (**software de evaluación**), que permite que las personas descarguen y usen el software con objeto de evaluarlo. Si quieren continuar el uso después del periodo de prueba, se pide que paguen por ello. En ciertos casos, el vendedor establece que aun cuando la prueba del software se descarga gratis, se necesita que el cliente se registre. Entonces es ilegal que el cliente dé una copia a otra persona, ya que ésta también debe registrarse. Algunos productos tienen elementos de protección que inhabilitan el software después del periodo de prueba o después de cierto número de usos. Sin embargo, aun si el software continúa en funcionamiento después de dicho periodo, el uso adicional viola el acuerdo que hizo el usuario al descargar el producto, por lo que será ilegal.

El software de un sistema de administración de base de datos comercial, como el que proviene de proveedores importantes, por lo general está protegido con el uso de los mismos métodos de protección de la propiedad intelectual que emplea otra clase de software. Las aplicaciones especiales que haya escrito una compañía para una base de datos, tales como una interfaz de usuario personalizada o reportes personalizados, también califican para las protecciones mencionadas del software. Las mismas reglas que se aplican a otro software se aplican a éste. Algunos proveedores importantes de DBMS permiten que los clientes descarguen una edición personal gratis de su software de la base de datos para su propio uso. Otros ofrecen la edición personal sólo durante un periodo de prueba limitado. Por lo general no se encuentran disponibles en forma gratuita las ediciones para una empresa.

Las bases de datos en sí mismas también califican para diferentes clases de protección de la propiedad intelectual. Por ejemplo, la base de datos puede protegerse según los derechos de autor porque se trata de una expresión, aun si los hechos contenidos en ella no pudieran protegerse. Esta protección se aplica aun cuando el dueño haya publicado los datos en la Web. Como todas las expresiones están protegidas de manera automática, aun si se publican sin el símbolo del copyright, los usuarios deben suponer que todos los datos en la Web están protegidos, a menos que se diga otra cosa. Por tanto, excepto para el uso justo, hacer copias de los datos en la Web es ilegal, a menos que se diga con claridad que hay libertad para copiarlos. La información en una base de datos también puede tratarse como secreto comercial, ya que se suele usar para dar una ventaja competitiva. En ese caso, por supuesto, los datos no se publican. Sin embargo, si una persona obtiene acceso a esta información privada, él o ella no están en libertad de usarla debido a las leyes sobre el secreto comercial. Los datos sensibles o aquellos que deben mantenerse seguros de acuerdo con las leyes de privacidad deben ser protegidos con el empleo de un subsistema de seguridad del sistema de administración de la base de datos y posiblemente otras salvaguardas, y deben almacenarse en forma encriptada, así como encriptarse siempre que se transmitan. Violar los elementos de seguridad para tener acceso no autorizado a los datos, entonces, es ilegal.

14.3 Aspectos de privacidad

Muchos de los datos que la gente proporciona sobre sí misma en el curso normal de sus vidas termina en una base de datos. Registros escolares, que contienen información personal como el nombre, dirección, fecha de nacimiento y detalles acerca del desempeño escolar, con frecuencia se conservan en una base de datos. Cuando las personas solicitan un empleo, se almacenan en una base de datos información sobre sus metas, historia laboral y antecedentes educativos, así como sus referencias, y permanecen en ella mucho tiempo después de que terminó el proceso de reclutamiento. Cuando las personas llenan las declaraciones de impuestos, dan información detallada sobre las fuentes y cantidades de sus ingresos, así como su nombre, dirección y número de seguridad social. En ocasiones las personas dan información sin desearlo acerca de sus hábitos de compra. Siempre que un consumidor hace una compra y paga con un cheque personal, tarjeta de crédito o débito, o usa su tarjeta de "cliente distinguido" con un número de identificación, esa compra se vincula con el cliente. Si la compra se hace a través de un sitio Web, por teléfono o por correo, o si el consumidor se registra para un acuerdo de garantía de servicio, se establece el mismo vínculo. En el proceso de registro, los clientes sin saberlo dan información personal como su nivel educativo, ingresos familiares, aficiones y otros datos no relacionados con la compra. Las personas dan información similar sobre sus antecedentes en el proceso de llenar formatos o cuando entran a concursos de distinto tipo. Cuando solicitan un préstamo o tarjeta de crédito, autorizan a las compañías a comprobar sus antecedentes de crédito.

La tecnología de bases de datos hace posible que se almacenen cantidades enormes de información acerca de los individuos. Los datos que están en el software se podrían usar para buscar registros obtenidos por bases de datos desarrolladas por distintas compañías o instituciones del gobierno. La tecnología de comunicaciones hace posible transferir toda clase de datos a terceras personas que podrían compilar sus propias bases con perfiles de los individuos. Todas estas tecnologías podrían usarse potencialmente para reunir información sobre las personas sin que éstas lo supieran o consintieran. Los aspectos de si es ético recabar ciertos datos sobre un individuo, usarlos sin su consentimiento y compartirlo con terceras personas son temas de mucho debate. Hay un conflicto entre el derecho del individuo a la privacidad y el deseo de los gobiernos y negocios a tener información que pudiera ser de utilidad a ellos.

14.3.1 Privacidad y seguridad

En el capítulo 9 se definió **privacidad** como el derecho de los individuos a tener control de la información sobre ellos. Los datos personales que se recaban y almacenan en bases de datos forman el grueso de la información sobre la que los individuos podrían estar preocupados. En general, la gente proporciona esta información de manera voluntaria con frecuencia sin darse cuenta de que puede almacenarse y quizá recibir un uso inapropiado. Al dar o permitir acceso a la información personal los consumidores dan lo que consideran es una pequeña cantidad de privacidad a cambio de beneficios reales o percibidos. Sin embargo, si las organizaciones o personas que inicialmente obtienen los datos, los pasan o venden a otros sin el consentimiento del consumidor, la privacidad de éste podría ser violada. Hay un desacuerdo respecto de cuánta privacidad ameritan las personas y lo que constituye una violación a ella.

La **seguridad de la base de datos** significa protegerla del acceso, modificación o destrucción no autorizada. El capítulo 9 tiene que ver con la seguridad de la base de datos desde un punto de vista técnico. Además de la necesidad de preservar y proteger los datos para que la organización funcione de manera fluida, los diseñadores de la base de datos tienen la responsabilidad de proteger la privacidad de los individuos respecto de quién guarda los datos.

Los diseñadores de la base de datos y sus administradores deben estar alertas de la legislación aplicable sobre la privacidad y deben ser capaces de establecer e implantar políticas corporativas que respeten los derechos a la privacidad de las personas. Los mecanismos de seguridad analizados en el capítulo 9 deben usarse para implementar dichas políticas.

14.3.2 La privacidad como un derecho humano

En un artículo de 1890, Justice Louis Brandeis y Samuel Warren definieron la privacidad como el "derecho a estar solo" ("The Right to Privacy", *Harvard Law Review* 4, 1890 pp. 193-220). Justice Brandeis escribió que los equipos modernos (en ese caso, equipo fotográfico más rápido) permitían que se cometieran violaciones de la privacidad contra alguien sin su consentimiento, y planteaba que la ley necesitaba extenderse para cubrir esas circunstancias. La colección y diseminación de cantidades enormes de información personal que hacen posibles las bases de datos y las tecnologías de las comunicaciones, por supuesto que caen en la categoría de violaciones potenciales de la privacidad según esta definición.

Los estudiosos modernos han definido la privacidad en términos más completos y dan una base filosófica para ella. Alan Westin, autor del trabajo fundamental escrito en 1967, *Privacity and Freedom*, definió la privacidad como "el deseo de las personas de escoger libremente en qué circunstancias y grado se expondrán a sí mismos, sus actitudes y comportamiento a otros" (*Privacity and Freedom*, Nueva York: Atheneum, 1967, p. 7). En 1990, el comité Calcutt dijo, "En ningún lado hemos encontrado una definición satisfactoria totalmente de lo que es privacidad", pero en su primer informe al respecto dio su propia definición: "el derecho del individuo a ser protegido contra la intrusión en su vida o asuntos personales, o los de su familia, por medios físicos directos o por la publicación de información" (**Report of the Committee on Privacy and Related Matters**, Chairman David Calcutt QC, 1990, Cmnd. 1102 Londres: HMSO, 1990, p. 7). En su libro sobre ética en la computación, Deborah Johnson (*Computer Ethics,* 3a. ed., Prentice Hall, 2001) afirma que la privacidad tiene un valor intrínseco porque es "un aspecto esencial de la autonomía", y la autonomía es un bien intrínseco. James Moor, en su artículo de 1997, "Towards a Theory of Privacy for the Information Age" (*Computers and Society* (27) 3, pp. 27-32), argumenta que la privacidad no es un valor fundamental, lo que él define como aquel valor que se encuentra en todas las culturas humanas. Afirma que los valores fundamentales en la vida son felicidad, libertad, conocimiento, aptitud, recursos y seguridad. Dice que la privacidad en sí misma no es un valor fundamental porque es posible que en ciertas sociedades no sea valorada, pero es una expresión del valor fundamental de la seguridad en nuestra sociedad. Dice que "a medida que las sociedades se hacen más grandes y son muy interactivas, pero menos íntimas, la privacidad se vuelve una expresión natural de la necesidad de seguridad". Richard Spinello identificó la "privacidad de información" en su libro *Cyber Ethics* como el "derecho a controlar la revelación de información personal y el acceso a ésta" (*Cyber Ethics*, Sudbury, MA: Jones y Bartlett 2000 p. 103).

La privacidad es reconocida como un derecho humano fundamental en las constituciones de muchos países, en la declaración de los derechos humanos de las Naciones Unidas, en Convenciones adoptadas por el Consejo Europeo, y en leyes aprobadas tanto en Estados Unidos como en todo el mundo. En la Declaración Universal de los Derechos Humanos adoptada en 1948, la Asamblea General de las Naciones Unidas proclamó en el artículo 12,

> Nadie debe estar sujeto a la interferencia arbitraria con su privacidad, familia, hogar o correspondencia, o en ataques a su honor y reputación. Toda persona tiene el derecho a la protección de la ley contra tales interferencias o ataques.

A pesar de la adopción de esta declaración por parte de las Naciones Unidas, la privacidad no goza de protección universal ni siquiera entre las naciones miembros. Entre países donde

existen leyes sobre la privacidad hay una considerable variación en su definición y en el grado en que está protegida.

14.3.2.1 Legislación sobre la privacidad en Estados Unidos

La constitución de Estados Unidos no aborda la privacidad individual de manera directa, pero la cuarta enmienda de la constitución protege a las personas contra escudriñamientos no razonables, por medio de garantizar:

> El derecho de la gente a estar segura en su persona, casas, documentos y efectos contra escudriñamientos e incautaciones no razonables, no debe ser violado, ni tampoco permitirse ninguna intromisión, sino por causa probable, apoyada por los hechos o afirmación, y en particular con la descripción del lugar en que se va a buscar y las personas o cosas por incautar.

La historia sobre la legislación acerca de privacidad y los intentos de las leyes en Estados Unidos demuestran una preocupación creciente por parte del público y los legisladores del valor de la privacidad. Un conjunto antiguo de principios que gobernaban la información privada era el Code of Fair Information Practices, que formaba parte de un informe de 1972 preparado para el US Department of Health, Education, and Welfare (HEW). Las prácticas de información justas incluían los principios siguientes:

1. No deben mantenerse sistemas de recopilación de datos personales cuya existencia sea secreta.
2. Debe haber una forma de que una persona sepa qué información sobre ella se encuentra en un registro y el uso que se le da.
3. Debe haber forma de que una persona impida que la información sobre ella que se obtuvo para cierto propósito se utilice o se disponga para otros fines sin su consentimiento.
4. Debe haber manera de que una persona corrija o modifique un registro de información identificable sobre ella.
5. Cualquier organización que cree, mantenga, use o distribuya registros de datos personales identificables, debe asegurarse de la confiabilidad de los datos para el uso que se pretende y debe tomar todas las precauciones para impedir que se les dé un uso inapropiado.

[(US Department of Health, Education, and Welfare, Secretary's Advisory Commmittee on Automated Personal Data Systems, Records, Computers, and the Rights of Citizens, viii (1973).]

Estos principios pueden considerarse la base para gran parte de las leyes de la privacidad que han sido aprobadas en Estados Unidos y que se dividen en dos categorías: las que dan protección contra la interferencia del gobierno y las que la dan contra la interferencia de otras instancias, inclusive empresas.

Leyes sobre la privacidad en Estados Unidos que conciernen a las instituciones de gobierno. La Freedom of Information Act de 1966 requiere que las instituciones del poder ejecutivo del gobierno federal, inclusive de los departamentos del gabinete y militares, corporaciones gubernamentales, instituciones reguladoras y otras, den acceso a sus registros. Con ciertas excepciones establecidas en la ley, dichas agencias deben dar copias de sus reglas, opiniones, órdenes, registros y procedimientos a los individuos que lo soliciten. El tema de los registros no necesita ocurrir por petición de la persona. Al respecto se aprobaron enmiendas en 1974, 1986 y 1996.

La Privacity Act de 1974 fue un paso importante en la protección de los derechos a la privacidad de los individuos contra interferencia del gobierno. Permite el acceso de los individuos a los registros gubernamentales sobre ellos. Su propósito manifiesto era,

> . . . balancear la necesidad del gobierno de mantener información sobre las personas con los derechos de éstas para protegerse contra invasiones no deseadas de su privacidad en relación con la obtención, mantenimiento, uso y revelación de información personal sobre ellos por parte de las instituciones federales.

Un componente de esta legislación era un intento por impedir que el gobierno usara los números del seguro social como "identificadores universales" que posibilitaran el uso de vincular información de distintas fuentes. El planteamiento general era que,

> [El congreso estaba . . .] preocupado por los abusos potenciales presentados por el uso creciente que hacía el gobierno de computadoras para almacenar y recuperar datos personales por medio de un identificador universal, como el número de seguridad social del individuo.

Había cuatro objetivos de política básica a los que se dirigía la ley:

1. Restringir la *revelación* de registros identificables personales mantenidos por las agencias.
2. Garantizar a los individuos derechos cada vez mayores para *acceder* a los registros que las organizaciones mantenían sobre ellos.
3. Garantizar a los individuos el derecho de buscar la *modificación* de los registros mantenidos por la agencia sobre ellos cuando se demostraba que no eran exactos, relevantes, oportunos o completos.
4. Establecer un código de "*prácticas de información justa*" que requerían que las agencias cumplieran normas establecidas para recabar, mantener y distribuir los registros.

En 1988, el Congreso aprobó la Computer Matching and Privacy Protection Act a fin de proteger a los individuos de ciertas actividades de búsqueda con computadoras que hacían las agencias gubernamentales. Decía que,

> Estas provisiones agregan requerimientos procesales para que los sigan las agencias cuando emprendan actividades de búsqueda por computadora; da materias de búsqueda con oportunidades de recibir aviso y refutar información adversa antes de que se niegue o termine un beneficio; y requiere que las instituciones involucradas en actividades de búsqueda establezcan consejos para la protección de datos a fin de que supervisen esas actividades.

Las Computer Matching and Privacy Protection Amendments de 1990 se agregaron a las provisiones de proceso en consideración a la ley.

La Patriotic Act de 2002, convertida en ley en octubre de 2001, ha debilitado las provisiones de ciertas leyes sobre la privacidad en el interés de la seguridad nacional. El propósito de la ley es combatir el terrorismo por medio de permitir que las instituciones de cumplimiento de la ley tengan más acceso a la información sobre planes terroristas potenciales. La ley relaja las restricciones sobre compartir información entre agencias del gobierno federal, lo que permite que los funcionarios encargados de hacer cumplir la ley obtengan "grabaciones clandestinas" para vigilar conversaciones por teléfonos celulares sospechosos de terrorismo, lo que da mucho poder para obtener registros de correos electrónicos de personas sospechosas y proporciona a las instituciones un acceso fácil a los registros bancarios a fin de impedir el lavado de dinero. Esta ley es muy controversial y se ha recibido cierto número de amparos contra ella.

Leyes sobre la privacidad en Estados Unidos que conciernen a las empresas privadas. Las leyes analizadas antes restringen el uso de datos personales por parte del gobierno, pero

no se aplican a las empresas privadas, en general, excepto para contratistas del gobierno. En vez de ello se aprobaron leyes individuales acerca de prácticas en varios sectores de negocios. Éstas incluyen las siguientes:

- Fair Credit Reporting Act de 1970.
- Family Educational Rights and Privacy Act de 1974.
- Right to Financial Privacy Act de 1978.
- Federal Managers Financial Integrity Act de 1982.
- Cable Privacy Protection Act de 1984.
- Cable Communications Policy Act de 1984.
- Electronic Communication Privacy Act de 1986 .
- Computer Security Act de 1987.
- Video Privacy Protection Act de 1988.
- Telephone Consumer Protection Act de 1991.
- Driver's Privacy Protection Act de 1994.
- Health Insurance Portability and Accountability Act de 1996.
- Children's Online Privacy Protection Act de 1998.
- Gramm Leach Bliley Financial Services Modernization Act de 1999.

En general, las leyes sobre la protección de la privacidad en Estados Unidos tratan de aplicar los principios de la HEW Code of Fair Information Practices Act a la industria a que se apliquen. Excepto para instancias cubiertas explícitamente por la legislación sobre privacidad, no impide que los negocios obtengan información sobre sus clientes y la compartan con terceras partes. Aun en instancias donde compartir información tiene restricciones, como la Gramm Leach Bliley Financial Services Modernization Act de 1999, el Congreso ha elegido aplicar restricciones con el uso de un enfoque "opcional" en vez de otro "restrictivo". Con el enfoque "restrictivo" una organización no puede compartir información de sus clientes a menos que éstos estén de acuerdo específicamente en ello, mientras que con el enfoque "opcional" los datos se pueden compartir a menos que el cliente solicite que no se haga. Se requiere que las organizaciones notifiquen a sus clientes de su derecho de hacer opcional el compartir los datos, pero si un cliente no responde, la organización puede suponer que dio su permiso. Por lo general se está de acuerdo en que es más restrictivo requerir un enfoque restrictivo que otro opcional, y muchos defensores de la privacidad en Estados Unidos quedaron desilusionados por la aprobación de las provisiones opcionales.

14.3.2.2 Legislación sobre la privacidad en Europa

En Europa existe una norma distinta, y esa diferencia tiene ciertas implicaciones de importancia para empresas de Estados Unidos que hacen negocios en el mercado europeo, ya que limita el flujo de datos a través de las fronteras. Mientras que las leyes de Estados Unidos históricamente han restringido las actividades del gobierno y manejan las actividades de negocios sólo sobre una base de sector por sector, las leyes europeas son más restrictivas con las actividades empresariales. También requieren un enfoque restrictivo más que otro opcional como el mecanismo para obtener la aprobación del cliente para compartir sus datos. La comunidad europea basó sus objetivos sobre los principios desarrollados por la Organización para la Cooperación y el Desarrollo Económico (OECD), organización internacional con 30 países miembros y relaciones de cooperación con otros 70. En 1980, la organización desarrolló normas llamadas políticas de información justa, que incluyen estos

ocho principios tomados de los lineamientos de la OECD acerca de la protección de la privacidad y flujos a través de las fronteras de datos personales (OECD, París, 1981):

- **Principio de la limitación de la obtención.**
  - Debe haber límites a la obtención de datos personales.
  - Los datos deben obtenerse por medios legales y justos.
  - Los datos deben obtenerse con el conocimiento o consentimiento del tema, donde sea posible y apropiado.
- **Principio de la calidad de los datos.** Los datos personales deben ser:
  - Relevantes para los propósitos para los que serán utilizados.
  - Recabarse sólo en el grado necesario para dichos propósitos.
  - Exactos, completos y actualizados.
- **Principio de la especificación del propósito.**
  - Los propósitos para los que se recaban datos personales deben especificarse antes de su obtención.
  - No deben usarse datos en contra de la persona excepto para cumplir con dichos propósitos.
- **Principio de la limitación del uso.** Los datos personales no deben compartirse o emplearse para propósitos distintos de los establecidos en el principio de especificación del propósito, excepto con el consentimiento del sujeto o por la autoridad competente.
- **Principio de las salvaguardas de la seguridad.** Deben utilizarse medidas de seguridad razonables para proteger los datos personales contra el acceso, uso, revelación, modificación, destrucción y pérdida no autorizados.
- **Principio de apertura.** Debe haber una política general de apertura acerca de las prácticas respecto de los datos personales. Debe ser fácil y posible conocer su existencia, naturaleza, propósito de uso y el nombre e información de contacto de la persona que los controla (llamado **controlador de los datos**).
- **Principio de participación individual.** Un individuo debe ser capaz de:
  - Saber si el controlador de los datos tiene información sobre él o ella.
  - Investigar de cuáles datos se trata en un tiempo razonable, con un costo razonable (si lo hubiera) y en forma comprensible.
  - Negarse a cualquier solicitud y obtención de datos relacionados con él o ella. Si la negativa tiene éxito, el individuo tiene el derecho de hacer que los datos se corrijan o borren.
- **Principio de disponibilidad.** Debe contarse con un controlador de los datos que haga cumplir estos principios:

En la convención para la protección de los individuos acerca del procedimiento automático de datos personales, que se propuso en 1981 y adoptó en 1985, el Consejo de Europa estableció la política para sus países miembros respecto del procesamiento de información personal. El resumen de la Convención establece lo siguiente:

> Esta Convención es el primer instrumento internacional que protege al individuo contra abusos que podrían relacionarse con la obtención y procesamiento de datos personales y que busca regular al mismo tiempo el flujo de datos personales a través de los países.

> Además de dar garantías respecto de la obtención de procesamiento de datos personales, reglamenta el procesamiento de datos "delicados" tales como la raza, política, salud, religión, vida sexual, o antecedentes criminales de una persona, en ausencia de las salvaguardas legales apropiadas. La Convención también garantiza el derecho que tiene un individuo a saber cuál información sobre él o ella se encuentra almacenada y, si es necesario, hacer que se corrija.
>
> Las restricciones a los derechos establecidos por la Convención sólo son posibles cuando estén en riesgo intereses superiores (p. ej., seguridad del Estado, defensa, etcétera).
>
> La Convención también establece ciertas restricciones a los flujos a través de las fronteras de los datos personales hacia países en las que las leyes no brinden una protección equivalente (Consejo de Europa, Estrasburgo, 1981).

Tanto los principios de la OECD como la Convención del Consejo de Europa, que son similares, se han usado como la base para las leyes de la privacidad en docenas de países. Las provisiones y redacción exactas varían de un país a otro, pero en general requieren que la información personal se obtenga de manera justa, sólo se utilice para el propósito especificado originalmente, sea relevante y adecuada para éste, no lo exceda, sea exacta y se destruya después de que su propósito se haya cumplido.

En 1995, la Unión Europea adoptó la directiva para protección de datos europeos, "protección de los individuos con respecto del procesamiento de datos personales y sobre el libre movimiento de éstos", lo que requería que todos sus países miembros adoptaran la legislación hacia 1998 para obligar a las mismas normas de protección de la privacidad, que son esencialmente aquellos de la OECD y de la COE. La directiva incluía una provisión para que los datos sobre los ciudadanos de los países de la Unión Europea disfruten del mismo nivel de protección cuando salgan de su país. Otros países han garantizado la recepción segura de dichos datos por medio de la adopción de leyes con las mismas normas, pero Estados Unidos no lo ha hecho. Por tanto, las compañías estadounidenses con operaciones en países socios de la Unión Europea y que tuvieran leyes sobre la privacidad del tipo estadounidense tenían problemas para obtener los datos necesarios para su negocio, ya que no podían obtener de manera legal información sobre sus clientes en dichas naciones. Se llegó a una solución parcial en 2000 entre la Unión Europea y la US Federal Trade Commission usando el mecanismo **Puerto Seguro**. Las compañías de Estados Unidos certifican que siguen las reglas del acuerdo del puerto seguro, que significa que cumplen con las reglas de Estados Unidos respecto al tratamiento de los datos. Las compañías deben ser certificadas anualmente por el Departamento de Comercio de Estados Unidos acerca de que cumplen con las normas, y establecer en sus enunciados de política sobre la privacidad que lo hacen. La certificación requiere que una compañía se suscriba a una agencia de vigilancia de autorregulación (como TRUSTe) o desarrolle su propia política de autorregulación. El Departamento de Comercio publica la lista de compañías que pertenecen al programa de puerto seguro. Obligar al cumplimiento de la ley es responsabilidad de la Federal Trade Commission (Comisión Federal del Comercio) o de otras instituciones federales estatales. El castigo es a través de multas elevadas y de la expulsión de la compañía ofensora del programa puerto seguro.

14.4 Factores humanos

14.4.1 Factores humanos en el desarrollo del software

Los factores humanos en el desarrollo del software se refieren a factores que promueven y facilitan el desempeño óptimo de parte del usuario. Estos factores incluyen características tanto físicas del usuario como elementos psicológicos. Los factores humanos tienen impor-

tancia vital en ciertos software críticos como el que se utiliza en los sistemas de conservación de la vida, plantas de energía nuclear, dispositivos de tratamiento médico, aeronaves, y muchas otras áreas en las que el error o falla puede tener consecuencias contra la vida. En los negocios, el diseño de los factores humanos reduce errores y aumenta la productividad, en especial en tareas repetitivas tales como la entrada de órdenes. El buen diseño que resulta de la consideración de factores humanos promueve la aceptación y satisfacción del usuario. Al diseñar con cuidado los factores humanos, los problemas físicos tales como la fatiga y psicológicos como el estrés se reducen. El resultado del diseño centrado en el usuario es un aumento en la comodidad de éste y en su seguridad, mayor productividad, más satisfacción por parte del usuario y una disminución de los errores. También abate los costos de capacitación y de manejar los errores y su recuperación.

Un enfoque de ingeniería del sistema de factores humanos incluye la consideración del usuario en cada etapa del proyecto del sistema. Un proyecto de sistema común incluye las etapas siguientes:

- **Conceptualización.** Los diseñadores identifican los objetivos del sistema y definen las especificaciones para éste.
- **Definición.** Los diseñadores definen el sistema, inclusive los subsistemas. Describen el ambiente en el que se usarán y la forma en que lo harán.
- **Diseño general.** Los diseñadores identifican todos los componentes del sistema y sus funciones. Los componentes incluyen hardware, software y componentes humanos. Analizan las tareas y necesidades.
- **Diseño de la interfaz.** Los diseñadores utilizan estándares de diseño apropiados para crear las interfaces del usuario. Aplican principios de diseño, datos estadísticos, modelación matemática, resultados de estudios empíricos y experiencia.
- **Desarrollo.** Los desarrolladores implementan el sistema, lo prueban y modifican si así lo aconsejan las pruebas.
- **Despliegue.** Cuando se despliega el sistema, los desarrolladores evalúan su uso y hacen las modificaciones necesarias.
- **Mantenimiento.** Una vez que el sistema funciona, se implantan procedimientos para su evaluación continua, evolución y mejora.

En cada etapa, los diseñadores y desarrolladores se benefician al tomar en cuenta los factores humanos. La inclusión de los usuarios en los análisis durante las etapas de conceptualización, definición y diseño para obtener sus aportaciones sobre el sistema propuesto en esos primeros momentos, ayuda a garantizar que el diseño del sistema esté centrado en el usuario. Los especialistas en factores humanos que emplea la empresa o que trabajen como consultores son miembros valiosos del equipo de diseño. En la etapa de diseño de la interfaz, los diseñadores deben incorporar principios en el diseño de ésta que hayan sido desarrollados por expertos en el campo. IEEE y otras organizaciones publican estándares para ingeniería de software que pueden ser utilizados, así como libros y artículos sobre diseño de sistemas que pueden ser usados. Otras fuentes de principios de diseño incluyen cierto número de excelentes institutos de investigación, como el University of Maryland Human-Computer Interaction Lab, el Human-Computer Interaction Institute en la Carnegie-Mellon University, el Media Lab en el Massachusetts Institute of Technology y el programa de la Stanford University acerca de la interacción humano-computadora. Las pruebas sobre la usabilidad deben continuar durante las fases de desarrollo, despliegue y mantenimiento, porque aun cambios pequeños que parezcan insignificantes para los desarrolladores llegan a tener un gran efecto en la posibilidad de uso.

14.4.2 La interfaz humano-base de datos

La medida definitiva del éxito de un proyecto de base de datos es el grado en que los usuarios lo utilizan en realidad para obtener la información que necesitan. El diseñador de una base de datos normalmente dedica una considerable cantidad de tiempo a asegurarse que la base de datos es útil, que satisface realmente las necesidades de información de la empresa. Como se dijo en el capítulo 2 y se ilustró en otros posteriores, el diseñador entrevista a los usuarios para determinar sus necesidades de datos, diseña el modelo conceptual, aplica técnicas como la normalización para garantizar la calidad del modelo, escoge el sistema de administración de base de datos que sea más apropiado, elabora el mapeo lógico, diseña el modelo interno, produce el sistema y desarrolla la base de datos. El diseñador puede estar satisfecho si la base de datos en sí misma está bien diseñada y es muy eficiente. Sin embargo, si no estuviera utilizable debido a la mala interfaz del usuario, los usuarios finales no quedarán satisfechos y se sentirían poco inclinados a usar la base de datos. La susceptibilidad del uso es un aspecto de la calidad que amerita el mismo tipo de atención cuidadosa que el resto del diseño de la base de datos.

Jakob Nielsen, experto en uso, hace la diferencia entre **utilidad**, que significa la funcionalidad del diseño, y la **usabilidad**. Ambas son atributos de la calidad y son esenciales para el éxito de un proyecto de base de datos. Sin utilidad la base de datos no provee lo que necesitan los usuarios, y sin usabilidad éstos no pueden obtener lo que necesitan debido a que la interfaz con el usuario es demasiado difícil. Tanto la utilidad como la usabilidad se miden por medio de métodos de investigación del usuario.

La Organización Internacional de Estándares (ISO) publica un documento de estándares sobre la usabilidad en la que se define ésta como "el grado en que puede usarse un producto por usuarios especificados a fin de alcanzar metas específicas con eficacia, eficiencia y satisfacción en un contexto de uso especificado". (ISO) *Ergonomic requirement for office with visual display terminals. (VDTs) –Parte II: Guidance on usability.*

Nielsen describe la usabilidad como "atributo de la calidad que evalúa la facilidad de uso de las interfaces de usuario" (Jakob Nielsen. "Usability 101" http://www.useit.com/alertbox/20030825.html). Él lista cinco componentes de usabilidad.

1. El **aprendizaje** se refiere a la facilidad con que los usuarios realizan sus tareas desde la primera vez que ven el diseño.
2. **Eficiencia.** Se refiere a la forma rápida en que los usuarios llevan a cabo tareas una vez que dominan el diseño.
3. **Memorización.** Se refiere a la facilidad en que los usuarios vuelven a ser eficientes al usar el diseño después de cierto periodo de no utilizarlo.
4. **Errores.** Se refiere al número y severidad de los errores que cometen los usuarios, y la forma en que se recuperan fácilmente de ellos.
5. **Satisfacción.** Se refiere a qué tan agradable es usar el diseño.

Para los usuarios finales, la usabilidad se relaciona mucho con la productividad. Nielsen sugiere que alrededor del 10% del presupuesto de un proyecto debe gastarse en la usabilidad. Él afirma que dicha inversión mejorará sustancialmente el software, duplicará la calidad de las mediciones en un sitio Web y mejorará una calidad métrica de Intranet casi por completo. En las mediciones grandes, dichas mejoras significan la reducción de los costos de capacitación a la mitad y la duplicación del número de transacciones que se pueden llevar a cabo por parte de los empleados o consumidores.

14.4.3 Prueba de usabilidad para aplicaciones de bases de datos

Cuando se diseña para usuarios interactivos, los diseñadores de bases de datos deben considerar los elementos del diseño interactivo, que incluyen el uso de metáforas (p. ej., apariencia y sensación del escritorio), calidad y consistencia de las claves de navegación, calidad de los mensajes de error, retroalimentación efectiva y componentes visuales tales como el arreglo, fuentes y color. El arreglo debe ser consistente de una pantalla a la otra. La pantalla no debe estar saturada, los colores y fuentes deben mejorar la facilidad de lectura, y el lenguaje debe ser sin ambigüedades, sencillo, claro y apropiado para los usuarios. Debe tenerse cuidado en utilizar el lenguaje y símbolos que no ofendan a ningún grupo o subgrupo. Deben elegirse ilustraciones y ejemplos que demuestren y aumenten la diversidad.

Es posible utilizar algunas técnicas básicas para estudiar la usabilidad. Nielsen sugiere hacer pruebas de usuario repetidas. El proceso consiste en encontrar algunos usuarios representativos, hacer que realicen tareas comunes y observar sus acciones, para notar dónde tienen éxito y dónde tienen problemas con la interfaz. Incluso un número tan reducido como cinco usuarios arroja información valiosa. Nielsen sugiere efectuar muchas pruebas pequeñas conforme avance el diseño, para revisar éste en forma interactiva entre una prueba y otra. El uso de los elementos del diseño interactivo es apropiado a fin de diseñar pruebas de varios componentes y distribuciones y para realizar pequeños estudios para ver el efecto que tienen los cambios. Pueden hacerse pequeños estudios en forma de estudios de caso. Es posible plantear hipótesis de prueba formales con el empleo de un número limitado de sujetos, siempre y cuando se utilicen las distribuciones estadísticas apropiadas como la *t* de Student. Sin embargo, antes de llegar al sistema final, deben probarse hipótesis formales con el empleo de métodos robustos aplicados a una muestra grande.

14.4.4 Normas éticas para los profesionales de la computación y el software

En su trabajo, los profesionales encuentran situaciones que requieren criterios morales. Muchas profesiones tienen códigos de ética para guiar los actos de sus miembros en tales situaciones. Las organizaciones profesionales del cómputo han creado códigos de conducta que guían a los trabajadores de la computación y el software. El primero de ellos es el Código de Ética y Conducta Profesional, adoptado por la Association for Computing Machinery (ACM), y se reproduce a continuación en su totalidad.

Código de Ética y Conducta Profesional de ACM

Adoptado por el Consejo de ACM 10/16/92

Preámbulo

Se espera un compromiso con la conducta ética profesional por parte de todos los miembros (con derecho a voto, asociados y estudiantes) de la Association for Computing Machinery (ACM).

Este Código consiste en 24 imperativos formulados como enunciados de responsabilidad personal, e identifica los elementos de dicho compromiso. Contiene muchas, si no es que todas, las situaciones que es probable que enfrenten los profesionales. La sección 1 describe las consideraciones éticas fundamentales, mientras que la sección 2 aborda otras adicionales, más específicas, de la conducta profesional. Los enunciados en la sección 3 se refieren más a individuos que tienen un rol de liderazgo, sea en el sitio de trabajo o en alguna institución de voluntarios en organizaciones como la ACM. En la sección 4 se dan principios que involucran el cumplimiento de este Código.

El Código se suministra con un conjunto de lineamientos, que dan explicaciones para ayudar a los miembros al tratar con varios aspectos de su contenido. Se espera que los lineamientos cambien con más frecuencia que el Código.

El Código y sus lineamientos buscan servir como base para tomar decisiones éticas en la conducción del trabajo profesional. En segundo lugar, sirve como base para determinar el mérito de una queja formal respecto de la violación de las normas éticas profesionales.

Debe notarse que aunque no se menciona la computación en los imperativos de la sección 1, el Código tiene que ver con el modo en que se aplican dichos imperativos fundamentales a la conducta propia como profesional de la computación. Estos imperativos se expresan en forma general para hacer énfasis en que los principios éticos que se aplican a la ética de la computación provienen de otros más generales.

Se entiende que algunas palabras y expresiones de un código de ética están sujetas a interpretaciones variables, y que cualquier principio ético puede entrar en conflicto con otros en situaciones específicas. Las preguntas relacionadas con conflictos éticos se responden mejor si se hace un análisis detenido de los principios fundamentales, más que con la atención a reglamentos detallados.

1. IMPERATIVOS MORALES GENERALES.

Como miembro de ACM, deberé…

1.1 Contribuir al bienestar de la sociedad y los seres humanos.

Este principio tiene que ver con la calidad de vida de todas las personas, y afirma que es una obligación proteger los derechos humanos fundamentales y respetar la diversidad de las culturas. El objetivo esencial de los profesionales de la computación es minimizar las consecuencias negativas de los sistemas de cómputo, inclusive las amenazas a la salud y seguridad. Cuando se diseñen o implementen sistemas, los profesionales de la computación deben tratar de garantizar que el producto de sus esfuerzos se utilice en formas socialmente responsables, satisfaga las necesidades sociales y evite los efectos dañinos a la salud y bienestar.

Además de un ambiente social seguro, el bienestar humano incluye un ambiente natural seguro. Por tanto, los profesionales de la computación que diseñen y desarrollen sistemas deben estar alertas, y alertar a los demás, de cualquier daño potencial al ambiente local o global.

1.2 Evitar dañar a otros.

"Dañar" significa herir o causar consecuencias negativas, como pérdidas indeseables de información o propiedad, daño a las propiedades o generar efectos no deseados en el ambiente. Este principio prohíbe el uso de la tecnología de la computación en formas que causen daño a cualquiera de los usuarios siguientes: público en general, empleados y empleadores. Las acciones dañinas incluyen la destrucción o modificación intencional de archivos y programas que lleven a una pérdida seria de recursos o al gasto innecesario de recursos humanos tales como el tiempo y esfuerzo requeridos para eliminar de los sistemas "virus de computadora".

Las acciones bien intencionadas, inclusive aquellas que cumplen los deberes asignados, pueden provocar un daño inesperado. En ese caso, la persona o personas responsables están obligadas a corregir o mitigar las consecuencias negativas

tanto como sea posible. Una forma de evitar el daño no intencional es considerar con cuidado los efectos potenciales en todos aquellos afectados por las decisiones tomadas durante el diseño e implementación.

Para minimizar la posibilidad de dañar a otros de manera involuntaria, los profesionales de la computación deben minimizar los malos funcionamientos por medio del seguimiento de estándares generalmente aceptados para el diseño y prueba de los sistemas. Además, con frecuencia resulta necesario evaluar las consecuencias sociales de los sistemas a fin de evaluar la probabilidad de causar algún daño serio a los demás. Si las características del sistema son desconocidas para los usuarios, compañeros o supervisores, el profesional de la computación es responsable en lo individual de cualquier daño que resulte.

En el ambiente de trabajo, el profesional del cómputo tiene la obligación adicional de reportar cualesquiera signos de peligros en el sistema que pueda tener como resultado un daño serio, personal o social. Si los superiores de alguien no actúan para evitar o mitigar dichos peligros, puede ser necesario "delatar" para ayudar a corregir el problema o reducir el riesgo. Sin embargo, el reporte caprichoso o equívoco de las violaciones puede ser dañino en sí mismo. Antes de reportar violaciones, deben evaluarse de manera exhaustiva todos los aspectos relevantes del incidente. En particular, debe ser creíble la evaluación del riesgo y la responsabilidad. Se sugiere buscar el consejo de otros profesionales del cómputo. Véase el principio 2.5, respecto de las evaluaciones a fondo.

1.3 Ser honesto y confiable.

La honestidad es una componente esencial de la confianza. Sin ésta, una organización no puede funcionar con eficacia. El profesional de la computación honesto no hará en forma deliberada afirmaciones falsas o inmerecidas sobre un sistema o su diseño, sino que promoverá la revelación completa de todas sus limitaciones y problemas pertinentes.

Un profesional del cómputo tiene el deber de ser honesto respecto de sus propias calificaciones, y en ninguna circunstancia debe tener conflictos de interés.

La membresía en organizaciones voluntarias tales como la ACM en ocasiones coloca a los individuos en situaciones en las que sus palabras o acciones podrían interpretarse como si tuvieran el "peso" de un grupo más grande de profesionales. Un miembro de ACM tendrá cuidado de no asumir la representación de esta organización, de sus posiciones y políticas o de cualesquiera de sus unidades.

1.4 Ser justo y actuar para no discriminar.

Los valores de igualdad, tolerancia, respeto por los demás, y los principios de igualdad ante la justicia son los que gobiernan este imperativo. Discriminar por cuestiones de raza, sexo, religión, edad, discapacidad, nacionalidad u otros factores parecidos, es una violación explícita de la política de ACM y no se tolerará.

Del uso o mal uso de la información y la tecnología llegan a resultar desigualdades entre diferentes grupos de personas. En una sociedad justa, todos los individuos tendrían igualdad de oportunidades para participar o beneficiarse del uso de los recursos de cómputo sin importar su raza, sexo, religión, edad, discapacidad, origen nacional u otros factores similares. Sin embargo, estos ideales no justifican el uso no autorizado de la computación para que proporcione una base para violar cualesquiera imperativos éticos de este código.

1.5 Honrar los derechos de propiedad, inclusive los derechos de autor y las patentes.

La violación de los derechos de autor, patentes, secretos comerciales y los términos de acuerdos de licencias están prohibidos por la ley en la mayor parte de circunstancias. Aun cuando el software no está protegido, tales violaciones son contrarias al comportamiento profesional. Las copias del software deben hacerse sólo con la autorización pertinente. No se perdonará la duplicación no autorizada de materiales.

1.6 Dar el crédito apropiado por la propiedad intelectual.

Los profesionales de la computación están obligados a proteger la integridad de la propiedad intelectual. En específico, no se debe tomar el crédito por las ideas o trabajo de otros, aun en aquellos casos en los que el trabajo no está protegido en forma explícita por el derecho de autor, patente, etcétera.

1.7 Respeto de la privacidad de los demás.

La tecnología de cómputo y comunicaciones permite la obtención e intercambio de información personal en una escala sin precedentes en la historia de la civilización. Hay un mayor potencial para violar la privacidad de individuos y grupos. Es responsabilidad de los profesionales mantener la privacidad e integridad de los datos que describen a los individuos. Esto incluye tomar precauciones para asegurarse de la exactitud de los datos, así como para protegerlos del acceso no autorizado o la revelación accidental a individuos inapropiados. Además, deben establecerse procedimientos que permitan que los individuos revisen sus registros y se corrijan las inexactitudes.

Este imperativo implica que en un sistema sólo debe recabarse la cantidad necesaria de información personal, que los periodos de conservación y disposición de la información se definan con claridad y se cumplan, y que la información personal recabada para un propósito específico no se utilice para otros sin el consentimiento del(os) individuo(s). Estos principios se aplican a las comunicaciones electrónicas, inclusive el correo electrónico, y prohíben los procedimientos que capturan o vigilan en forma electrónica los datos de los usuarios, inclusive sus mensajes, sin su permiso o autorización de buena fe en relación con la operación y mantenimiento del sistema. Los datos de los usuarios observados durante las tareas normales de la operación y mantenimiento de un sistema deben ser tratados con confidencialidad estricta, excepto en los casos en que hay evidencias de violación de la ley, de regulaciones organizacionales o de este Código. En estos casos, la naturaleza o contenido de la información debe revelarse sólo a las autoridades correspondientes.

1.8 Respetar la confidencialidad.

El principio de honestidad se extiende a aspectos de confidencialidad de la información siempre que alguien haya hecho una promesa explícita de honrar la confidencialidad, o de manera implícita cuando se tengan acceso a información privada que no se relaciona directamente con las tareas propias. La cuestión ética es respetar todas las obligaciones de confidencialidad ante los empleadores, clientes y usuarios, a menos que se eliminen por requerimientos de la ley o de otros principios de este Código.

2. RESPONSABILIDADES PROFESIONALES MÁS ESPECÍFICAS.

Como profesional de la computación de ACM, deberé…

2.1 Pugnar por alcanzar la calidad, eficacia y dignidad más elevadas tanto en el proceso como en los productos del trabajo profesional.

La excelencia es tal vez la obligación más importante de un profesional. El profesional de la computación debe luchar por lograr la calidad y ser consciente de las serias consecuencias negativas que resultan de la mala calidad de un sistema.

2.2 Adquirir y mantener la competencia profesional.

La excelencia depende de los individuos que aceptan la responsabilidad de adquirir y conservar la competencia profesional. Un profesional debe participar en el establecimiento de los estándares para los niveles de competencia apropiados, y luchar porque éstos se alcancen. El aumento del conocimiento técnico y la competencia se logra de varias maneras: con estudios independientes; asistencia a seminarios, conferencias o cursos; y con el involucramiento en organizaciones profesionales.

2.3 Conocer y respetar las leyes existentes sobre el trabajo profesional.

Los miembros de ACM deben obedecer las leyes locales, estatales, provinciales, nacionales e internacionales existentes, a menos que haya una base ética fuerte para no hacerlo. También deben obedecerse las políticas y procedimientos de las organizaciones en que se participa. Pero la obligatoriedad debe balancearse con el reconocimiento de que en ocasiones las leyes y reglamentos pueden ser inmorales o inapropiados y, por tanto, deben ponerse en tela de juicio. La violación de una ley o reglamento es ética si tienen una base moral inadecuada o cuando entran en conflicto con otras más importantes. Si alguien decide violar una ley o reglamento porque la considere como carente de ética, o por cualquier otra razón, se debe aceptar toda la responsabilidad por las acciones propias y sus consecuencias.

2.4 Aceptar y proporcionar la revisión profesional apropiada.

La calidad del trabajo profesional, en especial en la profesión del cómputo, depende de la revisión y crítica profesional. Siempre que sea apropiado, los miembros individuales deben buscar y utilizar la revisión de sus pares, así como brindar la revisión crítica del trabajo de otros.

2.5 Hacer evaluaciones exhaustivas y completas de los sistemas de cómputo y sus efectos, inclusive el análisis de los posibles riesgos.

Los profesionales de la computación deben tratar de ser perceptivos y objetivos cuando evalúen, recomienden y presenten descripciones y alternativas de un sistema. Ellos están en una posición que goza de confianza especial, por lo que tienen la responsabilidad también especial de hacer evaluaciones objetivas y creíbles a los empleadores, clientes, usuarios y público. Cuando haga evaluaciones, el profesional también debe identificar cualesquiera conflictos relevantes de interés, según se establece en el imperativo 1.3.

Como ya se dijo en el principio 1.2 sobre evitar el daño, deben reportarse cualesquiera signos de peligro en los sistemas a quienes tienen la oportunidad y responsabilidad de resolverlos. Véanse los lineamientos del imperativo 1.2 para más detalles sobre el daño, inclusive los reportes de violaciones profesionales.

2.6 Respetar los contratos, acuerdos y responsabilidades asignadas.

El respeto de los compromisos propios es asunto de integridad y honestidad. Para el profesional del cómputo esto incluye asegurarse que los elementos del sistema funcionan como se pretendía. Asimismo, cuando se contrate un trabajo con otra parte, se tiene la obligación de informar a ésta en forma apropiada sobre el avance hacia la conclusión del trabajo.

Un profesional de la computación tiene la responsabilidad de solicitar un cambio en cualquier asignación que sienta que no es capaz de cumplir como se necesita. Se debe aceptar el trabajo sólo después de una consideración cuidadosa y la revelación completa de los riesgos y preocupaciones al empleador o cliente. El primordial principio que subyace a esto es la obligación de aceptar la disponibilidad personal para el trabajo profesional. En ciertas ocasiones, otros principios éticos exigen una prioridad más alta.

No debe aceptarse un juicio de que una tarea específica no debiera llevarse a cabo. Una vez identificadas con claridad las preocupaciones y razones propias para emitir dicho juicio, pero sin la capacidad de procurar un cambio en la tarea, se puede estar obligado por contrato o la ley, a proceder como se acordó. El criterio ético del profesional de la computación debe ser la guía definitiva para decidir si continúa o no. Sin importar la decisión que se tome, se debe aceptar la responsabilidad de las consecuencias.

Sin embargo, la realización de los trabajos "contra el propio criterio" no libera al profesional de su responsabilidad por cualquier consecuencia negativa.

2.7 Mejorar la comprensión del público de la computación y sus consecuencias.

Los profesionales del cómputo tienen la responsabilidad de compartir el conocimiento técnico con el público, por medio de propiciar el entendimiento de la computación, inclusive los efectos de los sistemas y sus limitaciones. Este imperativo implica la obligación de contrarrestar cualquier punto de vista equivocado en relación con el cómputo.

2.8 Acceder a los recursos de cómputo y comunicaciones sólo cuando se esté autorizado para hacerlo.

El robo o la destrucción de propiedades tangibles y electrónicas están prohibidos por el imperativo 1.2, "evitar dañar a otros". La incursión y uso no autorizado de un sistema de cómputo o comunicaciones está regido por este imperativo. La incursión incluye acceder a las redes de comunicación y sistemas de cómputo, o a las cuentas o archivos asociados con estos sistemas sin autorización explícita para hacerlo. Los individuos y organizaciones tienen el derecho de restringir el acceso a sus sistemas en tanto no violen el principio de discriminación (*véase* 1.4). Nadie debe entrar o usar el sistema de computación, software o archivos de datos de otra persona sin el permiso de ésta. Siempre se debe tener la aprobación correspondiente antes de usar los recursos de un sistema, inclusive los puertos de comunicación, espacio de archivo, periféricos del sistema y tiempo de computadora.

3. IMPERATIVOS DE LIDERAZGO ORGANIZACIONAL.

Como miembro de ACM y líder organizacional, deberé…

NOTA DE ANTECEDENTES: Esta sección se basa en gran parte en el borrador del Código de Ética de IFIP, en especial en sus secciones sobre ética organizacional y

aspectos internacionales. Las obligaciones éticas de las organizaciones tienden a ser ignoradas en la mayor parte de los códigos de conducta profesional, tal vez debido a que están escritos desde la perspectiva del miembro individual. Este dilema se cancela si se enuncian los imperativos desde el punto de vista del líder de la organización. En este contexto, el "líder" es visto como cualquier miembro de la organización con responsabilidades de liderazgo o educativas. Por lo general, estos imperativos se aplican a las organizaciones y a sus líderes. En este contexto, las "organizaciones" son corporaciones, instituciones del gobierno y otros "empleadores", así como organizaciones profesionales de voluntarios.

3.1 Articular las responsabilidades sociales de los miembros de una unidad organizacional y promover su aceptación total.

Debido a que las organizaciones de todas clases tienen algún efecto sobre el público, deben aceptar su responsabilidad ante la sociedad. Los procedimientos y actitudes organizacionales orientados a la calidad y el bienestar de la sociedad reducirán el daño a los miembros del público, con lo que se atenderá el interés público y se cumplirá con la responsabilidad social. Entonces, los líderes de las organizaciones deben promover la participación completa en la satisfacción de las responsabilidades sociales al mismo tiempo que se cumple con la calidad.

3.2 Administrar el personal y los recursos para diseñar y construir los sistemas de información que mejoren la calidad de vida de trabajo.

Los líderes organizacionales son responsables de asegurarse que los sistemas de cómputo mejoren, no disminuyan, la calidad de vida laboral. Cuando implementen un sistema de cómputo, las organizaciones deben considerar el desarrollo personal y profesional, la seguridad física y la dignidad humana de todos los trabajadores. En el diseño del sistema para el sitio de trabajo deben tomarse en cuenta los estándares ergonómicos apropiados entre el humano y la computadora.

3.3 Promover y apoyar los usos apropiados y autorizados de los recursos de cómputo y comunicación de una organización.

Debido a que los sistemas de cómputo se convierten en herramientas que pueden dañar o beneficiar a una organización, el liderazgo tiene la responsabilidad de definir con claridad lo que constituye un uso apropiado o inapropiado de ellos. Si bien el número y alcance de tales reglas debe ser mínimo, una vez establecido su cumplimiento debe ser obligatorio.

3.4 Asegurar que los usuarios y aquellos que serán afectados por un sistema tengan sus necesidades articuladas con claridad durante la evaluación y diseño de los requerimientos; después se debe validar el sistema a fin de que cumpla los requerimientos.

Los usuarios del sistema actual, los usuarios potenciales y otras personas cuyas vidas se vean afectadas por un sistema, deben ver sus necesidades incorporadas en el enunciado de los requerimientos, y el cumplimiento de éstos debe asegurarse por medio de la validación del sistema.

3.5 Articular y apoyar políticas que protejan la dignidad de los usuarios y otras personas afectadas por un sistema de cómputo.

El diseño o implementación de sistemas que perjudiquen a individuos o grupos ya sea en forma deliberada o inadvertida, es éticamente inaceptable. Los profesionales de la computación situados en los puestos con poder de decisión deben verifi-

car que los sistemas se diseñen e implementen para proteger la privacidad personal y mejoren la dignidad del individuo.

3.6 Crear oportunidades para que los miembros de la organización aprendan los principios y limitaciones de los sistemas de cómputo.

Esto complementa el imperativo sobre el entendimiento del público (2.7). Las oportunidades educativas son esenciales para facilitar la participación óptima de todos los miembros de la organización, quienes deben disponer de ellas para que los ayuden a mejorar su conocimiento y habilidad de cómputo, inclusive con cursos que los familiaricen con las consecuencias y limitaciones de tipos especiales de sistemas. En particular, los profesionales deben estar alertas de los peligros de construir sistemas con base en modelos muy simplificados, en lo improbable que resulta anticipar y diseñar todas las condiciones posibles de operación, y otros aspectos relacionados con la complejidad de esta profesión.

4. CUMPLIMIENTO DEL CÓDIGO.

Como miembro de ACM, deberé…

4.1 Cumplir y promover los principios de este Código.

El futuro de la profesión computacional depende de la excelencia tanto técnica como ética. No sólo es importante que los profesionales de ACM se adhieran a los principios plasmados en este Código, sino que cada miembro debe estimular y apoyar el cumplimiento de otros miembros.

4.2 Tratar las violaciones de este código como algo que no es consistente con ser miembro de ACM.

Que los profesionales acepten un código de ética es en gran medida un acto voluntario. Sin embargo, si un miembro no cumple éste porque cometa faltas graves, terminará su membresía en ACM.

Este Código y los lineamientos suplementarios fueron desarrollados por la Fuerza de Tarea para la Revisión del Código de Ética y Conducta Profesional de ACM: Ronald E. Anderson, director; Gerald Engel, Donald Gotterbarn, Grace C. Hertlein, Alex Hoffman, Bruce Jawer, Deborah G. Johnson, Doris K. Lidtke, Joyce Currie Little, Dianne Martin, Donn B. Parker, Judith A. Perrolle y Richard S. Rosenberg. Esta Fuerza de Tarea fue organizada por ACM/SIGCAS, y recibió financiamiento del Fondo Discrecional de ACM SIG. Este Código y los lineamientos suplementarios fueron adoptados por el Consejo de ACM el 16 de octubre de 1992.

Además del Código de Ética y Conducta Profesional de ACM, que describe las prácticas para todos los profesionales de la computación, ACM e IEEE han creado en conjunto un código de ética y práctica profesional para guiar a los ingenieros de software. Estos lineamientos se aplican a los profesionales que diseñan software, inclusive sistemas de administración de bases de datos, software que mejore las capacidades de éstos o aplicaciones de bases de datos desarrolladas para un cliente.

Código de Ética para los Ingenieros de Software y su Práctica Profesional (Versión 5.2), según la recomienda la Fuerza de Tarea Conjunta de ACM/IEEE-CS, para la Ética de la Ingeniería de Software y su Práctica Profesional, y aprobada en conjunto por ACM e IEEE-CS como la norma para la enseñanza y práctica de la ingeniería de software.

PREÁMBULO

Las computadoras tienen un papel central y cada vez mayor en el comercio, industria, gobierno, medicina, educación, entretenimiento y sociedad en general. Los ingenieros de software son aquellos que contribuyen con su participación directa o enseñanza, al análisis, especificación, diseño, desarrollo, certificación, mantenimiento y prueba de sistemas de software. Debido al del papel que tienen en el desarrollo de sistemas de software, los ingenieros de la especialidad tienen oportunidades significativas para hacer el bien o causar un daño, permitir que otros hagan un bien o un daño, o influir en otros para que hagan el bien o un mal. Para garantizar en la medida de lo posible que sus esfuerzos se utilizarán para el bien, los ingenieros de software deben comprometerse a hacer de la ingeniería de software una profesión benéfica y respetada. De acuerdo con ese compromiso, los ingenieros de software deben comprometerse a seguir el Código de Ética y Práctica Profesional.

El Código contiene ocho Principios relacionados con el comportamiento y decisiones tomadas por los ingenieros de software, inclusive los practicantes, educadores, gerentes, supervisores y quienes establecen la política, así como los aprendices y estudiantes de la profesión. Los Principios identifican las relaciones con responsabilidad ética en las que participan los individuos, grupos y organizaciones, así como las principales obligaciones dentro de estas relaciones. Las Cláusulas de cada Principio son ilustraciones de algunas obligaciones incluidas en estas relaciones. Estas obligaciones se fundamentan en la persona del ingeniero de software, con especial atención para las personas afectadas por su trabajo, y los elementos únicos de la práctica de la ingeniería de software. El Código prescribe éstas como obligaciones de cualquiera que afirme ser o aspire a ser un ingeniero de software.

No se pretende que partes individuales del Código se usen por separado para justificar errores por omisión o comisión. La lista de Principios y Cláusulas no es exhaustiva. Las Cláusulas no deben leerse como algo que separa lo aceptable de lo inaceptable en la conducta profesional en todas las situaciones prácticas. En ciertas situaciones, quizá las normas estén en tensión entre sí o con normas de otras fuentes. Estas situaciones requieren que el ingeniero de software utilice su criterio ético para actuar del modo más consistente con el espíritu del Código de Ética y Práctica Profesional, en las circunstancias dadas.

Las tensiones éticas se enfrentan mejor por medio de la consideración cuidadosa de principios fundamentales, en lugar de una confianza ciega en reglamentos detallados. Estos Principios deben influir en los ingenieros de software para que consideren en forma amplia quién resulta afectado por su trabajo; para examinar si ellos o sus colegas tratan a los seres humanos con el debido respeto; para considerar cómo vería el público, si estuviera razonablemente informado, sus decisiones, y para considerar si sus actos serían juzgados en beneficio del profesional ideal que trabaja como ingeniero de software. En todos estos criterios, lo principal es la preocupación por la salud, seguridad y bienestar del público; es decir, el "Interés Público" es el centro de este Código.

El contexto dinámico y demandante de la ingeniería de software requiere un código que sea adaptable y relevante a las nuevas situaciones a medida que ocurren. Sin embargo, aun en esta generalidad, el Código brinda apoyo a los ingenieros de software y sus administradores, que necesitan emprender acciones positivas en un caso específico mediante la documentación del estado de la ética de la profesión. El Código provee un fundamento ético al que pueden recurrir los individuos que integran los equipos y el equipo como un todo. El Código ayuda a definir las accio-

nes que resultan impropias, en cuanto a la ética, pedir a un ingeniero o equipos de ingenieros de software.

El Código no es sólo para adjudicar la naturaleza de actos cuestionables; también tiene una función educativa importante. Como este Código expresa el consenso de la profesión respecto de aspectos éticos, es un medio de educar tanto al público como a quienes aspiran a ser profesionales sobre las obligaciones éticas de todos los ingenieros de software.

PRINCIPIOS

Principio 1: EL PÚBLICO

Los ingenieros de software deben actuar en forma consistente con el interés público. En particular, los ingenieros de software deben hacer lo siguiente, por tratarse de algo apropiado:

1.01. Aceptar toda la responsabilidad de su trabajo.

1.02. Armonizar con el bien público los intereses del ingeniero de software, el empleador, el cliente y los usuarios.

1.03. Aprobar el software sólo si tiene la creencia bien fundada de que es seguro, cumple las especificaciones, pasa las pruebas apropiadas, y no disminuye la calidad de vida, la privacidad o daña el ambiente. El efecto último del trabajo debe ser el bien público.

1.04. Revelar a las personas o autoridades apropiadas cualquier peligro real o potencial para el usuario, el público o el ambiente, que crean que se asocia razonablemente con el software o documentos relacionados.

1.05. Cooperar en los esfuerzos para abordar temas de preocupación pública grave ocasionada por el software, su instalación, mantenimiento, apoyo o documentación.

1.06. Ser justo y evitar el engaño en todos los terrenos, en particular en los públicos, que conciernen al software o documentos relacionados, métodos y herramientas.

1.07. Considerar temas de discapacidades físicas, asignación de recursos, desventaja económica y otros factores que disminuyan el acceso a los beneficios del software.

1.08. Estar motivado para dirigir de manera voluntaria sus habilidades profesionales a buenas causas, y contribuir a la educación del público respecto de la disciplina.

Principio 2: EL CLIENTE Y EL EMPLEADOR

Los ingenieros de software deben actuar en nombre del mejor interés de su cliente y empleador, y de modo consistente con el interés público. En particular, los ingenieros de software deben hacer lo siguiente, por ser apropiado:

2.01. Brindar servicio en sus áreas de competencia, ser honesto y franco en cuando a las limitaciones de su experiencia y educación.

2.02. No utilizar software obtenido o retenido de manera ilegal o poco ética, si conoce su origen.

2.03. Usar la propiedad de un cliente o empleador sólo en las formas autorizadas, y con el conocimiento y consentimiento de aquéllos.

2.04. Asegurarse de que cualquier documento en que se base haya sido aprobado, cuando se lo requiera, por alguien autorizado para ello.

2.05. Mantener privada y confidencial la información obtenida en su trabajo profesional, donde dicha confidencialidad sea consistente con el interés público y consistente con la ley.

2.06. Identificar, documentar, obtener evidencias e informar al cliente o empleador con rapidez, si es probable que, en su opinión, un proyecto fracase, sea demasiado caro, viole las leyes de la propiedad intelectual o cause problemas de otro tipo.

2.07. Identificar, documentar y reportar a su empleador o cliente aspectos significativos de preocupación social de los que esté al tanto, en el software o los documentos relacionados.

2.08. No aceptar en el exterior trabajos que vayan en detrimento del que realiza para su empleador original.

2.09. No promover ningún interés adverso para su empleador o cliente, a menos que esté comprometido un interés ético mayor; en ese caso, informar al empleador o a otra autoridad apropiada de la preocupación ética.

Principio 3: EL PRODUCTO

Los ingenieros de software deben asegurarse de que sus productos y modificaciones relacionadas cumplan los estándares profesionales más elevados posibles. En particular, deben hacer lo siguiente, por ser apropiado:

3.01. Luchar por lograr una calidad elevada, costo aceptable y programación razonable, asegurándose de que los intercambios significativos están claros y son aceptados por el empleador y el cliente, y están disponibles para la consideración del usuario y el público.

3.02. Garantizar las metas y objetivos apropiados y asequibles para cualquier proyecto en el que trabaje o se proponga hacerlo.

3.03. Identificar, definir y abordar los aspectos éticos, económicos, culturales, legales y ambientales relacionados con los proyectos del trabajo.

3.04. Garantizar que está calificado para cualquier proyecto en que trabaje o se proponga hacerlo, con la combinación adecuada de educación, capacitación y experiencia.

3.05. Garantizar que se usa un método apropiado para cualquier proyecto en que trabaje o se proponga trabajar.

3.06. Trabajar para seguir los estándares profesionales, cuando se disponga de ellos, más apropiados para la tarea en cuestión, y los abandone sólo cuando haya una justificación ética o técnica.

3.07. Esforzarse por entender por completo las especificaciones del software en que trabaje.

3.08. Garantizar que las especificaciones del software en que se trabaje se hayan documentado bien, satisfagan los requerimientos del usuario y tengan las aprobaciones indicadas.

3.09. Asegurar estimaciones cuantitativas realistas de costo, programación, personal, calidad y resultados de cualquier proyecto en que trabaje o se proponga laborar, y dar una evaluación de la incertidumbre de dichas estimaciones.

3.10. Garantizar la prueba, depuración y revisión adecuada del software y los documentos relacionados en que trabaje.

3.11. Asegurar la documentación adecuada, inclusive los problemas significativos descubiertos y las soluciones adoptadas, para cualquier proyecto en que trabaje.

3.12. Trabajar para desarrollar el software y los documentos relacionados que respeten la privacidad de quienes serán afectados por el software.

3.13. Tener el cuidado de utilizar sólo datos exactos obtenidos por medios éticos y legales, y emplearlos sólo en las formas autorizadas.

3.14. Mantener la integridad de los datos, con sensibilidad a las ocurrencias caducas o defectuosas.

3.15. Tratar todas las formas de mantenimiento de software con el mismo profesionalismo que los desarrollos nuevos.

Principio 4: CRITERIO

Los ingenieros de software deben mantener la integridad e independencia en su criterio profesional. En particular, deben hacer lo siguiente, por ser apropiado:

4.01. Atemperar todos los criterios técnicos en aras de la necesidad de apoyar y mantener los valores humanos.

4.02. Avalar sólo los documentos preparados bajo su supervisión o dentro de sus áreas de competencia y con las que estén de acuerdo.

4.03. Mantener la objetividad profesional con respecto de cualquier software o documentos relacionados que se les pida evaluar.

4.04. No incurrir en prácticas financieras cuestionables, como el soborno, la doble facturación u otras prácticas financieras impropias.

4.05. Revelar a todas las partes involucradas aquellos conflictos de interés que no puedan evitarse o evadirse de manera razonable.

4.06. Negarse a participar, como miembros o asesores, en órganos privados, gubernamentales o profesionales, relacionados con aspectos de software, en los que ellos, sus empleadores o clientes tengan conflictos de interés manifiestos.

Principio 5: ADMINISTRACIÓN

Los gerentes de ingeniería de software y los líderes deben suscribir y promover un enfoque ético de la administración del desarrollo y mantenimiento de software. En particular, quienes administren o dirijan a los ingenieros de software deben hacer lo siguiente, por ser apropiado:

5.01. Garantizar la buena administración de cualquier proyecto en que trabajen, inclusive los procedimientos efectivos para la promoción de la calidad y la reducción del riesgo.

5.02. Asegurar que los ingenieros de software estén informados de los estándares antes de aceptar el trabajo.

5.03. Garantizar que los ingenieros de software conocen las políticas y procedimientos del empleador en cuanto a protección de claves, archivos e información confidencial para el empleador o para otros.

5.04. Asignar el trabajo sólo después de tomar en cuenta las contribuciones apropiadas de educación y experiencia atemperadas por el deseo de acceder a más educación y experiencia.

5.05. Asegurar estimaciones cuantitativas realistas del costo, programación, personal, calidad y resultados de cualquier proyecto en que trabajen o se propongan hacerlo, y dar una evaluación de la incertidumbre de dichas estimaciones.

5.06. Atraer ingenieros de software en potencia sólo por medio de hacer la descripción completa y exacta de las condiciones de empleo.

5.07. Ofrecer una remuneración justa y adecuada.

5.08. No impedir de manera injusta que alguien asuma un puesto para el que está calificado.

5.09. Garantizar que hay un acuerdo justo acerca de la propiedad de cualquier software, procesos, investigaciones, escritura u otra propiedad intelectual a la que haya contribuido el ingeniero de software.

5.10. Participar en el proceso debido en las audiencias por violaciones que haga la política de un empleador respecto de este Código.

5.11. No pedir a un ingeniero de software que haga algo inconsistente con este Código.

5.12. No castigar a alguien por expresar sus preocupaciones sobre la ética de un proyecto.

Principio 6: LA PROFESIÓN

Los ingenieros de software deben hacer avanzar la integridad y reputación de la profesión en consistencia con el interés público. En particular, los ingenieros de software deben hacer lo siguiente, por ser apropiado:

6.01. Ayudar a crear un ambiente organizacional favorable a la actuación ética.

6.02. Promover el conocimiento público de la ingeniería de software.

6.03. Divulgar el conocimiento de la ingeniería de software con la participación en organizaciones profesionales, reuniones y publicaciones.

6.04. Dar apoyo, como miembros de la profesión, a otros ingenieros de software que se esfuercen por apegarse a este Código.

6.05. No promover en su propio interés a costa del de su profesión, cliente o empleador.

6.06. Obedecer todas las leyes que gobiernen su trabajo, a menos que por circunstancias excepcionales dicho cumplimiento sea inconsistente con el interés público.

6.07. Ser exacto al establecer las características del software en el que trabajen, evitando no sólo las afirmaciones falsas sino también aquellas de las que pueda suponerse en forma razonable que son especulativas, vacías, engañosas, equívocas o dudosas.

6.08. Tomar responsabilidad para detectar, corregir y reportar errores en el software y documentos asociados en que trabajen.

6.09. Garantizar que los clientes, empleadores y supervisores conocen el compromiso que tienen los ingenieros de software con este Código de ética, y las ramificaciones subsecuentes de dicho compromiso.

6.10. Evitar las asociaciones con negocios y organizaciones que estén en conflicto con este código.

6.11. Reconocer que las violaciones de este Código son inconsistentes con ser un profesional de la ingeniería de software.

6.12. Expresar sus preocupaciones a las personas involucradas cuando se detecten violaciones de este Código, a menos que esto sea imposible, contraproducente o peligroso.

6.13. Reportar las violaciones significativas de este Código a las autoridades correspondientes, cuando esté claro que la consulta con las personas involucradas en las faltas sea imposible, contraproducente o peligrosa.

Principio 7: LOS COLEGAS

Los ingenieros de software deben ser justos y dar apoyo a sus colegas. En particular, los ingenieros de software deben hacer lo siguiente, por ser apropiado:

7.01. Invitar a sus colegas a apegarse a este Código.

7.02. Ayudar a sus colegas en su desarrollo profesional.

7.03. Dar todo el crédito por el trabajo de otros y evitar tomar un crédito indebido.

7.04. Revisar el trabajo de otros en forma objetiva, desinteresada y bien documentada.

7.05. Escuchar en forma justa las opiniones, preocupaciones o quejas de un colega.

7.06. Ayudar a sus colegas a estar al tanto de las prácticas laborales estándares, inclusive las políticas y procedimientos para proteger las claves, archivos y otra información confidencial, así como de las medidas de seguridad en general.

7.07. No intervenir injustamente en la carrera de algún colega; sin embargo, la preocupación por el empleador, cliente o interés público puede hacer que los ingenieros de software cuestionen, de buena fe, la competencia de algún colega.

7.08. En situaciones fuera de sus propias áreas de competencia, solicitar las opiniones de otros profesionales competentes en ese campo.

Principio 8: LA PERSONA

Los ingenieros de software deben participar en la educación continua, respecto de la práctica de su profesión, y deben promover un enfoque ético de su práctica. En particular, los ingenieros de software deben esforzarse permanentemente por lo siguiente:

8.01. Aumentar su conocimiento de los desarrollos en el análisis, especificación, diseño, desarrollo, mantenimiento y prueba del software y documentos relacionados, así como la administración del proceso de desarrollo.

8.02. Mejorar su aptitud de crear software seguro, confiable y útil a un costo razonable y en un tiempo adecuado.

8.03. Mejorar su capacidad de producir documentación exacta, informativa y bien escrita.

8.04. Mejorar su comprensión del software y documentos relacionados con el que trabajan, y del ambiente en que se utilizará.

8.05. Mejorar su conocimiento de los estándares relevantes y de la ley que rige sobre el software y documentos relacionados con los que trabajen.

8.06. Mejorar su conocimiento de este Código, su interpretación y aplicación a su trabajo.

8.07. No dar un tratamiento injusto a nadie debido a prejuicios irrelevantes.

8.08. No influir en otros para que emprendan acciones que involucren la violación de este Código.

8.09. Reconocer que las violaciones personales de este Código son inconsistentes con ser un profesional de la ingeniería de software.

Este Código fue elaborado por la fuerza de tarea conjunta ACM/IEEE-CS para la Ética de la Ingeniería de Software y Prácticas Profesionales (SEEPP):

Comité Ejecutivo: Donald Gotterbarn (director), Keith Miller y Simon Rogerson;

Miembros: Steve Barber, Peter Barnes, Ilene Burnstein, Michael Davis, Amr El-Kadi, N. Ben Fairweather, Milton Fulghum, N. Jayaram, Tom Jewett, Mark Kanko, Ernie Kallman, Duncan Langford, Joyce Currie Little, Ed Mechler, Manuel J. Norman, Douglas Phillips, Peter Ron Prinzivalli, Patrick Sullivan, John Weckert, Vivian Weil, S. Weisband y Laurie Honour Werth.

14.5 Resumen del capítulo

La computarización ha creado situaciones nuevas que desafían los códigos morales de la sociedad. La tecnología de la computación genera tentaciones debido a factores tales como la velocidad, privacidad, anonimato, facilidad de copiado, atracción estética, disponibilidad de víctimas potenciales, alcance internacional, y el poder de destruir. Internet está muy desregulado debido a que no está sujeto a las reglas de un país único y a su cultura. Proporciona un foro para la libre expresión, pero abusos tales como la pornografía y la retórica del odio son agresivos. Otro aspecto es la protección de la propiedad intelectual tanto para Internet como para el contenido de las bases de datos y el software. El movimiento del Software Libre desafía el concepto del software como propiedad. La privacidad individual está comprometida por el amplio uso de la tecnología de cómputo para compartir información personal almacenada en bases de datos. Los oponentes de la legislación sobre la privacidad afirman que compartir los datos de los clientes entre las empresas es una forma de libre expresión. También están en debate las responsabilidades de los profesionales de la computación para hacer que sus productos sean más fáciles y seguros para los usuarios.

La **propiedad intelectual** se refiere a los productos de la mente, como la literatura, música, arte, arquitectura, inventos, fórmulas para productos y software. El fundamento de los derechos de la propiedad intelectual es que como el creador de una obra original ha invertido tiempo y recursos, él o ella tienen derecho a una retribución justa, que a su vez estimula a que la gente creativa produzca nuevas obras. La propiedad intelectual está protegida por varios mecanismos, inclusive el derecho de autor, patentes, secreto comercial, marcas registradas y marcas de servicio.

El derecho de autor (**copyright**) lo da el gobierno del país de origen de una obra durante un periodo limitado durante el cual el autor tiene derechos exclusivos para hacer copias, publicar, distribuir, exhibir o ejecutar en público la obra o usarla como base de un trabajo derivado. La obra debe ser original, tener forma tangible y estar fija en un medio. Cualquiera que cree una obra que pudiera ser protegida por el derecho de autor posee automáticamente éste, aun sin publicación, símbolo del copyright o registro formal. Cualquiera que desee usar la obra protegida debe obtener permiso del propietario y posiblemente pagar una tarifa, a menos que se trate de un **uso justo**. El uso justo permite el empleo limitado no personal de los materiales protegidos, incluso de una cantidad pequeña para fines educativos y otros usos similares. Los cuatro factores por considerar son el carácter del uso, la naturaleza de la obra, la cantidad de trabajo por utilizar y el efecto que tendría en el mercado de la obra

original. Los hechos, ideas y fórmulas siempre son de dominio público, pero un arreglo específico o expresión de ellos, como el que se encuentra en una base de datos, es susceptible de recibir la protección del derecho de autor. Los derechos de autor tienen reconocimiento internacional, pero cada país hace sus propias leyes al respecto. La mayor parte de éstas se basan en la **Convención de Berna**. En Estados Unidos, para obras creadas antes de 1978, los derechos eran para 95 años, mientras que las obras posteriores a ese año están protegidas durante la vida del autor más 70 años, y los trabajos por contrato están cubiertos de 95 a 120 años. La ley del derecho de autor ampara el software, tanto en código fuente como en código objeto. La **Ley de Protección de los Derechos de Autor Digitales del Milenio** tiene algunas provisiones controversiales relacionadas con las herramientas diseñadas para burlar los sistemas de protección o encriptado, y hace que el uso, manufactura u oferta de tales herramientas sea un delito. Hay excepciones para las bibliotecas, archivos e instituciones educativas para fines de evaluar la obra a fin de determinar si se adquiere una copia de ella. También hay excepciones para las instituciones que velan por el cumplimiento de la ley, para la ingeniería inversa a fin de que permita la operación entre productos, para la investigación en encriptado, reparaciones de computadoras y usos similares. La ley también limita la responsabilidad de los proveedores del servicio de Internet por las violaciones del derecho de autor que cometan sus suscriptores.

Otra forma de protección de la propiedad intelectual son las **patentes**, que garantizan el derecho de la propiedad para un inventor. Las patentes las emiten los gobiernos de países individuales. Una patente da al inventor el derecho de impedir que otros fabriquen, utilicen, ofrezcan a la venta, vendan o importen el invento. El término de una patente por lo general es de 20 años, tiempo durante el cual el inventor tiene la oportunidad de recuperar los costos de la creación de su invento, el cual debe ser útil, nuevo y no obvio. El inventor debe llenar una solicitud que incluya las especificaciones y describa el invento con todo detalle.

Un **secreto comercial** es cualquier información que se utilice en la operación de un negocio y que se mantenga en secreto y brinde una ventaja competitiva. En Estados Unidos, los secretos comerciales están protegidos por los estados. Estas leyes protegen a la compañía de la apropiación ilegal del secreto, y hacen de esto un delito. La protección permanece mientras la información sea un secreto. Algunos factores que usan los tribunales para determinar si algo es un secreto comercial son la amplitud con que se conozca la información dentro y fuera de la compañía, lo estricto de las medidas que se tomen para guardar el secreto, el valor de la información para el propietario y los competidores, la cantidad de tiempo, esfuerzo y dinero gastado en su desarrollo y la dificultad que tendrían otros para duplicarlo exactamente.

Las **marcas registradas** y las **marcas de servicio** también son elegibles para su custodia bajo las leyes de protección intelectual. Las marcas incluyen letras, números, palabras, dibujos, símbolos, colores, sonidos, fragancias, formas o empaques distintivos de los bienes.

El software y su paquetería pueden ser protegidos por las leyes de derechos de autor, patentes, marcas registradas o alguna combinación de éstas. Un cliente que compre software en realidad adquiere una **licencia de uso**, no el software en sí, que sigue siendo propiedad del dueño. La **fuente abierta** es libre y disponible para ser copiada. Otra categoría de software es el **shareware (software compartido)**, que permite que las personas lo descarguen y utilicen por un periodo de evaluación, después del cual pagan para continuar empleándolo. El software de un sistema de administración de base de datos comercial por lo general está protegido con los mismos métodos de protección de la propiedad intelectual usados con otras clases de software. Las aplicaciones personalizadas escritas para una base de datos también califican para las protecciones mencionadas. Las bases de datos en sí mismas también son susceptibles de recibir distintas clases de protección de la propiedad intelectual. Por ejemplo, la base de datos tiene protegidos sus derechos de autor porque es una expre-

sión, aun cuando los hechos que contiene no puedan protegerse. La información en una base de datos también puede ser tratada como secreto comercial, ya que se utiliza para obtener una ventaja competitiva.

La tecnología de bases de datos hace posible almacenar cantidades enormes de datos sobre las personas. Las tecnologías para buscar y comunicar datos podrían usarse para compilar y compartir información sobre los individuos sin su conocimiento o consentimiento. Los aspectos de si es ético recabar ciertos datos sobre una persona, usarla sin su consentimiento y compartirla con terceros, son temas de mucho debate, y las prácticas en Estados Unidos y la Unión Europea difieren de manera significativa. Hay un conflicto entre el derecho de un individuo a la privacidad y el deseo de los gobiernos y empresas de tener información que les resulte útil. La privacidad se reconoce como un derecho humano fundamental contenido en las constituciones de muchos países, en la declaración de los derechos humanos de las Naciones Unidas, en convenciones adoptadas por el Consejo Europeo, y en leyes aprobadas en Estados Unidos y en el mundo. Las leyes que regulan las actividades del gobierno en Estados Unidos incluyen la Freedom of Information Act, la Privacy Act de 1974 y la Computer Matching and Privacy Protection Act. La Patriot Act de 2002 ha debilitado las provisiones de algunas de las leyes sobre la privacidad, en aras de la seguridad nacional. Las leyes individuales se aprobaron para ordenar las prácticas en varios sectores de negocios. Algunos ejemplos son la Fair Credit Reporting Act de 1970, la Right to Financial Privacy Act de 1978, la Cable Privacy Protection Act de 1984, la Electronic Communication Privacy Act de 1986, la Video Privacy Protection Act de 1988, la Telephone Consumer Protection Act de 1991, la Health Insurance Portability and Accountability Act de 1996 y la Gramm Leach Bliley Financial Services Modernization Act de 1999. Por lo general, ellas requieren que los consumidores tengan un enfoque **opcional** para impedir que su información se comparta. Las leyes europeas son más restrictivas en las actividades de los negocios. Ellas requieren un enfoque **restrictivo** en lugar de uno opcional como el mecanismo para obtener la aprobación del cliente para compartir sus datos. Se basan en principios desarrollados por la Organización para la Cooperación y Desarrollo Económico (OECD), que incluye los principios de la limitación en la obtención, calidad de los datos, especificación del propósito, limitación en el uso, salvaguardas de seguridad, apertura, participación del individuo y disponibilidad. Los datos sobre ciudadanos de Estados Unidos deben abordarse en el mismo nivel de protección cuando salen del país. Por tanto, las compañías estadounidenses que hagan negocios en los países de la Unión Europea tendrán problemas para obtener los datos que necesitan para sus negocios, porque no podrían obtener de manera legal información de los clientes en esas naciones. Sin embargo, las compañías de Estados Unidos pueden certificar que siguen las reglas del acuerdo de **Puerto Seguro**, lo que significa que cumplen con las reglas de la Unión Europea respecto del tratamiento que dan a los datos, y así poder recabarlos.

Los **factores humanos** en el desarrollo del software se refieren a factores físicos y psicológicos que promuevan o faciliten el desempeño óptimo de parte del usuario. El diseño centrado en el usuario incrementa la comodidad de éste, la seguridad, productividad y satisfacción, y reduce los costos de la capacitación y los errores. Un enfoque de ingeniería del sistema de factores humanos incluye la consideración del usuario en cada etapa del proyecto de sistema. Los cinco componentes de la **usabilidad** son el aprendizaje, eficiencia, memorización, reducción de errores y satisfacción.

Las organizaciones profesionales de la computación han desarrollado códigos de conducta que proveen guías para los profesionales de la computación y el software. El Código de Ética y Conducta Profesional de ACM, describe las prácticas para todos los profesionales de la computación, y ACM e IEEE han creado en forma conjunta un código de ética y práctica profesional para orientar a los ingenieros de software.

Ejercicios

14.1 Defina los términos siguientes.

a. propiedad intelectual
b. derechos de autor (copyright)
c. patente
d. secreto comercial
e. marca registrada
f. marca de servicio
g. infringir
h. fuente abierta
i. software libre
j. bulto blackstoniano
k. uso justo
l. dominio público
m. Convención de Berna
n. WIPO
o. TRIPPS
p. Ley de Protección de los Derechos de Autor Digitales del Milenio
q. Freedom of Information Act
r. Privacy Act de 1974
s. identificador universal
t. Patriot Act de 2002
u. Gramm Leach Bliley Financial Services Modernization Act de 1999
v. opcional
w. OECD
x. controlador de datos
y. Puerto Seguro
z. usabilidad

14.2 Escoja una de las leyes mencionadas en este capítulo e investigue sus orígenes, provisiones e historia legislativa. Sintetice los argumentos a favor y en contra de ella. Prepare una presentación que describa sus descubrimientos.

14.3 Elija una de las convenciones o acuerdos internacionales mencionados en este capítulo e investigue su historia, provisiones, los países participantes y su relación con la legislación estadounidense en la misma área. Sintetice los argumentos a favor y en contra del acuerdo. Si fuera el presidente de Estados Unidos, ¿suscribiría el acuerdo? Prepare una presentación que describa sus descubrimientos.

14.4 Investigue el movimiento del software de fuente abierta y lea sobre su base filosófica. Escriba un ensayo para defender su posición al respecto.

14.5 Suponga que usted y un grupo de amigos han desarrollado un software innovador para el que desea la protección de la propiedad intelectual. Identifique el mecanismo que escogería y describa lo que necesitaría hacer (si hubiera algo) para establecer su

propiedad del software y hacer ésta obligatoria. ¿Se reconocerían sus derechos en otros países? Si no fuera así, ¿habría algún mecanismo que pudiera usar para que fueran reconocidos en el extranjero?

14.6 Suponga que tiene el puesto de administrador de la base de datos en una corporación grande. La compañía ha estado recabando datos sobre sus empleados, inclusive con la vigilancia de sus hábitos de trabajo por medio del registro de las teclas que oprimen, tomando el tiempo de sus conversaciones telefónicas con los clientes y revisando sus correos electrónicos en busca de correspondencia personal. Como ABD, se pide que colabore en el desarrollo de un sistema de registro para guardar dichos datos. ¿Tiene derecho la compañía para efectuar esa clase de vigilancia? ¿Cuál sería la responsabilidad profesional de usted en esa situación? ¿Los códigos de ética de ACM e IEEE tienen provisiones que lo orienten al respecto?

CAPÍTULO 15

Almacenes de datos (Data Warehouse) y minado de datos (Data Mining)

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Cómo se originaron los almacenes de datos
- Cómo difieren los almacenes de datos, de las bases de datos operativas
- Qué tipos de procesamiento soportan los almacenes de datos
- Cómo difiere OLAP de OLTP
- La arquitectura básica de un almacén de datos
- Qué modelos de datos se usan para almacenes de datos
- El significado de los términos rollup (exploración superficial), drill-down (exploración minuciosa), cross-tabulation (tabulación cruzada) y slice y dice (proyectar en dimensiones y sobre una dimensión)
- Cómo se expresan en SQL las consultas de los almacenes de datos
- Cómo se pueden usar los índices de mapa de bits y los índices combinados
- Cómo se manejan la materialización de vistas y el mantenimiento de vistas
- El propósito del minado de datos
- Los tipos de conocimiento que puede producir el minado de datos

- Cómo se usan los árboles de decisión, la regresión, las redes neuronales y el clustering (agrupamiento)
- Algunas áreas de aplicación para el minado de datos

15.1 Orígenes de los almacenes de datos

Muchas organizaciones que usan tecnología de base de datos estándar para recopilar, almacenar y procesar grandes cantidades de sus datos operativos comenzaron a ver más de cerca sus almacenes de datos actuales e históricos como fuentes de información para ayudarse a tomar mejores decisiones empresariales. Han desarrollado almacenes de datos para **sistemas de apoyo de decisiones** (**DSS**, por sus siglas en inglés) y aplicaciones similares. Decisiones como dónde abrir un nuevo almacén, a qué audiencia dirigirse para una campaña publicitaria, a cuáles clientes otorgar préstamos y cuándo ordenar artículos adicionales se pueden hacer con más confianza cuando se basan sobre un cuidadoso examen de los patrones encontrados en los datos existentes. Los proveedores de DBMS, incluidos Oracle e IBM, rápidamente agregaron características a sus líneas de productos para permitir el almacenamiento de los datos a partir de sus sistemas de base de datos estándar. Se han desarrollado nuevas y poderosas herramientas analíticas para extraer más información de los datos almacenados en tales almacenes. SQL:1999 contiene extensiones que apoyan las funciones requeridas por los almacenes de datos. Los datos en un almacén de datos con frecuencia se recopilan a partir de varios departamentos o sitios que pertenecen a una gran empresa. El término lo acuñó W. H. Inmon, quien describió un almacén de datos como "una colección de datos orientada a sujeto, integrada, no volátil, variable en el tiempo que se utiliza principalmente en toma de decisiones organizacionales" (Inmon, 2002). Un almacén de datos se establece para aplicaciones que apoyan decisiones, y no para procesamiento de transacciones ordinarias. Está optimizado para recuperación de datos, en oposición a procesamiento de transacciones.

15.2 Bases de datos operativas y almacenes de datos

Las bases de datos operativas tradicionales soportan **procesamiento de transacción en línea** (**OLTP**, por sus siglas en inglés), que de manera característica involucran un número limitado de transacciones repetitivas, cada una de las cuales afecta algunas tuplas en un momento en una base de datos relacional. Una base de datos como ésta se desarrolla para servir a las necesidades de información de los usuarios finales, y está diseñada para soportar sus operaciones empresariales diarias. La alta disponibilidad y desempeño eficiente son factores cruciales en el éxito de una base de datos operativa. Debe proporcionar apoyo para un gran volumen de transacciones y entregar respuestas a las consultas de los usuarios u otras operaciones en línea en un marco temporal corto. Una base de datos operativa se actualiza en tiempo real, conforme ocurren las transacciones del negocio. Actualizaciones, inserciones y borrados se deben realizar rápidamente para mantener la base de datos en un estado que refleje el entorno actual de la empresa.

En contraste, los almacenes de datos soportan **OLAP** (On-Line Analytical Processing: procesamiento analítico en línea), así como toma de decisiones. Los datos en un almacén de datos se pueden llevar directamente de bases de datos operativas múltiples, en diferentes periodos (datos históricos) y también pueden incluir datos de otras fuentes, datos resumidos y metadatos. Las fuentes pueden tener distintos modelos o estándares, pero el almacén de datos integra los datos de modo que los usuarios ven un modelo consistente. El almacén de datos por lo general contiene una cantidad muy grande de datos, y está optimizado para procesamiento eficiente de consultas y presentación de resultados para apoyo de decisiones. Las actualizaciones no son tan frecuentes como lo son en las bases de datos operativas, pero se realizan de manera periódica. Las aplicaciones OLAP por lo general deben pasar a través de grandes cantidades de datos para producir resultados. Los analistas examinan los datos almacenados en el almacén usando consultas complejas, que generalmente involucran ope-

radores de agrupación y agregación. Pueden hacer análisis en series de tiempo usando datos históricos. El **minado de datos** es el proceso de descubrir nueva información mediante búsqueda de grandes cantidades de datos. El propósito es descubrir patrones o tendencias en los datos que serán útiles para la organización.

15.3 Arquitectura de un almacén de datos

A diferencia de una base de datos operativa, para la cual se pueden especificar requisitos por adelantado, un almacén de datos debe diseñarse para apoyar consultas *ad hoc* y nuevos y no anticipados tipos de análisis. En la figura 15.1 se muestra una arquitectura típica. Los datos se toman de fuentes de datos, que pueden incluir bases de datos operativas múltiples, otras entradas como archivos independientes y datos ambientales como información geográfica o datos financieros. Los datos se deben extraer de las fuentes con el empleo de herramientas externas al sistema que puedan acomodar las diferencias entre las fuentes heterogéneas. Los datos se reformatean en un formato consistente. Los datos también se pueden verificar para integridad y validez, un proceso llamado **limpieza de datos**, para asegurar su calidad antes de cargarlos en el almacén. Luego los datos se ponen en el modelo de datos para el almacén y carga. El proceso de carga es una transacción larga, pues por lo general está involucrado un gran volumen de datos, de modo que el sistema debe usar herramientas de gestión de transacción para garantizar recuperación adecuada en el evento de falla durante la transacción de carga. El sistema de gestión de base de datos que soporta al almacén de datos tiene un catálogo de sistema que almacena metadatos, así como otros componentes del sistema de la base de datos. Luego el almacén de datos se usa para soportar consultas para OLAP, con el fin de proporcionar información para sistemas de apoyo de decisiones que usan los administradores para toma de decisiones estratégicas, y a fin de

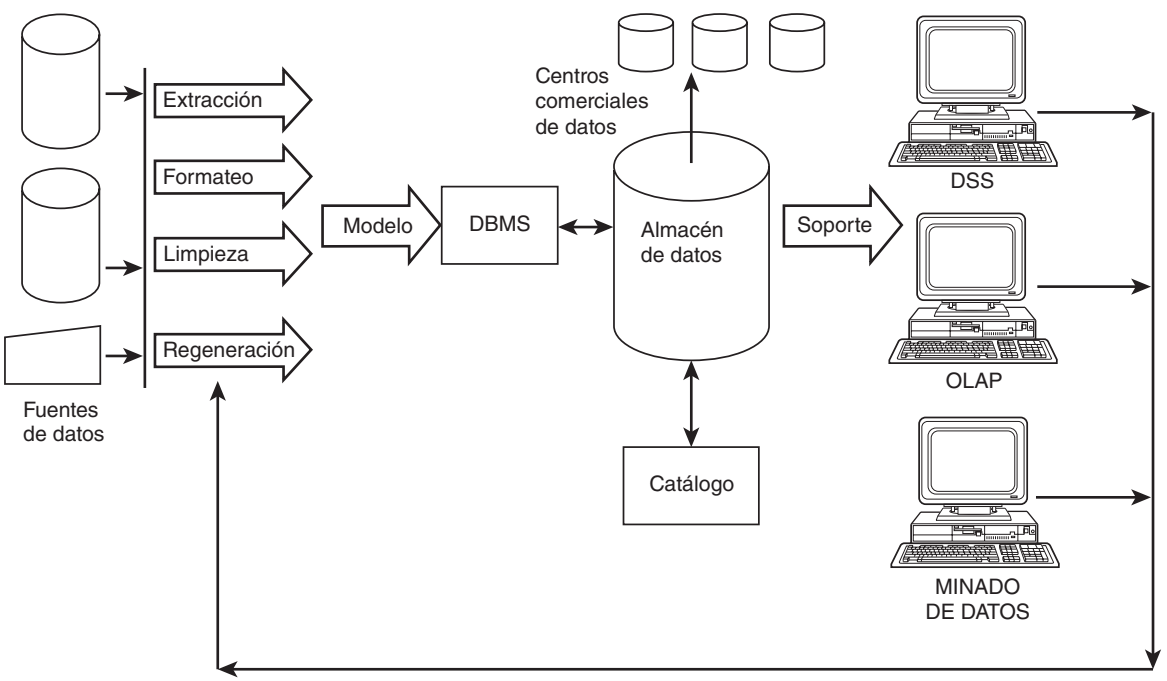


FIGURA 15.1

Arquitectura de un almacén de datos

proporcionar los datos para las herramientas de minado de datos que descubren nueva información acerca de patrones en los datos. Ciertos segmentos de los datos están organizados en subconjuntos llamados **data marts** ("mercado" de datos, subconjunto de información de un Dataware House), que se enfoca en sujetos específicos. Por ejemplo, un data mart podría contener información especializada acerca de un solo departamento dentro de la organización. Todos estos usos pueden resultar en nuevo conocimiento, que luego se puede usar como una fuente de datos desde la que se pueden formatear datos y ponerlos en el almacén. Los datos de todas las fuentes se deben refrescar de manera periódica. Si hay suficiente espacio de almacenamiento, los nuevos datos simplemente se agregan al almacén existente, y los datos antiguos se mantienen en tanto sean útiles. De otro modo, los datos que ya no se usan más se purgan de manera periódica, y se agregan nuevos datos. La frecuencia y ámbito de las actualizaciones depende del entorno. Los factores que se deben considerar para decidir la política de actualización incluyen cuánto almacenamiento está disponible, si el almacén necesita datos recientes, si puede estar fuera de línea durante la regeneración, y cuánto tardará el proceso para transmitir los datos, limpieza, formateo, carga y construcción de índices. La política usual es hacer una regeneración parcial de manera periódica.

15.4 Modelos de datos para almacenes de datos

Aunque los modelos en las fuentes de datos pueden variar, el almacén de datos en sí debe usar un solo modelo consistente que aloje las necesidades de los usuarios. Los almacenes de datos generalmente usan un modelo multidimensional. Los datos se pueden considerar como residentes en una matriz multidimensional llamada **cubo de datos**. La figura 15.2(a) muestra un cubo de datos tridimensional llamado Ventas. En el frente, la cara visible del cubo, se ven cifras de ventas para el mes de junio en cuatro departamentos (lácteos, abarrotes, vegetales, farmacia) en tres supermercados (store 1, store 2, store 3). Note que todas las tiendas tienen estos mismos cuatro departamentos. Las cifras pueden representar ventas en miles de dólares. Los datos de ventas para cada mes aparecen en forma de hoja de cálculo, que es un estilo de presentación que sería familiar para muchos usuarios. La tercera dimensión en este ejemplo es el tiempo, como se indica mediante las etiquetas para los meses de junio, julio, agosto, septiembre y octubre. Las cifras de ventas de cada mes para cada departamento en cada supermercado aparecen en una celda en la matriz tridimensional. Los usuarios pueden ver los datos por cualquier dimensión de interés para ellos. Por ejemplo, si un usuario quiere ver datos acerca de las ventas para cada departamento, el cubo puede **pivotear** o rotar para mostrar una dimensión de interés diferente, como se muestra en la figura 15.2(b). Aquí, la cara visible del cubo muestra las ventas del departamento de lácteos para los meses de junio a octubre para cada uno de los tres supermercados. Si rota sobre otro eje, podría examinar datos para cada tienda. La cara frontal del cubo mostraría, para una sola tienda, las ventas para cada departamento durante cada mes.

En un modelo multidimensional, es posible crear un nivel de granularidad más grueso al combinar o agregar datos, un proceso llamado **rollup** (exploración superficial). Por ejemplo, con el cubo de datos que se muestra en la figura 15.2(a), podría realizar un rollup sobre departamento, al combinar datos de todos los departamentos de cada tienda, para dar las ventas totales de cada tienda por cada mes, como se muestra en la figura 15.3(a). Se podría realizar un rollup sobre tiendas, lo que da los datos por departamento, sin importar la tienda, para cada mes. De igual modo, podría realizar un rollup sobre el mes. Una granularidad diferente de rollup se generaría al combinar los meses en temporadas, lo que muestra las ventas de la temporada de verano por departamento y tienda en una forma de hoja de cálculo, e igualmente para otoño, invierno y primavera.

FIGURA 15.2(a)
Cubo de datos de ventas

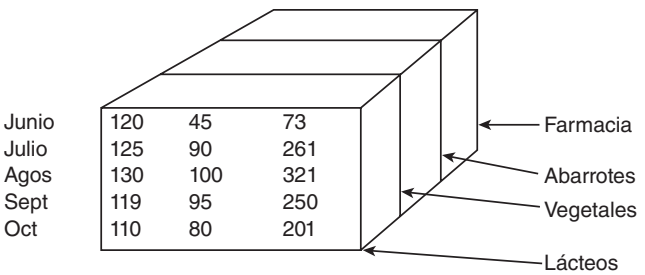


FIGURA 15.2(b)
Cubo de datos de ventas pivoteado

El proceso inverso es **drill-down** (exploración minuciosa). En este proceso se proporciona más detalle sobre cierta dimensión, usando granularidad más fina para los datos. Por ejemplo, podría realizar un rollup sobre departamento, lo que muestra los subdepartamentos o categorías en cada uno de los departamentos. La figura 15.3(b) proporciona un ejemplo en el que los departamentos se detallan en categorías para el mes de junio. El resto del cubo mostraría el mismo grado de detalle para los otros meses. Estos datos no se podrían obtener a partir del cubo de datos que se muestra en la figura 15.2(a), pues requiere que se almacenen datos más detallados, o que sea posible obtenerlo a partir de los datos almacenados en el almacén de datos.

Cuando el pivoteo y/o rollup de un cubo de datos da por resultado una presentación bidimensional estilo hoja de cálculo, es natural agregar totales para las filas y columnas, lo que forma una **tabulación cruzada** (cross-tabulation), como se ilustra en la figura 15.3(c). Este ejemplo muestra una tabulación cruzada de las cifras de ventas para tienda y mes, que muestra totales para cada tienda, totales para cada mes y el total para ambos. Con el pivoteo se podrían obtener tabulaciones cruzadas por departamento y tienda, o por departamento y mes.

Si examina una porción del cubo de datos con el uso de una selección donde especifique igualdad de condiciones para una o más dimensiones, este tipo de operación también se llama **slice** (proyectar en dimensiones), el cubo de datos, porque parece como si el usuario hubiera cortado a través del cubo en la dirección seleccionada. Por ejemplo, para el cubo de datos que se muestra en la figura 15.2(a), si especifica la condición `WHERE month = 'Julio'`, obtendría la hoja de cálculo para dicho mes, al tomar una proyección del cubo. Para el cubo rotado en la figura 15.2(b), si escribe `WHERE department = 'abarrotes'`, obtendría una proyección similar del cubo. Una operación adicional, llamada **dicing** (seleccionar sobre una dimensión), se realiza si especifica un rango de valores en una selección. Por ejemplo, si proyecta el cubo de datos en la figura 15.2(a) al especificar el mes de julio,

FIGURA 15.3(a)

Rollup de ventas sobre departamento

	Junio	Julio	Agos	Sept	Oct
Store1	459	351	340	345	360
Store2	336	340	334	330	370
Store3	636	650	645	599	700

FIGURA 15.3(b)

Drill-down de ventas sobre departamento que muestra el mes de junio

		Store 1	Store 2	Store 3
Farmacia				
	Vitaminas	25	85	53
	Cuidado de la piel	20	25	75
	Cuidado del cabello	50	40	25
	Remedios estomacales	15	20	50
	Remedios para resfriado	5	8	25
	Primeros auxilios	5	7	25
Abarrotes				
	Artículos enlatados	20	20	50
	Artículos horneados	40	20	50
	Artículos de papel	30	4	30
	Artículos de limpieza	25	1	10
	Detergentes	5	1	5
	Hogar	11	1	5
Vegetales				
	Verduras	40	30	100
	Frutas	48	29	60
Lácteos				
	Leche	90	40	65
	Mantequilla	10	2	4
	Queso	5	1	1
	Yogurt	5	1	1
	Helado	10	1	2

FIGURA 15.3(c)

Tabulación cruzada de ventas por tienda y mes

	Junio	Julio	Agos	Sept	Oct	Total
Store1	459	351	340	345	360	**1 855**
Store2	336	340	334	330	370	**1 710**
Store3	636	650	645	599	570	**3 100**
Total	**1 431**	**1 341**	**1 319**	**1 274**	**1 300**	***6 665***

puede seleccionar aún más al especificar `WHERE store = 'store2' OR store = 'store3'`.

No hay razón para limitar los datos en un cubo de datos a dos o tres dimensiones. Los diseñadores pueden almacenar fechas usando tantas dimensiones como quieran, si dichas dimensiones son de interés para ellos. Sin embargo, más allá de la tercera dimensión, no se puede dibujar una representación física de los cubos de datos. Los cubos de estas dimensio-

nes superiores se conocen como **hipercubos**. Todavía es posible aplicar el proceso de pivoteo, rollup y drill-down a los hipercubos.

Los primeros almacenes de datos almacenaban los datos usando arreglos multidimensionales, lo que crea sistemas **OLAP multidimensionales (MOLAP)**. Si en vez de ello se usa un modelo relacional, el sistema se describe como un sistema **OLAP relacional (ROLAP)**. Un almacén ROLAP consiste en múltiples tablas relacionales.

Un esquema ampliamente usado para almacenes de datos es un **esquema estrella**. Hay una tabla central de datos en bruto, llamada la **tabla de hechos**, que almacena datos observados no agregados. La tabla de hechos tiene algunos atributos que representan dimensiones y otros atributos dependientes que son de interés. Cada dimensión se representa mediante su propia tabla, y las **tablas de dimensiones** se pueden considerar como los puntos de una estrella cuyo centro es la tabla de hechos. Por ejemplo, en la figura 15.4(a), la tabla de hechos, ORDER (pedido), se muestra en el centro. Cada tupla de la tabla de hechos proporciona información acerca de un pedido, que involucra la venta de un producto a un cliente en un día particular. Para cada pedido, la tabla de hechos registra el `orderNumber` (número de pedido), `productNumber` (número de producto), `customerId` (identificación del cliente), `salespersonId` (identificación del vendedor), `date` (fecha), `unitPrice` (precio unitario) y `numberOrdered` (cantidad pedida). Hay tablas de dimensión para el producto, cliente, vendedor y fecha. Cada una de éstas se podría usar como una dimensión en un hipercubo, lo que permite plantear consultas acerca de las ventas de un producto parti-

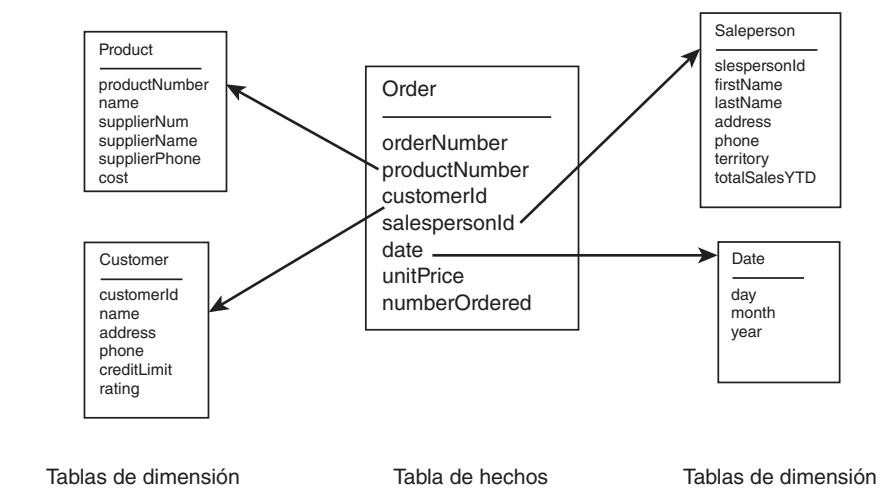


FIGURA 15.4(a)
Un esquema estrella

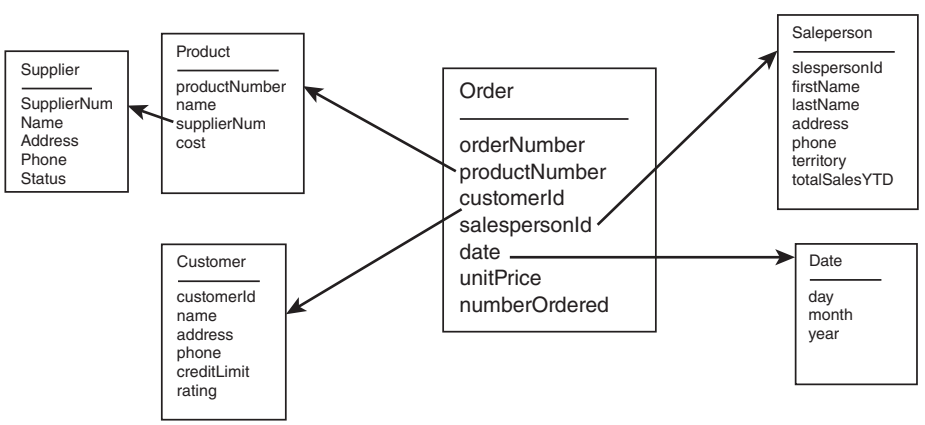


FIGURA 15.4(b)
Un esquema en copo de nieve

cular, las ventas a un cliente particular, las ventas hechas por un vendedor específico o las ventas hechas en una fecha específica. Las tablas de dimensión (`Product`, `Customer`, `Salesperson` y `Date`) se muestran en relación con sus correspondientes atributos de dimensión de la tabla de hechos. Por lo general, dichos atributos son claves externas en la tabla de hechos. Las tablas de dimensión proporcionan información adicional acerca de cada dimensión. Por ejemplo, la tabla de dimensión `Salesperson` proporciona más detalles acerca del vendedor que tomó el pedido. Los atributos restantes de la tabla de hechos, `orderNumber`, `unitPrice` y `numberOrdered`, son atributos dependientes. Una variación del esquema estrella es el **esquema copo de nieve** (snowflake), en el que las tablas de dimensión tienen ellas las mismas dimensiones, porque están normalizadas. Por ejemplo, en la figura 15.4(b) se muestra la dimensión `Product` que tiene una tabla de dimensión `Supplier` (proveedor). De la tabla `Product` se removieron todos los atributos del proveedor, excepto `supplierNum` (número de proveedor), pues dichos atributos son funcionalmente dependientes de `supplierNum`, y se les colocó en una tabla `Supplier` separada. Si alguna de las otras tablas no estuviese normalizada se habría hecho lo mismo con ellas, lo que crea un copo de nieve con la tabla de hechos como el centro.

15.5 Consultas de almacén de datos y extensión OLAP SQL:1999

Las funciones de agregación de SUM, COUNT, MAX, MIN y AVG son parte del estándar SQL92. Estas funciones se pueden usar en algunas consultas limitadas de proyección y selección dimensional para un almacén de datos. La forma general de tal consulta es:

```
SELECT <atributos de agrupamiento> <función de agregación>
FROM <tabla de hechos>, <tabla(s) de dimensiones>
WHERE <atributo = constante> . . . <atributo = constante>
GROUP BY <atributos de agrupamiento>;
```

Por ejemplo, para la tabla de hechos `Order` que se muestra en la figura 15.4(a) con esquema

```
Order (orderNumber, productNumber, customerId, salespersonId, date,
unitPrice, numberOrdered)
```

y la tabla de dimensión `Product` con esquema

```
Product (productNumber, name, supplierNum, supplierName, supplierPhone, cost)
```

podría escribir

```
SELECT productNumber, SUM(numberOrdered)
FROM Order O, Product P
WHERE O.productNumber = P.productNumber AND
       P.supplierNum = 'S101'
GROUP BY P.productNumber;
```

que da una lista de los productos suministrados por S101 y el número total de cada uno. Note que, para esta consulta, se proyectó sobre `Product` y se seleccionó sobre `supplierNum`.

Las consultas típicas para almacenes de datos requieren funciones de agregación adicionales. SQL:1999 incluye las funciones **stddev** (desviación estándar) y **variance** (varianza) para atributos sencillos. Éstas son medidas estadísticas estándar que indican cuán "disperso" del promedio está un conjunto de valores de datos. Por ejemplo, una desviación estándar alta para ventas mensuales totales en el departamento de lácteos significaría que la cantidad de ventas fluctúa ampliamente de mes a mes, mientras que una desviación estándar baja indicaría que las ventas permanecen bastante constantes a lo largo del periodo. Funciones esta-

dísticas adicionales son **correlation** (correlación) y **regression** (regresión), que se aplican a pares de atributos.

SQL:1999 también tiene funciones para calcular **rank** (clasificación) para valores de datos. Por ejemplo, si la tabla `Salesperson` tiene esquema

```
Salesperson(salespersonId, firstName, lastName, address, phone, territory,
totalSalesYTD)
```

tal vez quiera encontrar la clasificación del vendedor de acuerdo con su `totalSalesYTD` (ventas totales del año a la fecha). El comando

```
SELECT salespersonId, rank( ) over(order by (totalSalesYTD) desc)
FROM Salesperson
WHERE lastName = 'Jackson';
```

desplegará `salespersonId` y `rank` del vendedor Jackson. Si dos personas tienen la misma cantidad de ventas reciben la misma clasificación, pero se salta el siguiente valor de clasificación. Por ejemplo, si dos vendedores están empatados en las ventas más altas, ambos se clasifican como número 1, pero la siguiente persona se clasifica como número 3. Si no quiere saltarse números, puede usar la opción dense_rank() en lugar de la opción rank(). Note que la frase "orden por" en la línea SELECT simplemente dice cuál atributo usar para la clasificación. No despliega las tuplas en el orden de la clasificación. Si quiere que los resultados de todos los vendedores se desplieguen con quien tiene la clasificación más alta primero, en lugar del orden de las tuplas existentes, debe agregar una línea ORDER BY, como en:

```
SELECT salespersonId, rank( ) over(order by (totalSalesYTD) desc) as
salesRank
FROM Salesperson
ORDER BY salesRank;
```

La cláusula GROUP BY también se extendió en SQL:1999 con opciones especiales **CUBE** y **ROLLUP** para cubos de datos. Por ejemplo, si tiene una tabla de hechos `Sales` con esquema

```
Sales(storeNumber, departmentName, month, amount)
```

y tablas de dimensión para tienda, departamento y mes para los datos de supermercado representados por el cubo de datos en la figura 15.2, la consulta

```
SELECT departmentName, storeNumber, month, sum(amount)
FROM Sales
GROUP BY CUBE(departmentName, storeNumber, month);
```

produce las ocho posibles combinaciones GROUP BY para los tres atributos departamento, tienda y mes. El agrupamiento se realizará por Departamento solo, por tienda sola, por mes solo, por la combinación de tienda y mes, la combinación de tienda y departamento, la combinación de mes y departamento, la combinación de las tres y de ninguna. En contraste con el cubo, que da el agrupamiento sobre todas las posibles combinaciones de atributos mencionados, la opción ROLLUP permite al usuario especificar cuáles atributos se usarán para agregación. Si escribe

```
SELECT departmentName, storeNumber, month, sum(amount)
FROM Sales
GROUP BY ROLLUP(departmentName, storeNumber);
```

se agrega sobre todos los pares de valores `departmentName` y `storeNumber`, y también sobre `departmentName` solo, mas no sobre `storeNumber` solo.

15.6 Técnicas de indexado

La cantidad de datos en un almacén puede ser tan grande que son importantes los índices eficientes para que las consultas se ejecuten en una cantidad razonable de tiempo. Dado que los datos en un almacén de datos no se actualizan mediante transacciones ordinarias, es relativamente estático. Por tanto, una vez creados los índices para los datos, el costo de mantenerlos no es un factor. Al entorno de almacén de datos se aplican técnicas de indexado especiales, incluidos el indexado de mapa de bits y el indexado combinado, que son eficientes para una gran cantidad de datos estáticos.

Los **índices de mapa de bits** se pueden construir para cualquier atributo que tenga un número limitado de distintos valores posibles. Son especialmente adecuados si el dominio es pequeño. Para cada valor en el dominio se construye un vector de bits para representar dicho valor, al colocar un 1 en la posición para dicho valor. Por ejemplo, la figura 15.5 muestra la tabla `Faculty` con un índice de mapa de bits para `rank` y otro para `department`. Puesto que el primer registro `Faculty` tiene una clasificación de profesor, la primera fila del índice de clasificación tiene un 1 en la columna `Professor`. Puesto que el departamento del primer registro `Faculty` es `Art`, hay un 1 en la columna `Art` de la primera fila del índice departamento. Los índices de mapa de bits toman mucho menos espacio que los índices estándares y permiten procesamiento eficiente de algunas consultas directamente del índice. Por ejemplo, para la consulta SQL,

```
SELECT COUNT(*)
FROM Faculty
WHERE rank = 'Professor' AND department = 'Art';
```

los dos índices se pueden comparar al construir un nuevo vector de bits para usar un AND lógico de bits, que compara el cuarto bit del vector clasificación con el primer bit del vector departamento. El número de unos en el nuevo vector de bits es el número de profesores en el departamento de Arte. Procedimientos similares se pueden usar para acelerar otras operaciones, incluida la combinación.

FIGURA 15.5(a)
La tabla Faculty

Faculty			
facId	**name**	**department**	**rank**
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

FIGURA 15.5(b)
Un índice de mapa de bits para Rank

Instructor	Assistant	Associate	Professor
0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Art	CSC	History	Math
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	1	0	0

FIGURA 15.5(c)
Un índice de mapa de bits para Department

Puesto que la combinación es difícil cuando las tablas son grandes, los almacenes de datos pueden usar índices combinados para acelerar las consultas combinadas. La mayoría de las operaciones combinadas se realizan sobre claves externas. En un almacén de datos que use un esquema estrella, la operación combinación con frecuencia involucra comparar la tabla de hechos con las tablas de dimensión. Un **índice de combinación** relaciona los valores de una tabla de dimensión con las filas de la tabla de hechos. Para cada valor del atributo indexado en la tabla de dimensión, el índice almacena las ID de tuplas de todas las tuplas en la tabla de hechos que tenga dicho valor.

15.7 Vistas y materialización de vistas

Las vistas juegan un importante papel en los almacenes de datos, como lo hacen en las bases de datos operativas, al personalizar el entorno del usuario. Los operadores SQL, incluidos los operadores de extensión OLAP de CUBE y ROLLUP, pueden realizarse tanto en vistas como en tablas base. El comando SQL CREATE VIEW simplemente define la vista, y no crea alguna tabla nueva. Cuando se escribe una consulta en una vista, una forma como se puede ejecutar es a través de **modificación de consulta**, que sustituye la referencia en la línea WHERE mediante la definición de vista. Por ejemplo, puede tener una vista, `BestSalespeople` (mejor vendedor), definida por

```
CREATE VIEW BestSalespeople
AS SELECT S.salespersonId, S.lastName, S.firstName, S.phone, S.salesYTD
FROM Salesperson S
WHERE S.salesYTD > (SELECT AVG(salesYTD)
                    FROM Salesperson);
```

Si escribe una consulta usando la vista como

```
SELECT SUM (B.salesYTD)
FROM BestSalespeople B;
```

entonces, si usa la modificación de consulta, la referencia en la línea FROM se sustituye por la definición de vista, que resulta en:

```
SELECT SUM (B.salesYTD)
FROM (SELECT S.salespersonId, S.lastName, S.firstName, S.phone, S.salesYTD
   FROM Salesperson S
   WHERE S.salesYTD > (SELECT AVG(salesYTD)
                       FROM Salesperson) )AS B;
```

En un entorno de almacén de datos, donde consultas y vistas son muy complejas y donde los analistas usan el sistema en un entorno interactivo, la modificación de consulta puede resultar en demoras inaceptables en el tiempo de respuesta. Por tanto, un método alternativo de manejar las vistas es **materializarlas**, precalcularlas a partir de la definición y almace-

narlas para uso posterior. Para acelerar aún más el procesamiento, se pueden crear índices para las vistas materializadas.

El diseñador del almacén de datos debe considerar cuáles vistas materializar, y ponderar las restricciones de almacenamiento contra los posibles beneficios de acelerar las consultas más importantes. El diseñador también debe decidir acerca de la política de mantenimiento para las vistas materializadas. Cuando cambian las tablas base subyacentes, la vista también se debe actualizar. Esto se puede hacer como parte del proceso de actualización para las tablas base, una política llamada **mantenimiento de vista inmediato**, que hace que el proceso de regeneración se vuelva lento para el almacén de datos. Una alternativa es usar **mantenimiento de vista diferido**. Son posibles muchas políticas, incluidas:

- **Regeneración lenta**, que actualiza la vista cuando una consulta que usa la vista se ejecuta y la versión materializada actual es obsoleta
- **Regeneración periódica**, que actualiza la vista a intervalos regulares
- **Regeneración forzada**, que actualiza la vista después de realizar un número específico de actualizaciones a las tablas base subyacentes

El proceso de regeneración se puede realizar al recalcular toda la vista materializada. Sin embargo, para vistas complejas, de manera especial aquellas con combinaciones o agregados, esto puede ser muy costoso. En vez de ello, la regeneración se puede hacer de manera incremental, e incorporar solamente los cambios hechos a las tablas subyacentes.

15.8 Minado de datos

Minado de datos (data mining) significa descubrir nueva información a partir de conjuntos de datos muy grandes. Por lo general, el conocimiento descubierto está en la forma de patrones o reglas. Además de la tecnología de base de datos, el minado de datos usa técnicas de los campos de estadística e inteligencia artificial, de manera especial del aprendizaje de máquinas. Puesto que involucra grandes cantidades de datos es necesario tener una base de datos grande o un almacén de datos. El minado de datos se puede usar con una base de datos operativa, siempre que sea lo suficientemente grande. Antes se describió el minado de datos como una de las aplicaciones primarias para un almacén de datos, junto con OLAP y sistemas de apoyo de decisiones. Un almacén de datos que se use como fuente para el minado de datos debe incluir datos resumidos, así como datos en bruto tomados de las fuentes de datos originales tales como las bases de datos operativas. Puesto que los tipos de operaciones utilizados en el minado de datos difieren de las analíticas para OLAP y los sistemas de apoyo de decisiones, la aplicación de minado de datos debe considerarse en el diseño original del almacén. El minado de datos por lo general requiere conocimiento del dominio (por ejemplo, el entorno de la empresa a estudiar), así como conocimiento del proceso de minado de datos. El formato de datos requerido con más frecuencia es un "archivo plano" en el que todos los datos para cada caso de valores observados aparecen como un solo registro, en lugar de relacional u orientado a objeto. Si los datos no están representados como casos individuales, o si se necesitan combinaciones para colocar juntos todos los datos de un caso, se debe emplear considerable esfuerzo a fin de preparar los datos y reformatearlos. Por tanto, el diseño del almacén debe crearse teniendo en cuenta el minado de datos.

15.9 Propósito del minado de datos

Para la mayoría de las empresas, el propósito final del minado de datos es proporcionar conocimiento que dará a la compañía una ventaja competitiva, lo que permite ganar un

mayor beneficio. Las compañías usan minado de datos con la esperanza de que podrán lograr lo siguiente:

- Predecir el comportamiento futuro de los atributos. Por ejemplo, al estudiar los datos de los tres supermercados en el ejemplo anterior, probablemente sea posible predecir cifras de ventas para el mismo periodo el próximo año. Si se tiene una base de datos de salud pública que contenga datos acerca de la diseminación de influenza durante los cinco inviernos pasados, es probable que se pueda predecir el número de tales infecciones para el invierno venidero.
- Clasificar ítems al colocarlos en las categorías correctas. Por ejemplo, dados los datos acerca del valor crediticio de clientes en el pasado, puede clasificar a un nuevo cliente como con valía crediticia o sin valía crediticia. La clasificación también se usa en medicina, para determinar cuál de los muchos posibles diagnósticos es el apropiado para un paciente, con base en datos pasados acerca de otros pacientes y síntomas.
- Identificar la existencia de una actividad o un evento. Por ejemplo, si se conocen los patrones típicos de las ventas de acciones que estuvieron presentes en casos previos de comercio interno, y se ve que el mismo patrón ocurre una vez más, puede ser posible identificar un caso actual de comercio interno. Las compañías aseguradoras estudian patrones y características de reclamos previos conocidos como fraudulentos para determinar cuáles nuevas reclamaciones pueden ser fraudulentas.
- Optimizar el uso de los recursos de la organización. El minado de datos puede modelar escenarios para ayudar a determinar la mejor colocación de equipo, la forma más lucrativa para invertir dinero o la forma más eficiente para usar el tiempo disponible con la finalidad de maximizar la productividad o alcanzar alguna otra meta.

15.10 Tipos de conocimiento descubierto

En los sistemas expertos, el conocimiento se obtiene con el uso de deducción lógica. El motor de inferencia del sistema experto se usa para aplicar las leyes de la lógica a los hechos almacenados en una base de datos para deducir nuevos hechos en una forma mecánica. El minado de datos usa **inducción** en lugar de deducción. Examina un gran número de casos y concluye que existe un patrón o una regla. El conocimiento se puede representar en varias formas, incluso como reglas, árboles de decisión, redes neuronales o marcos.

Los resultados pueden incluir:

- **Reglas de asociación**, que tienen la forma $\{x\} \Rightarrow \{y\}$, donde x y y son eventos que ocurren al mismo tiempo. Un ejemplo típico involucra pares de productos que los clientes frecuentemente compran juntos, usando datos de canasta de mercado, que muestra cuáles ítems se compraron para una transacción. Por ejemplo, si un cliente compra pan es probable que compre mantequilla al mismo tiempo. La regla se puede expresar como una implicación:

  ```
  {pan} ⇒ {mantequilla}
  ```

  Note que tanto el lado izquierdo (pan en este ejemplo) como el lado derecho (aquí, mantequilla), pueden ser conjuntos de ítems en vez de ítems individuales. Dos medidas importantes conectadas con las reglas de asociación son **cobertura** y **precisión**. Para un conjunto de ítems, la cobertura es el porcentaje de transacciones en el conjunto de datos que contienen todos estos ítems incluidos en los lados izquierdo y derecho. Note que la transacción puede incluir ítems adicionales que no son parte de la asociación. Por ejemplo, si se tiene un millón de registros de ventas, y 100 000 de ellas incluyen tanto pan como mantequilla, la cobertura para la regla pan ⇒ mantequilla

es 10%. La precisión es una medida de con cuánta frecuencia la regla prueba ser verdadera; esto es, para los casos donde el lado izquierdo de la implicación está presente, precisión es el porcentaje de aquellos en los cuales el lado derecho también está presente. Para un millón de registros de ventas, acaso 500 000 incluyan pan, pero sólo 100 000 de las que incluyen pan también incluyen mantequilla, de modo que la precisión en la regla es de 20%.

- **Reglas de clasificación.** Clasificación es el problema de colocar instancias en la correcta de varias posibles categorías. El sistema se desarrolla para proporcionar un conjunto de instancias pasadas para las cuales se conoce la clasificación correcta, llamada **conjunto de formación**. Con estos ejemplos, el sistema desarrolla un método para clasificar de manera correcta un nuevo ítem cuya clase en la actualidad se desconoce. Un ejemplo clásico de una regla de clasificación es el problema de decidir cuáles clientes tendrán crédito garantizado, con base en factores como ingreso, propiedades y similares.
- **Patrones secuenciales.** Una aplicación típica de los patrones secuenciales es la predicción de que un cliente que compra un producto particular en una transacción seguirá con la compra de un producto relacionado en otra transacción. Por ejemplo, una persona que compra una impresora en una tienda de cómputo probablemente comprará papel en una visita posterior. Tales patrones se representan como secuencias. La secuencia {impresora}{papel} representa dos visitas por el mismo cliente en la que se observa el patrón secuencial, esto es: el cliente compró una impresora en la primera visita y papel en una visita posterior. El porcentaje de veces que ocurre tal secuencia en todo el conjunto de transacciones de venta es el apoyo para el patrón. A la primera subsecuencia {impresora} se le conoce como **predictor** de la segunda subsecuencia {papel}. La precisión para esta predicción es la probabilidad de que, cuando ocurra {impresora} en una visita {papel}, ocurrirá en una visita posterior. Esta probabilidad se puede calcular al examinar los datos en bruto de las transacciones de ventas observadas. Los patrones secuenciales pueden involucrar más de un ítem en cada subsecuencia, y más de una subsecuencia. En términos generales, si se encuentra que la secuencia S_1, S_2, . . . , S_n, donde cada S_i es un conjunto de ítems, es válida, entonces S_1 es un predictor desde S_2 hasta S_n.
- **Patrones de series de tiempo.** Una **serie de tiempo** es una secuencia de eventos que son todos del mismo tipo. Por ejemplo, si las ventas totales para un supermercado se calculan y registran al final de cada mes a través de un largo periodo, estas medidas constituyen una serie de tiempo. Los datos de series de tiempo se pueden estudiar para descubrir patrones y secuencias. Por ejemplo, puede observar los datos y encontrar el periodo más largo cuando las cifras de ventas continúan elevándose cada mes, o encontrar el declive más pronunciado de un mes al siguiente. Los precios de acciones, tasas de interés, tasas de inflación y muchas otras cantidades se pueden analizar usando series de tiempo.

15.11 Métodos utilizados

15.11.1 Árboles de decisión

Un método para desarrollar reglas de clasificación es elaborar un **árbol de decisión**, cuyo ejemplo se muestra en la figura 15.6. Este árbol se usa para tomar decisiones acerca de si se acepta un estudiante en una universidad, con base en una predicción de si el estudiante será un estudiante malo, promedio, bueno o excelente. Considera el promedio de bachillerato del estudiante y su calificación total en la prueba de aptitud escolar (SAT). Si el promedio

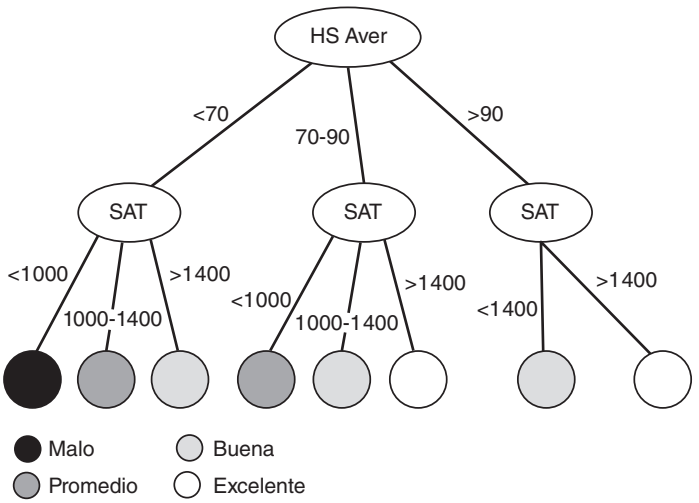


FIGURA 15.6
Un árbol de decisión

está por abajo de 70% y el SAT por abajo de 1 000, la predicción es que el aspirante será un mal estudiante en la universidad. En el otro extremo, si el promedio está por arriba de 90 y el SAT por arriba de 1 400, el sistema predice que el estudiante será excelente. El árbol se construiría inicialmente al examinar los registros de estudiantes pasados, y considerar sus características de ingreso de `HSAver` (promedio de bachillerato) y `SAT`. Estos atributos se llaman **atributos de partición**, porque permiten la descomposición o partición del conjunto de instancias de formación en clases desarticuladas. En las ramas se muestran las **condiciones de partición**. Note que las condiciones no tienen que ser idénticas a todas las ramas. Por ejemplo, si el promedio de bachillerato es arriba de 90, sólo se consideran calificaciones SAT arriba o abajo de 1 400, y no se usa el valor 1 000, como se hizo para las otras ramas.

15.11.2 Regresión

La regresión es un método estadístico para predecir el valor de un atributo Y (llamado variable dependiente), dados los valores de los atributos $X_1, X_2, \ldots, X_n$ (llamados variables independientes). Un ejemplo es usar las calificaciones SAT y el promedio de bachillerato del estudiante como variables independientes para predecir su promedio acumulado de calificaciones al final de los cuatro años de universidad, que es la variable dependiente. Muchos paquetes de software estadístico permiten a los usuarios identificar los potenciales factores que son útiles para predecir el valor de la variable dependiente. Al usar **regresión lineal**, el paquete encuentra entonces la contribución o peso de cada variable independiente para la predicción, en la forma de los coeficientes $a_0, a_1, \ldots, a_n$ para una función lineal que tenga la forma

$$Y = a_0 + a_1 X_1 + a_2 X_2 + \ldots + a_n X_n$$

La fórmula derivada representa una curva que ajusta los valores observados tan cercanamente como sea posible.

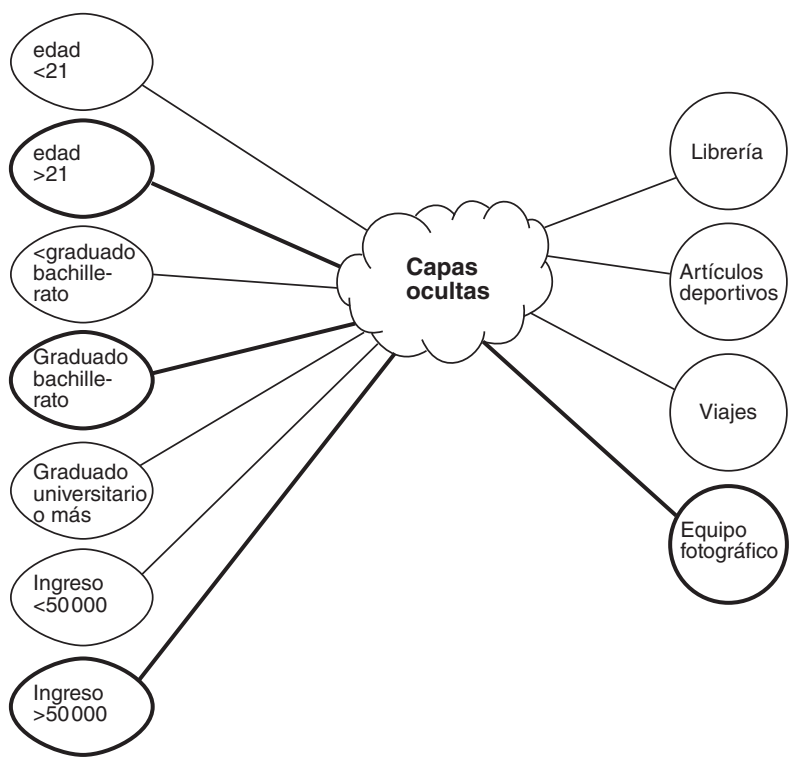
En el minado de datos, al sistema mismo se le puede pedir identificar las variables independientes, así como encontrar la función de regresión. Los sistemas de minado de datos también pueden usar regresión no lineal, con el uso de un enfoque de curva de ajuste, y encontrar la ecuación de la curva que ajusta los valores observados tan cercanamente como sea posible. También pueden lidiar con datos no numéricos.

15.11.3 Redes neuronales

Esta técnica incluye una variedad de métodos que usan un conjunto de muestras para todas las variables y así encontrar las relaciones más fuertes entre variables y observaciones. Los métodos se originaron en el campo de la inteligencia artificial. Usan una forma generalizada de regresión, con el uso de una técnica de curva de ajuste para encontrar una función a partir del conjunto de muestras. Las redes neuronales usan un método de aprendizaje, que se adapta conforme aprenden nueva información al examinar muestras adicionales. La figura 15.7 muestra un modelo muy simple de una red neuronal que representa compras de sitios en Internet. La meta del sistema es predecir cuáles clientes potenciales ordenarán de sitios Web específicos. En la predicción se usan muchas variables de entrada que involucran edad, educación e ingreso. La muestra evidencia que los graduados de bachillerato mayores de 21 años de edad, con ingreso mayor a 50 000 tienen más probabilidad de ordenar desde un equipo fotográfico desde un sitio en Internet. En la etapa de aprendizaje, a la red se le proporciona un conjunto de formación de casos que ofrecen hechos acerca de estos valores de entrada para una muestra de clientes, y también los sitios en Internet desde los que ordenó el cliente. Las capas ocultas se desarrollan por el sistema conforme examina estos casos, con el uso de técnicas de regresión generalizadas. Conforme se proporcionan casos adicionales, el sistema refina sus capas ocultas hasta que aprende a predecir correctamente cierto porcentaje del tiempo. Entonces se proporcionan casos de prueba para evaluar al sistema. Si se desempeña bien en los casos de prueba, el sistema se puede usar en nuevos datos donde el resultado se desconoce.

Las redes neuronales tienen varios inconvenientes que dificultan trabajar con ellas. Un gran problema es **sobreajustar** la curva. Los datos del conjunto de formación, como cualesquiera datos brutos reales, siempre tienen cierta cantidad de "ruido", inconsistencias o variaciones que en realidad no son significativas y que se deben ignorar. En vez de ello, la red puede

FIGURA 15.7
Modelo simple de una red neuronal



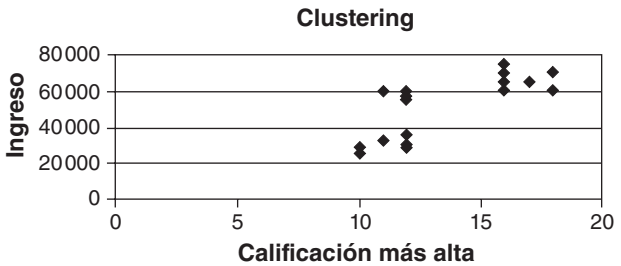


Figura 15.8
Clustering

acomodar su función de predicción para explicar dichos valores, lo que produce una curva que ajusta a la perfección los datos del conjunto de formación. Entonces la función de predicción se desempeñará deficientemente sobre datos nuevos. Un segundo problema es que el conocimiento de cómo el sistema hace sus predicciones está en las capas ocultas, de modo que los usuarios no tienen una buena visión del razonamiento empleado. A diferencia del modelo de regresión, donde los coeficientes muestran la aportación de cada atributo, los pesos asignados a los factores en el modelo no se pueden interpretar en una forma natural por parte de los usuarios. Aunque el modelo puede funcionar bien, el resultado puede ser difícil de entender e interpretar.

15.11.4 Clustering (agrupamiento)

El clustering (agrupamiento) o segmentación se refiere a los métodos utilizados para colocar tuplas en clusters o grupos que se pueden desarticular o traslapar. Con un conjunto de datos de formación, el sistema identifica un conjunto finito de clusters en los que las tuplas de la base de datos se pueden agrupar. Las tuplas en cada cluster son similares, comparten algunas propiedades y son diferentes a las tuplas en otros clusters. La similitud se mide con el uso de algún tipo de **función de distancia** que se define para los datos. Por ejemplo, si la edad es un atributo, la diferencia en edades de personas se podría usar como la función de distancia. En algunos problemas, las categorías se pueden organizar de manera jerárquica. La figura 15.8 muestra un ejemplo en el que las personas se agrupan por educación y niveles de ingreso. Muestra el grado más alto concluido por cada persona sobre el eje x y el ingreso anual de cada una en el eje y. A partir de los datos, aparece que existen tres clusters. Existen personas con bajo nivel educativo que tienen bajo ingreso, personas con bajo nivel educativo con ingreso moderadamente alto, y personas con alto nivel educativo con alto ingreso.

15.12 Aplicaciones del minado de datos

El minado de datos se usa en una amplia variedad de aplicaciones. Entre las más exitosas están las siguientes:

- **Venta al por menor**
  - **Customer relations management** (**CRM**, o administración de la relación con clientes) es una aplicación emergente del minado de datos. CRM permite a un minorista proporcionar una experiencia de venta personalizada a los clientes, con la finalidad de construir lealtad del cliente. El minado de datos se usa para identificar y anticipar las necesidades de los clientes, de modo que el minorista tendrá los ítems que solicite y será capaz de sugerir ítems relacionados que comprará el cliente. Por ejemplo, las librerías en Internet con frecuencia analizan las preferencias establecidas por el cliente y la historia de compra previa para sugerir nuevas compras, o para sugerir compras relacionadas con las actuales.

- La **administración de campañas de publicidad** usa minado de datos para identificar a los clientes que tendrán más probabilidad de realizar compras en respuesta a la publicidad. Para firmas de pedidos por correo se usa para identificar los hogares que responderán positivamente a los catálogos. Los datos pueden provenir de respuestas a ítems de correo directo previos o a ítems de prueba. El minado de datos identifica los factores que se asocian con una respuesta positiva para construir un modelo. Dicho modelo, que incluye datos demográficos y geográficos, se usa para identificar a las personas con más posibilidades de responder positivamente a nuevas campañas, con base en sus atributos demográficos y de otra índole.

- **Banca y finanzas**
  - **Calificaciones de crédito.** Minoristas, bancos y otros que extienden crédito a clientes pueden usar una variedad de técnicas de minado de datos para construir un modelo a fin de determinar si extienden crédito a nuevos solicitantes.
  - **Detección y prevención de fraudes.** El minado de datos se usa exitosamente para detectar transacciones fraudulentas con tarjetas de crédito ya sea después del hecho o en tiempo real, para negar la potencial transacción. Las compañías aseguradoras también usan minado de datos para determinar si los reclamos son fraudulentos. Las negociaciones con acciones se investigan en forma rutinaria para identificar posibles casos de negociación interna u otra actividad ilegal. La industria de telecomunicaciones también usa minado de datos para detectar uso fraudulento de sistemas telefónicos.
- **Fabricación**
  - **Optimización del uso de recursos.** Los modelos de minado de datos se pueden usar para determinar el mejor despliegue de equipo, recursos humanos y materiales.
  - **Optimización del proceso de fabricación.** El proceso de fabricación en sí se puede analizar para determinar la forma más efectiva en costo y/o eficiente para elaborar productos.
  - **Diseño de productos.** Al examinar datos acerca de defectos y reparaciones en los productos, los fabricantes pueden eliminar partes que es probable sean defectuosas, lo que mejora el diseño. Los resultados de investigación de mercado también se pueden usar para diseñar productos que preferirían los consumidores.
- **Medicina**
  - **Determinación de la efectividad de tratamientos.** Pacientes, proveedores de atención a la salud y aseguradoras médicas se benefician de poder determinar cuáles cursos de tratamiento son más efectivos. El minado de datos no sólo proporciona la base para acertados resultados estadísticos de estudios de varios tratamientos, sino también puede revelar factores ocultos que influyan en el resultado del tratamiento.
  - **Análisis del efecto de medicamentos.** Tanto antes como después de que nuevos medicamentos llegan al mercado, se debe recopilar y proporcionar una gran cantidad de datos acerca de los efectos de medicamentos. Los estudios de medicamentos a gran escala, que involucran enorme cantidad de datos, juegan un papel vital en la industria de atención a la salud.
  - **Descubrimiento de relaciones.** Es posible determinar los patrones entre atención del paciente y el resultado de los tratamientos. Además de las relaciones obvias entre el curso de tratamiento o medicamentos y los resultados del paciente, con el

minado de datos se pueden descubrir relaciones ocultas entre otros factores, como el escenario donde se proporciona la atención, y los resultados.

15.13 Resumen del capítulo

Los almacenes de datos guardan grandes cantidades de datos tomados de las bases de datos operativas utilizadas por una empresa, así como de otras fuentes de datos. Se usan para **sistemas de apoyo de decisiones (DSS)**, **procesamiento analítico en línea (OLAP)** y **minado de datos**. Los datos se toman de fuentes de datos con el uso de herramientas externas del sistema. Los datos extraídos se reformatean, limpian, ponen en el modelo adecuado y cargan en el almacén. También se pueden crear **data marts** que contienen datos especializados. Los almacenes usan frecuentemente un **modelo multidimensional**. Los datos se pueden representar con el uso de **cubos de datos** multidimensionales, que se pueden **pivotear** o rotar para mostrar una dimensión diferente. Si la dimensión es más que tres, se usa el término **hipercubo**. La exploración superficial (**rollup**) es un proceso de agregado de datos a lo largo de dimensiones, mientras que su inverso, la exploración minuciosa (**drill-down**), es un proceso que proporciona más detalle para alguna dimensión. Una **tabulación cruzada** es un despliegue en forma de hoja de cálculo con totales agregados a los datos.

La proyección dimensional (**slicing**) de un cubo de datos es equivalente a realizar una selección con igualdad de condiciones para una o más dimensiones, mientras que la selección dimensional (**dicing**) es equivalente a una selección de rango.

Los primeros sistemas OLAP multidimensionales almacenaban datos como arreglos multidimensionales, llamados sistemas **MOLAP**. Los sistemas OLAP relacionales, llamados **ROLAP**, usan tablas relacionales múltiples. Un **esquema estrella** usa una tabla central de valores de datos llamada **tabla de hechos**, con atributos que representan dimensiones. Cada dimensión tiene su propia **tabla de dimensión** que se conecta a la tabla de hechos. En una variación llamada **esquema copo de nieve**, las mismas tablas de dimensión tienen tablas de dimensión porque están normalizadas.

Las consultas para un almacén de datos pueden usar las funciones de agregación estándares de SQL, usualmente con opciones GROUP BY. SQL:1999 proporciona funciones adicionales para las medidas estadísticas de **desviación estándar**, **varianza**, **correlación** y **regresión**. También existe una función **rank** que regresa la clasificación de una tupla con respecto a algún atributo. La cláusula GROUP BY puede incluir la opción **GROUP BY CUBE** y **GROUP BY ROLLUP** para cubos de datos.

Es posible usar técnicas de indexado especiales en un entorno de almacén de datos para acelerar las consultas. Los **índices de mapa de bits** son útiles si el dominio de valores para un atributo es pequeño. Para cada tupla se construye un vector de bits, que muestra cuáles de los valores tiene la tupla al colocar un 1 en la posición apropiada del vector. Es posible responder algunas consultas directamente del índice, sin acceder a los registros de datos. Un **índice combinado** se construye al almacenar, para cada valor del atributo indexado en una tabla de dimensión, las ID de tupla de todas las tuplas en la tabla de hechos que tengan dicho valor para el atributo.

Por cuestiones de eficiencia, con frecuencia se crean vistas mediante **materialización de vista**, y se precalculan y almacenan para uso futuro. Para vistas materializadas también se pueden crear índices. Es necesaria una política de mantenimiento de vista para vistas materializadas. Puede ser inmediata o diferida. Si es diferida, la política de regeneración puede ser lenta, periódica o forzada.

Minado de datos significa descubrimiento de nueva información a partir de conjuntos muy grandes de datos. El propósito es ganar una ventaja competitiva al poder predecir compor-

tamiento, clasificar ítems, identificar una actividad o evento, u optimizar el uso de recursos. El conocimiento descubierto puede ser como **reglas de asociación**, que tienen medidas de cobertura y precisión. El conocimiento también puede expresarse como **reglas de clasificación**, **patrones secuenciales** o **patrones de series de tiempo**. El conjunto de datos que se utiliza para enseñar al sistema se llama **conjunto de formación**. Los métodos de minado de datos incluyen **árboles de decisión**, **regresión**, **redes neuronales** y **clustering** (agrupamiento). Las áreas de aplicación incluyen venta al por menor, banca y finanzas, fabricación y medicina.

Ejercicios

15.1 Defina los siguientes términos:

a. almacén de datos
b. sistema de apoyo de decisiones
c. OLAP
d. OLTP
e. minado de datos
f. data mart
g. limpieza de datos
h. cubo de datos
i. rollup
j. drill-down
k. pivoteo
l. slice y dice
m. índice de mapa de bits
n. índice combinado
o. materialización de vista
p. esquema estrella
q. regresión
r. red neuronal
s. clustering (agrupamiento)
t. árbol de decisión
u. regla de asociación con cobertura y precisión
v. conjunto de formación
w. serie de tiempo
x. sobreajuste
y. administración de la relación con clientes (CRM)

15.2 Sean las siguientes relaciones que representan una tabla de hechos, `Pedidos`, y tablas de dimensión asociadas `Clientes`, `Vendedor` y `Productos`.

Pedidos

CustId	prodId	salesPersonId	Cantidad
101	1	10	20
101	2	11	8
101	3	10	15
102	1	11	10
102	2	10	12
102	3	10	15
103	1	11	9
103	2	10	40
103	3	10	12

Clientes

custId	nombre	creditRating
101	Adams	10
102	Burke	12
103	Collins	15

Productos

prodId	nombre	Price
1	artefacto	3
2	tornillo	1
3	taladro	10

Vendedor

salesPersonId	lastName	firstName
10	Yates	Tom
11	Zorn	Steve

a. Dibuje un esquema estrella para estos datos.

b. Dibuje un cubo de datos similar al que se muestra en la figura 15.2, que muestre la cantidad de pedidos colocados por los clientes para productos a través del vendedor 10 como la cara visible. Sea cliente el eje x y producto el eje y en esta cara. El vendedor debe ser el eje z para la tercera dimensión. Si ningún cliente ha colocado pedido para un producto a través del vendedor 10, coloque un 0 en la celda.

c. Pivotee el cubo para mostrar pedidos colocados por clientes y vendedor, con producto como la tercera dimensión. La cara visible debe ser para el producto 1.

d. Pivotee el cubo para mostrar pedidos colocados por vendedor y producto. La cara visible debe ser para el cliente 101. Agregue totales para hacer de ésta una tabulación cruzada.

15.3 a. Para el cubo de datos del ejercicio 15.2(b) muestre el rollup sobre vendedor.

b. Suponga que existen dos tipos de artefactos, tornillos y taladros: pequeños y grandes. Muestre un drill-down sobre producto para el cubo de datos en el ejercicio 15.2(b).

c. Para el cubo de datos del ejercicio 15.2(b) muestre el slice que corresponda a la selección WHERE salesPersonId = 2.

d. Para el slice del ejercicio 15.3(c) realice un dice para el rango prodId < 2.

15.4 a. Para los datos en la tabla `Pedidos` del ejercicio 15.2, escriba índices de mapa de bits sobre `custId`, `prodId` y `salesPersonId`.

b. Diseñe una consulta que se pueda responder mediante el uso de los índices solos.

c. Escriba índices combinados para clientes, productos y vendedor.

15.5 En un supermercado se observaron las siguientes transacciones:

TransNo	CustId	Fecha	Hora	Canasta
1	10	Jun 06	10:00	{pan, leche, jugo}
2	11	Jun 06	10:10	{café, leche}
3	12	Jun 06	10:20	{pan, mantequilla, leche}
4	13	Jun 06	10:25	{pan, huevos}
5	14	Jun 06	10:30	{leche, jugo}
6	10	Jun 07	10:00	{mantequilla, carne}
7	11	Jun 07	10:05	{toallas de papel, jugo}
8	13	Jun 07	10:30	{mantequilla, toallas de papel}
9	14	Jun 07	10:45	{leche, pan}

a. ¿Cuál es la cobertura para la regla de asociación pan ⇒ leche? ¿Cuál es la precisión para esta regla?

b. ¿Cuál es la cobertura para el patrón secuencial {pan} {mantequilla}? ¿Cuál es la precisión para este patrón?

c. ¿Puede encontrar alguna otra asociación o patrón secuencial de interés en estos datos? Si es así, proporcione la cobertura y precisión para ellos.

d. Agregue algunos ítems o transacciones que produzcan una nueva regla de asociación con cobertura y precisión de al menos 50%.

e. Agregue algunas transacciones que produzcan un nuevo patrón secuencial con cobertura y precisión de al menos 50%.

APÉNDICE A: Organización física de datos

A.1 Organización de archivos

Las tareas de almacenar y recuperar registros en una base de datos se manejan mediante el sistema de gestión de base de datos y los métodos de acceso al sistema operativo. Por lo general, el usuario no está al tanto de los métodos que utiliza para ubicar y almacenar datos. Sin embargo, el administrador de la base de datos necesita estar familiarizado con la organización física de la base de datos. El ABD diseña la plantilla de datos y es posible que requiera elegir otras opciones físicas que afecten el desempeño de la base de datos. El propósito de esta sección es describir algunas de las técnicas de organización de archivos usadas en la gestión de la base de datos.

A.1.1 Medios de almacenamiento

Para almacenar, respaldar y procesar bases de datos se usan muchos medios.

1. Almacenamiento en disco
   Usualmente toda la base de datos se almacena en disco. A diferencia de la memoria principal, el almacenamiento en disco **no es volátil**, esto es, no se borra cuando el sistema de apaga. El disco es un dispositivo de almacenamiento de acceso directo (**DASD**, por sus siglas en inglés), lo que significa que a los datos se puede acceder en cualquier orden. Los discos vienen en varias formas y tamaños y, para usarse, se deben montar en **unidades de discos** (disqueteras, disk drives), que son dispositivos que permiten que los datos se almacenen y lean desde discos. Los discos pueden estar permanentemente montados en las unidades, o pueden ser portátiles. Aunque el disco es el tipo más común de almacenamiento para las bases de datos, tiene desventajas causadas por la tecnología. Aunque a los datos en un disco no les afectan las fallas en el sistema, ocurre un gran problema cuando fallan las unidades de disco y destruyen datos. Para discos magnéticos, las cabezas de lectura/escritura deben colocarse sobre la superficie del disco para la lectura o grabación de datos. El disco da vueltas bajo las cabezas de lectura/escritura. Problemas como **roturas de cabezas** ocurren debido al movimiento mecánico involucrado en el uso de los discos. Además, existen demoras para acceder a los datos causadas por el movimiento de las cabezas de lectura/escritura y el giro de los discos.
2. Cinta magnética
   La cinta magnética es un medio de almacenamiento no volátil que proporciona **acceso secuencial**, lo que significa que, para investigar un trozo de datos, es necesario pasar por los datos antes de ellos. La cinta no se usa para procesamiento ordinario de la base de datos, lo que usualmente requiere acceso directo. Sin

embargo, se usa de manera extensa para almacenar datos de archivo y para **respaldos** (backups). En el evento de que se destruya la copia del disco de la base de datos se usa la última cinta de respaldo para reconstruir la base de datos. La cinta también es uno de los medios más ampliamente usados para transferir datos de una organización a otra.

3. Memoria principal
   La memoria principal es **volátil**, lo que significa que se pierde cuando el sistema se apaga. Proporciona acceso directo a los datos almacenados en ella. La base de datos por lo general no encaja en la memoria principal, porque su tamaño es limitado y debe contener no sólo datos sino también programas de sistemas y programas de aplicaciones. No obstante, cualquier dato que se utilice debe llevarse primero hacia la memoria principal. Una porción de memoria principal, llamada **búfer**, se usa para contener los registros a procesar. Cuando se solicita un registro, el sistema comprueba si el registro ya está en el búfer. Si es así, simplemente pasa la dirección en el búfer al proceso que lo solicita. Si no, el sistema encuentra la ubicación en el disco donde se almacenó el registro y dirige la lectura del registro hacia el búfer. Por lo general lee más de un registro a la vez. Luego el sistema pasa la dirección en el búfer al proceso que lo solicita. Note que el registro todavía existe en el disco; el búfer contiene una copia. Acceder a los registros en el búfer muchas veces es más rápido que acceder a ellos en el disco. Por esta razón, los registros que se usan con más frecuencia a veces se mantienen en el búfer siempre que se use la base de datos.
4. Memoria caché
   El caché es una pequeña porción de la memoria principal que se construye con el uso de chips de memoria de muy alta rapidez. En las microcomputadoras, la mayor parte de la memoria principal consiste en **DRAM** (memoria de acceso aleatorio dinámico). Una pequeña porción de la memoria usa chips **SRAM** (memoria de acceso aleatorio estático) más costosos, que tienen tiempo de acceso más rápido. La SRAM es por naturaleza más rápida que la DRAM y, a diferencia de la DRAM, no necesita "refrescarse" o regenerarse antes de leerse. El resultado es acceso casi instantáneo, de modo que el procesador no espera los datos. Para usar caché de manera exitosa, el sistema tiene que adivinar cuáles datos necesitará a continuación e intentar asegurar que estén en el caché cuando se necesiten. Cuando se requieran los datos, el sistema busca primero en el caché, y luego en la DRAM más lenta. Mientras más alto sea el número de **impactos**, cuando el dato necesario realmente esté en el caché cuando se solicite, más rápido será el sistema. Existen procedimientos que usan los controladores de caché para decidir cuáles ítems conservar en el caché. Para una base de datos que use índices jerárquicos, como se describe en la sección A.2.4, los niveles superiores del caché deben conservarse en el caché siempre que la base de datos esté en uso.

A.1.2 Bloqueo de registros

Por lo general, el sistema lee más de un registro en el búfer, porque todos los datos se almacenan y transfieren en unidades de tamaño fijo llamadas **bloques** que usualmente pueden contener más de un registro. Un bloque aparece al programador como un conjunto contiguo de bytes en una sola pista de una sola superficie de disco. Un tamaño de bloque típico es un múltiplo de 1 024 bytes, pero un bloque también puede tener muchos miles de bytes. Por ejemplo, si la pista de un disco contiene 4 bloques, cada acceso de disco llevaría el contenido entero de aproximadamente un cuarto de pista al búfer. En ocasiones, a un bloque se le llama **página** o **registro físico**. Si se usa el término registro físico para hacer referencia a un bloque, entonces los registros de datos que se quieren (por ejemplo, registros estudiantiles, registros de empleados, registros de pedidos) deben referirse como **registros lógicos**.

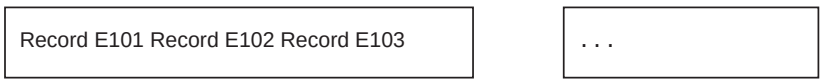


FIGURA A.1

Un bloque que contiene 3 registros lógicos

En general, entonces, un registro físico contendrá muchos registros lógicos. El número de registros lógicos en un bloque se llama **factor de bloqueo** para un archivo. La figura A.1 muestra un bloque con un factor de bloqueo de 3 (es decir: 3 registros por bloque). Toda la pista circular contendría 12 registros lógicos, pero sólo 4 físicos.

El bloqueo se realiza para ahorrar espacio. Para cada registro físico en una pista hay cierta cantidad de información adicional requerida. Al comienzo de cada registro físico existe un **header** que contiene muchos campos como información como dirección del registro físico, un indicador para decir si la pista está defectuosa, la longitud de la clave del registro, la longitud de los datos en el registro y dónde comienza el registro. Estos campos están separados por **espacios interregistro**, lo que aumenta aún más el número de bytes necesarios antes de encontrar los datos útiles. Si cada registro físico contiene sólo un registro lógico, habría un header para cada registro lógico. Cuando los registros se bloquean hay un solo header para cada bloque. La figura A.2(a) muestra 5 registros desbloqueados, mientras que la figura A.2(b) muestra 12 registros bloqueados, 3 por bloque en una pista que contiene 4 bloques. Puesto que el primer arreglo requiere 5 headers para los 5 registros lógicos (y físicos), mientras que el segundo sólo requiere 4 headers para los 12 registros lógicos, hay un ahorro en espacio de almacenamiento. La cantidad de espacio ahorrado depende del tamaño de los registros, el tamaño de los bloques, el tamaño de los espacios interregistro y la cantidad de información auxiliar requerida. Existe una negociación para este ahorro en almacenamiento. El sistema operativo siempre lleva un bloque completo al búfer, aun cuando el bloque pueda contener registros lógicos que no son necesarios para el proceso solicitante. Desde luego, si el ABD colocó en el mismo bloque registros que usualmente se procesan juntos, es probable que la siguiente solicitud sea para uno de los registros que ya están en el búfer, lo que por tanto ahorra otro acceso de disco. Para pasar la ubicación de búfer del registro lógico deseado al proceso solicitante, el sistema debe **desbloquear**, o dividir, el bloque en registros lógicos separados en el búfer. De igual modo, cuando se escriben registros, el sistema debe colocarlos en bloques en el búfer antes de escribirlos en el disco. Bloqueo y desbloqueo son procesos adicionales que se negocian para un uso más eficiente de almacenamiento. Por lo general, el ABD no tiene control sobre el tamaño de los bloques, pero puede ajustar el tamaño de los registros al tamaño del bloque. Si hay un deficiente ajuste entre tamaño de registro y tamaño de bloque, habrá mucho espacio desperdiciado en la base de datos.

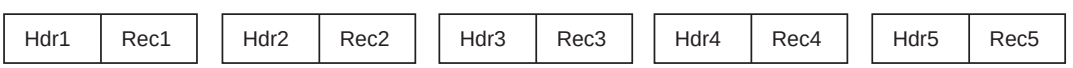


FIGURA A.2(a)

Registros desbloqueados con headers y espacios interregistro

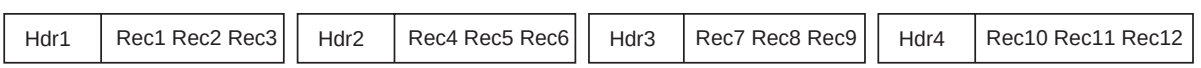


FIGURA A.2(b)

Registros bloqueados con factor de bloqueo de 3

FIGURA A.3(a)

Registros fragmentados

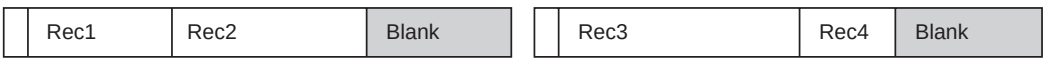


FIGURA A.3(b)

Registros desfragmentados

A.1.3 Formatos de registro

En la discusión se supuso que todos los registros lógicos tienen la misma longitud. Dado que trabajar con **registros de longitud fija** hace más fácil crear y manejar archivos, el ABD puede elegir crear la bases de datos con el uso de varios archivos, cada uno con un tipo de registro de longitud fija. Sin embargo, los registros pueden no ajustar de manera adecuada en el tamaño de bloque. Si hay espacio libre al final de los bloques sería posible colocar parte de un registro al final de un bloque y el resto del registro al principio del siguiente. Esta técnica se llama **registros fragmentados** y se ilustra en la figura A.3(a). Por simplicidad se ignoran los headers en esta ilustración. El problema con los registros fragmentados es que recuperar un registro a veces requiere dos accesos de disco. Por ejemplo, para recuperar el registro 3 tanto del bloque 1 como del bloque 2 se debe llevar al búfer. Por tanto, no se usa fragmentación. Una solución más simple es dejar espacio sin ocupar al final del bloque, como se muestra en la figura A.3(b). Desde luego, esta solución desperdicia espacio.

Algunos archivos de la base de datos tienen registros con diferentes longitudes, llamados **registros de longitud variable**. Distintas longitudes de registro resultan cuando diferentes tipos de registro se almacenan en el mismo archivo (porque están relacionados), o cuando registros del mismo tipo tienen campos de longitud variable o campos repetidos que ocurren un número distinto de veces en diferentes registros. Cuando la base de datos se carga por primera vez, los registros de longitud variable son fáciles de acomodar. Si se tienen formatos desbloqueados, simplemente se podría agregar un campo encabezado que muestre la longitud total del registro. Para registros bloqueados, cada registro lógico en el bloque está precedido por su longitud, como se muestra en la figura A.4(a), que dice al sistema dónde dejar de leer. Un problema con esta opción es que es imposible agregar datos adicionales a un registro de longitud variable sin reubicarlo. Por ejemplo, si necesita agregar 20 bytes más al primer registro en la figura A.4(a), tendría que reubicar el registro a un espacio más grande. Otro problema surge cuando se quiere borrar un registro. Una vez borrado un registro podría mover todos los registros posteriores a una ranura, pero esto puede involucrar rescribir una gran porción del archivo. En vez de ello, el sistema simplemente marca el registro como borrado y lo ignora cuando lee. Sin embargo, el espacio debe estar disponible cuando se hace una inserción, de modo que el sistema rastrea los espacios de los registros borrados. Si todos los registros tienen la misma longitud, el nuevo registro encajaría exactamente en el espacio. Sin embargo, los espacios dejados por los registros de longitud variable borrados son difíciles de reutilizar, pues los nuevos registros pueden no encajar exactamente en la ranura vacía. Por lo general, sólo un registro más corto se puede almacenar en el espacio dejado después de un borrado, lo que resulta en que muchos espacios quedan vacíos. Una posible solución al problema de reutilizar los espacios vacíos es identificar la longitud máxima para los registros y usar dicha longitud fija para todos los registros, lo que deja espacio vacío al final de los que son más cortos que el máximo. Esta técnica se muestra en la figura

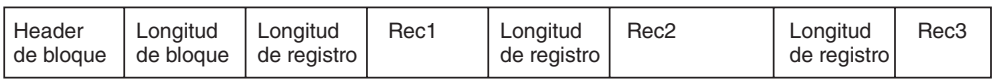


FIGURA A.4(a)

Registros con longitud variable con headers de longitud de registro

FIGURA A.4(b)

Uso de longitud máxima fija para todos los registros

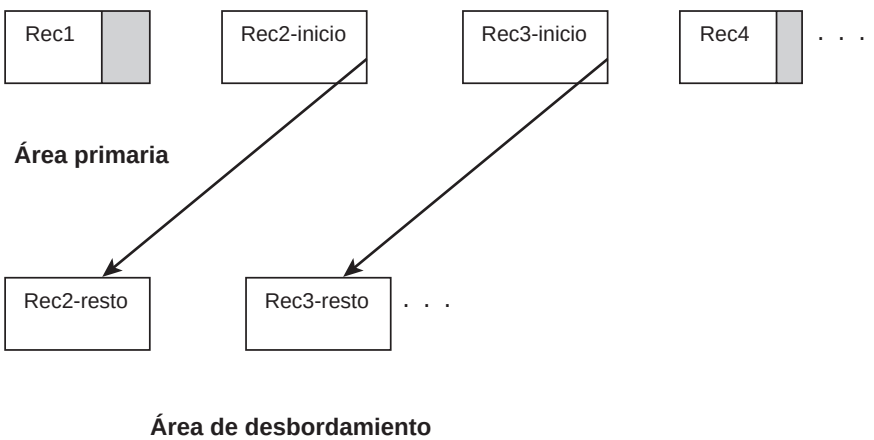


FIGURA A.4(c)

Uso de registros con longitud fija primaria y de desbordamiento

A.4(b). No obstante, si muchos registros son más cortos que el máximo, esto conduce a mucho espacio desperdiciado. Una mejor solución es establecer un **área primaria** (prime) con un espacio de registro de longitud fija de la longitud de registro más común y que un **área de desbordamiento** (overflow) para la partes de los registros que no encajan en el espacio usual. Para acceder a los datos desbordados se usa un **puntero**, un campo que contiene la dirección del desbordamiento, que se inserta en la porción del área primaria. Cuando se inserta un registro se comienza por almacenarlo en el área primaria. Si es muy largo para el espacio se pone el resto en el área de desbordamiento y se usa un puntero para conectar el principio del registro con su desbordamiento. La figura A.4(c) ilustra esta opción. Este método también permite que un registro crezca, pues se puede usar espacio de desbordamiento adicional según se requiera y conectar los punteros a los campos existentes del registro o del área primaria o del área de desbordamiento. También es fácil reutilizar espacio, porque cada registro dentro de la misma área tiene la misma longitud.

A.1.4 Organizaciones de archivos

La **organización de archivos** tiene que ver con la forma como se almacenan los datos de modo que se puedan recuperar cuando se necesiten. Incluye el orden físico y plantilla de registros en dispositivos de almacenamiento. Las técnicas utilizadas para encontrar y recu-

perar registros almacenados se llaman **métodos de acceso**. Puesto que los métodos de recuperación dependen en gran medida de la forma como se almacenan los registros, los términos "método de acceso" y "organización de archivos" se usan de manera intercambiable. El sistema operativo proporciona los métodos de acceso básicos y realiza la E/S real, pero el DBMS es responsable de solicitar este servicio del sistema operativo. Aunque existen muchos métodos de acceso disponibles en el sistema operativo, el procesamiento de la base de datos usa tres métodos básicos: secuencial, secuencial indexado y directo.

1. Organización de archivos secuencial
   En la **organización de archivos secuencial**, los registros se ordenan en secuencia física por el valor de un campo específico, llamado **campo de secuencia**. Con frecuencia el campo elegido es un campo **clave**, uno con valores únicos que se usan para identificar los registros. Los registros simplemente se ponen en el dispositivo de almacenamiento, por lo general cinta magnética, en orden creciente o decreciente por el valor del campo de secuencia. La figura A.5 ilustra un archivo Employee ordenado secuencialmente por la clave empId. Esta organización es simple, fácil de entender y fácil de manejar, pero es mejor para proporcionar **acceso secuencial**, y recuperar los registros uno después de otro en el mismo orden en el que se almacenan. No es bueno para **acceso directo** o **aleatorio**, que significa recoger un registro particular, porque por lo general requiere que pase sobre registros previos con la finalidad de encontrar el registro buscado. Tampoco es posible insertar un nuevo registro en medio del archivo. En la figura A.5 no se tendría espacio para insertar un nuevo empleado con empId de E103. Con organización secuencial, la inserción, borrado y actualización de registro se realizan al rescribir todo el archivo.

   El secuencial es el tipo de organización de archivos más antiguo y, a pesar de sus defectos, es adecuado para ciertas aplicaciones que usan **procesamiento por lotes** (batch) de un conjunto de registros. Por ejemplo, un programa de nómina usualmente requiere que se accede a cada registro de empleado en orden por empId. En una aplicación de nómina típica se tiene un archivo maestro de nómina con información permanente acerca del empleado y datos del año a la fecha acerca de ganancias y deducciones, y un archivo de transacción de nómina que contiene datos acerca de la semana pasada, como horas laboradas por cada empleado y cualquier cambio como borrado de registros antiguos o adiciones de algunos nuevos. El archivo de transacción se ordena por empId para coincidir con el orden del archivo maestro. Cuando el programa de nómina se corre, coincide con las ID de los registros maestro y transacción, calcula pago y deducciones, imprime comprobantes y recibos de pago, actualiza los totales del año a la fecha e inserta o borra registros según se indique mediante el archivo de transacción. En lugar de rescribir los registros en el archivo maestro, produce un nuevo archivo maestro cada semana. La figura A.6 resume este sistema, que es típico para procesamiento de archivo secuencial.

FIGURA A.5
Un archivo secuencial

E101	Jones,Jack	Rep. Ventas	Marketing	45000
E104	Smith,John	Asist. investigador	Investigación	35000
E110	Lyons,Mary	Investigador	Investigación	60000
E115	Chin,Greg	Planificador	Desarrollo	55000
. . .				

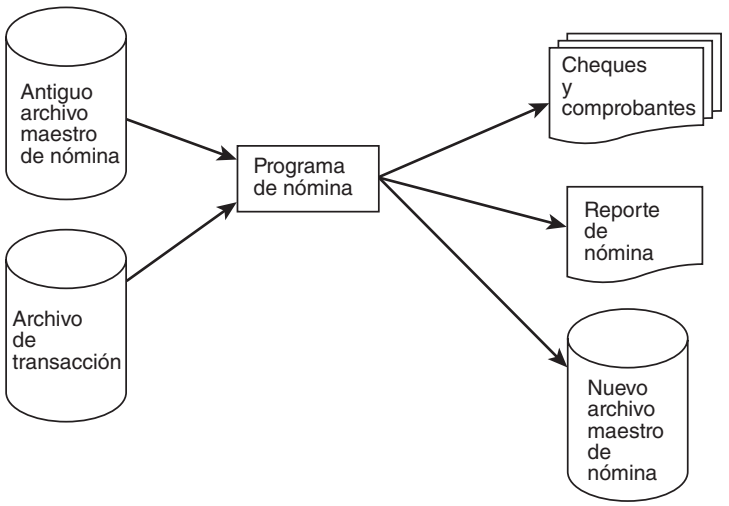


FIGURA A.6
Procesamiento por lotes con el uso de archivos secuenciales

Dado que mucho del procesamiento de la base de datos requiere acceso directo y actualización, inserción o borrado inmediatos de registros, la organización secuencial no es apropiada para procesamiento ordinario. Sin embargo, lo usan ampliamente los sistemas de gestión de bases de datos para producir dump o respaldos, copias de la base de datos que se conservan en cinta como datos de archivo, o en caso de que la base de datos se debe reconstruir después de una falla del disco.

2. Organización de archivos secuencial indexada
Es posible disfrutar las ventajas de los archivos secuenciales y aún así tener acceso directo al crear un índice, una tabla que diga dónde se almacenan registros particulares. Suponga que quiere conservar los registros de empleados en orden por empId porque se tienen muchas aplicaciones que acceden a ellos en dicho orden, pero se quiere tener posibilidad de ubicar un registro de empleado particular cuando sea necesario. Si los registros se almacenan en disco y pueden caber cinco por pista, se podría configurar el archivo como se muestra en la figura A.7(a). Puesto que el archivo está en orden por empId es fácil acceder a los registros de manera secuencial. Para proporcionar acceso directo podría crear un índice de pista **denso** que cite cada empId y proporcione la dirección (número de pista, en este caso) del registro con dicho valor empId. Sin embargo, muchas de las entradas de índice son innecesarias porque se puede obtener la misma información si se menciona sólo la clave más alta en cada pista, como se muestra en la figura A.7(b). Este índice **no denso** o **disperso** no tiene una entrada para cada registro, pero es suficiente para dar la ubicación de cada uno. Por ejemplo, para encontrar el registro del empleado E131, note que no puede estar en la pista 1 porque la ID de empleado es más alto que la mayor ID en dicha pista, E120. Dado que es más bajo que la mayor ID en la pista 2, E138, el registro, si existe, debe estar en alguna parte en la pista 2. Ahora la pista 2 se lee secuencialmente hasta que se encuentra el registro o se llega a un registro con un empId mayor sin encontrarlo.

Debido a que un archivo típico ocupa muchas pistas, el índice de pista será muy largo, y buscar en el índice puede consumir mucho tiempo. En consecuencia, se puede establecer otro nivel de índice, el índice cilindro. Si el archivo de empleados tiene 1 500 registros y cada cilindro tiene 30 pistas, y cada una contiene 5 registros, se necesitan 10 cilindros para el archivo. Se podría configurar un índice cilindro que dé la clave más alta en cada cilindro. Esto conduce al índice de pista correcto que menciona la clave más alta en cada pista del cilindro. La figura A.7(c) ilustra el uso de un

Pista 1	E101	E104	E110	E115	E120
Pista 2	E125	E130	E131	E134	E138
Pista 3	E140	E143	E145	E150	E153
	. . .				

FIGURA A.7(a)

Organización secuencial indexada

Clave más alta en la pista	Número de pista
E120	1
E138	2
E153	3
. . .	. . .

FIGURA A.7(b)

Índice de pista no denso

Clave más alta en cilindro	Número de cilindro
E820	1
E1236	2
E2121	3
. . .	. . .

Índice cilindro

Clave más alta en pista	Número de pista
E120	1
E138	2
E153	3
. . .	. . .
E820	30

Índice de pista para cilindro 1

Clave más alta en pista	Número de pista
E890	1
E908	2
E923	3
. . .	. . .
E1236	30

Índice de pista para cilindro 2

Clave más alta en pista	Número de pista
E1259	1
E1278	2
E1297	3
. . .	. . .
E2121	30

Índice de pista para cilindro 3

FIGURA A.7(c)

Índices cilindro y de pista

índice cilindro para ubicar un registro. Este proceso se podría realizar aún más al agrupar los cilindros en volúmenes, conjuntos de cilindros en el mismo o diferente empaque de disco, y configurar un índice de volumen o índice maestro que conduzca a un índice cilindro que conduzca a un índice de pista.

Las dificultades que se tienen con la inserción, borrado o actualización de registros en la organización de archivos secuencial se puede corregir con organización secuencial indexada porque puede limitarse la cantidad de reorganización que es necesario realizar. Los registros se pueden actualizar en lugar, al simplemente escribir sobre el registro antiguo dentro de la misma pista. Cuado se borra un registro se ubica la pista y se coloca una bandera de borrado al comienzo del registro para indicar que no se debe leer. Sin embargo, la inserción requiere más planeación. Se quiere tener posibilidad de insertar registros en orden secuencial, pero no se quiere rescribir todo el archivo siempre que se necesite insertar un registro. Por ejemplo, suponga que el archivo aparece exactamente como se muestra en la figura A.7(a) y se quiere insertar un nuevo empleado con empId E103. El registro pertenece a la primera pista, pero ahí no se tiene espacio para él. Si se inserta y mueven los otros registros, todas las pistas restantes tendrán que rescribirse y el índice crearse de nuevo. Se podrían anticipar inserciones al dejar algo de espacio o distribuir el espacio libre, en cada pista, para registros adicionales. Por ejemplo, si sólo se colocan cuatro registros en cada pista cuando el archivo se crea, se tendría espacio para un registro adicional en cada pista. Entonces se podría insertar E103 al colocarlo en su posición correcta y correr los otros registros a la pista 1. Sin embargo, ésta es una solución a corto plazo, pues ahora se tendrá un problema si se tiene que agregar otro nuevo registro de empleado, esta vez con ID E108. A fin de permitir un número desconocido de estas adiciones se crea un área de desbordamiento para registros que no encajen en sus pistas correctas. Para asegurar que es posible encontrar un registro desbordado se extiende el índice de pista al agregar un puntero de desbordamiento, que da la dirección del primer registro desbordado para cada pista. Los registros en el área de desbordamiento contienen campos puntero de modo que, si una pista tiene más de un registro de desbordamiento, el primero apuntará al segundo, el segundo al tercero, etc. La organización de archivos secuencial indexada la usan los sistemas operativos para gestionar algunos archivos no de base de datos, pero los archivos de base de datos no los gestionan los métodos de acceso secuencial indexado del sistema operativo. Sin embargo, las técnicas las usan los DBMS para establecer su propia organización de archivos indexada.

3. Organización de archivos directa
   Los sistemas de gestión de base de datos pueden usar organización de archivos directa para ayudar a gestionar la base de datos. Ésta es una de las organizaciones básicas utilizadas por el sistema operativo. Esta organización está diseñada para proporcionar acceso aleatorio, rápido acceso no secuencial directo a los registros. Con esta organización, los registros se pueden insertar en lo que parece ser un orden aleatorio, no en secuencia por valor de campo clave. A cada registro se le asigna una dirección relativa sobre la base del valor de un campo dentro del registro. Cuando debe almacenar un registro, el sistema toma el valor del campo especificado y por lo general realiza algún tipo de cálculo para derivar una dirección disponible para el registro. Normalmente, luego el registro se almacena en la dirección disponible. Cuando es momento de recuperar un registro, el sistema usa el valor clave proporcionado para adivinar dónde se debe almacenar el registro y va hacia dicha dirección para encontrarlo.

   Si los valores del campo elegido son simplemente enteros consecutivos, es asunto sencillo almacenar los registros. Por ejemplo, suponga que se crea una base de datos

con el fin de rastrear los pedidos de clientes para un proveedor específico. Si a todos los pedidos se les asignan números de pedido consecutivo, entonces el orden de los registros puede tener un campo llamado orderNumber, que se puede usar como la dirección relativa. Si el campo usa número no consecutivo o no es numérico es necesario convertir su valor en alguna forma. El esquema de conversión para valores numéricos se llama **esquema de dispersión** (hashing) y el campo sobre el que se realiza es el campo de claves hashing. Los valores no numéricos se convierten fácilmente en numéricos al usar algún tipo de código, por ejemplo, posición alfabética o valores ASCII. Una vez obtenido un número, existen muchos algoritmos posibles para derivar una dirección disponible. Por ejemplo, suponga que se almacenan registros de empleados y el campo de claves hashing es el número de seguridad social. Un ejemplo de número de seguridad social, expresado como campo numérico, es 123456789. Se busca un algoritmo que pueda tomar un número de nueve dígitos con posibles valores 0-999 999 999 y que lo convierta en una de las direcciones disponibles. No es adecuado usar el número de seguridad social como la dirección, porque ello requeriría un archivo con 1 000 000 000 de posiciones, la mayoría de las cuales estarían vacías, pues existen grandes brechas entre números de seguridad social entre un conjunto de empleados. Suponga, por ejemplo, que se tienen 1 000 posiciones para almacenamiento. Si sólo se tienen 800 empleados debería haber suficiente espacio en un archivo con 1 000 direcciones para todos sus registros. Sin embargo, debe recordar que se intenta mapear valores en el rango 0-999 999 999 en el rango 0-999, como se indica en la figura A.8. No importa si los números de seguridad social se mantienen en orden creciente, de modo que es aceptable mapear un número de seguridad social alto a una dirección baja, o uno bajo en una dirección alta, como se muestra en la figura A.8. Un método, llamado **método división/residuo**, es dividir por algún divisor fijo y tomar el residuo como la dirección. Para el ejemplo, si se usa 1 000 como el divisor, el residuo estará en el rango 0-999, y simplemente serán los últimos tres dígitos del número de seguridad social. Está garantizado que los valores están en el rango correcto para direcciones, 0-999. Por ejemplo, el número de seguridad social 085439598 da un residuo de 598 en la división entre 1 000, así que su dirección blanco es 598. La elección de 1 000 como el divisor fue arbitraria y se diseñó para facilitar los cálculos. De hecho, se acostumbra elegir divisores que sean números primos ligeramente menores que el número de direcciones disponibles. Existen muchos otros algoritmos de claves hashing.

Un gran problema, llamado **colisión**, ocurre cuando dos diferentes valores clave producen la misma dirección disponible. Entonces las claves se llaman **sinónimas**. Por ejemplo, se sabe que el número de seguridad social 085439598 produce la clave hashing 598 al usar división/residuo con un divisor de 1 000, pero también lo hace 998876598, de modo que estos dos números de seguridad social son sinónimos. Si anteriormente se insertó el registro del empleado con número de seguridad social 085439598 en la dirección disponible 598, o se podría almacenar el registro del empleado con número de seguridad social 998876598 en la dirección correcta. Un buen algoritmo de claves hashing es aquel que produce direcciones que coinciden con el rango, proporciona una distribución bastante uniforme de registros y minimiza las colisiones. Ningún algoritmo, sin importar cuán bien diseñado esté, elimina la colisión por completo. Por tanto, debe encontrar formas de manejar las colisiones. Una forma de minimizarlas, como ya se sugirió, es al elegir un buen esquema de claves. El método división/residuo con un número primo como el divisor es uno de los mejores. Sin embargo, puede haber algún patrón en los valores clave que causen muchas colisiones cuando se use este método. Por tanto, el ABD debe estar familiarizado con muchos esquemas y estudiar el efecto que cada uno tendrá sobre la distribución de los registros. Otra forma de minimizar las colisiones es calcular una

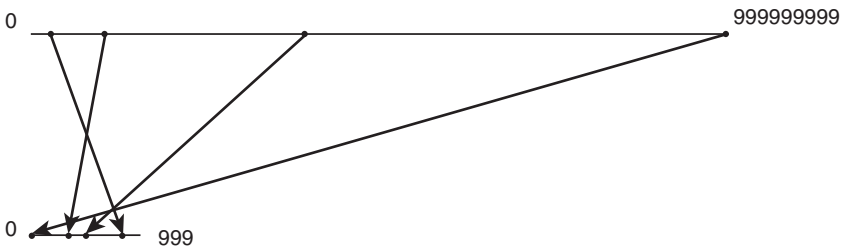


FIGURA A.8
Mapeo de número de seguridad social a direcciones de archivo

dirección de bloque (también llamado número repositorio) en lugar de una dirección de registro individual. Como se explicó antes, los registros en una base de datos por lo general se almacenan en bloques o páginas que pueden contener muchos registros lógicos. Para el archivo de empleados con 800 registros, se puede elegir un tamaño de página que contendrá cuatro registros de empleado. Aunque en teoría sólo se necesitarían 200 páginas para contener los 800 registros no se puede esperar que los registros estén perfectamente distribuidos, de modo que se permite un poco de espacio adicional. Se puede elegir reservar 250 páginas, o espacio para 1 000 registros. Esto daría una **densidad de empaquetado**, que es la razón porcentual de registros almacenados al número de espacios, de 80%. Ahora se cambia el algoritmo de claves de modo que produzca direcciones en el rango 0249 y se planea colocar cuatro registros en cada dirección o repositorio. Recuerde que un repositorio es un espacio que puede contener muchos registros. Por ejemplo, puede usar el método división/residuo con el número primo 241 como divisor y obtener direcciones en el rango 0-240. Ahora es posible almacenar cuatro sinónimos en el mismo repositorio. El efecto de colisiones se puede reducir aún más al aumentar el tamaño del repositorio. Por ejemplo, se puede dejar espacio para cinco registros en cada repositorio. Sin embargo, puede estar restringido por el sistema a tamaños de bloque fijos. Un segundo método de reducir el efecto de las colisiones es disminuir la densidad de empaquetado al aumentar el número de repositorios asignados. Sin embargo, conforme la densidad de empaquetado disminuye se desperdiciará más espacio, de modo que se negocia eficiencia de almacenamiento para mejor gestión de colisiones. La densidad de empaquetado de muestra de 80% es casi la más práctica. La densidad de empaquetado promedio está cerca de 60%.

Sin importar la eficiencia del algoritmo o el tamaño del repositorio, finalmente se llegará a un punto donde algún registro no encaja en el repositorio correcto. Existen dos formas de manejar este tipo de **desbordamiento**. El primero es buscar hacia delante hasta encontrar una rendija vacía o espacio de registro. Se puede encontrar espacio en el repositorio siguiente o en el que está después del asignado al registro. Si se busca hacia delante algún número predeterminado de direcciones (acaso cinco) sin encontrar espacio para el registro, simplemente no se puede insertar el registro y se tiene que reorganizar el archivo porque está muy lleno o denso en dicha región. Se elige un número predeterminado porque se quiere evitar búsquedas largas de registro, que demorarían el procesamiento. Un segundo método de lidiar con los desbordamientos es hacer a un lado un área de desbordamiento donde se inserten los registros que no encajan en sus repositorios correctos, y conectar el registro desbordado con su dirección correcta al colocar un puntero sinónimo en el registro almacenado en dicha dirección. Los registros en el área de desbordamiento también tienen punteros "sinónimo siguiente" que dan la dirección en el área de desbordamiento del sinónimo siguiente para la misma dirección disponible, de modo que todos los sinónimos para una dirección particular se pueden recuperar al seguir una cadena de punteros.

A.2 Estructuras de datos

El sistema de gestión de la base de datos usa varias técnicas con el fin de permitir la rápida recuperación directa de registros para aplicaciones, la recuperación sobre la base de relaciones entre registros y la recuperación mediante el valor de un campo no clave. Para hacerlo, el DBMS usa **estructuras de datos**, que son tipos de datos estructurados o compuestos hechos de tipos de datos más simples.

A.2.1 Archivos invertidos

Los archivos invertidos comúnmente se usan para permitir al sistema de gestión de base de datos recuperar registros sobre la base del valor de un campo no clave. Este campo, que puede o no tener valores únicos, se conoce como **clave secundaria**. Por ejemplo, se pueden tener registros de estudiantes ordenados físicamente en orden por ID de estudiante, como se muestra en la figura A.9(a). Este orden facilita al sistema operativo recuperar registros mediante ID, con el uso de acceso secuencial o secuencial indexado. Sin embargo, si quiere recuperar registros en orden alfabético por apellido es necesaria otra organización. Podría ordenar los registros por apellido cada vez, pero esto desperdicia tiempo de procesamiento. Se podrían almacenar registros en dos archivos diferentes, uno en orden por ID y otro en orden por apellido, pero esta duplicación de datos desperdicia espacio y a la larga hace que los datos se vuelvan inconsistentes. En vez de ello se puede crear un índice o lista invertida con el uso de `lastName` como el campo para el ordenamiento. A esto se le llama una lista invertida en el campo `lastName`. El índice secundario para la inversión se muestra en la figura A.9(b). Note que se usan direcciones relativas y se ignora el bloqueo de registros. Si se quiere acceder al archivo de estudiantes sobre la base de especialidad, se podría crear un índice secundario sobre `major`, con el uso de registros separados para cada valor repetido, como se muestra en la figura A.9(c). Si se consideran índices para cada uno de los campos en el registro `Student` se dice que el archivo está completamente invertido. Puesto que en la figura A.9 sólo dos de los campos están indexados, el ejemplo muestra un archivo parcialmente invertido. Los índices proporcionan acceso directo muy eficiente a los registros sobre la base de los valores del archivo indexado. Además, los índices mismos son suficientes para dar información como el número de especialidades CSC sin acceder al archivo en absoluto. Note que estos índices los creó el DBMS, no el método de acceso, aunque el DBMS necesita el método de acceso para decir las direcciones de registro para construir el índice. Cuando se recupera un registro con el uso del índice, el DBMS busca el valor apropiado del archivo indexado, determina la dirección relativa del registro deseado y pide al método de acceso recuperar el registro almacenado en dicha dirección. El método de acceso usa su propia técnica para encontrar y recuperar el registro. Los índices mismos, así como los archivos que indexa, se almacenan en las áreas bajo el control del DBMS.

A.2.2 Listas ligadas

Una segunda técnica para manejar claves secundarias o establecer algún otro orden deseado es la **lista ligada** o **cadena de punteros**. Una lista ligada se crea al agregar un campo de vínculo adicional a cada registro de datos. El campo contiene un **puntero**, la dirección del siguiente registro en la secuencia lógica a crear. La figura A.10(a) muestra registros de estudiantes en una lista ligada con vínculos ordenados en orden por `lastName`. Una vez más se usan direcciones relativas y se ignora el bloqueo. Note que debe identificar el registro cabecera, o primero, de la lista con el cual empezar, que es el registro 4 (Burns). Cuando se llega a dicho registro se observa el valor del vínculo para ver dónde aparece el siguiente registro lógico. Se sigue dicho vínculo al registro siguiente y se continúa así hasta que se

Student				
stuId	**lastName**	**firstName**	**major**	**credits**
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15

FIGURA A.9(a)
Archivo Student en orden por StuId

VALOR CLAVE	DIRECCIÓN RELATIVA
Burns	4
Chin	2
Jones	6
Lee	3
McCarthy	5
Rivera	7
Smith	1

FIGURA A.9(b)
Índice sobre lastName para el archivo Student

VALOR CLAVE	DIRECCIÓN RELATIVA
Art	4
CSC	7
History	1
History	3
Math	2
Math	5
Math	6

FIGURA A.9(c)
Índice sobre major para el archivo Student

alcanza el final de la cadena de punteros, identificado por un valor de vínculo nulo, que se escribe como 0. Si se desea se puede sustituir el puntero nulo al final de la cadena con la dirección de la cabeza de la lista, lo que por tanto crea una **lista ligada circular** o **anillo**. Un anillo permite alcanzar cualquier registro en la cadena desde cualquier otro. La figura A.10(b) ilustra una lista ligada circular de registros de empleados, esta vez con el uso de empId como el campo de ordenamiento. Una **lista doblemente ligada** es aquella en la que cada registro tiene dos punteros: uno **forward** o **siguiente** para indicar la ubicación del siguiente registro y uno **backward** o **previo** para indicar la ubicación del registro anterior. La figura A.10(c) muestra el archivo de empleados con el uso de una lista doblemente ligada para crear orden alfabético y alfabético inverso sobre `empName`. Es posible crear más de un orden lógico para el mismo archivo al usar dos o más campos punteros en cada registro. La figura A.10(d) muestra el archivo employee con el vínculo `empId`, que crea el orden por `empId`, y el vínculo `empName`, que crea el orden por `empName`.

La inserción es sencilla con una lista ligada. Simplemente se agrega el registro nuevo en el final físico del campo y se le incluye en el orden correcto al cambiar sólo dos vínculos. La figura A.11(a) muestra cómo insertar un nuevo registro de estudiante al final del archivo que se muestra en la figura A.10(a). El borrado es igualmente sencillo. Sólo se reajusta el puntero que solía dirigir hacia el registro borrado al establecerlo al valor puntero que aparecía en el registro borrado. La figura A.11(b) muestra cómo borrar un registro del archivo student original de la figura A.10(a). Para seguir la pista de cuáles rendijas están ocupadas

FIGURA A.10(a)

Lista ligada con punteros para lastName

Cabecera: 4

Student

stuId	lastName	firstName	major	credits	puntero
S1001	Smith	Tom	History	90	0
S1002	Chin	Ann	Math	36	6
S1005	Lee	Perry	History	3	5
S1010	Burns	Edward	Art	63	2
S1013	McCarthy	Owen	Math	0	7
S1015	Jones	Mary	Math	42	3
S1020	Rivera	Jane	CSC	3	1

FIGURA A.10(b)

Lista ligada circular con punteros para empId

Cabecera: 3

Employee

empId	lastName	firstName	dept	salary	puntero
E125	Jones	Mike	Marketing	38000	7
E110	Lyons	Mary	Investigación	50000	6
E101	Jones	Jack	Marketing	35000	4
E104	Smith	John	Investigación	30000	2
E120	Miranda	Jane	Ventas	48000	1
E115	Chin	Greg	Desarrollo	45000	5
E130	DiNoto	Steve	Investigación	55000	3

FIGURA A.10(c)

Lista doblemente ligada con punteros para lastName

Cabecera: 6

Cola: 4

Employee

empId	lastName	firstName	dept	salary	Puntero forward	Puntero backward
E125	Jones	Mike	Marketing	38000	2	3
E110	Lyons	Mary	Research	50000	5	1
E101	Jones	Jack	Marketing	35000	1	7
E104	Smith	John	Research	30000	0	5
E120	Miranda	Jane	Sales	48000	4	2
E115	Chin	Greg	Development	45000	7	0
E130	DiNoto	Steve	Research	55000	3	6

FIGURA A.10(d)

Lista ligada con punteros para empId y lastName

Cabecera empId: 3

Cabecera lastName: 6

Employee

empId	lastName	firstName	dept	salary	Puntero EmpId	Puntero lastName
E125	Jones	Mike	Marketing	38000	7	2
E110	Lyons	Mary	Investigación	50000	6	5
E101	Jones	Jack	Marketing	35000	4	1
E104	Smith	John	Investigación	30000	2	0
E120	Miranda	Jane	Ventas	48000	1	4
E115	Chin	Greg	Desarrollo	45000	5	7
E130	DiNoto	Steve	Investigación	55000	0	3

Student					
stuId	**lastName**	**firstName**	**major**	**credits**	**puntero**
S1001	Smith	Tom	History	90	0
S1002	Chin	Ann	Math	36	6
S1005	Lee	Perry	History	3	5
S1010	Burns	Edward	Art	63	2
S1013	McCarthy	Owen	Math	0	7
S1015	Jones	Mary	Math	42	**8**
S1020	Rivera	Jane	CSC	3	1
S1006	**Klein**	**Mark**	**CSC**	**0**	**3**

FIGURA A.11(a)

Añadido del registro S1006 al final del archivo Student

Cabecera: 4

Student					
stuId	**lastName**	**firstName**	**major**	**credits**	**puntero**
S1001	Smith	Tom	History	90	0
S1002	Chin	Ann	Math	36	**0**
S1005	Lee	Perry	History	3	5
S1010	Burns	Edward	Art	63	**6**
S1013	McCarthy	Owen	Math	0	7
S1015	Jones	Mary	Math	42	3
S1020	Rivera	Jane	CSC	3	1

FIGURA A.11(b)

Borrado del registro del estudiante S1002 del archivo Student original en la figura A.10(a)

Cabecera lastName: 4

Cabecera NO USADAS: 2

por los registros borrados, se hace una **recolección de basura** mediante otra lista ligada. Esta vez se tiene un header para las rendijas no utilizadas, lo que conduce al primer registro borrado, que a su vez apunta al segundo registro borrado (si existe alguno) y así por el estilo, como se muestra en la figura A.11(b). Cuando se necesita insertar un registro se puede reutilizar el espacio al colocar el registro en la dirección indicada por el header de espacio no usado y usar como el nuevo valor de header el valor puntero que solía aparecer ahí.

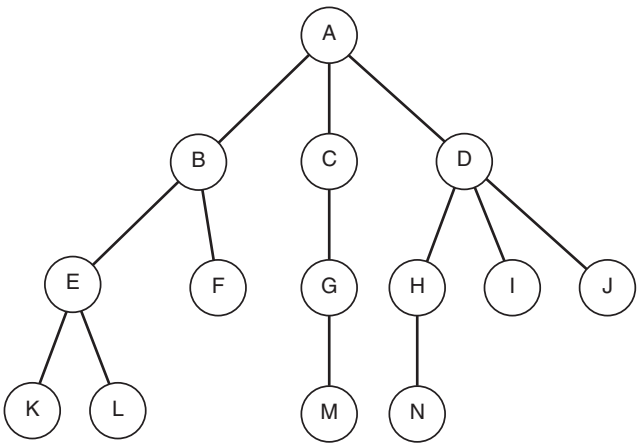
El DBMS es responsable de crear y mantener sus propias listas ligadas para varios órdenes lógicos. Para hacerlo, debe obtener direcciones del sistema operativo. Con el fin de permitir el seguimiento de los punteros, el sistema operativo debe usar organización secuencial indexada o directa como el método de acceso, la elección usual es el directo.

Es posible usar una combinación de listas invertida y ligada para claves secundarias con valores no únicos. El índice puede citar cada valor de clave secundaria sólo una vez, con un puntero hacia el primer registro que tenga dicho valor. El primer registro se convierte entonces en la cabecera de una lista ligada, que apunta al segundo registro con el mismo valor clave secundario, que a su vez apunta al tercer registro, etc. Una alternativa es usar un arreglo que contenga las direcciones de cada registro que tenga un valor particular para la clave.

A.2.3 Árboles

Muchos sistemas de gestión de base de datos usan una estructura de datos llamada **árbol** (tree) para contener índices. Un árbol es una estructura de datos que consiste en una jerar-

FIGURA A.12
Una estructura en árbol



quía de nodos. Los nodos contienen datos y están conectados mediante líneas o ramas. En el nivel más alto hay un solo nodo, llamado **raíz** del árbol. La raíz puede tener cualquier número de nodos dependientes, llamados sus **hijos**, directamente bajo ella. Estos nodos hijos, a su vez, pueden tener hijos dependientes de ellos. Una regla estricta para las estructuras en árbol es que cada nodo, con excepción de la raíz, tiene exactamente un padre, esto es, un nodo en el nivel inmediatamente arriba de ellos con el cual se relacionan. Las relaciones padre-hijo se muestran al dibujar una línea o un borde entre los nodos padre e hijo.

La figura A.12 representa un ejemplo de una estructura en árbol. En dicho ejemplo, el nodo A es la raíz. Los nodos B, C y D son sus hijos. B tiene dos hijos, E y F. C tiene un hijo, G, mientras que D tiene tres hijos, H, I y J. E tiene dos hijos, K y L. G tiene un hijo, M, y H tiene un hijo, N. A partir del diagrama, es claro que un nodo puede tener cero, uno o muchos hijos, pero un nodo sólo puede tener un padre. El nodo raíz no tiene padre. Un nodo que no tiene hijos se llama **hoja**, de modo que los nodos K, L, F, M, N, I y J son hojas. Note que las hojas pueden ocurrir en diferentes niveles. Los nodos que son hijos del mismo padre se llaman **hermanos**. Por ejemplo, a partir del diagrama, puede ver que los nodos E y F son hermanos, porque tienen el mismo padre, B. Para cualquier nodo, hay una sola ruta, llamada **ruta jerárquica**, desde la raíz hasta el nodo. Los nodos a lo largo de esta ruta se llaman **ancestros (ascendentes)** de dicho nodo. Por ejemplo, la ruta jerárquica hacia el nodo L comienza con A, pasa por B, luego E, y finalmente L. Por tanto, A, B y E son ancestros de L. De igual modo, para un nodo dado, cualquier nodo a lo largo de una ruta desde dicho nodo hasta una hoja se llama **descendiente**. Si visualiza un nodo como si fuese un nodo raíz en un nuevo árbol, el nodo y todos sus descendientes forman un **subárbol** de la estructura del árbol original. En el diagrama se ve que los descendientes de B son los nodos E, K, L y F. B forma la raíz del subárbol que lo contiene a él mismo y a todos sus descendientes.

A la raíz del árbol se le asigna el nivel 0. Sus hijos están en el nivel 1. Sus hijos están en el nivel 2, etc. La **altura** o **profundidad** de un árbol es el número máximo de niveles o, alternativamente, el número de nodos en la ruta jerárquica más larga desde la raíz hasta una hoja. El árbol en la figura A.12 tiene altura de 4. Se dice que un árbol está **equilibrado** si cada ruta desde el nodo raíz hasta una hoja tiene la misma longitud. El árbol en el ejemplo no está equilibrado porque la ruta de A a F tiene longitud 3, mientras que la ruta de A a K tiene longitud 4. El **grado** u **orden** de un árbol es el número máximo de hijos que tiene cualquier nodo. El árbol en la figura A.12 tiene orden 3. Un árbol **binario** es uno de orden 2, en el que cada nodo no tiene más de dos hijos. El ejemplo claramente no es un árbol binario, pues tanto A como D tienen tres hijos.

A.2.4 Árboles B+

Los árboles se usan para contener y procesar varias estructuras de base de datos, pero se usan ampliamente para indexar archivos. Una estructura llamada **árbol B+** se puede usar para almacenar un índice jerárquico eficiente y flexible que proporcione tanto acceso secuencial como directo a los registros. El índice consiste en dos partes, llamadas **conjunto índice** y **conjunto secuencia**. El conjunto secuencia está en el nivel inferior del índice (los nodos hoja) y consiste en todos los valores clave ordenados en secuencia con un puntero desde cada valor clave hasta su registro correspondiente en el archivo de datos. La figura A.13 ilustra un índice de árbol B+. Si observa el nivel inferior verá el conjunto secuencia que muestra todos los valores clave y sus punteros correspondientes que conducen hacia los registros de datos correspondientes. No se muestran los registros de datos, que se pueden ordenar de manera aleatoria o en cualquier secuencia física deseada. Se supone que los registros de datos no están bloqueados y que cada puntero conduce hacia un solo registro. Sin embargo, los punteros pueden conducir a **repositorios**, espacios que suelen contener muchos registros, si se desea. También notará que el puntero de la extrema derecha de cada nodo hoja, el **puntero horizontal**, se usa para vincular el nodo con el siguiente en el conjunto secuencia. Esto permite usar el conjunto secuencia para acceso secuencial al archivo. Todo lo que necesita es comenzar en el nodo hoja de la extrema izquierda y ubicar cada registro desde dicha hoja a su vez, luego seguir los punteros horizontales para llegar al siguiente nodo del conjunto secuencia, y así por el estilo.

El acceso directo a los registros se logra con el uso del conjunto índice, desde el nodo raíz y siguiendo una ruta jerárquica estricta hacia el nodo apropiado en el conjunto secuencia. El nodo raíz en la figura A.13 tiene la misma estructura que todos los otros nodos del árbol. En este ejemplo, tiene espacio para 3 valores clave y 4 punteros que podrían conducir a 4 nodos hijos, de modo que el árbol tiene orden 4 (en realidad, un nodo contendría muchos más valores clave y punteros). Dado que sólo se usan 2 valores clave y 3 punteros en el nodo raíz, los valores de la clave y el puntero de la extrema derecha se dejan en blanco, y sólo se muestran 3 nodos de nivel 1. El puntero de la extrema izquierda se usa para acceder a todos los registros cuyos valores clave son menores que 100. El puntero medio conduce a todos los registros con valores clave mayores que o iguales a 100, pero menores que 200, mientras que el puntero no nulo de la extrema derecha conduce a todos los registros con valores clave mayores que o iguales a 200. Al seguir el puntero de la extrema izquierda del nodo raíz se llega a un nodo de nivel 1 con espacio para 3 valores clave y 4 punteros. En este

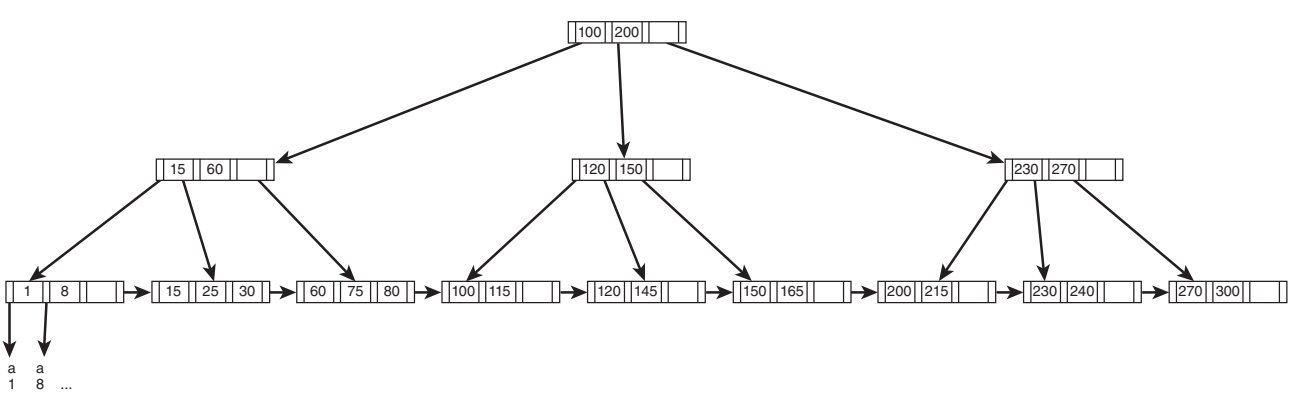


FIGURA A.13

Un árbol B+

nodo sólo se almacenan 2 valores clave, 15 y 60, y 3 punteros. El puntero de la extrema izquierda en este nodo se usa para acceder a todos los registros con valores clave menores que 15, el siguiente puntero para registros con claves mayores o iguales que 15 pero menores que 60, el siguiente para claves mayores que o iguales a 60 pero menores que 100, y el último está vacío en el momento. Si una vez más se sigue el puntero de la extrema izquierda, se llega al nodo de la extrema izquierda del conjunto secuencia. Éste tiene valores clave 1 y 8. El puntero de la extrema izquierda conduce al registro de datos con valor clave de 1, el siguiente al registro de datos con valor clave de 8, el tercero está en blanco, y el cuarto conduce al siguiente nodo del conjunto secuencia. Recordará que este puntero se usó para encadenar nodos de conjunto secuencia para procesamiento secuencial y no se usó para acceso directo.

Suponga que se quiere acceder al registro con valor clave 115. Comenzaría en el nodo raíz y seguiría el segundo puntero, porque el valor que se busca es mayor que 100 pero menor que 200. Esto conduce al nodo medio de nivel 1. Esta vez, seguiría el puntero de la extrema izquierda, pues el valor que busca es menor que 120. Esto conduce al conjunto secuencia, donde se encuentra el valor 115 en la segunda posición del nodo apropiado. El puntero a la izquierda de 115 conduce al registro de datos correcto.

Existen reglas estrictas para construir árboles B+, tales como:

- Si la raíz no es una hoja debe tener al menos dos hijos.
- Si el árbol tiene orden n, cada nodo interior (esto es, todos los nodos excepto la raíz y los nodos hoja), debe tener entre n/2 y n punteros (e hijos) ocupados. Si n/2 no es entero, redondee para determinar el número mínimo de punteros.
- El número de valores clave contenido en un nodo no hoja es 1 menor que el número de punteros.
- Si el árbol tiene orden n, el número de valores clave ocupados en un nodo hoja debe estar entre (n – 1)/2 y n – 1. Si (n – 1)/2 no es un entero, redondee para determinar el número mínimo de valores clave ocupados.
- El árbol debe estar equilibrado, esto es: cada ruta desde el nodo hasta una hoja debe tener la misma longitud.

Estos requisitos explican la eficiencia de un índice de árbol B+. En la práctica, cada nodo en el árbol es en realidad un bloque, de modo que se pueden almacenar mucho más que 3 valores clave y 4 punteros en un nodo típico. Si tiene espacio para 20 punteros en cada nodo, fácilmente podría ingresar a 1 000 registros con el uso de un índice de nivel 3. Dado que el nodo raíz usualmente se mantiene en la memoria principal durante el procesamiento, sólo necesitaría 2 accesos de disco para llegar al conjunto secuencia, o sólo un total de 3 accesos para llegar a cualquiera de los 1 000 registros de datos.

La inserción y el borrado de registros en un archivo de datos con un índice árbol B+ puede ser complicada, en particular si los nodos del índice se llenan o vacían demasiado. Primero considere el caso simple donde no surgen problemas cuando se insertan registros en el índice que se muestra en la figura A.13. Suponga que se quiere insertar un registro con valor clave de 5. Se ve que su clave pertenece al nodo hoja de la extrema izquierda, porque es menor que 15. Para colocarlo en el orden correcto se mueve el valor 8 y su puntero hacia la derecha y se inserta 5 y su puntero en la posición secuencial correcta, de modo que el índice árbol ahora aparece como en la figura A.14(a). El registro con el valor clave 5 se inserta en el archivo de datos, y su dirección se convierte en el valor puntero almacenado a la izquierda del 5 en el conjunto secuencia. El nodo hoja de la extrema izquierda ahora está completamente lleno, y no se pueden hacer otras inserciones sin cierta reorganización. Ahora suponga que se quiere borrar el registro con valor clave 80. El nodo de conjunto secuencia

que contiene 80 es el tercero desde la izquierda. Simplemente se borra el valor clave y su puntero, el nodo hoja ahora tendría sólo dos valores clave. Debe comprobar los requisitos para ver si dos claves son suficientes para un nodo hoja. Recuerde que este árbol tiene orden 4. La regla para hojas es que una hoja debe tener entre (n – 1)/2 y n – 1 claves. Al calcular (4 – 2)/2 se ve que el nodo hoja tiene suficientes valores clave. Por tanto, se borran el valor clave y su puntero, y se borra el registro de datos. La figura A.14(b) muestra el estado actual del índice, después de insertar 5 y borrar 80.

Ahora considere lo que ocurre cuando un nodo hoja queda muy lleno. Suponga que quiere insertar un registro con valor clave 20. El valor clave pertenece al segundo nodo hoja desde la izquierda, entre 15 y 25. Sin embargo, este nodo hoja ya está lleno, de modo que se debe reorganizar. El nodo existente debe **dividirse** en dos nodos. De inmediato se agrega un nuevo nodo hoja a la derecha del existente y los valores clave se dividen de modo que haya aproximadamente la mitad en cada una de las dos hojas resultantes. Ahora el nodo antiguo contendrá valores clave 15 y 20, con sus punteros, mientras que el nuevo contendrá los valores 25 y 30, con sus punteros. Sin embargo, debe considerar qué efecto puede tener esto sobre el padre del nodo hoja. Se ve que 25 debe aparecer en el nodo padre, que es el nodo

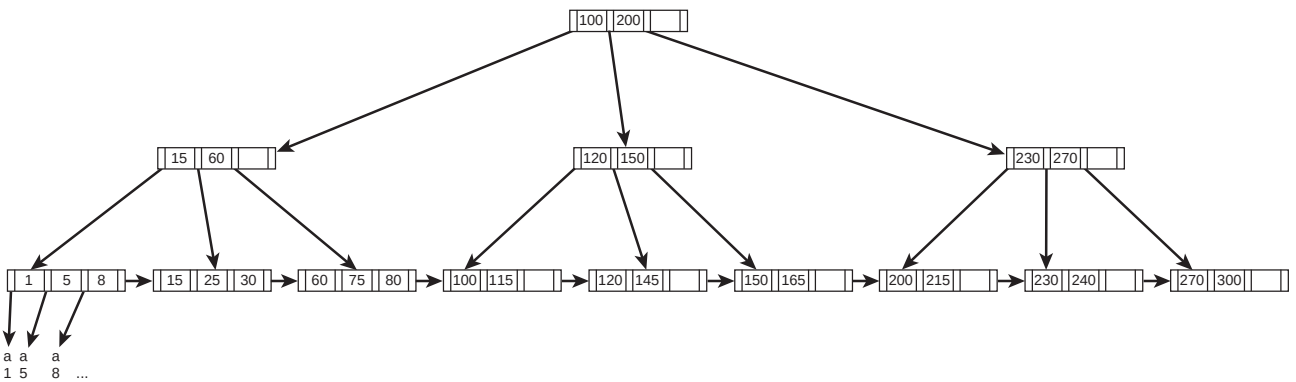


FIGURA A.14(a)

Inserción del valor clave 5

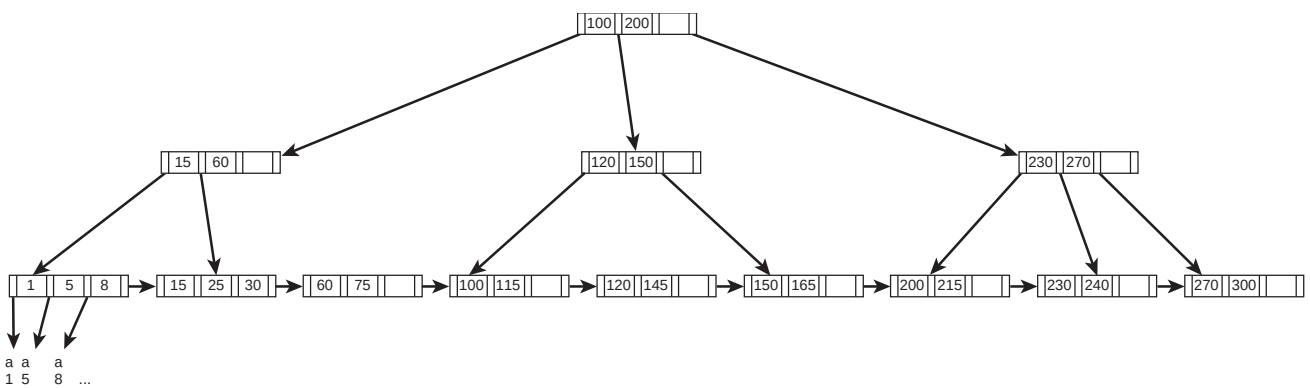


FIGURA A.14(b)

Índice después de insertar 5 y borrar 80

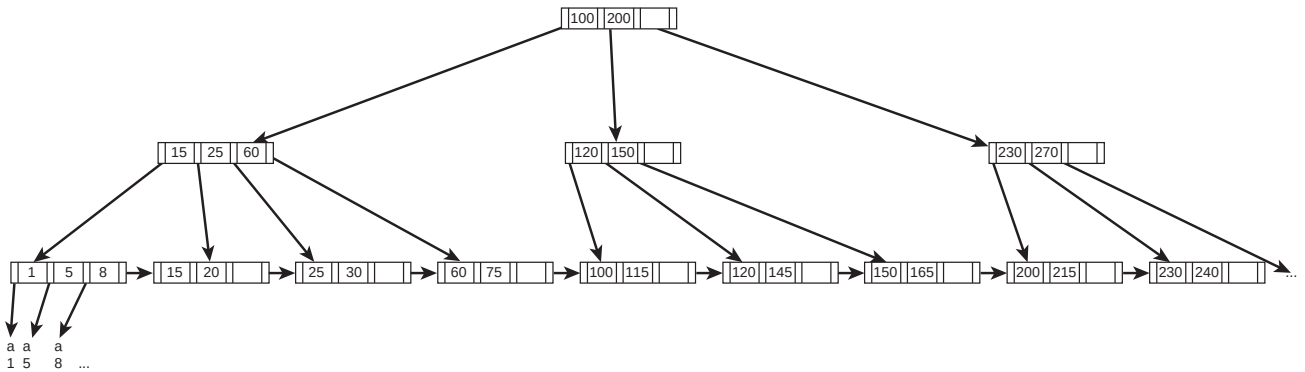


FIGURA A.14(c)

Índice después de insertar 5, borrar 80 e insertar 20

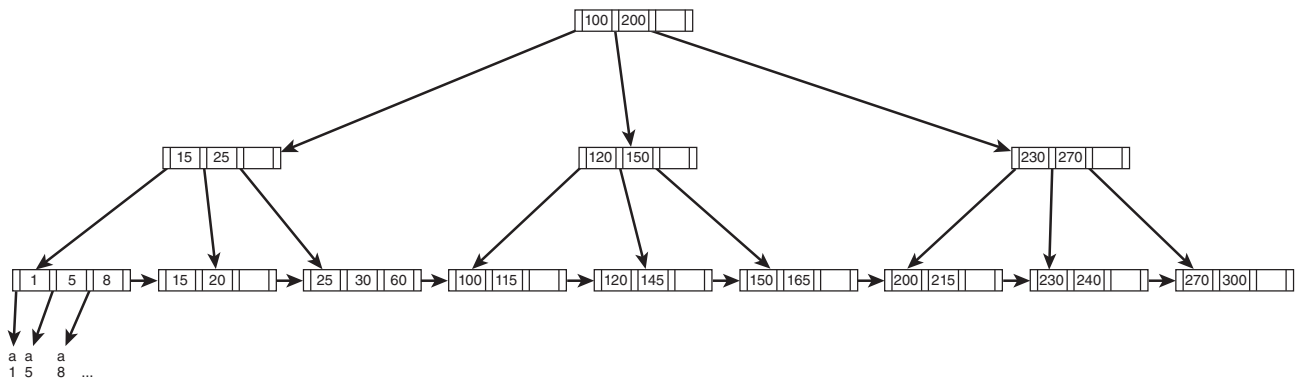


FIGURA A.14(d)

Índice después de insertar 5, borrar 80, e insertar 20 y borrar 75

de nivel 1 a la extrema izquierda. En consecuencia, dicho nodo se rescribe de modo que los valores clave aparecerán en secuencia adecuada, que es 15, 25, 60. Los punteros también se ajustan para conducir a los nodos hoja adecuados, incluido el nuevo. Ahora se puede insertar el registro de datos. La figura A.14(c) muestra el nuevo estado del índice. Se tiene fortuna de que el padre tenga suficiente espacio para el valor clave y el puntero a la nueva hoja. Si no fuese así, también tendría que dividir al padre, y ajustar su padre, el nodo raíz. Si el nodo raíz estuviera lleno, tendría que dividir la raíz, lo que requeriría la creación de un nivel arriba de la raíz actual, lo que resulta en la agregación de otro nivel al índice. Este ejemplo muestra por qué los índices de árbol B+ por lo general se crean con algunos espacios vacíos para permitir la inserción limitada sin dividir.

Ahora considere un ejemplo donde el borrado causa un problema. Comenzando con el índice que aparece en la figura A.14(c) borre el registro con valor clave de 75. El nodo hoja afectado es el cuarto desde la izquierda. Si tuviera que borrar el 75, esta hoja tendría sólo una clave restante, 60. Para un árbol de orden 4, las hojas deben tener un mínimo de 2 claves, de modo que no está permitido tener tal hoja. Note que, si la hoja no estuviera vacía, simplemente podría borrarla y ajustar el nodo padre. Sin embargo, contiene información que se necesita, a saber el valor clave de 60 y el puntero al registro correspondiente. Para

preservar esta información se busca un nodo hermano en el que se le pueda almacenar. El nodo inmediatamente a la izquierda tiene el mismo padre y contiene sólo 2 claves, 25 y 30. En consecuencia, se **fusionan**, o combinan, los dos nodos hoja hermanos en uno con los tres valores clave, 25, 30 y 60. También debe ajustar el nodo padre al borrar el valor de 60 y el puntero desde él hacia el nodo viejo. El resultado se muestra en la figura A.14(d). Note que, si el nodo padre se vuelve muy pequeño (menor que 2 punteros, para este índice), tendría que fusionar nodos de nivel 1. Si fusionar nodos de nivel 1 causa alguna vez que la raíz tenga menos de dos hijos, el índice pierde un nivel. En un caso donde un hermano está muy lleno para permitir fusión, sería necesario redistribuir punteros entre los dos hermanos (en gran medida como se hizo con la división) de modo que cada nodo tenga el número requerido.

A.2.5 Árboles B

Un índice de árbol B es similar a un índice de árbol B+, pero elimina almacenamiento redundante de algunas claves. En la figura A.14 los valores que aparecen en niveles superiores del índice se repiten en el conjunto secuencia. Por ejemplo, 100, 200, 15, 25 y otros aparecen en dos niveles. El índice se puede hacer ligeramente más eficiente para acceder a algunos registros al colocar punteros de registro de datos para dichos valores en el nivel más alto donde aparecen, en lugar de llevar los valores a todo lo largo del conjunto secuencia. Cada nodo no hoja se expandirá para incluir punteros de registros de datos así como los punteros usuales al siguiente nivel del índice. La figura A.15 muestra el nodo raíz del índice que se muestra en la figura A.14(d) con los punteros adicionales que se incluiría. Punteros similares aparecerían en todos los nodos no hoja. Los índices de árbol B son más eficientes para buscar si el valor que se desea aparece más alto en el índice que el conjunto secuencia, porque el número de accesos será menor que en un índice de árbol B+ correspondiente. Sin embargo, el mantenimiento del índice es más complicado. Además, un nodo de árbol B+ retiene más valores clave, como no contiene los punteros directos de registro de datos de un árbol B, de modo que se necesitan menos nodos para contener el índice. Finalmente, los nodos hoja de un árbol B no forman un conjunto secuencia completo que se pueda usar para acceso secuencial, como en el árbol B+.

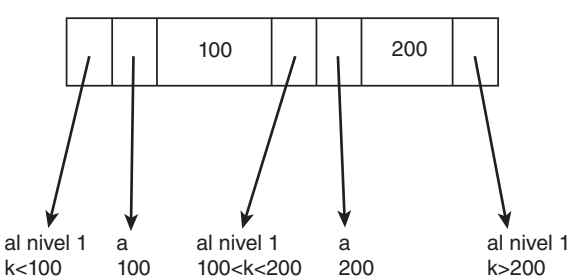


FIGURA A.15
Árbol B

APÉNDICE B: El modelo de red

El modelo de red, uno de los dos modelos de bases de datos más antiguos, se usó a principios de la década de 1960 en uno de los primeros sistemas de gestión de base de datos, **Integrated Data Store**, creado por Charles Bachman en General Electric. Otro de los primeros DBMS que usó el modelo de red, Integrated Database Management System (**IDMS**, sistema de gestión de base de datos integrado), fue desarrollado por Cullinet, que más tarde se fusionó con Computer Associates. La versión actual de este primer sistema, **CA-IDMS**, es un híbrido de las tecnologías de red y relacional, e incluye soporte para SQL. Aunque el modelo relacional desplazó al modelo de red para nuevos desarrollos de base de datos en el mercado, existen muchas bases de datos "legadas" basadas en el modelo de red, que todavía funcionan. Otras bases de datos legadas basadas en red contienen datos que se deben convertir a un formato relacional u orientado a objeto para nuevo desarrollo de bases de datos.

B.1 Modelo y terminología DBTG

Como se describió en la sección 1.7, CODASYL citó a una fuerza de tarea de bases de datos (**DBTG**) para crear estándares para bases de datos en la década de 1960, y dicho grupo o sus sucesores continuaron creando y publicando estándares durante muchos años. Introdujeron una terminología y estructura estándar, incluido el concepto de la arquitectura de base de datos en tres niveles, con un lenguaje de definición de datos para cada nivel. El nivel externo se especifica en el **subesquema**, que se escribe en el **subesquema de lenguaje de descripción de datos**. El nivel conceptual se especifica en el **esquema**, con el uso de **esquema de lenguaje de descripción de datos**. El **lenguaje de definición de almacenamiento de datos** (DSDL) se agregó en 1978 para describir el modelo interno y los detalles físicos de almacenamiento. Las propuestas de la DBTG también contenían especificaciones para un lenguaje de manipulación de datos para lenguajes anfitrión, mas no para consultas en línea, porque se supuso que los usuarios serían programadores. La figura B.1 muestra la arquitectura de un sistema de base de datos DBTG. La User Work Area (UWA, área de trabajo del usuario) que aparece en el nivel superior es un búfer a través del cual el programador o usuario se comunica con la base de datos, usando uno de los lenguajes anfitrión (host) tradicionales.

El modelo DBTG usa la **red** como su estructura básica de datos. Una red es un gráfico que consiste en nodos conectados mediante arcos dirigidos. Los nodos corresponden a tipos de registros y los arcos o vínculos a punteros. Con el término E-R, el modelo de red DBTG usa registros para representar entidades y punteros entre registros para representar relaciones. Sólo las relaciones binarias 1 a 1 o 1 a muchos se pueden representar

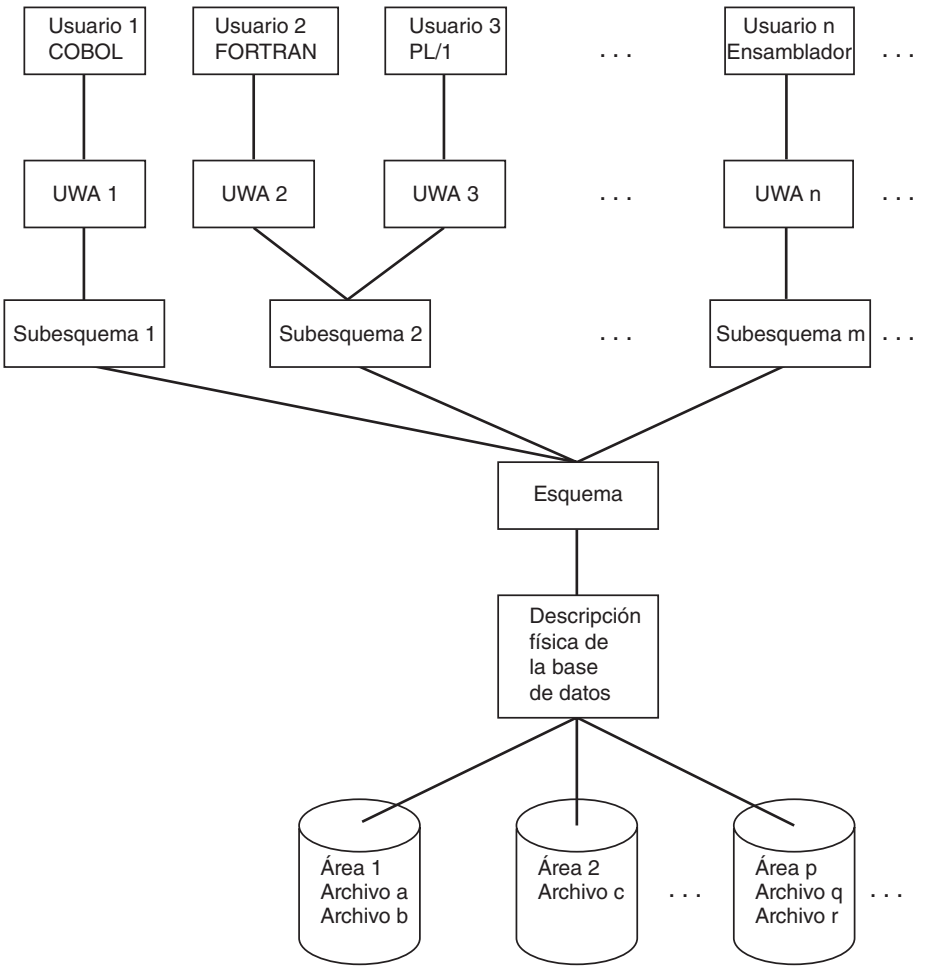


FIGURA B.1
Arquitectura de un sistema de modelo de red DBTG

directamente mediante vínculos. En una estructura de datos en red, a diferencia de una estructura en árbol, un nodo dependiente (llamado hijo o miembro), puede tener más de un padre o nodo propietario. La figura B.2 muestra un ejemplo de estructura en red.

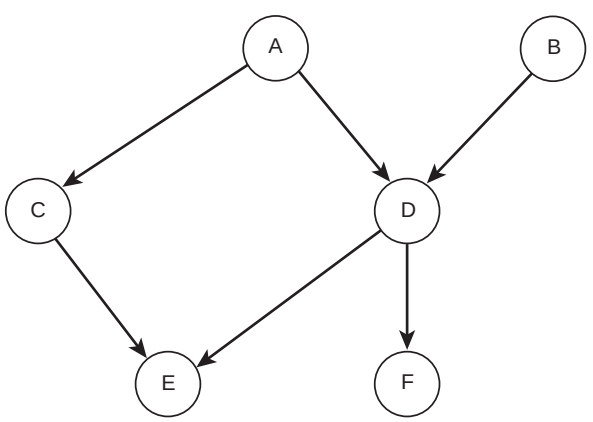


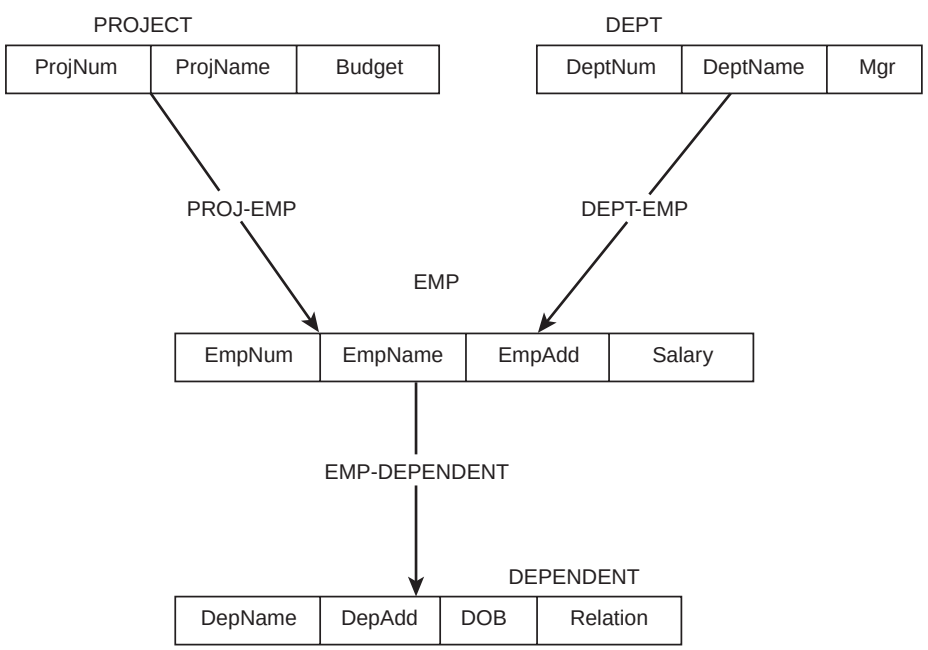
FIGURA B.2
Una estructura en red

B.2 Registros y conjuntos

Una base de datos en red consiste en cualquier número de **tipos de registros** nominados cuyas estructuras se describen por completo en el esquema. Un registro tiene cualquier número de **ítems de datos** o **campos**, que son las unidades de datos nominadas más pequeñas. Cada ítem de datos tiene un tipo de datos específico asociado con él en el esquema. Los ítems de datos se pueden agrupar en **agregados de datos**, que pueden estar anidados. Los diagramas de estructura de datos en red, como se muestran en la figura B.3, se pueden usar para representar estructuras de bases de datos. Este diagrama corresponde a un gráfico de red en el que los nodos se sustituyeron por rectángulos que representan registros y vínculos sustituidos por flechas que conectan los rectángulos. Los rectángulos se pueden subdividir para mostrar ítems de datos. Los vínculos por lo general son relaciones unidireccionales 1 a muchos, pero también se permiten relaciones 1 a 1 y bidireccionales. La base de la flecha viene desde el nodo propietario y la punta de la flecha apunta hacia el nodo miembro o dependiente en cada relación.

Una característica distintiva del modelo DBTG es el concepto de nombres de **conjuntos** para expresar relaciones. Un **tipo conjunto** consiste en un solo tipo **nodo propietario** y uno o más tipos nodo dependiente con el que se relaciona, llamados **tipos miembro**. Generalmente sólo existe un tipo miembro para un tipo conjunto dado. Por ejemplo, en la figura B.3 se muestran tres tipos conjuntos. Uno se llama PROJEMP, y tiene PROJECT como el tipo propietario y EMP como el tipo miembro. Un segundo, DEPT-EMP, tiene DEPT como el tipo propietario y EMP como el tipo miembro. El tercero, EMP-DEPENDENT, tiene EMP como el tipo propietario y DEPENDENT como el tipo miembro. Una **ocurrencia de un conjunto** es una colección de registros que tienen una ocurrencia propietaria y cualquier número de ocurrencias miembro asociadas. Por ejemplo, en la figura B.4 se muestra una ocurrencia del conjunto DEPT-EMP. Existen tantas ocurrencias de este tipo conjunto como existen registros DEPT. Si existe algún departamento sin empleados, dicha ocurrencia conjunto tiene un propietario y ningún miembro. Se dice que esta ocurrencia de conjunto está **vacía**. Una regla estricta para conjuntos es que un registro no puede ser miembro de dos ocurrencias del mismo tipo conjunto. Para el conjunto DEPT-EMP en este ejemplo,

FIGURA B.3
Un diagrama de estructura de datos en red



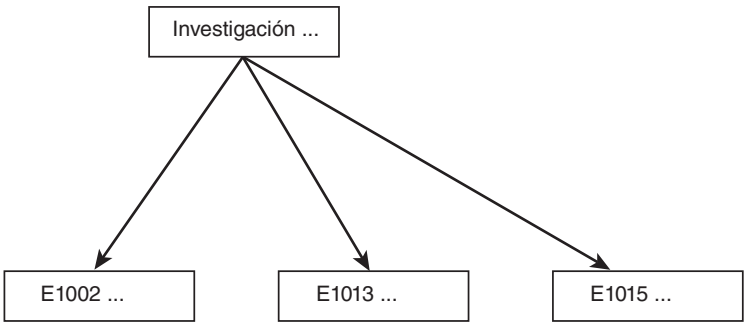


FIGURA B.4
Una ocurrencia del conjunto DEPT-EMP

esto significa que un empleado no puede pertenecer a más de un departamento. Sin embargo, un registro puede pertenecer a más de un tipo conjunto. Por ejemplo, un empleado también puede asociarse con un proyecto, de modo que un registro de empleado podría también ser propiedad de un registro proyecto en una ocurrencia del conjunto PROJ-EMP. Un tipo registro puede ser un tipo propietario en un conjunto y un tipo miembro en otro. Por ejemplo, de acuerdo con el diagrama de estructura de datos de la figura B.3, un registro EMP puede ser miembro de dos tipos conjunto, PROJ-EMP y DEPT-EMP, y propietario de un tipo conjunto, EMP-DEPENDENT. Por tanto, un registro EMP particular podría conectarse con un registro PROJECT, un registro DEPT y varios registros DEPENDENT.

El conjunto DEPT-EMP posibilita decir a cuál departamento pertenece un empleado al saber en cuál ocurrencia del conjunto DEPT-EMP está el registro EMP. Este conjunto transmite información, y mantiene una conexión lógica debido a su existencia. Tal conjunto se llama conjunto **portador de información** o **esencial**. La alternativa a un conjunto esencial es conservar el valor de uno o más campos de la ocurrencia del registro propietario en la ocurrencia miembro. Por lo general, ésta es una clave externa, como en el caso de almacenar DEPT en el registro empleado. Un conjunto con tal información en los registros miembro se llama conjunto **basado en valor**. Los conjuntos esenciales reducen la redundancia, conservan espacio y aseguran integridad de datos. Sin embargo, los conjuntos basados en valor garantizan que la información de membresía de conjunto se conserva incluso si los punteros de conjunto se pierden o destruyen accidentalmente.

Puesto que los conjuntos son el único método de mostrar relaciones entre registros, y dado que un conjunto sólo puede tener un propietario, no hay forma directa de representar una relación muchos a muchos en este modelo. No obstante, hay una forma indirecta de manejar el caso muchos a muchos. Recuerde que, en el modelo E-R, las relaciones muchos a muchos se representan como diamantes que conectan los conjuntos entidad asociados. Tales relaciones se representan en el modelo relacional al formar una tabla cuyas columnas consisten de las claves primarias de las entidades asociadas, junto con cualquier atributo descriptivo. El modelo de red usa una representación similar para las relaciones muchos a muchos. Cuando dos tipos registro tienen una relación muchos a muchos se crea un nuevo tipo registro, llamado **registro intersección** o un registro vínculo, que consiste sólo de las claves de los registros asociados, más cualquier información de intersección, esto es, cualquier atributo descriptivo cuyos valores dependen de la asociación. Considere el modelo que se muestra en la figura B.3. Dado que muestra un conjunto PROJ-EMP se supone que cada empleado se puede asignar sólo a un proyecto. Suponga, en vez de ello, que un empleado puede trabajar en varios proyectos de manera simultánea. La resultante relación muchos a muchos no se puede representar directamente. En su lugar, se tiene que crear un tipo registro intersección, para mantener la conexión entre un empleado y todos sus proyectos. Este registro intersección contiene `projNum`, `EmpNum` y cualquier dato de intersección, como el número de horas que se asignan a un empleado para trabajar en el proyecto, y la

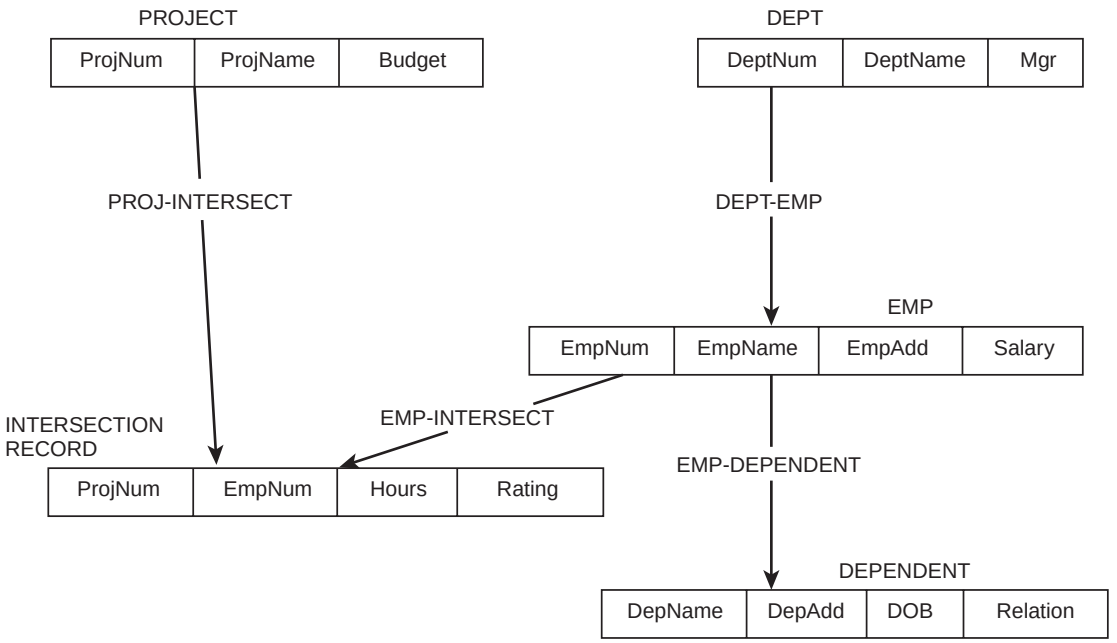


FIGURA B.5

Ejemplo de empleados con múltiples proyectos por empleado

calificación (rating) que recibe el empleado para trabajar en dicho proyecto. La figura B.5 muestra el modelo revisado para manejar el caso muchos a muchos. La figura B.6 muestra los registros intersección creados para la interacción del empleado E1015 con los proyectos J10 y J15, así como algunos otros empleados y proyectos.

En el modelo DBTG, todos los conjuntos se implementan mediante punteros. Se crea una **lista ligada** en la que la ocurrencia propietaria es la cabeza. La propietaria apunta al primer miembro, que apunta al segundo, etc., hasta que se llega al último miembro. Entonces el último miembro apunta de vuelta a la propietaria, lo que forma una cadena. La figura B.7(a) muestra cómo se representa una ocurrencia del conjunto DEPT-EMP, y la figura B.7(b) muestra cómo aparecen todas las ocurrencias DEPT-EMP. Note que DEPT y EMP simplemente son dos archivos separados, con punteros que los conectan. Aunque los punteros se muestran como flechas, en realidad son direcciones lógicas o claves de los registros disponibles. Considere la pregunta: ¿cuáles empleados trabajan en el departamento de investigación? Para responder, comience en el registro investigación en el archivo DEPT y siga su puntero EMP al registro EMP de E1002. Este registro apunta a E1013, que apunta a E1015, que apunta de vuelta al registro buscado en el archivo DEPT. Una vez de vuelta a la cabeza o propietario, se sabe que se llegó a todos los registros miembro. Ahora considere la consulta: ¿a cuál departamento pertenece E1004? Esta vez comience en el archivo EMP en el registro de E1004, siga su puntero al siguiente registro EMP para el mismo departamento, E1010, y siga su puntero, que va al registro marketing en el archivo DEPT. Por ende, al final se regresa al registro propietario correcto. Sin embargo, la lista ligada podría ser muy larga para departamentos con muchos empleados, de modo que se tiene la opción de crear punteros propietarios directamente en registros miembro. Esta opción duplica el número de punteros requeridos para un conjunto dado, pero facilita responder la segunda consulta. Otra opción es hacer la lista doblemente ligada, al colocar un puntero previo en cada registro. En consecuencia, un miembro conjunto puede tener tres punteros: uno al siguiente registro, uno al registro previo y uno al propietario.

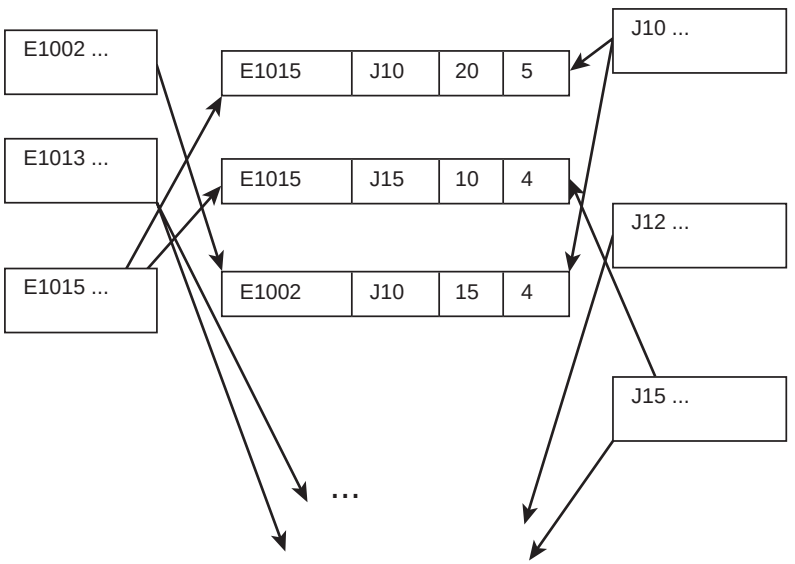


FIGURA B.6

Registros intersección para la relación Project-Employee muchos a muchos

Incluso si sólo se usan los punteros siguientes, si un registro pertenece a varios conjuntos contendrá muchos punteros para mostrar su ubicación en cada uno de los conjuntos. La pertenencia a dos conjuntos requiere un mínimo de dos punteros, pero podrían ser cuatro o seis punteros si se eligen opciones de punteros propietario y previo. El puntero principal

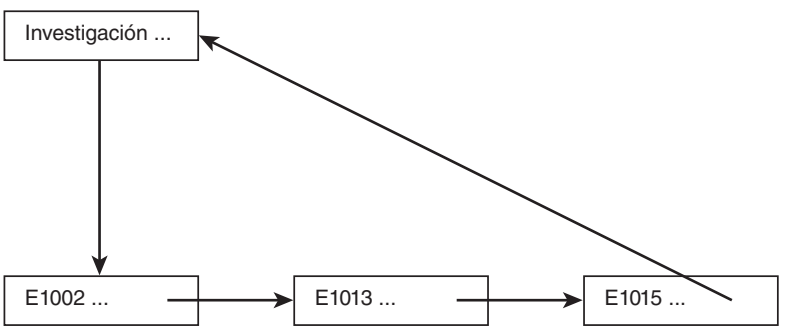


FIGURA B.7

Representación de conjuntos en listas ligadas

FIGURA B.7(a)

Ocurrencia sencilla del conjunto DEPT-EMP

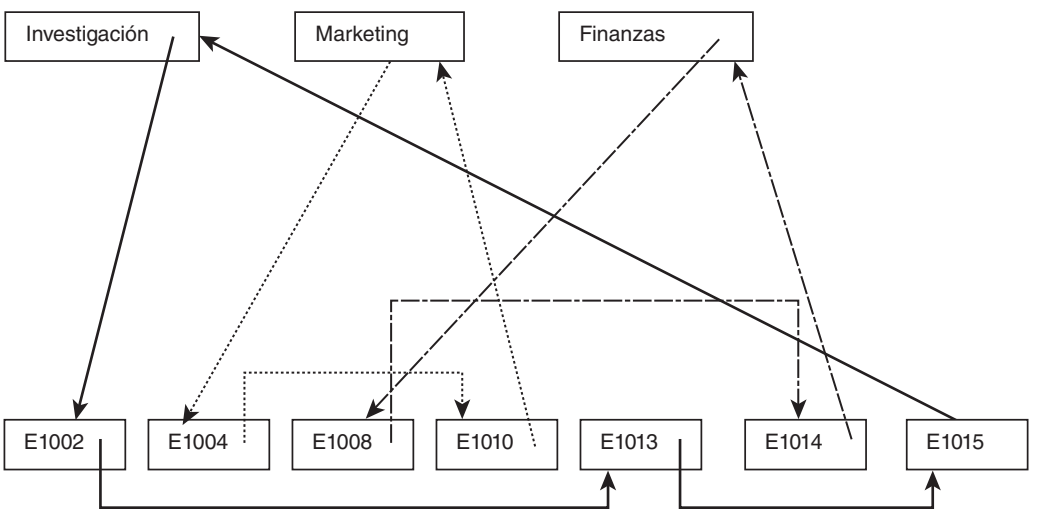


FIGURA B.7 (b)

Ocurrencias múltiples del conjunto DEPT-EMP

requerido incluso para una aplicación simple puede ser bastante pesado. En algunos sistemas de red, el diseñador elige los tipos de punteros al especificarlos en el esquema como el "modo" en la descripción del conjunto.

B.3 DDL DBTG

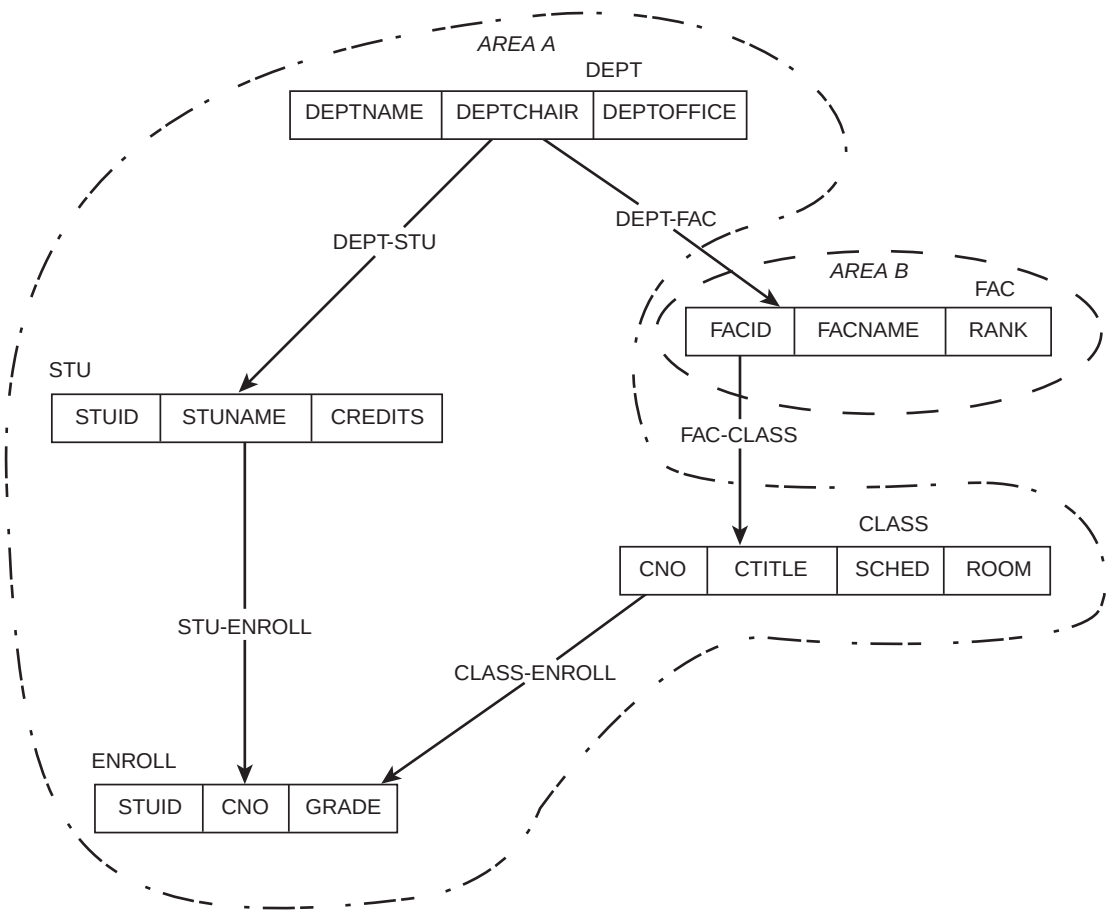
El lenguaje de definición de datos DBTG cambió a través de los años conforme se emitieron reportes adicionales por parte de DBTG y DDLC. Aquí se describen los conceptos más importantes en los estándares, y descripciones del esquema y el subesquema.

B.3.1 El esquema

Un esquema tiene cuatro secciones principales, a saber: la sección esquema, que nombra al esquema; la sección área, que identifica las áreas de almacenamiento y da otra información física; la sección registro, que proporciona una descripción completa de cada estructura de registro con todos sus ítems de datos y detalles de ubicación de registro, y la sección conjunto, que identifica a todos los conjuntos, los tipos propietario y miembro para cada uno, y otros detalles como el ordenamiento.

FIGURA B.8
Diagrama de estructura de datos de red para base de datos University

Con una versión del ejemplo University, y el diagrama de estructura de datos que se muestra en la figura B.8, el esquema aparece en la figura B.9. Las líneas se numeran por conve-



```
1   SCHEMA NAME IS UNIVDATA
2   AREA NAME IS AREAA
3   AREA NAME IS AREAB
4   PRIVACY LOCK IS HUSH
5   RECORD NAME IS DEPT
6           PRIVACY LOCK IS DEAN
7           WITHIN AREAA
8           LOCATION MODE IS CALC USING DEPTNAME
9           01    DEPTNAME PIC IS X(10)
10          01    DEPTCHAIR PIC IS X(15)
11          01    DEPTOFFICE PIC IS X(4)
12  RECORD NAME IS STU
13          WITHIN AREAA
14          LOCATION MODE IS CALC USING STUID
15          01    STUID PIC IS 9(5)
16          01    STUNAME TYPE IS CHARACTER 25
17          01    CREDITS PIC IS 999
18  RECORD NAME IS FAC
19          WITHIN AREAB
20          PRIVACY LOCK IS CHAIR
21          LOCATION MODE IS VIA DEPT-FAC SET
22          01    FACID PIC IS 9(5)
23          01    FACNAME TYPE IS CHARACTER 25
24          01    RANK PIC IS X(10)
25  RECORD NAME IS CLASS
26          WITHIN AREAA
27          PRIVACY LOCK IS REGISTRAR
28          LOCATION MODE IS CALC USING CNO
29          01    CNO PIC IS X(6)
30          01    CTITLE TYPE IS CHARACTER 50
31          01    SCHED PIC IS X(7)
32          01    ROOM PIC IS X(4)
33  RECORD NAME IS ENROLL
34          WITHIN AREAA
35          PRIVACY LOCK IS ADVISE
36          LOCATION MODE IS VIA STU-ENROLL SET
37          01    STUID PIC IS 9(5)
38          01    CNO PIC IS X(6)
39          01    GRADE PIC IS X
40  SET NAME IS DEPT-STU
41          MODE IS CHAIN
42          ORDER IS SORTED INDEXED BY DEFINED KEY
43          OWNER IS DEPT
44          MEMBER IS STU MANUAL OPTIONAL
45          ASCENDING KEY IS STUID
46          SET OCCURRENCE SELECTION IS THRU CURRENT OF SET
47  SET NAME IS DEPT-FAC
48          MODE IS CHAIN LINKED TO OWNER
49          ORDER IS SORTED INDEXED BY DEFINED KEY
50          OWNER IS DEPT
51          MEMBER IS FAC AUTOMATIC MANDATORY
52          ASCENDING KEY IS FACID
53          SET OCCURRENCE SELECTION IS THRU CURRENT OF SET
```

(*continúa*)

FIGURA B.9

Esquema DBTG para la base de datos University

FIGURA B.9
(Continuación)

```
54  SET NAME IS CLASS ENROLL
55          MODE IS CHAIN
56          ORDER IS LAST
57          OWNER IS CLASS
58          MEMBER IS ENROLL AUTOMATIC FIXED
59          SET OCCURRENCE SELECTION IS BY VALUE OF CNO
60  SET NAME IS FAC-CLASS
61          MODE IS CHAIN
62          ORDER IS FIRST
63          OWNER IS FAC
64          MEMBER IS CLASS MANUAL OPTIONAL
65          SET OCCURRENCE SELECTION IS THRU CURRENT OF SET
66  SET NAME IS STU-ENROLL
67          MODE IS CHAIN
68          ORDER IS FIRST
69          OWNER IS STU
70          MEMBER IS ENROLL AUTOMATIC FIXED
71          SET OCCURRENCE SELECTION IS BY VALUE OF STUID
```

niencia para discutir el esquema y no son necesarias. Muchas de las líneas en el ejemplo son opcionales y, dentro de las líneas, muchos de los ítems especificados son opcionales.

La sección esquema

La línea 1, la sección esquema, identifica ésta como una descripción del esquema y da el nombre definido por el usuario de la base de datos, UNIVDATA. Podría proporcionarse una cerradura para todo el esquema, al especificar PRIVACY LOCK IS *nombre*.

La sección área

La línea 2 comienza la sección área opcional, que proporciona una lista de las áreas de almacenamiento para la base de datos. En este caso, existen dos áreas, AREAA y AREAB. Los registros y conjuntos de la base de datos se dividen en áreas que se asignan a diferentes archivos o dispositivos físicos. Un propósito de esta opción es permitir al diseñador especificar cuáles registros deben colocarse en el mismo dispositivo o página, para permitir acceso eficiente de los registros que se relacionan de manera lógica. Por ejemplo, el propietario y miembros de un conjunto se pueden colocar en la misma área si usualmente se acceden en conjunto. Las áreas a las que se accede con frecuencia se colocarían en dispositivos más rápidos, mientras que los que no están en tal demanda pueden estar en los más lentos. Otro beneficio es que diferentes áreas pueden tener distintas restricciones de seguridad, de modo que áreas enteras se pueden ocultar de ciertos usuarios, o áreas que se pueden cerrar al simplificar el control de concurrencia. Por ejemplo, en la línea 4 se proporcionó una cerradura de privacidad, HUSH, para AREAB. No se especifican las actividades para las que se usa la cerradura de privacidad, pero se podría hacer. Las elecciones son UPDATE y/o RETRIEVAL o alguna función de apoyo disponible al ABD, y el acceso puede ser o EXCLUSIVE o PROTECTED para dichas actividades. A fin de especificar el tipo exacto de cerradura se podría sustituir la línea 4 con una especificación tal como PRIVACY LOCK FOR EXCLUSIVE UPDATE IS PROCEDURE KEEPOUT. Esta cerradura exclusiva permitiría a un usuario obtener esta cerradura con propósitos de actualización, a fin de excluir por completo a otros usuarios concurrentes. En el esquema se usó la literal HUSH como la cerradura, pero

en vez de ello se podría elegir un nombre de cerradura o un nombre de procedimiento. En la figura B.8 se indican las áreas mediante líneas punteadas.

La sección registro

Las líneas 5-39 contienen la sección registro. Cada tipo registro se identifica mediante nombre, como se muestra en las líneas 5, 12, 18, 25 y 33. Si se usan áreas, los nombres de áreas se incluirían en alguna parte dentro de la descripción de cada registro. Los datos de colocación del área se pusieron inmediatamente después de la línea de identificación de registro en cada caso, al especificar "WITHIN AREAA" (dentro de área A) o "WITHIN AREAB". La línea 6 proporciona la cerradura de privacidad, DEAN, para el archivo DEPT. Si se desea, se pueden proporcionar cerraduras a nivel registro. Para cada cerradura de registro se tiene una opción de especificar un valor, un nombre de cerradura o un nombre de procedimiento de cerradura. Aquí se aplica la cerradura a todas las actividades que involucran un registro, o sólo a algunas. Las actividades, que se explicarán en la sección B.4, son CONNECT, DISCONNECT, RECONNECT, ERASE, STORE, MODIFY, FIND y GET. Puede especificar cuál de las actividades requiere una cerradura al sustituir la línea 5 con una especificación como PRIVACY LOCK FOR INSERT IS PROCEDURE SECRET.

La línea 8 muestra el **modo location** (ubicación), CALC, para el registro DEPT. El modo location para un tipo registro es el método utilizado por el DBMS para darle una ubicación al registro. Las opciones del modo location incluyen CALC, VIA y DIRECT.

1. CALC, que es el método elegido para el registro DEPT, significa que el DBMS debe usar un esquema de dispersión (hashing) en un campo clave a fin de determinar la dirección física adecuada para el registro. En este caso, el esquema de dispersión se realiza sobre el campo DEPTNAME. Si se quiere indicar que el campo sobre el que se realiza el hashing (llamado clave CALC) es único, se agrega DUPLICATES ARE NOT ALLOWED. La opción CALC permite el acceso a un registro directamente basándose en el valor del campo clave CALC. La línea 14 muestra que se sustituyen los registros STU de acuerdo con el valor de STUID, y la línea 28 muestra que se sustituyen los registros CLASS al calcular las direcciones a partir del campo CNO. La cláusula DUPLICATES ARE NOT ALLOWED FOR *nombre_campo* se puede usar para cualquier campo cuyos valores deben ser únicos, no sólo para campos clave CALC. Ésta es una forma de reforzar una restricción de unicidad.
2. La opción VIA, que se usa para registros FAC y ENROLL, coloca los registros de acuerdo con su pertenencia en algún conjunto. En este caso, el DBMS colocará una instancia registro cerca de su instancia propietaria en un conjunto específico. Para los registros FAC, dado que DEPT-FAC es el único conjunto al que pertenecen, cada registro FAC se colocará cerca del registro del departamento en el que enseña, como se especifica en la línea 21 mediante VIA DEPT-FAC SET. Esto facilita encontrar a todos los miembros de un departamento. Para registros ENROLL, que pertenecen a dos conjuntos, se especifica en la línea 36 que dichos registros se deben colocar vía el conjunto STU-ENROLL, lo que significa que los registros de inscripción de un estudiante se colocarán en la misma página que el registro STU del estudiante. Esto facilitará encontrar los números de curso y calificaciones para todos los cursos que toma el estudiante, al ir a través del registro del estudiante. No es posible acceder a dichos registros directamente mediante valor clave.
3. DIRECT, la tercera opción, significa que la dirección la especificará el programador cuando se inserte el registro. Para recuperar el registro de manera directa, el usuario debe especificar su dirección. Esto significa que el programador acepta la responsabilidad por una tarea que normalmente realiza el DBMS, lo que, por tanto, pierde alguna independencia física de datos.

Al elegir el modo location (ubicación), el diseñador normalmente elige CALC cuando hay una clave y se necesita acceso directo, y elige VIA cuando no hay clave, cuando no es necesario acceso directo o cuando la consulta más común es "¿cuáles son los miembros de esta ocurrencia de conjunto?"

A continuación se describen los ítems de datos individuales. Para DEPT, son DEPTNAME, DEPTCHAIR y DEPTOFFICE. A cada ítem elemental se le asigna un tipo de datos, que puede darse en una cláusula PIC (o PICTURE) que muestre un tipo de datos y (opcional) factor de repetición, o una cláusula TYPE. Los tipos de datos usuales e ítems de formato incluidos en el lenguaje anfitrión están disponibles. Note los números de nivel que preceden a cada ítem. Se elige cualquier número para indicar el nivel más alto de todos y se usa un número mayor para un subnivel. Por ejemplo, si DEPTOFFICE consiste en dos campos, BUILDING y ROOM se escribirían:

```
01 DEPTOFFICE
   02 BUILDING PIC IS X
   02 ROOM PIC IS 999
```

Éste es un ejemplo de un agregado de datos no repetitivo. Para mostrar un ítem de datos o agregado de datos repetitivo es necesario mostrar el número de repeticiones, que puede ser una constante o una variable. Por ejemplo, si decide almacenar en el registro de estudiantes los clubes a los que pertenece un estudiante, agregaría:

```
01 NUMTIMES PIC IS 99
01 CLUBS OCCURS NUMTIMES TIMES
   02 CLUBNAME TYPE IS CHARACTER 20
   02 OFFICEHELD TYPE IS CHARACTER 15
```

En este caso, el número de clubes es una variable cuyo valor se almacena en NUMTIMES. Si el número de repeticiones es constante podría eliminar el campo NUMTIMES y simplemente escribir:

```
01 CLUBS OCCURS 5 TIMES
```

seguido por las líneas CLUBNAME y OFFICEHELD.

Es posible reforzar algunas restricciones de dominio al usar una opción CHECK para restringir el conjunto de valores disponibles para un ítem de datos, como en:

```
01 CREDITS PIC IS 999 CHECK IS LESS THAN 150
```

Este método también se puede usar para reforzar reglas de no nulos, como en:

```
01 STUID PIC IS 9(5) CHECK IS NOT NULL.
```

La sección conjunto

Una vez descritos todos los registros se describen los conjuntos. La sección conjunto en la figura B.9 comienza en la línea 40. Sólo hay tres líneas absolutamente requeridas para un conjunto. Éstas son:

```
SET NAME IS nombre_conjunto
   OWNER IS tipo_registro_propietario
   MEMBER IS tipo_registro_miembro
```

Sin embargo, se puede especificar otra información acerca del conjunto. Por ejemplo, con frecuencia se especifica el **modo**, o método utilizado para vincular registros en la misma ocurrencia de conjunto. Existen dos métodos básicos para conectar el conjunto propietario con los miembros y los miembros entre ellos. El primero es el que se ilustra en la figura B.7, la implementación de lista ligada, en la que el propietario apunta al primer miembro, que

apunta al siguiente, y así por el estilo, con el último miembro que apunta de vuelta al propietario. Esto se indica mediante la entrada:

```
MODE IS CHAIN.
```

Para crear punteros directos de vuelta al propietario, así como al siguiente miembro, se especifica:

```
MODE IS CHAIN LINKED TO OWNER
```

Para crear una lista doblemente ligada, con cada miembro apuntando al previo, se especifica:

```
MODE IS CHAIN LINKED TO PRIOR
```

El segundo método básico de ligar conjuntos miembros es tener un array de punteros en el registro propietario, que apunte directamente a cada miembro, del modo siguiente:

```
MODE IS ARRAY [DYNAMIC]
```

La opción dynamic permite al array crecer según se requiera. Con el modo array, los conjuntos miembros no se ligan directamente unos con otros.

La especificación opcional ORDER se usa para dar el orden lógico de los registros miembro dentro de una ocurrencia de conjunto. Es independiente del modo de ubicación del registro y se refiere al orden lógico creado por los punteros dentro del conjunto. Las dos elecciones básicas para orden, cada una con subopciones, son cronológica y clasificada (sorted). El orden cronológico significa que la posición lógica de un registro dentro de un conjunto está determinada por el momento cuando el registro entró en el conjunto. Dentro del orden cronológico existen cinco opciones.

1. `ORDER IS FIRST`. Esto significa que un nuevo registro miembro siempre se inserta al principio de la cadena puntero, de modo que el propietario apunta primero a él. Esto es apropiado cuando los registros más recientes son de más interés.
2. `ORDER IS LAST`. Esto significa que, cuando se inserta un nuevo registro, se convierte en el último en la cadena puntero, y apunta de vuelta al propietario. Esto se usa cuando no es tan probable que los registros más recientes se recuperen igual que los más antiguos.
3. `ORDER IS NEXT`. Esta opción se utiliza cuando los registros nuevos se insertarán mediante un programa de aplicaciones. Un registro nuevo se colocará después del miembro del conjunto al que se accedió de forma más reciente.
4. `ORDER IS PRIOR`. Esta opción significa que, cuando se inserta un nuevo registro mediante un programa de aplicaciones, se colocará después del miembro del conjunto con acceso más reciente.
5. `ORDER IS IMMATERIAL`. Esta opción significa que el sistema decidirá cuál posición en el conjunto tendrá un nuevo miembro, y lo colocará en la forma más eficiente.

El orden SORTED significa que los registros se colocan en la cadena de acuerdo con valor creciente o decreciente de alguna clave ordenada que es identificada. Si el conjunto es un tipo de un solo miembro, las opciones son simples. Escriba:

```
ORDER IS SORTED BY DEFINED KEY
ASCENDING [/DESCENDING] KEY IS nombre_campo_clave_clasificación
DUPLICATES ARE [NOT ALLOWED/FIRST/LAST]
```

El campo clave de clasificación no necesita tener valores únicos. Si los valores son únicos se dice que los duplicados no se permiten. Si no son únicos se elige si poner el duplicado antes de los valores antiguos (FIRST) o después de los valores antiguos (LAST). Si el conjunto es un tipo conjunto multimiembro existen tres opciones adicionales: `ORDER IS SORTED BY RECORD NAME` (orden por nombre de registro), `ORDER IS SORTED BY DEFINED KEYS`

(orden por claves definidas) y `ORDER IS SORTED WITHIN RECORD NAME` (orden dentro de nombre de registro).

Otra opción que se identifica es la clase de membresía. Debe elegir dos ítems: **set insertion class** (clase de inserción de conjunto) y **set retention class** (clase de retención de conjunto). La inserción de conjunto tiene que ver con cómo los miembros obtienen lugar en la ocurrencia del conjunto, mientras que retención de conjunto tiene que ver con el hecho de que si, una vez colocado en un conjunto, al miembro se le permite dejar el conjunto. Para inserción de conjunto las opciones son:

1. MANUAL, que significa que el miembro se debe colocar en un conjunto mediante un programa o usuario con el uso de un comando CONNECT para ligarlo a la ocurrencia de conjunto adecuada.
2. AUTOMATIC, que significa que cuando un nuevo registro miembro entra a la base de datos, el DBMS automáticamente lo conectará con la ocurrencia de conjunto adecuada, y establecerá de manera automática los punteros necesarios.

Para retención de conjunto, las opciones son:

1. FIXED, que significa que una vez que un miembro se colocó en una ocurrencia de conjunto, nunca puede removerse de dicha ocurrencia particular (a menos, desde luego, que el registro se borre por completo de la base de datos).
2. MANDATORY, que significa que una vez que un miembro se coloca en una ocurrencia de conjunto, debe permanecer en alguna ocurrencia de dicho conjunto, mas no necesariamente en la misma ocurrencia.
3. OPTIONAL, que significa que el registro miembro puede desconectarse por completo del conjunto y se le permite "flotar libremente" en la base de datos.

La cláusula SET OCCURRENCE SELECTION dice cuál método se usa para determinar a cuál ocurrencia de un conjunto debe pertenecer un registro miembro. Para comprender algunas de las opciones aquí, es necesaria la noción de **indicadores de actualidad** (currency indicators). Como se explicó en la sección B.2, el área de trabajo de usuario es un b a través del cual el programador o usuario se comunica con la base de datos. Contiene, entre otras cosas, indicadores de actualidad, que son punteros hacia los registros de la base de datos de varios tipos con acceso más recientes por el programa. Algunos de estos punteros se pueden usar en la selección de ocurrencia de conjunto.

Existen muchas opciones para la selección de conjunto, incluidas:

1. `SET OCURRENCE SELECTION IS THRU CURRENT OF SET` (la selección de ocurrencia de conjunto es a través del conjunto actual). Esta opción se eligió para los conjuntos DEPT-STU y DEPT-FAC. Significa que, cuando se inserta un nuevo registro miembro, el DBMS debe elegir la última ocurrencia del tipo registro propietario como el propietario. Por ejemplo, si se inserta una nueva especialidad en historia, se espera hasta que el programa de aplicación recupere el registro del departamento de historia y luego conecta el registro STU con esta ocurrencia de conjunto, que involucra la inserción de punteros. Para que la inserción de un nuevo registro FAC sea automática, el DBMS lo conectará correctamente a la ocurrencia de conjunto del DEPT actual.
2. `SET SELECTION IS BY VALUE OF` *nombre_campo* (selección de conjunto es por valor de nombre de campo). Con esta opción el DBMS determinará el valor actual del campo nombrado desde el área de trabajo del usuario y lo usará como una clave para encontrar el registro propietario correcto para el nuevo miembro. No es necesario tener de antemano el registro propietario en el área de trabajo, como en la opción anterior. El campo debe ser una clave o campo CALC para el tipo de registro propie-

tario asociado. Por lo general, el registro miembro no contiene la clave, aunque, para los dos ejemplos, CLASS-ENROLL y STU-ENROLL, los registros miembro sí contienen la clave del propietario.

3. `SET SELECTION IS STRUCTURAL` *campo_propietario*`=campo_miembro` (selección estructural). En este caso, el DBMS busca el campo especificado en el registro miembro y usa dicho valor para identificar la ocurrencia correcta del registro propietario. Por lo general, el campo propietario citado es una clave o campo CALC. Esta opción podría usarse para STU-ENROLL al escribir:

```
SET OCCURRENCE SELECTION IS STRUCTURAL STU.STUID= ENROLL.STUID
```

4. `SET OCURRENCE SELECTION IS THRU PROCEDURE FINDREC` (selección de ocurrencia de conjunto a través del procedimiento FINDREC). Aquí, el programa de aplicación FINDREC es responsable de encontrar la ocurrencia propietaria correcta e insertar el miembro en dicha ocurrencia.

Con frecuencia, la línea de selección de conjunto está acompañada por una línea CHECK que asegura el cumplimiento de cierta restricción que involucra al conjunto propietario y al miembro. Por ejemplo, puede tener:

```
CHECK IS STU.STUID=ENROLL.STUID
```

Esto no es necesario cuando se usa la opción de selección STRUCTURAL, porque en dicho caso se comprobó de manera automática.

B.3.2 El subesquema

El subesquema es una descripción de la visión externa de un usuario particular o conjunto de usuarios. La figura B.10(a) muestra un diagrama para un subesquema llamado COLLEGE. Proporciona independencia de datos al permitir que algunos ítems difieran del esquema. Del subesquema se pueden omitir áreas, registros, conjuntos e ítems de datos del esquema. Los ítems de datos, registros y conjuntos se pueden renombrar.

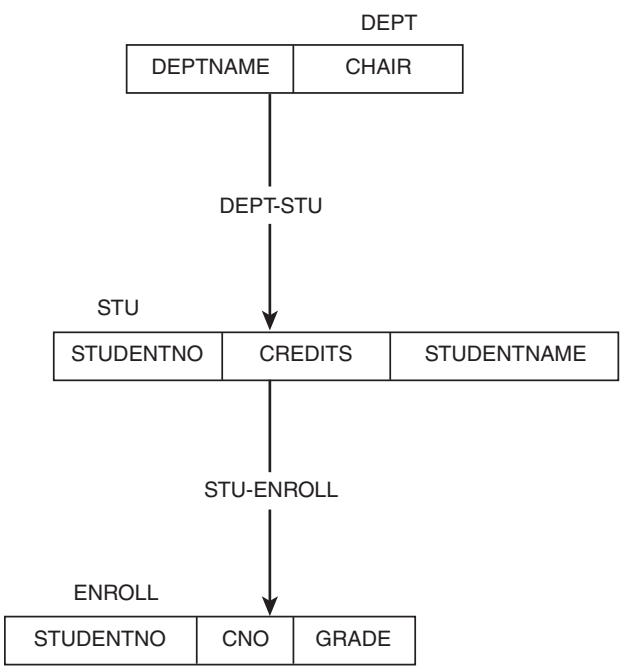


FIGURA B.10

Subesquema para el ejemplo University

FIGURA B.10(a)

Diagrama de estructura de datos para el subesquema COLLEGE

FIGURA B.10(b)
Subesquema para COLLEGE

```
TITLE DIVISION
SS    COLLEGE WITHIN UNIVDATA
MAPPING DIVISION
ALIAS SECTION
AD        STUDENTNO IS STUID
AD        STUDENTNAME IS STUNAME
STRUCTURE DIVISION
RECORD SECTION
01        DEPT
          02        DEPTNAME        PIC X(10)
          02        DEPTCHAIR       PIC X(15)
01        STU
          02        STUDENTNO       PIC 9(5)
          02        CREDITS         PIC 999
          02        STUDENTNAME     PIC X(15)
01        ENROLL ALL
SET SECTION
SD        DEPT-STU
SD        STU-ENROLL
```

Es posible cambiar el orden de los ítems de datos en un registro. Los ítems de datos elementales que no se agruparon en el esquema se pueden agrupar, o los ítems agrupados se pueden dividir en ítems de datos individuales. Los tipos de datos de algunos ítems suelen modificarse al cambiar las especificaciones PIC, TYPE o de longitud, y se pueden formar registros virtuales con el uso de ítems de datos a partir de registros conceptuales separados. El formato exacto del subesquema DDL depende del lenguaje anfitrión, de modo que no se usa una forma única. Los subesquemas usualmente contienen las siguientes divisiones, como se muestra en la figura B.10(b):

1. División TITLE (título), que proporciona el nombre del subesquema y del esquema asociado.
2. División MAPPING (mapeo), que proporciona los cambios en nombres del esquema al subesquema, si hay alguno.
3. División STRUCTURE (estructura), que se subdivide en las secciones AREA, RECORD y SET.
   a. La sección Area (si hay alguna) da los nombres de las áreas a incluir en el subesquema.
   b. La sección Record menciona los registros e ítems de datos dentro de ellos que se incluirán, posiblemente con sus nuevos nombres (de la división Mapping) y tipos de datos.
   c. La sección Set cita todos los conjuntos que se incluirán.

B.4 Manipulación de datos DBTG

B.4.1 Área de trabajo del usuario e indicadores de actualización

El DML DBTG es un lenguaje procedural que requiere que el usuario esté familiarizado con la estructura de la base de datos. Existen muchas opciones disponibles y el programador habilidoso puede usarlos para navegar en la base de datos de manera muy eficiente. Sólo se

considerará un pequeño subconjunto de los muchos comandos disponibles en el DML. Una comprensión de las operaciones básicas de estos comandos requiere más detalles acerca del área de trabajo del usuario. El área de trabajo del usuario es un búfer asignado a un programa. Contiene lo siguiente:

1. **Indicadores de actualización.** El área de trabajo tiene un conjunto de punteros que identifican los registros con acceso más reciente por parte del programa en las siguientes categorías:
   a. **Actualización de la unidad que se ejecuta.** Un puntero hacia el registro más reciente de cualquier tipo al que accedió el programa.
   b. **Actualización del tipo de registro.** Para cada tipo de registro identificado en el subesquema por el programa, el área de trabajo contiene la dirección del registro de dicho tipo con acceso más reciente por parte del programa.
   c. **Actualización del tipo de conjunto.** Para cada tipo de conjunto, el área de trabajo contiene la dirección del registro más reciente de dicho conjunto, ya sea el propietario o un miembro de una ocurrencia.
   d. **Actualización del área**. Para cada área o reino, el área de trabajo del usuario muestra la dirección del registro con acceso más reciente de dicha área, sin importar su tipo.
2. **Plantillas de registro**, una para cada tipo de registro al que el programa puede acceder. Conforme los registros de la base de datos se llevan al búfer, deben colocarse en las plantillas de registro adecuadas. Para la base de datos University, existen plantillas para registros DEPT, STU, FACULTY, CLASS y ENROLL.
3. **Banderas de estatus**, que contienen valores que muestran si fue exitosa la última operación de base de datos intentada. Un valor de 0 significa éxito, mientras que otros valores indican errores de varios tipos.

Por ejemplo, la figura B.11 muestra los indicadores de actualización para un programa que tiene acceso a toda la base de datos University. Para esta aplicación se tendría una actualiza-

	t=1	t=2	t=3
Tiempo actual	**FIND HISTORY**	**FIND F101**	**FIND S1001**
Unidad que se ejecuta	History	F101	S1001
DEPT	History	History	History
STU			S1001
FACULTY		F101	F101
CLASS			
ENROLL			
DEPT-STU	History	History	S1001
DEPT-FAC	History	F101	F101
STU-ENROLL			S1001
FAC-CLASS		F101	F101
CLASS-ENROLL			
AREAA	History	History	S1001
AREAB		F101	F101

FIGURA B.11
Indicadores de actualización para la aplicación University

ción de la unidad que se ejecuta; actualización de tipos de registro DEPT, STU, FACULTY, CLASS y ENROLL; actualización de los tipos de conjunto DEPT-STU, DEPT-FAC, STU-ENROLL, FAC-CLASS y CLASS-ENROLL; y actualización de las dos áreas, AREAA y AREAB. En la figura se muestran los cambios a los indicadores conforme el programa accede a diferentes registros en el tiempo 1, 2, 3, etc. Cuando el programa se inicia, todos los indicadores de actualización son nulos. Cuando el registro del departamento de historia se encuentra en el tiempo t = 1, la actualización de la unidad que se ejecuta, el tipo de registro DEPT, el conjunto DEPT-STU, el conjunto DEPT-FAC y AREAA se establecen todos para apuntar al registro del departamento de historia. Los indicadores de actualización para otros tipos de registro (STU, FACULTY, CLASS, ENROLL); los indicadores de actualización para los conjuntos en los que DEPT no participa (STU-ENROLL, FAC-CLASS, CLASS-ENROLL); y el indicador de actualización para cualquier área que no contiene un registro DEPT (AREAB) no resultan afectados. A continuación el programa encuentra el registro del miembro docente F101 en el tiempo t = 2. Esto se convierte en la actualización de la unidad que se ejecuta, el tipo de registro FACULTY, el conjunto DEPT-FAC, el conjunto FAC-CLASS y AREAB. Ningún otro indicador de actualización resulta afectado. Finalmente, el programa encuentra el registro del estudiante S1001 en el tiempo t = 3. Se convierte en la actualización de la unidad que se ejecuta, el tipo de registro STU, el conjunto DEPT-STU, el conjunto STU-ENROLL y AREAA, aunque ningunos otros punteros resultan afectados.

B.4.2 Comandos DML

Los principales comandos DML DBTG y sus funciones son los siguientes:

- **OPEN:** abre área(s) para procesamiento
- **CLOSE:** cierra área(s) cuando el procesamiento está completo
- **FIND:** encuentra la ubicación de un registro de base de datos
- **GET:** lleva el registro encontrado anteriormente al área de trabajo
- **MODIFY:** almacena los cambios a un registro actualizado en el área de trabajo
- **ERASE:** borra un registro de la base de datos
- **STORE:** agrega un nuevo registro a la base de datos
- **CONNECT:** coloca un nuevo miembro en una ocurrencia de conjunto
- **DISCONNECT:** remueve un miembro de una ocurrencia de conjunto
- **RECONNECT:** mueve un miembro de una ocurrencia de conjunto a otro

Cada uno de estos principales comandos DML tiene muchas diferentes formas y muchas opciones dentro de cada forma. Los comandos OPEN (o READY) y CLOSE (o FINISH) se refieren a áreas, y ya no aparecen en el lenguaje. Si se requiere, un comando OPEN se debe escribir para abrir todas las áreas a las que accederá el programa, y un comando CLOSE debe aparecer cuando el procesamiento se complete. Se considerarán algunas operaciones relativamente simples usando los otros comandos.

Find

Este comando simplemente ubica un registro en la base de datos. No lleva el registro a la plantilla del área de trabajo, pero hace que los indicadores de actualización relevantes cambien para reflejar la dirección del registro encontrado. El registro encontrado se convierte en la actualización de la unidad que se ejecuta, tipo de registro y tipo de conjunto para todos los conjuntos en los que participa o como propietario o como miembro. El comando

FIND usualmente se sigue por GET, MODIFY o ERASE. En una versión del FIND, primero se debe especificar el valor del campo CALC en un registro cuyo modo de ubicación es CALC, y luego pide al sistema encontrar (FIND) un registro con un valor coincidente. Se usaría el lenguaje anfitrión para asignar el valor de campo y luego escribir el FIND:

```
STUID = 'S1015'
FIND ANY STU USING STUID
```

La primera línea, en el lenguaje anfitrión, establece el valor de STUID, y la segunda línea pide al sistema de gestión de base de datos ubicar el primer registro STU con un STUID coincidente. De modo que FIND ANY realmente significa "encuentra el primero". Si especifica que no se permiten duplicados y quiere continuar buscando un segundo o registros posteriores después de encontrar el primer registro, escriba:

```
FIND DUPLICATE STU USING STUID
```

Este comando se puede usar en un bucle para continuar la búsqueda. Si quiere actualizar un registro use una forma ligeramente diferente del FIND, a saber:

```
FIND FOR UPDATE ANY STU USING STUID
```

o

```
FIND FOR UPDATE DUPLICATE STU USING STUID
```

El comando FIND se puede usar para ubicar registros de acuerdo con la membresía de conjunto, y es en especial útil cuando el modo location es VIA. Existen muchos métodos para ubicar miembros de conjunto, pero uno simple involucra encontrar al propietario de una ocurrencia de conjunto, y luego pide el miembro FIRST, NEXT, PRIOR, *n*-ésimo o LAST. El siguiente segmento ilustra cómo podría usar un registro STU para encontrar los registros ENROLL asociados.

```
MOVE 'S1002' TO STUID IN STU
FIND ANY STU USING STUID
FIND FIRST ENROLL WITHIN STU-ENROLL
DOWHILE (DBSTATUS = 0)
     //Lleva el registro al área de trabajo y lo procesa
     FIND NEXT ENROLL WITHIN STU-ENROLL
ENDDO
```

Una vez que encuentra un miembro de conjunto puede pedir su propietario dentro de cualquier conjunto. Por ejemplo, cuando tiene alguno de los registros ENROLL en el segmento anterior, pide la clase (CLASS) asociada al escribir:

```
FIND OWNER WITHIN CLASS-ENROLL
```

Get

Este comando, que sigue a FIND, lleva el registro encontrado al área de trabajo, donde el usuario puede realizar procesamiento adicional. Como ejemplo de GET, si quiere imprimir una lista de ID y nombres de todos quienes tienen especialidad en matemáticas, escriba:

```
DEPT.DEPTNAME = 'Math'
FIND ANY DEPT USING DEPTNAME
FIND FIRST STU WITHIN DEPT-STU
DO WHILE (DBSTATUS = 0)
     GET STU
     PRINT (STUID, STUNAME)
     FIND NEXT STU WITHIN DEPT-STU
END
```

Modify

Si un registro se ha de actualizar, use una forma especial del comando FIND: FIND FOR UPDATE. Entonces se hace un GET para llevar el registro al área de trabajo, donde se usa el lenguaje anfitrión para realizar los cambios. Después de actualizar el registro en el área de trabajo, el comando MODIFY se usa con el fin de sustituirlo en la base de datos. Para actualizar la categoría del docente F105 se encuentra para actualizar el registro FAC apropiado, se le lleva al área de trabajo con el uso de GET, se actualiza con el lenguaje anfitrión y se le coloca de vuelta en la base de datos con el comando MODIFY, del modo siguiente:

```
FAC.FACID = 'F105'
FIND FOR UPDATE ANY FAC USING FACID
GET FAC
RANK = 'Assistant'
MODIFY FAC
```

Erase

Para usar el comando ERASE, debe utilizar FIND FOR UPDATE en el registro deseado. Una vez encontrado el registro, bórrelo de la base de datos al escribir ERASE, como se muestra en el siguiente segmento:

```
MOVE 'ART103A' TO CNO IN CLASS
FIND FOR UPDATE ANY CLASS USING CNO
ERASE
```

Cuando borra un registro que es propietario de una instancia de conjunto, ¿qué ocurre con los miembros del conjunto? Cuando se borra el registro de ART103A surge un problema de integridad referencial, porque existen algunos registros de inscripción con este número de curso. Puede asegurar el borrado de todos los miembros de alguna instancia de conjunto propiedad de un registro al escribir ERASE ALL en lugar de ERASE. Si sólo escribe ERASE, como en el segmento anterior, el comando tendrá diferentes efectos sobre distintos conjuntos, dependiendo del estatus de retención de los registros miembro. Si en el esquema se especifica FIXED, todos los miembros se borrarán cuando el propietario se borre. Éste es el caso para CLASS-ENROLL, de modo que automáticamente se perderán todos los registros de inscripción cuando un curso se borre. Si la clase de retención es MANDATORY, el ERASE es ilegal y el sistema rechazará ejecutarlo hasta que todos los miembros del conjunto hayan sido removidos y reconectados a otra instancia. Para una clase de retención OPTIONAL, el propietario se borra y los miembros se liberan de la instancia, pero permanecen en la base de datos. Puesto que un registro podría ser un propietario de muchos conjuntos diferentes, cada uno con su propia clase retención, su borrado podría tener diferentes efectos sobre distintos conjuntos. Si un registro borrado sólo es un miembro del conjunto, no un propietario, los punteros simplemente se reajustan para conectar los miembros restantes.

Store

Este comando inserta nuevos registros en la base de datos. Primero debe construir el registro en el área de trabajo, con el lenguaje anfitrión, y luego colocarlo en la base de datos mediante STORE. Para crear e insertar un nuevo registro STU escriba:

```
STU.STUID = 'S1050'
STU.STUNAME = 'Marks,Joseph'
CREDITS = 0
STORE STU
```

Cuando inserta este registro, no pertenece a alguna instancia de conjunto. Colocar el registro de estudiante en la instancia de conjunto DEPT-STU adecuada requiere un comando CONNECT.

Connect

Si un registro tipo miembro ya está en la base de datos, mas no conectado a un conjunto, o porque su clase de inserción es MANUAL o porque su clase de retención es OPTIONAL, y se desconectó de una instancia de conjunto, entonces puede usar el comando CONNECT para colocarlo en una instancia de conjunto. Recuerde que, si la clase de inserción es AUTOMATIC, el DBMS es responsable de insertarlo en la instancia de conjunto y, una vez insertado por cualquier medio, si la clase de retención es FIXED o MANDATORY, el registro miembro no se puede desconectar del conjunto. Si se coloca inmediatamente después el siguiente segmento del segmento de programa previo, causaría que el registro del estudiante recién creado se inserte en la instancia de conjunto DEPTSTU para el departamento de historia.

```
DEPT.DEPTNAME = 'History'
FIND ANY DEPT USING DEPTNAME
CONNECT STU TO DEPTSTU
```

En general, para conectar un miembro a una instancia, debe tener al propietario de la instancia como al actual de su tipo de registro. El efecto de CONNECT es ajustar los punteros de modo que el registro se coloque en la posición lógica correcta en la instancia correcta.

Disconnect

Este comando sólo funciona para miembros de conjunto cuya clase de retención sea OPTIONAL. Remueve dichos registros de la instancia de conjunto, aunque los deja en la base de datos. Los registros todavía pueden ser miembros de otros conjuntos, o bien no pertenecer a conjunto alguno. Para remover un estudiante de un departamento, escriba:

```
STU.STUID = 'S1050'
FIND FIRST STU USING STUID
DISCONNECT STU FROM DEPTSTU
```

Esta operación está permitida pues STU tiene clase de retención OPTIONAL en DEPTSTU.

Reconnect

Si un miembro tiene retención MANDATORY u OPTIONAL se puede mover de una instancia de conjunto a otra mediante un comando RECONNECT. El siguiente segmento mueve F105 de CSC a Math:

```
FAC.FACID = 'F105'
FIND FIRST FAC USING FACID
DEPT.DEPTNAME = 'Math'
FIND FIRST DEPT USING DEPTNAME
RECONNECT FAC IN DEPTFAC
```

Se examinaron los principales comandos, considerando al menos una forma para cada uno. Sin embargo, existen muchas opciones adicionales, en particular para el comando FIND, que se describen en los propósitos DBTG o se usan en varios sistemas de gestión de base de datos basadas en red. Elegir diferentes formas del mismo comando puede tener un tremendo impacto sobre el rendimiento, y los programadores habilidosos usan su conocimiento de la estructura de almacenamiento para hacer la manipulación de datos tan eficiente como sea posible.

B.5 Conversión de EE-R y modelo de red

B.5.1 Mapeo de red a EE-R

Puesto que muchas bases de datos basadas en red son sistemas heredados, los diseñadores de bases de datos en ocasiones tienen la tarea de convertir dichos sistemas a otros modelos, como el relacional o el orientado a objetos. Con frecuencia, sólo están disponibles el esquema y subesquemas, no la documentación original del diseño, y es necesario algo de ingeniería inversa de software para determinar la estructura de la base de datos. Una conversión de modelo se debe realizar al dibujar el diagrama de estructura de los datos en red que correspondan al esquema, y convertir dicho diagrama a un diagrama EE-R o una representación similar del modelo lógico. El diagrama EE-R se puede mapear entonces del modo usual. Es una tarea simple y directa traducir un esquema a un diagrama de red, como se ilustra por la correspondencia entre las figuras B.9 y la B.8. La tarea de convertir un modelo de datos en red en un diagrama EE-R también es relativamente directa. Los tipos de registro se convierten en entidades, y los conjuntos se convierten en relaciones. Del mapeo resulta un diagrama EE-R simple y útil. Sin embargo, el diseñador debe examinar cuidadosamente el esquema para determinar si alguna información semántica se pierde en la conversión. Debe considerar los siguientes casos excepcionales para recapturar esta información:

- Los agregados de datos se deben convertir a atributos estructurados o atributos repetitivos, dependiendo de su forma.
- Los ítems de datos virtuales se deben convertir a atributos calculados.
- Los registros de intersección, identificados mediante la presencia de las claves de al menos otros dos registros y conjuntos acompañantes en los que dichos registros aparezcan como propietarios, se deben sustituir por relaciones muchos a muchos.
- Para cualquier conjunto basado en valor, donde la clave del propietario aparezca en los registros miembros, la relación uno a muchos que sustituye el conjunto elimina la necesidad de tener la clave del propietario en el registro miembro en el modelo EE-R. Si ocasionalmente el diagrama se mapea a un modelo relacional se usará una clave externa y el atributo restaurado. Para un modelo objeto-relacional u orientado a objeto se puede usar una referencia.
- Los conjuntos esenciales se deben examinar para ver si el miembro en la relación del conjunto representa una entidad débil.
- Si existe algún registro ligado que consista sólo de punteros, los conjuntos en los que aparezca como miembros se deben examinar para ver si dichos registros representan relaciones ternarias o de orden superior. Los registros ligados también pueden tener algunos atributos descriptivos, que se convierten en atributos de la relación.
- Cualquier conjunto en el que los miembros tengan membresía automática deben examinarse para ver si existe una jerarquía en la que el propietario del conjunto sea un padre.

B.5.2 Mapeo de EE-R a red

Para mapear un diagrama EE-R a un diagrama de estructura de datos en red, el plan básico es hacer a cada entidad un tipo de registro, cada atributo un ítem de datos en un registro, y cada relación un tipo de conjunto. Sin embargo, existen algunas dificultades introducidas por relaciones no binarias, relaciones muchos a muchos, atributos de relaciones, entidades débiles y jerarquías.

- Para el caso simple en el que entidades fuertes se relacionen mediante una relación binaria 1 a muchos sin atributos descriptivos, cada entidad se convierte en un tipo de registro y la relación se mantiene al hacer la entidad "uno" la propietaria de un tipo conjunto en el que la entidad "muchos" sea miembro. Los agregados de datos se pueden usar para representar atributos compuestos o repetitivos, y los campos calculados pueden sustituirse mediante ítems de datos virtuales.
- Para relaciones binarias 1 a 1 sin atributos descriptivos, las dos entidades se pueden combinar en un solo tipo registro, o un tipo conjunto se puede crear con una entidad como el registro propietario y el otro como el tipo de registro miembro.
- Para resolver una relación binaria muchos a muchos, se construyen **registros de intersección** que consisten en las claves de las entidades asociadas y cualquier atributo descriptivo. Las entidades originales se convierten en registros propietarios de conjuntos en los cuales el registro de intersección es el tipo miembro. Esta técnica se describe con detalle en la sección B.2.
- Para relaciones binarias uno a muchos con información de intersección, deberá considerar cuidadosamente si los datos podrían colocarse en el registro para el lado "muchos". Si no, debido al hecho de que el puntero de conjunto no puede contener dato alguno, es necesario crear un nuevo tipo de registro, llamado registro **link** (vinculación) o **junction** (combinación), que consiste sólo en la información de intersección de los ítems de datos. Este registro link se convertirá en el tipo miembro en dos tipos de conjunto, cada uno propiedad de los registros de entidad originales. Tal vez quiera colocar punteros padre directos en los registros link.
- Para resolver relaciones ternarias, existen dos técnicas. El método más simple, posiblemente útil si no existen atributos descriptivos, es sustituir dos o tres relaciones ternarias para la ternaria. Por ende, si las entidades A, B y C se relacionan mediante la relación A-B-C, puede preservar la información al formar las dos relaciones binarias A-B y B-C, o las tres relaciones binarias A-B, A-C y B-C. La otra posibilidad es crear un nuevo tipo de registro, llamado registro **link** o **dummy** (ficticio), que consiste sólo de punteros sin campos de datos o con un campo subrogado generado por el sistema. Si existen atributos descriptivos para la relación se pueden agregar campos para ellos al registro link. Entonces cada una de las entidades se convierte en un registro que es propietario de un registro donde el registro link es el tipo miembro.
- Para relaciones generales *n*-arias se extiende el método descrito anteriormente para relaciones ternarias.
- Para entidades débiles puede usar conjuntos esenciales para conectar entidades propietarias a entidades miembro débiles, o "migrar" la clave de la entidad fuerte al registro de entidad débil y usar una restricción estructural, si está disponible, para reforzar la integridad.
- Cuando el diagrama EE-R contenga una relación de generalización, existen tres formas de proceder. La primera es que cada entidad se convierta en un tipo de registro, y convertir la entidad de nivel superior en la propietaria de un tipo conjunto multimiembro donde los registros de entidad de nivel inferior sean los miembros. El conjunto debe tener clase AUTOMATIC FIXED o AUTOMATIC MANDATORY. Alternativamente, convierta cada entidad en un tipo de registro, y separe los tipos de conjunto para cada una de las entidades subtipo, todas con clase AUTOMATIC FIXED. Una segunda posibilidad es tener un solo tipo de registro para toda la generalización, con la entidad de nivel superior representada por atributos comunes y las entidades subtipo representadas mediante campos variantes. Cómo se hace esto depende del lenguaje anfitrión. La tercera alternativa es crear tipos de registro sólo

para los subtipos, y repetir los atributos de la entidad de nivel superior en el registro de cada subtipo. Cada una de estas opciones pierde cierta información acerca de la relación entre los elementos de la generalización.

Excepto por estas situaciones, la transformación de un diagrama EE-R en un diagrama de estructura de datos en red es bastante directa, con registros que sustituyen las entidades y conjuntos que sustituyen las relaciones.

APÉNDICE C: El modelo jerárquico

Como el modelo de red, el modelo jerárquico se remonta a la década de 1960. Information Management System (**IMS**, sistema de gestión de información), uno de los dos primeros grandes DBMS, lo desarrollaron North American Aviation e IBM para el proyecto de alunizaje Apollo. IMS se convirtió en el sistema de gestión de base de datos jerárquico dominante en el mercado y durante muchos años fue el más ampliamente usado de todos los DBMS. Todavía existen muchas bases de datos heredadas en uso que se gestionan mediante IMS.

C.1 Estructuras de datos jerárquicas

La estructura utilizada por las bases de datos jerárquicas es familiar a los usuarios de computadoras de estructuras de archivos jerárquicas. El modelo jerárquico no es tan flexible o fácil de comprender como el modelo relacional. Utiliza el **árbol** como su estructura de datos básica. Como se describió en la sección 13.4, en la discusión del modelo de datos semiestructurado que también tiene una representación jerárquica, un árbol es un gráfico dirigido que consiste en una jerarquía de nodos, con un solo nodo, llamado **raíz**, en el nivel más alto.

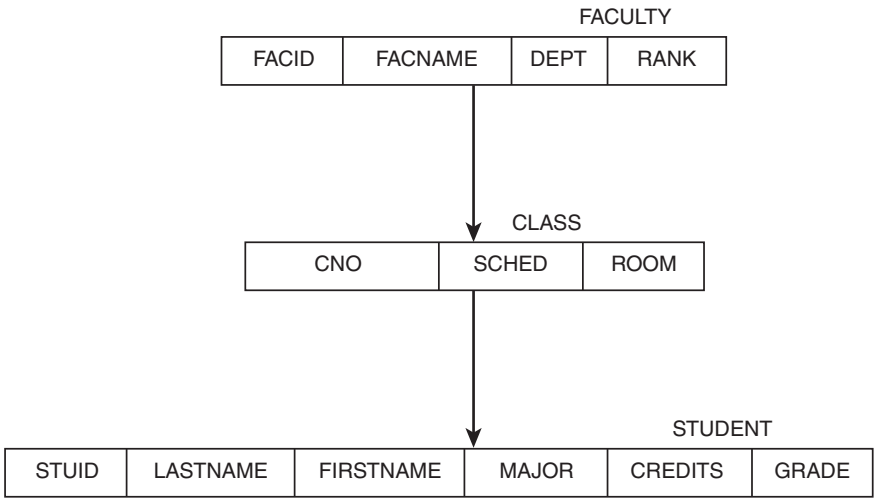
La sección A.2.3 del apéndice A contiene una descripción más completa de los árboles. Un árbol se puede representar en almacenamiento en una forma **plana** (flat) al colocar datos de nodos dependientes después de los datos de nodo padre. Un método de lectura y recorrido de todos los nodos del árbol se llama **recorrido preordenado** (preorder traversal) y se usa en esta representación. Este recorrido se puede resumir mediante las siguientes reglas, a partir de la raíz:

1. Lea el nodo si todavía no se ha leído,
2. De otro modo, lea el hijo más a la izquierda que todavía no se ha leído,
3. De otro modo, si no quedan descendientes para leer, regrese al nodo padre y repita este procedimiento sobre él.

Para recorrer el árbol, comience en la raíz y recorra sus subárboles de izquierda a derecha, de manera recursiva. Comience en la raíz, luego lea al hijo más a la izquierda, luego su hijo más a la izquierda y continúe de esta forma hasta que llegue a una hoja en la rama más a la izquierda. Luego suba un nivel hacia el padre de dicha hoja, y desde ahí proceda hacia su hijo no leído más a la izquierda. Continúe hasta leer cada nodo. Cuando lea un nodo por primera vez, imprímalo. El recorrido preordenado del árbol en la figura A.12 produce

```
A B E K L F C G M D H N I J
```

FIGURA C.1
Representación jerárquica de la base de datos University



En el modelo de base de datos jerárquica, una base de datos consiste en una colección de instancias de **un solo tipo de árbol** (es decir, un solo diagrama de estructura árbol). El árbol no necesita ser o binario o equilibrado. En este modelo, cada nodo o **segmento** contiene uno o más **campos** o ítems de datos que representan atributos que describen una entidad. Todo el árbol puede ser una descripción de una sola entidad, o bien usarse diferentes segmentos dentro de un árbol para representar distintas entidades, pero relacionadas. La figura C.1 muestra un posible diagrama de estructura de datos para un árbol que representa el ejemplo University. En esta estructura se eligió hacer FACULTY el nodo o segmento raíz, con sus campos FACID, FACNAME, DEPT y RANK. CLASS se representa como un hijo de FACULTY, el cual muestra que hay una relación 1 a muchos entre cada registro FACULTY y sus registros CLASS asociados. Es posible visualizar esta relación al decir que los registros CLASS no existen independientemente, sino que están anidados dentro de los registros FACULTY. Note que se omite el campo FACID del segmento CLASS. Puesto que la clase está dentro de un registro faculty particular, no hay necesidad de repetir el FACID. Su valor se "hereda" del padre. También se tiene el registro STUDENT como un segmento del árbol, y se representa como un hijo de CLASS. Al segmento STUDENT se agrega GRADE pues no hay ambigüedad acerca del curso al cual se refiere la calificación del estudiante.

Entre CLASS y STUDENT se implica una relación 1 a muchos. Se puede visualizar el registro STUDENT como anidado dentro del registro CLASS. Al colocar el registro STUDENT dentro del registro CLASS, que a su vez está dentro del registro FACULTY, se muestra cuáles clases imparte un miembro del personal docente, y cuáles estudiantes están inscritos en dichas clases, de modo que ya no se necesitan los registros ENROLL. Esto demuestra que las relaciones se representan en el modelo jerárquico mediante colocación física de registros o punteros que simulan tal colocación, en lugar de por los datos mismos.

La figura C.2 muestra una instancia del árbol FACULTY-CLASS-STUDENT. En una instancia del árbol, un nodo hijo puede aparecer cero, una o muchas veces, aun cuando ocurra exactamente una vez en el diagrama de estructura de árbol. En este ejemplo, para un registro FACULTY dado, puede haber tantos segmentos CLASS como se necesite, y cada uno de éstos puede tener un número arbitrario de segmentos STUDENT. Múltiples apariciones del mismo tipo de nodo hijo dentro de la misma instancia del segmento padre se llaman **gemelos**. Por tanto, los dos segmentos CLASS son gemelos, porque tienen la misma instancia de segmento padre, el segmento FACULTY de F110. Además, los registros de los estudiantes S1002 y S1010 son gemelos, porque tienen el mismo padre, MTH103C. El registro del estu-

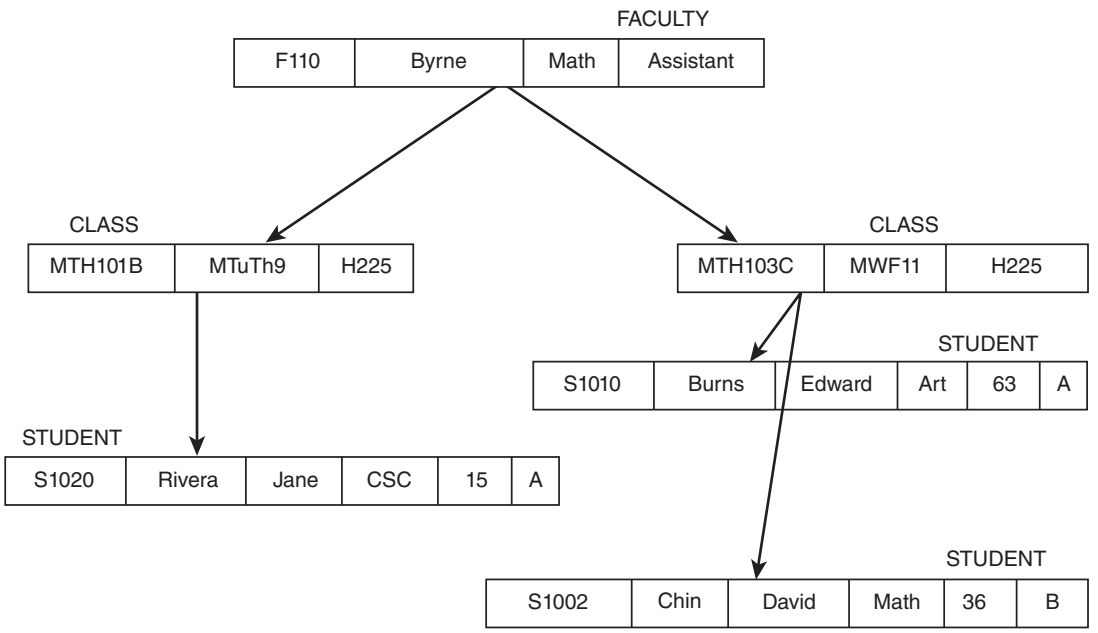


FIGURA C.2
Una instancia del árbol University

diante S1020 no es gemelo de otros dos registros de estudiante, pues tiene una instancia de padre diferente, a saber, MTH101B. En una instancia de árbol, los hermanos son segmentos que son de diferentes tipos, pero tienen la misma instancia de segmento padre. Si un segmento particular no aparece en una instancia, ninguno de sus segmentos dependientes puede aparecer. Para el ejemplo, esto significa que ningún segmento de estudiante puede aparecer sin un correspondiente segmento class, ni puede aparecer una clase sin un correspondiente segmento faculty.

Esta instancia se puede representar en almacenamiento al usar recorrido preordenado para crear una representación plana del registro, como se muestra en la figura C.3. En dicha representación, se ve que el segmento FACULTY aparece primero, seguido por el primer segmento CLASS, que es seguido por el primer STUDENT en dicha clase. Puesto que sólo hay un estudiante en la clase, continúe con el segundo segmento CLASS, seguido por el primer STUDENT en dicha clase, luego el segundo STUDENT. Si hubiera otro segmento CLASS para este miembro del personal docente, aparecería a continuación. Dado que no hay más clases para el profesor Byrne, sigue el próximo registro FACULTY. Note que cada nuevo segmento raíz señala el comienzo de una nueva instancia del árbol. Para este ejemplo, existen tantas instancias como miembros del personal docente. El DBMS sabe que un segmento particular es, por ejemplo, un segmento faculty en lugar de otro segmento student. En el modelo jerárquico, a cada tipo de segmento se le asigna un **código tipo** único, de acuerdo con su posición en el diagrama de estructura de árbol. Los códigos tipo están determinados por el recorrido preordenado de la estructura del árbol, de modo que la raíz tiene código tipo 1, su hijo más a la izquierda tiene código tipo 2, el hijo más a la izquierda de dicho hijo tiene código tipo 3, etc. Para el ejemplo de la figura C.1, FACULTY tiene código tipo 1, CLASS tiene código tipo 2 y STUDENT tiene código tipo 3. Para la estructura de árbol en la figura A.12, los códigos tipo son los siguientes:

```
NODO        A B E K L F C G M D  H  N  I  J
CÓDIGO TIPO 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

	Faculty			Class			Student					Class		
....	F110	Byrne	Assistant	MTH101B	MTuTh9	H225	S1020	Rivera	Jane	CSC	A	MTH103C	MWF11	H225

	Student						Student						
	S1002	Chin	David	Math	36	B	S1010	Burns	Edward	Art	63	A	*siguiente registro faculty...*

FIGURA C.3

Representación plana de árbol

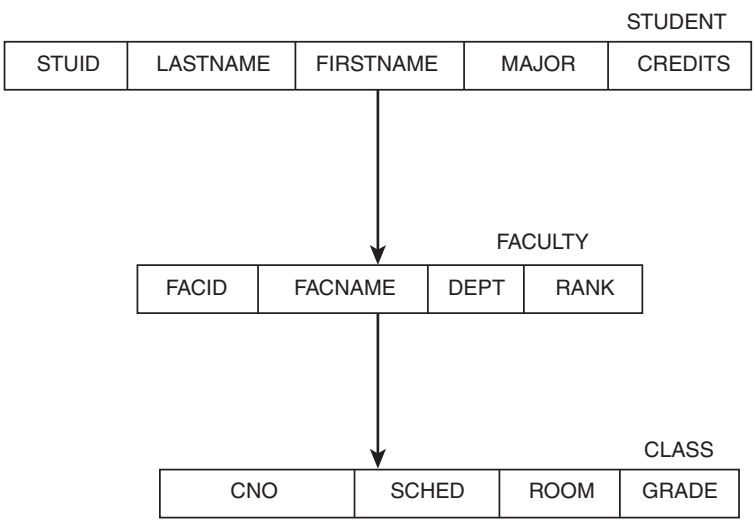
En almacenamiento, el código tipo, junto con otra información de cabecera, se almacena como **prefijo** para el segmento, pero no lo ve el usuario. Esto permite al DBMS distinguir segmentos uno de otro. Cada nueva instancia de árbol comienza con un código tipo 1.

La estructura que se muestra en la figura C.1 representa a todos los registros en la base de datos como información acerca de los miembros del personal docente. No hay registros CLASS o STUDENT independientes, porque éstos son segmentos dependientes de un árbol en el que FACULTY es el nodo raíz. Las relaciones entre estos segmentos se representan mediante posiciones en el árbol. La estructura en este ejemplo debe facilitar la respuesta a las preguntas: "encuentre información acerca de todas las clases impartidas por el profesor X" y "encuentre información acerca de todos los estudiantes en las clases del profesor X". Para responder cualquiera de éstas se encuentra el registro del profesor X y se lee. Sin embargo, es muy difícil responder la pregunta "encuentre información acerca del estudiante Y". Para hacerlo, es necesario buscar de manera secuencial con el fin de encontrar el registro FACULTY para un profesor que tenga al estudiante Y en su clase. Esto da la ID, nombre, especialidad y créditos de Y, así como la calificación para dicho curso. Sin embargo, es necesario continuar la búsqueda para obtener todas las calificaciones de Y. Note que la mayor parte de la información de los estudiantes está duplicada para cada clase en la que está inscrito el estudiante. Esta elección desperdicia espacio y crea problemas de consistencia si una instancia se actualiza y otras no. La pregunta acerca del estudiante Y podría ser más fácil de responder con el uso de una estructura diferente para el árbol. En la figura C.4 se muestra un diagrama de estructura de árbol alternativa.

Ahora la base de datos contiene información de estudiantes, con los datos faculty y class subordinados a los datos student. Mientras que la pregunta del estudiante se vuelve más

FIGURA C.4

Estructura de árbol alternativa para la base de datos University



fácil de responder, la pregunta "encuentra información acerca del profesor X" es más difícil, porque es necesario buscar registros estudiantiles para encontrar un estudiante que tenga al profesor X como maestro. Esta vez, la información faculty se repite para cada estudiante que tenga un docente particular como profesor. La pregunta "encuentra información acerca de todos los estudiantes en las clases del profesor X" es mucho más difícil, porque se tiene que buscar en toda la base de datos, y leer el registro de cada estudiante para ver si el profesor X aparece como docente, y, si es así, regresar para leer la información del estudiante. Esto ilustra las negociaciones de estructuras particulares para el modelo jerárquico, porque el nodo raíz domina a todos los otros.

C.2 Arquitectura IMS

IMS usa el modelo jerárquico, pero permite una estructura que es ligeramente más flexible que una jerarquía estricta, a través del uso de **relaciones lógicas**. IMS usa un sublenguaje de datos llamado **DL/1**, lenguaje de datos uno. IMS se desarrolló antes de que los modelos de datos abstractos se definieran, de modo que no ilustra perfectamente la arquitectura de tres niveles para las bases de datos. Hablando en forma burda, a nivel conceptual, un entorno IMS puede tener muchas **bases de datos físicas**. Cada base de datos es un conjunto de registros de base de datos física, todas instancias de una sola estructura de árbol. Por ejemplo, puede tener la base de datos FACULTY, que contendría todas las instancias de los registros FACULTY, que dentro de ellos contienen datos de personal docente, clases y estudiantes. Podría haber otras bases de datos físicas que contengan información acerca de edificios, equipo, datos financieros, etc. Cada registro de base de datos física consiste en un ordenamiento jerárquico de instancias de segmento, estrictamente secuenciados, como se muestra en la figura C.2. La representación física también se puede ver en la figura C.3, o este ordenamiento se puede simular con punteros. Para cada una de las bases de datos físicas se escribe una definición completa, llamada **DBD** o descripción de base de datos. Estas DBD físicas se almacenan en forma de objeto para su uso por IMS. Las DBD físicas deben seguir todas las restricciones para árboles; en particular, ningún nodo puede tener más de un padre.

Los usuarios de IMS pronto descubren que esta restricción dificulta lidiar con segmentos que eran dependientes de más de un segmento, de modo que IMS se modificó para permitir **bases de datos lógicas**, así como físicas. Las bases de datos lógicas están en algún lugar entre el nivel conceptual y el externo para IMS. Una base de datos lógica puede ser un subconjunto de una sola base de datos física, o bien formarse al combinar segmentos de dos bases de datos físicas. La base de datos lógica no se almacena en la forma jerárquica usual como tal, sino que su estructura se crea a partir de las bases de datos físicas subyacentes mediante el uso de punteros. Los punteros mismos en realidad se almacenan con los segmentos, de modo que existe cierta base física para la estructura. Existen varias restricciones sobre los segmentos elegidos para las bases de datos lógicas. Cada base de datos lógica debe tener su propia DBD.

Al nivel externo, cada usuario tiene una visión de la base de datos conocida como el **bloque de especificación de programa** (PSB, por sus siglas en inglés). El PSB del usuario está constituido por uno o más **bloques de comunicación de programa** (PCB, por sus siglas en inglés), cada uno de los cuales define un esquema externo para un programa particular. Un PCB debe contener el nombre de la DBD, ya sea físico o lógico, sobre el cual se basa, de modo que un PCB es en realidad un subconjunto o reorganización menor de una DBD. Al nivel físico, cada base de datos física se almacena en un conjunto de datos físicos. IMS proporciona varios métodos de acceso que usan los métodos de acceso básicos del sistema operativo. La figura C.5 muestra la arquitectura IMS básica.

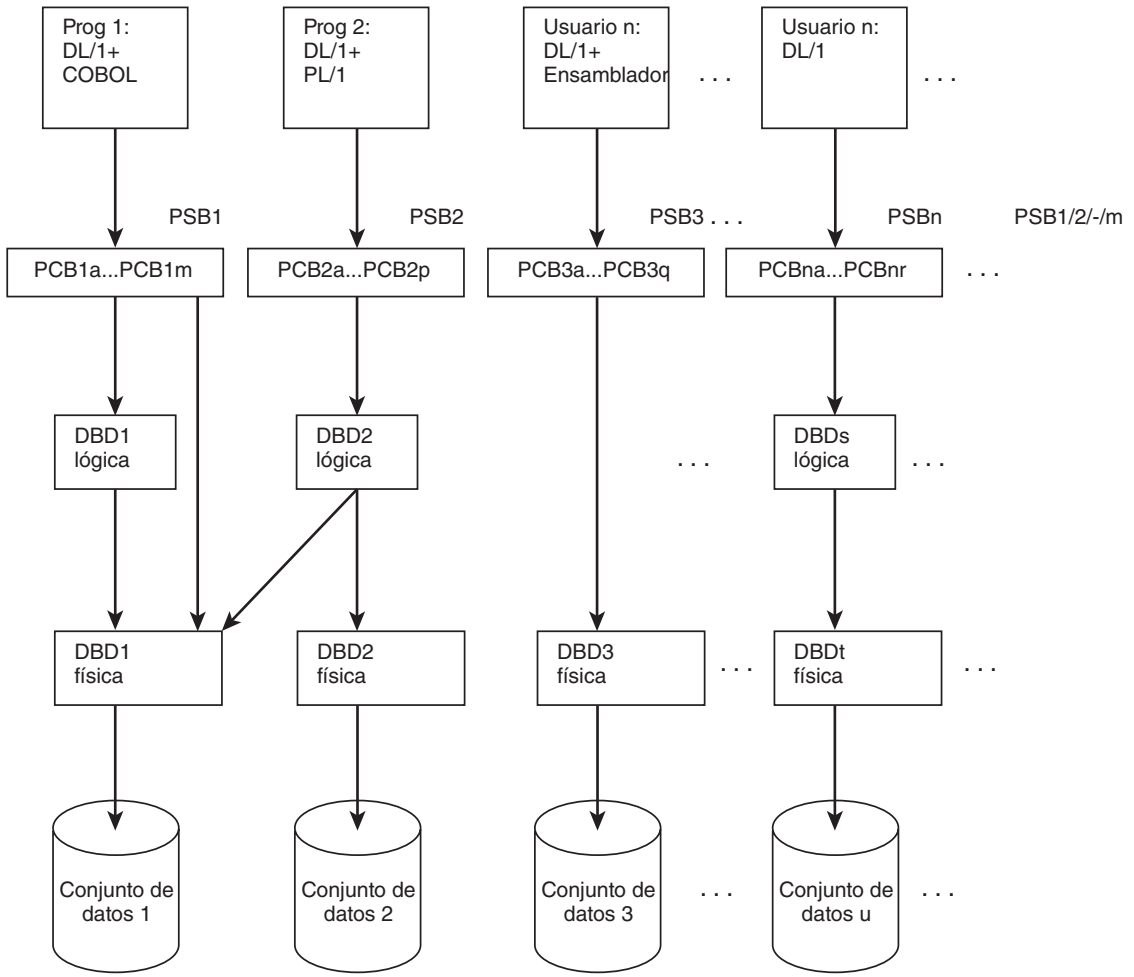


FIGURA C.5

Arquitectura IMS

C.3 Bases de datos físicas IMS

Una base de datos física IMS es un ordenamiento jerárquico de segmentos físicos, cada uno de los cuales consiste en un conjunto de campos. La estructura de los registros de la base de datos física está determinada por el diagrama de estructura de árbol, que fija los códigos tipo. Cada registro de la base de datos física comienza con un nodo raíz, o código tipo 1. El orden de los segmentos del mismo tipo está determinado por un campo llamado **campo secuencia**, que se puede considerar como una clave (posiblemente no única). Para el diagrama de estructura de árbol de la figura C.1 se puede usar FACID como el campo secuencia para FACULTY, CNO como el campo secuencia para CLASS, y STUID como el campo secuencia para STUDENT. Los registros de la base de datos física se almacenan entonces en orden por FACID, que es el campo secuencia para los nodos raíz. Dentro de un registro faculty particular, las clases están en orden por CNO. Dentro de un registro class particular, los estudiantes están en orden por STUID. Toda la base de datos está en orden por la combinación de código tipo y valor de campo secuencia. Para cualquier segmento se puede definir el **valor clave de secuencia jerárquica** como el código tipo y el valor de campo secuencia de dicho segmento, precedido por el código tipo y el valor del campo

secuencia de su padre, precedido por la misma información para su padre, etc., todo el camino de vuelta al nodo raíz. Por ejemplo, para el estudiante Burns, el valor clave de secuencia jerárquica es:

1 F110 2 MTH103C 3 S1010

La base de datos física se almacena en orden por valor clave de secuencia jerárquica, o en secuencia física como se muestra en la figura C.3, o con el uso de punteros u otras técnicas para crear dicha secuencia. Una noción relacionada con el valor clave de secuencia jerárquica es la de **clave completamente concatenada**. Para cualquier segmento, la clave completamente concatenada es el valor de campo secuencia de dicho segmento, precedido por el valor del campo secuencia de su padre, y lo mismo de vuelta a la raíz. Es similar al valor clave de secuencia jerárquica, excepto que se omiten los códigos tipo. Para el estudiante Burns, la clave completamente concatenada es:

F110 MTH103C S1010

C.4 Bases de datos lógicas IMS

Considere una base de datos que conserve información acerca de departamentos (dept), empleados (employee) y proyectos (project). Un departamento puede tener cualquier número de empleados, pero cada empleado está asignado sólo a un departamento, de modo que hay una relación padre-hijo entre departamento y empleado. De igual modo, se supone que cada proyecto puede tener muchos empleados, y cada empleado trabaja en, cuando mucho, un proyecto, de modo que entre proyecto y empleado también hay una relación padre-hijo. Entre el departamento de un empleado y un proyecto no hay relación. Cada empleado tiene muchas habilidades (skill) y cada proyecto tiene muchos gastos (expenditures). Estos datos se pueden representar con el uso de dos árboles, como se muestra en las figuras C.6(a) y (b). Note que cada registro de empleado, con sus registros de habilidad asociados, se almacena dos veces. Además de desperdiciar espacio, esta organización a la larga conduce a inconsistencia de datos, pues los datos de empleado se pueden actualizar en una base de datos mas no en otra.

IMS permite la sustitución del segmento EMP físico en un árbol mediante un puntero al correspondiente segmento EMP en el otro. Por ejemplo, el registro EMP se podría dejar en el árbol DEPT, y el registro EMP en el árbol PROJECT se podría sustituir con un puntero, como se muestra en la figura C.7. Todavía se tienen dos bases de datos físicas. La primera, la base de datos físicas DEPT-EMP-SKILL que consiste en DEPT como raíz, EMP como el hijo físico y SKILL como el nieto, se muestra en la figura C.8(a). La segunda, PROJECT-EXPEND, consiste en PROJECT como un nodo raíz y EXPEND como su hijo, se muestra en la figura C.8(b). También se tiene una base de datos lógica, con PROJ como la raíz, EMP como su hijo lógico, y SKILL como el hijo de EMP y EXPEND, como se muestra en la figura C.8(c). La base de datos lógica es posible por los punteros que se muestran en la figura C.7. En la base de datos lógica, PROJ (o PROJECT) es el padre lógico y EMP es el hijo lógico. Esto significa que EMP tiene dos padres: DEPT, que es su padre físico, y PROJ, que es su padre lógico. Los programas que usan algunos de los árboles físicos tienen PCB que se refieren a una de las dos DBD físicas, y los programas que usan el árbol lógico tienen PCB que se basan en la DBD lógica para PROJ-EMP-SKILL-EXPEND.

Existen algunas reglas estrictas en cuanto a las bases de datos lógicas, incluidas las siguientes:

- La raíz de una base de datos lógica debe ser la raíz de una base de datos física.
- Un segmento no puede ser tanto un hijo lógico como un padre lógico.

FIGURA C.6(a)

Base de datos física DEPT-EMP-SKILL

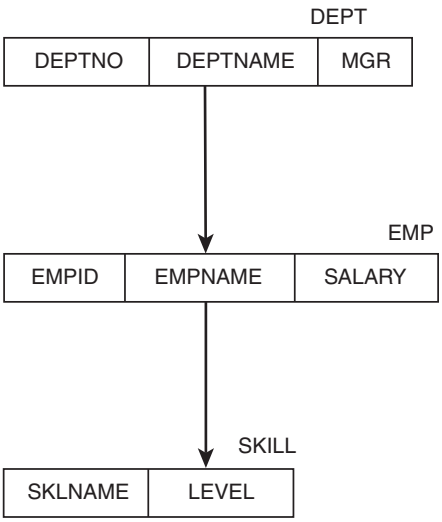


FIGURA C.6(b)

Base de datos física PROJECT-EMP-SKILL-EXPEND

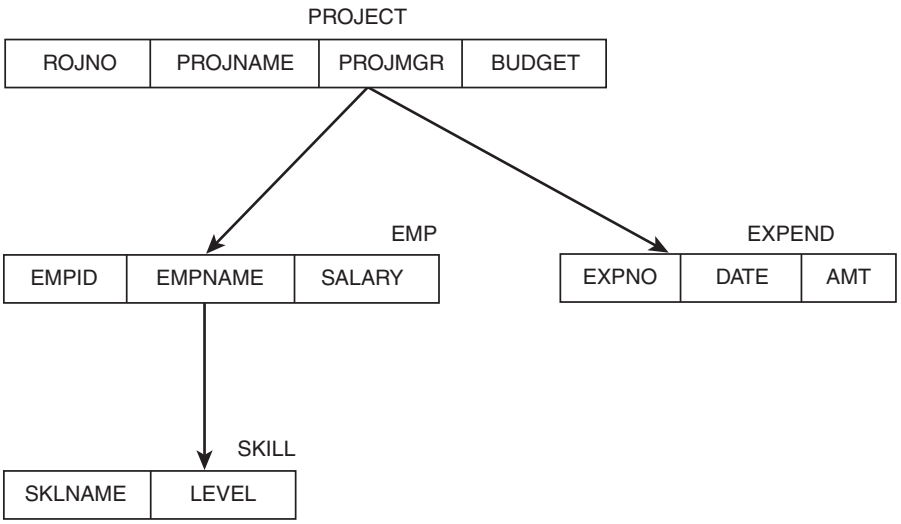
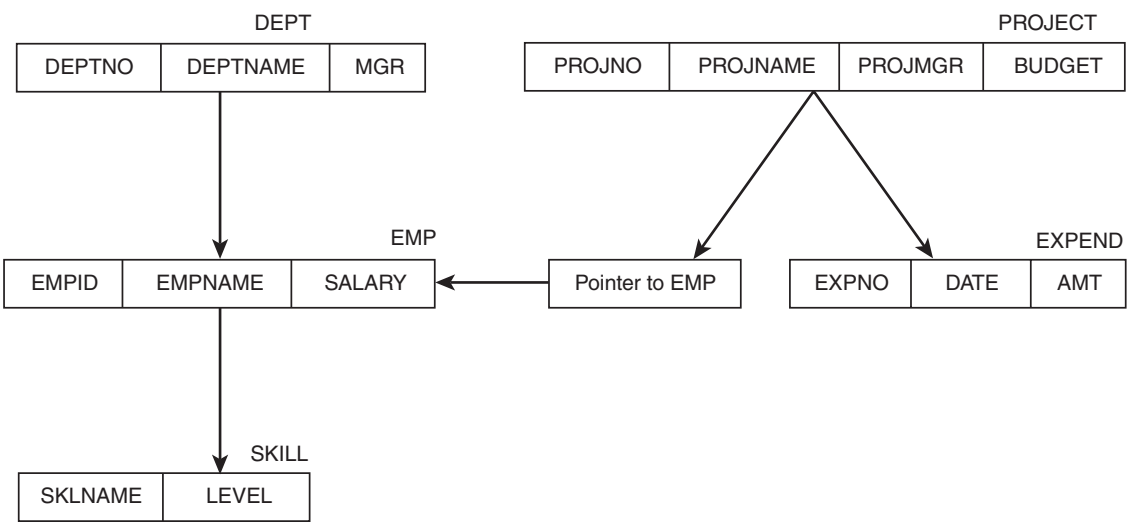


FIGURA C.7

Sustitución de un nodo mediante un puntero



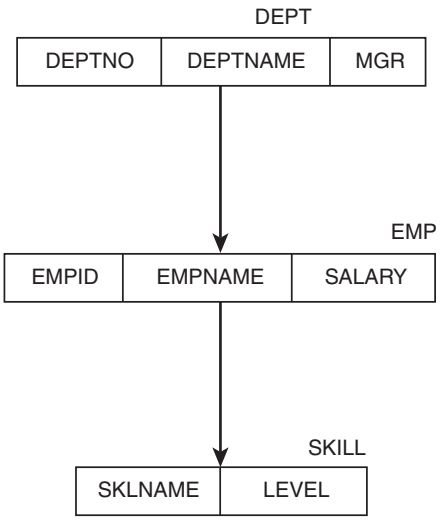


FIGURA C.8(a)

Base de datos física para DEPTDBD

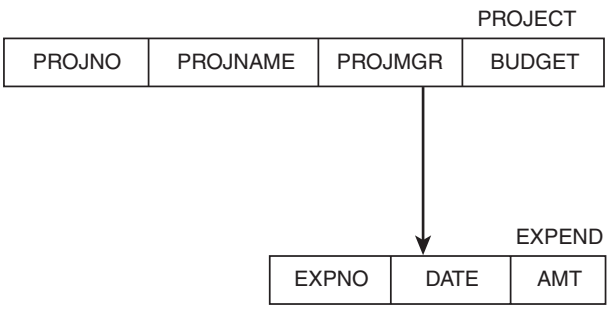


FIGURA C.8(b)

Base de datos física para PROJDBD

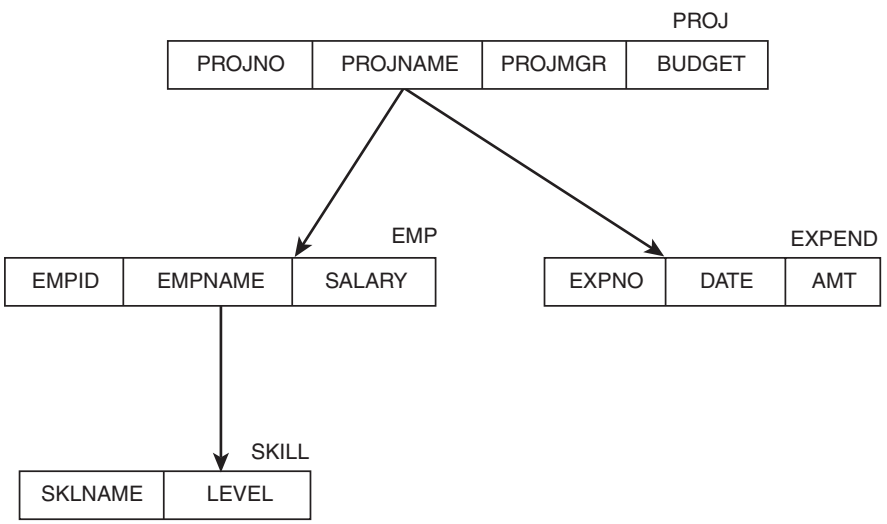


FIGURA C.8(c)

Base de datos lógica para LOGLPROJ

- Cualquier tipo de segmento en la estructura de árbol física, junto con todos sus dependientes, se puede omitir de la estructura lógica.
- Dentro de un segmento, los campos se pueden omitir o reordenar.
- En la DBD física que contiene al padre lógico, la descripción de segmento tiene que nombrar al hijo lógico y la DBD física donde se puede encontrar, antes de describir cualquiera de los campos del segmento.
- En la DBD física que contiene al hijo lógico, tienen que nombrarse los padres físico y lógico.

- El segmento de hijo lógico puede contener "datos de intersección", esto es, información que sea funcionalmente dependiente de la combinación de su padres físico y lógico. Puesto que el segmento ya contiene suficientes datos para dar la clave completamente concatenada de su padre físico, sólo la clave completamente concatenada de su padre lógico es necesaria para que los datos de intersección sean significativos.

Un tipo segmento que se incluye en una base de datos lógica se llama **segmento sensible**, uno del que está al tanto el usuario de la base de datos. Dentro de un segmento sensible, los campos que se incluyen en la base de datos lógica se llaman **campos sensibles**.

C.5 Bloques de control

La definición de base de datos IMS se implementó mediante **bloques de control**, descripciones que crean bases de datos físicas, bases de datos lógicas y bloques de comunicación de programa. Estas descripciones se escriben con el uso de los siguientes formatos, y se compilan y almacenan en forma de objeto para uso del programa de control IMS.

C.5.1 DBD físicas

Para cada base de datos física, una DBD física da una descripción completa de la estructura de árbol. La figura C.9 muestra una DBD simplificada para la base de datos FACULTY-CLASS-STUDENT. La misma DBD recibe un nombre de cuando mucho ocho caracteres. Se identifica cada segmento, junto con su padre en el árbol, y cada uno de los campos del segmento, con longitud, posición dentro del segmento y (opcional) tipo. Los tipos permitidos son C (carácter), P (decimal empacado) y X (hexadecimal), con C como el predeterminado. El orden de los segmentos dentro de la DBD está determinado por recorrido preordenado de la estructura del árbol. Cada segmento normalmente tiene un campo llamado campo secuencia, que puede tener valores únicos (U) o no únicos (M) dentro de la instancia del padre, con único como el predeterminado. Si aparece un campo secuencia, se coloca primero. El campo secuencia determina el orden de los gemelos, dos instancias del mismo segmento dentro de un padre, o dos instancias de la raíz, que se pueden considerar gemelos dentro de la base de datos. Las primeras dos líneas de la DBD también identifican

FIGURA C.9
DBD física para la estructura de árbol FACULTY-CLASS-STUDENT

```
DBD       NAME=FACDBD, ACCESS=HDAM...
DATASET   ...detalles de dispositivo físico de almacenamiento...
SEGM      NAME=FACULTY,BYTES=39,...,PARENT=0
FIELD     NAME=(FACID,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD     NAME=FACNAME,BYTES=20,START=5,TYPE=C
FIELD     NAME=DEPT,BYTES=10,START=25,TYPE=C
FIELD     NAME=RANK,BYTES=5,START=35,TYPE=C
SEGM      NAME=CLASS,BYTES=18,...,PARENT=FACULTY
FIELD     NAME=(CNO,SEQ,U),BYTES=7,START=1,TYPE=C
FIELD     NAME=SCHED,BYTES=7,START=8,TYPE=C
FIELD     NAME=ROOM,BYTES=4,START=15,TYPE=C
SEGM      NAME=STUDENT,BYTES=39,...,PARENT=CLASS
FIELD     NAME=(STUID,SEQ,U),BYTES=6,START=1,TYPE=C
FIELD     NAME=STUNAME,BYTES=20,START=7,TYPE=C
FIELD     NAME=MAJOR,BYTES=10,START=27,TYPE=C
FIELD     NAME=CREDITS,BYTES=2,START=37,TYPE=P
FIELD     NAME=GRADE,BYTES=1,START=39,TYPE=C
```

el método de acceso a utilizar (HSAM, HISAM, HDAM o HIDAM), y el conjunto de datos físicos a usar. Cada línea de segmento contiene la frecuencia (número de instancias) y tipo de punteros utilizados, pero aquí se omitirán estos detalles.

C.5.2 DBD lógicas

Una DBD lógica es similar a una DBD física, que muestra una estructura jerárquica de segmentos. Sin embargo, los segmentos se toman de una o más bases de datos físicas, y sus fuentes se deben identificar en la DBD lógica. Los nombres de segmentos pueden cambiar de los que aparecen en las fuentes. Puesto que los segmentos pueden reorganizarse de sus árboles físicos, es necesario identificar el padre (relativo a la base de datos lógica) de cada segmento. En una instancia de un árbol lógico, dos segmentos del mismo tipo que tengan el mismo padre lógico se llaman **gemelos lógicos**. Además de la DBD lógica, las DBD físicas que contienen al padre lógico y al hijo lógico deben mostrar las relaciones lógicas. Una línea LCHILD aparece inmediatamente después de la línea SEGM para el padre lógico, y los padres físico y lógico se identifican en la línea SEGM para el hijo lógico. Esto se hace para ahorrar espacio para los punteros necesarios a fin de crear la relación lógica. Usted puede distinguir una DBD lógica de una física al examinar las primeras dos líneas de la descripción. En una DBD lógica, la primera línea contiene la frase ACCESS=LOGICAL y la segunda especifica DATASET LOGICAL. En la figura C.10 se muestran la DBD para la base de datos lógica PROJ-EMP-SKILL-EXPEND, y las DBD físicas modificadas para las bases de datos físicas subyacentes. Por cuestiones de simplicidad se omiten algunos detalles.

C.5.3 PCB

Un programa de aplicación se presenta con un subconjunto de bases de datos físicas o lógicas mediante el uso del **bloque de comunicación de programa**, PCB. El PCB nombra la base de datos subyacente y cita cada segmento sensible a incluir en la vista externa de dicho programa. Si algunos campos de un segmento no se incluirán, sólo menciona los campos sensibles y sus posiciones relativas. Si no se mencionan campos, todo el segmento es sensible. Si un segmento se omite del PCB, entonces ninguno de sus dependientes se puede incluir. Las **opciones de procesamiento**, PROCOPT, especifican las operaciones que el programa puede realizar. Incluyen G (get, obtener), I (insert), R (replace, sustituir), D (delete) y L (load, cargar). Si un programa necesita que un segmento se incluya para proporcionar una ruta jerárquica hacia un descendiente, pero en realidad no se permite ver los datos en dicho segmento, al programa sólo se le concede sensibilidad de clave, lo que se indica mediante PROCOPT=K para dicho segmento. Cuando un programa usa un segmento, IMS sigue la pista de su ubicación en la base de datos al registrar su clave completamente concatenada. Para reservar espacio para el **área de retroalimentación de clave** donde se almacena la clave, la longitud de la clave completamente concatenada más larga a la que el programa puede acceder se registra en la especificación KEYLEN=xxx. Esto se calcula al considerar todas las posibles rutas jerárquicas en el PCB. La figura C.11 muestra un PCB para un programa que accede a partes de la base de datos FACULTY-CLASS-STUDENT, cuya DBD aparece en la figura C.9. En la primera línea, la frase TYPE=DB indica que este PCB es para una definición de base de datos, en lugar de una transacción en línea. El programa que usa este PCB tiene acceso a todos los segmentos FACULTY y CLASS, pero no se permite ver los campos MAJOR, CREDITS o GRADE del segmento STUDENT.

La figura C.12 muestra un PCB para la base de datos lógica PROJ-EMP-SKILL-EXPEND cuya DBD aparece en la figura C.10. En este ejemplo, al programa no se le permite leer el registro PROJ en absoluto, pero, dado que su clave es necesaria para que haya una ruta jerárquica a los segmentos EMP, SKILL y EXPEND, el PCB tiene sensibilidad de clave en

FIGURA C.10(a)
DBD DEPTDBD

```
DBD       NAME=DEPTDBD,ACCESS=HIDAM...
DATASET   ...detalles de dispositivo físico de almacenamiento...
SEGM      NAME=DEPT,BYTES=39,...,PARENT=0
FIELD     NAME=(DEPTNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD     NAME=DEPTNAME,BYTES=15,START=5,TYPE=C
FIELD     NAME=MGR,BYTES=20,START=20,TYPE=C
SEGM      NAME=EMP,BYTES=29,...,PARENT=(DEPT,(PROJECT,PROJDBD)
FIELD     NAME=(EMPID,SEQ,U),BYTES=5,START=1,TYPE=C
FIELD     NAME=EMPNAME,BYTES=20,START=6,TYPE=C
FIELD     NAME=SALARY,BYTES=4,START=26,TYPE=P
SEGM      NAME=SKILL,BYTES=11,...,PARENT=EMP
FIELD     NAME=(SKLNAME,SEQ,U),BYTES=10,START=1,TYPE=C
FIELD     NAME=LEVEL,BYTES=1,START=11,TYPE=C
```

FIGURA C.10(b)
DBD PROJDBD

```
DBD       NAME=PROJDBD,ACCESS=HDAM...
DATASET   ...detalles de dispositivo físico de almacenamiento...
SEGM      NAME=PROJECT,BYTES=43,...,PARENT=0
LCHILD    NAME=(EMP,DEPTDBD)...
FIELD     NAME=(PROJNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD     NAME=PROJNAME,BYTES=15,START=5,TYPE=C
FIELD     NAME=PROGMGR,BYTES=20,START=20,TYPE=C
FIELD     NAME=BUDGET,BYTES=4,START=40,TYPE=P
SEGM      NAME=EXPEND,BYTES=14,...,PARENT=PROJECT
FIELD     NAME=(EXPNO,SEQ,U),BYTES=5,START=1,TYPE=C
FIELD     NAME=DATE,BYTES=6,START=6,TYPE=C
FIELD     NAME=AMT,BYTES=3,START=12,TYPE=P
```

FIGURA C.10(c)
DBD LOGLPROJ

```
DBD       NAME=LOGLPROJ,ACCESS=LOGICAL
DATASET   LOGICAL
SEGM      NAME=PROJ,SOURCE=(PROJECT,PROJDBD)
SEGM      NAME=EMP,SOURCE=(EMP,DEPTDBD),PARENT=PROJ
SEGM      NAME=SKILL,SOURCE=(SKILL,DEBTDBD),PARENT=EMP
SEGM      NAME=EXPEND,SOURCE=(EXPEND,PROJDBD),PARENT=PROJ
```

FIGURA C.10
DBD físicas modificadas y DBD lógica

```
PCB       TYPE=DB,DBDNAME=FACDBD,KEYLEN=17
SENSEG    NAME=FACULTY=PROCOPT=G
SENSEG    NAME=CLASS,PARENT=FACULTY=PROCOPT=(G,I)
SENSEG    NAME=STUDENT,PARENT=CLASS,PROCOPT=(G,I,R,D)
SENFLD    NAME=STUID,START=1
SENFLD    NAME=STUNAME,START=7
```

FIGURA C.11
Un PCB para la base de datos FACULTY-CLASS-STUDENT

```
PCB       TYPE=DB,DBDNAME=LOGLPROJ,KEYLEN=19
SENSEG    NAME=PROJ,PROCOPT=K
SENSEG    NAME=EMP,PARENT=PROJ,PROCOPT=G
SENFLD    NAME=EMPID,START=1
SENFLD    NAME=EMPNAME,START=6
SENSEG    NAME=SKILL,PARENT=EMP,PROCOPT=G
SENSEG    NAME=EXPEND,PARENT=PROJ,PROCOPT=G
```

FIGURA C.12
Un PCB para la base de datos lógica PROJECT-EMP-SKILL

este segmento. El campo SALARY de empleados está oculto de este programa, pero todos los segmentos SKILL y EXPEND están disponibles.

C.6 Comandos DL/1: GU, GN, GNP, GHU, GHN, GHNP, ISRT, DLET, REPL

El lenguaje de manipulación de datos para IMS, llamado DL/1, por lo general se invoca desde un programa de lenguaje anfitrión. Se deben pasar muchos parámetros cuando se llama IMS desde este tipo de programa. La siguiente discusión de algunos comandos DL/1 presenta una versión simplificada del lenguaje. Se supone que cada programa de aplicación que accede a una base de datos IMS a través de un PCB tiene un área de trabajo de programa, en la que se almacena lo siguiente:

- **Punteros de actualización** que proporcionan la clave completamente concatenada del segmento de acceso más reciente por parte del PCB.
- **Plantillas**, o estructuras de registro para cada tipo de segmento almacenado en la base de datos. Conforme se recuperan los segmentos, dichas plantillas se llenan con los datos correspondientes.
- **Una bandera de STATUS** indica si la última operación solicitada fue exitosa. Se supondrá que un valor cero significa que la última operación tuvo éxito.

Sin importar la organización de almacenamiento real usada, el DML DL/1 se escribe como si la base de datos se almacenara en una forma parecida a cinta estrictamente secuencial, como se ilustra en la figura C.3. Por ende, parece recorrer la base de datos al comenzar con la primera instancia de árbol, según determina el campo secuencia del nodo raíz, e ir a través de dicha primera instancia en recorrido preordenado, continuar con la segunda instancia, recorrerla de la misma forma y así por el estilo hasta la última instancia del árbol en la base de datos.

La operación de recuperación es un comando GET. Tiene varias formas, como se muestra en la figura C.13. Cuando se ejecuta un GET, IMS encuentra el segmento solicitado en la base de datos, establece un puntero de actualización a su clave completamente concatenada, copia el segmento en la correspondiente plantilla de área del programa y establece el STATUS a un valor cero para indicar una operación exitosa.

DL/1 usa **argumentos de búsqueda de segmento** (SSA, por sus siglas en inglés) para identificar segmentos. Un SSA consiste en el nombre de un segmento y, posiblemente, un conjunto de paréntesis en el que se escribe una condición. La condición puede incluir comparación y operadores booleanos para expresar una restricción en el valor de cualquier campo en el segmento. Para las operaciones GU e ISRT, por lo general los SSA tienen que proporcionar una ruta jerárquica completa a partir de la raíz. Las operaciones GN y GNP no necesariamente requieren un SSA. Si se proporciona una ruta debe ser jerárquica, aunque no necesariamente comience con la raíz. Estas reglas también se aplican a las versiones GET HOLD de los mismos comandos.

FIGURA C.13
Comandos DML DL/1

DL/1	Significado	Operación
GU	Get Unique (obtener único)	Recupera el primer segmento que satisface la condición en los SSA
GN	Get Next (obtener siguiente)	Recupera el siguiente segmento cualificado, con el recorrido preordenado
GNP	Get Next within Parent (obtener siguiente dentro de padre)	Recupera el siguiente segmento cualificado, pero sólo dentro del padre actual
GHU GHN GHNP	Get Hold Unique (obtener retener único) Get Hold Next (obtener retener siguiente) Get Hold Next within Parent (obtener retener siguiente dentro de padre)	Como los anteriores, pero permite posterior DLET y REPL
ISRT	Insert (insertar)	Inserta un nuevo segmento
DLET	Delete (borrar)	Borra el segmento existente a retener
REPL	Replace (sustituir)	Sustituye el segmento existente a retener

Cuando se escribe un GET, el segmento nombrado en el último SSA es aquel que se recupera y copia en la plantilla. Los siguientes ejemplos muestran operaciones sobre la base de datos lógica PROJ-EMP-SKILL-EXPEND. Se supondrá que el PCB permite el acceso a todos los campos de todos los segmentos, y la realización de otras operaciones, del modo siguiente:

```
PCB      TYPE=DB,DBDNAME=LOGLPROJ,KEYLEN=19
SENSEG   NAME=PROJ,PROCOPT=(G,I,R,D)
SENSEG   NAME=EMP,PARENT=PROJ,PROCOPT=(G,R)
SENSEG   NAME=SKILL,PARENT=EMP,PROCOPT=(G,R)
SENSEG   NAME=EXPEND,PARENT=PROJ,PROCOPT=(G,I,R,D)
```

Aun cuando se sepa que este PCB se basa en una base de datos lógica, IMS actúa como si fuese una física; esto es: supone que hay árboles físicos, representados en forma parecida a cinta, correspondiente a esta estructura.

C.6.1 Get Unique

Este comando se usa para obtener una posición inicial en la base de datos. Debe usarse primero para que GN o GNP tengan significado. Se le puede considerar como "obtener el primer segmento que satisfaga los siguientes SSA". Cada SSA consiste en un nombre de segmento, seguido opcionalmente por la condición a satisfacer entre paréntesis. Si el segmento requerido es un nodo raíz, sólo hay un segmento nombrado, como en el siguiente ejemplo:

- *Uso de GU con SSA sencillo:* Encontrar el registro del proyecto supervisado por Smith.

  ```
  GU PROJ (PROJMGR='Smith');
  ```

  El DBMS busca a través de la base de datos y examina cada nodo raíz hasta que encuentra uno donde el PROJMGR sea Smith y recupera dicho registro. Note que se detiene en el primero de tales proyectos. Si existe un segundo proyecto supervisado por Smith, no encontrará dicho registro. Note también que no recupera todo el árbol, sólo el nodo raíz o PROJ.

- *GU con múltiples SSA y ninguna condición:* Encuentre la primera habilidad del primer empleado mencionado con dicha habilidad.

```
GU PROJ,
   EMP,
   SKILL;
```

  Puesto que no se sabe cuál proyecto se quiere, se tiene que proporcionar un SSA para PROJ sin condición. Esto hace que el IMS vaya al primer registro de proyecto. De igual modo, escriba un SSA para EMP sin condición, que lo llevará al primer segmento de empleados para dicho proyecto, siempre que exista. Si no existe empleado para el primer proyecto, IMS continúa al siguiente proyecto, etc., hasta que encuentra un segmento de empleado. Entonces tomará el primer segmento de habilidad para dicho empleado a menos que, desde luego, dicho empleado no tenga la habilidad, en cuyo caso continuará hasta que encuentre tal segmento.

- *GU con múltiples SSA y condiciones:* Encontrar el nivel de entrada de datos habilidad para el empleado E101, que está asignado al proyecto P1001.

```
GU PROJ(PROJNO='P1001'),
   EMP(EMPID='E101'),
   SKILL(SKILLNAME='data entry');
```

  Se busca hacia delante hasta que se encuentra el segmento PROJ para P1001, luego continúa dentro de dicha instancia hasta que encuentra el registro EMP para E101 y, dentro de él, encuentra el registro SKILL para la entrada de datos.

A partir de estos ejemplos se ve que GU siempre encuentra la primera instancia que satisface el predicado. Por lo general se debe especificar una ruta jerárquica completa, que comience con el nodo raíz, pero no es necesario escribir una condición junto al nombre de cada segmento a lo largo de la ruta. Si quiere especificar segmentos con exactitud escriba una condición para algún campo del segmento. Sólo se recupera el segmento inferior nombrado, no todos los segmentos a lo largo de la ruta.

C.6.2 Get Next

Una vez establecida una posición en la base de datos mediante un GU, GN recupera el siguiente segmento disponible, en relación con el actual. Recuerde que el modelo parecido a cinta se usa para determinar qué significa "siguiente". Para recuperar varios segmentos, el programador puede usar GN dentro de un bucle. El bucle se escribe en el lenguaje anfitrión.

- *Uso de GN para recuperación sencilla:* Encontrar el segundo empleado para el proyecto Jupiter.

```
GU PROJ (PROJNAME='Jupiter'),
   EMP;
GN EMP;
```

  Use un GU para encontrar al primer empleado, y a continuación un GN para encontrar el segundo. Note que sólo el segundo registro se lleva al área de trabajo.

- *GN en un bucle:* Encontrar todos los registros de proyecto, comenzando con el primer proyecto que tenga un presupuesto (budget) mayor a 10 000 dólares.

```
GU PROJ (BUDGET >10000);
while (STATUS=0)
   GN PROJ;
end;
```

Sólo se quiere recuperar segmentos PROJ, uno a la vez. Use la condición dada para escribir un GU, y luego escriba un bucle en el lenguaje anfitrión, con GN para encontrar todos los segmentos PROJ restantes.

- *GN en un bucle con procesamiento adicional:* Encuentre todos los segmentos EXPEND, imprima cada uno y sume sus cantidades.

```
TOTAMT = 0;
GU PROJ,
   EXPEND;
while (STATUS=0)
   TOTAMT = TOTAMT + AMT;
   print (EXPNO, DATE, AMT);
   GN EXPEND;
end;
```

El GU establece una posición en la base de datos en el primer segmento EXPEND. Cuando la posición se establece, el campo AMT se agrega a la variable de programa TOTAMT y los datos de segmento se imprimen. El GN en el bucle encontrará todos los segmentos EXPEND, imprimirá cada uno y sumará las cantidades.

- *GN sin SSA:* Obtener todos los segmentos en la base de datos.

```
GU PROJ;
while (STATUS=0)
   GN;
end;
```

Se sabe que el primer segmento debe ser uno de PROJ. Después de ello, se quiere recuperar cada segmento que se encuentre, sin importar su tipo, así que se usa GN sin SSA.

C.6.3 Get Next within Parent

GNP recupera el segmento mencionado en el SSA, siempre que exista, dentro del padre actual. GNP no permitirá que IMS se mueva fuera de la actual instancia padre.

- *Uso de GNP con un SSA:* Encontrar todos los gastos (expenditures) para el proyecto Jupiter.

```
GU PROJECT (PROJNAME='Jupiter');
while (STATUS=0)
   GNP EXPEND;
end;
```

Se coloca en el registro del proyecto Jupiter y se piden todos los segmentos EXPEND dentro de dicho padre. En este ejemplo, el padre se "fija" mediante un GU.

- *Uso de GNP con múltiples SSA:* Encontrar todos los empleados que tengan salarios mayores a 50 000 dólares asignados al proyecto Jupiter.

```
GU PROJ (PROJNAME='Jupiter');
while (STATUS=0)
   GNP EMP(SALARY>50000);
end;
```

- *GNP que usa un "nieto":* Encontrar todas las habilidades que pertenecen a todos los empleados asignados al proyecto Jupiter.

  El "padre" en GNP siempre es el último segmento al que accedió un GU o un GN. Una vez elegido dicho segmento, el valor del "puntero padre" se establece y cualquier

descendiente se puede recuperar mediante un GNP, sin importar el nivel del árbol involucrado.

```
GU PROJECT(PROJNAME='Jupiter');
while (STATUS=0)
   GNP SKILL;
end;
```

- *Uso de GN y GNP juntos para segmentos cruzados:* Obtener todas las habilidades de los empleados que ganen menos de 50 000 dólares.

  Una vez colocado al comienzo de la base de datos se usa un bucle con un GN para ir a cada segmento EMP disponible a la vez. Cuando se obtiene tal segmento se fija como el padre, usando un bucle anidado, y se obtiene cada uno de sus segmentos hijos SKILL.

```
GU PROJ;
while (more EMP)
   GN EMP(SALARY<50000);
   while (more SKILL)
          GNP SKILL;
   end; //para SKILL
end; // para EMP
```

C.6.4 Get Hold Unique, Get Hold Next, Get Hold Next within Parent

Estos comandos se usan en la misma forma que GU, GN y GNP, respectivamente, excepto que se deben usar en lugar de dichos comandos cuando los segmentos recuperados se vayan a actualizar o borrar.

C.6.5 REPL, DLET e ISRT

El enunciado de sustitución (REPL) se utiliza cuando un segmento se actualiza. Primero, el segmento se recupera con el uso de un comando GHU, GHN o GHNP. Una vez en el área de trabajo, el segmento se puede modificar según se requiera, usando enunciados de lenguaje anfitrión. Luego se sustituye en la base de datos mediante el comando REPL.

- *Actualización de un registro:* Para el empleado E120 en el proyecto P20, cambiar el nivel de habilidad de contabilidad a 4.

  Use un GHU para dirigirse al segmento SKILL requerido y retenerlo para procesamiento. Use el lenguaje anfitrión para actualizar el valor LEVEL. Luego sustituya el segmento.

```
GHU PROJ(PROJNO='P20'),
    EMP(EMPID='E120'),
    SKILL(SKILLNAME='Bookkeeping');
// Cambia el nivel de habilidad en el área I/O
   LEVEL = 4;
// Sustituye el segmento en la base de datos
   REPL;
```

- *Borrar un registro:* Borrar el registro del empleado E101 del proyecto P25.

  Use un GHU para ubicar el registro deseado, seguido por un comando DLET para borrarlo.

```
GHU PROJ(PROJNO='P25'),
    EMP(EMPID='E101');
DLET;
```

En realidad esto no funcionaría, porque el PCB muestra que no se tiene el procopt D para el segmento EMP. En consecuencia, la petición se rechazaría. Sin embargo, la discusión continúa como si estuviese permitido. Cuando se borra un segmento, todos sus descendientes también se borran, lo que significa que los registros SKILL para este empleado en esta instancia también se borrarían. Si los empleados pudieran asignarse a más de un proyecto, y si la intención real de este borrado es remover al empleado de la base de datos, sería importante ubicar y remover todas las instancias de su registro, de modo que se debe usar un bucle para buscar todos los registros de proyecto para ver si el empleado aparece de nuevo.

- *Inserción de un segmento:* Insertar un nuevo segmento SKILL para el empleado E220 en el proyecto P30. Al empleado se le debe dar un nivel de 2 para procesador de palabra.

  Primero cree el nuevo segmento en el área de trabajo. Luego inserte el segmento, dé una ruta jerárquica completa y use condiciones en los SSA suficientes para garantizar que el segmento se insertará en la instancia correcta. Se proporcionan los valores de los campos secuencia (únicos), de modo que se sabe que sólo hay una de tales instancias.

```
// Cree el registro skill
  SKILLNAME = 'procesador de palabras';
  LEVEL = 2;
// Ahora insértelo
  ISRT PROJECT(PROJNO='P30'),
     EMP(EMPID='E220'),
     SKILL;
```

  Desafortunadamente, puesto que no se tiene el procopt I para el segmento SKILL, esto fracasará. Sin embargo, si se tuviera dicho procopt, los comandos anteriores insertarían un nuevo segmento, siempre que el segmento padre ya exista. IMS es responsable de comprobar el campo secuencia del segmento de modo que se insertará en el orden correcto entre los segmentos SKILL hijos para el padre especificado. De nuevo, si es necesario, debe escribir un bucle para actualizar todos los registros para este empleado, sin importar el PROJECT.

C.7 Códigos de comando D, F y V

Una de las principales dificultades en la manipulación de una base de datos IMS es que el sistema está diseñado para buscar hacia delante, ir desde una posición presente para recorrer un árbol con el uso de preorden. Es difícil regresar en un árbol. El uso de códigos de comando permite "regresar". Esto es posible porque IMS puede mantener más de un indicador de actualización. Ya se sabe que el último segmento con acceso se considera el segmento actual, pero, además, cada uno de sus ancestros es el segmento actual de su tipo. Con el ejemplo PROJ-EMP-SKILL-EXPEND, si sólo tiene que recuperar un segmento SKILL, entonces el segmento EMP que es su padre es el actual de EMP, y el PROJ que es su abuelo es el actual de PROJ. Esto es cierto incluso si no se especificaron valores para ellos en el SSA. Los códigos de comando sacan ventaja de estos indicadores de actualización. Los códigos de comando se pueden insertar después del nombre del segmento en un SSA al agregar un asterisco y la letra o letras adecuadas. Los códigos de comando D, F y V se ilustran, con el ejemplo PROJ-EMP-SKILL-EXPEND, pero note que los procopts de PCB necesitarían modificarse para que algunos de ellos funcionen de manera adecuada.

- *Uso del código de comando D:* Encontrar el primer proyecto al que está asignado el empleado E133.

El código de comando D se usa para recuperar ancestros de un nodo. Para cualquier segmento en la ruta jerárquica para la que se escribe *D, los datos de dichos segmentos se concatenarán y colocarán en el área I/O, junto con los datos para el segmento en el fondo de la ruta.

```
GU PROJ*D,
   EMP(EMPID='E133');
```

Resultado: Ahora los segmentos PROJ y EMP están en el área de trabajo. Note que, si tiene una ruta jerárquica larga, podría usar *D para aquellos segmentos que quiera recuperar, y no para otros.

- *Uso del código de comando F:* Encontrar el primer empleado asignado al proyecto para el cual se realizó el número de gasto X500.

  Este comando se usa para "regresar" dentro de un registro padre actual para encontrar la primera instancia del tipo segmento especificado. Se puede usar un GU para encontrar el segmento EXPEND deseado y sería capaz de encontrar el padre asociado con el uso de un código de comando D. Sin embargo, no se busca al padre, sino a un hermano. Por desgracia, ya se recorrió el segmento EMP para llegar al segmento EXPEND. Por tanto, es necesario "regresar" bajo el padre actual.

```
GU  PROJ,
    EXPEND(EXPNO='X500'),
GNP EMP*F;
```

  El GU lo coloca en el registro PROJ correcto. El *F hace que el IMS regrese al principio del registro padre actual (es decir, el PROJ actual). Una vez ahí, el GNP EMP obtiene el primer segmento EMP.

- *Uso del código de comando V para sustituir GNP:* Obtener todas las habilidades del empleado E144 del proyecto Jupiter.

  El código de comando V mantiene el IMS dentro del segmento actual del tipo indicado al intentar satisfacer una petición. Por lo general puede usarse en lugar de GNP, pero en realidad es una técnica más general que GNP.

```
GU  PROJ(PROJNAME='Jupiter'),
    EMP(EMPID='E144');
while (more skill segments)
    GN EMP*V,
       STUDENT;
end;
```

- *Ejemplo más general de código de comando V:* Obtener el primer empleado para el proyecto con número de gasto X500.

  Aunque este problema se hizo en un ejemplo anterior, ahora se abordará con el uso del código de comando V. Se usará un GU para encontrar el segmento EXPEND, y luego un código de comando V para encontrar y permanecer dentro del actual del tipo segmento PROJ, mientras se regresa usando un código de comando F para encontrar el primer segmento EMP dentro de dicho PROJ.

```
GU EXPEND(EXPNO='X500');
GN PROJ*V,
   EMP*F;
```

  Esto es más simple que el método anterior, porque tiene posibilidad de usar un GU para pedir el primer segmento EXPEND que satisfaga la condición sin especificar el PROJ. En ejemplos anteriores se dio una ruta jerárquica completa a un segmento con

la finalidad de establecer indicadores de actualidad para sus ancestros. Ahora se puede usar GU para pedir cualquier tipo de segmento directamente y luego usar el código de comando V para establecer un padre actual. Note que GN PROJ*V EMP*F significa "permanecer dentro del registro PROJ actual, regresar para encontrar el primer segmento EMP".

En general, el uso de códigos de comando da por resultado acceso más eficiente que el uso de GNP.

C.8 Mapeo de modelo EE-R a modelo jerárquico

Puesto que muchas bases de datos heredadas son jerárquicas, con frecuencia surge el problema de convertirlas en bases de datos relacionales o posrelacionales. La ruta de migración puede requerir convertir de un diagrama de árbol a un diagrama EE-R con un paso intermedio. La creación de un diagrama de árbol a partir de una DBD es trivial, porque la DBD describe la estructura de árbol con gran detalle. El plan básico de convertir un diagrama de árbol a un diagrama EE-R es convertir segmentos en entidades y convertir relaciones padre-hijo, incluidas las lógicas, en relaciones EE-R. Sin embargo, debido a las severas limitaciones del modelo jerárquico, en la conversión se puede perder cierta información semántica. El diseñador debe examinar cuidadosamente los resultados del mapeo directo para determinar si se pueden hacer refinamientos al diagrama EE-R que resulte. El mapeo se puede realizar del modo siguiente:

- Cada segmento se puede representar inicialmente como una entidad con los campos como atributos. Si el campo secuencia es único se convierte en la clave.
- Cada relación padre-hijo inicialmente se puede representar como una relación 1 a muchos entre la entidad padre y la entidad hijo.
- Las relaciones se deben examinar para ver si realmente son relaciones muchos a muchos, que no se pueden representar en el modelo jerárquico, excepto por repetición.
- Algunos segmentos hijo en un modelo jerárquico pueden representar atributos estructurados o atributos repetitivos del segmento padre. En este caso, el segmento hijo no es una entidad separada y se debe combinar con la entidad padre.
- Algunos segmentos hijo pueden representar entidades débiles y se deben representar como tales en el diagrama EE-R.
- Un segmento puede tener cuando mucho dos padres, de modo que el diagrama se debe examinar para ver si una entidad tiene más relaciones de las que el modelo jerárquico puede capturar. Pueden necesitarse relaciones binarias, ternarias o de orden superior adicionales.

Para mapear un modelo E-R a un modelo jerárquico, el diagrama EE-R tiene que convertirse a diagramas de estructura de árbol. El método general para hacerlo es cambiar cada entidad a un tipo segmento, cada atributo de una entidad en un campo de un segmento, y cada relación en una relación padre-hijo. Entonces los resultados se combinan en una o más estructuras de árbol. No hay un "mejor método" único para hacer el mapeo de un diagrama E-R a un diagrama de árbol.

Bibliografía

Abiteboul, S., R. Hull y V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

Abiteboul, S. y P. Kanellakis. "Object Identity as a Query Language Primitive", SIGMOD Conference, 1989: 159-173.

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices. *Software Engineering Code of Ethics and Professional Practice.* ACM,1999.

Agrawal, R., T. Imielinski y A. Swami. "Mining Association Rules between Sets of Items in Large Databases", SIGMOD Conference, 1993: 207-216.

Agrawal, R. y R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases", VLDB Conference, 1994: 487-499.

Aho, A., C. Beeri y J. Ullman. "The Theory of Joins in Relational Databases", ACM Transactions on Database Systems 4(3): 297-314 (1979).

Aho, A. y J. Ullman. "The Universality of Data Retrieval Languages", POPL Conference, 1979: 110-120.

Albano, A., L. Cardelli y R. Orsini. "Galileo: A Strongly-Typed, Interactive Conceptual Language", ACM Transactions on Database Systems 10(2): 230-260 (1985).

Association for Computing Machinery. *ACM Code of Ethics and Professional Conduct.* ACM, 1992.

Armstrong, W. "Dependency Structures of Data Base Relationships", IFIP Congress, 1974: 580-583.

Atzeni, P. y V. De Antonellis. *Relational Database Theory.* Benjamin/Cummings, 1993.

Bancilhon, F. y R. Ramakrishnan. "An Amateur's Introduction to Recursive Query Processing Strategies", SIGMOD Conference, 1986: 16-52.

Bancilhon, F., D. Maier, Y. Sagiv y J. Ullman. "Magic Sets and Other Strange Ways to Implement Logic Programs", Symposium on Principles of Data Systems, 1986: 1-16.

Banerjee, J., H. Chou, J. Garza, W. Kim, D. Woelk, N. Ballou y H. Kim. "Data Model Issues for Object-Oriented Applications", ACM Transactions on Information Systems 5(1): 3-26 (1987).

Banerjee, J., W. Kim, H. Kim y H. Korth. "Semantics y Implementation of Schema Evolution in Object-Oriented Databases", SIGMOD Conference, 1987: 311-322.

Batini, C., S. Ceri y S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach.* Benjamin/Cummings, 1992.

Batini, C., M. Lenzerini y S. Navathe. "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computing Surveys 18(4): 323-364 (1986).

Bayer, R. y E. McCreight. "Organization and Maintenance of Large Ordered Indices", Acta Informatica 1(3): 173-189 (1972).

Beeri, C. y R. Ramakrishnan. "On the Power of Magic", Symposium on Principles of Data Systems, 1987: 269-283.

Berners-Lee, T., R. Caillian, A. Lautonen, H. Nielsen y A. Secret. "The World Wide Web", Communications of the ACM 13(2): Agosto, 1994.

Bernstein, P. "Synthesizing Third Normal Form Relations from Functional Dependencies", ACM Transactions on Database Systems 1(4): 277-298 (1976).

Bernstein, P. y N. Goodman. "Concurrency Control in Distributed Database Systems", ACM Computing Surveys 13(2): 185-221 (1981).

Bernstein, P., V. Hadzilacos y N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

Bischoff, J. y T. Alexander (eds.). *Data Warehouse: Practical Advice from the Experts*. Prentice-Hall, 1997.

Blackstone, W. *Commentaries on the Laws of England.* Clarendon Press, 1765-1769.

Carey, M., D. DeWitt y S. Vandenberg. "A Data Model and Query Language for EXODUS", SIGMOD Conference, 1988: 413-423.

Cattell, R., *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1993.

Ceri, S. y G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill Book Company, 1984.

Ceri, S., G. Gottlob y L. Tanca. *Logic Programming and Databases.* Springer-Verlag, 1990.

Chen, P. "The Entity-Relationship Model—Toward a Unified View of Data", ACM Transactions on Database Systems 1(1): 9-36 (1976).

CODASYL. *CODASYL Data Base Task Group April 71 Report,* ACM, 1971.

Codd, E. "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM 13(6): 377-387 (1970).

Codd, E. "Further Normalization of the Data Base Relational Model", IBM Research Report, San Jose, California RJ909 (1971).

Codd, E. "Relational Completeness of Data Base Sublanguages", IBM Research Report, San Jose, California RJ987 (1972).

Codd, E. "Extending the Database Relational Model to Capture More Meaning", ACM Transactions on Database Systems 4(4): 397-434 (1979).

Codd, E. "Is Your DBMS Really Relational?" *Computerworld.* 14 de octubre, 1985.

Codd, E. "Does Your DBMS Run By the Rules?" *Computerworld.* 21 de octubre, 1985.

Comer, D. "The Ubiquitous B-Tree", ACM Computing Surveys 11(2): 121-137 (1979).

Das, S. *Deductive Databases and Logic Programming.* Addison-Wesley, 1992.

Date, C. *An Introduction to Database Systems* (8a. ed.). Addison-Wesley, 2004.

DeWitt, D., S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao y R. Rasmussen. "The Gamma Database Machine Project", IEEE Transactions on Knowledge and Data Engineering 2(1): 44-62 (1990).

DeWitt, D., R. Katz, F. Olken, L. Shapiro, M. Stonebraker y D. Wood. "Implementation Techniques for Main Memory Database Systems", SIGMOD Conference, 1984: 1-8.

Elmasri, R. y S. Navathe. *Fundamentals of Database Systems* (4a. ed.). Addison-Wesley, 2004.

Eswaran, K., J. Gray, R. Lorie y I. Traiger. "The Notions of Consistency and Predicate Locks in a Database System", Communications of the ACM 19(11): 624-633 (1976).

Fagin, R. "Multivalued Dependencies and a New Normal Form for Relational Databases", ACM Transactions on Database Systems 2(3): 262-278 (1977).

Fagin, R., J. Nievergelt, N. Pippenger y H. Strong. "Extendible Hashing—A Fast Access Method for Dynamic Files", ACM Transactions on Database Systems 4(3): 315-344 (1979).

Fishman, D., D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbæk, B. Mahbod, M. Neimat, T. Ryan y M. Shan. "Iris: An Object-Oriented Database Management System", ACM Transactions on Information Systems 5(1): 48-69 (1987).

Gallaire, H., J. Minker y J. Nicolas. "Logic and Databases: A Deductive Approach", ACM Computing Surveys 16(2): 153-185 (1984).

Garcia-Molina, H., J. Ullman y J. Widom. *Database Systems: The Complete Book.* Prentice Hall, 2002.

Gifford, D. "Weighted Voting for Replicated Data", SOSP Proceedings, 1979: 150-162.

Goldberg, A. y D. Robson. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, 1983.

Graefe, G. "Query Evaluation Techniques for Large Databases", ACM Computing Surveys 25(2): 73-170 (1993).

Graefe, G. y D. DeWitt. "The EXODUS Optimizer Generator", SIGMOD Conference, 1987: 160-172.

Gray, J. "Notes on Data Base Operating Systems", en Bayer, R., M. Graham y G. Seegmuller. (eds.). *Operating Systems: An Advanced Course.* Springer-Verlag, 1978: 393-481.

Gray, J., A. Bosworth, A. Layman y H. Pirahesh. "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab y Sub-Total", International Conference on Database Engineering, 1996: 152-159.

Gray, J., P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, G. Putzolu y I. Traiger. "The Recovery Manager of the System R Database Manager", ACM Computing Surveys 13(2): 223-243 (1981).

Gray, J. y A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

Guttman, A. "R-Trees: A Dynamic Index Structure for Spatial Searching", SIGMOD Conference, 1984: 47-57.

Halligan, R. y & R. Weyand. "The Sorry State of Trade Secret Protection", *Corporate Counsellor.* Agosto 2001.

Hammer, M. y D. McLeod. "Database Description with SDM: A Semantic Database Model", ACM Transactions on Database Systems 6(3): 351-386 (1981).

Haskin, R. y R. Lorie. "On Extending the Functions of a Relational Database System", SIGMOD Conference, 1982: 207-212.

Henschen, L. y S. Naqvi. "On compiling queries in recursive first-order databases", Journal of the ACM 31(1): 47-85 (1984).

House of Commons Committee on Privacy and Related Matters. *Report of the Committee on Privacy and Related Matters, Chairman David Calcutt QC,* (XLVII) Cm. 1102, HMSO, 1990.

Hull, R. y R. King. "Semantic Database Modeling: Survey, Applications y Research Issues", ACM Computing Surveys 19(3): 201-260 (1987).

Inmon, W. *Building the Data Warehouse* (3a. ed.) John Wiley & Sons, 2002.

International Standards Organization. *Ergonomic requirements for office work with visual display terminals (VDTs)—Part 11: Guidance on usability* ISO9241-1:1997.

Jarke, M. y J. Koch. "Query Optimization in Database Systems", ACM Computing Surveys 16(2): 111-152 (1984).

Johnson, D. *Computer Ethics* (3a. ed.). Prentice Hall, 2001.

Kent, W. *Data and Reality,* North-Holland, 1978.

Kim, W. "On Optimizing an SQL-like Nested Query", ACM Transactions on Database Systems 7(3): 443-469 (1982).

Kizza, J. *Ethical and Social Issues in the Information Age* (2a. ed.). Springer, 2003.

Klug, A. "Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions", Journal of the ACM 29(3): 699-717 (1982).

Knuth, D. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973.

Lamport, L. "Time, Clocks y the Ordering of Events in a Distributed System", Communications of the ACM 21(7): 558-565 (1978).

Litwin, W. "Linear Hashing: A New Tool for File and Table Addressing", VLDB Conference, 1980: 212-223.

Loney, K. y G. Koch. *Oracle9i: The Complete Reference.* Oracle Press, 2002.

Loomis, M. *Object Databases: The Essentials.* Addison-Wesley, 1995.

Maier, D. *The Theory of Relational Databases.* Computer Science Press, 1983.

Maier, D., J. Stein, A. Otis y A. Purdy. "Development of an Object-Oriented DBMS", OOPSLA Conference, 1986: 472-482.

McFarland, Michael. "Intellectual Property, Information and the Common Good", Ethics and Technology Conference 1999: 88-95.

Melton, J. y A. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann, 1993.

Mohan, C., D. Haderle, B. Lindsay, H. Pirahesh y P. Schwarz. "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging", ACM Transactions on Database Systems 17(1): 94-162 (1992).

Moor, J. "Towards a Theory of Privacy for the Information Age", Computers and Society (27)3: 27-32(1997).

Mylopoulos, J., P. Bernstein y H. Wong. "A Language Facility for Designing Database-Intensive Applications", ACM Transactions on Database Systems 5(2): 185-207 (1980).

Nielsen, J. *Designing Web Usability: The Practice of Simplicity.* New Riders Publishing, 2000.

Nievergelt, J., H. Hinterberger, K. Sevcik. "The Grid File: An Adaptable, Symmetric Multikey File Structure", ACM Transactions on Database Systems 9(1): 38-71 (1984).

O'Neil, P. y E. O'Neil. *Database Principles, Programming, Performance (*2a. ed.). Morgan Kaufmann, 2003.

Organisation for Economic Co-operation and Development. *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data.* OECD, 1981.

Ozsu, M. T. y Valduriez, P. *Principles of Distributed Database Systems* (2a. ed.). Prentice-Hall, 1999.

Papadimitriou, C. "The serializability of concurrent database updates", Journal of the ACM 26(4): 631-653 (1979).

Ramakrishnan, R. y J. Gehrke. *Database Management Systems* (3a. ed.). McGraw-Hill, 2003.

Ricardo, C. *Database Systems: Principles, Design and Implementation.* Macmillan, 1990.

Salzberg, B. *File Structures, An Analytic Approach.* Prentice-Hall, 1988.

Samet, H. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

Schek, H. y M. Scholl. "The relational model with relation-valued attributes", Information Systems 11(2): 137-147 (1986).

Selinger, P., M. Astrahan, D. Chamberlin, R. Lorie y T. Price. "Access Path Selection in a Relational Database Management System", SIGMOD Conference, 1979: 23-34.

Shapiro, L. "Join Processing in Database Systems with Large Main Memories", ACM Transactions on Database Systems 11(3): 239-264 (1986).

Shasha, D. *Database Tuning—A Principled Approach.* Prentice-Hall, 1992.

Sheth, A. y J. Larson. "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys 22(3): 183-236 (1990).

Shipman, D. "The Functional Data Model and the Data Language DAPLEX", ACM Transactions on Database Systems 6(1): 140-173 (1981).

Silberschatz, A., H. Korth y S. Sudarshan. *Database System Concepts* (4a. ed.). McGraw-Hill, 2002.

Simon, A. *Strategic Database Technology: Management for the Year 2000.* Morgan Kaufmann, 1995.

Smith, J. y D. Smith. "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems 2(2): 105-133 (1977).

Snodgrass, R. "The Temporal Query Language TQuel", ACM Transactions on Database Systems 12(2): 247-298 (1987).

Spinello, R. *Cyber Ethics: Morality and Law in Cyberspace.* Jones and Bartlett, 2000.

Stonebraker, M. "Implementation of Integrity Constraints and Views by Query Modification", SIGMOD Conference, 1975: 65-78.

Stonebraker, M., Ed. *Readings in Database Systems* (2a. ed.). Morgan Kaufmann, 1994.

Stonebraker, M. y L. Rowe. "The Design of Postgres", SIGMOD Conference, 1986: 340-355.

Stonebraker, M., E. Wong, P. Kreps y G. Held. "The Design and Implementation of INGRES", ACM Transactions on Database Systems 1(3): 189-222 (1976).

Teorey, T., D. Yang y J. Fry. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", ACM Computing Surveys 18(2): 197-222 (1986).

Thomas, R. "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases", ACM Transactions on Database Systems 4(2): 180-209 (1979).

Ullman, J. "Implementation of Logical Query Languages for Databases", ACM Transactions on Database Systems 10(3): 289-321 (1985).

Ullman, J. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

Ullman, J. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989.

United States Congress. *Digital Millennium Copyright Act.* Public Law 105-304 (1998).

United States Copyright Office, Library of Congress. *Copyright Basics (Circular 1).* 2004.

United States Department of Health, Education and Welfare, Secretary's Advisory Committee on Automated Personal Data Systems. *Records, computers, and the Rights of Citizens.* 1973.

United States Patent and Trademark Office. *General Information Concerning Patents.* 2003.

Valduriez, P. "Join Indices", ACM Transactions on Database Systems 12(2): 218-246 (1987).

Warren S. y L. Brandeis. "The right to privacy", *Harvard Law Review* 4:193-220 (1890).

Westin, A. *Privacy and Freedom.* Atheneum, 1967.

Wiederhold, G. *Database Design* (2a. ed.). McGraw Hill, 1983.

Wong, E. y K. Youssefi. "Decomposition—A Strategy for Query Processing", ACM Transactions on Database Systems 1(3): 223-241 (1976).

World Intellectual Property Organization. *WIPO Intellectual Property Handbook: Policy, Law and Use.* WIPO, 2001.

World Trade Organization. *A Summary of the Final Act of the Uruguay Round.* 1994.

World Trade Organization, *Uruguay Round Agreement: TRIPS Part II—Standards concerning the availability, scope and use of Intellectual Property Rights.* 1994

Zaniolo, C. S. Ceri, C. Faloutsos, R. Snodgrass, V. Subrahmanian y R. Zicari. *Advanced Database Systems.* Morgan Kaufmann, 1997.

Índice analítico

A

B

C

D

E

N

O

P

R

S

T

U

V

W

X