PROCESAMIENTO DE BASES DE DATOS

OCTAVA EDICIÓN

FUNDAMENTOS, DISEÑO E IMPLEMENTACIÓN

DAVID M. KROENKE

PEARSON Prentice Hall ®

PROCESAMIENTO DE BASES DE DATOS

PROCESAMIENTO DE BASES DE DATOS

Fundamentos, diseño e implementación

Octava edición

David M. Kroenke

TRADUCCIÓN
Dra. Ana Elizabeth García Hernández
Escuela Superior de Ingeniería Química e Industrias Extractivas,
Instituto Politécnico Nacional

REVISIÓN TÉCNICA
M. en C. Juan Raúl Esparza Martínez
Departamento de Computación Básica,
Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey

PEARSON Educación®

MÉXICO • ARGENTINA • BRASIL • COLOMBIA • COSTA RICA • CHILE
ESPAÑA • GUATEMALA • PERÚ • PUERTO RICO • VENEZUELA

Datos de catalogación bibliográfica

KROENKE, DAVID M.

Procesamiento de Bases de Datos
Fundamentos, diseño e implementación

PEARSON EDUCACIÓN, México, 2003

ISBN: 970-26-0325-0
Área: Universitario

Formato: 21 × 27 cm Páginas: 688

Edición en español
Editor: Guillermo Trujano Mendoza
e-mail: guillermo.trujano@pearsoned.com
Supervisor de desarrollo: Lorena Pontones
Supervisor de producción: Enrique Trejo Hernández

Edición en inglés
Publisher: Natalie E. Anderson
Executive Editor, MIS: Bob Horan
Associate Editor: Kyle Hannon

OCTAVA EDICIÓN, 2003

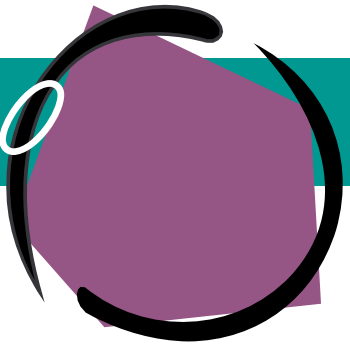
ISBN 970-26-0325-0

Impreso en México. *Printed in Mexico.*
1 2 3 4 5 6 7 8 9 0 - 05 04 03 02

RESUMEN DE CONTENIDO

CONTENIDO

PREFACIO



De acuerdo con Alan Greenspan, Presidente de la Reserva Federal de los Estados Unidos de América, la información tecnológica ha permitido aumentos sin precedente en la productividad de los negocios. En tanto que Internet tiene la mayoría de los créditos, detrás del escenario de la base de datos la tecnología desempeña un papel clave. Después de todo, Internet es sólo un sistema de comunicación; su valor primordial radica en los datos y en la información que transmite hacia y desde las bases de datos.

Las novedades del punto com pueden ocasionar que los estudiantes se pregunten si el valor de estas tecnologías declinará con el tiempo. Nada puede estar más lejos de la verdad. Lou Gestner, presidente de IBM, estableció hace varios años los verdaderos beneficios de Internet y describió las tecnologías que existirían, sólo después de que éstas hubieran sido aceptadas por la corriente principal: la corporativa de Estados Unidos —por las así llamadas compañías de la "vieja economía"—. En la aplicación actual de esta tecnología de bases de datos subyacen importantes oportunidades para cada tipo de negocios y actividades empresariales (así como para los futuros profesionales de bases de datos).

Lo anterior significa que nunca ha existido un momento mejor para estudiar el procesamiento de bases de datos. De las bases de datos de los escritorios personales, a las grandes bases de datos interorganizacionales distribuidas en las computadoras de la red mundial, las bases de datos están aumentando en forma importante los activos de las empresas. La tecnología de las bases de datos se utiliza en mercadotecnia, ventas, producción, operaciones, finanzas, cuentas bancarias, administración y, de hecho, en todas las disciplinas empresariales, con el fin de aumentar la productividad en sus respectivas actividades.

Además, después del frenesí de años recientes con respecto a las nuevas tecnologías y sus productos, los elementos principales de la administración moderna de las bases de datos ahora se han vuelto claras. Continúa siendo esencial el conocimiento conceptual de la modelación y diseño de bases de datos; así mismo, el modelo relacional y el SQL son tan importantes como en el pasado. La administración de bases de datos, especialmente la tecnología que apoya a los administradores de bases de datos multiusuarios, ha adquirido mayor importancia debido a que todas las que usan la nueva tecnología son multiusuario.

Además, la tecnología para las bases de datos publicadas en la Web, especialmente las arquitecturas tritier y multitier, XML, páginas de servidor activo (ASP) y páginas del servidor Java (JSP), han surgido como vencedoras entre muchos contendientes para la publicación de la base de datos. De acuerdo con estas tecnologías, tanto ODBC, con OLE DB, como JDBC continúan siendo importantes.

En resumen, la tecnología de bases de datos es más importante que nunca y es necesario enseñarla de una manera mucho más clara que durante los cinco años anteriores.

Características de la presente edición

De acuerdo con lo que ya comentamos, la segunda parte de este texto ha sido reescrita por completo. Casi todo el contenido de los capítulos 11 al 16 es nuevo. Las principales tareas de la administración de bases de datos se estudian en el capítulo 11, en el 12 se ilustran con Oracle, y en el capítulo 13 de nuevo con el servidor SQL. En el capítulo 14 se estudian las tecnologías básicas para la publicación de bases de datos en la Web, y dichas tecnologías se ilustran para ODBC, OLE DB, IIS, y ASP en el capítulo 15, y de

nueva cuenta con JDBC, JSP, y MySQL en el capítulo 16. El capítulo 17 incluye información sobre OLAP, mientras que el 18 introduce construcciones con el nuevo objeto relacional de Oracle.

Dirigir todos estos temas a una sola meta es un reto y creo que debemos considerar muy seriamente que es necesario dedicarse todo un año al estudio de las bases de datos. Mientras tanto, si quiere estudiar sólo un tema y no tiene tiempo, esta edición ha sido escrita para que usted elija entre tres series de tecnologías alternativas.

Específicamente, considerando el modelado de datos, el texto se refiere al modelo entidad-relación y al modelo de objeto semántico. Si el tiempo es breve, quizás usted quiera abarcar sólo el modelo E-R, porque es el más popular. De manera similar, considerando las bases de datos de usuarios múltiples, elija Oracle en el capítulo 12 o el Servidor SQL en el capítulo 13, dependiendo de las necesidades de los profesionales en su comunidad. Finalmente, considere la publicidad en la Web y si el tiempo es breve, elija IIS, ASP y ODBC en el capítulo 15; o Java, JDBC y JSP en el 16. No se perderá la continuidad si selecciona sólo uno de estos. Por supuesto, si no tiene restricciones de tiempo, todos los temas son importantes.

Concepto	Alternativa 1	Alternativa 2
Modelación de datos	Modelo entidad-relación capítulos 3 y 6	Modelo de objeto semántico capítulos 4 y 7
DBMS[1] multiusuario	Oracle capítulo 13	SQL Server capítulo 14
Publicidad en la Web	IIS, ASP, ODBC capítulo 15	Java, JDBC, JSP capítulo 16

Esta edición incluye también nuevas series de ejercicios al final del capítulo, los cuales se aplican a una compañía pequeña que produce, comercializa y promociona una línea de estufas de campamento. El objetivo de estos ejercicios es capacitar a los estudiantes para que apliquen los conocimientos que obtengan de cada capítulo en una pequeña, real, aunque limitada aplicación.

Panorama capítulo por capítulo

Este texto consta de siete partes. La primera introduce al procesamiento de bases de datos. El capítulo 1 ilustra aplicaciones simples, define términos básicos y bosqueja la historia del procesamiento de bases de datos. El capítulo 2 ilustra el desarrollo de una base de datos simple y una aplicación usando Microsoft Access XP.

La segunda parte versa sobre el modelado de datos. En el capítulo 3 se analiza el modelo entidad-relación y muestra cómo se ha integrado a UML, o al lenguaje de modelación unificado. El capítulo 4 presenta el modelo de objeto semántico, una alternativa de modelación de datos para el modelo E-R. El tema de la tercera parte es el diseño de la base de datos. En el capítulo 5 se analizan el modelo relacional y la normalización. El capítulo 6 aplica las ideas de los capítulos 3 y 5 para transformar los modelos entidad-relación en diseños de bases de datos relacionales. En el capítulo 7 se trabajan las ideas de los capítulos 4 y 5 para transformar los modelos de los objetos semánticos en diseños de bases de datos relacionales.

La cuarta parte aborda los fundamentos de la implementación de la base de datos relacional. El capítulo 8 presenta un panorama, el 9 estudia un proceso SQL y el 10 describe el diseño de las aplicaciones de la base de datos relacional. En la quinta parte se

[1] Database management system, DBMS por sus siglas en inglés.

aborda la administración de una base de datos multiusuarios. En el capítulo 11 se describe la administración de la base de datos y se analizan importantes aspectos de un proceso de base de datos multiusuarios, incluyendo control de concurrencia, seguridad, respaldo y recuperación. Las ideas que se presentan en el capítulo 11 se ilustran con Oracle en el capítulo 12. En el capítulo 12 también se ilustra SQL para la definición de datos. De igual manera, el 13 refleja el análisis del capítulo 11 para ilustrar la administración de bases de datos multiusuarios usando SQL Server.

La publicación de bases de datos en la Web se analiza en la sexta parte. En el capítulo 14 se establecen los fundamentos del procesamiento de la red, las arquitecturas multicapa y XML. En el capítulo 15 se aplican estos conceptos utilizando la tecnología Microsoft, incluyendo ODBC, OLE DB, IIS y ASP. En el capítulo 16 se aplican los conceptos del capítulo 14 utilizando Java; esto incluye JDBC, JSP y MySQL. Los conceptos se ilustran con ejemplos utilizando Linux y Apache Tomcat. Después, en el capítulo 17 se abordan los usos de la administración de datos y se analiza OLAP.

La séptima parte sólo contiene un capítulo en el cual se analiza el procesamiento de bases de datos orientadas a objetos. Lo nuevo en este capítulo es un análisis del objeto relacional de Oracle, sus características y funciones. El apéndice A contiene un breve comentario con respecto a las estructuras de datos, y el apéndice B ilustra el uso del diseñador de tablas, un producto que se puede utilizar para desarrollar los modelos de objeto semántico y convertirlos en diseños de bases de datos y páginas ASP.

Reconocimientos

El cambio drástico en la segunda mitad de esta edición se debe a conversaciones muy constructivas y útiles con mi editor, Bob Horan, así como a las revisiones excepcionalmente provechosas e ilustrativas de las siguientes personas:

Jack Becker, Universidad del Norte de Texas
William D. Burg, Universidad de Alabama en Birmingham
Bhushan Kapoor, Universidad Estatal de California-Fullerton
Donald. R. Moscato, Colegio Iona
Nancy L. Russo, Universidad del Norte de Illinois
Behrooz Saghafi, Universidad Estatal de Chicago
Joseph L. Sessum, Universidad Estatal Kennesaw
Ashraf Shirani, Universidad estatal de San José
Ludwig Slusky, Universidad estatal de California-Los Ángeles

Además, a Marty Murray del Colegio de la Comunidad de Portland quien me alentó y ayudó a desarrollar el material de Oracle en el capítulo 12. Estoy en deuda con Warner Schyerer de la Corporación SoundDev, en Seattle, por la idea de incluir JSP y MySQL en el capítulo 16. También gracias a Jude Stoller, de SoundDev, quien revisó el material. Agradezco especialmente a Chris Wilkens de Safaridog.com, quien me condujo a través del proceso de instalación de Linux, Apache y Tomcat.

Gracias a Microsoft por comercializar la versión para estudiantes de su producto. También gracias a Thorsten Ganz, presidente de Coolstrategy.com, quien hizo disponible el diseñador de tablas para estudiantes utilizando este texto y a Kenji Kawai, el cual continúa desempeñando un papel fundamental en el desarrollo de los productos basados en el modelo de objeto semántico. Gracias especiales a Kyle Hannon de Prentice-Hall quien dirigió el texto y los componentes de apoyo y se aseguró que funcionaran bien; también agradezco a Vanessa Nuttry, de Prentice-Hall, la cual hábilmente dirigió el proceso de producción. Por último, expreso mi agradecimiento a Lynda por su amor, apoyo y aliento.

El procesamiento de bases de datos es un tema fascinante que se ha desarrollado y ha evolucionado durante más de treinta años, y creo que continuará cuando menos en los siguientes años. ¡A usted lector, le deseo un gran éxito en su exploración de muchas oportunidades futuras!

David Kroenke
Seattle, Washington

PARTE I

INTRODUCCIÓN

Los dos capítulos que conforman la parte I introducen el tema del procesamiento de la base de datos. El capítulo 1 describe cuatro aplicaciones fundamentales de ésta y analiza sus ventajas con respecto al antiguo procesamiento de sistemas de archivos. También se define el término *base de datos* y se estudia la historia de su procesamiento. En el capítulo 2 se describen los elementos de una base de datos y se analizan las funciones de un sistema de administración de base de datos (DBMS, por sus siglas en inglés). Esta introducción concluye en el capítulo 2 con el resumen de las tareas que son necesarias para desarrollar una base de datos y sus aplicaciones.

La parte I proporciona un panorama con respecto a las necesidades de las bases de datos, la naturaleza de sus componentes y sus aplicaciones. El propósito es establecer el fundamento para estudiar los detalles relativos a los conceptos y la tecnología de la base de datos en los capítulos siguientes.

CAPÍTULO 1

Introducción al procesamiento de bases de datos

El procesamiento de bases de datos ha sido siempre un tema importante en el estudio de los sistemas de información. Sin embargo, en los últimos años la expansión de Internet y el drástico desarrollo de la nueva tecnología para Internet ha hecho del conocimiento de la tecnología de bases de datos una de las carreras más apasionantes. La tecnología de bases de datos permite que las aplicaciones de Internet den un paso más allá del simple folleto publicitario que caracterizó a las primeras aplicaciones. Al mismo tiempo, proporciona un medio estandarizado y de rápido acceso para dar a conocer a los usuarios el contenido de bases de datos. Ninguno de estos nuevos desarrollos ignora la necesidad de bases de datos de aplicaciones básicas que fueron vitales para los intereses de los negocios antes de que surgiera Internet. Simplemente aumentan la importancia que tiene el conocimiento de bases de datos.

Para muchos estudiantes este tema es agradable e interesante, aunque lo consideren un gran reto. El diseño y desarrollo de bases de datos implica arte e ingeniería. La comprensión de los requerimientos del usuario y su traducción en diseños eficaces de bases de datos es un proceso artístico. Transformar esos diseños en bases de datos físicas con plena funcionalidad, y hacer aplicaciones de alto rendimiento es un proceso de ingeniería. Ambos aspectos están llenos de retos y son acertijos intelectuales divertidos.

Debido a la inmensa necesidad tecnológica de bases de datos, a las habilidades que usted desarrollará y al conocimiento que obtendrá en este curso, le auguramos que tendrá muchas oportunidades de trabajo. El objetivo de este texto es proporcionarle fundamentos sólidos sobre la tecnología fundamental de bases de datos para que pueda iniciar una afortunada carrera en este campo, si así lo decide.

➤ CUATRO EJEMPLOS DE BASES DE DATOS

El propósito de una base de datos es ayudar a las personas a dar seguimiento a las cosas. Las aplicaciones clásicas de bases de datos se refieren al seguimiento de artículos tales como órdenes, clientes, empleos, empleados, llamadas telefónicas, u otros aspectos de interés para una persona de negocios. Recientemente se ha aplicado la tecnología de

bases de datos en Internet, no sólo en el caso de aplicaciones clásicas, sino también en el de nuevas aplicaciones como anuncios dirigidos a las características del cliente y al seguimiento de las observaciones de éste y de sus hábitos de compra en las páginas Web. Estas bases de datos incluyen datos fotográficos, de audio y de vídeo, así como información tradicional como nombres, antecedentes y números telefónicos. Los cuatro ejemplos que veremos a continuación ilustran el uso de la tecnología de las bases de datos a través de este amplio rango de aplicaciones.

Pintora de casas Mary Richards

Mary Richards es una pintora profesional de casas que es la propietaria y operadora de una pequeña compañía, integrada por ella, otro pintor profesional y, cuando es necesario, pintores de medio tiempo. Mary ha estado en el negocio durante diez años y se ha ganado la reputación de ser profesional y ofrecer precios razonables. Ella consigue la mayoría de los trabajos con clientes que la contratan en varias ocasiones y también a través de las referencias y recomendaciones que dan sus clientes a otras personas; además consigue algo de trabajo con los constructores y diseñadores profesionales de interiores.

Los clientes recuerdan mejor a Mary de lo que ella los recuerda a ellos. De hecho, algunas veces se avergüenza cuando un cliente le llama y le dice algo como: "Hola Mary, habla John Maples. Pintaste mi casa hace tres años." Ella sabe que debería recordar a todo aquel que le llama por el trabajo que realizó, pero pinta más de 50 casas al año y es difícil que recuerde algunos clientes en particular. Esta situación empeora cuando ellos le dicen: "A mi vecino le gustó el trabajo que realizaste en nuestra casa y le gustaría algo similar para la suya."

Con el fin de ayudar a su memoria y dar un mejor seguimiento a sus registros, Mary tenía un especialista que desarrolló una base de datos y la aplicación que utiliza en su computadora personal. La base de datos almacena registros de los clientes, trabajos y fuentes de referencias en forma de tablas, como se muestra en el ejemplo de la figura 1-1.

Éste es el trabajo de un programa llamado sistema administrador de base de datos (DBMS) que almacena y recupera datos de estas tablas. Por desgracia, cuando estos da-

➤ FIGURA 1-1

Tablas de datos para la pintora de casas Mary Richards

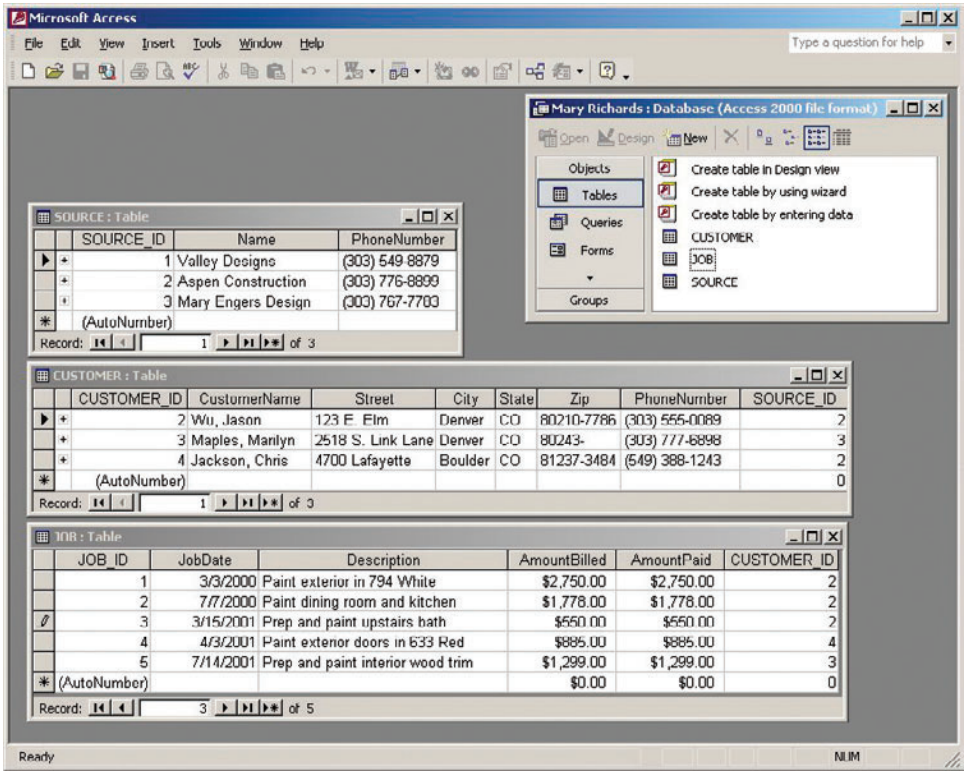
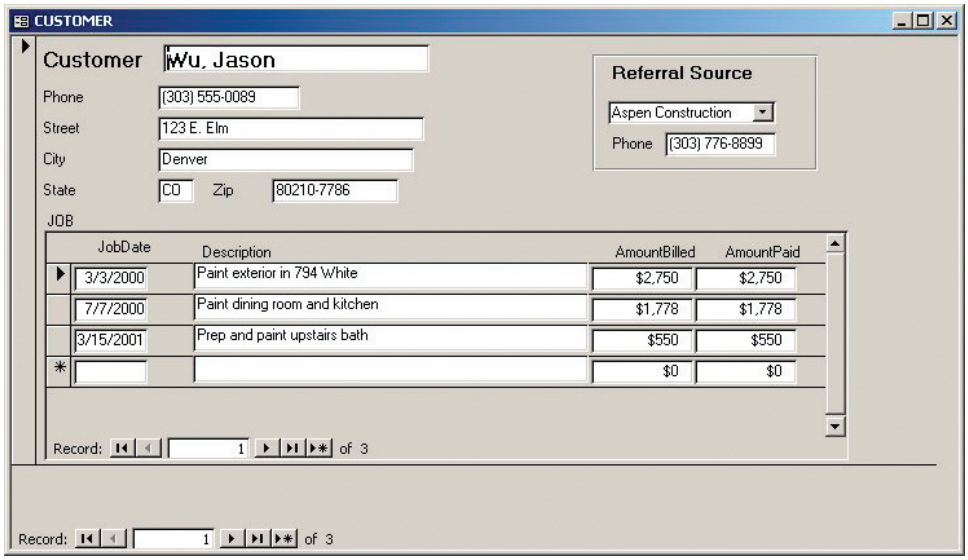


FIGURA 1-2

Muestra de la forma de entrada de datos de la pintora de casas Mary Richards

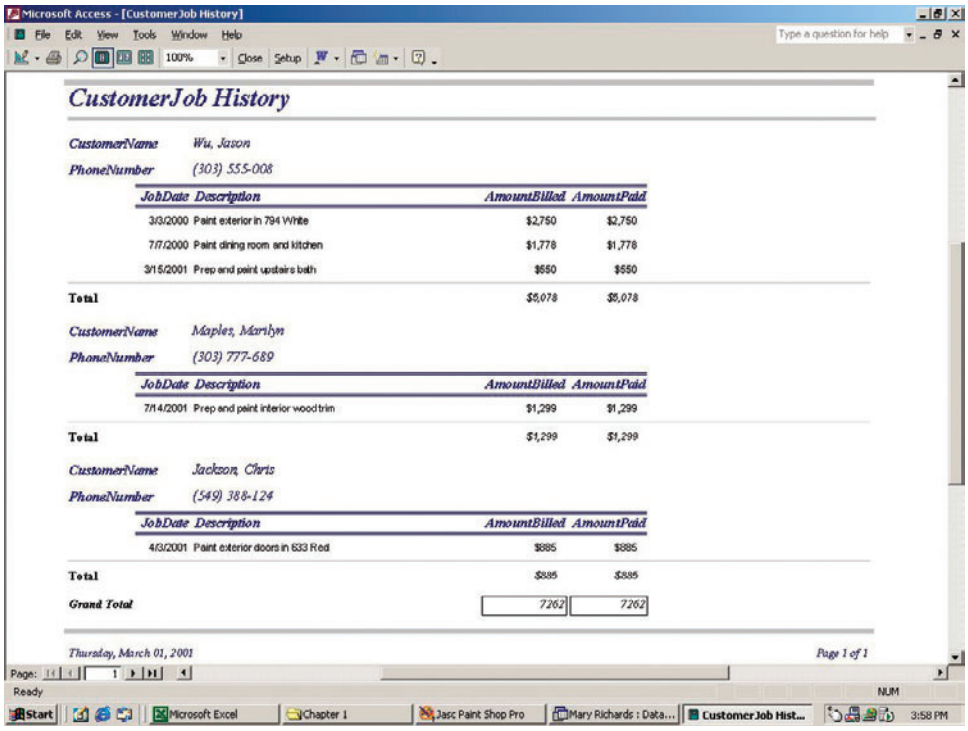


tos están en forma de tablas no son muy útiles para Mary, porque lo que ella necesita es saber cómo se relacionan entre sí los clientes, trabajos y referencias; por ejemplo, qué trabajos se han hecho para determinado cliente, o quién ha llegado gracias a una recomendación de alguien en particular.

Para lograr esto, el asesor de Mary creó una aplicación de base de datos que procesa formas de entrada de datos y produce reportes. Considere como ejemplo la forma de la figura 1-2. Aquí, Mary teclea los datos de sus clientes, tales como nombre, teléfono y dirección. También los relaciona con una fuente de referencia en particular y teclea los datos de los trabajos que realizó para ellos. Dichos datos se pueden mostrar entonces en forma de reportes, como el que se muestra en la figura 1-3. Otros usos de la base de datos incluyen el registro de las cotizaciones, el seguimiento de las fuentes de referencia y la producción de etiquetas para enviar por correo la publicidad de ventas directas.

FIGURA 1-3

Muestra de un reporte para la pintora de casas Mary Richards



La aplicación de la base de datos y el DBMS procesan la forma y almacenan los datos que se introducen en tablas como las de la figura 1-1. De igual manera, la aplicación y el DBMS extraen los datos de dichas tablas para crear un reporte similar al de la figura 1-3.

Considere nuevamente los datos de la figura 1-1 y observe que los renglones en las tablas son referencias cruzadas y están vinculados entre sí. Cada JOB (TRABAJO) contiene CUSTOMER_ID (identificación del cliente) (CLIENTE_ID), el CLIENTE que pagó por ese TRABAJO y cada CLIENTE lleva SOURCE_ID (la identificación de la fuente) (FUENTE_ID), o la persona que lo recomendó. Estas referencias se usan para combinar datos y producir formas y reportes como los que se muestran en las figuras 1-2 y 1-3.

Como se podrá imaginar, es poco probable que Mary sepa cómo diseñar las tablas de la figura 1-1, cómo utilizar un DBMS para crear esas tablas, y cómo desarrollar la aplicación para elaborar formas y reportes. Cuando termine este curso, usted sabrá cómo usar la tecnología de bases de datos para crear una, así como sus aplicaciones. También será capaz de diseñar y manipular tablas para crear formatos y reportes de mayor complejidad.

Casa de música Treble Clef

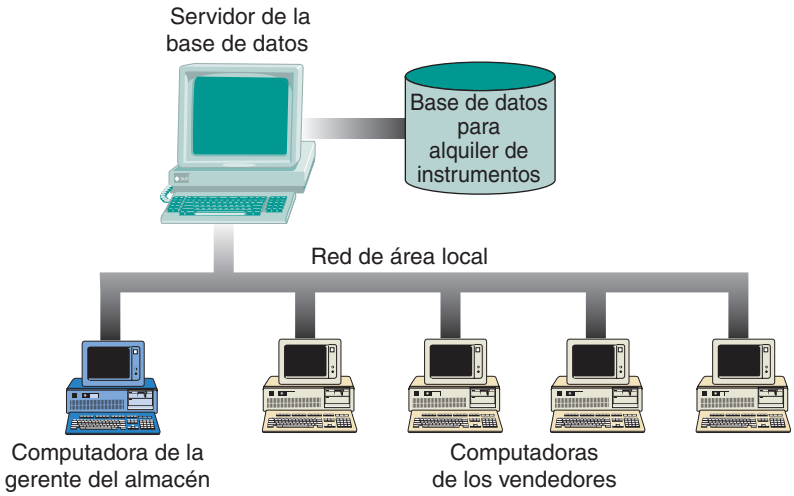
La base de datos de Mary Richards se llama base de datos de un *solo usuario*, porque sólo uno tiene acceso a ésta en un momento determinado. En algunos casos, esta limitación es demasiado restrictiva; mucha gente necesita tener acceso a la base de datos simultáneamente desde múltiples computadoras. Estas bases de datos *multiusuarios* son más complicadas porque el DBMS y la aplicación deben sostener el trabajo del usuario sin la interferencia de los otros usuarios.

La casa de música Treble Clef usa una aplicación de base de datos para mantener el seguimiento de los instrumentos musicales que renta. Necesita una aplicación multiusuarios porque, durante los periodos de trabajo, varios vendedores pueden rentar instrumentos musicales al mismo tiempo. También, la gerente del almacén necesita tener acceso a la base de datos de renta para determinar cuándo ordenar más instrumentos de determinado tipo, sin que se interrumpa el proceso de alquiler al hacer esto.

El almacén de la casa Treble Clef tiene una red local que conecta a varias computadoras personales con un servidor, el cual mantiene la base de datos de alquiler, como se muestra en la figura 1-4. Cada empleado tiene acceso a la aplicación de la base de datos mediante las tres formas ilustradas en la figura 1-5. La Customer form (forma que el cliente llena) se usa para mantener los datos de éste, la Rental Agreement form (forma

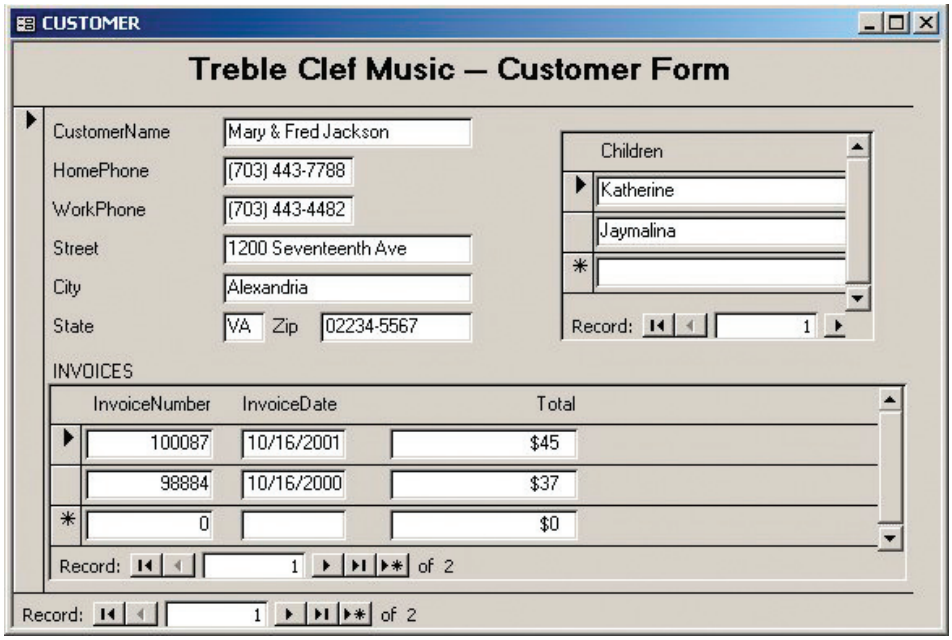
➤ FIGURA 1-4

Red de área local con el servidor de la base de datos que usa la casa de música Treble Clef

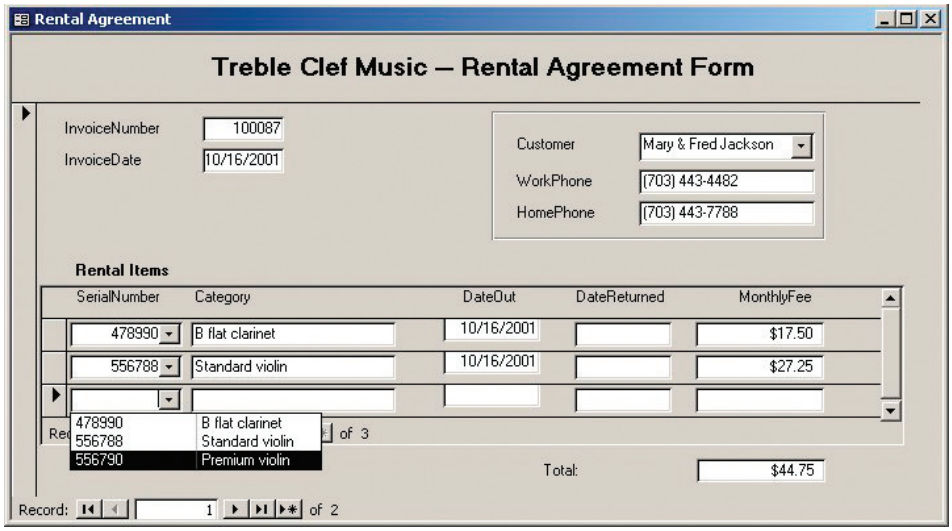


➤ FIGURA 1-5

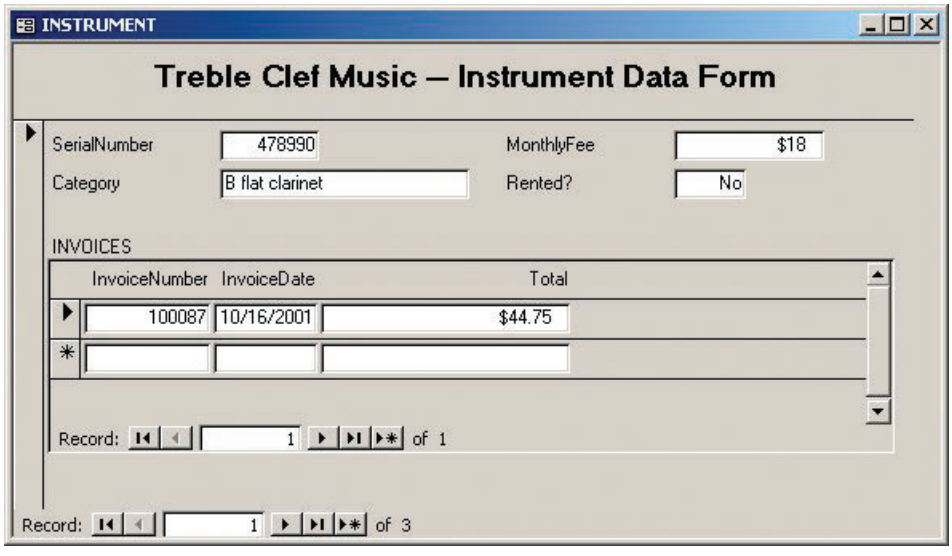
Tres formas que usa la casa de música Treble Clef: (a) Forma para el cliente, (b) Forma del convenio de renta y (c) Forma del instrumento



(a)



(b)



(c)

del convenio de renta) se utiliza para darle seguimiento a los instrumentos que han sido rentados, si ya fueron regresados o no, y la Instrument Data form (forma de datos del instrumento) que se utiliza para mostrar sus características y los antecedentes del alquiler.

Para comprender los problemas que se deben enfrentar en una base de datos multiusuarios, considere lo que sucedería cuando dos clientes intentaran rentar el clarinete plano-B al mismo tiempo. El DBMS y los programas de aplicación deben detectar que esta situación está ocurriendo e informar a los empleados que tienen que escoger otro instrumento.

Oficina estatal de licencias y registro de vehículos

Ahora consideremos una aplicación aún mayor de la tecnología de base de datos: una oficina estatal de licencias y de registro de vehículos. Tiene 52 centros de pruebas de manejo, expedición de licencias para conductores, renovación de licencias de manejo, y también 37 oficinas que expiden registros de vehículos.

El personal tiene acceso a una base de datos para realizar su trabajo. Antes de que a las personas se les otorgue o renueve su licencia de conducir, hay que verificar sus registros en la base de datos para buscar posibles infracciones de tránsito, accidentes o arrestos. Estos datos se utilizan para determinar si la licencia debe o no ser renovada, o si se debe otorgar con ciertas limitaciones. De igual manera, el personal del departamento de registro de automóviles tiene acceso a la base de datos para determinar si un auto ha sido registrado antes y, si es así, quién lo registró, o si existe algún asunto importante que impida expedir el registro.

Esta base de datos tiene cientos de usuarios, incluyendo no sólo al personal de las licencias y registros, sino al del departamento estatal de contribuciones y del departamento jurídico. No es de extrañar que la base de datos sea grande y compleja, con más de 40 diferentes tablas de datos, muchas de las cuales contienen cientos de miles de renglones.

Las bases de datos de las grandes organizaciones, como la oficina de licencias y registros, fueron las primeras aplicaciones de este tipo de tecnología. Estos sistemas han existido durante 20 o 30 años y se han modificado para satisfacer los cambios que ocurrieron durante ese periodo. Otros ejemplos de bases de datos organizacionales se relacionan con el procesamiento de cuentas en bancos e instituciones financieras, sistemas de producción y de suministro de material en fábricas grandes, procesamiento de registros médicos en hospitales, y en compañías de seguros y agencias gubernamentales.

Actualmente muchas organizaciones están adaptando sus aplicaciones de bases de datos para permitir a los clientes tener acceso, e incluso cambiar sus datos, por medio de Internet. Si usted llegara a trabajar en una gran organización importante, probablemente le podrían asignar ese proyecto.

Centro de reservaciones de la isla Calvert

La isla Calvert es muy poco conocida; es una isla hermosa en la costa oeste de Canadá. Para promover el turismo en un mercado internacional, la Cámara de Comercio de la isla Calvert ha desarrollado un sitio Web que tiene tres finalidades:

- Promover la belleza y las oportunidades recreativas que brinda la isla
- Obtener y almacenar datos de nombres y direcciones de los visitantes del sitio Web para darles seguimiento con el envío de promociones por correo
- Obtener y almacenar solicitudes de reservación para hoteles, cabañas y servicios turísticos y comunicárselo a los vendedores

Se utilizan dos bases de datos para apoyar a este sitio Web. La primera es una base de datos promocional que almacena datos, fotos, vídeoclips y algunos pequeños soni-

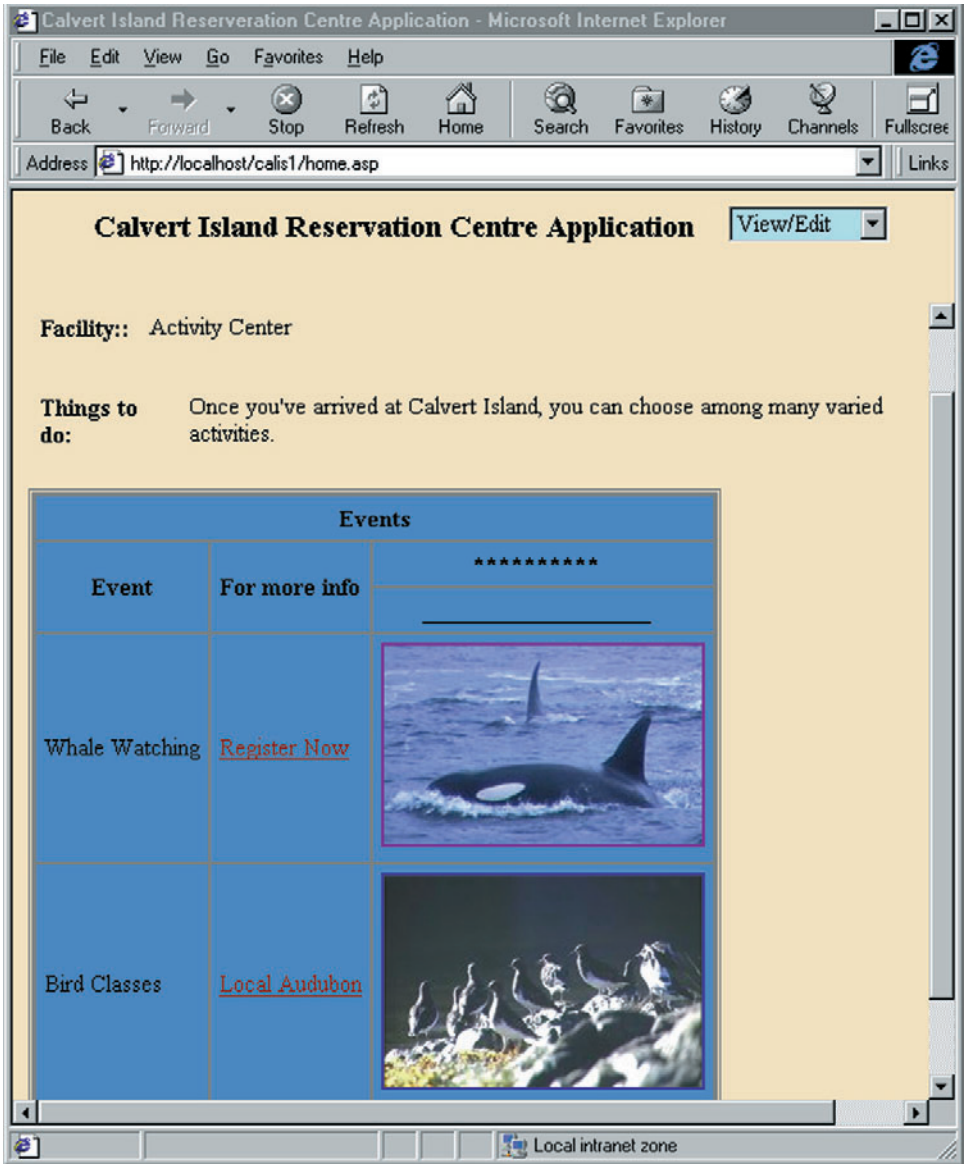
dos del lugar, actividades e instalaciones en la isla Calvert. Esta base de datos tiene dos tipos de usuarios: el normal, que ingresa sólo con fines de lectura y utiliza exploradores estándar. Estos usuarios pueden posicionarse y hacer "clic" alrededor del sitio Web para ver las actividades e instalaciones que les pueden interesar. Detrás de escena, una aplicación de base de datos está extrayendo datos y elementos multimedia de la base de datos promocional (véase la figura 1-6).

El segundo tipo de usuario de la base de datos promocional es un empleado de la Cámara de Comercio que se encarga del sitio. El empleado agrega, cambia, borra datos y archivos multimedia en la base de datos, como por ejemplo cambios de promociones. Los vendedores entran y salen del programa y dan respuesta a las preguntas de los usuarios.

Además de la base de datos promocional, las aplicaciones en el sitio Web procesan una base de clientes y reservaciones, la cual almacena los datos que ingresan cuando los visitantes del sitio Web llenan un formato de encuesta y solicitan una reservación. Los datos acerca del nombre del cliente, la dirección postal y de correo electrónico, intereses, preferencias y las solicitudes de reservación se conservan en la base de datos.

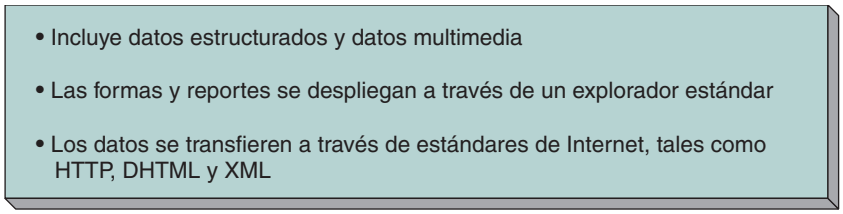
➤ FIGURA 1-6

Página Web del Centro de reservaciones de la isla Calvert



➤ FIGURA 1-7

Características de las aplicaciones de tecnología de Internet



Cuando ingresa una solicitud, la aplicación la envía a un vendedor a través del correo electrónico. Periódicamente se prepara un resumen de los reportes de reservación y se le envía por correo electrónico a los vendedores para que den seguimiento a los posibles clientes, pero también sirve para otros propósitos de administración.

Tres características principales de la base de datos de la isla Calvert la distinguen de las aplicaciones anteriores. En primer lugar, una gran parte de la primera base de datos contiene no sólo datos estructurados, tales como los nombres de los vendedores, sino también corrientes de bits no estructuradas de archivos multimedia. Segundo, el contenido de la aplicación se reparte al usuario a través de un explorador estándar. Las formas que se usan para Treble Clef y la oficina de licencias tienen un formato específico que creó el diseñador y sólo cambia cuando se modifica la aplicación. En contraste, los usuarios de la isla Calvert ven la forma en un formato que se determina no sólo mediante las aplicaciones, sino también por la etiqueta, la versión y las opciones locales que usan sus exploradores.

La tercera característica diferente de la aplicación de la isla Calvert es que se usó la tecnología estandarizada orientada a la Web para transferir datos entre el explorador, la aplicación y la base de datos. Se utilizan **protocolo de transferencia de hipertexto (HTTP,** por sus siglas en inglés**), lenguaje dinámico para marcar hipertexto (DHTML,** por sus siglas en inglés**),** y **el lenguaje de marcado extendible (XML,** por sus siglas en inglés**).** Usando estos medios estándar cualquier usuario que tenga un explorador puede acceder a esta aplicación. El software no debe estar instalado previamente en sus computadoras. Consecuentemente, el uso de esta aplicación es virtualmente ilimitado. Analizaremos el papel de HTTP, DHTML y XML para las aplicaciones de bases de datos en los capítulos 14 al 16. (Véase la figura 1-7.)

Comparación de aplicaciones de bases de datos

Estos ejemplos representan una muestra con respecto a los usos de la tecnología de bases de datos. Cientos de miles de bases de datos son como la que usa la pintora de casas Mary Richards: de un solo usuario con una cantidad relativamente pequeña de datos, digamos de tan sólo 10 megabytes. Las formas y reportes por lo general son simples y sencillas.

Hay otras bases, como la que usa la casa de música Treble Clef, que tienen más de un usuario, pero casi siempre un poco menos de 20 o 30 usuarios juntos. Contienen una cantidad moderada de datos, digamos 50 o 100 MB. Las formas y reportes son necesariamente bastante más complicados para apoyar diferentes funciones de negocios.

Las bases de datos más grandes, como las del registro de autos, tienen miles de usuarios y trillones de bytes de datos. Están en uso muchas aplicaciones diferentes, y cada una tiene sus propias formas y reportes. Por último, algunas bases de datos implican el uso de la tecnología de Internet y de procesos, de caracteres y datos multimedia tales como fotografías, sonidos, animaciones, películas y similares. En la figura 1-8 se resumen las características de estos tipos de bases de datos.

Cuando usted concluya este libro deberá ser capaz de diseñar y crear bases de datos y aplicaciones como las que se usaron con Mary Richards y la casa Treble Clef. Probablemente no podrá crear una tan grande y compleja como la de la oficina de registro de vehículos, pero podrá desempeñarse como un miembro eficaz de un equipo que diseñe y cree una. También deberá ser capaz de crear una base de datos pequeña o mediana utilizando la tecnología de Internet.

➤ FIGURA 1-8

Características de los diferentes tipos de bases de datos

Tipo	Ejemplo	Número típico de usuarios concurrentes	Tamaño común de la base de datos
Personal	Pintora de casas Mary Richards	1	<10 Megabytes
Grupo de trabajo	Casa de música Treble Clef	<25	<100 Megabytes
Organizacional	Licencias y registros	Cientos a miles	>1 trillón de bytes
Internet	Reservaciones para la isla Calvert	Cientos a miles	<Cualquiera

➤ LA RELACIÓN DE LOS PROGRAMAS DE APLICACIONES Y LOS DBMS

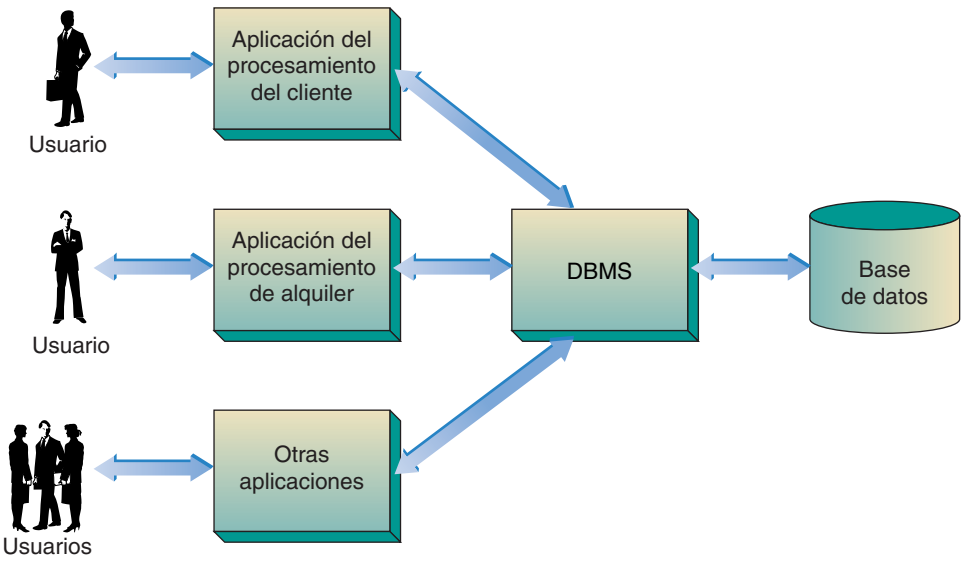
Todos los ejemplos anteriores y, en realidad, todas las aplicaciones de bases de datos tienen la estructura general que se muestra en la figura 1-9: el usuario interactúa con una aplicación que hace interfaz alternadamente con el DBMS, el cual tiene acceso a los datos de la base de datos.

En una época la frontera entre el programa de aplicación y el DBMS estaba claramente definida. Las aplicaciones estaban escritas en lenguajes de tercera generación tales como COBOL, y esas aplicaciones invocaban a los DBMS para la organización de los datos. De hecho, esto aún se hace con mayor frecuencia en las bases de datos de macrocomputadoras.

Sin embargo, actualmente las características y funciones de muchos productos DBMS se han desarrollado tanto que ahora el propio DBMS puede procesar grandes partes de la aplicación. Por ejemplo, la mayoría de los productos DBMS contiene escritores de reportes y generadores de formas que se pueden integrar en una aplicación. Este hecho es importante para nosotros por dos razones. Primera, aunque la mayor parte de es-

➤ FIGURA 1-9

Relaciones de usuarios, aplicaciones de bases de datos, DBMS, y base de datos



te texto considera el diseño y la formación de bases de datos, con frecuencia nos referiremos al diseño y desarrollo de la aplicación de la base de datos. Después de todo, ningún usuario quiere sólo una base de datos. En realidad, los usuarios quieren formas, reportes y consultas que estén basadas en sus datos.

Segunda, de vez en cuando observará una sobreposición entre el material analizado en esta clase y el que estudió en su clase de desarrollo de sistemas, debido a que desarrollar aplicaciones de bases de datos eficaces requiere muchas de las habilidades que ha aprendido o aprenderá en su clase de desarrollo de sistemas. Asimismo, actualmente la mayoría de las clases de desarrollo de sistemas comprenden el diseño de bases de datos. La diferencia entre los dos cursos es el énfasis. Aquí, nuestro enfoque se centra en el diseño, la construcción y el procesamiento de bases de datos. En una clase de sistemas, el interés principal radica en el desarrollo de los sistemas de información, muchos de los cuales usan tecnología de bases de datos.

➤ SISTEMAS DE PROCESAMIENTO DE ARCHIVOS

La mejor forma para entender la naturaleza general y las características de las bases de datos actuales es ver las características de los sistemas que precedieron al uso de la tecnología de bases de datos. Estos sistemas revelan los problemas que ha resuelto dicha tecnología.

Los primeros sistemas de información de negocios almacenaban grupos de registros en archivos por separado y eran llamados sistemas de procesamiento de archivos. Por ejemplo, la figura 1-10 representa dos sistemas de procesamiento de archivos que podría usar Treble Clef. Un sistema procesa los datos de CUSTOMER y otro los datos del RENTAL (ALQUILER).

Aunque los sistemas de procesamiento de archivos han mejorado mucho con respecto a los sistemas manuales de registro, tienen importantes limitaciones:

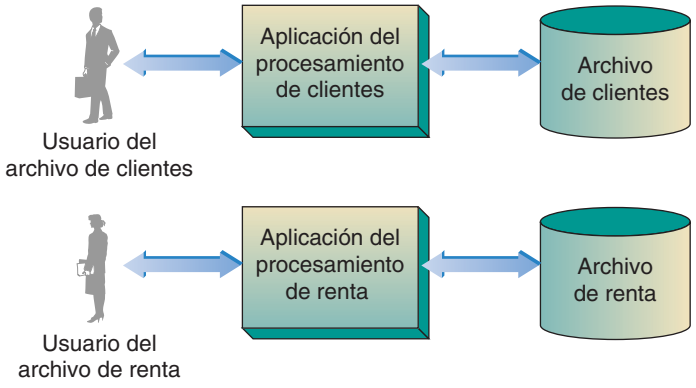
- Los datos están separados y aislados
- La mayoría de los datos están duplicados
- Los programas de aplicación dependen de los formatos de los archivos
- Con frecuencia los archivos son incompatibles entre sí
- Es difícil representar los datos de acuerdo con las perspectivas de los usuarios

DATOS SEPARADOS Y AISLADOS

Los vendedores de Treble Clef necesitan relacionar a sus clientes con los instrumentos que alquilan. Para el sistema de la figura 1-10, los datos necesitan extraerse de algún modo de los archivos CUSTOMER y RENTAL y combinarse en un tercer archivo. Con el procesamiento de archivos esto es difícil de realizar. Primero, los analistas de sistemas y los programadores deben determinar qué partes de cada archivo son necesarias; entonces, tienen que decidir cómo se relacionan entre sí los archivos, y por último, deben coordinar el pro-

➤ FIGURA 1-10

Sistemas de procesamiento de archivos



cesamiento de éstos de tal modo que se extraigan los datos correctos. La coordinación de dos archivos es bastante difícil, pero ¡imagínese la tarea de coordinar 10 o más de ellos!

Duplicación de datos

En el ejemplo de Treble Clef se puede almacenar muchas veces el nombre de un cliente, su dirección y otros datos. Esto es, los datos de CUSTOMER son almacenados una vez y otra más por cada contrato de RENTAL que tenga el cliente. Si bien con esta duplicación de datos se desperdicia espacio de archivo, este no es el problema más serio, sino que esa duplicación afecta la **integridad de datos.**

Un conjunto de datos tiene integridad si éstos son lógicamente consistentes. Con frecuencia se tienen datos de integridad pobre cuando están duplicados. Por ejemplo, si un cliente cambia su nombre o dirección, entonces deben actualizarse todos los archivos que contienen los datos; pero lo peligroso es que todos los archivos *no* puedan actualizarse y causen discrepancias entre sí. En el ejemplo de Treble Clef el cliente no puede tener una dirección para un registro de RENTAL y una dirección diferente para un segundo registro de RENTAL.

Los problemas de integridad de datos son serios. Si los elementos de datos difieren, se producirán resultados inconsistentes e incertidumbre. Si un reporte de una aplicación es diferente al de otra, ¿quién decidirá cuál es el correcto? Cuando los resultados son inconsistentes se duda de la credibilidad de los datos almacenados, e incluso de la propia función MIS.

Dependencia del programa de aplicación

Con el procesamiento de archivos los programas de aplicación dependen de los formatos de los archivos. Usualmente, en sistemas de procesamiento de archivos las formas físicas de archivos y registros son parte del código de aplicación. En COBOL, por ejemplo, los formatos de los archivos están escritos en DATA DIVISION. El problema es que cuando se hacen cambios en los formatos de archivos, también se deben modificar los programas de aplicaciones.

Por ejemplo, si se modifica el registro del cliente para extender el campo de zip code (código postal) de cinco a nueve dígitos, se deben modificar todos los programas que emplean el registro del cliente, aun cuando no se utilice el campo de código postal. Lo anterior se debe a que ahí puede haber 20 programas que procesen el archivo CUSTOMER; esto significa que cuando hay cambio el programador tiene que identificar todos los programas afectados, modificarlos y después probarlos nuevamente; estas tareas consumen tiempo y tienden a producir errores. También, los programadores requieren modificar los programas que no usen el campo cuyo formato ha cambiado, lo cual es una pérdida de dinero.

Archivos incompatibles

Una de las consecuencias que tiene la dependencia de datos del programa es que los formatos de los archivos dependen del lenguaje o del producto que se usó para generarlo. De esta forma, el formato de un archivo procesado con un programa de COBOL es diferente al de uno procesado con un programa C++.

Como resultado, los archivos no se pueden combinar o comparar rápidamente. Se supone, por ejemplo, que el archivo FILE-A contiene los datos de CUSTOMER, que incluyen el CustomerNumber (NúmerodeCliente), y el archivo FILE-B contiene los datos de RENTAL, que también incluyen NúmerodeCliente. Suponga que una aplicación requiere que combinemos registros donde coinciden los números de clientes. Si el FILE-A fuera procesado mediante un programa de Visual Basic, y el FILE-B con un programa C++, necesitaríamos convertir ambos archivos a una estructura común antes de poder combinar los registros. Esto tomaría tiempo y a veces sería difícil. Los problemas empeoran conforme aumenta el número de archivos que hay que combinar.

La dificultad de representar los datos desde la perspectiva de los usuarios

Es difícil representar los datos del procesamiento de archivos de una manera que parezca natural a los usuarios. Ellos quieren ver los datos de RENTAL en un formato como el de la figura 1-5(b). Pero para mostrarlos de esta forma se necesitan extraer, combinar y presentar juntos varios archivos diferentes. Esta dificultad surge porque con el procesamiento de archivos las relaciones entre los registros no se representan o procesan rápidamente. Debido a que un sistema de procesamiento de archivos no puede determinar de inmediato cuál CUSTOMER ha alquilado qué instrumento, es difícil producir una forma que muestre las preferencias del cliente.

➤ SISTEMAS DE PROCESAMIENTO DE BASE DE DATOS

La tecnología de bases de datos se desarrolló para superar las limitaciones de los sistemas de procesamiento de archivos. Para entender cómo, compare el sistema de procesamiento de archivos de la figura 1-10 con el sistema de bases de datos de la figura 1-9. Los programas de procesamiento de archivos acceden directamente a los archivos de datos almacenados. En contraste, los programas de procesamiento de la base de datos invocan al DBMS para tener acceso a los datos almacenados. Esta diferencia es significativa porque facilita la programación de la aplicación; es decir, los programadores de aplicaciones no tienen que preocuparse por cómo se almacenan físicamente los datos, y quedan en libertad de concentrarse en asuntos que sean importantes para el usuario, en lugar de distraerse con aquellos que competen al sistema de computación.

Datos integrados

En un sistema de base de datos todos los datos de aplicación están almacenados en un medio simple llamado **base de datos.** Un programa de aplicación puede ordenarle al DBMS que acceda a los datos del cliente, a los de las ventas, o a ambos. Si los dos son necesarios, el programador de la aplicación sólo especifica cómo se combinan y el DBMS realiza las operaciones necesarias. Así, el programador no es responsable de escribir los programas para consolidar los archivos, como se debe hacer en el sistema de la figura 1-10.

Reducción de datos duplicados

Con el procesamiento de bases de datos la duplicación es mínima. Por ejemplo, en la base de datos de Treble Clef el número de cliente, nombre y dirección se almacenan una sola vez. Siempre que el DBMS necesita estos datos puede recuperarlos y después modificarlos, sólo requiere autorización. Debido a que los datos se almacenan en un solo lugar, los problemas de integridad son menos comunes y hay menos oportunidad de que existan discrepancias entre las múltiples copias de los mismos elementos de los datos.

Independencia de datos/programas

El procesamiento de bases de datos reduce la dependencia de los formatos del archivo. Todos los formatos de registro se almacenan en la misma base (junto con los datos) y el DBMS tiene acceso a ellos, no mediante los de aplicación. A diferencia de los programas de procesamiento de archivos, los de aplicación de base de datos no necesitan incluir el formato de todos los registros y los archivos que procesan. En cambio, sólo los programas de aplicación deben contener una definición (la longitud y el tipo de datos) de cada uno de los datos que se necesitan de la base. El DBMS localiza los datos en los registros y maneja otras transformaciones similares.

La independencia de datos/programas minimiza el impacto de los cambios en el formato de los datos en los programas de aplicación. Los cambios en los formatos se introdu-

cen al DBMS, el cual actualiza los datos y mantiene la relación con la estructura de la base de datos. La mayoría de las veces los programas de aplicación no se enteran de que el formato ha cambiado. Esto también significa que cuando se agregan, cambian o borran datos de la base sólo deben modificarse los programas que usan esos datos en particular. En aplicaciones que constan de docenas de programas esto representa un considerable ahorro de tiempo.

Representación fácil de las perspectivas del usuario

Como descubrirá a través de este texto, la tecnología de bases de datos hace posible representar en una forma sencilla los objetos que se encuentran en el mundo del usuario. Las formas de la figura 1-5 pueden producirse rápidamente a partir de una base de datos, ya que están almacenadas en ella las relaciones entre los registros de datos.

➤ DEFINICIÓN DE UNA BASE DE DATOS

El término *base de datos* tiene muchas interpretaciones diferentes. Se ha usado para referirse a un conjunto de tarjetas indexadas a los volúmenes y volúmenes de datos que un gobierno recopila acerca de sus ciudadanos. En este texto, usamos este término con un significado específico: *una base de datos es un conjunto autodescriptivo de registros integrados*. Es importante comprender plenamente esta definición.

Una base de datos es autodescriptiva

Una base de datos es autodescriptiva: contiene, además de los datos fuente del usuario, una descripción de su propia estructura. Esta descripción se llama **diccionario de datos** (o también **directorio de datos** o **metadatos**).

En este sentido, una base de datos es similar a una biblioteca, la cual integra una colección de libros autodescriptivos. Además de los libros, la biblioteca contiene un catálogo de tarjetas que los reseñan. De la misma manera, el diccionario de datos (que es parte de la base, al igual que el catálogo de tarjetas es parte de la biblioteca) describe lo que contiene la base de datos.

¿Por qué es tan importante la característica autodescriptiva de una base de datos? En primer lugar, porque fomenta la independencia de datos/programas; es decir, hace posible determinar la estructura y el contenido de la base de datos examinándola. No es necesario adivinar qué contiene, ni necesitamos mantener documentación externa del archivo, o de los formatos de registro (como se hace en los sistemas de procesamiento de archivos).

En segundo lugar, si cambiamos la estructura de los datos en la base (por ejemplo, cuando se agregan nuevos datos a un registro existente), sólo se introduce el cambio en el diccionario de datos. Se necesita cambiar muy pocos programas, si es que realmente es muy necesario. En la mayoría de los casos sólo se debe suplir aquellos programas que procesan los datos modificados.

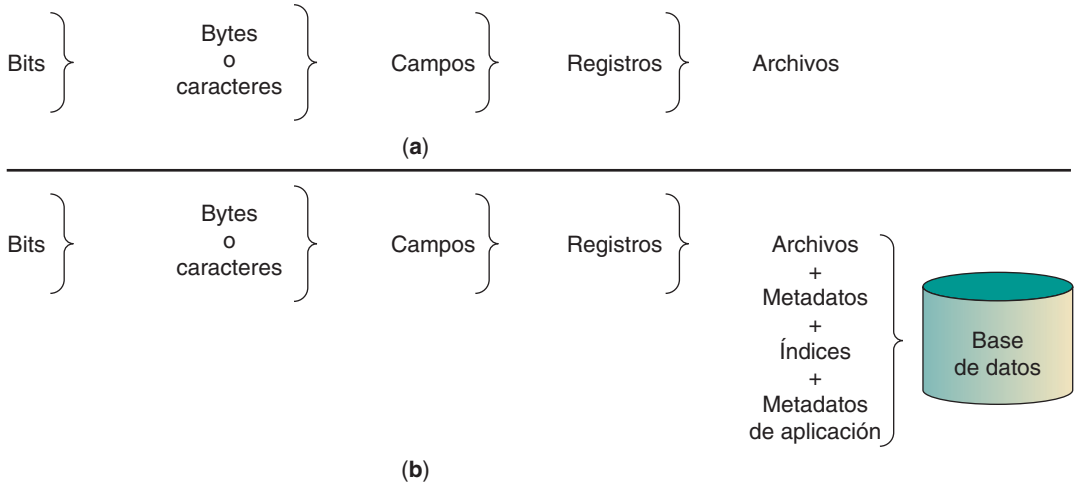
Una base de datos es un conjunto de registros integrados

La jerarquía estándar de los datos es la siguiente: los bits se agrupan en bytes o caracteres; los caracteres se agrupan en campos; los campos integran registros, y los registros se agrupan en archivos (véase la figura 1-11 [a]). Es tentador seguir la tendencia establecida y decir qué archivos conforman las bases de datos. Aunque este enunciado es verdadero, no funciona del todo bien.

Una base de datos incluye archivos de datos del usuario, pero también más. Como mencionamos anteriormente, una base de datos contiene una descripción de sí misma en los metadatos. Además, incluye **índices** que se utilizan para representar las relaciones entre los datos, y también para mejorar el desarrollo de las aplicaciones de la base

➤ FIGURA 1-11

Jerarquía de los elementos de datos en: (a) sistemas de procesamiento de archivos, y (b) sistemas de base de datos



de datos. Por último, la base con frecuencia contiene datos acerca de las aplicaciones que usa. La estructura de la forma de entrada de datos, o de un reporte, a veces es parte de la base. A esta última categoría de datos se le llama **metadatos de aplicación.** Así, una base contiene los cuatro tipos de datos que se muestran en la figura 1-11(b): archivos de datos del usuario, metadatos, índices y metadatos de aplicación.

Una base de datos es un modelo de un modelo

Una base de datos es un modelo. Es tentador decir que es un modelo de la realidad, o de alguna parte de la realidad en la medida en que se relaciona con un negocio. Sin embargo, esto no es verdad. Una base de datos no es un modelo de la realidad ni de una parte de ésta, sino un modelo del *modelo del usuario*. Por ejemplo, la base de datos de Mary Richards es un modelo de la forma en que ella percibe su negocio. Para ella, su negocio tiene clientes, trabajos y referencias, así que su base de datos contiene representaciones de los hechos que relacionan esas características. Los nombres y las direcciones de los clientes, las fechas y las descripciones de sus trabajos, y los nombres de sus fuentes de referencias son medidas importantes para ella, desde el punto de vista de su negocio.

Las bases de datos varían en su nivel de detalle. Algunas son simples y burdas. Una lista de clientes y de las cantidades que éstos adeudan es una representación aproximada del modelo mental de Mary. Una representación más detallada incluye trabajos, referencias y los viajes que se efectúan para llevar a cabo cada trabajo. Una representación muy detallada contiene la cantidad y el tipo de pintura que se utilizó, cuántas brochas se necesitaron y las horas que requirió cada tarea en particular; por ejemplo, medir, pintar madera, pintar paredes, limpiar, etcétera.

El grado de detalle que debe incorporarse a una base de datos depende de la información que se requiera. Evidentemente, mientras más información se necesita, la base de datos debe ser lo más detallada posible. Decidir la cantidad apropiada de detalles es una parte importante del trabajo para el diseño de una base de datos. Como lo descubrirá, el criterio principal es el nivel de detalle que está en la mente de los usuarios.

La base de datos es un modelo dinámico porque cambia los negocios. Las personas van y vienen, los productos surgen y se descontinúan, el dinero se gana y se gasta. De la misma manera en que ocurren estos cambios, los datos que representan los negocios también deben ser modificados. De lo contrario, los datos se volverán obsoletos y representarán erróneamente al negocio.

Las **transacciones** son representaciones de sucesos. Cuando éstos ocurren, las transacciones respectivas se deben procesar en función de la base de datos. Para realizar lo anterior, alguien (un empleado que introduce datos, un vendedor, o un cajero) activa un programa de proceso de transacción e introduce los datos correspondientes. El programa llama al DBMS para modificar la base de datos. Los programas transacción-procesamiento usualmente producen representaciones o respuestas impresas, tales como las confirmaciones de órdenes o los recibos.

➤ HISTORIA DEL PROCESAMIENTO DE LA BASE DE DATOS

El procesamiento de la base de datos fue utilizado originalmente en corporaciones importantes y en grandes organizaciones, como la base de los sistemas de procesamiento de grandes transacciones. Un ejemplo que ya hemos considerado es la oficina de licencias y registro de vehículos. Después, conforme las microcomputadoras ganaron popularidad, la tecnología de bases de datos emigró a las micros y fue utilizada por un sólo usuario, como las aplicaciones de la base de datos personal que describimos en el caso de Mary Richards. Después, conforme se conectaron en conjunto las micros en grupos de trabajo, la tecnología de bases de datos avanzó hacia grupos de trabajo, como en el ejemplo de Treble Clef. Finalmente, ahora las bases de datos se usan para las aplicaciones de Internet e intranet.

EL CONTEXTO ORGANIZACIONAL

La aplicación inicial de la tecnología de bases de datos fue resolver problemas con el sistema de procesamiento de archivos que analizamos anteriormente. A mediados de 1960 las grandes corporaciones estaban produciendo datos con una rapidez impresionante en los sistemas de procesamiento de archivos, pero esos datos se volvían difíciles de manejar y el desarrollo de los nuevos sistemas era cada vez más complicado. Además, se requería que los procesos de administración fueran capaces de relacionar los datos de un sistema de archivo con los de otro.

Las limitaciones del procesamiento de archivos impidieron la fácil integración de los datos. Sin embargo, la tecnología de bases de datos prometió una solución a estos problemas, y las grandes compañías comenzaron a desarrollar bases de datos organizacionales. Las compañías centralizaron sus datos operativos, tales como órdenes de trabajo, inventarios y datos de contabilidad en estas bases. Las aplicaciones fueron inicialmente sistemas de procesamiento de transacción en el ámbito organizacional.

Primero, cuando la tecnología era nueva, las aplicaciones eran difíciles de desarrollar y había muchas fallas. Incluso las aplicaciones que funcionaban eran lentas y poco confiables, el hardware de la computadora no podía manejar rápidamente el volumen de las transacciones, los técnicos de desarrollo aún no habían descubierto formas más eficientes para almacenar y recuperar datos, y los programadores aún no tenían experiencia en el acceso a bases de datos, o a veces sus programas no trabajaban correctamente.

Las compañías se enfrentaron a otra desventaja del procesamiento de bases de datos: la vulnerabilidad. Si un sistema de procesamiento de archivos fallaba, sólo esa aplicación en particular era eliminada del proceso; pero si la base de datos fallaba, todas las aplicaciones dependientes serían eliminadas.

Gradualmente la situación mejoró. Los ingenieros de software y hardware aprendieron a construir sistemas lo suficientemente poderosos como para manejar muchos usuarios a la vez, con la rapidez suficiente para mantener la carga de trabajo diaria de las transacciones. Se planearon nuevas formas de controlar, proteger y respaldar las bases de datos. Evolucionaron los procedimientos normales para el procesamiento de dichas bases, y los programadores aprendieron a escribir códigos más eficientes y sostenibles. A mediados de la década de 1970, las bases de datos podían manejar eficientemente aplicaciones de procesos organizacionales confiables. La mayoría de estas aplicaciones aún funcionan, ¡después de más de 25 años de haber sido creadas!

El modelo relacional

En 1970, E. F. Codd publicó un artículo[1] de vanguardia en el que aplicó conceptos de una rama de las matemáticas llamada álgebra relacional al problema del almacenamiento de grandes cantidades de datos. El artículo de Codd inició un movimiento en la comunidad vinculada a bases de datos, que en muy pocos años llevó a la definición del **modelo de las bases de datos relacionales.** Éste constituye una forma particular de estructuración y procesamiento de una base de datos y se analiza detalladamente en los capítulos 5 y 9-14.

La ventaja del modelo relacional es que los datos se almacenan de tal forma que minimizan la duplicación y se eliminan ciertos tipos de errores de procesamiento que pueden ocurrir cuando se almacenan datos de otras maneras. Los datos se guardan en tablas, con renglones y columnas, como los de la figura 1-1.

De acuerdo con el modelo relacional, no todas las tablas son igualmente deseables. Utilizando un proceso llamado *normalización*, una tabla se puede cambiar por dos o más que ya existen. En el capítulo 5 aprenderá con mayor detalle el proceso de normalización.

Otra ventaja importante del modelo relacional es que las columnas contienen datos que relacionan un renglón con otro. Por ejemplo, en la figura 1-1 CUSTOMER_ID en la tabla JOB se relaciona con CUSTOMER_ID en la tabla CUSTOMER. Esto hace la relación entre los renglones visibles al usuario.

Al principio se pensaba que el modelo relacional permitiría al usuario obtener información de las bases de datos sin ayuda de los profesionales de MIS. Parte de lo razonable de esta idea era que las tablas conforman construcciones simples que son intuitivamente comprensibles. Además, puesto que las relaciones están almacenadas en los datos, los usuarios podrían combinar los renglones cuando fuera necesario. Por ejemplo, para acceder a un registro de RENTAL, un usuario de la casa Treble Clef podría combinar un renglón de la tabla CUSTOMER con los renglones de la tabla RENTAL.

Lo anterior volvió este proceso demasiado difícil para la mayoría de los usuarios. Por lo tanto, nunca se cumplió la promesa del modelo relacional como medio de acceso a la base de datos para aquellos que no son especialistas. En retrospectiva, el principal beneficio del modelo relacional era que proporcionaba una forma estándar para que los especialistas (¡como usted!) estructuraran y procesaran una base de datos.

Productos DBMS para microcomputadoras

En 1979 una pequeña compañía llamada Ashton-Tate introdujo un producto para microcomputadora: el dBase II (que se pronuncia "dibeis dos") al que se le llamó un DBMS relacional. Con una estrategia profesional exitosa, Ashton Tate distribuyó gratis más de 100,000 copias para aquellos que compraron la entonces nueva microcomputadora Osborne. Muchos de los que compraron esas computadoras eran pioneros en la industria de las microcomputadoras. Comenzaron a inventar aplicaciones de microcomputadoras usando dBase, y el número de aplicaciones dBase creció rápidamente. Como resultado, Ashton-Tate se convirtió en una de las primeras y principales corporaciones en la industria de la microcomputación. Después, Borland compró Ashton-Tate, el cual ahora vende la línea de productos dBase.

Sin embargo, el éxito de este producto confundió y desvirtuó el objetivo del procesamiento de la base de datos. El problema era que de acuerdo con la definición prevaleciente a finales de la década de 1970, dBase II no era un DBMS ni tampoco era relacional. En realidad, era un lenguaje de programación con capacidades generalizadas de procesamiento de archivos. Los sistemas que se desarrollaron con dBase II se parecían mucho más a los que se muestran en la figura 1-10 que a los de la figura 1-9. El millón de usuarios de dBase II pensaban que estaban usando un DBMS relacional cuando, en realidad, no era cierto.

Así, los términos *sistema de administración de base de datos* y *base de datos relacional* eran usados libremente cuando comenzó el auge de las microcomputadoras. La mayoría

[1] E. F. Codd, "A relational Model of Data for Large Shared Databanks", *Communications of the ACM*, junio de 1970, pp. 377-387.

de las personas que procesaban una base de datos en microcomputadora realmente administraban archivos y no tenían los beneficios del procesamiento de la base de datos, aunque no se daban cuenta. La situación ha cambiado a medida que el mercado de la microcomputadora se ha vuelto más maduro y sofisticado. Los productos dBase IV y dBase han avanzado, al igual que Foxpro, y son verdaderamente productos DBMS *relacionales*.

Aunque dBase fue un pionero en la aplicación de la tecnología de bases de datos en microcomputadoras, al mismo tiempo otros vendedores comenzaron a trasladar sus productos de las macrocomputadoras a las microcomputadoras. Oracle, Focus e Ingress son tres ejemplos de productos DBMS que voltearon hacia las microcomputadoras. Éstos son realmente productos DBMS y la mayoría estaría de acuerdo en que también son relacionales.

Un impacto del avance de la tecnología con respecto a las bases de datos en la microcomputadora fue la drástica mejoría en las interfaces que emplean los usuarios de los DBMS. Los usuarios de sistemas de microcomputadoras no toleran las torpes y desordenadas interfaces con las que generalmente trabajan los usuarios en los productos DBMS de macrocomputadoras. Así, conforme se inventaron productos DBMS para microcomputadoras, las interfaces de los usuarios se hicieron más fáciles de usar. Esto fue posible porque los productos DBMS operan en computadoras dedicadas y porque computadoras más poderosas están disponibles para procesar la interfaz del usuario. Actualmente los productos DBMS son mejores y más poderosos, con interfaces de usuario gráficas tales como Microsoft Windows.

La combinación de las microcomputadoras, el modelo relacional y las interfaces de usuario bastante mejoradas permitieron que la tecnología de la base de datos pasara de un contexto organizacional a uno de computadora personal. Cuando ocurrió esto, aumentó el número de lugares en los que se usaba la tecnología de las bases de datos. En 1980 había casi 10000 sitios que usaban productos DBMS en Estados Unidos; ahora hay aproximadamente ¡40 millones de sitios!

Aplicaciones de bases de datos cliente-servidor

A mediados de 1980 los usuarios comenzaron a conectar sus microcomputadoras por separado utilizando una red de área local (LAN, por sus siglas en inglés). Dichas redes permitieron a las computadoras el envío de datos de una a otra, a velocidades inimaginables. Las primeras aplicaciones de esta tecnología compartían periféricos, tales como discos rápidos de gran capacidad, impresoras costosas y graficadoras, y facilitaban la intercomunicación a través del correo electrónico. Sin embargo, con el tiempo los usuarios también querían compartir las bases de datos, lo que condujo al desarrollo de aplicaciones multiusuarios en las LAN.

La arquitectura multiusuarios basada en LAN es muy diferente a la que se usa en las bases de datos de macrocomputadoras. Con una de éstas sólo se requiere una CPU en el procesamiento de la aplicación de base de datos, pero con los sistemas LAN pueden confluir simultáneamente varias CPU. El hecho de que esta situación fuera ventajosa (mayor funcionalidad), pero problemática (coordinar las acciones independientes de las CPU), condujo a un nuevo estilo de procesamiento de bases de datos multiusuarios, que se denomina **arquitectura de base de datos cliente-servidor.**

No todo el procesamiento de bases de datos en una LAN es un proceso cliente-servidor. Una forma simple de procesamiento, aunque menos poderosa, se llama **arquitectura de archivos compartidos.** Una compañía como Treble Clef podría usar mejor esta última, puesto que su organización es pequeña y tiene requerimientos de procesamiento modestos. Sin embargo, los grandes grupos de trabajo podrían requerir un procesamiento cliente-servidor. En el capítulo 17 describiremos estos planteamientos y los analizaremos detalladamente.

Bases de datos utilizando la tecnología de Internet

Como se mostró en el ejemplo del centro de reservaciones de la isla Calvert, la tecnología de bases de datos se usa en conjunto con la de Internet para publicar bases de da-

tos en la Web. Esta misma tecnología se usa para publicar aplicaciones sobre intranets corporativas y organizacionales. Algunos expertos creen que con el tiempo todas las aplicaciones de las bases de datos serán distribuidas con el uso de exploradores y de las tecnologías de Internet relacionadas, incluso bases de datos personales que sean "editadas" para una sola persona.

Así, son dos categorías de aplicaciones de bases de datos las que usan la tecnología Internet. Una consiste en las aplicaciones puras de la base de datos en la Web, tales como las aplicaciones de la isla Calvert; otra consiste en la base personal tradicional, grupos de trabajo, y bases de datos organizacionales que no se publican en Internet, pero que usan la tecnología de los exploradores, DHTML y XML. Debido a que es correcto referirse a esta segunda categoría como las bases de datos de Internet, este texto se referirá a ambas categorías como *bases de datos que utilizan la tecnología de Internet.*

Esta categoría maneja la carga actual de la tecnología de las bases de datos. También, como se describe en el capítulo 14, la XML en particular satisface las necesidades de las aplicaciones de las bases de datos excepcionalmente buenas, y es la base de muchos nuevos productos y servicios de base de datos.

Procesamiento de bases de datos distribuidas

Antes de concluir este estudio sobre la historia del procesamiento de las bases de datos, necesitamos analizar dos aspectos que son importantes en teoría, pero que no han sido ampliamente adoptados. El primer procesamiento son las bases de datos distribuidas y el segundo, las bases de datos orientadas a objetos. Analizaremos estos temas con más detalle en los capítulos 17 y 18, respectivamente.

Las aplicaciones de bases de datos organizacionales resuelven los problemas del procesamiento de archivos y permiten un procesamiento de datos organizacionales más integrado. Los sistemas de base de datos personal y de grupos de trabajo causan que la tecnología de las bases de datos esté aún más cerca de los usuarios para permitir el acceso a la base de datos de administración local. Las **bases de datos distribuidas** combinan estos tipos de procesamientos permitiendo las personales, las de grupos de trabajo y las organizacionales para combinarse en bases de datos integradas, pero distribuidas. Así, en teoría, ofrecen aun mayor flexibilidad de acceso a los datos y al procesamiento, pero por desgracia también tienen muchos problemas sin resolver.

La esencia de las bases de datos distribuidas es que se divulgan todos los datos de la organización en muchas computadoras —microcomputadoras, servidores LAN y macrocomputadoras— que se comunican entre sí conforme efectúan los procesos. Los objetivos de los sistemas de bases de datos distribuidas son hacer parecer que cada usuario es el único que tiene acceso a los datos de la organización y proporcionar la misma consistencia, exactitud y menor tiempo del que el usuario podría ocupar si no estuviera usando la base de datos distribuida.

Entre los problemas más apremiantes de las bases de datos distribuidas están los de seguridad y control. Permitir que muchos usuarios (pueden ser cientos) tengan acceso a la base de datos y controlar lo que hacen son tareas complicadas.

Los procesos de sincronización y coordinación pueden ser difíciles. Si un grupo de usuarios baja y actualiza parte de la base de datos y después envía esos cambios a la macrocomputadora, ¿cómo puede evitar el sistema que mientras tanto otro usuario intente usar la versión de los datos que se encuentra en la macrocomputadora? Imagine que este problema involucra docenas de archivos y cientos de usuarios que usan muchísimas piezas del equipo de cómputo.

Mientras que las transiciones del procesamiento de las bases de datos, desde la organizacional hasta la personal y las de los grupos de trabajo, eran relativamente fáciles, las dificultades que enfrentaban los diseñadores de la base de datos y los ingenieros del DBMS distribuido eran monumentales. En realidad, aun cuando durante más de 25 años se ha desarrollado trabajo sobre sistemas de bases de datos distribuidas, todavía persisten problemas importantes. Microsoft ha definido y está construyendo una arquitectura de procesamiento de distribución y una serie de productos de apoyo llama-

do **servidor de transacciones Microsoft (MTS,** por sus siglas en inglés). Aun cuando MTS lo ha prometido —y entre todas las compañías Microsoft es la que tiene los recursos para desarrollar y comercializar un sistema así— no se sabe con certeza si las bases de datos distribuidas podrán satisfacer las necesidades del proceso organizacional actual. Para más información sobre este tema vea el análisis de BD OLE en el capítulo 15.

Los DBMS orientados a objetos (ODBMS)

A finales de la década de 1980 se comenzó a usar un nuevo estilo de programación llamado *programación orientada a objetos* (OOP, por sus siglas en inglés), el cual tiene una orientación muy diferente a la programación tradicional, como se explica en el capítulo 18. En resumen, las estructuras de datos procesados con OOP son mucho más complicadas que las desarrolladas con lenguajes tradicionales. Estas estructuras de datos también son difíciles de almacenar en los productos relacionales DBMS. Como resultado, una nueva categoría de productos DBMS llamados *sistemas de bases de datos orientadas a objetos* está evolucionando para almacenar y procesar estructuras de datos OOP.

Por muchas razones, la OOP aún no se ha usado ampliamente en los sistemas de información de negocios. En primer lugar, las aplicaciones OOP son muy difíciles de usar y es muy costoso desarrollarlas. En segundo, la mayoría de las organizaciones tienen millones o billones de bytes de datos que ya están organizados en bases de datos relacionales y están poco dispuestas a enfrentar el costo y el riesgo de convertir esas bases de datos a un formato ODBMS. Por último, la mayoría de los ODBMS que se han desarrollado apoyan aplicaciones de ingeniería y no tienen características y funciones que sean apropiadas o rápidamente adaptables a las aplicaciones de negocios de información.

Consecuentemente, para el futuro previsible, los ODBMS ocupan un nicho en las aplicaciones de sistemas de información comerciales. En el capítulo 18 analizaremos OOP, bases de datos orientadas a objetos, y una forma híbrida de Oracle llamada base de datos orientada a objetos, pero la mayor parte del análisis se enfoca en el modelo relacional, puesto que las tecnologías se refieren a lo que usted comúnmente usará durante los primeros cinco años de su carrera.

➤ RESUMEN

El procesamiento de bases de datos es uno de los cursos más importantes en la currícula de sistemas de información. Las habilidades y los conocimientos sobre bases de datos tienen gran demanda, no sólo en cuanto a las aplicaciones tradicionales, sino también a las aplicaciones que usan la tecnología de Internet para redes públicas y privadas.

La tecnología de bases de datos se usa en muchas aplicaciones. Algunas sirven para un usuario en una sola computadora; otras se utilizan para grupos de trabajo de 20 a 30 personas en una LAN; otras más las utilizan cientos de usuarios e implican billones de bytes de datos. Recientemente, la tecnología de bases de datos se ha combinado con la de Internet para apoyar las aplicaciones multimedia en redes públicas y privadas.

Los componentes de una aplicación de base de datos son la base de datos misma, el sistema de administración de ésta (los DBMS) y los programas de aplicación. Algunas veces dichos programas están separados por completo del DBMS; otras, algunas partes muy importantes de la aplicación se proporcionan mediante características y funciones del DBMS.

Los sistemas de procesamiento de datos almacenan información en archivos por separado, y cada uno contiene un tipo de datos diferente. Los sistemas de procesamiento de archivos tienen varias limitaciones. Con archivos por separado, es difícil combinar los datos almacenados en archivos diferentes, como los datos que con frecuencia se duplican entre los archivos, lo cual da como resultado problemas de integri-

dad. Los programas de aplicación dependen de los formatos de archivo, lo cual causa problemas de mantenimiento cuando cambian los formatos y los archivos se hacen incompatibles; por lo tanto, se requiere la conversión de archivos. También es difícil representar datos desde las perspectivas de los usuarios.

Los sistemas de procesamiento de bases de datos se desarrollaron para superar estas limitaciones. En el medio ambiente de las bases de datos, el DBMS es la interfaz entre los programas de aplicación y las bases de datos. Los datos están integrados y se reduce la duplicación. Los cambios en los formatos físicos de los datos almacenados sólo afectan al DBMS. Si se cambian, agregan o borran conjuntos de datos, pocos programas de aplicación requerirán mantenimiento. Con la tecnología de bases de datos es fácil representar objetos en el medio ambiente de los usuarios.

Una base de datos es un conjunto autodescriptivo de registros integrados. Es autodescriptivo porque contiene una descripción de sí misma en un diccionario de datos, el cual también se conoce como directorio de datos o metadatos. Una base de datos es un conjunto de registros integrados porque la relación entre éstos se almacena en la base. Este arreglo permite al DBMS construir incluso objetos complicados mediante la combinación de datos con base en las relaciones almacenadas. Las relaciones con frecuencia se almacenan en las bases de datos importantes. Así, las tres partes que conforman una base son: la aplicación de datos, el diccionario y los datos importantes.

La tecnología de bases de datos se desarrolló en varias etapas. Las primeras bases se concentraban en el procesamiento de transacción de datos organizacionales. Después, el modelo relacional, junto con las microcomputadoras, permitió al usuario el uso de aplicaciones personalizadas. Con la llegada de las LAN, los departamentos comenzaron a implementar bases de datos cliente-servidor para grupos de trabajo. Actualmente, Internet y las aplicaciones tradicionales están siendo repartidas mediante el uso de la tecnología de Internet. El procesamiento de distribución y bases orientadas a objetos son temas importantes en el procesamiento de bases de datos. Sin embargo, hasta el momento no han tenido éxito comercial ni su uso se ha difundido ampliamente en las aplicaciones para los negocios.

➤ PREGUNTAS DEL GRUPO 1

1.1 ¿Por qué es un tema importante el procesamiento de las bases de datos?

1.2 Describa la naturaleza y las características de una aplicación de base de datos para un solo usuario, como la que se empleó en el caso de Mary Richards.

1.3 Describa la naturaleza y las características de una aplicación de bases de datos que utiliza un grupo de trabajo, como en el caso de la casa de música Treble Clef.

1.4 Describa la naturaleza y las características de una aplicación de bases de datos que emplea una organización como la de la oficina de licencias y registro de vehículos.

1.5 Describa la naturaleza y las características de una aplicación de bases de datos en una organización como la del centro de reservación de la isla Calvert.

1.6 Explique la naturaleza y la función de cada uno de los componentes de la figura 1-9.

1.7 ¿Cómo cambia con el tiempo la relación entre los programas de aplicación y el DBMS?

1.8 Enumere las limitaciones de los sistemas de procesamiento de archivos, como se describió en este capítulo.

1.9 Explique de qué manera la tecnología de la base de datos superó las limitaciones que enumeró en su respuesta de la pregunta 1.8.

1.10 Defina el término *base de datos*.

1.11 ¿Qué son los metadatos? ¿Qué son los índices? ¿Cuáles son las aplicaciones de los metadatos?

1.12 Explique por qué una base de datos es un modelo. Describa la diferencia entre un modelo de la realidad y un modelo de un modelo de la realidad de un usuario. ¿Por qué es importante esta diferencia?

1.13 Dé un ejemplo, diferente al de este capítulo, sobre una aplicación de bases de datos personales.

1.14 Proporcione un ejemplo, diferente al de este capítulo, referente a una aplicación de base de datos de un grupo de trabajo.

1.15 Mencione un ejemplo, diferente al que se menciona en este capítulo, sobre una aplicación de bases de datos en una empresa grande.

1.16 ¿Cuáles eran algunas de las debilidades de las primeras aplicaciones de las bases de datos?

1.17 ¿Cuáles son dos de las principales ventajas del modelo relacional?

1.18 Resuma los eventos en el desarrollo de los productos DBMS para microcomputadora.

1.19 ¿Cuál fue el factor principal que dio lugar a las aplicaciones de la base de datos de grupo de trabajo?

1.20 ¿Cuál es la diferencia entre las arquitecturas cliente-servidor y las multiusuarios?

1.21 ¿Cuál es la diferencia entre una aplicación de base de datos de Internet y una que usa la tecnología de Internet?

1.22 Explique la naturaleza general del procesamiento distribuido. ¿Cuáles son algunos de los problemas que representa?

1.23 Describa el propósito de una base de datos orientada a objetos. ¿Por qué estas bases de datos no han sido las más aceptadas para las aplicaciones de sistemas de información?

➤ PROYECTOS

A. Ingrese al sitio Web de un fabricante de computadoras, como por ejemplo Dell (www.dell.com). Use el sitio Web para determinar qué modelo de computadora portátil (*laptop*) recomendaría con un costo máximo de $2500*. ¿Cree que se usan una o más bases de datos para manejar este sitio? Si es así, mencione cuáles funciones o características del sitio Web podrían ayudar a la tecnología de las bases de datos, considerando tanto la definición de una base de datos como las ventajas de su procesamiento.

B. Entre al sitio Web de un vendedor de libros al detalle, como por ejemplo Amazon (www.amazon.com). Use el sitio Web para localizar la biografía más reciente de William Wordsworth. Desde su punto de vista, ¿cree que se usan una o más bases de datos para este sitio? Si es así, ¿cuáles características y funciones del sitio Web cree que deberían ser de más ayuda para la tecnología de base de datos, considerando tanto las definiciones de una base de datos como las ventajas del procesamiento de base de datos?

➤ PREGUNTAS DEL PROYECTO FIREDUP

La compañía FiredUp es un negocio pequeño, cuyos propietarios son Curt y Julia Robard. La sede está en Brisbane, Australia. FiredUp fabrica y vende una estufa para campo ligera, llamada FiredNow. Curt, quien trabajó previamente como ingeniero aeroes-

*Tome en cuenta el lector que todas las cifras que aparecen en este libro se expresan en dólares estadounidenses, a menos que se indique lo contrario. (N. de la R.)

pacial, inventó y patentó un mechero con el que la estufa permanecerá encendida aun cuando el viento sea superior a los 160 kilómetros por hora. Julia, una diseñadora industrial muy capaz, desarrolló un elegante diseño plegadizo pequeño, ligero, fácil de instalar y muy estable. Los Robard fabrican las estufas en su cochera y las venden directamente a sus clientes a través de Internet, fax o correo.

Los propietarios de FiredUp necesitan dar seguimiento a las estufas que han vendido, con el fin de poder entrar en contacto con sus usuarios en caso de fallas del producto u otros asuntos relacionados con éste. También piensan que podrían usar su lista de clientes para comercializar otros productos que desarrollen en el futuro.

A. ¿Qué base de datos cree que sería la apropiada para que FiredUp pudiera dar seguimiento a sus estufas y dispusiera de los datos de sus clientes? Explique en qué circunstancias sería apropiada una base de datos y en cuáles no sería adecuada. Describa cuándo convendría una base de datos personal. ¿En qué casos sería conveniente una base de datos de un grupo de trabajo? ¿En qué circunstancias le convendría a FiredUp tener una base de datos en Internet?

B. Aplique el mismo problema al registro de un producto que vende la compañía Café Starbucks. Digamos, por ejemplo, que dicha compañía quiere desarrollar la capacidad de dar seguimiento a los clientes que le compran máquinas para café exprés. ¿Cómo responde a las preguntas del inciso A para este caso en particular?

CAPÍTULO 2

Introducción al desarrollo de una base de datos

Este capítulo presenta un panorama sobre el desarrollo y la aplicación de una base de datos. Comenzamos con una descripción de los elementos que conforman una base de datos y con un análisis de las características y funciones de un DBMS. A continuación ilustramos la creación y el funcionamiento de una base de datos. Por último, analizamos las estrategias comunes del desarrollo de dicha base. El objetivo del presente capítulo es crear una perspectiva para abordar las descripciones detalladas de la tecnología en los siguientes capítulos.

➤ LA BASE DE DATOS

La figura 2-1 muestra los componentes principales de un sistema de base de datos. El **DBMS** procesa la **base de datos,** y lo utilizan tanto los programadores como los usuarios, quienes pueden ingresar al DBMS directa o indirectamente mediante los **programas de aplicación.** En esta sección analizaremos la base de datos y en las siguientes secciones, el DBMS y los metadatos de aplicaciones.

Como establecimos en el capítulo anterior, una base de datos contiene cuatro elementos principales: los datos del usuario, los metadatos, los índices y los metadatos de aplicaciones.

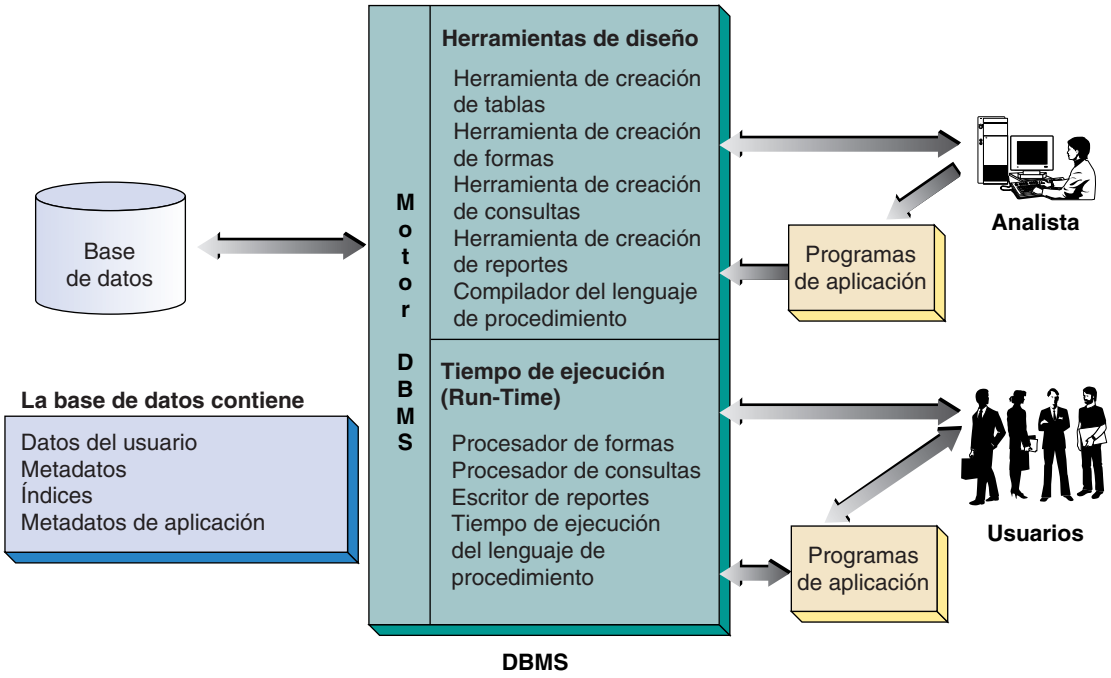
DATOS DEL USUARIO

Actualmente la mayoría de las bases de datos representan a los datos del usuario como relaciones. El término *relación* lo definiremos formalmente en el capítulo 5. Por el momento, considere una relación como una tabla de datos. Las columnas de ésta contienen campos o atributos, y los renglones, registros de entidades particulares en el campo de los negocios.

No todas las relaciones son igualmente convenientes, algunas están mejor estructuradas que otras. El capítulo 5 describe un proceso llamado normalización, que se uti-

➤ FIGURA 2-1

Componentes de los sistemas de bases de datos



liza para crear relaciones bien estructuradas. Para tener una idea sobre la diferencia entre las relaciones pobremente estructuradas y las bien estructuradas, considere la relación R1 (NombredelEstudiante, TeléfonodelEstudiante, NombredelAsesor, TeléfonodelAsesor) con los siguientes datos:

NombredelEstudiante	TeléfonodelEstudiante	NombredelAsesor	TeléfonodelAsesor
Baker, Rex	232-8897	Parks	236-0098
Charles, Mary	232-0099	Parks	236-0098
Johnson, Beth	232-4487	Jones	236-0110
Scott, Glenn	232-4444	Parks	236-0098
Zylog, Frita	232-5588	Jones	236-0110

El problema con esta relación es que contiene datos respecto a dos temas diferentes: estudiantes y asesores. Una relación estructurada de esta manera presenta diversos problemas cuando se actualiza. Por ejemplo, si el asesor Parks cambia su número telefónico, habrá que modificar tres renglones de datos. Por esta razón, sería mejor representar los datos mediante las dos relaciones R2 (NombredelEstudiante, TeléfonodelEstudiante, NombredelAsesor) con los datos:

NombredelEstudiante	TeléfonodelEstudiante	NombredelAsesor
Baker, Rex	232-8897	Parks
Charles, Mary	232-0099	Parks
Johnson, Beth	232-4487	Jones
Scott, Glenn	232-4444	Parks
Zylog, Frita	232-5588	Jones

y R3 (NombredelAsesor, TeléfonodelAsesor) con los datos:

NombredelAsesor	TeléfonodelAsesor
Parks	236-0098
Jones	236-0110

Si un asesor cambia su teléfono sólo habrá que modificar un renglón. Por supuesto, para producir un reporte que muestre los nombres de los estudiantes junto con los números telefónicos de los asesores, se tendrían que combinar los renglones de estas dos tablas. Sin embargo, almacenar por separado las relaciones y combinarlas cuando se produce un reporte resulta mucho mejor que almacenarlas como una tabla combinada.

METADATOS

Como establecimos en el capítulo 1, una base de datos se describe por sí misma, lo que significa que contiene una descripción de su estructura como parte de sí misma. Esta descripción se llama **metadatos.** Debido a que los productos DBMS están diseñados para almacenar y manejar tablas, la mayoría de los productos almacenan los metadatos en forma de tablas, a las que a veces se denomina **tablas del sistema.**

La figura 2-2 muestra un ejemplo de los metadatos guardados en las dos tablas del sistema. La primera almacena una lista de tablas que se encuentran en la base datos, indicando cuántas columnas se encuentran en cada tabla y qué columna(s) se encuentra(n) en la llave primaria. Dicha columna es la única que identifica un renglón. La segunda tabla almacena una lista de columnas en cada tabla y el tipo y longitud de los datos de cada columna. Estas dos tablas son comunes de las tablas de sistema; otras parecidas almacenan listas de índices, llaves, procedimientos almacenados y cosas por el estilo.

➤ FIGURA 2-2

Ejemplos de metadatos

Tabla del sistema de tablas

Nombre de la tabla	Número de columnas	Llave primaria
Estudiante	4	NúmerodeEstudiante
Asesor	3	NombredelAsesor
Curso	3	NúmerodeReferencia
Registro	3	{NúmerodelEstudiante, NúmerodeReferencia}

Tabla de las columnas del sistema

Nombre de la columna	Nombre de la tabla	Tipo de datos	Longitud*
NúmerodeEstudiante	Estudiante	Entero	4
Nombre	Estudiante	Texto	20
Apellido	Estudiante	Texto	30
Especialidad	Estudiante	Texto	10
NombredelAsesor	Asesor	Texto	25
Teléfono	Asesor	Texto	12
Departamento	Asesor	Texto	15
NúmerodeReferencia	Curso	Entero	4
Título	Curso	Texto	10
NúmerodeHoras	Curso	Decimal	4
NúmerodeEstudiante	Registro	Entero	4
NúmerodeReferencia	Registro	Entero	4
Calificación	Registro	Texto	2

* Las longitudes están en bytes, que son iguales al número de caracteres de los datos del texto.

Almacenar metadatos en tablas no sólo es eficiente para el DBMS; también conviene para los analistas debido a que pueden utilizar las mismas herramientas de consulta para los metadatos como las que se emplean para los datos del usuario. En el capítulo 9 analizaremos un lenguaje llamado SQL que se utiliza para consultar y actualizar las tablas para los metadatos y los datos del usuario.

Con el propósito de mostrarle un ejemplo de cómo puede utilizar el SQL, suponga que ha desarrollado una base de datos con 15 tablas y 200 columnas. Recuerde que muchas de las columnas contienen el tipo de datos *moneda*, pero no puede recordar cuáles. Al utilizar el SQL puede ingresar a la tabla de las columnas del sistema para descubrir cuáles son las columnas que contienen ese tipo de datos.

Índices

Un tercer tipo de datos de la base mejora el funcionamiento y el acceso a la base de datos. Estos datos, a los que a veces se les llama **datos significativos,** constan principalmente de índices, aunque algunas veces se utilizan otros tipos de estructuras de datos, tales como las listas vinculadas (véase el Apéndice A para un análisis de índices y de las listas vinculadas).

La figura 2-3 muestra una tabla de datos de estudiantes y dos índices. Para demostrar la utilidad de tener estos índices, suponga que los datos de NúmerodeEstudiante están almacenados en un disco en orden ascendente y que el usuario quiere imprimir un reporte de los datos de los estudiantes por apellido. Para efectuarlo, se pueden extraer todos los datos de la tabla fuente y ordenarlos; a menos que ésta sea pequeña, este proceso requiere tiempo. Alternativamente, se puede crear un índice por apellido, como el de la figura 2-3. Las entradas en este índice están ordenadas por el valor de Apellido, por lo que las entradas del índice se pueden leer y utilizar para ingresar los datos de los estudiantes por orden.

➤ FIGURA 2-3

Ejemplos de índices de una base de datos

Ejemplo de tabla ESTUDIANTE

Númerode Estudiante	Nombre	Apellido	Especialidad
100	James	Baker	Contaduría
200	Mary	Abernathy	Sistemas de información
300	Beth	Jackson	Contaduría
400	Eldridge	Johnson	Mercadotecnia
500	Chris	Tufte	Contaduría
600	John	Smathers	Sistemas de información
700	Michael	Johnson	Contaduría

Índice por apellido

Apellido	NúmerodeEstudiante
Abernathy	200
Baker	100
Jackson	300
Johnson	400, 700
Smathers	600
Tufte	500

Índice por especialidad

Especialidad	NúmerodeEstudiante
Contaduría	100, 300, 500, 700
Sistemas de información	200, 600
Mercadotecnia	400

Ahora suponga que los datos de los estudiantes deben imprimirse junto con la especialidad de éstos. De nuevo, se pueden extraer los datos de la tabla de referencia y clasificarlos, o se puede construir un índice como el índice por especialidad y utilizarlo como lo describimos.

Los índices no sólo se utilizan para la clasificación, sino para el rápido acceso a los datos. Por ejemplo, un usuario desea ingresar sólo aquellos estudiantes que tienen el valor de "Sistemas de información" en la especialidad. Sin un índice, habría que revisar toda la tabla de referencia; pero cuando éste existe, se puede encontrar la entrada del índice y utilizarla para localizar todos los renglones idóneos. Aunque no se necesitan índices para una tabla con pocos renglones, como la de ESTUDIANTE en la figura 2-3, considere una tabla que tenga 10000 o 20000 renglones de datos. En ese caso, la clasificación y la revisión serían muy lentas.

Los índices son útiles para las operaciones de clasificación y revisión, pero tienen un costo. Cada vez que se actualiza un renglón en la tabla ESTUDIANTE, también se deben actualizar los índices. Esto no necesariamente es malo; sólo significa que los índices no son independientes y por lo tanto deben reservarse para los casos en que sean realmente necesarios.

Metadatos de aplicación

El cuarto y último tipo de datos en la base son los **metadatos de aplicación,** los cuales se utilizan para almacenar la estructura y el formato de formas del usuario, reportes, consultas y otros componentes de aplicación. No todos los productos DBMS sustentan componentes de aplicación, y en aquellos productos que lo hacen no todos almacenan la estructura de dichos componentes como metadatos de aplicación en la base de datos. Sin embargo, la mayoría de los productos DBMS modernos sí almacenan estos datos como parte de la base. En general, ni los analistas ni los usuarios ingresan directamente a los metadatos de aplicación, sino que utilizan herramientas del DBMS para procesarlos.

➤ DBMS

Los productos DBMS varían considerablemente en cuanto a características y funciones. Los primeros de estos productos se desarrollaron en la década de 1960 para ser usados en macrocomputadoras y tenían características muy primitivas. Desde entonces, los productos DBMS han sido continuamente perfeccionados y mejorados no sólo para procesar mejor los datos de las bases, sino también para incorporar características que faciliten la creación de las aplicaciones de dichas bases.

En este capítulo utilizaremos Microsoft Access 2002 para ilustrar las capacidades de los productos DBMS. Esto obedece a que Access 2002 tiene las características y funciones de un DBMS moderno. Sin embargo, no es el único DBMS y su elección en este caso no significa una preferencia especial por encima de otros productos similares, como el Lotus Approach.

Como se muestra en la figura 2-1, las características y funciones de un DBMS pueden dividirse en tres subsistemas: herramientas de diseño, tiempo de ejecución y motor DBMS.

El subsistema de herramientas de diseño

El subsistema de herramientas de diseño consta de un conjunto de implementos que facilitan el diseño y la creación de la base de datos y de sus aplicaciones. Por lo general incluye herramientas para la creación de tablas, formas, consultas y reportes. Los productos DBMS también proporcionan lenguajes e interfaces de programación para programar con

lenguajes. Por ejemplo, Access tiene dos lenguajes: un macrolenguaje que no requiere conocimientos profundos sobre programación y una versión de BASIC llamada Visual Basic.

Subsistema del run-time (tiempo de ejecución)

El subsistema run-time[1] procesa los componentes de aplicación que se desarrollan al utilizar las herramientas de diseño. Por ejemplo, Access 2002 tiene una facilidad de run-time que materializa las formas y conecta los elementos de la forma con los datos de la tabla. Suponga que se ha descrito una forma que incluye un cuadro de texto, el cual expone el valor de NúmerodeEstudiante de la tabla ESTUDIANTE. Durante la ejecución, cuando se abre la forma, el procesador run-time de ésta extrae el valor del NúmerodeEstudiante del renglón actual en la tabla ESTUDIANTE y lo despliega en la forma. Todo esto es automático; una vez que se crea la forma, ni el usuario ni el analista necesitan hacer nada. Otros procesadores run-time responden a consultas e imprimen reportes. Además, hay un componente del run-time que procesa los requerimientos del programa de aplicación para leer y escribir los datos de la base.

Aunque no se muestra en la figura 2-1, también los productos DBMS deben proporcionar una interfaz para los lenguajes estándares tales como el C++ y el Java. Usted aprenderá más acerca de esto en los capítulos 15 y 16.

El motor DBMS

El tercer componente del DBMS es el motor DBMS, el cual es el intermediario entre las herramientas de diseño y los subsistemas run-time y los datos. El motor DBMS recibe los requerimientos de los otros dos componentes —planteados en términos de tablas, renglones y columnas— y traduce los requerimientos en órdenes dirigidas al sistema operativo para leer y escribir datos en medios físicos.

El motor DBMS también está involucrado en el manejo de transacciones, bloqueo, respaldo y recuperación. Como se muestra en el capítulo 11, las acciones en contra de la base de datos con frecuencia deben hacerse como un entero. Cuando se procesa una orden, por ejemplo los cambios en las tablas CLIENTE, PEDIDO e INVENTARIO deben realizarse en conjunto; o se hace en todas o en ninguna. El motor DBMS ayuda a coordinar las actividades para asegurarse de que se aplicó todo o nada del grupo.

Microsoft proporciona dos motores distintos para Access 2002: el motor Jet y el SQL Server; el primero se utiliza para pequeñas bases de datos personales y de trabajo en grupo. El SQL Server, el cual es un producto independiente de Microsoft, se utiliza para grandes bases de datos departamentales y organizacionales de tamaños pequeño a mediano. Cuando usted crea una base de datos utilizando las capacidades originales para generación de tablas de Access 2002 (bases de datos almacenadas con el sufijo .mdb), está utilizando el motor Jet (Jet Engine). Cuando crea proyectos de Access 2002 (con sufijo .adp) está creando una interfaz de aplicación para el motor del SQL Server.

➤ CREACIÓN DE LA BASE DE DATOS

El *esquema de una base de datos* define la estructura de ésta, sus tablas, relaciones, dominios y reglas del negocio. El esquema de una base de datos es un diseño, la base sobre la cual se construyen ésta y las aplicaciones.

---

[1] No confunda el término *subsistema run-time* con el término *producto de run-time*. Algunos vendedores utilizan el término *producto de run-time* para referirse a aquel que incluye los componentes run-time y el motor DBMS, pero no al subsistema de las herramientas de diseño. Dicho producto puede utilizarse para procesar una aplicación que ya ha sido desarrollada. El propósito de los productos run-time es reducir el costo de la aplicación al usuario final. Por lo general, el producto run-time es menos costoso (a veces gratuito) que el DBMS completo. De ahí que sólo el analista compre el producto completo, que incluye el subsistema de las herramientas de diseño; los usuarios finales sólo compran el producto run-time.

Ejemplo de un esquema

Para ilustrar un esquema y saber por qué es importante, considere un ejemplo. El Colegio Highline es una pequeña escuela que imparte actividades físicas. Su departamento de actividades estudiantiles patrocina ligas locales de atletismo, y el gran problema consiste en no perder de vista al equipo de atletismo, que ha sido asignado a varios capitanes de equipo. Para este sistema se utilizan los siguientes componentes del esquema:

TABLAS La base de datos contiene dos tablas:[2]

CAPTAIN (CaptainName, Phone, Street, City, State, Zip)
(CAPITÁN [NombredelCapitán, Teléfono, Calle, Ciudad, Estado, CódigoPostal])

y

ITEM (Quantity, Description, DateOut, DateIn)
(ARTÍCULO [Cantidad, Descripción, FechadeSalida, FechadeEntrada])

donde los nombres de las tablas se muestran fuera de los paréntesis y los de las columnas, entre paréntesis.

Ni CaptainName ni Description son necesariamente un nombre único; dos capitanes podrían fácilmente llamarse "Mary Smith", o seguramente hay muchos artículos denominados "Soccer Balls". Para asegurarse de que cada renglón puede ser identificado (la importancia de esto se aclarará en capítulos posteriores), se agregarán dos columnas de números únicos a estas tablas, como se muestra a continuación:

CAPTAIN (CAPTAIN_ID, CaptainName, Phone, Street, City, State, Zip)
(CAPITÁN [CAPITÁN_ID, NombredelCapitán, Teléfono, Calle, Ciudad, Estado, CódigoPostal])
ITEM (ITEM_ID, Quantity, Description, DateOut, DateIn)
(ARTÍCULO [ARTÍCULO_ID, Cantidad, Descripción, FechadeSalida, FechadeEntrada])

RELACIONES. La relación entre estas dos tablas es como sigue: un renglón de CAPTAIN se relaciona con muchos renglones de ITEM; sin embargo, un renglón ITEM se relaciona con un renglón CAPTAIN. La notación para una relación como ésta es **1:N** y se pronuncia "uno a N" o "uno a muchos". El término **1:N** significa que un renglón de la primera tabla se relaciona con muchos renglones de la segunda.

Para las tablas que se muestran aquí no hay manera de distinguir qué renglón de CAPTAIN se relaciona con otros renglones de ITEM. De ahí que, para mostrar dicha relación, se le agrega CAPTAIN_ID al renglón ITEM. La estructura completa de las dos tablas es la siguiente:

CAPTAIN (CAPTAIN_ID, CaptainName, Phone, Street, City, State, Zip)
ITEM (ITEM_ID, Quantity, Description, DateOut, DateIn, CAPTAIN_ID)

Con esta estructura es fácil determinar qué capitán ha tomado prestado determinado ITEM. Por ejemplo, para descubrir quién pidió el artículo 1234, se examina el renglón correspondiente y se encuentra el valor de CAPTAIN_ID almacenado en tal renglón. Entonces se puede utilizar dicho valor para almacenar el nombre y el número telefónico de ese capitán.

DOMINIOS. Un dominio[3] es un conjunto de valores que puede tener una columna. Considere los dominios de las columnas de la tabla ITEM. Suponga que tanto CAPTAIN-ID como Quantity son números enteros, que Description es un texto de longitud 25, que

[2] Como se muestra en los capítulos 3 al 7, la tarea más importante y difícil para el desarrollo de una base de datos es diseñar la estructura de la tabla. Al iniciar este ejemplo con las tablas ya definidas se ha pasado por alto una gran parte del proyecto.

[3] Este análisis se simplifica considerablemente al concentrarse en los componentes del sistema de una base de datos. En el capítulo 4 encontrará un análisis más completo sobre los dominios.

DateOut y DateIn tienen el dominio de la fecha, y que también CAPTAIN_ID tiene el dominio de números enteros. Además del formato físico, también se necesita decidir si alguno de los dominios será único para la tabla. En este ejemplo, quisimos que ITEM-ID fuera único, y por lo tanto se debe especificar su dominio de esta manera. Puesto que un capitán puede haber tomado más de un artículo, CAPTAIN_ID no es único para la tabla ITEM.

También se deben especificar los dominios de las columnas CAPTAIN. CAPTAIN_ID es entero y el resto de las columnas son textos de diferentes longitudes. CAPTAIN_ID debe ser único en la tabla CAPTAIN.

REGLAS DEL NEGOCIO. El último elemento del esquema de una base de datos son las reglas del negocio, las cuales son restricciones en las actividades del negocio y necesitan reflejarse en la base de datos al igual que en sus aplicaciones. Los siguientes son ejemplos de reglas del negocio del Colegio Highline:

1. Para pedir prestado cualquier artículo, un capitán debe tener un número telefónico local.
2. En ningún momento ningún capitán tendrá en préstamo más de siete balones de fútbol.
3. Los capitanes deben regresar todos los artículos cinco días después de que termine el semestre.
4. Ningún capitán podrá pedir más equipo si ha retrasado la entrega de algún artículo.

Las reglas del negocio son una parte muy importante del esquema, debido a que especifican las restricciones de los valores de datos permitidos que deben imponerse, sin importar la manera en que los cambios de los datos afecten al motor DBMS. Independientemente de que la petición de un cambio de datos provenga del usuario de una forma, de una consulta-requerimiento actualizada, o de un programa de aplicación, el DBMS debe asegurarse de que el cambio no viole ninguna regla.

Por desgracia, las reglas del negocio son impuestas de diferentes maneras por distintos productos DBMS. Con Access 2002 pueden definirse ciertas reglas en el esquema y ser puestas en práctica automáticamente. Con productos como el SQL Server y ORACLE, se imponen reglas del negocio adicionales mediante un dispositivo denominado *procedimientos almacenados (stored procedures)*. En algunos casos, el producto DBMS no tiene capacidad para imponer las reglas del negocio necesarias y éstas deben codificarse como programas de aplicación. Analizaremos este tema con mayor detalle en el capítulo 10.

Creación de tablas

Una vez que se ha diseñado el esquema, el siguiente paso es crear tablas de bases de datos utilizando las herramientas para la creación de tablas del DBMS. La figura 2-4 muestra la forma que se utiliza con Microsoft Access para crear la tabla ITEM. El nombre de cada columna de la tabla está representado en la columna Field Name, y la representación de los datos de la columna se especifica en la columna Data Type. La columna Description se utiliza para registrar la documentación opcional y los comentarios con respecto a las columnas de la tabla. Los datos adicionales —como longitud del texto, formato del campo, título y otros— se especifican en los campos de entrada en la columna de la forma inferior a mano izquierda.

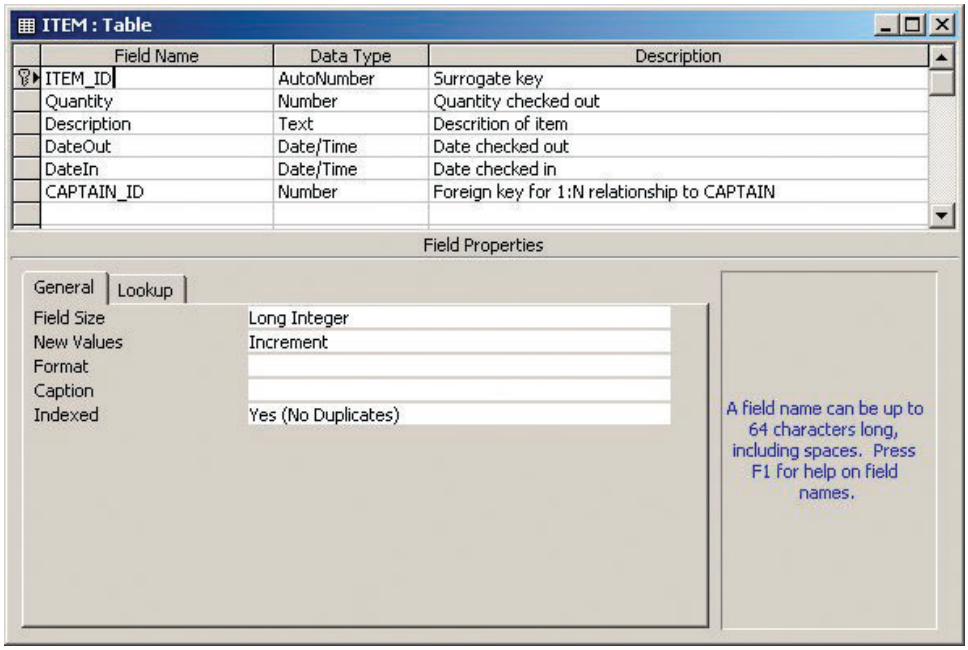
En la figura 2-4 la información importante se encuentra en la columna ITEM_ID. Observe que la propiedad indexada en la parte inferior de la forma ha sido programada como Yes (No Duplicates), lo que significa que un índice de valores únicos se creará para la columna ITEM_ID. Para completar la definición de la base de datos, la tabla CAPTAIN se crea de una manera similar.

Definición de relaciones

La relación entre CAPTAIN e ITEM es 1:N, lo cual se representa en el esquema mediante la colocación de la llave de CAPTAIN en la tabla ITEM. En la figura 2-4 colocamos CAPTAIN_ID en la tabla ITEM. Una columna como la de CAPTAIN_ID en la tabla ITEM a veces se denomina **llave externa** debido a que es la llave de una tabla que es externa a la tabla en la cual reside. Cuando se crean formas, consultas y reportes, el DBMS puede pro-

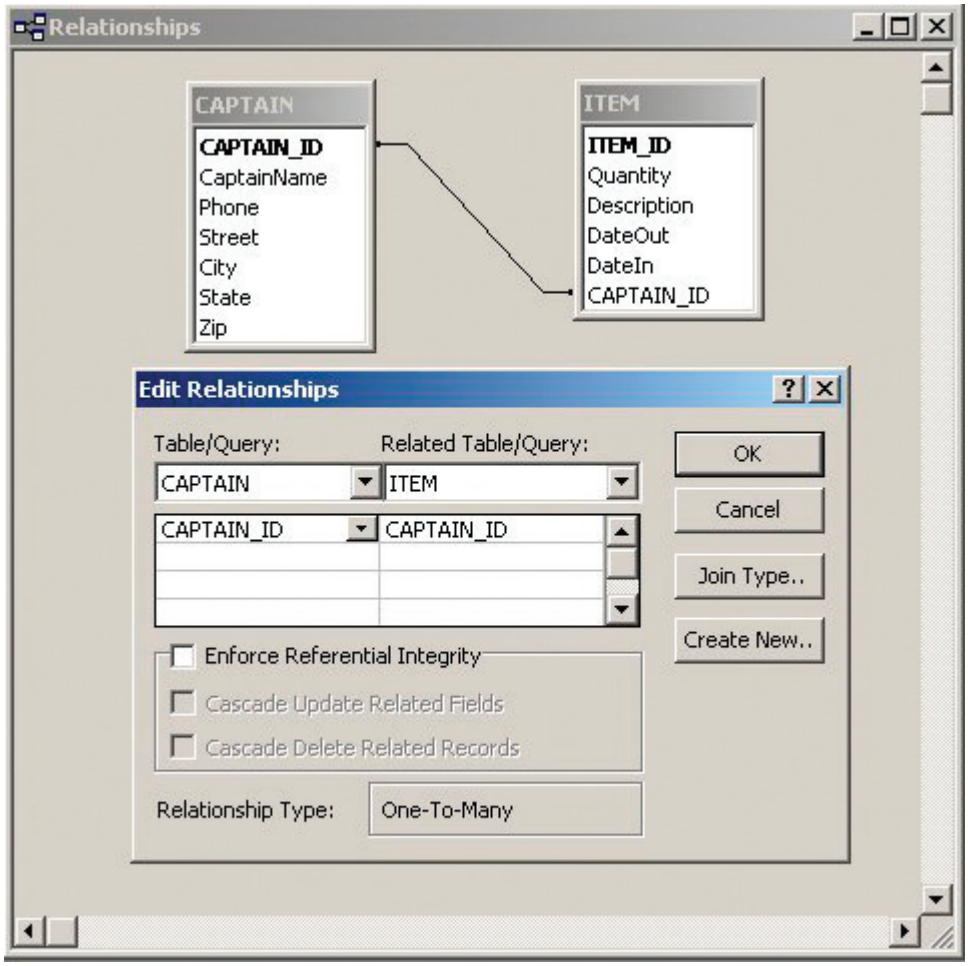
➤ FIGURA 2-4

Creación de una tabla con Microsoft Access 2002



➤ FIGURA 2-5

Declaración de una relación con Microsoft Access 2002



porcionar más servicios y ayudar al analista si tiene conocimiento sobre qué CAPTAIN_ID en la tabla ITEM es una llave externa de CAPTAIN.

Los productos DBMS varían en la manera en que declaran esta condición. Con Microsoft Access la declaración se hace al establecer la relación entre la llave y la llave externa, como se muestra en la figura 2-5. CAPTAIN_ID de la tabla principal (CAPTAIN) está programada para igualarse con CAPTAIN_ID de la tabla relacionada (ITEM).

Una de las ventajas de establecer una relación al DBMS es que siempre que se coloquen las columnas de dos tablas en una forma, consulta, o reporte, el DBMS sabrá cómo relacionar los renglones de las tablas. Aunque esto puede establecerse cada vez para cada forma, consulta o reporte, hacerlo una vez ahorra tiempo y reduce la posibilidad de cometer errores. Por ahora, ignore otros elementos en la ventana de Edición de Relaciones (Edit Relationships). Usted aprenderá acerca de ellos a medida que avancemos. Una vez que se hayan definido las tablas, columnas y relaciones, el siguiente paso será construir los componentes de las aplicaciones.

➤ COMPONENTES DE APLICACIONES

Una aplicación de base de datos consta de formas, consultas, reportes, menúes y programas de aplicación. Como se muestra en la figura 2-1, las formas, consultas y reportes pueden ser definidos utilizando las herramientas que trae el DBMS. Los programas de aplicación deben escribirse ya sea en un lenguaje que sea parte del DBMS, o en uno estandarizado y conectado a la base de datos mediante el DBMS.

Formas

La figura 2-6 muestra tres presentaciones diferentes de los datos en las tablas CAPTAIN y ITEM. En la figura 2-6(a) se muestran los datos en una hoja de cálculo o formato tabular. El usuario puede hacer "clic" en el signo que está al inicio de cada renglón para desplegar los registros de ITEM que se relacionan con un renglón específico de CAPTAIN. Esto se ha realizado para los datos de Abernathy en el segundo renglón de esta figura. Observe que dos de los renglones de ITEM se relacionaron con Abernathy.

La figura 2-6(b) ilustra una segunda presentación mediante una forma de datos de entrada, la cual muestra los datos para un solo capitán a la vez. A los usuarios sin experiencia les parecerá más fácil de usar que el formato tabular.

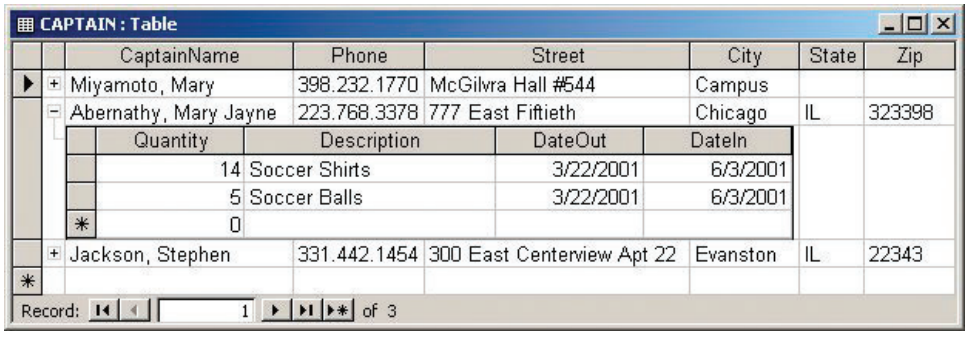
La página Captain Registration (Registro de Capitán) que se muestra en la figura 2-6(c) se puede utilizar en Internet o en la intranet de la universidad e ingresarla por medio del Internet Explorer de Microsoft. Tal uso requerirá que esté almacenada en un servidor Web, tal como el Servidor de Información de Internet. Aprenderá más sobre esto en los capítulos del 14 al 16. Por ahora, sólo observe que la herramienta de formas de Access 2002 puede utilizarse para crearlas.

Access 2002 genera automáticamente la forma tabular para cada tabla definida en la base de datos. Sin embargo, las formas de entrada de los datos se deben crear utilizando las herramientas para generarlas. La figura 2-7 muestra una manera de crearla. La fuente de datos de dicha forma ha sido programada para la tabla CAPTAIN (que no se muestra en la figura 2-7). Access muestra una ventana denominada *field list (lista de campo)*, la cual despliega las columnas de la tabla CAPTAIN. En esta figura, el usuario ha arrastrado el nombre del capitán de la lista de campos en la forma. Como respuesta, Access crea una etiqueta de control para CaptainName y un control del cuadro de texto que se utilizará para ingresar y desplegar los valores para CaptainName. En este punto, se dice que el cuadro de texto está *limitado a* la columna CaptainName de la tabla CAPTAIN. Otras columnas están limitadas de la misma manera; las de la tabla ITEM se limitan a la forma mediante un dispositivo llamado subforma. Access también tiene una herramienta experta para el diseño de formas, que se puede utilizar para crear algunas como la de la figura 2-6(b).

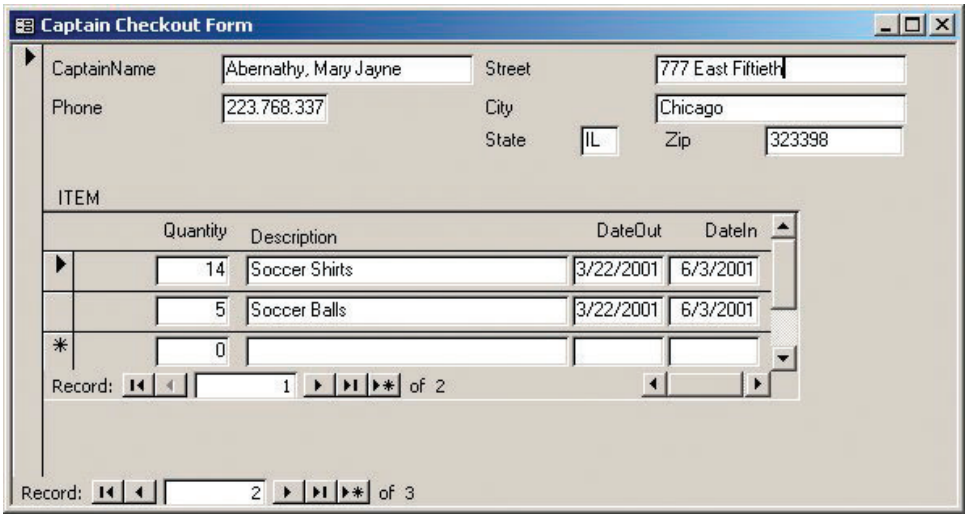
Ninguna de estas formas despliega las columnas CAPTAIN_ID o ITEM_ID. Dichos ID han estado escondidos del usuario. Sin embargo, tras bambalinas, el DBMS automá-

➤ FIGURA 2-6

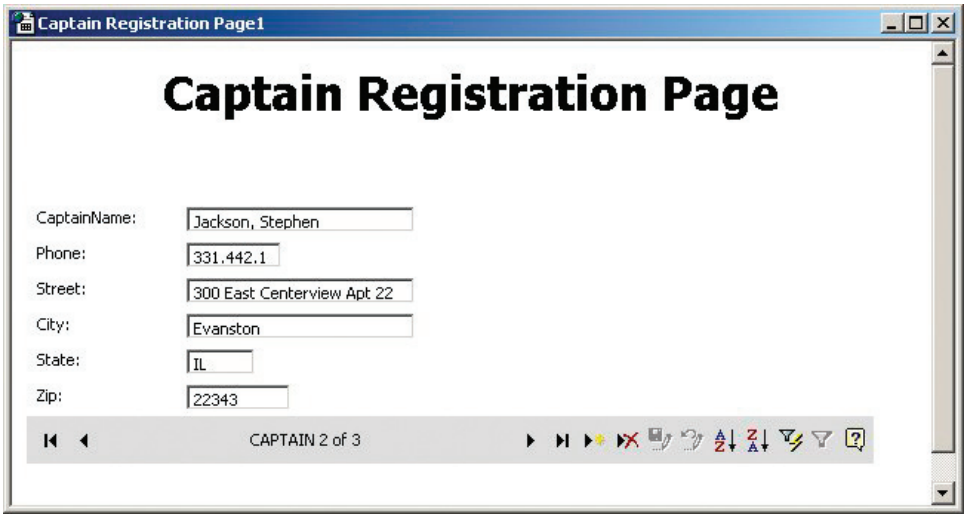
Presentaciones de los datos CAPTAIN y ITEM (CAPITÁN y ARTÍCULO)



(a) Forma tabular



(b) Forma de entrada de datos

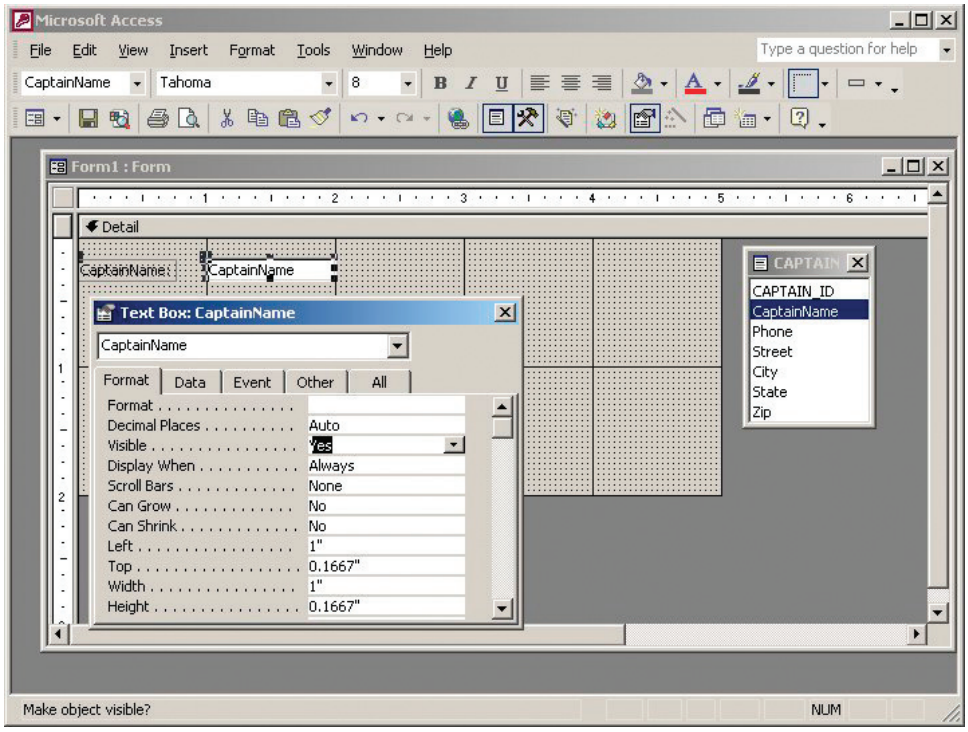


(c) Forma de entrada de búsqueda de datos

ticamente asigna nuevos valores cada vez que el usuario hace que se cree un renglón nuevo de CAPTAIN o de ITEM. Por lo tanto, cuando el usuario abre una forma en blanco, automáticamente el DBMS crea un renglón nuevo en CAPTAIN y asigna un valor a CAPTAIN_ID. Así, cuando el usuario agrega nuevos renglones ITEM para ese capitán, el DBMS crea nuevos valores de ITEM_ID para cada nuevo renglón ITEM, y coloca el valor actual de CAPTAIN_ID en la columna CAPTAIN_ID en el renglón ITEM. Examine

➤ FIGURA 2-7

Creación de una forma con Microsoft Access 2002



de nuevo la figura 2-4. La representación de los datos de ITEM_ID ha sido puesta en un AutoNumber (AutoNúmero). Esto instruye a Access para que asigne valores a ITEM_ID cuando se crean nuevos renglones. Cuando se creó la tabla CAPTAIN (la cual no se muestra), se estableció una programación similar a CAPTAIN_ID. Sin embargo, observe que para CAPTAIN_ID en la tabla ITEM no se estableció un AutoNumber. Esto se debe a que el valor de CAPTAIN_ID se creó al mismo tiempo que el renglón CAPTAIN; así que ese valor se copia en el campo CAPTAIN_ID en la tabla ITEM cuando se conecta un renglón ITEM a un capitán en particular.

¿Por qué se ocultan estos ID? La razón es que no tienen ningún significado para los usuarios. El Colegio Highline no asigna estos ID a los capitanes o a determinados artículos que son prestados. Si fuera así, entonces los ID se utilizarían y serían visibles. En su lugar, éstos han sido creados sólo para que cada renglón de cada tabla sea únicamente identificable para Access. Puesto que estos ID no significan nada para el usuario, están ocultos. Los identificadores como éste se llaman **llaves sustitutas.**

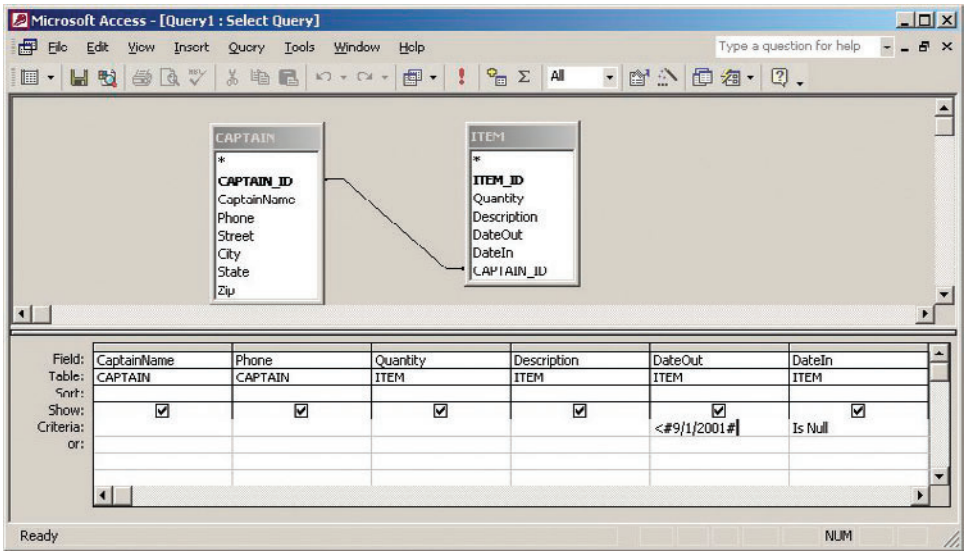
CONSULTAS

De vez en cuando los usuarios desean consultar los datos para responder consultas o identificar determinados problemas o situaciones. Por ejemplo, suponga que al inicio del semestre de otoño del 2001, uno de los usuarios desea saber si algún equipo que fue revisado antes del primero de septiembre del 2001, no se ha vuelto a revisar. De ser así, deseará conocer el nombre del capitán, su número telefónico y la cantidad y descripción de los artículos que tiene prestados.

Existen un sinfín de maneras para expresar dicha consulta. Una es utilizar el lenguaje de acceso a datos SQL, el cual se explica en el capítulo 9; otra es usar la **consulta mediante ejemplo (query by example, QBE).** La figura 2-8 muestra la creación de esta consulta por medio de QBE de Microsoft Access. Para crearla, el usuario coloca en la ventana de consulta los nombres de las tablas que serán consultadas. Esto ya se ha hecho en la sección anterior de la forma en que se muestra en la figura 2-8. Puesto que la relación entre CAPTAIN e ITEM ya ha sido definida para Access (en la figura 2-5), Access sabe que las dos tablas están unidas por CAPTAIN_ID, como lo muestra la línea dibujada entre CAPTAIN_ID en los dos cuadros de texto en la figura 2-8.

➤ FIGURA 2-8

Creación de una consulta con Microsoft Access 2002



Posteriormente, el creador de la consulta indica cuáles columnas de datos regresarán mediante la consulta. Con Access, esto se realiza cuando se arrastran los nombres de las columnas de los cuadros de texto en la cuadrícula y se posicionan en la entrada principal inferior de la forma. En la figura 2-8 las columnas CaptainName, Phone, Quantity, Description, DateOut, y DateIn se han colocado en la consulta. Los criterios de la consulta se especifican en el renglón llamado Criteria los cuales son los datos que deben estar antes de (<) 9/1/2001. (En Access, los datos están entre signos de número [#].) Asimismo, el valor de DateIn será nulo, lo que significa que no se ha especificado ningún valor para DateIn. El resultado de esta consulta sobre ejemplos de datos se muestra en la figura 2-9. Observe que todo el equipo que se muestra se prestó antes del 9/1/2001, como quedó establecido en la definición de consulta.

Con Access, y con la mayoría de los productos DBMS, las consultas pueden guardarse como parte de la aplicación, de manera que puedan regresarse cuando sea necesario. Además, se pueden parametrizar, lo cual significa que pueden construirse de manera que se acepten los valores de los criterios al mismo tiempo en que se ejecutan. Por ejemplo, la consulta en la figura 2-8 puede parametrizarse de modo que el usuario ingrese el valor de DateOut cuando la consulta se ejecute. Aparecerán cualesquiera de los artículos que fueron prestados antes de dicha fecha, pero que no se han vuelto a prestar.

Un tercer tipo de consulta, que es más fácil para los usuarios, se denomina **consulta por forma (query by form, QBF).** Con ésta, el usuario escribe las restricciones para la consulta en una forma de entrada de datos y presiona el botón de búsqueda. El DBMS encuentra todas las instancias que concuerdan con las restricciones específicas. Para la consulta del ejemplo, en el formato de la figura 2-6, el usuario ingresaría <#9/1/2001# en el campo DateOut e Is Null en el campo DateIn, y presionaría el botón de Query_by_Form. El DBMS encontraría los registros de todos los capitanes que coincidieran con las restricciones de la consulta. Query by form es una manera más

➤ FIGURA 2-9

Ejemplo del resultado de la consulta de la figura 2-8

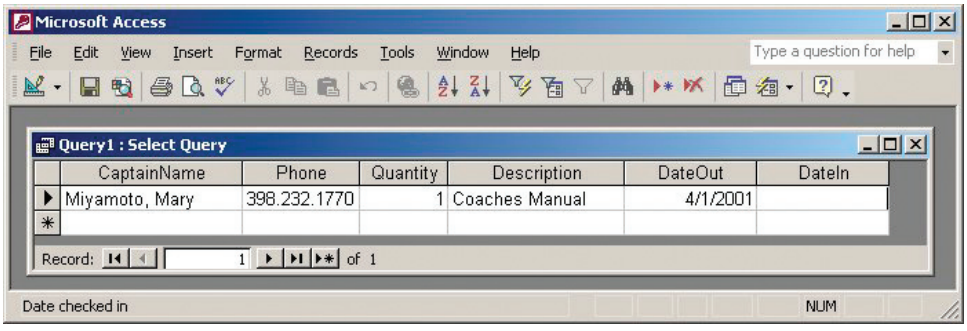


FIGURA 2-10

Ejemplo del reporte del Colegio Highline

Captain Equipment Report

CaptainName	*Miyamoto, Mary*		
Phone	*398.232.1770*		
DateOut	**Quantity**	**Description**	**DateIn**
4/1/2001	1	Coaches Manual	
4/1/2001	7	Soccer Balls	6/10/2001
4/10/2001	25	Blue Soccer Shirts	6/10/2001
CaptainName	*Abernathy, Mary Jayne*		
Phone	*223.768.3378*		
DateOut	**Quantity**	**Description**	**DateIn**
3/22/2001	5	Soccer Balls	6/3/2001
3/22/2001	14	Soccer Shirts	6/3/2001

nueva y moderna en comparación con la consulta por ejemplo, y es probable que la reemplace en muchas aplicaciones.

REPORTES

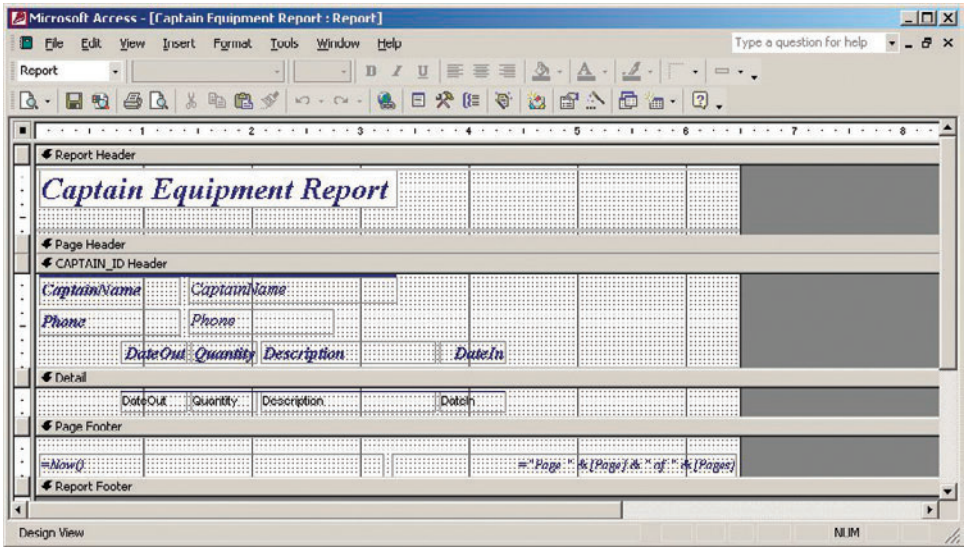
Un reporte es una representación con formato de la información que se encuentra en una base de datos. El reporte de la figura 2-10 contiene una sección para cada capitán de equipo y una lista de los artículos que se les han prestado a cada uno. Por ejemplo, los artículos a cargo del capitán "Miyamoto Mary" se muestran en la primera sección del reporte.

El desarrollo de un reporte es similar a preparar una forma de entrada de datos, aunque en ocasiones es más fácil debido a que un reporte sólo se utiliza para mostrar datos. Otras veces, la elaboración de un reporte es más difícil debido a que frecuentemente su estructura es más complicada que la de las formas.

La figura 2-11 muestra la definición de los reportes de Access 2002 para el reporte que se incluye en la figura 2-10. Éste es un ejemplo de un **escritor de reportes agrupados;** se llama así debido a que el reporte se divide en secciones o grupos. Como se

FIGURA 2-11

Desarrollo de un reporte con Microsoft Access 2002



muestra, existen grupos superiores, un grupo de detalle y grupos inferiores. El grupo superior muestra el nombre del reporte y el grupo inferior, la fecha en que se imprimió, el número de página y el número total de páginas. El encabezado de la página está vacío, pero el encabezado de CAPTAIN_ID tiene los datos y las etiquetas para la sección de detalles, la cual muestra los artículos prestados a un capitán.

Hay un problema cuando los dos campos de datos de este reporte están por separado. Para que sea eficaz, es mejor colocar DateOut entre Description y DateIn. Esto se puede hacer rápidamente arrastrando y posicionando las etiquetas y las cajas de texto, y es representativo del tipo de cambios que se realizan cuando se usan estas herramientas.

En general, los reportes pueden tener muchas secciones. En un ejemplo más complicado, es decir, matrícula curso-alumno, un reporte puede ser agrupado por COLEGIO, DEPARTAMENTO, CURSO y ESTUDIANTE. En este caso, el reporte tendría tres secciones de encabezados y una línea de detalles.

Menúes

Los menúes se utilizan para organizar componentes de la aplicación, de tal manera que sean más accesibles para el usuario final y proporcionen control sobre las actividades de éste. La figura 2-12 muestra un menú de ejemplo para la aplicación de Highline. La línea en la parte superior de la forma presenta las opciones de más alto nivel: File, Forms, Queries, Reports y Help (Archivo, Formas, Consultas, Reportes y Ayuda). Las letras subrayadas representan las teclas de acceso rápido. Si el usuario oprime <Alt> y la letra subrayada, se desplegará el submenú de opciones.

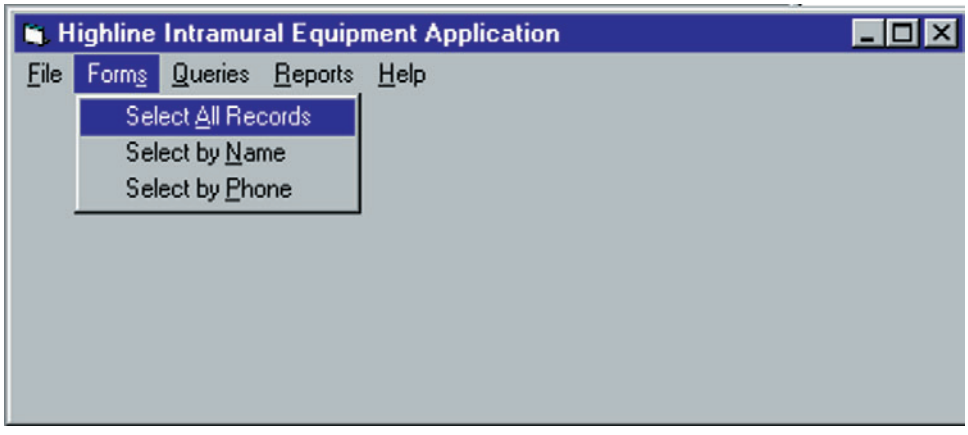
En la figura 2-12 el usuario ha tecleado <Alt> más la letra *S*. Las opciones del submenú son Select All Records, Select by Name y Select by Phone. De nuevo, las teclas de acceso rápido pueden utilizarse para seleccionar una de estas partes al teclear (Alt) y la letra subrayada de la opción. Los menúes hacen que la aplicación sea más accesible para el usuario mostrando cuáles son las opciones disponibles y ayudándole a seleccionar las que quiere realizar. Asimismo, se pueden utilizar los menúes para controlar el acceso del usuario a formas, reportes y programas. Algunas aplicaciones sacan ventaja de esto al cambiar dinámicamente las partes del menú después de que el usuario ingresa.

Programas de aplicación

El componente final de una aplicación de una base de datos radica en los programas de aplicación. Como se mencionó anteriormente, dichos programas pueden escribirse en un lenguaje específico para el DBMS o en un lenguaje estándar que se interconecta con

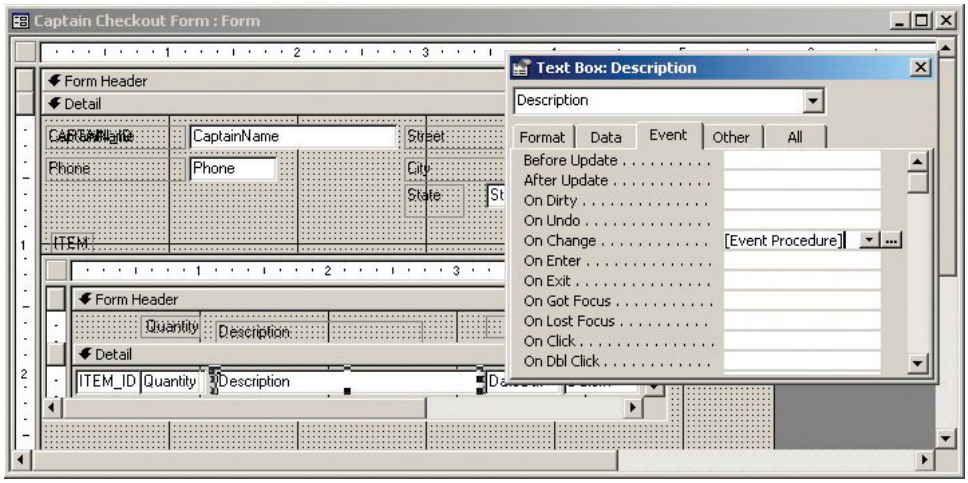
➤ FIGURA 2-12

Ejemplo de menú



➤ FIGURA 2-13

Captación de un evento en la forma de captura



el DBMS a través de un programa de interfaz predefinido. Aquí se utilizarán Microsoft Visual Basic y Access 2002.

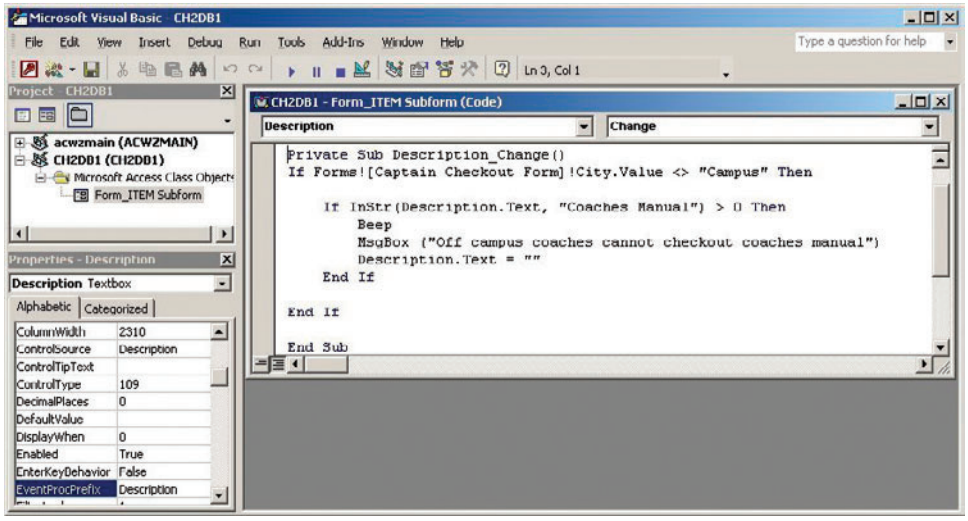
Suponga que Highline tiene una regla de negocio en la que los capitanes que viven fuera del campus no pueden tomar prestado el manual del entrenador. No se sabe la razón de esta regla; probablemente se deba a que contiene instrucciones secretas que proporcionan a los equipos de Highline una ventaja competitiva, y el colegio Highline está tratando de controlar la distribución de los manuales. Quizá son caros y si se pierden los estudiantes que viven en el campus tendrán que pagarlos. Cualquiera que sea la razón, Highline desea la aplicación de una base de datos para hacer que esta regla se cumpla.

Existen muchas maneras de hacerlo. Una de ellas, que se muestra en la figura 2-13, se refiere a captar el cambio en el cuadro Text Box: Description en la subforma de ITEM de Captain Checkout Form, e insertar un código para poner en vigor la regla cada vez que Description cambie. En la figura 2-13 el analista ha abierto las propiedades para el cuadro de texto de Description, y ha establecido On Change en [Event Procedure]. Cuando esto se llevó a cabo, Access abrió la ventana codificada que se muestra en la figura 2-14.

El analista escribió una sección de Visual Basic que se ejecutará cuando se presente el cambio. Primero, el código determina si el valor de City (Ciudad) no es igual a la condición "Campus". Si es así, entonces se requerirá la función InStr para determinar si los contenidos del cuadro de texto (Description.Text) contienen la condición de "Coaches Manual" (Manual del Entrenador). Si es así, se escucha un sonido, se genera un cuadro de mensaje y el cuadro de texto de Description se pone en blanco.

➤ FIGURA 2-14

Código de Visual Basic para poner en vigor la regla del negocio



Este código se ejecuta cada vez que el usuario cambia el valor del cuadro de texto de Description. Existen otros medios más eficaces de imponer esta regla, pero el punto aquí es sólo mostrarle cómo se puede integrar el código de aplicación con las formas de la base de datos.

Por supuesto, también es posible escribir un código que se ejecute independientemente de cualquier forma o reporte. Los programas pueden ser escritos para lectura y actualización de los datos de la base, de la misma manera en que pueden leer y actualizar otro tipo de archivos. Verá ejemplos de esto más adelante, especialmente en el capítulo 15 donde se muestra el uso de Microsoft ADO para leer y actualizar los datos de un Servidor Web, así como en el capítulo 16 en el que se utilizará Java con el mismo propósito.

➤ PROCESOS PARA EL DESARROLLO DE UNA BASE DE DATOS

Se han escrito volúmenes y volúmenes sobre el desarrollo de los sistemas de información en general, y de manera particular con respecto al desarrollo de las aplicaciones de una base de datos. Por consiguiente, no es necesario analizar a profundidad los procesos del desarrollo de sistemas; sin embargo, concluiremos este capítulo con un panorama de los procesos que se utilizan para desarrollar bases de datos y aplicaciones de éstas.

Estrategias generales

Una base de datos es un modelo del modelo de los usuarios referente a las actividades de su negocio. Por lo tanto, con el fin de construir una base de datos eficaz, así como las aplicaciones relacionadas, el equipo analista debe entender plenamente el modelo de los usuarios. Para hacerlo, el equipo construye un modelo de datos que identifica las cosas que se almacenarán en la base de datos y define su estructura y las relaciones entre éstas. Tal entendimiento debe existir antes que se realice el proceso de desarrollo, mediante entrevistas con los usuarios y la construcción de un documento de requerimientos. La mayoría de este tipo de esquemas incluyen el uso de **prototipos,** que son bases de datos y aplicaciones de ejemplos que representan diversos aspectos del sistema que será creado.

Existen dos estrategias generales para desarrollar una base de datos: arriba-abajo y abajo-arriba. El **desarrollo de lo general a lo particular** deriva de lo general a lo específico. Comienza con el estudio de las metas estratégicas de la organización, los medios por los cuales se pueden lograr estas metas, así como los requerimientos de información que se deben satisfacer para alcanzar las metas, y los sistemas que se requieren para proporcionar dicha información. A partir de tal estudio se construye un modelo de datos abstractos.

Mediante este modelo de alto nivel, el equipo analista trabaja progresivamente descendiendo hacia descripciones y modelos más y más detallados. Los modelos de nivel intermedio se expanden también con mayor detalle hasta que las bases de datos particulares y las aplicaciones pueden ser identificadas. Se selecciona entonces una o más de estas aplicaciones para el desarrollo. Con el tiempo, todo el modelo de datos de alto nivel se transforma en modelos de nivel inferior y se crean todos los sistemas indicados, las bases de datos y las aplicaciones.

El **desarrollo de lo particular a lo general** opera en el orden contrario de la abstracción al comenzar con la necesidad de desarrollar un sistema específico. Los medios de selección del primer sistema varían de una empresa a otra. En algunas, el comité directivo elige la aplicación; en otras, los usuarios son quienes la seleccionan; en cambio en otras, la voz con mayor autoridad en la oficina ejecutiva es la que tiene el poder de decisión.

Por cualquier medio, se selecciona un sistema específico para el desarrollo. El equipo analista obtiene entonces las declaraciones y los requerimientos al considerar las salidas y entradas de cualquier sistema con una base computacional, mediante el análisis de las formas y los reportes para los sistemas manuales, así como mediante la entrevis-

ta con los usuarios para descubrir sus necesidades respecto a nuevos reportes, formas, consultas y otros requerimientos. A partir de todo esto, el equipo desarrolla el sistema de información. Si éste involucra una base de datos, el equipo utiliza las especificaciones de los requerimientos para construir un modelo de datos, y a partir de éste diseña e implementa la base de datos. Cuando se concluye este sistema, comienzan otros proyectos con el fin de construir sistemas de información adicionales.

Los partidarios del enfoque de lo general a lo particular afirman que es superior al enfoque que va de lo particular a lo general, debido a que los modelos de datos (y sistemas subsecuentes) se construyen con una perspectiva global. Aseguran que dichos sistemas poseen mejores interfaces que ningún otro, que son más consistentes y requieren mucho menos repetición de trabajo y modificaciones.

Los partidarios del enfoque de lo particular a lo general sostienen que es superior al enfoque de lo general a lo particular debido a que es más rápido y menos riesgoso. Afirman que el modelo de lo general a lo particular da como resultado muchos estudios que son difíciles de completar y que el proceso de planeación a menudo resulta en la obstrucción del análisis. Aunque el modelo de lo particular a lo general no necesariamente da como resultado el mejor conjunto de sistemas, sí produce rápidamente sistemas útiles. Los beneficios de éstos comienzan a ser más rápidos que con el modelo de lo general a lo particular y pueden compensar mucho más en cualquier repetición o modificación que se necesite realizar para ajustar el sistema a una perspectiva global.

Este texto explica las herramientas y técnicas que se pueden utilizar con cualquier estilo de desarrollo de sistemas. Por ejemplo, aunque el modelo entidad-relación (capítulo 3) y el modelo de objeto semántico (capítulo 4) trabajan tanto con el desarrollo de lo general a lo particular como con el de lo particular a lo general, el enfoque entidad-relación es particularmente eficaz con el desarrollo de lo general a lo particular, y el enfoque de objeto semántico es especialmente eficaz con el desarrollo de lo particular a lo general.

Modelación de datos

Como ya establecimos, la meta más importante de la fase de requerimientos es la creación de un modelo de datos de los usuarios. Ya sea que se realice en el estilo de lo general a lo particular o en el de lo particular a lo general, involucra la entrevista con los usuarios, documentación de los requerimientos y, a partir de éstos, la construcción del modelo de datos y prototipos. Dicho modelo identifica lo que se almacenará en la base de datos y define su estructura e interrelación.

Por ejemplo, considere la figura 2-15(a) en la que se muestra una lista de órdenes que hizo un vendedor durante un periodo específico. Para que este reporte se produzca mediante la aplicación de una base de datos, ésta debe contener los datos mostrados; por ende, los analistas de la base de datos necesitan examinar el reporte y trabajar en los datos que deben almacenarse en la base. En este caso, deben existir datos con respecto a los vendedores (nombre y región) y acerca de los pedidos u órdenes (empresa, fecha del pedido y cantidad).

El desarrollo de una base de datos es complicado por el hecho de que no sólo existe un requerimiento sino muchos, los cuales a menudo se traslapan. El reporte en la figura 2-15(b) se refiere también a vendedores, pero, en lugar de pedidos, emite cheques de comisión. A partir de este reporte se puede suponer que existen diferentes tipos de pedidos y que cada uno tiene una tasa de comisión diferente.

Los pedidos que están en el reporte de la figura 2-15(b) se relacionan de alguna manera con los de la figura 2-15(a), pero el modo en que lo hacen no es del todo claro. El equipo analista debe determinar esta relación mediante la inducción de reportes y formas, entrevistas con los usuarios, conocimiento del equipo en cuestión, así como otras fuentes.

MODELACIÓN DE DATOS BASADA EN REFERENCIAS. Cuando los usuarios dicen que necesitan formas y reportes con datos y estructuras específicas, su petición implica un modelo basado en las cosas que forman parte de su mundo. Sin embargo, no pueden describir exactamente qué modelo es ése. Si un analista le preguntara a un

➤ FIGURA 2-15

Ejemplos de dos reportes relacionados: (a) ejemplo del reporte de VENTAS y (b) ejemplo del reporte de COMISIONES

Lista de pedidos del vendedor
03-Oct-2001

Nombre	Región	NombredelaCompañía	FechadelPedido	Cantidad
Kevin Dougherty	**Oeste**			
		Cabo Controls	9/12/2001	$2,349.88
				$2,349.88
Mary B. Wu	**Oeste**			
		Ajax Electric	9/17/2001	$23,445.00
		American Maxell	9/24/2001	$17,339.44
				$40,784.44
			Gran total:	**$43,134.32**

(a)

Reporte del cheque de comisión del vendedor
03-Oct-2001

Nombre	NúmeroLocal	FechadelCheque	TipodeCheque	CantidaddelCheque
Kevin Dougherty	**232-9988**			
		9/30/2001	XZ	$487.38
				$487.38
Mary B. Wu	**232-9987**			
		9/30/2001	C	$237.44
		9/30/2001	A	$1,785.39
				$2,022.83
			Gran total:	**$2,510.21**

(b)

usuario común y corriente: "¿Cuál es su estructura mental del modelo de datos con respecto a los vendedores?", el usuario, en el mejor de los casos, se sorprendería y estaría alerta; esto se debe a que muchos usuarios no piensan de esta manera.

En lugar de lo anterior, los analistas deben inferir, a partir de las declaraciones de los usuarios sobre las formas y los reportes, la estructura y las relaciones de las cosas que se almacenarán en la base de datos. Deberán entonces registrar esas inferencias en un modelo de datos que se transforme en el diseño de una base de datos, y dicho diseño se implementará por medio de un DBMS. Así es como se construyen las aplicaciones que producen reportes y formas para los usuarios.

La construcción de un modelo de datos es un proceso por inferencia. Los reportes y las formas son como las sombras: se proyectan en un muro. Los usuarios pueden describir las sombras, pero no las formas que dan origen a esas sombras. De ahí que los analistas deban deducir, trabajar en diferentes sentidos, revertir o replantear la ingeniería de las estructuras y de las relaciones de dichas formas, a partir de las sombras.

Por desgracia, este proceso por inferencia es más arte que ciencia. Es posible aprenderse las herramientas y las técnicas para la modelación de datos; de hecho dichas herra-

mientas y técnicas son el tema de los dos siguientes capítulos, pero el uso de éstas es un arte que requiere de la experiencia guiada por la intuición.

La calidad del modelo es importante. Si el modelo de datos documentados refleja exactamente el modelo de datos que hay en la mente de los usuarios, existe una excelente oportunidad de que las aplicaciones que resulten se acerquen más a las necesidades de los usuarios. Pero si el modelo de datos documentados refleja incorrectamente dicho modelo, es poco probable que la aplicación se acerque a lo que ellos realmente quieren.

MODELADO EN SISTEMAS MULTIUSUARIO. El proceso de modelación de datos se vuelve mucho más complejo en el caso de un grupo de trabajo multiusuarios y bases de datos organizacionales, debido a que muchos usuarios pueden imaginarse diferentes modelos de datos. Ocasionalmente éstos son inconsistentes, aunque la mayoría de las veces dichas inconsistencias pueden solucionarse. Por ejemplo, los usuarios pueden emplear el mismo término para diferentes cosas, o diferentes términos para las mismas cosas.

Sin embargo, a veces las diferencias no pueden ser conciliadas. En esos casos, el analista de la base de datos debe documentar las diferencias y ayudar a los usuarios a resolverlas, y esto por lo general significa que ciertas personas deben cambiar la manera en la cual ven al mundo que les rodea.

Un reto aún mayor está en los grandes sistemas, en los que ningún usuario tiene un modelo de la estructura completa. Cada usuario comprende parte del modelo de datos del grupo de trabajo o de la organización, pero ninguno lo entiende completamente. En tales casos, la base de datos se convierte en la unión lógica de las piezas del modelo del grupo de trabajo o de la organización, y los analistas deben documentar esa unión lógica en el modelo de datos. Lo anterior puede ser muy difícil.

CONFUSIÓN ACERCA DEL TÉRMINO *MODELO*. Los capítulos 3 y 4 presentan dos herramientas alternativas para la construcción de modelos de datos: el de entidad-relación y el modelo de objeto semántico. Ambos son estructuras que describen y documentan los requerimientos de los datos de los usuarios. Para evitar confusión, observe los diferentes usos del término *modelo*. El equipo analiza los requerimientos y construye un *modelo de los datos de los usuarios*, o un *modelo de los datos de los requerimientos*. Este modelo es una representación de la estructura y de las relaciones de lo que se necesita en la base de datos para sustentar los requerimientos de los usuarios. Con el fin de crear el modelo de datos de los usuarios, el equipo analista utiliza herramientas denominadas modelos de datos entidad-relación y de objeto semántico, las cuales consisten en estándares de lenguaje y diagramas para representar el modelo de los datos de los usuarios. Su papel en el desarrollo de la base de datos es similar al del diagrama de flujos y del pseudocódigo en la programación.

➤ RESUMEN

Los componentes de un sistema de una base de datos son la propia base de datos, el DBMS y los programas de aplicación, los cuales los utilizan analistas y usuarios. Una base de datos contiene datos, metadatos, índices y metadatos de aplicación. Actualmente la mayoría de las bases representan a los datos como relaciones o tablas, aunque no todas las relaciones son igualmente deseables. Las relaciones indeseables pueden mejorarse a través de un proceso denominado normalización. Los metadatos a menudo son almacenados en tablas llamadas tablas de sistema.

Las funciones y características de un DBMS pueden agruparse en tres subsistemas. El subsistema de las herramientas de diseño define la base de datos y la estructura de las aplicaciones, o de los componentes de éstas. Las funciones del subsistema run-time son la materialización de las formas, reportes y consultas mediante la lectura y escritura de los datos de una base. El motor DBMS es el intermediario entre los otros dos subsistemas y el sistema operativo. Recibe requerimientos establecidos en términos de tablas, renglones y columnas y los traduce en lectura y escritura.

Un esquema es una descripción de la estructura de una base de datos e incluye descripciones de tablas, relaciones, dominios y reglas del negocio. Los renglones de una tabla pueden relacionarse con los de otras. Este capítulo ilustró una relación 1:N entre los renglones de las tablas; asimismo, existen otros tipos de relaciones, como veremos en el siguiente capítulo.

Un dominio es un conjunto de los valores que una columna puede tener. Se debe especificar un dominio para cada columna de cada tabla.

Por último, las reglas del negocio son restricciones de las actividades del negocio que deben reflejarse en la base de datos y en las aplicaciones de ésta.

Las facilidades del DBMS se utilizan para crear estructuras de tablas, definir relaciones y crear formas, consultas, reportes y menúes. Los productos DBMS también incluyen dispositivos para interactuar con programas de aplicación por escrito, ya sea en lenguajes específicos del DBMS o en lenguajes estándares como Java.

Puesto que la base de datos es un modelo del modelo del negocio de los usuarios, el desarrollo de una base de datos empieza con el aprendizaje y registro de dicho modelo. Algunas veces esto se expresa en prototipos de la aplicación, o en los componentes de la aplicación que será construida.

Los dos estilos generales de desarrollo son de lo general a lo particular, que deriva de lo general a lo específico, y el desarrollo de lo particular a lo general, que procede de lo específico a lo general. Con el desarrollo de lo general a lo particular las aplicaciones se llevan a cabo desde una perspectiva global; con el de lo particular a lo general, las aplicaciones se desarrollan más rápido. Algunas veces se utiliza una combinación de ambos enfoques.

Los modelos de datos se construyen mediante un proceso que implica la inferencia a partir de las declaraciones de los usuarios. Se reúnen las formas, los reportes y las consultas, y los analistas trabajan en dirección inversa para deducir las estructuras que los usuarios se imaginan. Esto es necesario debido a que la mayoría de los usuarios no pueden describir exactamente sus modelos de datos. La modelación de datos puede ser especialmente difícil y desafiante en las aplicaciones multiusuarios, en las cuales los propósitos de los usuarios pueden contraponerse y por lo tanto ninguno puede visualizar el propósito integral de la actividad del negocio.

El término *modelo de datos* se utiliza de dos maneras: para referirse a un modelo del propósito de los usuarios con respecto a sus datos, y para denotar las herramientas que se utilizan con el fin de definir el propósito de los usuarios en cuanto a sus datos.

➤ PREGUNTAS DEL GRUPO I

2.1 Nombre los componentes principales del sistema de una base de datos y explique brevemente la función de cada uno.

2.2 Dé un ejemplo, aparte del ya mencionado en este capítulo, sobre una relación que tienda a presentar problemas cuando se actualiza. Utilice la relación R1 como base, referencia.

2.3 Transforme la relación de su respuesta en la pregunta 2.2 en dos o más relaciones que no tengan problemas de actualización. Utilice las relaciones R2 y R3 como ejemplos.

2.4 Explique las funciones de los metadatos y de las tablas de sistema.

2.5 ¿Cuál es la función de los índices? ¿En qué momento son deseables y cuál es su costo?

2.6 ¿Cuál es la función de los metadatos de aplicación? ¿Cómo difieren de los metadatos?

2.7 Explique las características y funciones del subsistema de las herramientas de diseño de un DBMS.

2.8 Describa las características y funciones del subsistema run-time de un DBMS.

2.9 Explique las características y funciones del motor DBMS.

2.10 ¿De qué consta el esquema de una base de datos? Enumere sus componentes.

2.11 ¿Cómo se representan las relaciones en el diseño de una base de datos relacional? Proporcione un ejemplo sobre dos tablas con una relación 1:N y explique cómo se expresa la relación en los datos.

2.12 ¿Qué es un dominio y por qué es importante?

2.13 ¿Qué son las reglas del negocio? Dé un ejemplo sobre posibles reglas del negocio para las relaciones en su respuesta a la pregunta 2.11.

2.14 ¿Qué es una llave externa? ¿Cuáles columnas en su respuesta a la pregunta 2.11 representan una llave externa?

2.15 Explique el propósito de las formas, consultas y menúes.

2.16 Explique la diferencia entre la consulta mediante ejemplo y la consulta por forma (QBF).

2.17 ¿Cuál es la primera tarea importante en el desarrollo de una base de datos y aplicaciones relacionadas?

2.18 ¿Cuál es la función de un prototipo?

2.19 Describa el desarrollo de lo general a lo particular. ¿Cuáles son sus ventajas y desventajas?

2.20 Describa el desarrollo de lo particular a lo general. ¿Cuáles son sus ventajas y desventajas?

2.21 Explique los dos significados diferentes del término *modelo de datos*.

➤ PREGUNTAS DEL GRUPO II

2.22 Implemente una base de datos con las relaciones CAPTAIN e ITEM, en cualquier DBMS al que pueda tener acceso. Utilice uno de los productos DBMS para ingresar los datos a cada una de estas relaciones. Cree y procese una consulta para utilizar el dispositivo del DBMS y procesar una consulta que identifique aquellos artículos que fueron revisados antes del 1 de septiembre de 2001, así como los que aún no han sido revisados. Imprima el nombre del capitán, su teléfono y la cantidad y descripción de cualesquiera de dichos artículos.

2.23 Entreviste a un analista profesional con respecto a la aplicación de una base de datos, e investigue el proceso que él utiliza para desarrollarla. ¿Ese desarrollo es de lo general a lo particular, de lo particular a lo general, o usa alguna otra estrategia? ¿Cómo construye modelos de datos este analista y con qué herramientas? ¿Cuáles son los problemas principales que por lo general surgen al desarrollar una base de datos?

2.24 Considere la declaración: "Una base de datos es un modelo del modelo de la realidad de los usuarios." ¿En qué difiere de la declaración: "Una base de datos es un modelo de la realidad"? Suponga que dos analistas están en desacuerdo sobre un modelo de datos, y uno de ellos afirma: "Mi modelo es una mejor representación de la realidad." ¿Qué quiere decir esa persona? ¿Qué diferencias podrían surgir si un analista cree más en la primera declaración que en la segunda?

➤ PREGUNTAS DEL PROYECTO FIREDUP

Considere la situación de la compañía FiredUp, cuyo caso se presentó al final del capítulo 1. Cada una de las estufas viene con un formato adjunto de registro del producto que incluye los siguientes datos:

NombredelComprador, Dirección, NúmerodeDepartamento, Ciudad, Estado/Provincia, CódigoPostal, País, CorreoElectrónico, NúmeroTelefónico, FechadeCompra, y NúmerodeSerie

Suponga que FiredUp decide crear una base de datos con las siguientes tablas:

CLIENTE (Nombre, Dirección, NúmerodeDepartamento, Ciudad, Estado/Provincia, CódigoPostal, País, CorreoElectrónico y NúmeroTelefónico)

y

COMPRA (FechadeCompra y NúmerodeSerie)

A. Construya una tabla de datos a manera de ejemplo que esté basada en la estructura de CLIENTE. Incluya por lo menos cuatro renglones en su tabla. Para las consultas de la A a la G sólo enumere los datos utilizando un procesador de texto.

B. ¿Cuál de las columnas de la tabla CLIENTE puede utilizarse para identificar un renglón único de la tabla? Tal columna a veces se denomina *llave primaria*, como lo aprenderá posteriormente en este texto.

C. Construya una tabla de datos acorde a la estructura de COMPRA. Incluya por lo menos cuatro renglones en su tabla.

D. ¿Cuáles de las columnas de la tabla COMPRA pueden utilizarse como la llave primaria de COMPRA?

E. Cuando se utilizan las tablas anteriormente definidas no hay manera de relacionar a un cliente en particular con su estufa. Una manera de hacerlo sería añadir NúmerodeSerie de COMPRA a CLIENTE. La tabla CLIENTE aparecerá entonces como:

CLIENTE (Nombre, Dirección, NúmerodeDepartamento, Ciudad, Estado/Provincia, CódigoPostal, País, CorreoElectrónico, NúmeroTelefónico y NúmerodeSerie)

Copie los datos de su ejemplo CLIENTE y agréguele la columna NúmerodeSerie. Nombre a esta nueva tabla CLIENTE1.

F. Una técnica alternativa para representar la relación de dos tablas sería colocar CorreoElectrónico de CLIENTE en COMPRA. La tabla COMPRA aparecerá entonces como:

COMPRA (FechadeCompra, NúmerodeSerie, CorreoElectrónico) Copie su ejemplo Datos de COMPRA y agréguele la columna CorreoElectrónico. Nombre a esta nueva tabla COMPRA1.

G. Ahora tiene tres posibles estructuras de bases de datos.

BD1: CLIENTE1 con COMPRA

BD2: CLIENTE con COMPRA1, y

BD3: CLIENTE1 con COMPRA1

¿En qué circunstancias recomendaría utilizar la estructura BD1?
¿Cuándo recomendaría específicamente el uso de la estructura BD2?

H. ¿En qué casos aconsejaría utilizar la estructura BD3?

PARTE II

MODELACIÓN DE DATOS

La modelación de datos es el proceso de crear una representación lógica de la estructura de una base de datos. Para que sea correcta, la modelación debe considerar todas las perspectivas de los usuarios acerca de los datos. La modelación es la tarea más importante en el desarrollo de las bases y sus aplicaciones. Si la modelación de datos representa erróneamente los puntos de vista del usuario, las aplicaciones le resultarán muy difíciles de usar, incompletas y se sentirá frustrado. La modelación de datos es la base de todo el trabajo posterior del desarrollo de bases de datos y sus aplicaciones.

La parte II describe dos herramientas diferentes de la modelación de datos; el capítulo 3 considera el modelo entidad-relación (E-R), que tiene una gran cantidad de seguidores entre los profesionales de las bases de datos, y en el capítulo 4 se describe el modelo del objeto semántico, que tiene menos adeptos pero se le considera más completo y fácil de usar que el modelo E-R.

Ambos modelos proporcionan un lenguaje que expresa la estructura de los datos y las relaciones en el medio ambiente del usuario. La modelación de datos expresa un diseño lógico de datos de la misma manera en que un diagrama de flujo expresa el diseño lógico de un programa.

CAPÍTULO 3

El modelo entidad-relación

Este capítulo describe e ilustra el uso del **modelo entidad-relación (modelo E-R),** que presentó Peter Chen en 1976.[1] En su artículo, Chen estableció los fundamentos del modelo, los cuales a partir de entonces ha ampliado y modificado él mismo, así como muchos otros.[2,3] Además, el modelo E-R ha formado parte de varias herramientas CASE, que lo han modificado. Actualmente no existe un solo modelo estándar E-R que sea comúnmente aceptado, sino que hay una serie de construcciones comunes de las cuales derivan la mayoría de las variantes E-R. Este capítulo describe estas construcciones comunes y muestra cómo se utilizan. Los símbolos que se emplean para expresar el modelo E-R difieren considerablemente. No sólo analizaremos los símbolos tradicionales, sino también aquellos que se usan en el Lenguaje de modelación unificado (UML, por sus siglas en inglés), el cual constituye una herramienta de diseño que se está expandiendo rápidamente entre los programadores orientados a objetos, y que incorpora al modelo E-R.

➤ ELEMENTOS DEL MODELO ENTIDAD-RELACIÓN

Los elementos clave del modelo E-R son entidades, atributos, identificadores y relaciones. Consideremos cada uno de éstos.

[1] P. P. Chen, "The Entity-Relationship Model—Towards a Unified View of Data", *ACM Transactions on Database Systems*, enero de 1976, pp. 9-36.
[2] T. J. Teorey, D. Yang, y J. P. Fry, "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", *ACM Computing Surveys*, junio de 1986, pp. 197-222.
[3] Thomas A. Bruce, *Designing Quality Databases with IDEF1X Information Models* (Nueva York; Casa de Publicidad Dorset, 1992).

Entidades

Una **entidad** es algo que se puede identificar en el medio ambiente de trabajo de los usuarios; es decir, aquello a lo cual los usuarios quieren dar seguimiento. Algunos ejemplos de entidades son EMPLEADO (EMPLOYEE) Mary Doe, CLIENTE 12345, ORDEN-VENTA (SALES-ORDER) 1000, VENDEDOR (SALESPERSON) John Smith, y PRODUCTO (PRODUCT) A4200. Las entidades de determinado tipo se agrupan en **clases de entidades.** Así, la clase de entidad EMPLEADO es un conjunto de todas las entidades EMPLEADO. En este texto, las clases de entidades están impresas con letras mayúsculas.

Es importante comprender las diferencias entre una clase de entidad y una instancia de entidad. Una *clase de entidad* es un conjunto de entidades y se describe mediante la estructura o formato de las entidades en esa clase. Una *instancia* de entidad es la representación de una entidad en particular, tal como CLIENTE 12345; se describe mediante los valores de los atributos de ésta. Por lo general existen muchas instancias de entidad en una clase de entidad. Por ejemplo, dentro de la clase CLIENTE, existen muchas instancias, una para cada cliente representado en la base de datos. Una clase de entidad y dos de sus instancias se muestran en la figura 3-1.

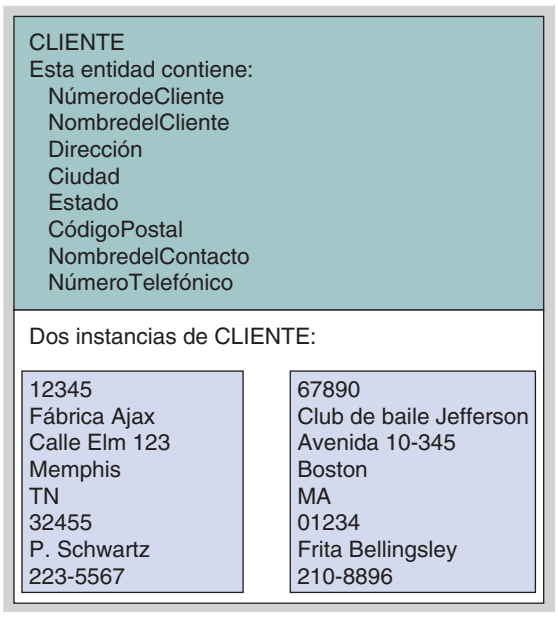
Atributos

Las entidades tienen **atributos,** o **propiedades,** como a veces se les llama, que describen las características de la entidad. Algunos ejemplos de atributos son: NombredelEmpleado, FechadeContrato y CódigodeAptitudesdeTrabajo. En este texto, los atributos están impresos en mayúsculas y minúsculas. El modelo E-R supone que todas las instancias de cierta clase de entidad tienen los mismos atributos.

La definición original del modelo E-R incluye atributos multivalor y compuesto. Un ejemplo de un atributo compuesto es la Dirección, la cual consta del grupo de atributos {Calle, Ciudad, Estado/Provincia, CP}. Un ejemplo de un atributo multivalor es NombredelContacto en CLIENTE, donde más de un nombre de una persona está asociado con un cliente específico. Un atributo puede ser tanto multivalor como compuesto; por ejemplo, el atributo compuesto Teléfono {CódigodeÁrea, Número} podría ser multivalor para permitir una gran cantidad de números telefónicos. Muchas implementaciones del modelo E-R ignoran los atributos compuestos de un solo valor. Requieren atributos multivalor (compuestos o no) para ser transformadas en entidades, como se muestra a continuación.

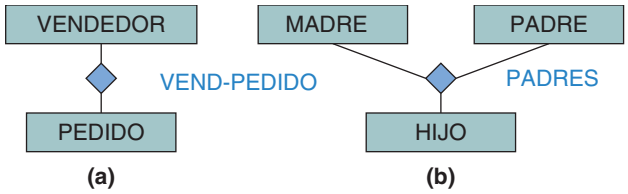
➤ FIGURA 3-1

CLIENTE: un ejemplo de entidad



➤ FIGURA 3-2

Relaciones de grados diferentes: (a) ejemplo de relación de grado 2, y (b) ejemplo de relación de grado 3



Identificadores

Las instancias de una entidad tienen **identificadores,** los cuales son atributos que nombran, o identifican, las instancias de una entidad. Por ejemplo, las instancias EMPLEADO podrían ser identificadas mediante el NúmerodeSeguroSocial (SocialSecurityNumber), el NúmerodeEmpleado (EmployeeNumber), o el NombredelEmpleado (EmployeeName). Las instancias EMPLEADO no parecen ser identificadas por atributos tales como Salario o FechadelContrato, porque normalmente no se usan en una función de selección. De igual manera, los CLIENTEs podrían ser identificados por NúmerodeCliente o NombredelCliente, y PEDIDOs por el NúmerodePedido.

El identificador de una instancia de entidad consta de uno o más de los atributos de ésta. Un identificador puede ser **único** o **no único.** Si es único, su valor identificará solamente un ejemplo de entidad. Si no lo es, el valor identificará una serie de instancias. El NúmerodeEmpleado es más parecido a un identificador único, mientras que NombredelEmpleado es similar a un identificador no único (por ejemplo, puede haber varios John Smith).

Los identificadores que constan de dos o más atributos se llaman **identificadores compuestos.** Algunos ejemplos son {CódigodeÁrea, NúmeroLocal}, {NombredelProyecto, NombredelaTarea}, y {Nombre, Apellido, ExtensiónTelefónica}.

Relaciones

Las entidades pueden asociarse con otras mediante **relaciones.** El modelo E-R contiene tanto clases de relaciones como instancias de relaciones.[4] Las *clases de relaciones* son asociaciones entre las clases de entidad, y las *instancias de relaciones* son asociaciones entre las instancias de entidad. Las relaciones pueden tener atributos.

Una clase de relación puede involucrar muchas clases de entidades. El número de éstas en la relación es el **grado** de ésta. En la figura 3-2(a) la relación VEND-PEDIDO es de grado 2 porque involucra dos clases de entidades: VENDEDOR y PEDIDO. La relación PADRES en la figura 3-2(b) es de grado 3, porque involucra tres clases de entidades: MADRE, PADRE e HIJO. Las relaciones de grado 2 son muy comunes y con frecuencia se denominan **relaciones binarias.**

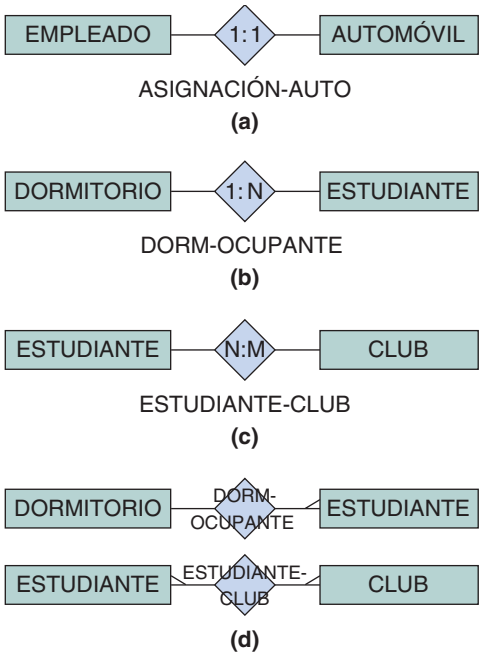
TRES TIPOS DE RELACIONES BINARIAS. La figura 3-3 muestra tres tipos de relaciones binarias. En una relación 1:1 (léase "uno a uno"), una instancia de entidad de un tipo se relaciona con una instancia de una sola entidad de otro tipo. En la figura 3-3(a), la relación de ASIGNACIÓN-AUTO (AUTO-ASSIGNMENT) asocia a un único EMPLEADO con un solo AUTO. De acuerdo con este diagrama, ningún empleado tiene más de un automóvil asignado y ningún automóvil se asigna a más de un empleado.

La figura 3-3(b) muestra el segundo tipo de relación, 1:N (léase "uno a N" o "uno a muchos"). En esta relación, denominada DORM-OCUPANTE (DORM-OCCUPANT),

[4] Con el fin de abreviar, algunas veces eliminaremos la palabra *instancia* cuando el contexto indique claramente que se refiere a una instancia, en lugar de a una clase de entidad.

➤ FIGURA 3-3

Tres tipos de relaciones binarias: (a) relación binaria 1:1, (b) relación binaria 1:N, (c) relación binaria N:M, y (d) representación de relación con pata de gallo



una sola instancia de DORMITORIO relaciona a muchas instancias de ESTUDIANTE. De acuerdo con este esquema, un dormitorio tiene muchos estudiantes, pero un estudiante tiene sólo un dormitorio.

Las posiciones del 1 y de la N son significativas. El 1 está cerca de la línea que conecta con DORMITORIO, lo cual significa que el 1 se refiere al lado DORMITORIO de la relación, y la N está cerca de la línea que conecta con ESTUDIANTE, lo cual significa que la N se refiere a la parte ESTUDIANTE de la relación. Si el 1 y la N se invirtieran y la relación se escribiera N:1, un DORMITORIO tendría un ESTUDIANTE y un ESTUDIANTE tendría muchos DORMITORIOS. Éste, por supuesto, no es el caso.

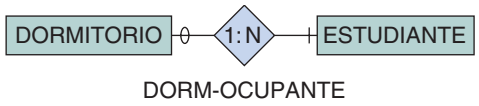
La figura 3-3(c) muestra el tercer tipo de relación binaria, N:M (léase "N a M" o "muchos a muchos"). Esta relación se llama ESTUDIANTE-CLUB y relaciona las instancias de ESTUDIANTE con las de CLUB. Un estudiante puede reunirse en más de un club, y un club puede tener muchos estudiantes.

Los números dentro del diamante de la relación muestran la cantidad máxima de entidades que puede haber en un lado de la relación. Estas restricciones se llaman **cardinalidad máxima** de la relación. La relación en la figura 3-3(b), por ejemplo, se dice que tiene cardinalidad máxima de 1:N. Pero las cardinalidades no están limitadas a los valores que se muestran aquí. Es posible, por ejemplo, que la cardinalidad máxima sea distinta de 1 y N. La relación entre EQUIPO-DE-BÁSQUETBOL y JUGADOR, por ejemplo, podría ser 1:5, lo cual indicaría que un equipo de básquetbol tiene más de cinco jugadores.

Las relaciones de los tipos que se muestran en la figura 3-3 a veces se denominan **relaciones TIENE-UN.** Este término se utiliza porque una entidad *tiene una* relación con otra entidad. Por ejemplo, un EMPLEADO tiene un AUTO; un ESTUDIANTE tiene un DORMITORIO; y un CLUB tiene ESTUDIANTEs.

DIAGRAMAS DE ENTIDAD-RELACIÓN. Las representaciones de la figura 3-3 se llaman **diagramas entidad-relación** o **diagramas E-R.** Éstos, aunque están estandarizados, se construyen libremente. De acuerdo con este estándar, las clases de entidad se muestran con rectángulos; las relaciones, con diamantes, y la cardinalidad máxima

➤ FIGURA 3-4



Relación que muestra la cardinalidad mínima

de la relación se indica dentro del diamante;[5] el nombre de la entidad, dentro del rectángulo, y el nombre de la relación está cerca del diamante.

Aunque en algunos diagramas E-R el nombre de la relación está dentro del diamante, esto puede hacer que el diagrama parezca difícil, ya que los diamantes pueden ser grandes y estar fuera de escala con el fin de incluir el nombre de la relación. Para evitar esto, los nombres de las relaciones a veces se escriben sobre el diamante. Cuando el nombre se coloca dentro o en la punta del diamante, la cardinalidad de la relación se muestra por medio de "patas de gallo" en las líneas que se conectan a la(s) entidad(es) en la mayoría de los lados de la relación. En la figura 3-3(d) se muestran las relaciones DORM-OCUPANTE y ESTUDIANTE-CLUB con esas "patas de gallo".

Como ya lo establecimos, la cardinalidad máxima indica el número máximo de entidades que pueden estar involucradas en una relación. Los diagramas no indican la mínima. Por ejemplo, la figura 3-3(b) muestra que un estudiante está relacionado, como máximo, con un dormitorio, pero no muestra si un estudiante *debe estar* relacionado con una instancia de dormitorio.

Se utilizan varias formas diferentes para mostrar la **cardinalidad mínima.** Una de ellas, que se ilustra en la figura 3-4, es poner una marca a lo largo de la línea de relación para indicar que debe existir una entidad en la relación, y colocar un óvalo para señalar que ahí puede o no haber una entidad en la relación. En tal sentido, la figura 3-4 muestra que un DORMITORIO debe tener una relación cuando menos con un ESTUDIANTE, pero que un ESTUDIANTE no necesariamente debe estar relacionado con un DORMITORIO. La relación completa de restricciones indica que un DORMITORIO tiene una cardinalidad mínima de uno y una cardinalidad máxima de muchas entidades de ESTUDIANTE. Un ESTUDIANTE tiene una cardinalidad mínima de cero y una cardinalidad máxima de una entidad DORMITORIO.

Puede existir una relación entre entidades de la misma clase. Por ejemplo, la relación HABITACIONES-CON (ROOMS-WITH) se podría definir en la entidad ESTUDIANTE. La figura 3-5(a) muestra esta relación, y la figura 3-5(b) ejemplifica las instancias de entidades que conforman esta relación. A las relaciones entre entidades de una sola clase a veces se les llama **relaciones recursivas.**

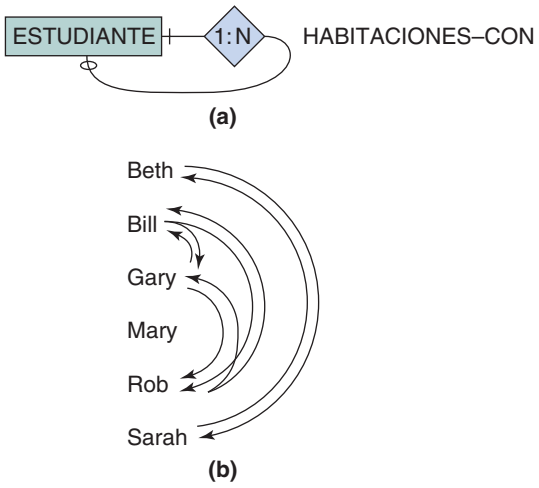
MUESTRAS DE ATRIBUTOS EN LOS DIAGRAMAS ENTIDAD-RELACIÓN. En algunas versiones de los diagramas E-R los atributos se muestran en óvalos, que están conectados a la entidad o a la relación que pertenecen. La figura 3-6(a) ejemplifica las entidades ESTUDIANTE y DORMITORIO y la relación DORM-OCUPANTE con los atributos. Como se muestra, DORMITORIO tiene los atributos NombredelDormitorio, Ubicación, y NúmerodeHabitaciones, y ESTUDIANTE tiene los atributos NúmerodeEstudiante, NombredelEstudiante y AñodelEstudiante. La relación DORM-OCUPANTE tiene el atributo Renta, el cual muestra la cantidad que paga un estudiante en particular por determinado dormitorio.

Si una entidad tiene muchos atributos, enumerarlos de esta manera en el diagrama E-R puede ocasionar desorden y dificultad para interpretarlos. En estas instancias, los atributos de entidad se listan por separado, como se muestra en la figura 3-6(b). Muchas herramientas CASE muestran estos atributos en Windows pop-up.

[5] Los símbolos gráficos que surgieron con el modelo (los cuales se describen aquí) no son los más indicados para mostrar un modelo en un sistema de Interfaz Gráfica de Usuario (GUI, por sus siglas en inglés), como Macintosh o como Microsoft Windows. En realidad, el modelo E-R fue desarrollado antes de que fuera popular cualquier sistema GUI. Los símbolos UML que se muestran posteriormente en este capítulo se utilizan con más facilidad en un medio ambiente gráfico.

➤ FIGURA 3-5

Relación recursiva



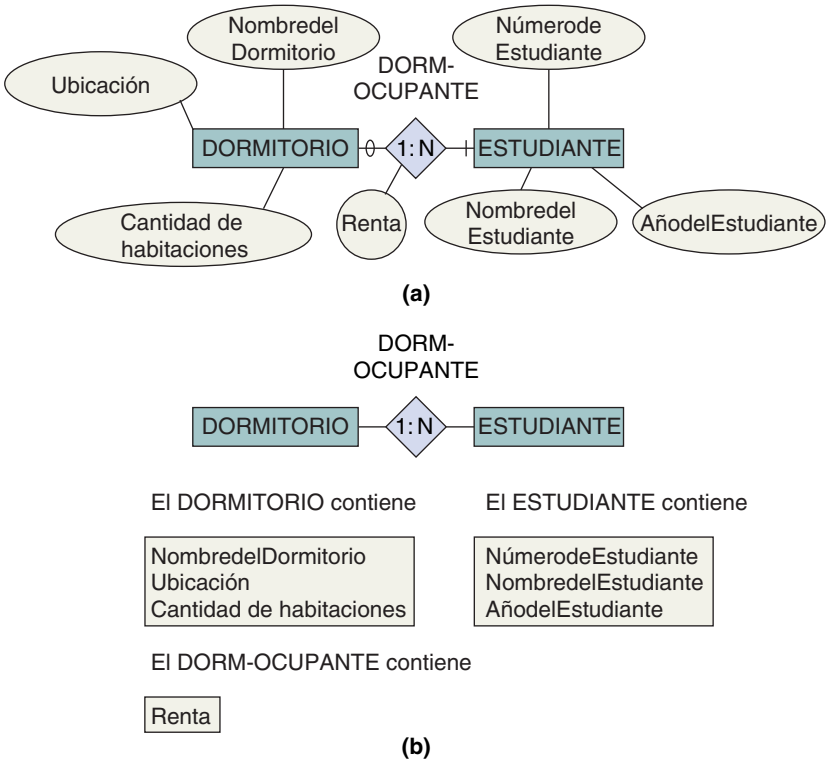
Entidades débiles

El modelo entidad-relación define un tipo especial de entidad llamada **entidad débil.** Ésta no puede existir en la base de datos, a menos que también haya otro tipo de entidad en la base de datos. Una entidad que no es débil se llama **entidad fuerte.**

Para entender a las entidades débiles considere una base de datos humana, con clases de entidades EMPLEADO y ENCARGADO. Suponga que el negocio tiene una regla referente a que una instancia EMPLEADO *puede existir* sin tener una relación con cualquier entidad ENCARGADO, pero una entidad ENCARGADO *no puede existir* sin tener una relación con una entidad particular EMPLEADO. En este caso, ENCARGADO es una entidad débil. Esto significa que los datos de ENCARGADO se pueden almacenar en la base de datos únicamente si ENCARGADO tiene una relación con una entidad EMPLEADO.

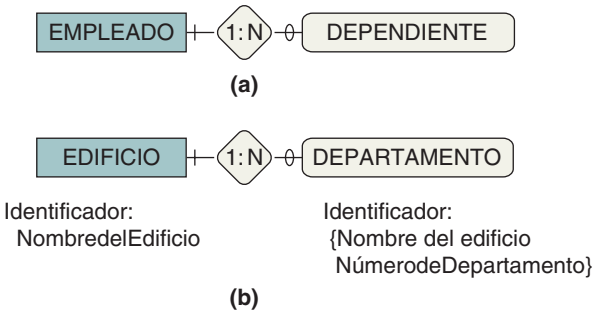
➤ FIGURA 3-6

Propiedades que se muestran en los diagramas entidad-relación: (a) diagrama entidad-relación que muestra las propiedades, y (b) diagrama entidad-relación con las propiedades listadas por separado



➤ FIGURA 3-7

Entidades débiles: (a) ejemplo de una entidad débil, y (b) entidad dependiente de ID



Como se muestra en la figura 3-7(a), las entidades débiles se representan mediante las esquinas redondeadas del rectángulo de la entidad. Además, la relación de la cual depende de la existencia de la entidad se muestra en un diamante que también tiene las esquinas redondeadas. Alternativamente, en algunos diagramas E-R (que no se muestran aquí), las entidades débiles se representan usando una línea doble para delimitar el rectángulo de la entidad débil, y diamantes dobles para denotar la relación de la cual depende.

El modelo E-R incluye un tipo especial de entidad débil llamada **entidad dependiente de un identificador.** En ésta, el identificador de una entidad incluye al de otra. Considere las entidades EDIFICIO (BUILDING) y DEPARTAMENTO (APARTMENT). Suponga que el identificador EDIFICIO es NombredelEdificio, y que el identificador de DEPARTAMENTO es el identificador compuesto {NombredelEdificio, NúmerodeDepartamento}. Puesto que el identificador de DEPARTAMENTO contiene al de EDIFICIO (NombredelEdificio), entonces DEPARTAMENTO es dependiente del identificador en EDIFICIO. Compare la figura 3-7(b) con la figura 3-7(a). Otra manera de pensar esto es que, tanto lógica como físicamente, un DEPARTAMENTO no puede existir a menos que haya un EDIFICIO.

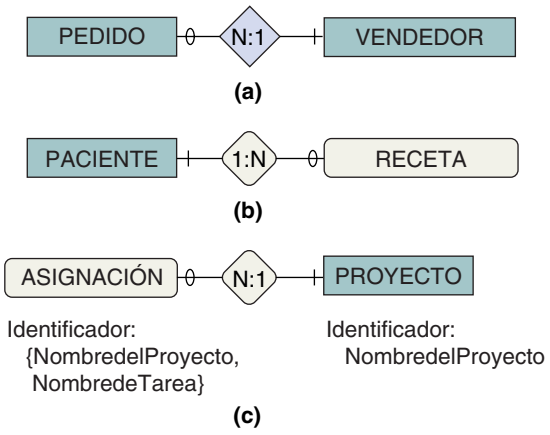
Las entidades dependientes de un identificador son comunes. Otro ejemplo es la entidad VERSIÓN en la relación entre PRODUCTO y VERSIÓN, donde PRODUCTO es un software y VERSIÓN es la salida de ese producto de software. El identificador de PRODUCTO es NombredelProducto y el identificador de VERSIÓN es {NombredelProducto, NúmerodeEdición}. Un tercer ejemplo es EDICIÓN en la relación entre LIBRO DETEXTO y EDICIÓN. El identificador de LIBRODETEXTO es el título y el identificador de EDICIÓN es {Título, NúmerodeEdición}.

Por desgracia, existe una ambigüedad oculta en la definición de entidad débil, y los diseñadores de bases de datos (al igual que algunos autores de libros de texto) la interpretan de manera diferente. La ambigüedad, en un sentido estricto, radica en lo siguiente: si una entidad débil es definida como cualquier entidad cuya presencia en la base de datos depende de otra entidad, entonces cualquier entidad que participe en una relación con una cardinalidad mínima de 1 en una segunda entidad, es una entidad débil. De esta manera, en una base de datos académica, si ESTUDIANTE debe tener un ASESOR, entonces ESTUDIANTE es una entidad débil, porque una entidad ESTUDIANTE no puede ser almacenada sin un ASESOR. Sin embargo, esta interpretación también parece amplia para algunas personas. Un ESTUDIANTE no es físicamente dependiente de un ASESOR (a diferencia de DEPARTAMENTOs y EDIFICIOs), y un ESTUDIANTE no es lógicamente dependiente de un ASESOR (¡a pesar de lo que le pueda parecer a cualquiera de los dos!); por lo tanto, ESTUDIANTE debe ser considerada una entidad fuerte.

Para evitar estas situaciones, algunas personas interpretan la definición de entidad débil de manera más restringida. Para que sea una entidad débil debe depender *lógicamente* de otra. De acuerdo con esta definición, tanto ENCARGADO como DEPARTAMENTO podrían ser consideradas entidades débiles; pero ESTUDIANTE, no. Un ENCARGADO no puede ser un encargado a menos que algo dependa de él, y un DEPARTAMENTO no puede existir sin un EDIFICIO del cual formar parte. Sin embargo, ESTUDIANTE puede existir lógicamente sin ASESOR, aunque lo requiera una regla de negocio.

➤ FIGURA 3-8

Ejemplos de entidades requeridas



Para ilustrar esta interpretación, consideremos varios ejemplos. Suponga que un modelo de datos incluye la relación entre PEDIDO y VENDEDOR (figura 3-8[a]). Aun cuando podemos decir que un PEDIDO debe tener un VENDEDOR, no necesariamente requiere uno para existir (el PEDIDO puede ser una venta en efectivo en la que no se registra al vendedor). De ahí que la cardinalidad mínima de 1 derive de una regla de negocios, no de una necesidad lógica. Así, PEDIDO requiere un VENDEDOR pero su existencia no depende de él, y PEDIDO puede ser considerado una entidad fuerte.

Ahora, considere la relación entre PACIENTE (PATIENT) y RECETA (PRESCRIPTION) de la figura 3-8(b). Aquí, una RECETA no puede existir lógicamente sin un PACIENTE, así que no sólo tiene la cardinalidad mínima 1, sino que depende de la existencia de PACIENTE. Así, RECETA es una entidad débil. Por último, considere ASIGNACIÓN (ASSIGNMENT) en la figura 3-8(c), donde el identificador de ASIGNACIÓN contiene al identificador de PROYECTO. Aquí, no sólo ASIGNACIÓN tiene una cardinalidad mínima de 1, y no sólo ASIGNACIÓN es dependiente de la existencia de un PROYECTO, sino también ID es dependiente de PROYECTO, puesto que su llave incluye la llave de otra entidad. De esta manera, ASIGNACIÓN es una entidad débil.

En este texto definimos a las entidades débiles como aquellas que deben depender lógicamente de otra. Por lo tanto, no todas las entidades que tienen una cardinalidad mínima de 1 en su relación con otra entidad son débiles. Sólo aquellas que son lógicamente dependientes se denominan débiles. Esta definición también implica que todas las entidades dependientes de un identificador son débiles. Además, cada entidad débil tiene una cardinalidad mínima de 1 en la entidad de la cual depende; pero cada entidad que tiene una cardinalidad mínima de 1 no necesariamente debe ser débil.[6]

REPRESENTACIÓN DE ATRIBUTOS MULTIVALOR CON ENTIDADES DÉBILES.

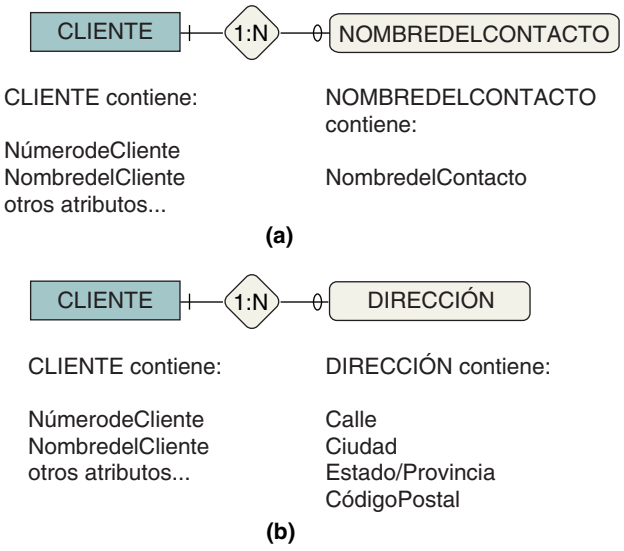
Los atributos multivalor se representan en un diagrama E-R con el fin de crear una nueva entidad débil para representar el atributo multivalor y construir una relación uno a muchos. Por ejemplo, la figura 3-9(a) muestra la representación del atributo multivalor NombredelContacto en CLIENTE. Una nueva entidad débil llamada NOMBREDELCONTACTO se crea con un solo atributo NombredelContacto. La relación entre CLIENTE y NOMBREDELCONTACTO es uno a muchos. La entidad construida debe ser débil porque es lógicamente dependiente de la entidad que tiene el atributo multivalor.

La figura 3-9(b) muestra la representación del atributo compuesto multivalor Dirección. La nueva entidad débil DIRECCIÓN contiene todos los atributos del compuesto: Calle, Ciudad, Estado/Provincia, CP.

[6] Este análisis omite las instancias en las que la cardinalidad mínima es mayor a 1. La lógica es similar, pero la entidad depende de un grupo de entidades.

➤ FIGURA 3-9

Representación de atributos multivalor con entidades débiles



ENTIDADES SUBTIPO

Algunas entidades contienen conjuntos opcionales de atributos; éstas a menudo se representan utilizando subtipos.[7] Considere, por ejemplo, CLIENTE, con atributos NúmerodeCliente, NombredelCliente y CantidadqueAdeuda. Suponga que un CLIENTE puede ser una persona, una sociedad o una empresa y que los datos adicionales que se pueden almacenar dependen de qué tipo sean. Suponga que los datos adicionales son los siguientes:

CLIENTE-PERSONA
Dirección, NúmerodeSeguroSocial
CLIENTE-SOCIEDAD
NombredelSocioAdministrador, Dirección, NúmerodeIdentificaciónFiscal
CLIENTE-EMPRESA
PersonadeContacto, Teléfono, NúmerodeIdentificaciónFiscal

Una posibilidad es asignar todos estos atributos a la entidad CLIENTE, como se muestra en la figura 3-10(a). En este caso, algunos de los atributos no son aplicables. NombredelSocioAdministrador no tiene significado para un cliente o una empresa, y por lo tanto, no puede tener ningún valor.

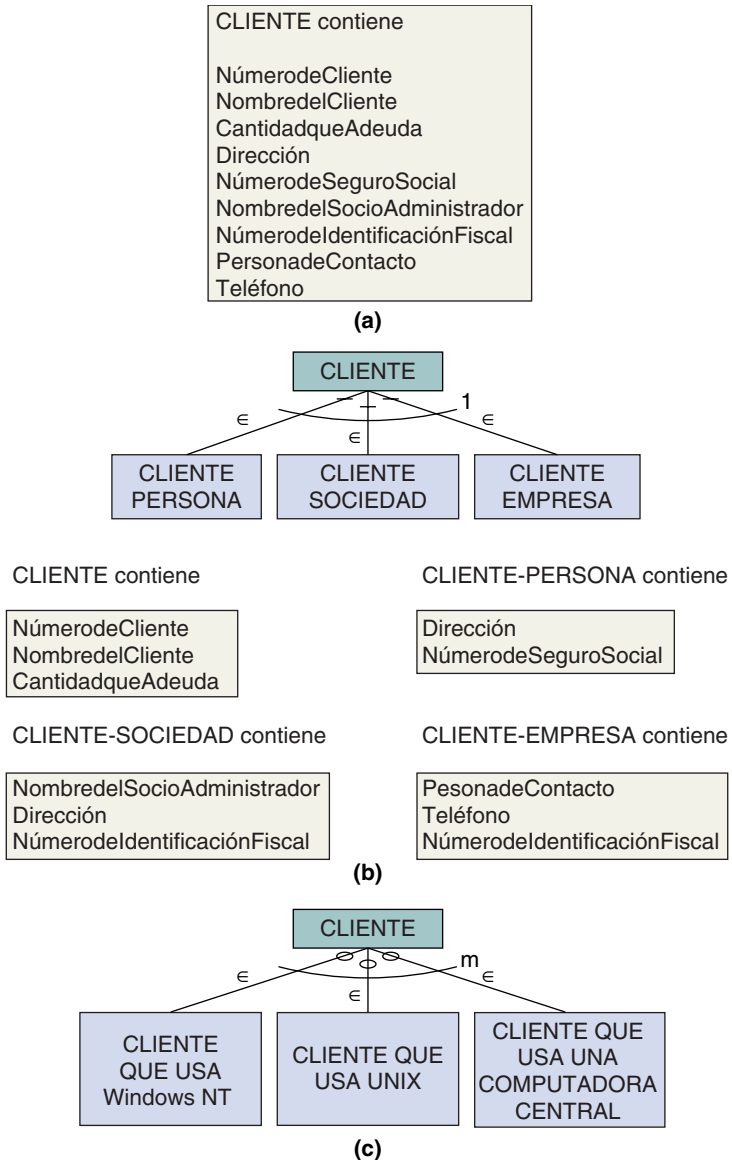
Un modelo más preciso definiría tres entidades subtipo, como se muestra en la figura 3-10(b). Aquí las entidades CLIENTE-PERSONA, CLIENTE-SOCIEDAD y CLIENTE-EMPRESA se muestran como **subtipos** de CLIENTE. A su vez, CLIENTE es un **supertipo** de las entidades CLIENTE-PERSONA, CLIENTE-SOCIEDAD y CLIENTE-EMPRESA.

El símbolo ∈ junto a las líneas de relación indica que CLIENTE-PERSONA, CLIENTE-SOCIEDAD y CLIENTE-EMPRESA son subtipos de CLIENTE. Cada entidad subtipo debe pertenecer al supertipo CLIENTE. La línea curva con un 1 próximo indica que la entidad CLIENTE debe pertenecer únicamente a un subtipo. Significa que los subtipos son excluyentes y que se requiere uno de ellos.

[7] Los subtipos se agregaron al modelo E-R después de la publicación del artículo original de Chen, y son parte de lo que se llama el *modelo E-R extendido*.

➤ FIGURA 3-10

Entidades subtipo: (a) CLIENTE sin entidades subtipo, (b) CLIENTE con entidades subtipo, y (c) subtipos no excluyentes con supertipo opcional



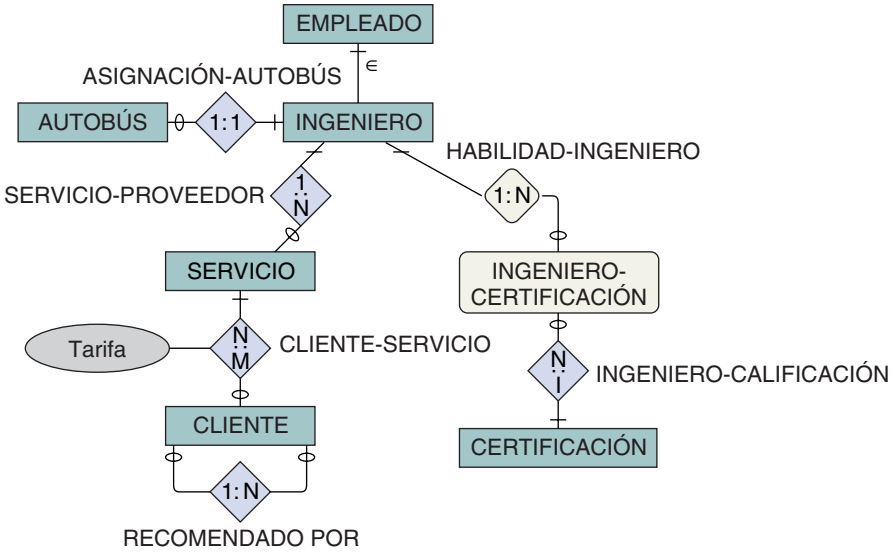
Las entidades con una relación *ES-UN* deberían tener el mismo identificador, puesto que representan diferentes aspectos de una misma cosa. En este caso, el identificador es Número de Cliente. Esta situación contrasta con la relación TIENE-UN que se muestra en la figura 3-3, donde las entidades representan aspectos de diferentes cosas.

Las jerarquías de generalización tienen una característica especial llamada **herencia,** lo cual significa que las entidades en subtipos heredan atributos de la clase de entidad supertipo. Por ejemplo, CLIENTE-SOCIEDAD, hereda de CLIENTE NombredelCliente y CantidadqueAdeuda.

Las razones para utilizar subtipos en la modelación de datos difieren de las razones para usarlos en programación orientada a objetos. De hecho, la única razón para usarlos en el modelo de datos es evitar situaciones en las cuales se requieran algunos atributos que sean nulos. Por ejemplo, en la figura 3-10(a), si el NúmerodeSeguroSocial tiene un valor, entonces los últimos cuatro atributos deben ser nulos. La situación puede ser más obvia en aplicaciones médicas; por ejemplo, el hecho de preguntarle a un hombre enfermo cuántos embarazos ha tenido. Los valores nulos se analizan con mayor detalle en el capítulo 6.

➤ FIGURA 3-11

Ejemplo de un diagrama entidad-relación



Ejemplo de diagrama E-R

La figura 3-11 es un ejemplo de diagrama E-R con todos los elementos del modelo E-R que hemos estudiado. Muestra las entidades y las relaciones para una compañía consultora de ingeniería que analiza la construcción y estado de casas y de otros edificios e instalaciones.

Hay un tipo de entidad para los empleados de la compañía. Debido a que algunos EMPLEADOs son INGENIEROs, hay un subtipo de relación entre EMPLEADO e INGENIERO. Cada INGENIERO debe ser un EMPLEADO; el INGENIERO tiene una relación de 1:1 con AUTOBÚS, y cada AUTOBÚS debe estar asignado a un INGENIERO, pero no todos los INGENIEROs tienen un AUTOBÚS.

Los INGENIEROs proporcionan SERVICIOs a los CLIENTEs. Un INGENIERO puede proporcionar desde cero hasta muchos servicios, pero un SERVICIO debe ser proporcionado sólo por determinado INGENIERO, y ese servicio únicamente lo puede realizar ese ingeniero en particular. Los CLIENTEs tienen muchos SERVICIOs, y un SERVICIO puede ser solicitado por muchos CLIENTEs. Un CLIENTE debe haber comprado por lo menos un SERVICIO, pero un SERVICIO no necesariamente tiene que haber tenido CLIENTEs. La relación CLIENTE-SERVICIO tiene un atributo Tarifa, el cual muestra la cantidad que un cliente en particular paga por un SERVICIO determinado. (En este diagrama no se muestran otros atributos de entidades y relaciones.)

Algunas veces unos CLIENTEs envían a otros, lo cual se indica mediante la relación recursiva RECOMENDADO-POR. Un CLIENTE puede recomendar a uno o a varios CLIENTEs más. Un CLIENTE puede o no haber sido recomendado por otro cliente, pero un CLIENTE puede ser recomendado sólo por otro CLIENTE.

La entidad INGENIERO-CERTIFICACIÓN muestra que un INGENIERO ha concluido su educación y ha obtenido la aprobación necesaria para tener un certificado. Un INGENIERO puede obtener varias CERTIFICACIONes. La existencia de INGENIERO-CERTIFICACIÓN depende del INGENIERO a través de la relación HABILIDAD-INGENIERO. La CERTIFICACIÓN es la entidad que describe determinada certificación.

Documentación de reglas del negocio

En el capítulo 2 definimos un esquema de base de datos que consta de tablas, relaciones, dominios y reglas del negocio. Podemos obtener o inferir los primeros tres a partir de un modelo E-R, pero no podemos obtener reglas del negocio de ese modelo. Así, algunas veces estas reglas se agregan al modelo E-R durante la etapa de la modelación de datos.

El modelo E-R se desarrolla a partir de un análisis de requerimientos de los usuarios, durante el cual con frecuencia surgen las reglas del negocio; de hecho, los analistas de sistemas deben poner mucha atención en las preguntas que hacen con respecto a los requerimientos.

Considere las entidades AUTOBÚS e INGENIERO de la figura 3-11. ¿La empresa tiene reglas para determinar a quién se le asigna un AUTOBÚS? Si no hay suficientes AUTOBUSes para que se le asigne uno a cada INGENIERO, ¿con qué regla se determina a quién se le asigna uno? Podría ser que la aplicación de la base de datos sea asignar autobuses dependiendo de cuál INGENIERO tenga que prestar el mayor número de SERVICIOs fuera de la oficina durante cierto periodo, o alguna otra norma similar.

Otro ejemplo se refiere a la asignación de INGENIEROs a SERVICIOs. Probablemente hay reglas relacionadas con el tipo de INGENIERO-CERTIFICACIÓN que un INGENIERO debe tener para que se le asignen determinados tipos de SERVICIOs. Por ejemplo, para inspeccionar un edificio de departamentos, el INGENIERO puede necesitar una licencia de INGENIERO profesional. Incluso, si no hay una ley que dicte esta regla, la política de la compañía puede ser que este requisito se cumpla.

Las reglas del negocio se pueden o no aplicar a través del DBMS, o por medio del programa de aplicación. Algunas veces las reglas del negocio están escritas en un manual de procedimientos que deben seguir los usuarios de la base de datos. En este punto no es importante la manera en que habrán de imponerse las reglas; lo que interesa es documentarlas para que sean parte de los requerimientos del sistema.

El modelo entidad-relación y las herramientas CASE

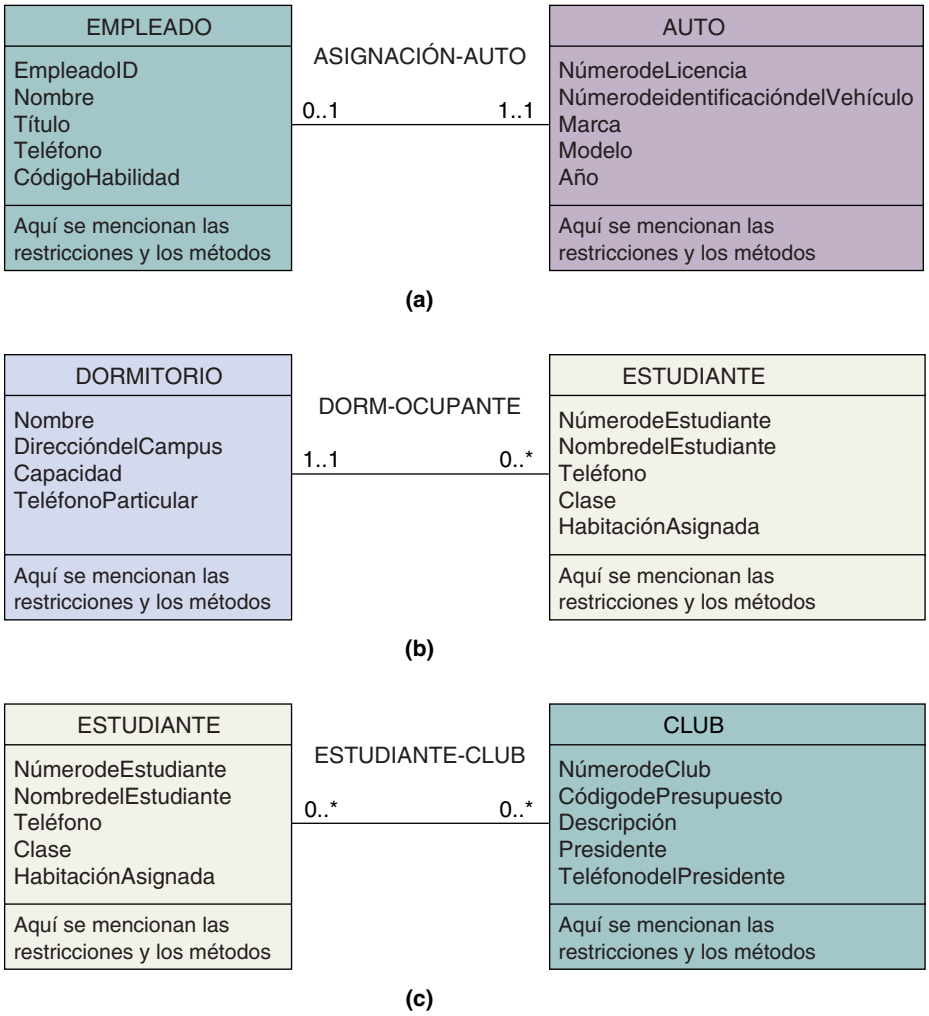
Desarrollar un modelo de datos utilizando el modelo entidad-relación ha sido más fácil en años recientes, ya que las herramientas para construir los diagramas E-R están incluidas en muchos productos CASE que son muy populares. Productos tales como IEW, IEF, DEFT, ER-WIN y Visio tienen facilidades de dibujo y diagramación para crear diagramas E-R. Estos productos también integran entidades con las relaciones de la base de datos que los representan, los cuales pueden facilitar la administración, el manejo y el mantenimiento de la base de datos.

Para nuestro análisis no consideramos el uso de las herramientas CASE. Pero si en la universidad a la que usted asiste hay dicha herramienta, úsela para crear los diagramas E-R en los ejercicios que se le asignen. Los diagramas E-R creados usando estas herramientas por lo general son visualmente más agradables, y muy fáciles de cambiar y adaptar.

➤ DIAGRAMAS DE ENTIDAD-RELACIÓN DE ESTILO UML

El Lenguaje de modelación unificado (UML, por sus siglas en inglés) es un conjunto de estructuras y técnicas para la modelación y el diseño de programas orientados a objetos (OOP) y aplicaciones. El UML es tanto una metodología para el desarrollo de sistemas OOP como un conjunto de herramientas para apoyar el desarrollo de dichos sistemas. El UML ha adquirido importancia en el Grupo de administración de objetos (Object Management Group), organización que ha estado desarrollando modelos OOP, tecnología y estándares desde la década de 1980. También ha comenzado a tener un amplio uso entre los profesionales de OOP. El UML es la base de las herramientas de diseño orientadas a objetos de Rational Systems.

Debido a que es una metodología de aplicación de desarrollo, UML es tema para un curso de desarrollo de sistemas, y en este caso no es de mucho interés para nosotros. Sin embargo, usted puede encontrar diagramas entidad-relación de estilo UML, así que será mejor que se familiarice con dicho estilo. Conviene tener en cuenta que en el caso del diseño de la base de datos se trata de los mismos diagramas tradicionales de entidad-relación.



➤ FIGURA 3-12

Representación (a) del UML de una relación 1:1, (b) representación UML de una relación 1:N, y (c) representación UML de una relación N:M

Entidades y relaciones UML

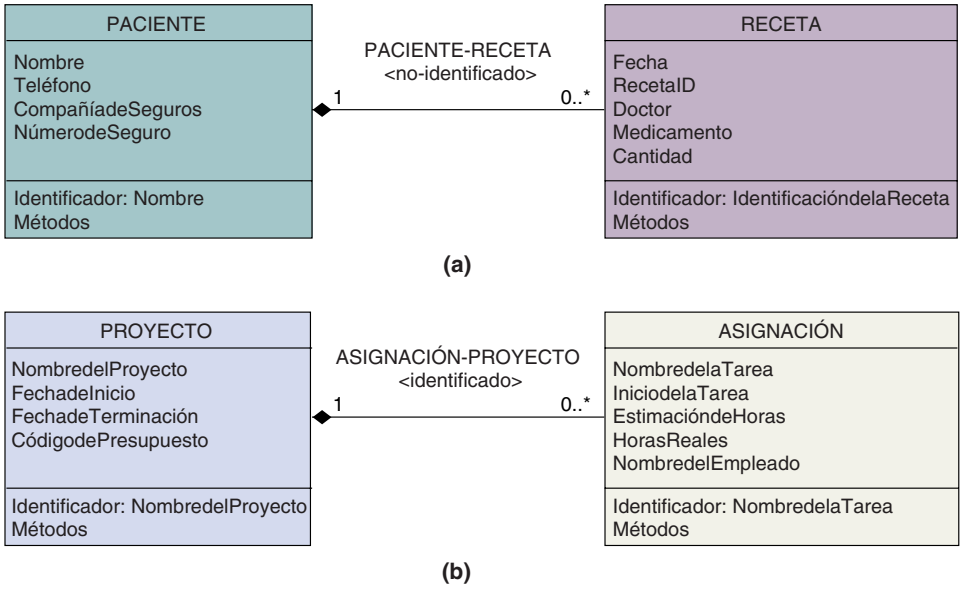
La figura 3-12 muestra la representación UML de los diseños en la figura 3-3. Cada entidad está representada por una **clase de entidad,** la cual se muestra como un rectángulo con tres segmentos. El segmento superior muestra el nombre de la entidad y otros datos que analizaremos. El segundo segmento enumera los nombres de los atributos de la entidad, y el tercero documenta las restricciones y enumera los métodos (procedimientos del programa) que corresponden a la entidad.

Las relaciones se muestran con una línea entre dos entidades. Las cardinalidades se representan en el formato *x..y*, donde *x* es lo mínimo que se requiere y es lo máximo permitido. Así, 0..1 significa que no se requiere una entidad y *y* que cuando mucho se permite una. Un asterisco significa que se permite un número ilimitado. Así, 1..* quiere decir que se requiere una y se permite un número ilimitado. Examine las figuras 3-12(a)-(c) referentes a los ejemplos de relaciones de cardinalidad máxima 1:1, 1:N y N:M.

REPRESENTACIÓN DE ENTIDADES DÉBILES. La figura 3-13 muestra la representación UML de entidades débiles. Un diamante lleno se coloca en la línea padre de la entidad débil (la entidad de la cual depende ésta). En la figura 3-13(a), RECETA es la entidad débil y PACIENTE es la entidad padre. Todas las entidades débiles tienen una entidad padre, y de esta forma la cardinalidad al lado de la entidad débil es siempre

FIGURA 3-13

Representación UML de entidades débiles. (a) Entidad débil no dependiente de un identificador, y (b) entidad débil dependiente de un identificador



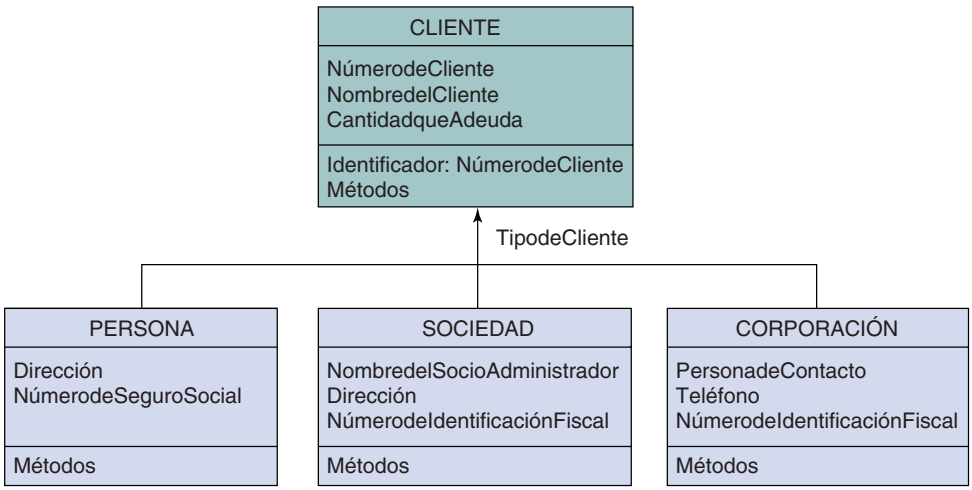
una relación 1..1. Ya que esto es así, la cardinalidad en la entidad padre se muestra simplemente como 1.

La figura 3-13(a) muestra una entidad débil que no es una entidad dependiente de un identificador. Esto se denota por la expresión <no-identificado> en la relación PACIENTE-RECETA. La figura 3-13(b) muestra una entidad débil que es dependiente de un identificador. Esto se especifica con la etiqueta <identificado>.

REPRESENTACIÓN DE SUBTIPOS. El UML representa los subtipos como se muestra en la figura 3-14. En ésta, PERSONA, SOCIEDAD y CORPORACIÓN, subtipos de CLIENTE, se permiten. De acuerdo con esta figura, determinado CLIENTE podría ser uno, dos o tres de estos subtipos. En una instancia así, esto no tiene sentido; un CLIENTE debería ser uno y solamente uno de estos tipos. La versión actual de UML no proporciona un significado al documento exclusivamente. Sin embargo, esta notación podría ser agregada al diagrama UML.

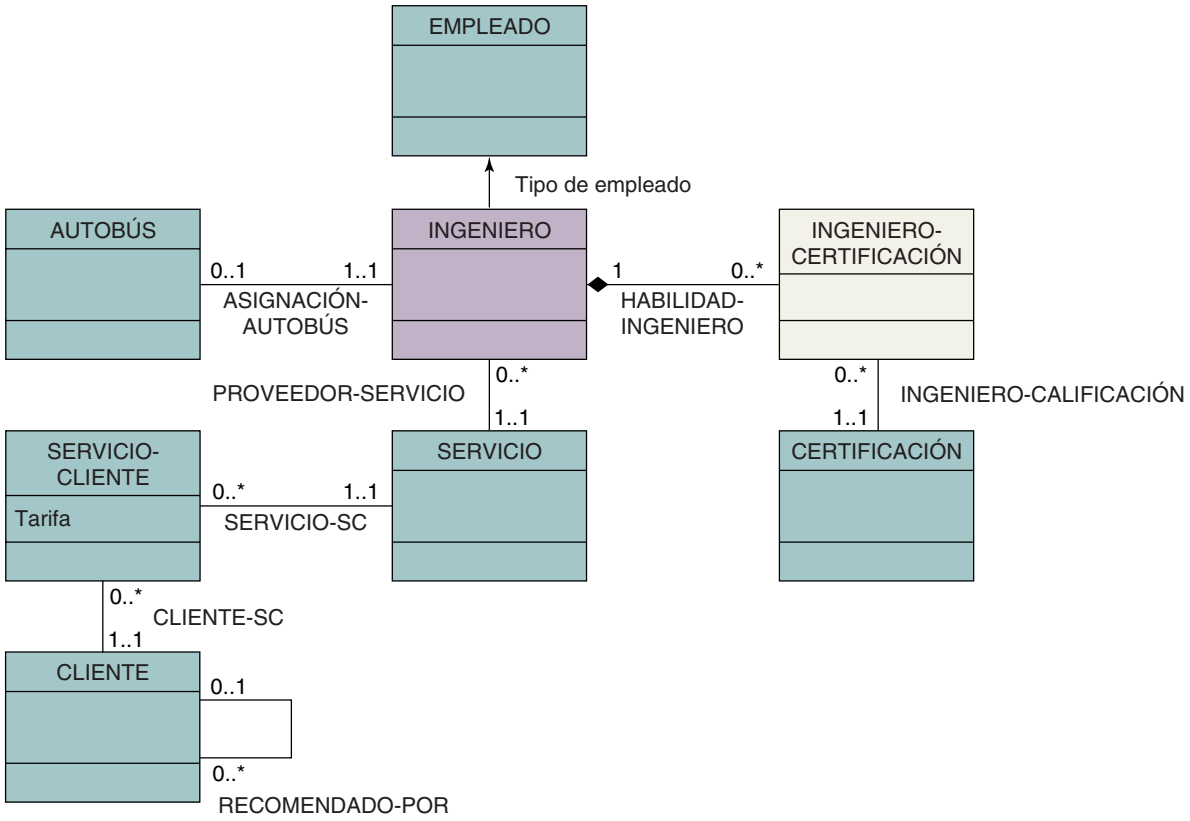
FIGURA 3-14

Representación de subtipos UML



➤ FIGURA 3-15

Versión UML del diagrama E-R de la figura 3-11

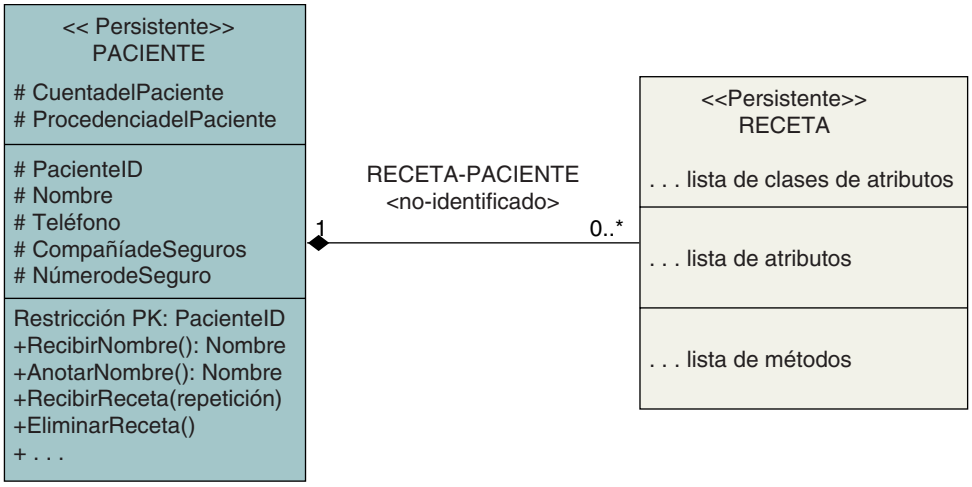


La figura 3-15 representa una versión UML del diagrama entidad-relación que se mostró en la figura 3-11. Debido a que la relación entre SERVICIO y CLIENTE tiene un atributo —Tarifa—, una entidad por separado CLIENTE-SERVICIO ha sido definida para cumplir este atributo. Ésta es una práctica estándar cuando se utilizan las herramientas UML. También observe la representación de la relación recursiva, RECOMENDADO-POR.

Construcciones OOP introducidas por medio de UML

Debido a que UML es una tecnología orientada a objetos, varias construcciones OOP han sido agregadas a las clases de entidad UML. Aquí abordaremos estas ideas y las desarrollaremos en el capítulo 18. Primero, las clases de todas las entidades que serán almacenadas en la base de datos están etiquetadas con la palabra clave <Persistente>. Esto significa que los datos deberán seguir existiendo aun cuando el objeto que se procese esté destruido. En términos simples, significa que la clase de entidad va a ser almacenada en la base de datos.

A continuación, las clases de entidad UML son tomadas en cuenta para las **clases de atributos,** que difieren de los atributos de entidad porque se relacionan con las clases de todas las entidades de un tipo determinado. Así, la figura 3-16, CuentadelPaciente de PACIENTE es un atributo del conjunto de todos los PACIENTEs en la base de datos. ProcedenciadelPaciente es un atributo que documenta la procedencia de todos los PACIENTEs en la base de datos.



➤ FIGURA 3-16

Clases de entidad UML con construcciones OOP

Como verá, estas clases de atributos no tienen razón de ser cuando se utiliza el modelo relacional. En lugar de esto, en algunos casos, atributos tales como CuentadelPaciente no se almacenan en la base de datos, sino que son calculados en el tiempo de ejecución, o run-time. En otros casos, se ingresa una nueva entidad que contenga las clases de atributos. Para la entidad de la figura 3-16 una nueva entidad, llamada PROCEDENCIA-PACIENTE, podría ser definida para contener los atributos CuentadelPaciente y ProcedenciadelPaciente. En este caso todas las entidades en PACIENTE estarán conectadas a PROCEDENCIA-PACIENTE.

Una tercera característica es que UML utiliza la notación orientada a objetos para la visibilidad de los atributos y los métodos. Los atributos que están precedidos por un + son públicos; los antecedidos por un # son protegidos, y aquellos con un – son privados. En la figura 3-16, Nombre en PACIENTE es un atributo protegido.

Estos términos surgen de la disciplina de la programación orientada a objetos. Se puede tener acceso a un atributo *público* y cambiar mediante algún método de cualquier objeto. Se puede invocar un método público con cualquier método de cualquier objeto. *Protegido* quiere decir que el atributo o método sólo es accesible mediante métodos de esta clase o de sus subclases, y *privado* significa que es accesible sólo mediante métodos de esta clase.

Por último, están las entidades UML con restricciones específicas y métodos en el tercer segmento de las clases de entidades. En la figura 3-16 se coloca una restricción de llave primaria limitada en PacienteID. Esto simplemente significa que PacienteID es un identificador único. Además, la figura 3-16 documenta que RecibirNombre() se crea para proporcionar acceso público (observe el + al frente de RecibirNombre) para el atributo Nombre; AnotarNombre() se usa para colocar su valor y el método RecibirReceta() se puede usar para intercambiar en el conjunto de entidades Receta con respecto a esta entidad PACIENTE.

El papel actual del UML en el procesamiento de bases de datos

Las ideas que se ilustran en la figura 3-16 se encuentran en aguas oscuras, donde se combinan el procesamiento de la base de datos y el pensamiento de objetos. Esta notación orientada a objetos no encaja en las prácticas y procedimientos del procesamiento de bases de datos comerciales. El concepto de que un atributo de entidad puede estar oculto en un objeto no tiene sentido, a menos que únicamente los programas orientados a objetos se procesen en la base de datos y, aun entonces, dichos programas deben procesar los datos de acuerdo con esa política. Con excepción de los productos y aplicaciones DBMS orientados a objetos (OODBMS, por sus siglas en inglés), esto nunca se hace.

En cambio, la mayoría de los productos comerciales DBMS tiene características que permiten a todos los tipos de programas acceder a la base de datos y procesar cualquier dato, por lo que tienen autoridad de seguridad. Es más, con dispositivos como el

generador de consultas de Microsoft Access 2002 (figura 2-8) no hay forma de limitar el acceso a los valores de los atributos a un solo objeto.

Así, usted debe saber cómo interpretar diagramas entidad-relación estilo UML. Éstos se pueden utilizar para el diseño de bases de datos, tal como se puede hacer con los diagramas tradicionales de estilo E-R. Sin embargo, actualmente la notación orientada a objetos que introducen tiene un valor práctico limitado. Vea el capítulo 18 para mayor información acerca de este tema.

➤ EJEMPLOS

La mejor forma que hay para obtener capacidad con cualquier herramienta de modelación es estudiar ejemplos y utilizarla para hacer sus propios modelos. El resto de este capítulo presenta dos instancias de aplicaciones para ayudarle con la primera tarea. Después de estudiar dichos ejemplos, puede trabajar los problemas que están al final del capítulo.

EJEMPLO 1: CLUB DE BAILE JEFFERSON

El club Jefferson imparte clases de baile de salón y ofrece lecciones privadas y en grupo; cobra $45 la hora por estudiante (o pareja) cuando se trata de una lección privada, y $6 la hora por estudiante en el caso de una lección en grupo. Las lecciones privadas se imparten a cualquier hora, desde el medio día hasta las 10:00 p.m. seis días a la semana. Las lecciones en grupo se imparten sólo en las tardes.

El club de baile emplea dos tipos de instructores: asalariados de tiempo completo, e instructores de medio tiempo. La remuneración de los instructores de tiempo completo consiste en una cantidad fija por semana y a los de tiempo parcial se les paga cierta cantidad por una tarde, o por impartir sólo una clase.

Además de las lecciones, el club patrocina a la semana dos presentaciones de baile social con música grabada. La cuota de admisión es de $5 por persona. El baile de los viernes por la noche es el más popular y en promedio se reúnen 80 personas; el baile del domingo por la noche atrae aproximadamente a 30. El propósito de los bailes es proporcionar a los estudiantes un lugar para que practiquen sus habilidades. No se sirven alimentos ni bebidas.

Al club de baile Jefferson le gustaría desarrollar un sistema de información para dar seguimiento a los estudiantes y a las clases que éstos han tomado. Al administrador también le gustaría saber cuántas lecciones y de qué tipo ha impartido cada maestro, y poder calcular el costo promedio por lección de cada uno de sus instructores.

ENTIDADES

La mejor manera para comenzar un modelo entidad-relación es determinar las entidades potenciales. Las entidades normalmente se representan con nombres (lugares, personas, conceptos, sucesos, equipos, etc.) en documentos o en entrevistas. Una búsqueda de ejemplos previos sobre nombres importantes que relacionan al sistema de información revela la siguiente lista:

- Lección privada
- Lección en grupo
- Maestro
- Maestro de tiempo completo
- Maestro de tiempo parcial
- Baile
- Cliente

Evidentemente, los nombres *lección privada* y *lección en grupo* tienen algo en común, al igual que los nombres *maestro, maestro de tiempo completo* y *maestro de tiempo*

parcial. Una solución es definir una entidad LECCIÓN, con subtipos LECCIÓN-PRIVADA y LECCIÓN-EN-GRUPO y otra entidad MAESTRO, con subtipos MAESTRO-DE-TIEMPO-COMPLETO y MAESTRO-DE-TIEMPO-PARCIAL. Las entidades adicionales son BAILE y CLIENTE.

Como se estableció en el capítulo 2, la modelación de datos es arte y ciencia. La solución antes descrita es sólo una de las soluciones factibles. Otra posibilidad sería eliminar LECCIÓN y MAESTRO de la lista del párrafo anterior y suprimir todos los subtipos. Una tercera opción sería eliminar LECCIÓN (puesto que lección nunca se mencionó como un nombre), pero mantener MAESTRO y sus subtipos. Aquí elegimos el tercer caso porque parece ser el más adecuado considerando la información que tenemos. Así, la lista de entidades es LECCIÓN-PRIVADA, LECCIÓN-EN-GRUPO, MAESTRO, MAESTRO-DE-TIEMPO-COMPLETO, MAESTRO-DE-TIEMPO-PARCIAL, BAILE y CLIENTE.

Elegir entre estas alternativas requiere analizar los requerimientos y considerar las implicaciones de diseño de cada una. A veces ayuda considerar los atributos de las entidades. Si, por ejemplo, la entidad LECCIÓN no tiene más atributos que su identificador, entonces no es necesaria.

Relaciones

Para empezar, MAESTRO tiene dos entidades subtipo: MAESTRO-DE-TIEMPO-COMPLETO y MAESTRO-DE-TIEMPO-PARCIAL. Un profesor determinado debe corresponder a uno u otro, así que los subtipos son mutuamente excluyentes.

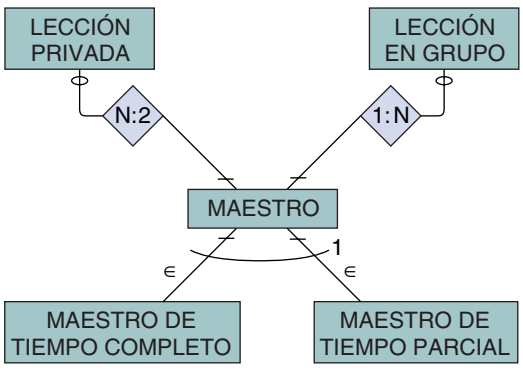
A continuación considere la relación entre MAESTRO y LECCIÓN-PRIVADA y LECCIÓN-EN-GRUPO. Un MAESTRO puede impartir muchas LECCIONes PRIVADAs y normalmente una LECCIÓN-PRIVADA la imparte un solo MAESTRO. Sin embargo, una plática posterior con el administrador reveló que para bailarines avanzados, especialmente los que se preparan para competencias, a veces participan dos maestros en una lección privada. Por lo tanto, la relación entre MAESTRO y LECCIÓN-PRIVADA debe ser de muchos a muchos. Supongamos que sólo un profesor es asignado a una lección en grupo. Las relaciones antes descritas se muestran en la figura 3-17.

Los CLIENTEs deben tomar LECCIONes-PRIVADAs o LECCIONes-EN-GRUPO. Algunas veces una persona toma una lección y otras veces una pareja. Hay dos formas en que se puede modelar esta situación. Una entidad PAREJA puede ser definida como una relación uno a dos con CLIENTE, o CLIENTE o PAREJA pueden tener una relación con LECCIÓN-PRIVADA. Supongamos que las parejas no toman lecciones en grupo, o que si lo hacen no es importante almacenar esa información en la base de datos. Esta alternativa se muestra en la figura 3-18(a).

La existencia de LECCIONes PRIVADAs depende de CLIENTE o PAREJA. Esto es, una lección no puede existir a menos que se imparta a cualquiera, a un CLIENTE o a una PAREJA. El 1 en el extremo derecho de la línea horizontal, debajo de CLIENTE y PAREJA, in-

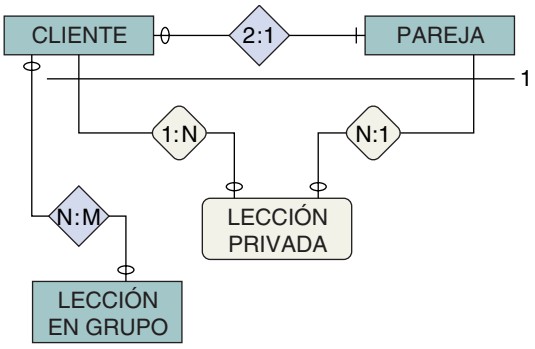
➤ FIGURA 3-17

Diagrama E-R inicial para el club de baile Jefferson

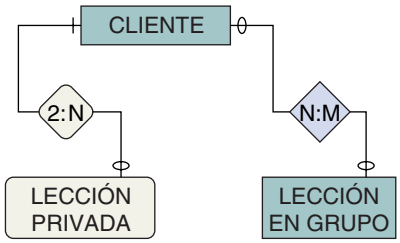


➤ FIGURA 3-18

Alternativas para la representación de CLIENTE: (a) diagrama E-R que muestra la entidad PAREJA, y (b) diagrama E-R sin parejas



(a)



(b)

dica que la LECCIÓN-PRIVADA debe tener cuando menos un CLIENTE o una PAREJA, lo cual tiene sentido puesto que LECCIÓN-PRIVADA depende de ellos.

Otra alternativa es no representar a las parejas, sino modelar la relación entre CLIENTE y LECCIÓN-PRIVADA como muchos a muchos. Específicamente, esta relación es uno o dos a muchos y se muestra en la figura 3-18(b). Aunque el modelo no se detalla como el de la figura 3-18(a), puede ser suficiente para los propósitos del club de baile.

La última relación posible es entre BAILE y otras entidades. Tanto los clientes como los profesores asisten a los bailes, y los analistas deben decidir si es importante o no almacenar estas relaciones. ¿Realmente Jefferson necesita saber qué clientes asisten a cuáles bailes? ¿El administrador del club en realidad quiere que quede registrado en el sistema de información computarizado cuando los clientes atraviesan la puerta? ¿Los clientes también desean que ese hecho quede registrado? Probablemente ésta no es una relación que necesite o deba ser almacenada en la base de datos.

La situación entre BAILE y MAESTRO es diferente. A Jefferson le gusta que sus maestros asistan a todos los bailes. Con el fin de hacer que este requerimiento sea equitativo, el administrador ha programado la asistencia de los maestros a los bailes. Desarrollar y registrar este programa requiere que la base de datos contenga la relación BAILE-MAESTRO, la cual es muchos a muchos.

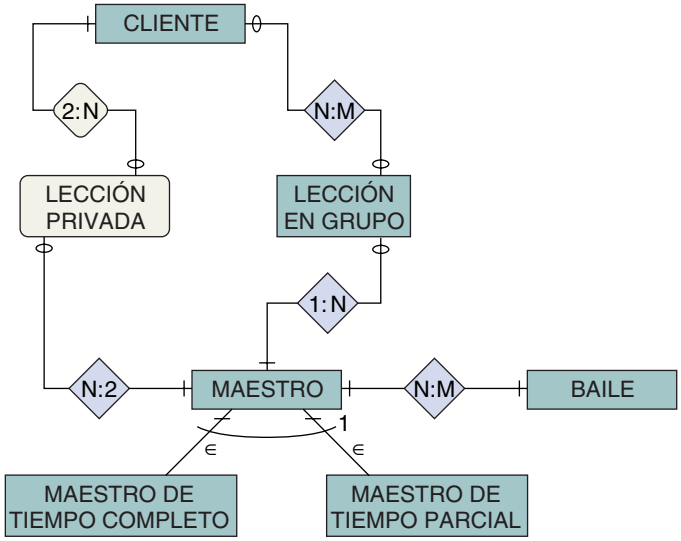
Diagrama E-R final para el club de baile Jefferson

La figura 3-19 muestra un diagrama E-R para el modelo descrito en esta sección. No hemos puesto nombre a las relaciones en este diagrama. Aunque al hacerlo los diagramas resultarían como era de esperarse, para nuestros datos, nombrar las relaciones no aportaría nada.

La existencia de LECCIONes PRIVADAs depende de CLIENTE, pero las LECCIONes-EN-GRUPO no, debido a que las lecciones en grupo son programadas mucho antes de que cualquier cliente se inscriba, y a que se seguirán realizando aunque los clientes no asistieran. Sin embargo, esta situación no es verdad en cuanto a las lecciones privadas porque sólo se programan a solicitud del cliente. También observe que este modelo no representa parejas.

➤ FIGURA 3-19

Diagrama final E-R para el club de baile Jefferson



Una vez que se ha desarrollado un modelo como éste, se debe verificar su precisión y que los requerimientos hayan sido satisfechos. Por lo general esto se hace con los usuarios.

Evaluación del modelo de datos E-R

Es más fácil y más barato corregir errores al principio del desarrollo de la base de datos que al final. Por ejemplo, cambiar la cardinalidad máxima de un relación de 1:N a N:M en la etapa de modelación de datos sólo es cuestión de registrar el cambio en el diagrama E-R. Pero una vez que se ha diseñado y cargado la base de datos con información y los programas de aplicación escritos para procesarla, hacer ese cambio requerirá muchísimo trabajo y posiblemente semanas para realizarlo. Por lo tanto, es importante evaluar el modelo de datos antes del diseño.

Una técnica de evaluación es considerar el modelo de datos E-R en el contexto de las posibles consultas que se podrían plantear a una base de datos con la estructura que implica el modelo. Por ejemplo, observe el diagrama de la figura 3-19. ¿Qué preguntas podrían contestarse con una base de datos que se implementara de acuerdo con este diseño?

- ¿A quiénes se impartieron las lecciones privadas?
- ¿Cuáles clientes han tomado una lección privada con Jack?
- ¿Quiénes son los maestros de tiempo completo?
- ¿Quiénes son los maestros que están programados para asistir al baile del viernes?

Cuando se evalúe el modelo de datos E-R, usted puede formular estas preguntas y pedir a los usuarios que redacten su propia lista de preguntas. Esas preguntas pueden ser comparadas con el diseño para comprobar su pertinencia. Por ejemplo, suponga que los usuarios preguntan cuáles clientes asistieron al baile del viernes pasado. Los diseñadores del modelo de datos en la figura 3-19 concluirían que su diseño no fue el correcto, porque no es posible contestar esta pregunta utilizando su modelo E-R. Si se debe responder esa pregunta, entonces hay que construir una relación entre CLIENTE y BAILE.

Evidentemente, este proceso informal y libremente estructurado no se puede utilizar para *probar* que un diseño es correcto. Sin embargo, ésta es una técnica pragmática que se puede utilizar para verificar el correcto potencial de un diseño. ¡Y es mejor esto que no hacer ninguna evaluación!

Ejemplo 2: Paseos en barco San Juan

Paseos en barco San Juan es una agencia que alquila embarcaciones a sus clientes mediante una tarifa determinada. Dicha agencia no tiene embarcaciones propias, sino que las rentas en nombre de los propietarios que desean obtener ingresos cuando no usan sus botes. San Juan cobra una cuota por sus servicios. Se especializa en alquileres de embarcaciones que se pueden utilizar para pasear durante varios días o semanas. El barco más pequeño en su inventario es de 28 pies de largo y el más grande es de 51 pies.

Cada embarcación está completamente equipada cuando se alquila. La mayor parte del equipo lo proporcionan los dueños, pero San Juan agrega una parte. El dueño proporciona aditamentos fijos tales como radios, brújulas, indicadores de profundidad, así como estufas y refrigeradores. Otro equipo que proporciona el propietario no forma parte del barco, por ejemplo velas, cuerdas, anclas, lanchas, salvavidas, y, en la cabina, platos, cubiertos de plata, utensilios de cocina, ropa de cama y similares. La empresa pone equipo que también podría considerarse como provisiones: diagramas, libros de navegación, tablas de mareas y corrientes, jabón, toallas para trastos, papel higiénico y artículos similares.

Parte importante de las responsabilidades de San Juan es registrar el equipo del barco. Suele ser costoso; en particular, el que no está fijo se puede perder fácilmente o ser robado. Los clientes son responsables de todo el equipo durante el tiempo que alquilan el barco.

A San Juan le gusta conservar los registros exactos de sus clientes y de los alquileres no sólo con fines de mercadotecnia, sino también para registrar los viajes que los clientes han hecho. Algunos itinerarios y condiciones climatológicas son más peligrosos que otros, y le gusta saber cuál es la experiencia de sus clientes.

El negocio principal de San Juan es únicamente el alquiler de los barcos, lo que significa que no prestan servicios de tripulación, tales como marineros u otros ayudantes. Sin embargo, en algunos casos los clientes solicitan los servicios de algún ayudante y entonces la casa contrata a ese personal por horas.

Los barcos de paseo con frecuencia necesitan mantenimiento. Los contratos que hace San Juan con los propietarios incluyen registros precisos de todas las actividades y costos de mantenimiento, incluyendo actividades normales tales como limpieza o cambios de aceite del motor, así como reparaciones no programadas. En algunos casos durante un paseo se necesitan algunas reparaciones. Por ejemplo, el motor de un barco puede fallar cuando éste está lejos de las instalaciones de San Juan. En ese caso, los clientes se comunican por radio y la empresa determina la mejor alternativa de reparación y envía personal al barco descompuesto. Para tomar estas decisiones, los despachadores necesitan información acerca de las opciones de reparación, así como antecedentes en cuanto a calidad y costos.

Antes de que continúe leyendo, intente usted mismo producir un diagrama entidad-relación. Examine los enunciados anteriores y busque los nombres que parezcan importantes para el diseño. Después compruebe las posibles relaciones entre las entidades. Por último, enumere los atributos comunes para cada entidad o relación.

Entidades

El modelo de datos que requiere la casa de alquiler de embarcaciones San Juan es más complicado que el del club de baile Jefferson. Las entidades potenciales se muestran en la figura 3-20(a).

Primero considere las entidades que se relacionan con el equipo. Existen varias clases diferentes de equipo y este dato sugiere la posibilidad de modelar los subtipos. Sin embargo, ¿por qué San Juan tiene a su cargo el cuidado del equipo? En realidad no quiere conocer las características de cada artículo, por ejemplo la longitud de la cadena de cada ancla. Su objetivo es registrar los artículos y de qué tipo son, de tal manera que pueda determinar si falta alguno o si está dañado. Esto se puede hacer sin conservar los registros detallados de los subtipos particulares del equipo. Así, para este diseño, colocamos todos los tipos de equipo en la entidad EQUIPO.

➤ FIGURA 3-20

Entidades para Paseos en barco San Juan

ARRENDAMIENTO
BARCO
CLIENTE
PROPIETARIO
EQUIPO
EQUIPO PROPORCIONADO POR EL PROPIETARIO
EQUIPO FIJO DEL PROPIETARIO
EQUIPO REMOVIBLE DEL PROPIETARIO
EQUIPO PROPORCIONADO POR SAN JUAN
ITINERARIO-CLIMA
PASEO
TRIPULACIÓN POR HORAS
MANTENIMIENTO PROGRAMADO
MANTENIMIENTO NO PROGRAMADO
REPARACIONES
INSTALACIONES PARA REPARACIONES

(a) Entidades posibles para Paseos en barco San Juan

ARRENDAMIENTO o PASEO(sinónimos)
BARCO
CLIENTE
PROPIETARIO
EQUIPO
ITINERARIO CLIMA
TRIPULACIÓN POR HORAS
MANTENIMIENTO PROGRAMADO
REPARACIONES o MANTENIMIENTO NO PROGRAMADO(sinónimos)
INSTALACIONES PARA REPARACIÓN

(b) Entidades seleccionadas para el diseño E-R

La propiedad del equipo se establece definiendo una relación entre EQUIPO y PROPIETARIO. Si se le permite a Paseos en barco San Juan ser una instancia de PROPIETARIO, todo el equipo que posee puede ser incluido en esta relación. De manera similar, como en el caso de la descripción, no hay motivo para definir de manera diferente al equipo que está fijo del que no lo está. Se puede elaborar una lista de datos sin esta división. La figura 3-20(b) muestra la lista final de entidades. Observe que ARRENDAMIENTO y PASEO son sinónimos, pues se refieren a la misma transacción. Aquí incluimos ambos para que puedan ser relacionados al caso de la descripción.

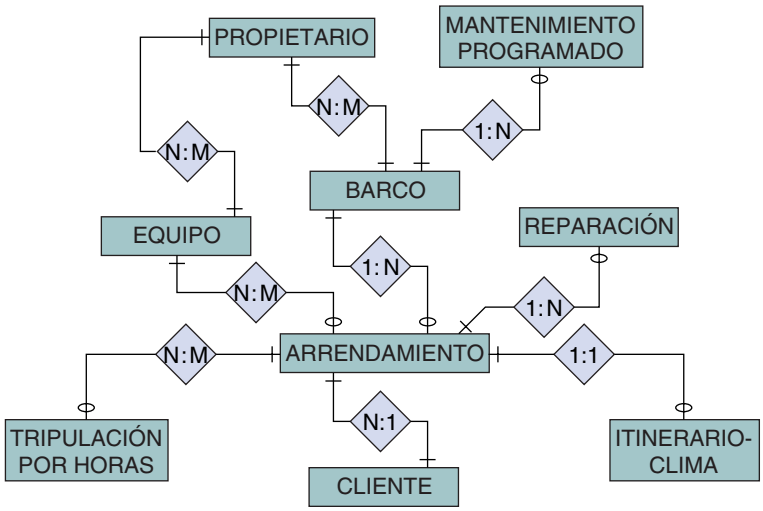
Es posible que MANTENIMIENTO PROGRAMADO deba combinarse con MANTENIMIENTO NO PROGRAMADO. Una forma para decidirlo, es examinar los atributos de cada una de estas entidades. Si son iguales, ambas deben unirse. También observe que REPARACIÓN y MANTENIMIENTO-NO-PROGRAMADO se definen como sinónimos.

Relaciones

La figura 3-21 es un diagrama entidad-relación de Paseos en barco San Juan. En su mayoría, las relaciones en este diagrama son claras, pero la relación entre EQUIPO y ARRENDAMIENTO es discutible. Uno puede decir que EQUIPO debería estar relacionado con BARCO y no con ARRENDAMIENTO, o que EQUIPO debería relacionarse con BARCO (el equipo que forma parte del barco) y el demás equipo debería estar relacionado con ARRENDAMIENTO. Estos cambios serían alternativas factibles para el diseño que se muestra en la figura 3-21.

➤ FIGURA 3-21

Diagrama E-R para Paseos en barco San Juan



Observe también que MANTENIMIENTO PROGRAMADO se relaciona con BARCO, pero que REPARACIÓN (MANTENIMIENTO NO PROGRAMADO) se relaciona con ARRENDAMIENTO. Esto implica que no se requiere ninguna acción de reparación cuando el barco no está rentado. Quizás esto no sea realista.

Finalmente, ARRENDAMIENTO y ITINERARIO-CLIMA tienen una relación 1:1 y también poseen los mismos atributos de identificación. Por lo tanto, sería posible, e incluso preferible, combinarlos en una clase de entidad.

➤ BASES DE DATOS COMO MODELOS DE MODELOS

Como puede ver, hay diferentes formas de modelar una situación de negocios, y la variedad es aún más grande conforme la aplicación se vuelve más compleja. Con frecuencia son factibles docenas de modelos y puede ser difícil elegir entre ellos.

A veces, cuando evalúan alternativas, los analistas del proyecto estudian y discuten qué modelo de datos representa mejor al mundo real. Estas discusiones están equivocadas. Las bases de datos no modelan al mundo real, aunque éste es un error común. Las bases de datos son modelos de los modelos del mundo de los usuarios (o, para ser más precisos, de su mundo de negocios). La pregunta que debe hacerse al evaluar modelos de datos alternativos no es: "¿Este diseño representa exactamente al mundo real?", sino "¿Este diseño representa exactamente el modelo del medio ambiente de los usuarios?" El objetivo es desarrollar un diseño que corresponda a la concepción mental de los usuarios.

De hecho, Immanuel Kant y otros filósofos declararon que es imposible que los seres humanos construyan modelos de lo que realmente existe, y afirmaron que la esencia de las cosas es eternamente desconocida para los humanos.[8] Extendiendo estos argumentos a los sistemas computacionales, Winograd y Flores declararon que, en las sociedades, los humanos construyen sistemas de prueba que les permiten operar con éxito en el mundo. Un conjunto de pruebas no es un modelo al infinito de la realidad,

[8] En su libro *Prolegómenos para cualquier metafísica futura*, Kant afirmó que no podemos ir más allá de toda experiencia posible, ni formar un concepto definitivo de lo que las cosas en sí mismas pueden ser. Declaró que aún no estamos en libertad de abstenernos completamente de investigarlas, porque la experiencia nunca satisface por completo a la razón; pero al contestar las preguntas, nos remitimos más y más al pasado y no quedamos satisfechos de la solución total. Immanuel Kant, *Prolegomena to Any Future Metaphysics* (Indianápolis: Bobbs-Merril, 1950), pág. 100.

sino, más bien, sólo un sistema social que permite a los usuarios coordinar sus actividades con éxito, y no se puede decir nada más.[9]

Por lo tanto, los sistemas de computación necesitan modelar y representar las comunicaciones de sus usuarios con otras personas. No modelan nada más que el sistema de pruebas y comunicaciones. Así que aprenda a preguntarse: "¿Este modelo refleja exactamente las percepciones de los usuarios y los modelos mentales de su mundo? ¿Los ayudará a responderse consistente y eficazmente el uno al otro, así como a sus clientes?" No tiene sentido hacer un análisis para reivindicar que su modelo es una mejor representación de la realidad. Lo importante es desarrollar uno que represente bien el modelo del medio ambiente de negocio del usuario.

➤ RESUMEN

El modelo entidad-relación fue desarrollado por Peter Chen. Con ese modelo, las entidades se definen como cosas identificables y de importancia para los usuarios. Todas las entidades de cierto tipo forman una clase de entidad. Una entidad particular se denomina instancia. Las entidades tienen atributos que describen sus características, y uno o más atributos identifican a una entidad.

Las relaciones son asociaciones entre entidades. El modelo E-R explícitamente define relaciones; cada relación tiene un nombre, y existen clases de relaciones así como entidades de relaciones. Las relaciones pueden tener atributos.

El grado de la relación es el número de entidades que participan en ésta. La mayoría son binarias. Los tres tipos de relaciones binarias son 1:1, 1:N y N:M.

En los diagramas entidad-relación las entidades se muestran en rectángulos y las relaciones en diamantes. La cardinalidad máxima se muestra dentro del diamante; la mínima se indica por medio de una línea cortada o de un óvalo. Las relaciones que conectan ejemplos de entidades de la misma clase son recursivas. Los atributos se pueden mostrar en un diagrama E-R en elipses, o en una tabla por separado.

Una entidad débil es la que depende de la existencia de otra; una entidad que no es débil se llama entidad fuerte. Las entidades débiles se muestran en rectángulos con las esquinas redondeadas, y la relación de la cual depende la entidad se indica por medio de un diamante con las esquinas redondeadas. En este texto definimos a una entidad débil como aquella que depende lógicamente de otra. Una entidad puede tener una cardinalidad mínima de 1 en una relación con otra entidad y no necesariamente es una entidad débil. Los atributos multivalor se representan con entidades débiles.

Algunas entidades tienen subtipos que definen subconjuntos de entidades similares. Los subtipos heredan atributos de su entidad padre, el supertipo. Las relaciones TIENE-UN conectan entidades de diferentes tipos y los identificadores de las entidades son diferentes. Las relaciones ES-UN son subtipos y los identificadores de las entidades son los mismos.

Cuando se ha desarrollado un modelo de datos, los diseñadores deben considerar las reglas del negocio que pueden limitar el procesamiento contra las entidades. Cada entidad en el modelo debería ser evaluada a la luz de posibles adiciones, cambios y eliminaciones de datos. En particular, con frecuencia las eliminaciones son fuente de importantes restricciones para el procesamiento. Cuando se descubren las reglas del negocio deben ser documentadas en el modelo de datos.

El modelo E-R es una parte importante de muchos productos CASE, los cuales proporcionan herramientas para la construcción y el almacenamiento de diagramas E-R. Algunas herramientas CASE integran las construcciones E-R con estructuras de datos en los repositorios CASE. El Lenguaje de modelación unificado (UML, por sus siglas en inglés) ha definido un nuevo estilo de diagramas entidad-relación. Le convendría familiarizarse con los diagramas de ese estilo; pero también debe darse cuenta de que cuan-

[9] Terry Winograd y F. Flores, *Understanding Computers and Cognition* (Reading, MA: Addison-Wesley, 1986).

do se crea el diseño de una base de datos no hay una diferencia fundamental entre el estilo tradicional y el UML.

Una vez terminados, los modelos E-R deben ser evaluados. Una técnica es enumerar las consultas que se podrían responder utilizando el modelo de datos. Después, mostrar esta lista a los usuarios, a los cuales se les pedirá que piensen en preguntas adicionales. Entonces, se evalúa el diseño contra esas preguntas para asegurarse de que el modelo pueda responderlas.

Las bases de datos no modelan el mundo real, sino el modelo del mundo de negocios de los usuarios. El criterio apropiado para juzgar un modelo de datos es si corresponde o no al modelo de los usuarios. No tiene caso discutir cuál es el que se adapta mejor al mundo real.

➤ PREGUNTAS DEL GRUPO I

3.1 Defina *entidad* y dé un ejemplo.

3.2 Explique la diferencia entre una clase de entidad y una instancia de entidad.

3.3 Defina *atributo* y dé ejemplos para la entidad que describió en la pregunta 3.1.

3.4 Explique qué es un atributo compuesto y dé un ejemplo.

3.5 En su respuesta a la pregunta 3.3, ¿qué atributo identifica a la entidad?

3.6 Defina *relación* y dé un ejemplo.

3.7 Explique la diferencia entre una clase de relación y una entidad de relación.

3.8 Defina *grado de relación*. Proporcione un ejemplo diferente al de este texto con respecto a una relación con un grado mayor que 2.

3.9 Enumere y dé un ejemplo sobre los tres tipos de relaciones binarias. Dibuje un diagrama E-R para cada uno.

3.10 Defina los términos *cardinalidad máxima* y *cardinalidad mínima*.

3.11 Nombre y dibuje los símbolos utilizados en los diagramas entidad-relación para: (a) entidad, (b) relación, (c) entidad débil y su relación, (d) relación recursiva, y (e) entidad subtipo.

3.12 Mencione un ejemplo de diagrama E-R para las entidades DEPARTAMENTO y EMPLEADO, las cuales tengan una relación 1:N. Suponga que un DEPARTAMENTO no necesita tener algún EMPLEADO, pero que cada EMPLEADO debe tener un DEPARTAMENTO.

3.13 Dé un ejemplo referente a una relación recursiva y muéstrelo en un diagrama E-R.

3.14 Muestre ejemplos de atributos para DEPARTAMENTO y EMPLEADO (con base en la pregunta 3.12). Utilice los símbolos de estilo UML.

3.15 Defina el término *entidad débil* y proporcione un ejemplo sobre alguno que no se mencione en este libro.

3.16 Explique la ambigüedad en la definición del término *entidad débil*. Explique qué interpretación se le da a dicho término en este texto. Dé ejemplos, diferentes a los del libro, sobre cada tipo de entidad débil.

3.17 Defina el término *entidad dependiente de un identificador* (ID-dependiente) y proporcione algún otro ejemplo diferente a los que ya se mencionaron.

3.18 Muestre cómo usar una entidad débil para representar el atributo multivalor para Habilidad en una entidad EMPLEADO. Indique las cardinalidades máxima y mínima en ambos lados de la relación. Use los símbolos tradicionales.

3.19 Muestre cómo usar una entidad débil para representar el atributo compuesto multivalor Teléfono, que contiene los atributos de un solo valor CódigodeÁrea, NúmeroTelefónico. Suponga que Teléfono aparece en una entidad llamada VENDEDOR.

Indique las cardinalidades máxima y mínima en ambos lados de la relación. Utilice los símbolos de estilo UML.

3.20 Describa las entidades de subtipos y dé algún ejemplo diferente a los que están en este texto.

3.21 Explique el término *herencia* y muestre cómo se aplica a su respuesta de la pregunta 3.20.

3.22 Explique la diferencia entre una relación TIENE-UN y una relación ES-UN, y dé un ejemplo sobre cada una.

3.23 ¿Cómo son abordadas las reglas del negocio en un modelo E-R?

3.24 Describa por qué es importante evaluar un modelo de datos una vez que ha sido creado. Resuma una técnica para la evaluación de un modelo de datos y explique qué técnica se podría usar para evaluar el modelo de datos de la figura 3-21.

➤ PREGUNTAS DEL GRUPO II

3.25 Cambie el diagrama E-R en la figura 3-19 para incluir una entidad LECCIÓN. Permita a LECCIÓN PRIVADA y LECCIÓN EN GRUPO ser subtipos de LECCIÓN. Modifique las relaciones según sea necesario. Use los símbolos tradicionales.

3.26 Cambie el diagrama E-R en la figura 3-19 para excluir MAESTRO. Modifique la relación como sea necesario. Use los símbolos de estilo UML.

3.27 ¿Cuál de los modelos de la figura 3-19 y de sus respuestas a las preguntas 3.25 y 3.26 prefiere? Explique por qué.

3.28 Cambie el diagrama de la figura 3-21 para incluir los subtipos de equipo. Suponga que el equipo de Paseos en barco San Juan pertenece a ARRENDAMIENTO, y otro equipo a BARCO. Modele las diferencias entre BARCO relacionado con EQUIPO fijo de los barcos y el BARCO relacionado con EQUIPO que no sea fijo. ¿Qué beneficios agregan más complejidad a este modelo?

➤ PROYECTOS

A. Desarrolle un diagrama E-R para una base de datos que apoye las necesidades de seguimiento de la siguiente organización: la Agencia Metropolitana de Viviendas (AMV) es una organización no lucrativa que se dedica al desarrollo y mejoramiento de la vivienda de interés social. La AMV opera en un área metropolitana con aproximadamente 2.2 millones de habitantes.

La AMV conserva datos referentes a la ubicación, disponibilidad y condiciones de alojamiento con bajo costo en 11 diferentes zonas censadas en el área metropolitana. Dentro de los límites de estas zonas hay aproximadamente 250 edificios diferentes que proporcionan alojamiento barato. En promedio, cada edificio consta de 25 departamentos u otras unidades.

La AMV conserva datos acerca de cada zona censada, incluyendo los límites geográficos, ingresos medios de la población, servidores públicos, negocios destacados, principales inversionistas de conformidad con los atributos de esa zona, y otros datos demográficos y económicos. También mantiene una cantidad limitada de datos acerca de la criminalidad. Para cada edificio, la AMV almacena nombre, dirección, tamaño, nombre del o de los propietarios, dirección, nombre y dirección del o de los hipotecarios, renovaciones y reparaciones, y disponibilidad de instalaciones para personas discapacitadas. Además, la AMV conserva una lista por unidad dentro de cada edificio, incluyendo el tipo de unidad, tamaño, número de recámaras, número de baños, instalaciones en la cocina y el comedor, localización del edificio y cualquier observación especial. A la

AMV le gustaría conservar los datos referentes a las tasas promedio de ocupación por unidad, pero, a la fecha, no ha podido compilar o almacenar esos datos. Sin embargo, la AMV tiene información acerca de si una unidad está ocupada o no.

La AMV funciona como un centro de información de la vivienda y ofrece tres servicios básicos. Primero, trabaja con políticos, congresistas y grupos de abogados para apoyar la legislación que se encarga del desarrollo de la vivienda de interés social a través de incentivos fiscales, desarrollo de zonas preferenciales y otros incentivos legislativos. Para lograrlo, la AMV proporciona información acerca de la vivienda de bajo costo para los gobiernos estatales, del municipio y de la ciudad. Segundo, mediante discursos, seminarios, pláticas en convenciones y otras actividades de relaciones públicas, la AMV lucha por aumentar la conciencia de la comunidad acerca de la necesidad de que exista el alojamiento de interés social. Por último, la AMV proporciona información sobre la disponibilidad de la vivienda de interés social a otras agencias que trabajan con comunidades de bajos ingresos y con población sin hogar.

B. Ingrese al sitio Web de un fabricante de computadoras, por ejemplo Dell (www.dell.com). Use el sitio Web para determinar qué computadora portátil podría comprar para un usuario que tenga un presupuesto de $10000. Cuando esté en el sitio Web, piense acerca de la estructura de una posible base de datos de sistemas y subsistemas de computación para apoyar ese sitio.

Desarrolle un diagrama E-R de una base de datos de sistemas y subsistemas computacionales para ese sitio Web. Muestre todas las entidades y relaciones cuando menos con dos o tres atributos por entidad. Indique las cardinalidades máximas y mínimas a ambos lados de cada relación. Las posibles entidades son SISTEMA-BASE, OPCIÓN-MEMORIA, TARJETA-DE-VÍDEO, e IMPRESORA. Por supuesto, hay una gran variedad de posibles entidades. Modele cualquier atributo multivalor como se muestra en el texto. Use los subtipos donde sea apropiado. Para evitar que este proyecto se extienda demasiado, limite el diseño a las necesidades de alguien que esté decidiendo una compra.

C. Ingrese al sitio Web de un vendedor de libros, por ejemplo Amazon (www.amazon.com). Use el sitio Web para determinar los tres mejores libros de XML (siglas en inglés de *Extended Markup Language*, o Lenguaje extendido de marcaje) para alguien que apenas está aprendiendo sobre el tema. Cuando visite el sitio Web piense en la estructura de una posible base de datos de libros, autores, temas y temas afines.

Desarrolle un diagrama E-R de una base de datos de libros para este sitio Web. Muestre todas las entidades y relaciones y cuando menos dos o tres atributos por entidad. Indique las cardinalidades máximas y mínimas para ambos lados de cada relación. Las posibles entidades son TÍTULO, AUTOR, EDITORIAL, COPIA y TEMA. Por supuesto, hay muchas más entidades posibles. Modele cualquier atributo multivalor como se muestra en el texto. Use subtipos donde sea apropiado. Para evitar que este proyecto se expanda demasiado, suponga que sólo les dará seguimiento a los libros. Además, limite su diseño a las necesidades de alguien que está buscando libros que quiere comprar. No considere pedido del cliente, entrega del pedido, orden de compra y otros procesos de negocios.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Considere la situación de FiredUp que analizamos al final de los capítulos 1 y 2. Suponga que FiredUp ahora ha desarrollado una línea de tres estufas diferentes: FiredNow, FiredAlways y FiredAtCamp. Además, suponga que los propietarios están vendiendo refacciones de cada estufa y que también las reparan. Algunas reparaciones son gratuitas porque están dentro del periodo de garantía de la estufa; en otras sólo se cobran las refacciones, y en otras más se cobran las refacciones y la mano de obra. FiredUp quiere

tener el seguimiento de todos estos datos. Cuando se solicitaron mayores detalles a los propietarios, hicieron la siguiente lista:

CLIENTE: Nombre, Dirección, Calle, NúmerodeDepartamento, Ciudad, Estado/Provincia, CP, País, CorreoElectrónico, NúmeroTelefónico

ESTUFA: NúmerodeSerie, Tipo, FechadeFabricación, ClavedelInspector

FACTURA: NúmerodeFactura, Fecha, Cliente, con una lista de artículos y precios a los que fueron vendidos, PrecioTotal

REPARACIÓN: NúmerodeReparación, Cliente, Estufa, Descripción, con una lista de artículos que fueron utilizados en la reparación y el costo de éstos, cuando proceda, y CantidadTotal de la reparación

PARTE: Número, Descripción, Costo, PreciodeVenta

A. Elabore un diagrama entidad-relación de una base de datos para FiredUp. Establezca las cardinalidades máxima y mínima de las relaciones entre entidades, según lo considere apropiado. Explique su razonamiento para cada valor de cardinalidad. Use las entidades débiles como lo juzgue conveniente. No use subtipos. Mencione cualquiera de las entidades dependientes de un identificador, si las hay.

B. Modifique su diagrama entidad-relación en su respuesta a la pregunta A mediante la representación de una FACTURA y REPARACIÓN con los subtipos apropiados. ¿Bajo qué circunstancias es mejor este diseño que el de la respuesta a la pregunta A?

C. Suponga que FiredUp quiere dar seguimiento a la casa, fax y números de teléfono celular, así como a múltiples direcciones de correo electrónico para cada uno de sus clientes. Modifique su diagrama E-R para permitir valores múltiples como NúmeroTelefónico y CorreoElectrónico.

D. Suponga que FiredUp desarrolla diferentes versiones del mismo producto de estufa. Desarrolle la versión 1 de FiredNow, la versión 2 de FiredNow, y así sucesivamente. Modifique su diagrama entidad-relación de la pregunta A anterior como sea necesario para considerar esta situación.

E. Cuando se les pregunta a los usuarios a qué datos quieren dar seguimiento, no necesariamente recuerdan todas sus necesidades. Usando sus conocimientos sobre pequeñas operaciones de negocios, haga una lista de las entidades que ellos puedan haber olvidado. Muestre el potencial de las relaciones entre estas entidades en un diagrama E-R. ¿Cómo determinaría qué datos adicionales de FiredUp se necesitan?

CAPÍTULO 4

El modelo de objeto semántico

El presente capítulo analiza el modelo de objeto semántico, el cual, al igual que el modelo E-R que estudiamos en el capítulo 3, se utiliza para crear modelos de datos. Como se muestra en la figura 4-1, el equipo analista entrevista a los usuarios y analiza sus reportes, formas y consultas, y a partir de eso construye un modelo de los datos que proporcionan los usuarios. Este modelo de datos se transforma posteriormente en el diseño de una base de datos.

La forma particular del modelo de datos depende de las estructuras que se utilizaron para construirlo. Si se utilizó un modelo E-R, éste tendrá entidades, relaciones, etc. Si se empleó un modelo semántico, entonces tendrá objetos semánticos y construcciones relacionadas, las cuales analizaremos en este capítulo.

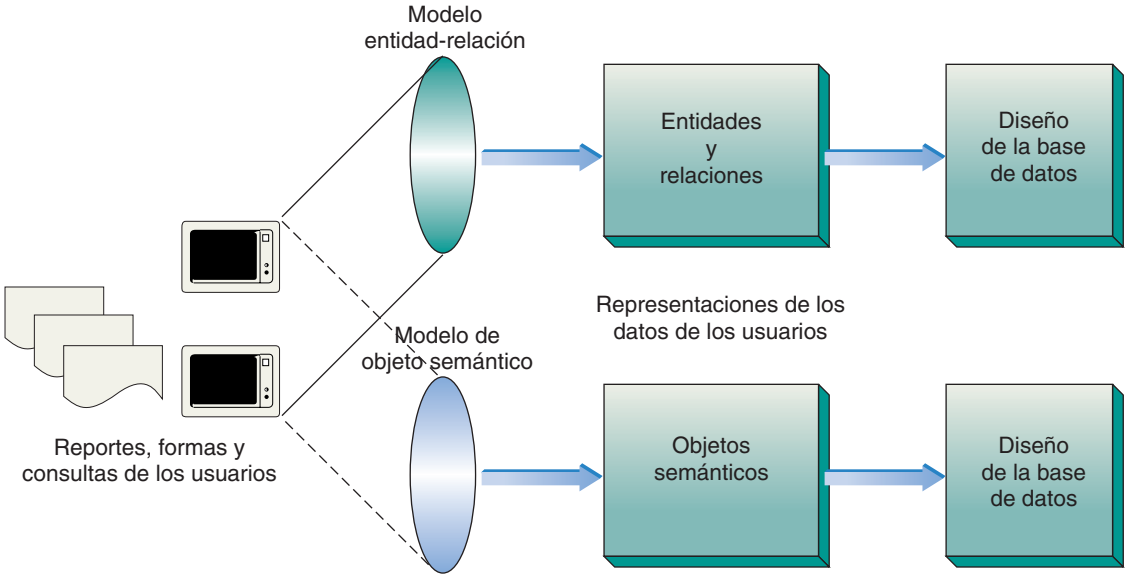
El modelo E-R y el modelo de objeto semántico son como lentes con los cuales los analistas de bases de datos observan mientras estudian y documentan los datos de los usuarios. Ambos lentes funcionan y finalmente dan como resultado el diseño de una base de datos. Los analistas utilizan distintos lentes para formar el diseño; sin embargo, y debido a que los lentes crean distintas imágenes, los diseños que producen no pueden ser exactamente los mismos. Cuando desarrolla una base de datos, usted debe determinar qué enfoque usar, de la misma manera en que un fotógrafo necesita decidir qué lentes utilizará. Cada enfoque tiene fortalezas y debilidades, las cuales analizaremos al final de este capítulo.

El modelo de objeto semántico se presentó por primera vez en 1988, en la tercera edición de este libro. Dicho modelo está basado en conceptos que desarrollaron y publicaron Codd y Hammer y McLeod.[1] El modelo de objeto semántico es un modelo de datos. Es diferente al **procesamiento de bases de datos orientadas a objetos,** el cual estudiaremos en el capítulo 18. Ahí usted aprenderá de qué manera los propósitos, las características y construcciones de la modelación de objeto semántico difieren del procesamiento de la base de datos orientada a objetos.

[1] E. F. Codd, "Extending the Relational Model to Capture More Meaning", *ACM Transactions on Database Systems*, diciembre de 1976, pp. 397-424; y Michael Hammer y Dennis McLeod, "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems*, septiembre de 1981, pp. 351-386.

➤ FIGURA 4-1

Uso de diferentes modelos para el diseño de bases de datos



➤ OBJETOS SEMÁNTICOS

El propósito de una aplicación de una base de datos es proporcionar formas, reportes y consultas, de tal manera que los usuarios puedan dar seguimiento a entidades u objetos que son importantes para su trabajo. Las metas para las primeras etapas del desarrollo de la base de datos consisten en determinar las cosas que serán representadas en ésta, con el fin de especificar las características de dichas cosas y establecer las relaciones entre todas ellas.

En el capítulo 3 nos referimos a estas cosas como entidades. En el presente capítulo les llamaremos **objetos semánticos,** o algunas veces sólo objetos. La palabra *semántico* quiere decir significado, y un objeto semántico es aquél que modela, en parte, el significado de los datos de los usuarios. Los objetos semánticos modelan las percepciones de los usuarios con mayor precisión que el modelo E-R. Utilizamos el adjetivo *semántico* con la palabra *objeto* para distinguir los objetos analizados en este capítulo de los objetos definidos en los lenguajes de programación orientada a objetos (OOP).

DEFINICIÓN DE OBJETOS SEMÁNTICOS

Las entidades y los objetos son similares en ciertos aspectos y diferentes en otros. Comenzaremos con las similitudes. Un objeto semántico es una representación de algunas cosas identificables en el ambiente de trabajo de los usuarios. De manera más formal, un objeto semántico es un *conjunto de atributos que describen suficientemente una identidad bien definida*.

Al igual que las entidades, los objetos semánticos se agrupan en clases. Una clase de objetos tiene un *nombre* que la distingue de otras y que corresponde a los nombres de las cosas que representa. Por consiguiente, una base de datos que apoya a los usuarios en su trabajo con registros de estudiantes tiene una clase de objetos llamada ESTUDIANTE. Observe que los nombres de la clase de objetos, al igual que los nombres de la clase de entidades, se escriben con letras mayúsculas. Un objeto semántico en particular es una instancia de la clase. Por lo tanto, "William Jones" es una instancia de la clase ESTUDIANTE y "Contabilidad" es una instancia de la clase DEPARTAMENTO.

Como sucede con las entidades, un objeto tiene un *conjunto de atributos*. Cada atributo representa una característica de la entidad que se representa. Para fines de este ejemplo, el objeto ESTUDIANTE puede tener atributos como Nombre, DirecciónParticular,

DirecciónDelCampus, FechadeNacimiento, FechadeGraduación, y Especialidad. Este conjunto de atributos también es una *descripción suficiente*, lo cual significa que los atributos representan todas las características que los usuarios necesitan para realizar su trabajo. Como establecimos al final del capítulo 3, las cosas en el mundo tienen un conjunto infinito de características; no es posible representarlas todas. En lugar de eso, representamos aquellas que son necesarias para que los usuarios satisfagan sus necesidades de información, de modo que puedan realizar con éxito su trabajo. La descripción suficiente también significa que los objetos son completos en sí mismos. Todos los datos que se requieren sobre un CLIENTE, por ejemplo, se localizan en el objeto CLIENTE, así que no necesitamos buscar en otro lado para encontrar esos datos.

Los objetos representan *entidades bien definidas*; esto es, son ciertas cosas que los usuarios reconocen como independientes y separadas a las que desean dar seguimiento, y a partir de eso elaborar reportes. Estas entidades son los nombres acerca de los cuales se va a producir la información. Para entender mejor el término *identidad bien definida*, recuerde la diferencia que hay entre los objetos y las instancias de los objetos. CLIENTE es el nombre de un objeto, y "CLIENTE 12345" es el nombre de una instancia de un objeto. Cuando decimos que un objeto representa una identidad bien definida, queremos decir que los usuarios consideran cada *instancia* de un objeto como única e identificable por derecho propio.

Finalmente, observe que las identidades que representan a los objetos pueden o no tener una existencia física. Por ejemplo, EMPLEADOs existe físicamente, pero PEDIDOs, no. Los PEDIDOs son, por sí mismos, modelos de un acuerdo contractual para proporcionar algunos bienes o servicios bajo ciertos términos y condiciones. No son objetos físicos, sino representaciones o acuerdos. Por consiguiente, alguna cosa no necesita ser física para que sea considerada como un objeto; sólo necesita ser identificable por derecho propio en la mente de los usuarios.

Atributos

Los objetos semánticos tienen atributos que definen sus características. Existen tres tipos de atributos: los **atributos simples** que tienen sólo un elemento, como por ejemplo FechadeContratación, NúmerodeFactura, TotaldeVentas; los **atributos grupales,** los cuales son combinaciones de otros atributos, como por ejemplo Dirección, que contiene los atributos {Calle, Ciudad, Estado, CódigoPostal}, o también NombreCompleto, el cual contiene los atributos {Nombre, InicialdelSegundoNombre, Apellido}, y los **atributos de los objetos semánticos,** que establecen una relación entre un objeto semántico y otro.

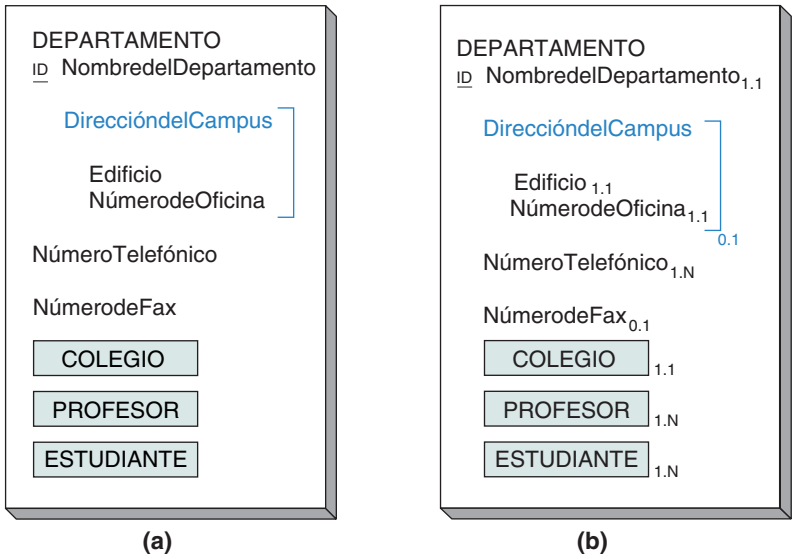
Para entender mejor estos enunciados observe la figura 4-2(a), la cual es un ejemplo de un **diagrama del objeto semántico,** o **diagrama del objeto.** Los equipos de analistas utilizan dichos diagramas para resumir las estructuras de los objetos y representarlos visualmente. Los objetos se muestran en rectángulos verticales. El nombre del objeto aparece en la parte superior y los atributos están escritos en orden después del nombre del objeto.

El objeto DEPARTAMENTO contiene un ejemplo de cada uno de los tres tipos de atributos. NombredelDepartamento, NúmeroTelefónico y NúmerodeFax son atributos simples, y cada uno representa sólo un elemento de datos. DirecciónDelCampus es un atributo grupal que contiene los atributos simples Edificio y NúmerodeOficina. Finalmente, COLEGIO, PROFESOR y ESTUDIANTE son atributos del objeto semántico, lo cual significa que esos objetos están conectados a DEPARTAMENTO y lógicamente contenidos en éste.

Los atributos del objeto o, como se les llama a veces, **enlaces del objeto,** significan que cuando un usuario piensa en un DEPARTAMENTO no sólo estará considerando NombredelDepartamento, DirecciónDelCampus, NúmeroTelefónico y NúmerodeFax, sino también COLEGIO, PROFESORes y ESTUDIANTEs que se relacionan con dicho departamento. Ya que COLEGIO, PROFESOR y ESTUDIANTE son objetos, el modelo de datos completo también contiene diagramas de objetos para éstos. El objeto COLEGIO contiene atributos del colegio; el objeto PROFESOR tiene atributos del profesorado, y el objeto ESTUDIANTE posee atributos de los estudiantes.

➤ FIGURA 4-2

Diagrama del objeto DEPARTAMENTO: (a) objeto DEPARTAMENTO y (b) objeto DEPARTAMENTO con cardinalidades



CARDINALIDAD DE LOS ATRIBUTOS. Cada atributo en un objeto semántico tiene una cardinalidad mínima y una cardinalidad máxima. La mínima indica la cantidad de instancias del atributo que deben existir para que el objeto sea válido; por lo general este número es 0 o 1. Si es 0 no se requiere que el atributo tenga un valor. Si es 1, debe tener un valor. Aunque es poco común, a veces la cardinalidad mínima puede ser mayor que 1. Por ejemplo, el atributo JUGADOR en un objeto llamado EQUIPO DE BÁSQUETBOL puede tener una cardinalidad mínima de 5, puesto que este es el número más pequeño de jugadores que se necesitan para formar un equipo.

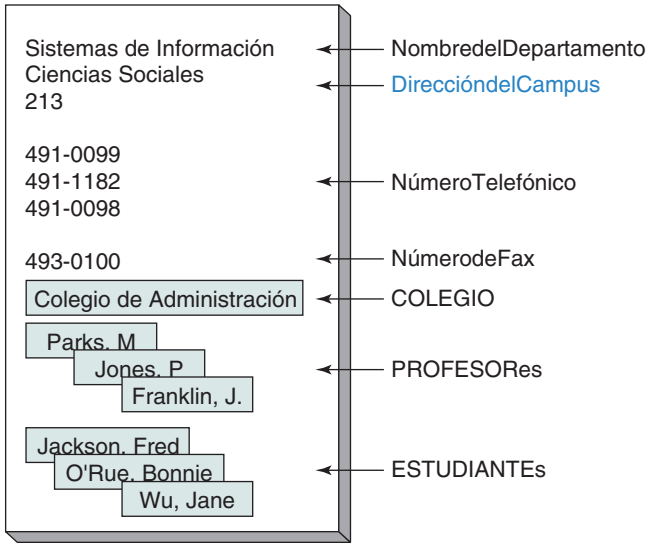
La cardinalidad máxima indica el número máximo de instancias del atributo que el objeto puede tener; por lo regular es 1 o N. Si es 1, el atributo no puede tener más de una instancia; si es N, el atributo puede tener muchos valores y el número absoluto no está especificado. Algunas veces la cardinalidad máxima es un número específico, como por ejemplo 5, lo cual significa que el objeto sólo puede contener exactamente cinco instancias del atributo. Por ejemplo, el atributo JUGADOR en EQUIPO DE BÁSQUETBOL puede tener una cardinalidad máxima de 15, lo cual indicaría que no se pueden asignar más de 15 jugadores a la lista de un equipo.

Las cardinalidades se muestran como subíndices de los atributos en el formato **N.M.,** donde N es la cardinalidad mínima y M la máxima. En la figura 4-2(b), la cardinalidad mínima de NombredelDepartamento es 1, y la máxima también es 1, lo cual significa que se requiere exactamente un valor de NombredelDepartamento. La cardinalidad de NúmeroTelefónico es 1.N, y esto significa que se requiere un DEPARTAMENTO para tener cuando menos un NúmeroTelefónico, aunque puede tener muchos. La cardinalidad de 0.1 en NúmerodeFax quiere decir que un DEPARTAMENTO puede tener cero o un NúmerodeFax.

Las cardinalidades de los grupos y sus atributos pueden ser muy sutiles. Considere el atributo DirecciónCampus. Sus cardinalidades son 0.1, lo cual significa que un DEPARTAMENTO no necesita tener una dirección, o cuando mucho tener una. Ahora, examine los atributos dentro de DirecciónCampus. Tanto Edificio como NúmerodeOficina tienen las cardinalidades 1.1. Probablemente usted se está preguntando cómo un grupo puede ser opcional si en dicho grupo se requieren los atributos. La respuesta es que las cardinalidades operan sólo entre el atributo y el poseedor de éste. La cardinalidad mínima de DirecciónCampus indica que no necesita existir un valor para la dirección en DEPARTAMENTO. Pero las cardinalidades mínimas de Edificio y NúmerodeOficina indican que tanto Edificio como NúmerodeOficina deben existir en DirecciónCampus. Por lo tanto, no necesita aparecer un grupo de DirecciónCampus, pero si aparece uno debe tener un valor tanto para Edificio como para NúmerodeOficina.

➤ FIGURA 4-3

Una instancia del objeto DEPARTAMENTO en la figura 4-2



INSTANCIAS DE OBJETOS. Los diagramas de los objetos para DEPARTAMENTO que se muestran en la figura 4-2 son un formato, o una estructura general, que puede utilizarse para cualquier departamento. Una instancia del objeto DEPARTAMENTO se muestra en la figura 4-3, con cada valor del atributo para un departamento en particular. El NombredelDepartamento es Sistemas de Información y se localiza en la Oficina 213 del Edificio de Ciencias Sociales. Observe que existen tres valores para NúmeroTelefónico, debido a que el Departamento de Sistemas de Información tiene tres líneas telefónicas en esa oficina. Otros departamentos pueden tener menos o más, pero cada uno tiene cuando menos una.

Además, existe una instancia de COLEGIO, el Colegio de Administración, y existen valores múltiples para los atributos de los objetos PROFESOR y ESTUDIANTE. Cada uno de estos atributos de los objetos es un objeto completo y tiene todos los atributos definidos de un objeto de tal tipo. Para que este diagrama siga siendo simple sólo se muestran los nombres de identificación en cada una de las instancias del atributo del objeto.

Un diagrama del objeto es una representación de la percepción del usuario con respecto a un objeto de su ambiente de trabajo. Por lo tanto, en la mente del usuario el objeto DEPARTAMENTO incluye todos estos datos. Un DEPARTAMENTO lógicamente contiene datos sobre el COLEGIO en el que reside, así como de los PROFESORes y ESTUDIANTEs que se relacionan con dicho departamento.

ATRIBUTOS PAREADOS. El modelo de objeto semántico no tiene ninguna relación de objetos independientes en un solo sentido. Si un objeto contiene a otro, el segundo abarcará al primero. Por ejemplo, si DEPARTAMENTO contiene el atributo de objeto COLEGIO, entonces COLEGIO contendrá al atributo de objeto asociado DEPARTAMENTO. Estos atributos de objetos son denominados **atributos pareados,** puesto que siempre existen en pareja.

¿Por qué deben ser pareados los atributos de los objetos? La respuesta radica en la manera en que los seres humanos consideran las relaciones. Si el Objeto A tiene una relación con el Objeto B, entonces el Objeto B tendrá una relación con el Objeto A. Por lo menos, B se relaciona con A en la relación de "cosas que se relacionan con B". Si este argumento no le resulta claro, trate de imaginarse una relación independiente entre dos objetos. Eso es imposible.

Identificadores de objetos

Un **identificador de objetos** es uno o más de los atributos de los objetos que los usuarios emplean para identificar las instancias de objetos. Dichos identificadores son nombres potenciales para un objeto semántico. Por ejemplo, en CLIENTE, los posibles identificadores son IdentificacióndelCliente y NombredelCliente. Cada uno es un atri-

buto que los usuarios consideran como nombre válido de las instancias CLIENTE. Compare estos identificadores con atributos como FechadelPrimerPedido, PreciodelaMercancía y NúmerodeEmpleados. Dichos atributos no son identificadores porque los usuarios no piensan en ellos como nombres de las instancias CLIENTE.

Un **identificador de grupo** es aquel que tiene más de un atributo. Algunos ejemplos son {Nombre, Apellido}, {Nombre, NúmeroTelefónico} y {Estado, NúmerodeLicencia}.

Los identificadores de grupo pueden ser únicos o no, dependiendo de cómo vean sus datos los usuarios. Por ejemplo, NúmerodeFactura es un identificador único de PEDIDO, pero NombredelEstudiante no es un identificador único de ESTUDIANTE. Por ejemplo, puede haber dos estudiantes llamadas "Mary Smith". Si es así, los usuarios emplearán el NombredelEstudiante para identificar a un grupo de uno o más estudiantes y entonces, si es necesario, utilizarán valores de otros atributos para identificar a un miembro de dicho conjunto en particular.

En los diagramas de objeto semántico, los identificadores de objetos están representados por las letras *ID* junto al atributo. Si el identificador es único, estas letras aparecerán subrayadas. Por ejemplo, en la figura 4-2(b), el atributo NombredelDepartamento es un identificador único de DEPARTAMENTO.

Normalmente, si se va a utilizar un atributo como identificador se requerirá su valor. Asimismo, por lo general no existe más que un valor de un atributo identificador para un objeto determinado. Por lo tanto, en la mayoría de las instancias la cardinalidad de un atributo de identificación (ID) es 1.1, y por consiguiente usaremos este valor como predeterminado.

Sin embargo, hay casos (relativamente pocos), en los cuales la cardinalidad de un identificador es diferente a 1.1. Por ejemplo, considere el atributo Alias (apodo) en el objeto semántico PERSONA. Una persona no necesita tener un alias, o probablemente él o ella tengan diversos alias. De ahí que la cardinalidad de Alias sería 0.N.

Al mostrar los subíndices de todos los atributos se desordenaría el diagrama de objeto semántico. Para simplificar, supondremos que las cardinalidades de los atributos identificadores con valores simples son 1.1 y las cardinalidades de otros atributos con valores simples son 0.1. Si la cardinalidad del atributo con valor simple es diferente a estas suposiciones, lo mostraremos en el diagrama; de lo contrario, omitiremos los subíndices de los atributos con valores simples.

Dominios de atributos

El **dominio** de un atributo es una descripción de los posibles valores de éste. Las características de un dominio dependen del tipo de atributo. El dominio de un atributo simple consiste tanto en una descripción física como en una descripción semántica. La descripción física indica el tipo de datos (por ejemplo, numéricos contra cadenas), su longitud y otras restricciones o condiciones (por ejemplo, que el primer carácter deba ser alfabético, o que el valor no exceda de 9999.99). La descripción semántica indica la función o el propósito del atributo; esto lo distingue de otros atributos que puedan tener la misma descripción física.

Por ejemplo, el dominio de NombredelDepartamento puede definirse como "el conjunto de cadenas con más de siete caracteres que representan los nombres de los departamentos de la Universidad Highline". La frase *cadenas con más de siete caracteres* es la descripción física del dominio, y la frase *que representa los nombres de los departamentos de la Universidad Highline* es la descripción semántica. La descripción semántica diferencia a las cadenas de siete caracteres que representan los nombres de los departamentos, de las cadenas similares que representan, por así decirlo, nombres de cursos, edificios o algún otro atributo.

En algunos casos, la descripción física del dominio de un atributo simple es una **lista enumerada,** es decir, el conjunto de valores específicos de un atributo. Por ejemplo, el dominio del atributo Color podría ser la lista enumerada {"Azul", "Amarillo", "Rojo"}.

El dominio de un atributo grupal también tiene descripciones física y semántica. La primera es una lista de todos los atributos en el grupo y el orden que ocupan. La descrip-

ción semántica es la función o el propósito del grupo. De esta manera, la descripción física de dominio DirecciónCampus (en la figura 4-2) es la lista {Edificio, NúmerodeOficina}; la descripción semántica es *la localización de una oficina en la Universidad Highline*.

El dominio del atributo de un objeto es el conjunto de las instancias del objeto de este tipo. Por ejemplo, en la figura 4-2 el dominio del atributo del objeto PROFESOR es el conjunto de todos los ejemplos del objeto PROFESOR en la base de datos. El dominio del objeto COLEGIO es el conjunto de todos los COLEGIOs en la base de datos. En cierta forma, el dominio del atributo de un objeto es una lista dinámicamente enumerada, la cual contiene todas las instancias de determinado tipo de objeto.

Vistas de objetos semánticos

Los usuarios tienen acceso a los valores de los atributos del objeto a través de las aplicaciones de la base de datos que proporciona formas de entrada de datos, reportes y consultas. En la mayoría de las instancias dichas formas, reportes y consultas no requieren que se tenga acceso a todos los atributos de un objeto. Por ejemplo, la figura 4-4 muestra dos vistas de aplicación de DEPARTAMENTO. Algunos atributos de DEPARTAMENTO (por ejemplo, NombredelDepartamento) son visibles en ambas vistas de aplicación, otros atributos sólo son visibles en una vista. Por ejemplo, ESTUDIANTE sólo es visible en la vista de ListadeEstudiantes; pero PROFESOR, sólo en la Vista de Personal.

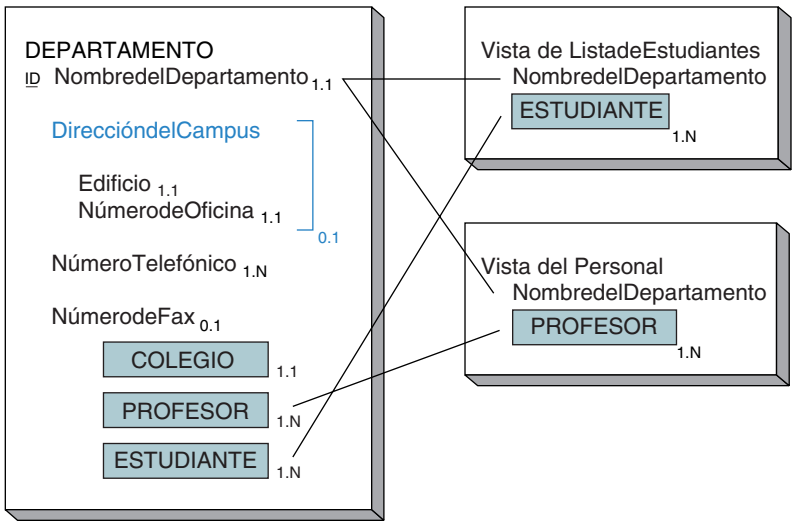
La parte de un objeto que está visible en una aplicación en particular se denomina **vista del objeto semántico,** o simplemente **vista.** Consta del nombre del objeto más una lista de todos los atributos visibles.

Las vistas se utilizan de dos maneras. Cuando usted crea una base de datos, puede emplearlas para desarrollar el modelo de datos. Observe nuevamente la figura 4-1. Cuando crean el modelo de datos, los analistas de la base de datos y de las aplicaciones trabajan de manera retrospectiva; esto es, comienzan con las formas, reportes y consultas que los usuarios dicen necesitar, y por lo tanto trabajan de manera inversa en el diseño de la base de datos. Para hacer esto, el equipo selecciona una forma, reporte o consulta y determina la vista que debe haber para que éstos se creen. Posteriormente, el equipo selecciona la siguiente forma, reporte o consulta y hace lo mismo. Entonces se integran las dos vistas. Este proceso se repite hasta que la estructura de toda la base de datos haya sido creada.

La segunda manera en la que se utilizan las vistas se presenta después de que la estructura de la base de datos ha sido creada. En este punto, se construyen vistas para dar soporte a las nuevas formas, reportes y consultas basadas en la estructura de la base de datos. Algunos ejemplos de este segundo uso los veremos en la Parte IV en la que analizaremos el SQL Server y Oracle.

➤ FIGURA 4-4

Vistas de ListadeEstudiantes y de Personal del objeto semántico DEPARTAMENTO



➤ CREACIÓN DE MODELOS DE DATOS CON OBJETOS SEMÁNTICOS

Esta sección ilustra un proceso para desarrollar objetos semánticos, en el cual los analistas examinan la interfaz de aplicación —formas, reportes y consultas— y trabajan de manera retrospectiva (ingeniería de reversa) con el fin de derivar la estructura del objeto. Por ejemplo, para modelar la estructura del objeto DEPARTAMENTO primero reunimos todos los reportes, formas y consultas basadas en DEPARTAMENTO. A partir de éstos, definimos el objeto DEPARTAMENTO el cual permite que se construyan dichos formatos y consultas.

Sin embargo, para una aplicación totalmente nueva no se examinarán reportes, formas o consultas computacionales. En este caso, los analistas comienzan por determinar a qué objetos necesitan dar seguimiento los usuarios. Posteriormente, mediante entrevistas con ellos, el equipo determina qué atributos del objeto son importantes. A partir de ahí, se pueden construir prototipos de formas o reportes que después se utilizarán para perfeccionar el modelo de datos.

UN EJEMPLO: BASE DE DATOS ADMINISTRATIVA DE LA UNIVERSIDAD HIGHLINE

Suponga que la administración de la Universidad Highline quiere dar seguimiento a los datos referentes a departamento, facultad y especialidad de cada estudiante. Además, suponga que la aplicación necesita producir cuatro reportes (figuras 4-5, 4-7, 4-9 y 4-11). Nuestro objetivo es examinarlos y mediante ingeniería de reversa determinar los objetos y atributos que deben ser almacenados en la base de datos.

EL OBJETO COLEGIO. El ejemplo de reporte en la figura 4-5 se refiere específicamente a un colegio: el de Administración. Éste es sólo un ejemplo del reporte; la Universidad Highline tiene reportes similares de otros colegios, como el de Artes y Ciencias y el Colegio de Ciencias Sociales. Cuando se crea un modelo de datos es importante reunir suficientes ejemplos para formar uno que sea representativo de todos los reportes. Aquí suponemos que el de la figura 4-5 es representativo.

Al examinar el reporte, encontramos datos específicos del colegio, tales como nombre, rector, número de teléfono y dirección del campus, así como información sobre cada uno de los departamentos. Esto *sugiere* que la base de datos puede contener los objetos COLEGIO y DEPARTAMENTO, con una relación entre ambos.

Esta información preliminar está documentada en los diagramas de los objetos en la figura 4-6. Observe que hemos omitido las cardinalidades de los atributos simples, los cuales tienen una cardinalidad de 0.1.

➤ FIGURA 4-5

Ejemplo del reporte COLEGIO

Colegio de Administración
Rector, Mary B. Jefferson

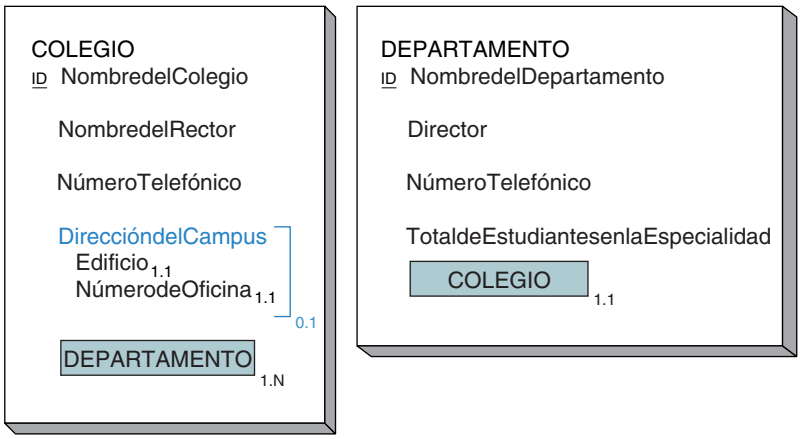
NúmeroTelefónico: 232-1187

Dirección del Campus:
Edificio de Administración, Oficina 100

Departamento	Director	Número Telefónico	Total de estudiantes en la especialidad
Contabilidad	Jackson, Seymour P.	232-1841	318
Finanzas	HeuTeng, Susan	232-1414	211
Sistemas de información	Brammer, Nathaniel D.	236-0011	247
Administración	Tuttle, Christine A.	236-9988	184
Producción	Barnes, Jack T.	236-1184	212

➤ FIGURA 4-6

Primera versión de los objetos COLEGIO y DEPARTAMENTO



La cardinalidad de DEPARTAMENTO dentro de COLEGIO es 1.N, lo cual indica que un COLEGIO debe tener cuando menos un DEPARTAMENTO y que puede tener varios. Esta cardinalidad mínima no puede deducirse a partir del reporte en la figura 4-5; por el contrario, se preguntó a los usuarios si podía existir un colegio sin departamentos, y su respuesta fue que no.

Asimismo, observe que la estructura de DEPARTAMENTO se infiere de los datos que se muestran en la figura 4-5. Ya que los atributos de los objetos siempre están pareados, COLEGIO se muestra en DEPARTAMENTO; sin embargo, estrictamente hablando, este hecho no puede determinarse a partir de la figura 4-5. Igual que con el atributo DEPARTAMENTO en COLEGIO, se pidió a los usuarios que determinaran las cardinalidades de ese atributo, que son 1.1, lo cual significa que un DEPARTAMENTO se debe relacionar únicamente con un COLEGIO.

Paralelamente, hemos interpretado el reporte en la figura 4-5 para indicar que los grupos de datos repetitivos se refieren a DEPARTAMENTO como objeto independiente. De hecho, a menudo estos grupos repetitivos son una señal de que existe otro objeto. *Sin embargo, éste no es siempre el caso*. El grupo repetitivo puede ser también un atributo grupal que se presenta para tener diversos valores.

Quizás usted se pregunte cuál es la diferencia entre los datos de objetos repetitivos y los de grupos repetitivos. No existe ninguna regla inflexible y automática, debido a que la respuesta depende de la manera en que los usuarios ven su mundo. Consecuentemente, el mejor enfoque es consultar a los usuarios sobre la semántica de los datos. Pregunte si estos datos de grupos repetitivos son sólo una parte del colegio, o si se refieren a algo más que existe por sí mismo. Si se trata del primer caso, constituyen un atributo grupal; pero si se trata del segundo caso, entonces estamos hablando de un objeto semántico. También busque otros reportes (o formas, o consultas). ¿Los usuarios tienen uno para los departamentos? Si es así, entonces se confirmaría la suposición de que DEPARTAMENTO es un objeto semántico. De hecho, el personal de Highline utiliza dos reportes referentes a DEPARTAMENTOs. Este hecho comprueba más adelante la noción de que debe definirse un objeto DEPARTAMENTO.

Asimismo, los grupos de atributos que representan un objeto independiente por lo general tienen atributos de identificación obvios. Los automóviles tienen un NúmerodeIdentificación o NúmerodeLicencia; los productos tienen un NúmerodeProducto o SKU. Los pedidos tienen un NúmerodePedido. Sin embargo, el grupo de atributos (FechadeMedición, PresióndelosNeumáticos) no tiene un identificador obvio. Cuando le pregunte a un usuario sobre dicho identificador de grupo, le responderá algo como: "¿presión de los neumáticos de qué?" Puede ser la presión de los neumáticos de un auto, de un autobús, de un remolque o de algún otro vehículo. Por lo tanto, dicho grupo sería un atributo grupal dentro de otro objeto —el cual constituye la respuesta a la pregunta "¿de qué?"

EL OBJETO DEPARTAMENTO. El reporte DEPARTAMENTO que se muestra en la figura 4-7 contiene datos departamentales, junto con una lista de los profesores que es-

➤ FIGURA 4-7

Ejemplo de reporte de DEPARTAMENTO

Departamento de Sistemas de Información
Colegio de Administración

Director: Brammer, Nathaniel D.
Teléfono: 236-0011
Dirección del Campus: Edificio de Ciencias Sociales, Oficina 213

Profesor	Oficina	Teléfono
Jones, Paul D.	Ciencias Sociales, 219	232-7713
Parks, Mary B.	Ciencias Sociales, 308	232-5791
Wu, Elizabeth	Ciencias Sociales, 207	232-9112

tán asignados a dichos departamentos. Observe que este reporte contiene datos referentes a la dirección del campus del departamento. Puesto que estos datos no aparecen en el objeto en la figura 4-6, necesitamos agregarlos al objeto DEPARTAMENTO, como se hizo en la figura 4-8. Este ajuste es común en el procesamiento de modelos de datos. Esto es, los objetos semánticos se ajustan continuamente como reportes, formas y consultas nuevos que serán identificados y analizados.

EL OBJETO PROFESOR. El reporte en la figura 4-7 no sólo indica que el objeto DEPARTAMENTO necesita ser modelado, también sugiere que se puede necesitar otro objeto para representar los datos del profesor. Por lo tanto, se le agregó al modelo un objeto PROFESOR, como se muestra en la figura 4-8. La identificación de PROFESOR, que es NombredelProfesor, no es única; esto se denota con el hecho de no subrayar ID en la figura 4-8.

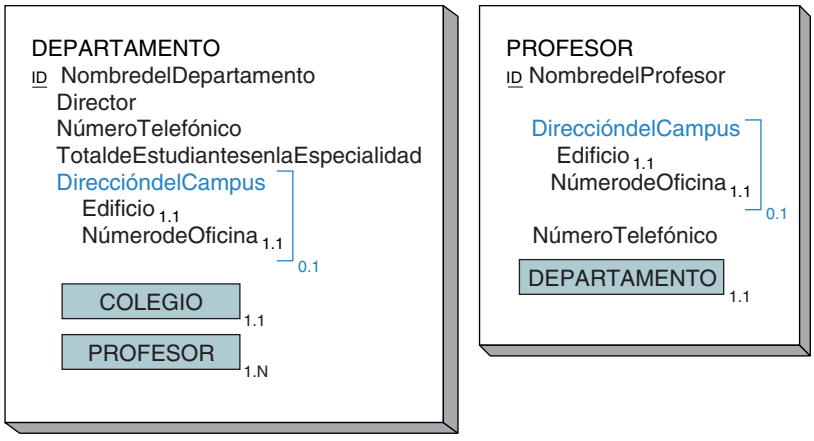
De acuerdo con los diagramas de objetos que se muestran en la figura 4-8, un DEPARTAMENTO debe tener cuando menos un PROFESOR y puede tener muchos profesores; pero un PROFESOR debe tener sólo un DEPARTAMENTO. Por consiguiente, de acuerdo con este modelo, se prohíben los nombramientos adjuntos. Esta restricción es parte de las reglas del negocio que se deben obtener de las entrevistas con los usuarios.

La figura 4-9 muestra un segundo reporte sobre un departamento, el cual se refiere a éste y a los estudiantes que cursan determinada especialidad. Es común tener dos reportes sobre un objeto, porque documentan puntos de vista diferentes sobre una misma cosa. Además, la existencia de este segundo reporte refuerza la noción de que departamento es un objeto en la mente de los usuarios.

EL OBJETO ESTUDIANTE. El reporte de la figura 4-9 proporciona datos sobre estudiantes que se especializan en una área del departamento, lo que implica que estos es-

➤ FIGURA 4-8

Objetos ajustados DEPARTAMENTO y NUEVO PROFESOR



➤ FIGURA 4-9

Segundo ejemplo de reporte de DEPARTAMENTO

Lista de estudiantes de especialidad
Departamento de Sistemas de Información

Director: Brammer, Nathaniel D. Teléfono: 236-0011

NombredelEstudiantedeEspecialidad	NúmerodeEstudiante	Teléfono
Jackson, Robin R.	12345	237-8713
Lincoln, Fred J.	48127	237-8713
Madison, Janice A.	37512	237-8713

tudiantes también son un objeto. El objeto DEPARTAMENTO debe contener un objeto ESTUDIANTE y un objeto PROFESOR, como se observa en la figura 4-10.

El objeto ESTUDIANTE en la figura 4-10 contiene los atributos NombredelEstudiante, NúmerodeEstudiante y NúmeroTelefónico, los cuales son los atributos listados en el reporte de la figura 4-9. Observe que NombredelEstudiante y NúmerodeEstudiante son identificadores. Los atributos NombredelEstudiante no son únicos, pero los de NúmerodeEstudiante sí lo son.

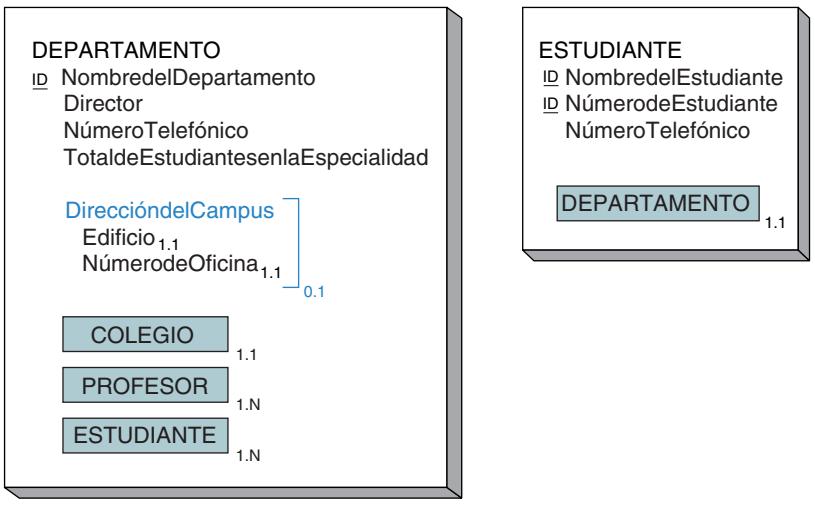
La figura 4-11 es un ejemplo de otro reporte sobre un estudiante —la carta de aceptación que la Universidad Highline envía a los aspirantes a ingresar—. Aunque es una carta, también es un reporte; probablemente la produjeron mediante una lista de correo con un procesador de textos.

Estos elementos de datos en la carta, que se deben almacenar en la base de datos, se muestran con letras negritas. Además de los datos referentes al estudiante, la carta también contiene datos sobre el DEPARTAMENTO de su especialidad y de su asesor. Puesto que un asesor es un PROFESOR, esta carta comprueba la necesidad de tener por separado un objeto PROFESOR. Los diagramas de los objetos revisados PROFESOR y ESTUDIANTE se muestran en la figura 4-12. De acuerdo con el objeto ESTUDIANTE, tanto DEPARTAMENTO como PROFESOR tienen un solo valor (una cardinalidad máxima de 1). Por tal razón, un estudiante tiene, cuando mucho, un departamento de especialidad y un asesor, y se requiere que tenga ambos.

El objeto ESTUDIANTE en la figura 4-12 corresponde a los datos que se muestran en la carta de la figura 4-11. Sin embargo, puede ser que el estudiante tenga en realidad más de una especialidad; en tal caso, DEPARTAMENTO y PROFESOR tendrían valores múltiples. Este hecho no puede determinarse sólo a partir de este formato de carta, por lo que se necesitan documentos y entrevistas adicionales para descubrir si se permiten varias especializaciones. Aquí supondremos que sólo se permite una.

➤ FIGURA 4-10

Objetos ajustados DEPARTAMENTO y nuevo ESTUDIANTE



➤ FIGURA 4-11

Carta de aceptación

Sr. Fred Parks
Calle Elm No. 123
Los Ángeles, California 98002

Estimado **Sr. Parks:**

Usted ha sido aceptado como estudiante de especialidad en el Departamento de **Contabilidad** de la Universidad Highline, a partir del **semestre de otoño** del **2002.** La oficina del Departamento de Contabilidad se localiza en el Edificio de **Administración**, oficina **210.**

Su asesora es la profesora **Elizabeth Johnson,** cuyo número telefónico es el **232-8740;** su oficina es la número **227** y se localiza en el edificio de **Administración.** Por favor haga una cita con ella tan pronto como llegue al campus.

¡Felicidades y bienvenido a la Universidad Highline!

Sinceramente,

Jan P. Smathers
Presidente

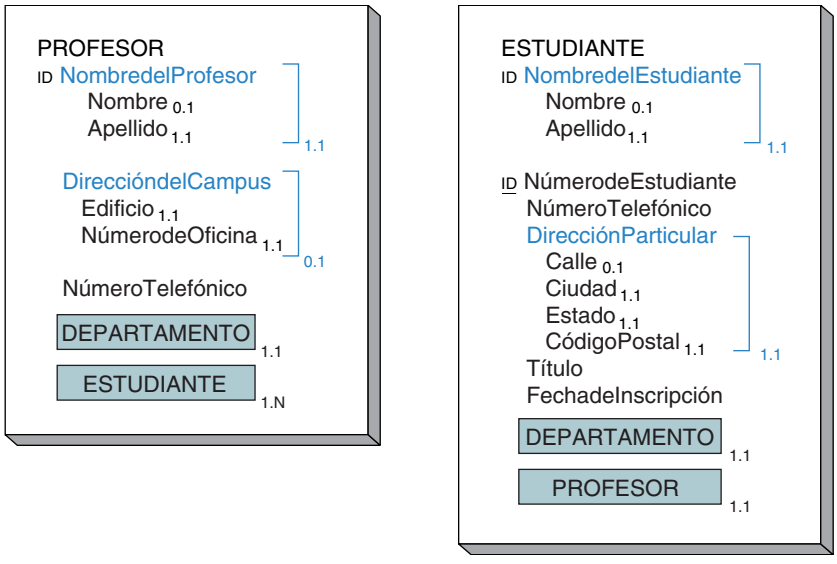
JPS/rkp

El formato de nombre del estudiante se proporciona primero en el de *nombre y apellido* en la parte superior de la carta, y después en el formato *apellido* en el saludo inicial. Si la presentación de los nombres en este formato es un requisito, entonces el atributo NombredelEstudiante (en la figura 4-10) no será suficiente y en su lugar se debe definir el grupo Nombre, Apellido. Esto se hizo en la figura 4-12. Asimismo observe que el nombre del asesor está en el formato {Nombre, Apellido}, lo cual significa que el nombre del PROFESOR debe cambiarse también.

Además, esta carta indica que los nombres en las direcciones y los saludos deben estar precedidos por el título "Sr." o "Srita." Si hay que hacer esto, debe colocarse un atributo adicional en ESTUDIANTE. Una alternativa es registrar el género del estu-

➤ FIGURA 4-12

Objetos ajustados PROFESOR y ESTUDIANTE



diante y seleccionar el título con base en este atributo. Otra alternativa es almacenar el título por sí mismo. La ventaja de esta segunda alternativa es que se pueden almacenar títulos diferentes a "Sr." y "Srita.", tales como "Dr.", "Ing.", etcétera.

Como generalmente se documenta, el modelo no requiere un título diferente a Sr. o Srita. No obstante, es probable que se puedan necesitar títulos adicionales; por lo tanto, la segunda alternativa parece más convincente y por esta razón en la figura 4-2 se ha agregado el atributo Título a ESTUDIANTE.

De nuevo, estos cambios ilustran la naturaleza repetitiva del modelo de datos. Las decisiones del diseño con frecuencia necesitan ser reflexionadas y revisadas muchas veces. Tal iteración no significa que el proceso de diseño sea deficiente, lo cual de hecho es muy común.

Especificación de objetos

La figura 4-13 muestra los diagramas de los objetos completos para la base de datos de la Universidad Highline. Se han realizado pocos cambios: NombredelRector y Director se han modelado en el formato {Nombre, Apellido} de manera que todos los nombres

FIGURA 4-13

Conjunto completo de diagramas de objetos semánticos

➤ FIGURA 4-14

Especificaciones de objetos para la base de datos de la Universidad Highline: (a) especificaciones de objetos semánticos, y (b) especificaciones de dominio

Nombre del objeto	Nombre de la propiedad	Card. mín.	Card. máx.	Estatus del ID	Nombre del dominio
COLEGIO	NombredelColegio	1	1	ID	NombredelColegio
	NombredelRector	1	1		NombredelaPersona
	Nombre	0	1		Nombre
	Apellido	1	1		Apellido
	NúmeroTelefónico	0	1		NúmeroTelefónico
	DirecciónCampus	0	1		DirecciónCampus
	Edificio	1	1		Edificio
	NúmerodeOficina	1	1		NúmerodeOficina
	DEPARTAMENTO	1	N		DEPARTAMENTO
DEPARTAMENTO	NombredelDepartamento	1	1	ID	NombredelDepartamento
	Director	1	1		NombredelaPersona
	Nombre	0	1		Nombre
	Apellido	1	1		Apellido
	NúmeroTelefónico	0	1		NúmeroTelefónico
	TotaldeEstudiantesenla-	0	1		TotaldeAlumnosenla-
	Especialidad		1		Especialidad
	DirecciónCampus	0	1		DirecciónCampus
	Edificio	1	1		Edificio
	NúmerodeOficina	1	1		NúmerodeOficina
	COLEGIO	1	N		COLEGIO
	PROFESOR	1	N		PROFESOR
	ESTUDIANTE	1			ESTUDIANTE
PROFESOR	NombredelProfesor	1	1	ID	NombredelaPersona
	Nombre	0	1		Nombre
	Apellido	1	1		Apellido
	DirecciónCampus	0	1		DirecciónCampus
	Edificio	1	1		Edificio
	NúmerodeOficina	1	1		NúmerodeOficina
	NúmeroTelefónico	0	1		NúmeroTelefónico
	DEPARTAMENTO	1	1		DEPARTAMENTO
	ASESOR	1	N		ESTUDIANTE
ESTUDIANTE	NombredelEstudiante	1	1	ID	NombredelaPersona
	Nombre	0	1		Nombre
	Apellido	1	1		Apellido
	NúmerodeEstudiante	1	1	ID	NúmerodeEstudiante
	NúmeroTelefónico	0	1		NúmeroTelefónico
	DirecciónParticular	1	1		Dirección
	Título	0	1		Título
	FechadeInscripción	0	1		FechadelTrimestre
	DEPARTAMENTO	1	1		DEPARTAMENTO
	ASESOR	1	1		PROFESOR

(a)

estén en un formato similar. Para mejorar la precisión del modelo, el atributo PROFESOR en ESTUDIANTE ha sido renombrado como ASESOR. La instancia PROFESOR que se conecta con una instancia ESTUDIANTE por medio de este atributo no es solamente cualquiera de los profesores del ESTUDIANTE, es un PROFESOR en particular que funge como asesor de ese ESTUDIANTE, por lo que el término *ASESOR* es más preciso que el de *PROFESOR*.

El dominio de este atributo es fijo. El dominio de ASESOR es PROFESOR, así como el dominio del atributo PROFESOR fue PROFESOR. Este atributo incluso señala o se conecta con las instancias del objeto semántico PROFESOR. El cambio de nombre es sólo eso: un mejoramiento en la especificación del papel que desempeña el dominio de PROFESOR en los objetos semánticos de ESTUDIANTE. Un cambio similar se hizo en PROFESOR. El atributo ESTUDIANTE fue renombrado como ASESORADO, pero este atributo aún se conecta con los objetos del dominio ESTUDIANTE.

➤ FIGURA 4-14

(Continuación)

Nombre del dominio	Tipo[a]	Descripción semántica	Descripción física
Dirección	G	Otra dirección	Calle Ciudad Estado CódigoPostal
Edificio	S	Nombre de un edificio en el campus	Texto 20
DirecciónDelCampus	G	Una dirección en el campus	Edificio NúmerodeOficina
Ciudad	S	Un nombre de ciudad	Texto 25
COLEGIO	OS	Uno de los diez colegios de la Universidad Highline	Véase la tabla de especificación del objeto semántico
NombredelColegio	S	Nombre oficial de un colegio en la Universidad Highline	Texto 25
DEPARTAMENTO	OS	Un departamento académico en el campus	Véase la tabla de especificación del objeto semántico
Nombredel-Departamento	S	El nombre oficial de un departamento académico	Texto 25
Nombre	S	Parte del nombre de pila de NombredelaPersona	Texto 20
Apellido	S	Parte del apellido del NombredelaPersona	Texto 30
TotaldeAlumnosenla-Especialidad	F	Total de los estudiantes asignados a un departamento específico	Entero, valores {de 0 a 999}; formato 999
NúmerodeOficina	S	El número de una oficina en el campus	Texto 4
NombredelaPersona	G	Nombre y apellido de un administrador, profesor o estudiante	Nombre Apellido
NúmeroTelefónico	S	NúmeroTelefónico con código de área	Texto 4
PROFESOR	OS	El nombre de un miembro de tiempo completo de la facultad de la Universidad Highline	Véase la tabla de especificación del objeto semántico
Fecha del trimestre	S	Trimestre académico y año	Texto 3; los valores [*q01*, donde *q* = uno de {'F', 'W', 'S', 'M'} y *01* es el número decimal del 00 al 99]
Estado	S	La abreviación de dos dígitos de un estado	Texto 2
Calle	S	Dirección de una calle	Texto 30
ESTUDIANTE	OS	Una persona que ha sido aceptada para estudiar en la Universidad Highline	Véase la tabla de especificación del objeto semántico
NúmerodeEstudiante	S	La ID asignada a un estudiante aceptado en la Universidad Highline	Entero; valores {de 10000 a 99999}; formato 99999
Título	S	El título de los individuos que se utilizarán en las direcciones	Texto 3; valores {Sr., Srita.}
CódigoPostal	S	Un código postal de nueve dígitos	Texto 10; formato 99999-9999

[a]F = Fórmula
G = Grupo
S = Simple
OS = Objeto semántico

(b)

La figura 4-14 presenta una especificación tabular del modelo de datos. Los objetos semánticos y los atributos se definen en la especificación del objeto semántico, y los dominios en la especificación del dominio. La primera tabla es una presentación alternativa de la información en los diagramas de los objetos semánticos, y su interpretación es directa.

La segunda tabla, la de dominio, proporciona información sobre los dominios que no están disponibles en los diagramas de los objetos semánticos. Como establecimos anteriormente, un dominio tiene tanto una descripción semántica como una física. La descripción semántica de cada dominio se muestra en la columna Descripción y la descripción física se muestra en la columna Especificación. La columna de Descripción se explica por sí misma.

La especificación para los dominios incluye una descripción física y, en ciertos casos, un conjunto de valores y un formato. Por ejemplo, NúmerodeEstudiante se especifica como un entero con valores entre 10000 y 99999 y con un formato de cinco dígitos decimales. (En esta tabla, un 9 en una especificación del formato significa un dígito decimal.) Otros dominios se documentan de manera similar. Título es un ejemplo de un dominio enumerado cuyos valores para Título son Sr., o Srita. La descripción física de un dominio de grupo consiste en una lista de los dominios incluidos en el grupo. La descripción física de un objeto semántico es sólo una referencia de la descripción del objeto semántico.

El dominio de EspecialidadesTotales es un ejemplo de un cuarto tipo de dominio, el **dominio de fórmula.** Las fórmulas representan atributos computados a partir de otros valores. El dominio TotaldeAlumnosenlaEspecialidad es el conteo de los objetos ESTUDIANTE, los cuales están conectados a un objeto DEPARTAMENTO determinado. No deberíamos tratar de documentar los medios por los cuales este cómputo se llevará a cabo en la definición del dominio. Al respecto, todo aquello que es importante documenta la necesidad y la especificación de la fórmula.

➤ TIPOS DE OBJETOS

Esta sección describe e ilustra siete tipos de objetos. Para cada uno, examinamos un reporte o forma y mostramos cómo modelar el de un objeto. Más adelante, en el capítulo 7, transformamos cada uno de esos tipos de objetos.

En esta sección se utilizan tres términos nuevos: **atributo de un valor** es el que tiene una cardinalidad máxima de 1; **atributo de valores múltiples** es aquel cuya cardinalidad máxima es mayor que 1; y **atributo que no es de objeto** es un atributo simple o grupal.

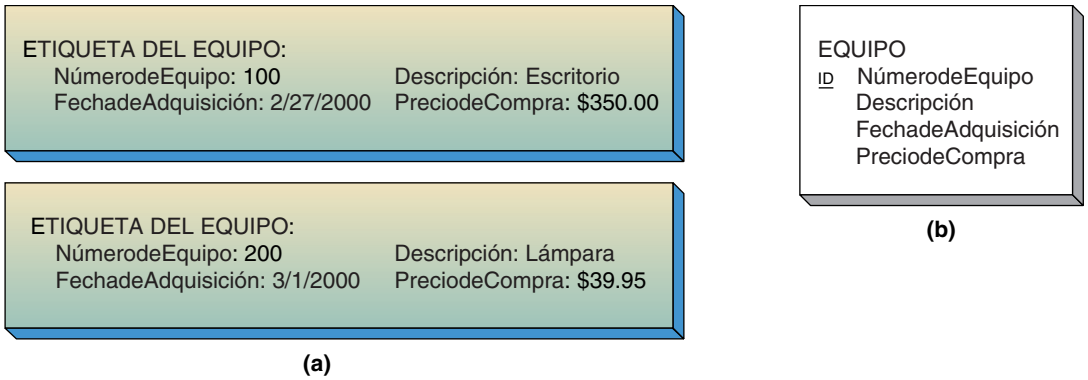
OBJETOS SIMPLES

Un **objeto simple** es un objeto semántico que contiene sólo un valor, atributos simples o grupales. En la figura 4-15 se muestra un ejemplo. El inciso (a) de esta figura muestra dos casos de un reporte llamado *Etiqueta de equipo*. Estas etiquetas se aplican a los artículos del equipo de oficina con el fin de facilitar el seguimiento del inventario. Dichas etiquetas pueden considerarse como un reporte.

La figura 4-15(b) muestra un objeto simple, EQUIPO, el cual modela la Etiqueta de equipo. Los atributos del objeto incluyen los artículos que se muestran en la etiqueta: NúmerodeEquipo, Descripción, FechadeAdquisición y PreciodeCompra. Observe que ninguno de estos atributos tiene valores múltiples, y que ninguno es un atributo de objeto. Por lo tanto, EQUIPO es un objeto simple.

➤ FIGURA 4-15

Ejemplo de un objeto simple: (a) reportes basados en un objeto simple, y (b) el objeto simple EQUIPO



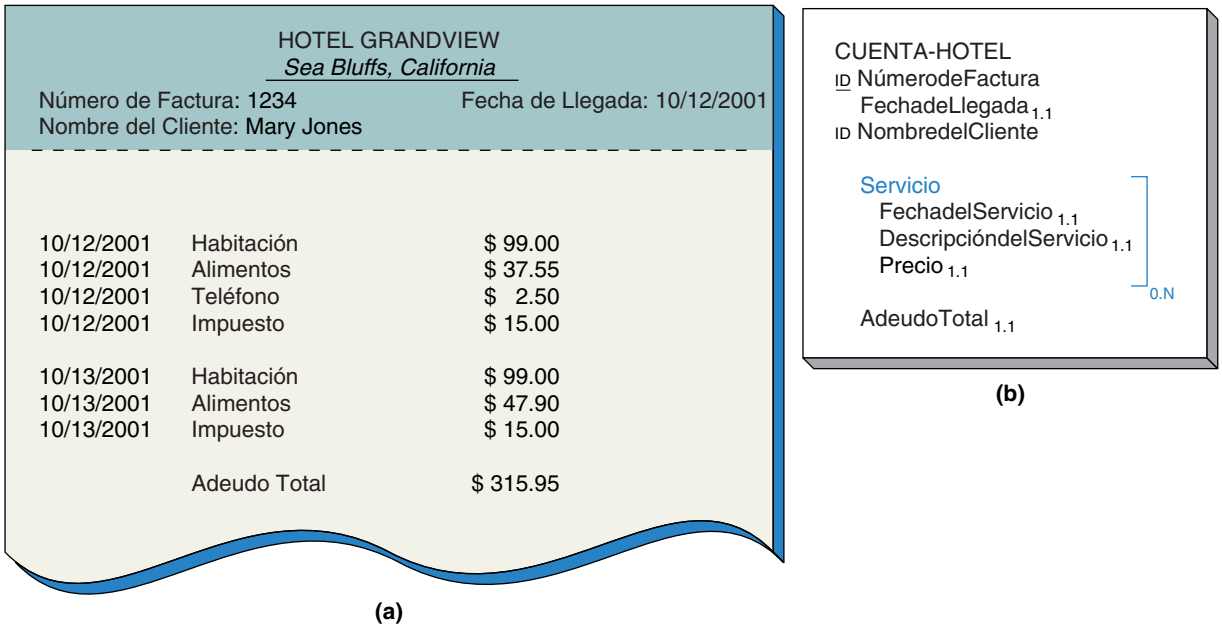
OBJETOS COMPUESTOS

Un **objeto compuesto** es un objeto semántico que contiene uno o más atributos de valores múltiples, simples o grupales, pero no atributos de objetos. La cuenta de hotel que se muestra en la figura 4-16(a) da origen a la necesidad de un objeto compuesto. Dicha cuenta incluye datos relacionados con ésta: NúmerodeFactura, FechadeLlegada, NombredelCliente y AdeudoTotal. También contiene un grupo de atributos que se repite para los servicios proporcionados al cliente. Cada grupo incluye FechadeServicio, DescripcióndelServicio y Precio.

La figura 4-16(b) muestra un diagrama de objeto para el objeto CUENTA-HOTEL. Servicio es un atributo de grupo que tiene una cardinalidad máxima de N, lo cual significa que el grupo FechadeServicio, DescripcióndelServicio, Precio puede presentarse muchas veces en un ejemplo del objeto semántico CUENTA-HOTEL.

➤ FIGURA 4-16

Ejemplo de un objeto compuesto: (a) reporte basado en un objeto compuesto, y (b) objeto compuesto CUENTA-HOTEL



Servicio no se modeló como un objeto semántico independiente; por el contrario, se le consideró un atributo dentro de una CUENTA-HOTEL. Este diseño es conveniente debido a que no considera uno solo de los cargos de un huésped como algo independiente, y por ese motivo los servicios en la cuenta del huésped no tienen identificadores propios. Ningún empleado intenta introducir un Servicio, excepto en el contexto de una cuenta. El empleado introduce los datos para el número de cuenta 1234 y después, en el contexto de dicha cuenta, introduce los cargos, o cobra una cuenta e introduce los cargos adicionales que forman parte de ésta.

La cardinalidad mínima de Servicio es 0, lo cual significa que un objeto CUENTA-HOTEL puede existir sin ningún dato de Servicio. Esto permite que inicie una cuenta cuando el cliente se registra y antes de que haya ningún cargo. Si la cardinalidad mínima fuera 1, entonces no podría iniciarse ninguna CUENTA-HOTEL hasta que hubiera cuando menos un cargo. Esta decisión del diseño debe realizarse desde el punto de vista de las reglas del negocio. Puede ser que la política del hotel sea que no comience la cuenta hasta que haya un cargo. Si es así, entonces la cardinalidad mínima de Servicio debería ser 1.

Un objeto compuesto puede tener más de un atributo de valores múltiples. La figura 4-17(a) muestra una cuenta de hotel que tiene atributos de valores múltiples para NombredelCliente, así como un grupo de valores múltiples para los cargos de los servicios. Cada uno de estos grupos es independiente uno del otro. Por ejemplo, la segunda instancia de NombredelCliente no se asocia lógicamente con el segundo Servicio.

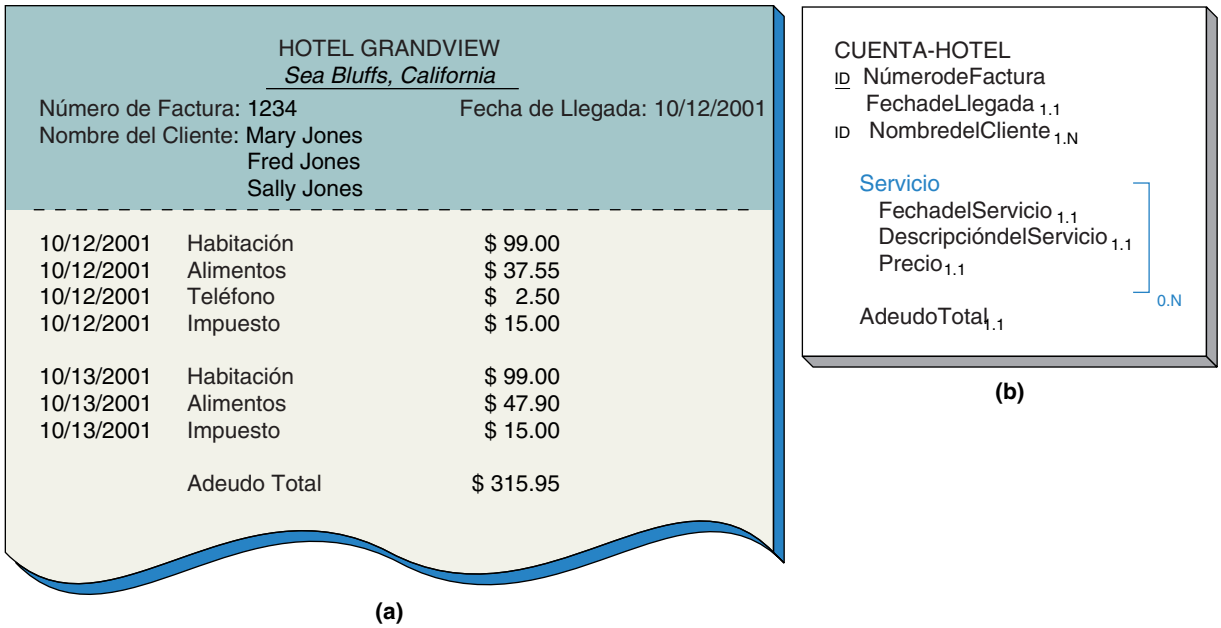
La figura 4-17(b) es un diagrama de objeto para la cuenta del hotel en la figura 4-17(a). NombredelCliente se muestra como un atributo de valores múltiples. No se incluye en el paréntesis de los cargos del servicio debido a que las repeticiones de NombredelCliente no tienen nada que ver con las repeticiones de los servicios. Como podemos observar, los dos son independientes.

Tanto los atributos simples como los grupales pueden ser de valores múltiples. Por ejemplo, en la figura 4-17(a), NombredelCliente es un atributo simple de valores múltiples. Por sí mismo, es suficiente para el objeto que es considerado compuesto.

Los atributos de valores múltiples pueden anidarse uno dentro del otro. Por ejemplo, suponga que es importante registrar los gastos individuales dentro de un Servicio.

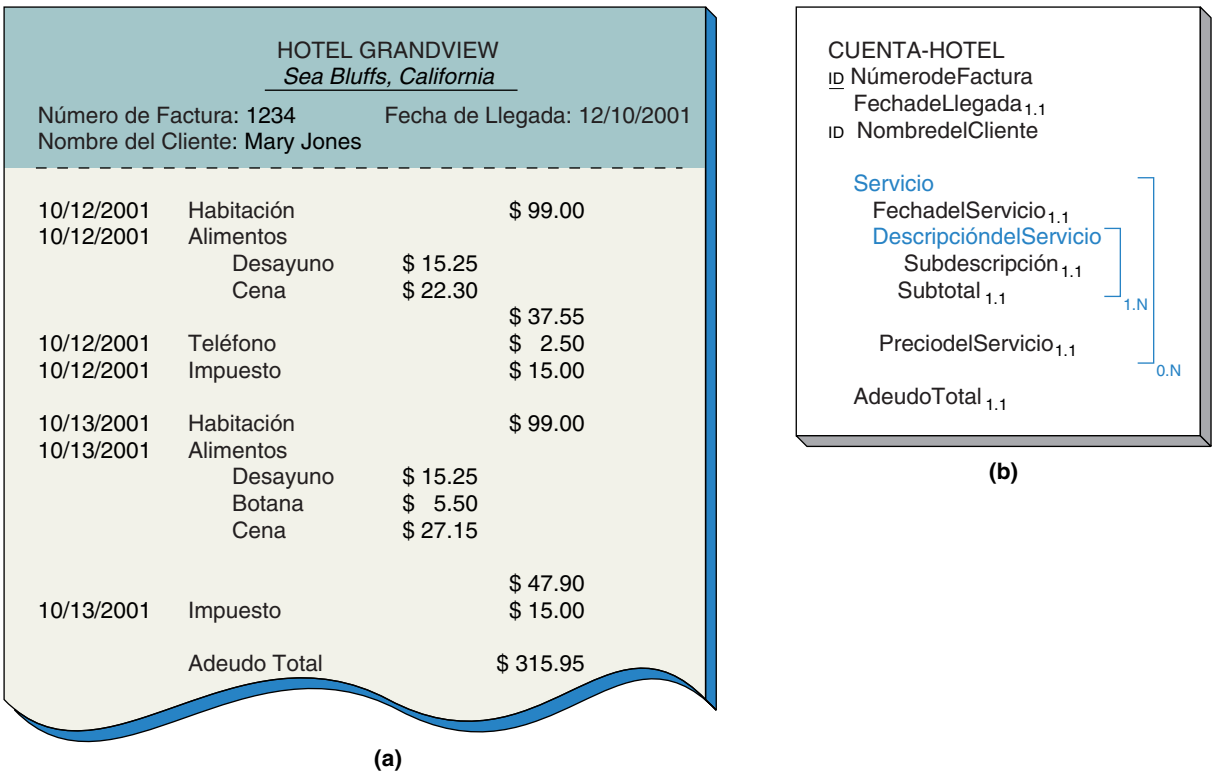
➤ FIGURA 4-17

Un objeto compuesto con dos grupos: (a) CUENTA-HOTEL con valores múltiples en Nombre del Cliente, y (b) CUENTA-HOTEL con dos grupos de valores múltiples



➤ FIGURA 4-18

Un objeto compuesto con grupos anidados: (a) CUENTA-HOTEL con subdescripciones de servicios, y (b) CUENTA-HOTEL con grupos anidados de valores múltiples



En la forma de la figura 4-18(a) se subdividen los cargos. Por ejemplo, alimentos se desglosa por comidas. En la figura 4-18(b) se presenta un diagrama de objeto para estos atributos anidados y se muestran los cargos por servicios que tienen subelementos.

A manera de repaso, diremos que un objeto compuesto es el que contiene uno o más atributos simples de valor múltiple o atributos grupales. No tiene atributos de objeto.

Objetos combinados

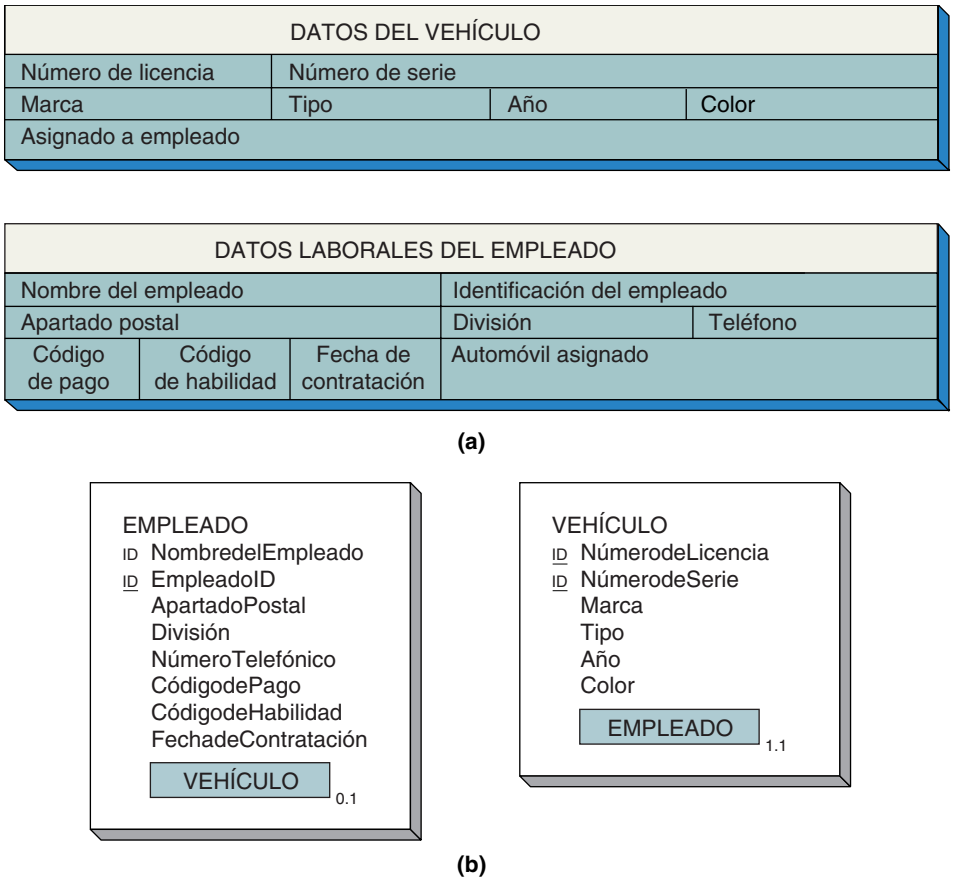
Un **objeto combinado** contiene cuando menos un atributo de un objeto. La figura 4-19(a) muestra dos formas de ingreso de datos diferentes. Una de éstas, que utiliza el Departamento de Finanzas de la compañía, se emplea para llevar el registro de los vehículos. La segunda forma se usa para guardar los datos de los empleados. De acuerdo con estas formas, se asigna un vehículo a un empleado y un empleado tiene asignado cuando mucho un vehículo.

No podemos decir a partir de estas formas si un automóvil debe ser asignado a un empleado o si cada empleado debe tener uno. Para obtener dicha información, tendríamos que preguntar a los usuarios en los departamentos de finanzas o de recursos humanos. Suponga que descubrimos que un EMPLEADO no necesita tener un VEHÍCULO, pero que se debe asignar un VEHÍCULO a un empleado.

La figura 4-19(b) muestra los diagramas de objeto para EMPLEADO y VEHÍCULO. Un EMPLEADO tiene un VEHÍCULO como uno de sus atributos, y VEHÍCULO, a su vez, tiene EMPLEADO como uno de sus atributos. Puesto que tanto EMPLEADO como VEHÍCULO contienen atributos de objeto, ambos son objetos combinados. Además, debido a que ningún atributo es de valores múltiples, la relación de EMPLEADO con VEHÍCULO es uno a uno, o 1:1.

➤ FIGURA 4-19

Objetos combinados con propiedades pareadas 1:1: (a) ejemplo de las formas de ingreso de los datos de Vehículo y Empleado, y (b) objetos combinados EMPLEADO y VEHÍCULO



En la figura 4-19(a) las formas de Empleado y Vehículo se interrelacionan. Es decir, los DATOS DEL VEHÍCULO tienen el campo de asignación del Empleado, y los DATOS LABORALES DEL EMPLEADO tienen el campo del Automóvil asignado. Pero éste no siempre es el caso; a veces la relación puede aparecer sólo en una dirección. Considere el reporte y la forma de la figura 4-20(b), los cuales se refieren a dos objetos: DORMITORIO y ESTUDIANTE. En el Reporte de ocupación del dormitorio, podemos apreciar que los usuarios piensan en un dormitorio conforme a los atributos correspondientes a dormitorio (Dormitorio, AsistenteResidente, NúmeroTelefónico), y también en los atributos correspondientes a los estudiantes (NombredelEstudiante, NúmerodeEstudiante, Clase) que viven en el dormitorio.

Por otra parte, la forma datos del estudiante muestra sólo los datos de éste; no incluye ningún dato acerca del dormitorio. La dirección del campus puede contener la dirección de un dormitorio, pero si es así, aparentemente no es tan importante como para documentarlo en el formato. En el proyecto de desarrollo de una base de datos esta posibilidad puede comprobarse mediante una entrevista con los usuarios. Aquí supondremos que la forma datos del estudiante no incluye los datos del dormitorio.

Como establecimos anteriormente, los atributos del objeto siempre se presentan pareados. Incluso si las formas, reportes y consultas indican que sólo se puede apreciar una parte de la relación, siempre existen ambas partes de ésta. Por analogía, un puente que conecta dos islas toca a ambas y puede ser utilizado en las dos direcciones, incluso si el puente es, por costumbre o por ley, de un solo sentido.

Cuando no puede encontrarse ningún reporte o ninguna forma para documentar una parte de una relación, el equipo analista puede preguntarle a los usuarios acerca de la cardinalidad de dicha relación. En este caso, el equipo necesitaría saber cuántos DORMITORIOs puede tener un ESTUDIANTE y si un ESTUDIANTE tiene que estar relacionado con un DORMITORIO. Aquí suponemos que las respuestas a estas preguntas son que un ESTUDIANTE está relacionado sólo con un DORMITORIO y que puede no

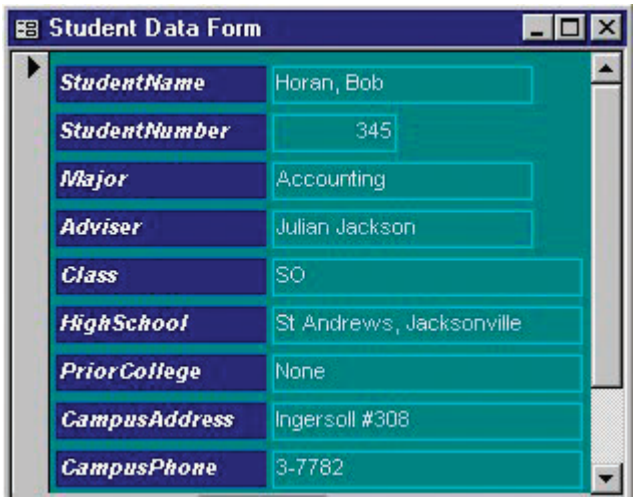
➤ FIGURA 4-20

Objetos combinados con las propiedades pareadas 1:N: (a) ejemplo de Reporte de ocupación del dormitorio y de la Forma de datos del estudiante (Student Data Form), y (b) objetos combinados DORMITORIO y ESTUDIANTE

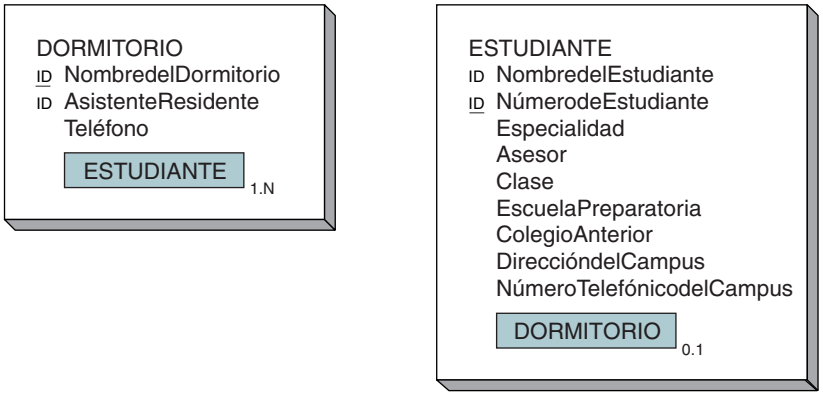
REPORTE DE OCUPACIÓN DE DORMITORIO

Dormitorio	Asistente Residente	Teléfono
Ingersoll	Sarah y Allen French	3-5567

Nombre del estudiante	Número de estudiante	Clase
Adams, Elizabeth	710	OS
Baker, Rex	104	FR
Baker, Brydie	744	JN
Charles, Stewart	319	OS
Scott, Sally	447	OS
Taylor, Lynne	810	FR



(a)



(b)

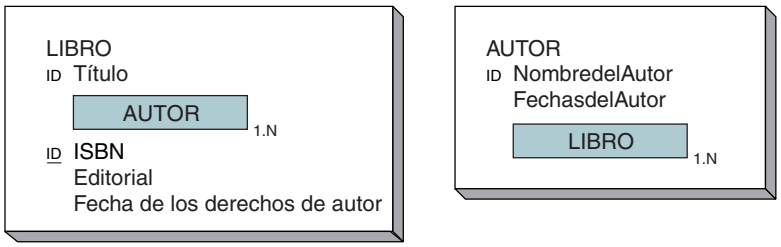
estar relacionado con ningún DORMITORIO. Así, en la figura 4-20(b) DORMITORIO contiene valores múltiples de ESTUDIANTE y ESTUDIANTE contiene un valor de DORMITORIO; por consiguiente, la relación de DORMITORIO con ESTUDIANTE es de uno a muchos, o 1:N.

Una tercera ilustración de los objetos combinados aparece en la figura 4-21(a). A partir de estas dos formas podemos deducir que varios autores pueden escribir un libro (esto deriva de la forma Book Stock Data) y que un autor puede escribir varios libros (proviene de Book in Stock, by Author). Por consiguiente, en la figura 4-21(b) el objeto LIBRO contiene muchos valores de AUTOR, y AUTOR contiene varios valores de LIBRO. De ahí que la relación entre LIBRO y AUTOR sea de muchos a muchos, o

➤ FIGURA 4-21

Objetos combinados con propiedades pareadas N:M: (a) forma Ingreso de datos en una librería (Bookstore Data Entry Form), y (b) objetos LIBRO y AUTOR

(a)



(b)

N:M. Además, un LIBRO debe tener un AUTOR, y un AUTOR (para serlo) debe haber escrito cuando lo menos un LIBRO. De este modo, ambos objetos tienen una cardinalidad mínima de uno.

La figura 4-22 resume los cuatro tipos de objetos compuestos. En general, OBJETO1 puede contener un máximo de uno o varios OBJETOs2. De manera similar, OBJETO2 puede contener uno o varios OBJETOs1. Usaremos esta tabla para analizar el diseño de una base de datos en el capítulo 7.

OBJETOS HÍBRIDOS

Los **objetos híbridos** son combinaciones de objetos combinados y compuestos. En particular, un objeto híbrido es un objeto semántico que cuando menos tiene un atributo grupal de valores múltiples e incluye un atributo del objeto semántico.

La figura 4-23(a) es una segunda versión del reporte sobre la ocupación del dormitorio que se muestra en la figura 4-20(a). La diferencia radica en que la tercera columna de los datos del estudiante contiene Renta, en lugar de Clase. Ésta es una diferencia importante puesto que renta no es un atributo del ESTUDIANTE, sino que pertenece a la combinación ESTUDIANTE y DORMITORIO y es un atributo de DORMITORIO.

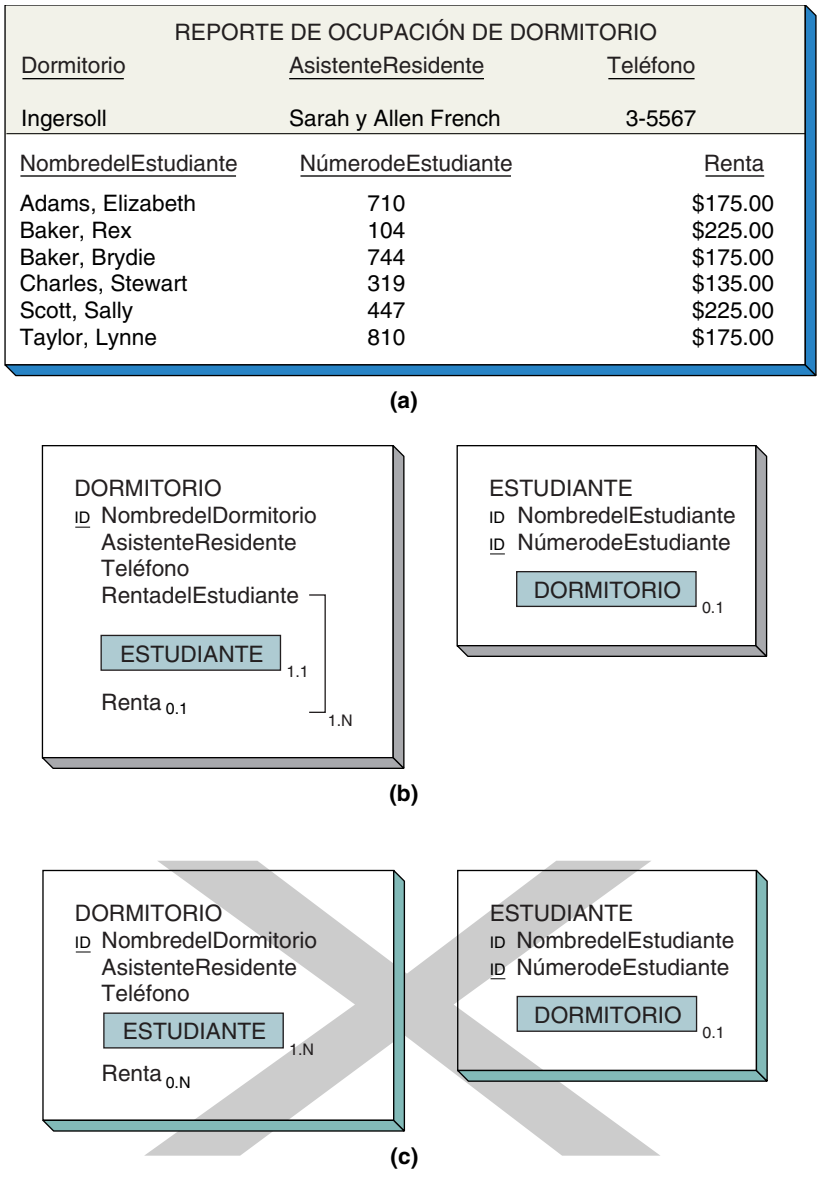
➤ FIGURA 4-22

Cuatro tipos de objetos compuestos

	El objeto1 puede contener		
El objeto2 puede contener		Uno	Muchos
	Uno	**1:1**	**1:N**
	Muchos	**M:1**	**M:N**

➤ FIGURA 4-23

Objeto híbrido DORMITORIO: (a) Reporte de ocupación de dormitorio con la propiedad Renta; (b) objetos DORMITORIO y ESTUDIANTE correctos, y (c) objetos DORMITORIO y ESTUDIANTE incorrectos



La figura 4-23(b) es un diagrama de objeto que modela esta forma. DORMITORIO contiene un grupo de valores múltiples que tienen el atributo del objeto ESTUDIANTE y el atributo del no objeto Renta. Esto significa que Renta se parea con ESTUDIANTE en el contexto de DORMITORIO.

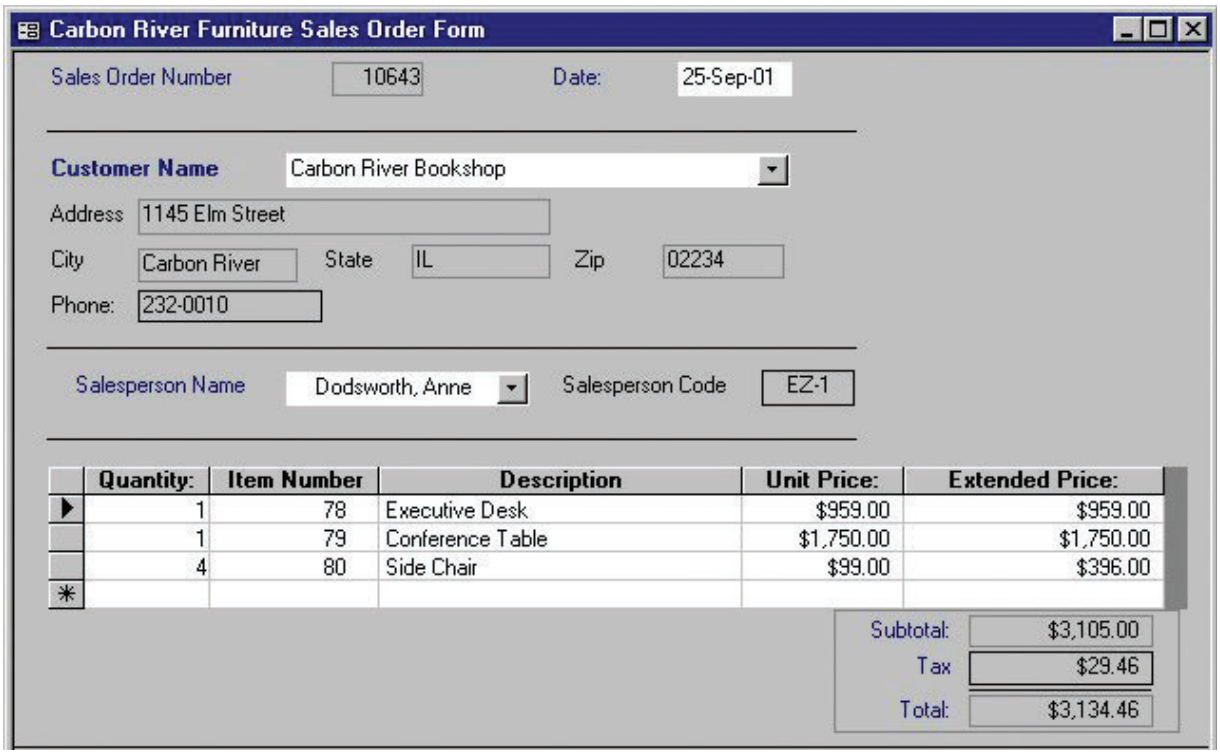
Ahora examine el objeto alternativo DORMITORIO en la figura 4-23(c). Éste es un modelo *incorrecto* del reporte en la figura 4-23(a), ya que muestra que Renta y ESTUDIANTE son independientemente valores múltiples, lo cual es incorrecto debido a que RENTA y ESTUDIANTE son valores múltiples como un par.

La figura 4-24(a) muestra una forma basada en otro objeto híbrido. La forma Sales Order Form contiene datos acerca de un pedido (Sales Order Number, Date, Subtotal, Tax, Total) (Número de Pedido, Fecha, Subtotal, Impuesto, Total), datos respecto a un CUSTOMER (CLIENTE) y un SALESPERSON (VENDEDOR), así como un grupo de valores múltiples que contienen por sí mismos datos acerca de los artículos del pedido. Además, los datos de ITEM, o ARTÍCULO (Item Number, Description, y Unit Price) (Número de Artículo, Descripción y Precio Unitario) aparecen dentro del grupo de valores múltiples.

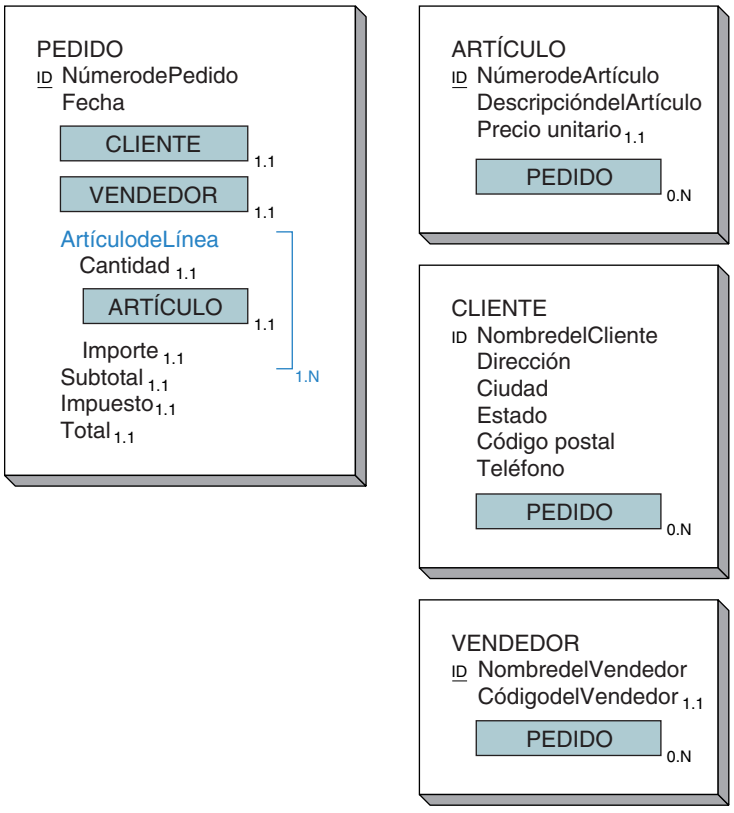
La figura 4-24(b) muestra el objeto semántico PEDIDO. Éste contiene los atributos de los no objetos Número de Pedido, Fecha, Subtotal, Impuesto y Total (Sales Order Number, Date, Subtotal, Tax y Total.) Asimismo, contiene atributos de los objetos

➤ FIGURA 4-24

Objetos híbridos PEDIDO (SALES-ORDER) y objetos relacionados: (a) forma Pedido (Sales Order Form), y (b) objetos para modelar la forma Pedido



(a)



(b)

CLIENTE (CUSTOMER) y VENDEDOR (SALESPERSON) y un grupo multivalor que representa cada artículo de línea en el pedido. El grupo contiene atributos de los no objetos Cantidad e Importe (Quantity y ExtendedPrice) y el atributo del objeto ARTÍCULO (ITEM).

Los diagramas de los objetos en la figura 4-24(b) son ambiguos en un aspecto que puede o no ser importante, dependiendo de la aplicación. De acuerdo con el diagrama del objeto ARTÍCULO, un ARTÍCULO puede conectarse con más de un PEDIDO. Pero debido a que el grupo de valores múltiples LíneadelArtículo está encapsulado (oculto) en este diagrama de PEDIDO no es claro si un ARTÍCULO puede presentarse *una o muchas veces* en el mismo PEDIDO.

En general, existen cuatro interpretaciones de la cardinalidad máxima de los atributos pareados en el objeto híbrido PEDIDO:

1. Un ARTÍCULO sólo puede aparecer en un PEDIDO y en sólo uno de ArtículodeLínea dentro de PEDIDO.
2. Un ARTÍCULO sólo puede aparecer en un PEDIDO, pero en varios ArtículodeLínea distintos dentro de dicho PEDIDO.
3. Un ARTÍCULO puede aparecer en varios PEDIDOs distintos pero sólo en ArtículodeLínea dentro de cada uno de esos PEDIDOs.
4. Un ARTÍCULO puede aparecer en distintos PEDIDOs y en diferentes ArtículodeLínea dentro de dichos PEDIDOs.

Cuando sea importante distinguir entre estas instancias debe utilizarse la siguiente notación: si el Caso 1 o el 2 están vigentes, la cardinalidad máxima del atributo del objeto híbrido debe terminarse como 1. Por lo tanto, en el caso de este ejemplo, la cardinalidad máxima de PEDIDO en ARTÍCULO se establece en 1. Si un ARTÍCULO sólo aparece en ArtículodeLínea de un PEDIDO (Caso 1), debe indicarse que tiene una Identificación única en dicho grupo. De lo contrario (Caso 2), no es necesario que se indique. Estas dos instancias se muestran en la figura 4-25(a) y (b).

Si los Casos 3 o 4 están vigentes, la cardinalidad máxima del atributo del objeto híbrido debe determinarse como N. Para este ejemplo la cardinalidad máxima de PEDIDO en ARTÍCULO se determina como N. Además, si un ARTÍCULO sólo aparece en un ArtículodeLínea de un PEDIDO (Caso 3), debe indicarse que tiene una Identificación única en dicho grupo. De lo contrario (Caso 4), no es necesario indicarlo. Estas dos instancias se muestran en la figura 4-25(c) y (d).

Objetos de asociación

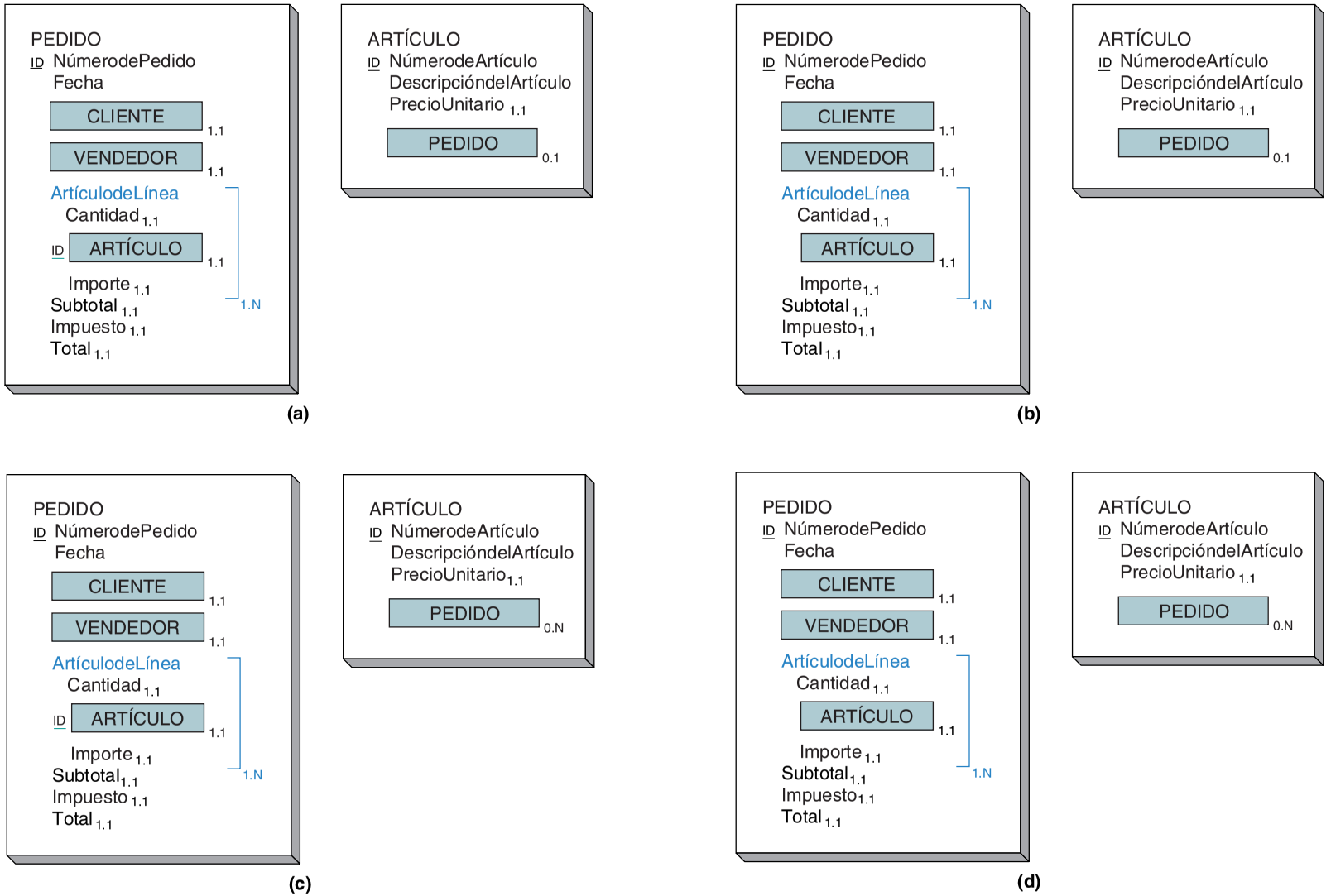
Un **objeto de asociación** relaciona dos (o más) objetos y almacena datos que son peculiares para dicha relación. La figura 4-26(a) muestra un reporte y dos pantallas de ingreso de datos que originan la necesidad de un objeto de asociación. El reporte contiene datos acerca de los vuelos en una aerolínea, datos de un avión en particular, así como del piloto asignado a dicho vuelo. Las dos formas de ingreso de datos se refieren a un piloto y un avión.

En la figura 4-26(b), VUELO es un objeto de asociación que vincula a dos objetos: AVIÓN y PILOTO, y almacena datos sobre su asociación. VUELO contiene AVIÓN y PILOTO, pero AVIÓN y PILOTO contienen valores múltiples de VUELO. Este patrón particular de la asociación entre dos (o más) objetos con los datos ocurre con frecuencia, especialmente en las aplicaciones que involucran la asignación de dos o más cosas. Otros ejemplos son un TRABAJO que asigna un ARQUITECTO a un CLIENTE, una TAREA que asigna un EMPLEADO a un PROYECTO y un PEDIDO-DE-COMPRA que asigna un VENDEDOR a un SERVICIO.

Para el ejemplo de la figura 4-26, el objeto de asociación VUELO tiene un identificador propio: el grupo {NúmerodeVuelo, Fecha}. Con frecuencia los objetos de asociación no tienen identificadores propios, en tal caso el identificador es la combinación de los identificadores de los objetos que se asocian.

FIGURA 4-25

Ejemplos de los cuatro casos de cardinalidad máxima en un objeto híbrido: (a) ARTÍCULO en un PEDIDO, (b) ARTÍCULO en (posiblemente) varios ArtículodeLínea de un PEDIDO, (c) ARTÍCULO en un ArtículodeLínea (posiblemente) varios PEDIDOS, y (d) ARTÍCULO en (posiblemente) varios ArtículodeLínea de (tal vez) varios PEDIDOs



➤ FIGURA 4-26

Ejemplos de un objeto de asociación: (a) ejemplos del reporte de vuelo y formatos, y (b) objetos VUELO, PILOTO, AVIÓN

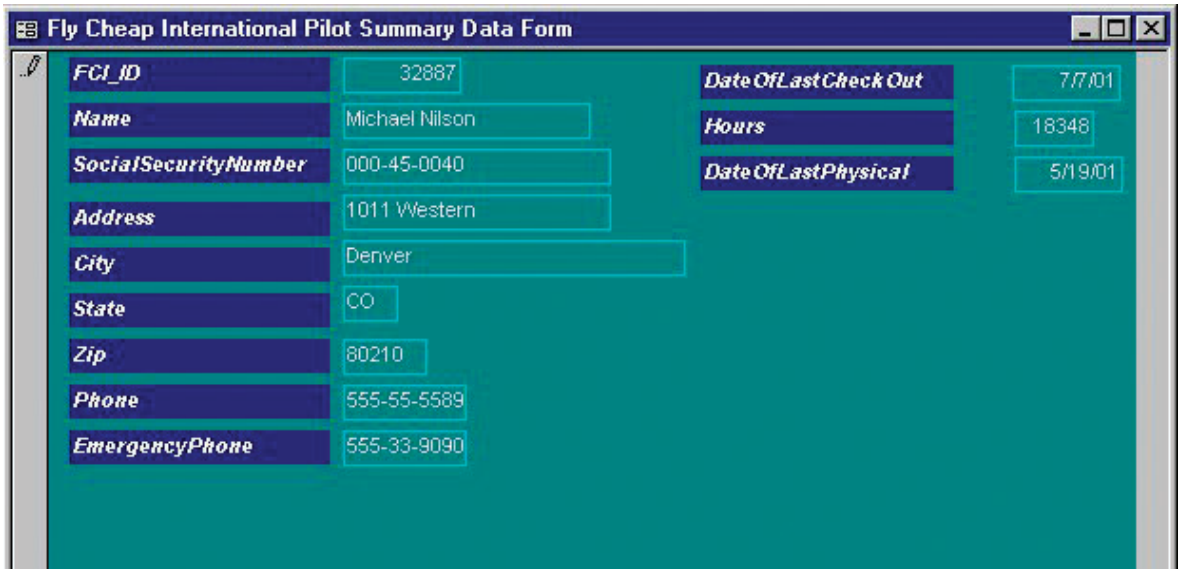
COMPAÑÍA FLY CHEAP INTERNATIONAL
Reporte de los datos de la planificación del vuelo

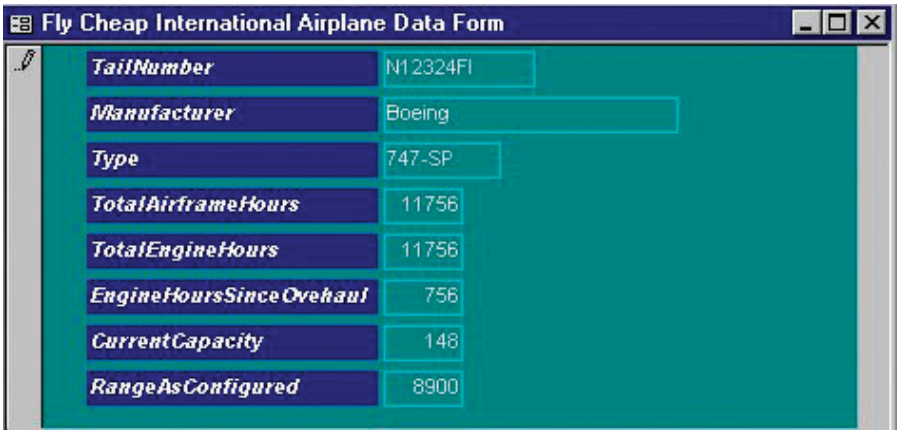
NÚMERO DE VUELO *FC-17* FECHA *7/30/2001*
CIUDAD DE ORIGEN *Seattle* DESTINO *Hong Kong*
COMBUSTIBLE EN EL DESPEGUE

PESO EN EL DESPEGUE

AVIÓN
- Número de la cola *N1234FI*
- Tipo *747-SP*
- Capacidad *148*

PILOTO
- Nombre *Michael Nilson*
- Identificación de vuelo *32887*
- Horas de vuelo *18,348*



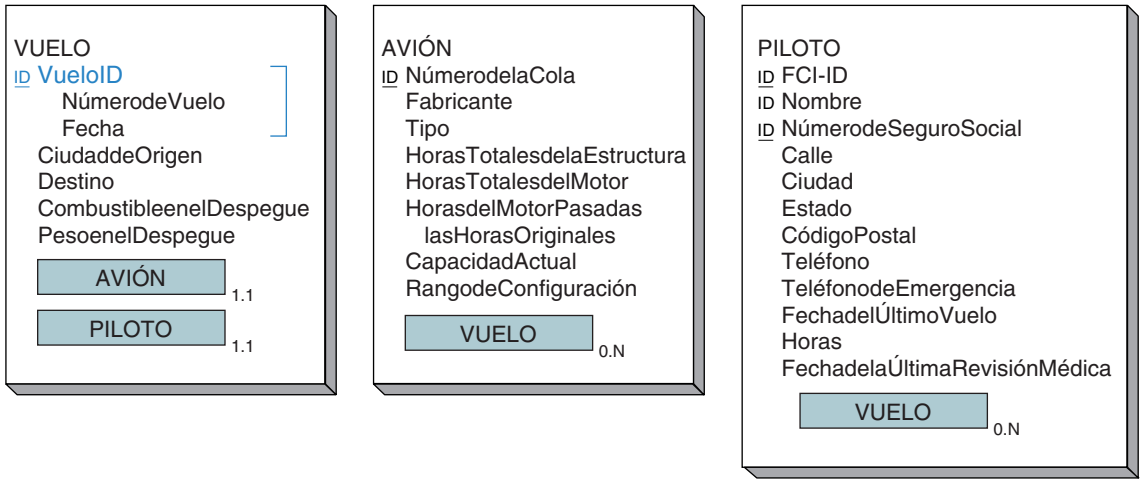


(a)

(continúa)

➤ FIGURA 4-26

(Continuación)



(b)

Para comprender mejor esto, considere la figura 4-27(a), la cual muestra un reporte de la asignación de arquitectos a proyectos. Aunque ésta no tiene ningún identificador obvio, en realidad el identificador es la combinación de {NombredelProyecto, NombredelArquitecto}. Sin embargo, estos atributos pertenecen a PROYECTO y ARQUITECTO y no a ASIGNACIÓN. Por lo tanto, el identificador de ASIGNACIÓN es la combinación de dichos identificadores de las cosas que están asignadas.

La figura 4-27(b) muestra los diagramas de los objetos para este caso. PROYECTO y ARQUITECTO son atributos del objeto de ASIGNACIÓN, y el grupo {PROYECTO, ARQUITECTO} es el identificador de la ASIGNACIÓN. Esto significa que la combinación de un ejemplo de PROYECTO y un ejemplo de ARQUITECTO identifica una ASIGNACIÓN en particular.

Observe que el identificador de AsignaciónID en la figura 4-27(b) no es único, por lo tanto indica que un arquitecto puede ser asignado a un proyecto más de una vez. Si esto no es correcto, debe aclararse que el identificador no es único. También, si se puede asignar un empleado a un proyecto más de una vez, y si por alguna razón es importante que se tenga un identificador único para una ASIGNACIÓN, se debe agregar al grupo el atributo Fecha, o algún otro que indique el tiempo (Semana, Trimestre, etcétera).

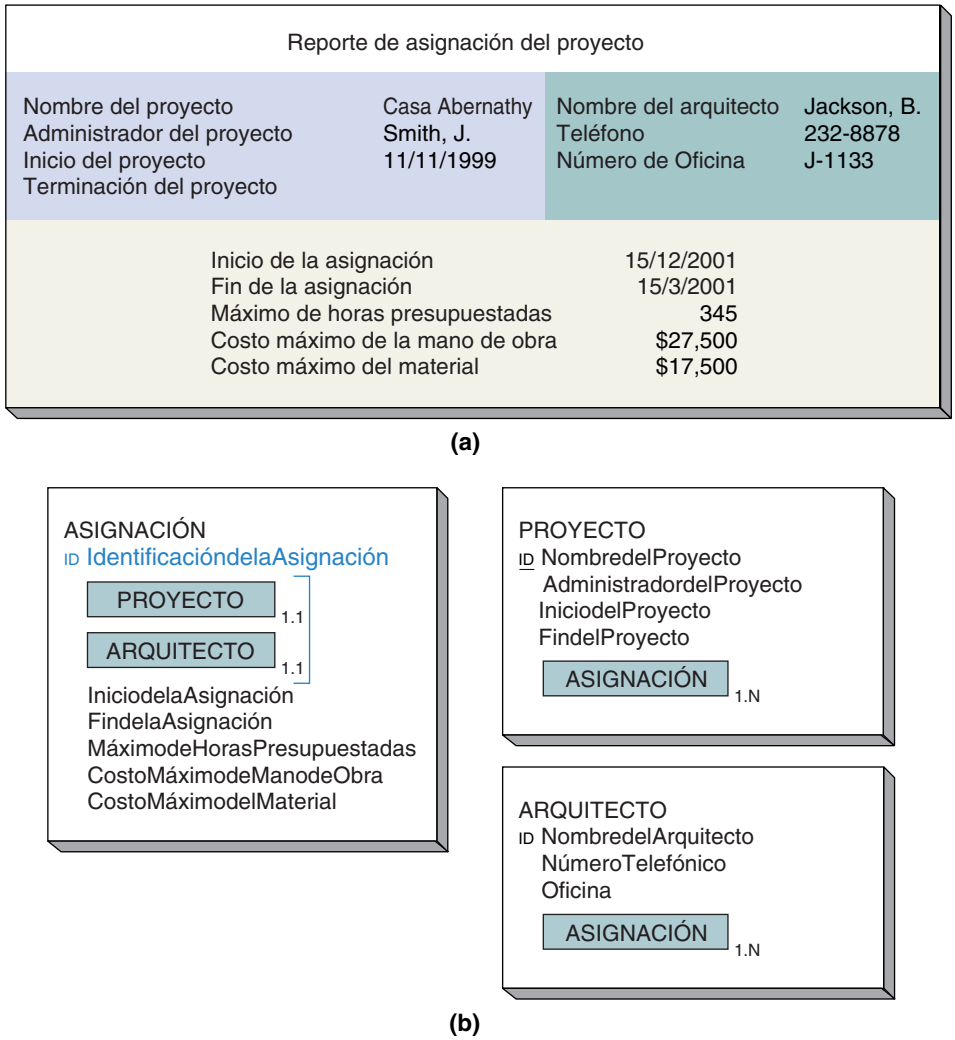
Objetos padre-subtipo

Para entender a los objetos padre y subtipo, considere al objeto EMPLEADO en la figura 4-28(a). Algunos de los atributos en EMPLEADO pertenecen a todos los empleados y otros sólo a los empleados que son administradores. El objeto en la figura 4-28(a) no es muy preciso debido a que los atributos orientados al administrador no son apropiados para los empleados que no son administradores.

En la figura 4-28(b) se muestra un modelo mejor, en el cual el objeto EMPLEADO contiene un objeto subtipo, ADMINISTRADOR. Todos los atributos orientados al administrador se han trasladado al objeto ADMINISTRADOR. Los empleados que no son administradores tienen una instancia del objeto EMPLEADO y ninguna del objeto ADMINISTRADOR. Los empleados que son administradores tienen una instancia de EMPLEADO y una instancia de ADMINISTRADOR. En este ejemplo, al objeto EMPLEADO se le denomina **objeto padre** u **objeto supertipo;** y al objeto ADMINISTRADOR se le denomina **objeto subtipo.**

➤ FIGURA 4-27

Objeto de asociación ASIGNACIÓN: (a) ejemplo del reporte de asignación, y (b) objeto ASIGNACIÓN con objeto de ID semántico



El primer atributo de un subtipo es el atributo padre y se denota mediante el subíndice P. Se requieren siempre los atributos padre. Los identificadores del subtipo son los mismos identificadores que los del padre. En la figura 4-28, NúmerodeEmpleado y NombredelEmpleado son identificadores de EMPLEADO y ADMINISTRADOR.

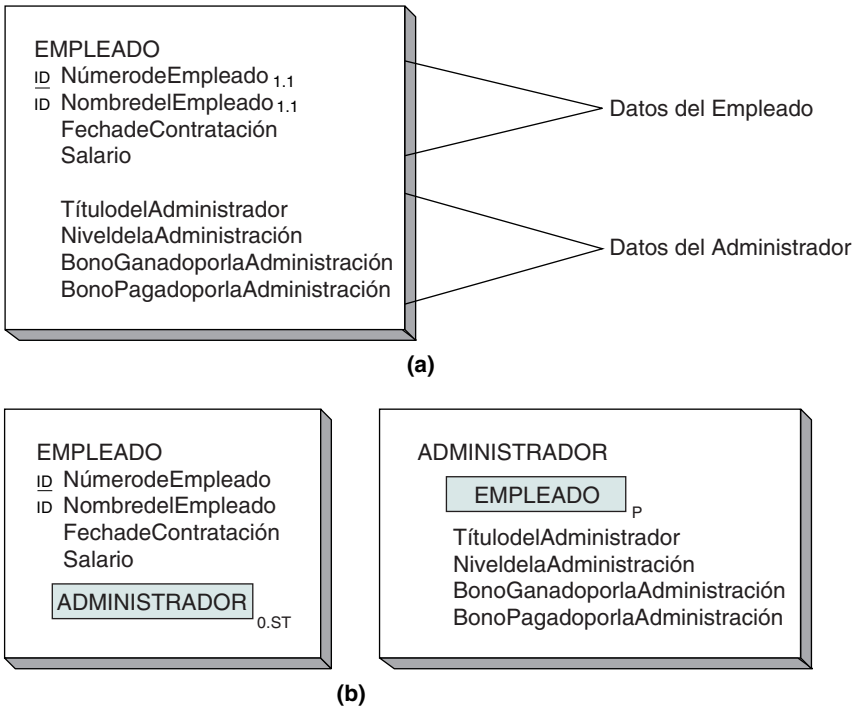
Los atributos de subtipo se muestran con el subíndice 0.ST o 1.ST. El primer dígito (0 o 1) es la cardinalidad máxima del subtipo. Si es 0, el subtipo es opcional, y si es 1 se requiere el subtipo. (Un subtipo requerido no tiene sentido en este ejemplo, pero sí lo tendrá para otros ejemplos de mayor complejidad.) El subíndice *ST* indica que el atributo es un subtipo, o un atributo ES-UN.

Los objetos padre y subtipo tienen una característica importante denominada herencia. Un subtipo adquiere, o *hereda*, todos los atributos de su padre, y por lo tanto un ADMINISTRADOR hereda todos los atributos de un EMPLEADO. Además, el padre adquiere todos los atributos de sus subtipos, y un EMPLEADO que es ADMINISTRADOR adquiere todos los atributos de un ADMINISTRADOR.

Un objeto semántico puede contener más de un atributo de subtipo. La figura 4-29 muestra un segundo objeto de EMPLEADO el cual tiene dos atributos de subtipo, ADMINISTRADOR y PROGRAMADOR. Puesto que todos estos atributos son opcionales, un EMPLEADO puede no tener ninguno, o tener uno o ambos de estos subtipos. Esto significa que algunos empleados no son ni administradores ni programadores; algunos son administradores pero no programadores; otros son programadores pero no administradores, y algunos más son programadores y administradores.

➤ FIGURA 4-28

Necesidad del subtipo ADMINISTRADOR: (a) EMPLEADO sin el subtipo, y (b) EMPLEADO con el subtipo ADMINISTRADOR

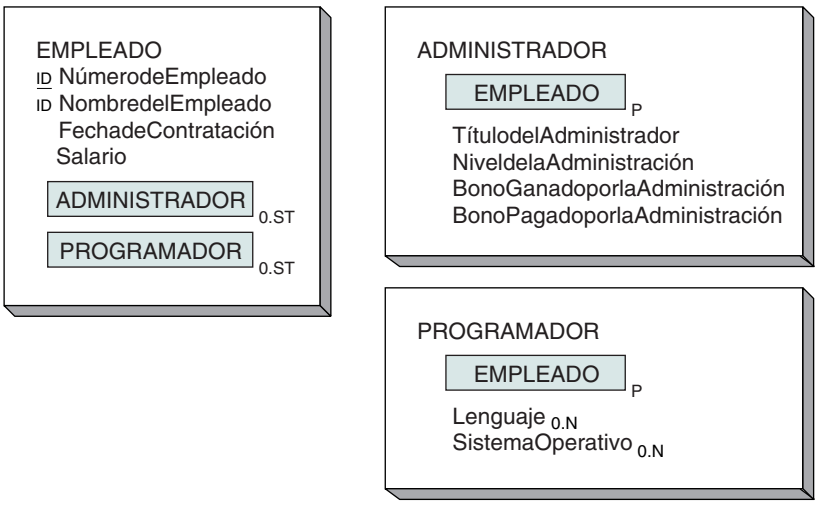


Algunas veces los subtipos se excluyen unos a otros. Esto es, un VEHÍCULO puede ser un AUTO o un AUTOBÚS, pero no ambos. Un CLIENTE puede ser una PERSONA, una SOCIEDAD, o una EMPRESA, pero sólo uno de esos tres tipos. Cuando los subtipos se excluyen unos a otros se colocan en un grupo de subtipo, y al grupo se le asigna un subíndice del formato *X.Y.Z*. X es la cardinalidad mínima, 0 o 1, dependiendo de si se requiere o no el grupo subtipo. Y y Z son las sumas del número de atributos en el grupo a los que se les permite tener un valor. Y es el número mínimo requerido y Z es el número máximo permitido.

La figura 4-30(a) muestra los tres tipos de CLIENTE como un grupo de subtipos. El subíndice del grupo, 0.1.1, significa que no se requiere del subtipo, pero si éste existe debe haber un mínimo de uno y un máximo de uno (o exactamente uno) de los subtipos en el grupo. Observe que cada uno de los subtipos tiene el subíndice 0.ST, lo cual significa que todos son opcionales, tal y como debe ser. Si se requirieran todos, la suma máxima tendría que ser tres y no uno. Esta notación es lo suficientemente poderosa

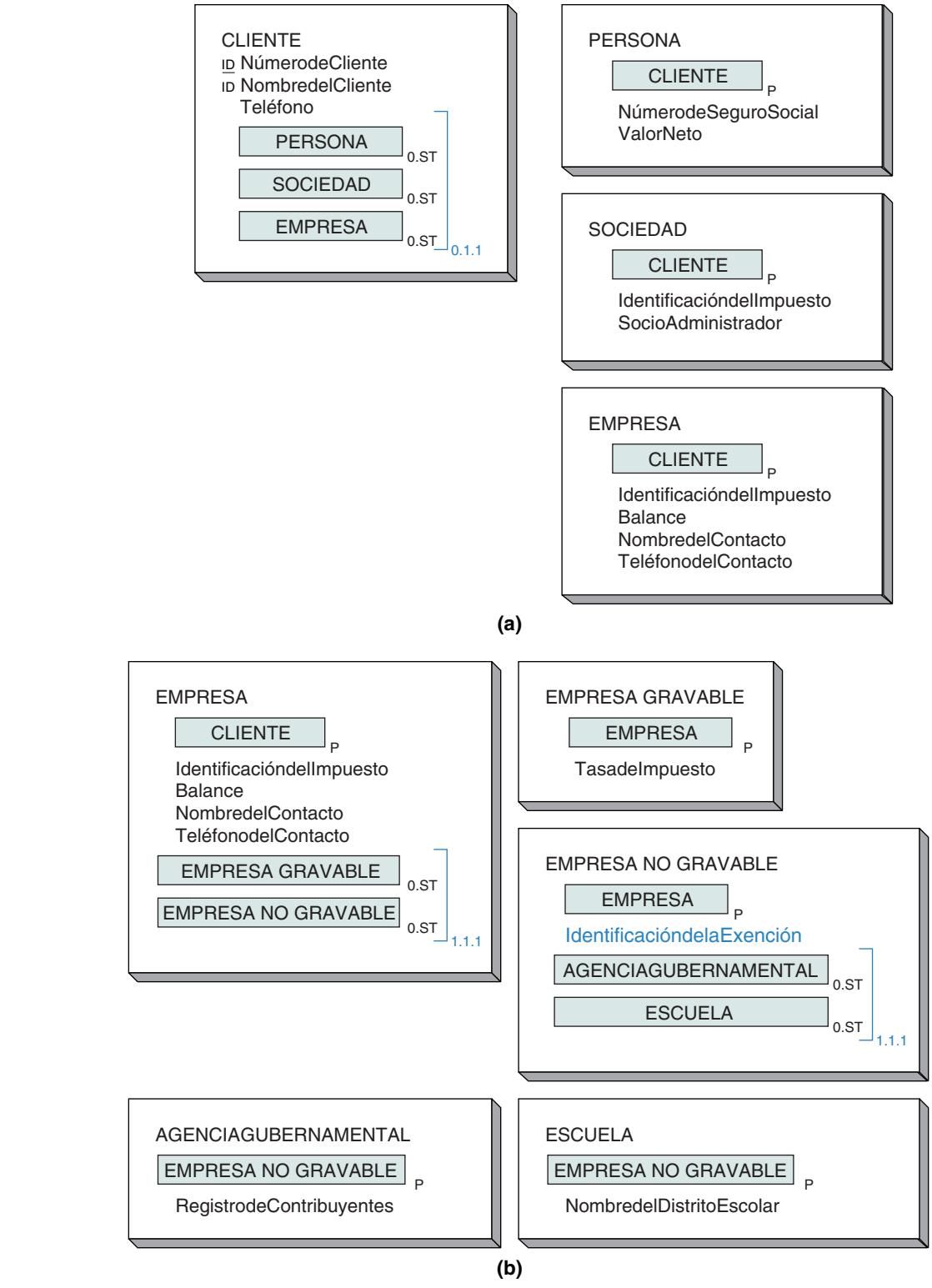
➤ FIGURA 4-29

EMPLEADO con dos propiedades subtipo



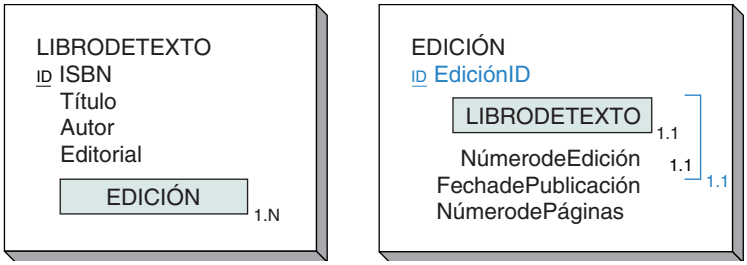
➤ FIGURA 4-30

Subtipos: (a) exclusivo y (b) anidado



➤ FIGURA 4-31

Ejemplo de un objeto arquetipo-versión



como para emplearla en situaciones en las que sea necesaria una lista de tres, cinco o siete de 10 subtipos.

Incluso se pueden modelar restricciones mucho más complejas cuando los subtipos se anidan. El grupo de subtipos de la figura 4-30(b) modela una situación en la que el subtipo EMPRESA debe ser una EMPRESA SUJETA A IMPUESTOS o una EMPRESA NO SUJETA A IMPUESTOS. Si es una EMPRESA NO SUJETA A IMPUESTOS, debe ser una AGENCIA GUBERNAMENTAL o una ESCUELA. En este ejemplo sólo se muestran pocos atributos de no objetos. En realidad, si se requiriera de dicha estructura compleja probablemente existirían más atributos.

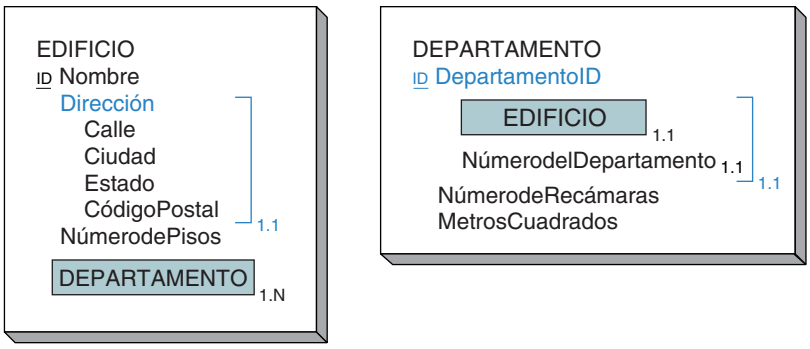
OBJETOS ARQUETIPO-VERSIÓN

El tipo final de un objeto es el **objeto arquetipo-versión.** Un objeto arquetipo es un objeto semántico que produce otros objetos semánticos, los cuales representan versiones, publicaciones o ediciones del arquetipo. Por ejemplo, en la figura 4-31 el objeto arquetipo LIBRODETEXTO produce los objetos de versiones de EDICIÓNes. De acuerdo con este modelo, los atributos Título, Autor y Editorial pertenecen al objeto LIBRODETEXTO, y los atributos NúmerodeEdición, FechadePublicación y NúmerodePáginas pertenecen a la EDICIÓN del LIBRODETEXTO.

El grupo ID en EDICIÓN consta de dos partes: LIBRODETEXTO y Número de edición; éste es el patrón general para una ID de un objeto de versión. La primera parte de la ID contiene al objeto arquetipo y la segunda parte es un atributo simple que identifica la versión dentro del arquetipo. La figura 4-32 muestra otro ejemplo de los objetos arquetipo-versión.

➤ FIGURA 4-32

Otro ejemplo de un objeto arquetipo-versión



➤ COMPARACIÓN DEL OBJETO SEMÁNTICO CON EL MODELO E-R

El modelo E-R y el de objeto semántico tienen similitudes y diferencias. Son similares en tanto que son herramientas para entender y documentar la estructura de datos de los usuarios. Ambos procuran modelar la estructura de las cosas en el mundo de los usuarios, así como las relaciones entre ellos.

La diferencia principal entre los dos modelos es la orientación. El modelo E-R considera básico el concepto de *entidad*. Las entidades y sus relaciones son consideradas los átomos, por así decirlo, de un modelo de datos. Estos átomos pueden combinarse para formar lo que el modelo E-R denomina *puntos de vista del usuario*, los cuales son combinaciones de entidades cuyas estructuras son similares a la del objeto semántico.

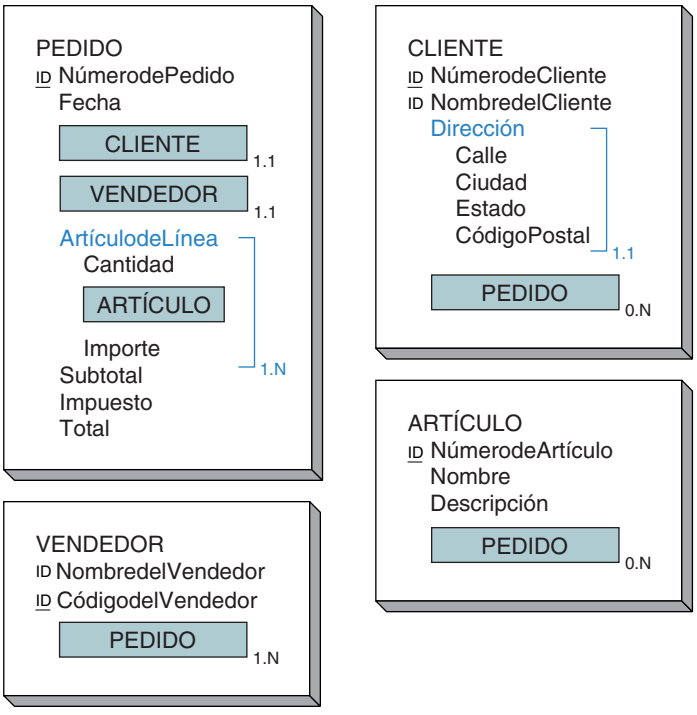
El modelo de objeto semántico considera como básico al concepto del *objeto semántico*. El conjunto de objetos semánticos en un modelo de datos es un mapa de la estructura esencial de las cosas que el usuario considera importantes. Estos objetos son los átomos del mundo de los usuarios y, por lo tanto, constituyen las unidades más pequeñas distinguibles que los usuarios desean procesar. Se pueden desglosar en partes aún más pequeñas dentro del DBMS (o la aplicación), pero esas partes pequeñas no son de interés o utilidad para los usuarios.

De acuerdo con la perspectiva del objeto semántico, las entidades, como se definen en el modelo E-R, no existen, sólo son piezas o trozos de las entidades reales. Las únicas entidades que tienen significado para los usuarios son, de hecho, los objetos semánticos. Otra manera de establecer esto es afirmar que los objetos semánticos son *semánticamente autocontenidos*, o que son *semánticamente completos*. Considere un ejemplo. La figura 4-33 muestra cuatro objetos semánticos: PEDIDO, CLIENTE, VENDEDOR y ARTÍCULO. Cuando un usuario dice: "Muéstrame el pedido número 2000", quiere decir que se le muestre el PEDIDO como está modelado en la figura 4-33. Esto incluye, entre otros atributos, los datos de CLIENTE. Debido a que CLIENTE es parte de PEDIDO, el objeto PEDIDO incluye a CLIENTE.

La figura 4-34 es un modelo E-R de los mismos datos y contiene las entidades de PEDIDO, CLIENTE, VENDEDOR, ARTÍCULO-DE-LÍNEA e INVENTARIO. La entidad PE-

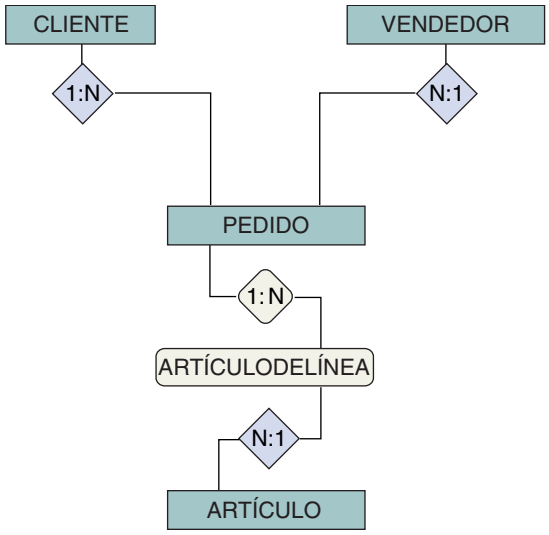
➤ FIGURA 4-33

PEDIDO y objetos semánticos relacionados



➤ FIGURA 4-34

Modelo entidad-relación de PEDIDO y CLIENTE



DIDO incluye los atributos NúmerodePedido, Fecha, Subtotal, Impuestos y Total. Si un usuario dijera: "Enséñame el pedido número 2000" y sólo obtuviera los atributos Fecha, Subtotal, Impuesto y Total, se sentiría desilusionado. Quizá la respuesta de la mayoría de los usuarios sería: "¿Dónde están los demás datos?" Esto es, la entidad PEDIDO no representa el significado que los usuarios tienen de la identidad distinta PEDIDO. La entidad sólo es una parte de PEDIDO.

Al mismo tiempo, cuando un usuario (quizás el mismo) dice: "Muéstrame al cliente 12345", seguramente desea que se le muestren todos los datos modelados para CLIENTE en la figura 4-33, incluyendo NombredelCliente, todos los atributos del grupo Dirección y todos los PEDIDOs relacionados con dicho CLIENTE. En la figura 4-34 la entidad CLIENTE sólo tiene los atributos NombredelCliente, Calle, Ciudad, Estado, CódigoPostal. Si el usuario dijera: "Muéstrame al cliente ABC" y se le dieran sólo estos datos, de nuevo reclamaría: "No, esto sólo es parte de lo que quiero."

De acuerdo con la visión del objeto semántico, las entidades E-R no son necesarias. Los objetos semánticos pueden transformarse fácilmente en diseños de bases de datos sin haber considerado antes las entidades del modelo E-R. Son, por así decirlo, casas incompletas construidas en el proceso de cambio del paradigma de las estructuras de datos computacionales al paradigma del usuario.

Otra diferencia es que los objetos semánticos contienen más metadatos que las entidades. En la figura 4-33 el modelo de objeto semántico registra el hecho de que NúmerodeCliente es un identificador único en la mente de los usuarios. Puede o no utilizarse como un identificador para la tabla principal, pero esto no es importante para el modelo de datos. Además, el Nombre del cliente no es el único identificador para los usuarios. Por otra parte, los objetos semánticos representan el hecho de que existe un grupo semántico de atributos denominado *Dirección*. Este grupo contiene otros atributos que forman la dirección. El hecho de que este grupo exista será importante cuando se diseñen las formas y los reportes. Finalmente, los objetos semánticos indican que un ARTÍCULO puede estar relacionado con más de un PEDIDO, pero sólo puede estar relacionado con un ArtículodeLínea dentro de dicho PEDIDO. Este hecho no puede mostrarse en el diagrama de entidad-relación.

En el análisis final, decida cuál de las figuras 4-33 y 4-34 le dan una mejor vista de lo que debe contener una base de datos. Mucha gente descubre que los límites dibujados alrededor de los objetos semánticos y los paréntesis en torno a los atributos grupales les ayudan a tener una mejor vista de la imagen global del modelo de datos.

➤ RESUMEN

Los modelos E-R y de objeto semántico se utilizan para interpretar los requerimientos y construir los modelos de datos de los usuarios. Estos modelos son como lentes a través de los cuales los analistas observan cuando estudian y documentan los datos de los usuarios. Finalmente, ambos llevan al diseño de una base de datos.

Un objeto semántico es un conjunto designado de atributos que describen suficientemente a una entidad distinta. Los objetos semánticos se agrupan en clases y las clases y las instancias de los objetos semánticos tienen nombres. Por ejemplo, el nombre de una clase es EMPLEADO y el nombre de una instancia es Empleado 2000.

Los atributos representan características de las identidades que están siendo representadas. El conjunto de atributos es suficiente en tanto que represente todas las características que el usuario necesita para realizar su trabajo. Los objetos representan identidades distintas, instancias que los usuarios consideran como independientes y por separado. Las identidades distintas representadas pueden o no ser físicas; pero sí son representativas, como por ejemplo un contrato.

Los atributos de objetos pueden ser datos simples, grupos u otros objetos semánticos. Los diagramas de objetos sintetizan la estructura de los objetos. Los nombres de los objetos se escriben con letras mayúsculas en la parte superior del diagrama. Los atributos de los no objetos se indican con mayúsculas iniciales, y los atributos grupales se indican con paréntesis.

Todos los atributos tienen una cardinalidad mínima que indica cuántas instancias del atributo deben existir para que el objeto sea válido. Asimismo, tienen una cardinalidad máxima que indica el número máximo de instancias del atributo permitidas. La cardinalidad se escribe en el formato N.M, donde N es la cardinalidad mínima y M es la máxima. Para reducir el desorden en un diagrama de objetos, si la cardinalidad del atributo de un valor simple es 0.1, no se muestra. Pero la cardinalidad siempre se muestra para los atributos de objeto y grupales.

Los atributos de objeto siempre se parean. Si un primer objeto tiene una relación con un segundo objeto, este último debe tener una relación con el primero. Los identificadores de objetos son atributos que sirven, en la mente de los usuarios, para identificar objetos. Los identificadores pueden ser únicos o no únicos. Cualquier tipo de atributo puede ser un identificador. Los identificadores se muestran con las letras ID junto al atributo. Si éste es único, entonces se subrayan las letras ID. Por lo general las cardinalidades de un identificador son 1.1.

El dominio de un atributo es el conjunto de todos los valores posibles que puede tener. Los dominios tienen una definición física y una semántica. Existen tres tipos de dominios: simple, grupal y de objeto semántico.

Las aplicaciones procesan los objetos a través de las vistas de usuarios. Una vista de un objeto consta del nombre de éste y de todos los atributos visibles desde dicha vista. Con frecuencia la definición de una vista y un objeto es un proceso iterativo.

El proceso de desarrollar un conjunto de diagramas de objetos es iterativo. Se examinan las formas y los reportes; se documenta un conjunto inicial de objetos, y se revisan los reportes y las formas nuevas para revelar los objetos y cambios en los objetos existentes. Este proceso continúa hasta que se hayan examinado todas las formas y reportes.

Existen siete tipos de objetos. Los objetos simples no tienen atributos de valores múltiples ni atributos de objetos. Los objetos compuestos tienen atributos de valores múltiples, pero no tienen atributos de objetos. Los objetos compuestos tienen atributos de objetos, y los objetos híbridos combinan a los objetos combinados y compuestos. Los objetos de asociación relacionan a dos o más objetos distintos. Los objetos de subtipo se utilizan para representar las especializaciones de los objetos. Finalmente, los objetos versión-arquetipo se emplean para modelar objetos que contienen datos de la base junto con versiones o variaciones múltiples.

El modelo E-R considera básicas a las entidades, mientras que el modelo de objeto semántico asume como básicos a los objetos semánticos. El modelo de objeto semántico también contiene más información referente a la interpretación de los datos que el modelo E-R.

➤ PREGUNTAS DEL GRUPO I

4.1 Explique por qué el modelo E-R y el modelo de objeto semántico son como lentes.

4.2 Defina al *objeto semántico*.

4.3 Explique la diferencia entre el nombre de la clase de un objeto y el nombre de una instancia del objeto. Proporcione un ejemplo sobre cada uno.

4.4 ¿Qué se requiere para que un conjunto de atributos sea una descripción suficiente?

4.5 Explique las palabras *identidad bien definida* cuando corresponden a la definición de un objeto semántico.

4.6 Explique por qué una línea de artículo de un pedido no es un objeto semántico.

4.7 Enumere los tres tipos de atributos.

4.8 Dé un ejemplo sobre cada uno de los siguientes atributos:

a. un atributo simple de un valor único
b. un atributo grupal de un valor único
c. un atributo simple de valores múltiples
d. un atributo grupal de valores múltiples
e. un atributo de objeto simple
f. un atributo de objeto de valores múltiples

4.9 ¿Qué es la cardinalidad mínima? ¿Cómo se utiliza? ¿Qué tipo de atributos tiene?

4.10 ¿Qué es la cardinalidad máxima? ¿Cómo se utiliza? ¿Qué tipo de atributos tiene?

4.11 ¿Qué son los atributos pareados? ¿Por qué son necesarios?

4.12 ¿Qué es un identificador de objeto? Mencione un ejemplo referente a un identificador de objeto de atributo simple, y un ejemplo de un identificador de objeto de un atributo grupal.

4.13 Defina el *dominio de un atributo*. ¿Cuáles son los tipos de dominio de un atributo? ¿Por qué es necesaria una descripción semántica?

4.14 ¿Qué es una vista del objeto semántico? Dé un ejemplo sobre un objeto y dos vistas diferentes a los ejemplos que se mencionan en este texto.

4.15 Mencione un ejemplo de un objeto simple, diferente al que analizamos en este capítulo.

4.16 Proporcione tres ejemplos de objetos compuestos diferentes a los de este capítulo. Uno de los ejemplos deberá tener sólo un atributo simple de valores múltiples; el otro, dos grupos de valores múltiples independientes; y el tercero, grupos de valores múltiples anidados.

4.17 Mencione un ejemplo de cuatro conjuntos de objetos combinados diferentes a los que analizamos en este capítulo. Un conjunto deberá tener una relación 1:1; el otro, una relación 1:N; otro más, una relación M:1, y el conjunto restante una relación M:N.

4.18 Mencione un ejemplo de un objeto híbrido diferente al de este capítulo.

4.19 Proporcione un ejemplo de un objeto de asociación, y dos objetos compuestos diferentes a los analizados en este capítulo.

4.20 Proporcione un ejemplo de un objeto de supertipo con tres objetos de subtipo diferentes a los del presente capítulo.

4.21 Dé un ejemplo referente a los objetos arquetipo-versión diferente al que se proporcionó en este capítulo.

4.22 Explique las similitudes entre el modelo E-R y el modelo de objeto semántico.

4.23 Explique las diferencias principales entre el modelo E-R y el modelo de objeto semántico.

4.24 Explique la razón por la cual las entidades, según lo define el modelo E-R, no existen realmente.

4.25 Muestre cómo el modelo E-R y el modelo de objeto semántico representarían los datos que fundamentan la forma de PEDIDO en la figura 4-24(a), y explique las diferencias principales.

➤ PREGUNTAS DEL GRUPO II

4.26 Modifique el diagrama del objeto semántico en la figura 4-13 para incluir a los objetos CLASE, PRESENTACIÓN DE LA CLASE, e INSCRIPCIÓN. Suponga que INSCRIPCIÓN es un objeto de asociación que relaciona a un ESTUDIANTE con una CLASE.

4.27 Modifique el diagrama del objeto semántico en la figura 4-13 para incluir al objeto COMITÉ. Suponga que varios PROFESORes son asignados a un comité y que un COMITÉ incluye a varios PROFESORes. Cree el objeto JUNTA como un objeto arquetipo-versión, el cual represente las juntas de un COMITÉ.

4.28 Modifique su respuesta a la pregunta 4.27, de tal manera que JUNTA sea un grupo de valores múltiples dentro de COMITÉ. ¿Este modelo es mejor que el de la pregunta 4.27? Justifique su respuesta.

➤ PROYECTOS

A. Desarrolle un modelo de objeto semántico para el caso AMV en el proyecto "A" al final del capítulo 3.

B. Ingrese a la página Web de un fabricante de computadoras, por ejemplo Dell (www.dell.com). Utilice el sitio de Internet para determinar qué computadora portátil compraría para un usuario potencial que tiene un presupuesto de $10,000. Mientras usa el sitio de Internet, piense en la estructura de una posible base de datos de sistemas y subsistemas computacionales para apoyar este sitio.

Desarrolle un modelo de objeto semántico de la base de datos de sistemas y subsistemas computacionales para este sitio de Internet. Los objetos posibles son SISTEMA DE BASE, OPCIÓN DE MEMORIA, TARJETA DE VÍDEO e IMPRESORA. Muestre las relaciones de los objetos y cuando menos dos o tres atributos por objeto. Indique el tipo de cada objeto semántico. Para evitar que este proyecto se dispare en tamaño, restrinja su diseño a las necesidades de alguien que realiza una decisión de compra.

C. Ingrese al sitio de una librería como Amazon (www.amazon.com) y seleccione tres de los mejores libros sobre XML (Lenguaje extendido de marcaje) para alguien que apenas está comenzando a estudiar ese tema. Conforme use el sitio de Internet, piense en la estructura de una posible base de datos de libros, autores, títulos y temas relacionados.

Desarrolle un modelo de objeto semántico de una base de datos de libros para ese sitio de Internet. Los objetos posibles son TÍTULO, AUTOR, EDITORIAL y TEMA. Muestre las relaciones de los objetos y cuando menos dos o tres atributos por objeto. Indique el tipo de cada objeto semántico. Para evitar que este proyecto se dispare en tamaño, suponga que sólo dará seguimiento a libros; además, restrinja su diseño a las

necesidades de alguien que busca comprar libros. No considere el pedido del cliente, surtido de la mercancía, ni otros procesos de negocio.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Considere la situación de la compañía FiredUp que analizamos al final de los capítulos 1 y 2. Suponga que esa compañía ha desarrollado una línea de tres estufas nuevas: FiredNow, FiredAlways y FiredAtCamp. Además suponga que los dueños están vendiendo refacciones para cada línea y también hacen reparaciones. Algunas reparaciones se llevan a cabo sin ningún cargo debido a que todavía tienen garantía; en otras, sólo se cobran las refacciones; y en algunas más se cobran la mano de obra y las refacciones. La compañía FiredUp desea realizar un seguimiento de todos estos datos. Cuando se les pidieron detalles adicionales los dueños hicieron la siguiente lista:

CLIENTE: Nombre, Dirección, NúmerodeDepartamento, Ciudad, Estado/Provincia, CódigoPostal, País, CorreoElectrónico, NúmeroTelefónico

ESTUFA: NúmerodeSerie, Tipo, FechadeFabricación, InicialesdelInspector

FACTURA: NúmerodeFactura, Fecha, Cliente, con una lista de artículos vendidos y sus precios, PrecioTotal

REPARACIÓN: NúmerodeReparación, Cliente, Estufa, Descripción, con una lista de las refacciones que se utilizaron y su costo, si es el caso, y CantidadTotal de la reparación

PARTE: Número, Descripción, Costo, PreciodeVenta

A. Cree un conjunto de objetos semánticos para una base de datos de la compañía FiredUp. Establezca las cardinalidades mínima y máxima de todos los atributos como lo considere apropiado. Explique las razones en las que se basó para establecer cada valor de cardinalidad. Utilice tantos tipos de objetos semánticos como considere necesario, pero no utilice los subtipos. Esta tarea es más fácil si descarga el *Tabledesigner*, que es la herramienta que se describe en el apéndice B para hacer el diagrama de un objeto semántico. Utilícela para definir sus objetos semánticos e imprimir los reportes de los objetos semánticos que le propondrá a su instructor. Vea el apéndice B y pregunte a su instructor antes de llevar a cabo esto.

B. Modifique sus diagramas de objetos en su respuesta a la pregunta A mediante la representación de FACTURA y REPARACIÓN como subtipos. ¿Bajo qué circunstancias es mejor este diseño que el de su respuesta a la pregunta A?

C. Suponga que la compañía FiredUp desea hacer un seguimiento de números telefónicos particulares, números de fax y teléfonos celulares, así como de las múltiples direcciones de correo electrónico de sus clientes. Modifique sus objetos para tomar en cuenta los valores múltiples de NúmeroTelefónico y CorreoElectrónico.

D. Suponga que la compañía FiredUp desarrolla diferentes versiones del mismo producto de estufa. Por lo tanto, desarrollaron "FiredNow", versión 1, y "FiredNow", versión 2, etc. Modifique sus objetos en la anterior pregunta A como sea necesario para responder a esta situación.

E. Cuando se les pregunta a los usuarios sobre los datos que desean registrar, no siempre recuerdan todo lo que necesitan. Empleando su conocimiento sobre las operaciones de un negocio pequeño, haga una lista de los objetos semánticos que los usuarios podrían haber olvidado. Asegúrese de mostrar las relaciones entre los objetos. ¿Cómo determinará si se necesitan cualquiera de estos datos adicionales en la compañía FiredUp?

F. Si usted contestó las preguntas del proyecto de la compañía FiredUp al final del capítulo 3, compare el modelo entidad-relación desarrollado para dicho capítulo con el modelo de objeto semántico que estudiamos aquí. ¿Cuál de los dos prefiere? ¿Cuál cree que entenderían mejor los dueños de la compañía FiredUp?

PARTE III

DISEÑO DE BASES DE DATOS

Los capítulos correspondientes a la parte III analizan el diseño de bases de datos. El capítulo 5 presenta el modelo relacional y la normalización. El modelo relacional es importante porque con él se expresan la mayoría de los diseños de las bases de datos, y porque es el fundamento de la mayoría de los productos DBMS actuales.

La normalización es importante porque constituye una técnica para comprobar la calidad del diseño relacional. Con base en los conceptos del capítulo 5, en el capítulo 6 abordamos el proceso de transformación de modelos de datos de entidad-relación en diseños relacionales independientes de un DBMS. A continuación, en el capítulo 7 se describe el proceso para transformar modelos de datos de objeto semántico en dichos diseños.

CAPÍTULO 5

El modelo relacional y la normalización

El modelo relacional es importante por dos razones. Primera, debido a que los elementos de construcción del modelo relacional son extensos y generales, y se pueden usar para expresar diseños de bases de datos independientes de un DBMS. Segunda, el modelo relacional es la base de casi todos los productos DBMS. Por lo tanto, es esencial comprender los conceptos de este modelo.

El presente capítulo aborda las bases del modelo relacional y explica los conceptos fundamentales de la *normalización*. Empezaremos por señalar el hecho de que no todas las relaciones son iguales; algunas son mejores que otras. La normalización es un proceso para convertir una relación que tiene ciertos problemas, en dos o más relaciones que no los tienen. Lo que es más importante: la normalización se puede usar como un lineamiento para comprobar la pertinencia y validez de las relaciones. Se han realizado muchos trabajos teóricos en torno a la pregunta: ¿qué es una relación bien estructurada? Este trabajo se llama *normalización* porque uno de los pioneros en la tecnología de bases de datos, E. F. Codd, definió varias *formas normales* de relaciones. En el presente capítulo examinaremos la normalización, incluyendo los resultados de teoremas que son útiles e importantes para los profesionales de bases de datos. Las pruebas de estos teoremas, así como un tratamiento más formal y riguroso de estos temas, los puede encontrar en los trabajos de C. J. Date y J. D. Ullman.[1]

[1] C. J. Date, *An Introduction to Database Systems*, Sexta edición (Reading, MA: Addison-Wesley, 1994); y J. D. Ullman y Jennifer Widom, *A First Course in Database Systems* (Upper Saddle River, NJ: Prentice Hall, 1997).

➤ EL MODELO RELACIONAL

Una **relación** es una tabla bidimensional. Cada renglón tiene datos que pertenecen a alguna cosa o a una parte de ésta. Cada columna de esta tabla contiene datos referentes a un atributo. Algunas veces los renglones se llaman **tuples** y las columnas, **atributos.**

Los términos *relación, tuple* y *atributo* derivan de las matemáticas relacionales, que constituyen la fuente teórica de este modelo. Los profesionales de Sistemas de información gerencial (MIS, por sus siglas en inglés) consideran más adecuada la analogía de los términos *archivo, registro* y *campo*, y la mayoría de los usuarios encuentran más razonables los términos *tabla, renglón* y *columna*. La figura 5-1 resume esta terminología.

Para que una tabla sea una relación debe cumplir ciertas restricciones.[2] Primero, las celdas de la tabla deben ser de un valor único; no se permite repetir grupos ni tener series en calidad de valores.[3] Todas las entradas en cualquier columna deben ser del mismo tipo. Por ejemplo, si la tercera columna en el primer renglón de una tabla contiene NúmerodeEmpleados, entonces la tercera columna también debe contener NúmerodeEmpleados en todos los demás renglones de la tabla. Cada columna tiene un nombre único y el orden de las columnas en la tabla no es importante. Por último, dos renglones en la tabla no pueden ser idénticos y el orden de los renglones no tiene importancia.

La figura 5-2 es un ejemplo de tabla. Observe que hay siete renglones de cuatro columnas. Si fuésemos a reacomodar las columnas (es decir, colocando NúmerodeEmpleado en el extremo izquierdo) o a reordenar los renglones (quizás en orden ascendente por edad), obtendríamos una tabla equivalente.

La figura 5-2 muestra una ocurrencia o instancia de una tabla. El formato generalizado, EMPLEADO (Nombre, Edad, Sexo, NúmerodeEmpleado) se denomina estructura de la relación, y esto es lo que la mayoría de las personas entiende cuando usa el término *relación*.

Para comprender la normalización necesitamos definir dos términos importantes: **dependencia funcional** y **llave.**

DEPENDENCIAS FUNCIONALES

Una *dependencia funcional* es una relación entre uno o más atributos. Suponga que si asignamos el valor de un atributo podemos obtener (o buscar) el valor de otro. Por ejemplo, si conocemos el valor de NúmerodeCuentadelCliente, podemos encontrar el valor de BalancedelCliente. Si esto es verdad, podemos decir que BalancedelCliente es *funcionalmente dependiente* de NúmerodeCuentadelCliente.

En términos generales, el atributo Y es funcionalmente dependiente del atributo X si el valor de X determina al valor de Y. Dicho de otra manera, si conocemos el valor de X podemos obtener el valor de Y.

Las ecuaciones representan dependencias funcionales. Por ejemplo, si conocemos el precio de un artículo y la cantidad de artículos que han sido vendidos, podemos calcular el precio total de esos artículos:

PrecioTotal = PreciodelArtículo × Cantidad

➤ FIGURA 5-1

Terminología relacional equivalente

Modelo relacional	Programador	Usuario
Relación	Archivo	Tabla
Tuple (renglón)	Registro	Renglón
Atributo	Campo	Columna

[2] E. F. Codd, "A Relational Model of Data for Large Shared Databanks", *Communications of the ACM*, junio de 1970, pp. 377-387.

[3] Esto no significa que los valores deban tener una longitud fija. Por ejemplo, un campo de longitud variable es un valor perfectamente legítimo. Sin embargo, se permite *un* solo valor.

➤ FIGURA 5-2

Relación EMPLEADO

	Atributo 1 Nombre	Atributo 2 Edad	Atributo 3 Género	Atributo 4 NúmerodeEmpleado
Tuple 1	Anderson	21	F	010110
Tuple 2	Decker	22	M	010100
.	Glover	22	M	101000
.	Jackson	21	F	201100
.	Moore	19	M	111100
.	Nakata	20	F	111101
Tuple 7	Smith	19	M	111111

En este caso, podríamos decir que PrecioTotal es funcionalmente dependiente de PreciodelArtículo y de Cantidad.

Las dependencias funcionales entre atributos en una relación por lo general no implican ecuaciones. Por ejemplo, suponga que los estudiantes tienen un número único de identificación (ID del estudiante, o EID) y que cada uno tiene una y sólo una especialidad. Considerando el valor de un EID, podemos encontrar la especialidad del estudiante, por lo que la especialidad es funcionalmente dependiente de EID. También considere las microcomputadoras en un laboratorio de cómputo. Cada una tiene únicamente un tamaño de memoria principal, por lo que TamañodeMemoria es funcionalmente dependiente de NúmerodeSeriedelaComputadora.

A diferencia de una ecuación, estas dependencias funcionales no se pueden resolver usando aritmética; en vez de eso, se listan en una base de datos. De hecho, uno puede argumentar que el almacenamiento y la recuperación de las dependencias funcionales es la única razón para tener una base de datos.

Las dependencias funcionales se escriben usando la siguiente notación:

EID : Especialidad
NúmerodeSeriedelaComputadora : TamañodeMemoria

La primera expresión se lee como "EID determina funcionalmente a Especialidad", o "Especialidad es dependiente de EID". Los atributos en el lado izquierdo de la flecha se llaman **determinantes.**

Como se ha establecido, si EID determina a Especialidad, un valor particular de EID será pareado con *un* solo valor de Especialidad. A la inversa, un valor de Especialidad podrá ser pareado con *uno o más* valores diferentes de EID. Suponga que un estudiante tiene un valor EID de 123 y se especializa en contabilidad. Cuando EID y Especialidad se encuentren juntas en una relación, el valor EID de 123 será siempre pareado con el valor de Especialidad en Contabilidad. Sin embargo, lo opuesto no es cierto, ya que Especialidad en Contabilidad se puede parear con varios valores de EID (varios estudiantes pueden tener la especialidad en contabilidad). Consecuentemente, podemos afirmar que la relación de EID con Especialidad es de muchos a uno (N:1). En general, se puede decir que A determina a B, la relación de valores de A y B es N:1.

Las dependencias funcionales pueden involucrar grupos de atributos. Considere la relación CALIFICACIONES (EID, NombredeClase, Calificación). La combinación de un EID y un NombredeClase determina una calificación, una dependencia funcional que se escribe

(EID, NombredeClase) → Calificación

Observe que tanto EID como NombredeClase son necesarios para determinar Calificación. No podemos subdividir la dependencia funcional porque ni EID ni NombredeClase determinan la calificación por sí mismos.

Note la diferencia en los siguientes patrones: Si X → (Y, Z), entonces X → Y y X → Z. Por ejemplo, si EID → (NombredelEstudiante, Especialidad), entonces EID → NombredelEstudiante y EID → Especialidad. Pero si (X, Y) → Z, entonces en general no es

➤ FIGURA 5-3

Relación ACTIVIDAD

ACTIVIDAD (EID, Actividad, Cuota)
Llave: EID
Datos de muestra

EID	Actividad	Cuota
100	Esquí	200
150	Natación	50
175	Squash	50
200	Natación	50

verdad que X → Y o Y → Z. Por lo tanto, si (EID, NombredeClase) → Calificación, entonces ni EID ni NombredeClase determinan por sí mismos a Calificación.

Llaves

Una *llave* es un grupo de uno o más atributos que identifica únicamente a un renglón. Considere la relación ACTIVIDAD en la figura 5-3, cuyos atributos son EID, Actividad y Cuota. El significado de un renglón es que un estudiante se inscribe en una actividad con la cuota especificada. Se supone que al estudiante sólo se le permite participar en una sola actividad a la vez. En este caso, un valor de EID determina un único renglón y esto es una llave.

Las llaves también pueden estar compuestas por un grupo de atributos tomados en conjunto. Por ejemplo, si a los estudiantes se les permitiera inscribirse en distintas actividades al mismo tiempo, sería posible que un valor de EID apareciera en dos o más renglones de la tabla, y de esta forma EID podría no identificar únicamente a un renglón. Quizá se requerirían algunas combinaciones de atributos (EID, Actividad).

Por otra parte, existe un sutil pero importante aspecto en el párrafo anterior. Si los atributos son llaves o no, y si son dependencias funcionales o no, esto no se determina a través de un conjunto de reglas abstractas, sino mediante los modelos mentales de los usuarios, y con base en las reglas del negocio de la empresa u organización que utiliza la base de datos. En este ejemplo, ya sea EID la llave (EID, Actividad), o alguna otra combinación, la llave se determina por completo mediante la semántica fundamental de las personas de la empresa que usan la base de datos. Debemos preguntar a los usuarios para resolver estas interrogantes. Recuerde que todas las suposiciones que hagamos acerca de dependencias funcionales, llaves, restricciones y similares están determinadas por los modelos mentales de los usuarios.

Después de entrevistar a los usuarios, suponga que descubrimos que los estudiantes tienen, en efecto, permiso para participar en varias actividades al mismo tiempo. Esta situación se representa mediante la relación ACTIVIDADES, que se muestra en la figura 5-4. Como establecimos, EID *no* es una llave de esta relación. Por ejemplo, el estudiante 100 se ha inscrito en esquí y en golf, y el valor EID de 100 aparece en dos ren-

➤ FIGURA 5-4

Relación ACTIVIDADES

EID	Actividad	Cuota
100	Esquí	200
100	Golf	65
150	Natación	50
175	Squash	50
175	Natación	50
200	Natación	50
200	Golf	65

glones diferentes. De hecho, para esta relación, ningún atributo solo es una llave, por lo que ésta debe ser una combinación de dos o más atributos.

Considere la combinación de dos atributos de esta tabla. Existen tres posibilidades: (EID, Actividad), (EID, Cuota) y (Actividad, Cuota). ¿Alguna de estas combinaciones es una llave? Si es así, únicamente debe identificar un renglón. Nuevamente, para decidir preguntas como ésta debemos consultar con los usuarios. No podemos depender de datos muestra como los de la figura 5-4, o confiar en nuestras propias suposiciones para tomar una decisión.

Después de hablar con los usuarios, suponga que descubrimos que las distintas actividades pueden tener la misma cuota. En este caso, la combinación (EID, Cuota) no puede determinar un renglón único. Por ejemplo, el estudiante 100 puede inscribirse en dos actividades diferentes cuyo costo en conjunto es de $200. Esto significaría que la combinación (100, $200) ocurre dos veces en la tabla, así que esta combinación no puede ser una llave.

¿Puede ser una llave la combinación (Actividad, Cuota)? ¿La combinación (Esquí, $200) determina un renglón único? No, debido a que muchos estudiantes pueden participar en esquí. ¿Qué pasa con (EID, Actividad)? Considerando lo que nos han dicho los usuarios, ¿una combinación de valores para EID y Actividad puede determinar un renglón único? Sí, en tanto no necesitemos conservar registros de las diferentes ocasiones en las que un estudiante se inscribe en una actividad determinada. En otras palabras, ¿esta tabla será usada para registrar sólo las actividades actuales de los estudiantes, o se supone que también se conservarán los registros de las actividades anteriores?

Nuevamente, debemos consultar con los usuarios para responder esta pregunta. Supongamos que nos dicen que sólo hay que conservar los registros de actividades actuales. Entonces la combinación (EID, Actividad) puede determinar un renglón único, y consecuentemente (EID, Actividad) es la llave para esta relación. Si los usuarios especificaron que se conservaran los registros de actividades actuales y pasadas, la relación en la figura 5-3 tendrá renglones duplicados. Debido a que esto lo prohibe la definición de relación, necesitaremos agregar otros atributos tales como Fecha.

Esto nos genera un punto importante. Cada relación tiene cuando menos una llave. Esto puede ser verdad porque no hay relación que pueda tener renglones duplicados y, por lo tanto, en el extremo, las llaves constan de todos los atributos de la relación.

Dependencia funcional, llaves y unicidad

Muchos estudiantes confunden los conceptos dependencia funcional, llaves y unicidad. Para evitar esta confusión, considere lo siguiente: un determinante de una dependencia funcional puede o no ser única en una relación. Si sabemos que A determina a B y que A y B están en la misma relación, aún no sabemos si A es única en esta relación. Sólo tenemos conocimiento de que A determina a B.

Por ejemplo, en la relación ACTIVIDADES, Actividad determina funcionalmente a Cuota, e incluso puede haber varias instancias de una Actividad particular en la relación. La dependencia funcional sólo establece que cada vez que Actividad ocurre con Cuota, ésta siempre ocurre con el mismo valor de Cuota. Esto es, esquí siempre costará $200, sin importar cuántas veces aparezca el valor Esquí en la tabla.

A diferencia de los determinantes, las llaves son siempre únicas. Una llave determina funcionalmente a todo el renglón. Si se duplica el valor de la llave, se duplicará el tuple completo. Pero lo anterior no está permitido porque, por definición, los renglones en una relación deben ser únicos. Así, cuando decimos que un atributo (o una combinación) es una llave, sabemos que será única. Si (EID, Actividad) es una llave, por ejemplo, la combinación (100, Esquí) sólo ocurrirá una vez en una relación.

Para comprobar su comprensión sobre estos conceptos, intente explicar por qué, en la relación ACTIVIDAD de la figura 5-3, EID es tanto un determinante como una llave, pero Actividad es un determinante y no una llave. (Recuerde que la relación en la figura 5-3 refleja la política de que un estudiante puede participar, cuando mucho, en una actividad.)

➤ NORMALIZACIÓN

Por desgracia, no todas las relaciones son igualmente deseables. Una tabla que cumple la mínima definición de una relación puede no tener una estructura eficaz o apropiada. Para algunas relaciones el cambio de datos puede tener consecuencias indeseables, llamadas **anomalías de modificación.** Éstas se pueden eliminar redefiniendo la relación entre dos o más relaciones. En la mayoría de los casos son preferibles las relaciones redefinidas, o **normalizadas.**

ANOMALÍAS DE MODIFICACIÓN

Nuevamente considere Actividad en la figura 5-3. Si elimin——ramos el tuple para Estudiante 100, perderíamos no sólo el hecho de que el estudiante 100 es un esquiador, sino también que esquiar cuesta $200. Esto se llama **anomalía de eliminación;** esto es, al suprimir los hechos acerca de una entidad (que Estudiante 100 es un esquiador), inadvertidamente eliminamos los hechos de otra entidad (que esquiar cuesta $200). Con una supresión, o borrado, se pierden hechos acerca de dos entidades.

La misma relación se puede usar para ilustrar una **anomalía de inserción.** Suponga que deseamos almacenar el hecho de que bucear cuesta $175, pero no podemos ingresar este dato en la relación ACTIVIDAD hasta que un estudiante bucee. Esta restricción parece ridícula. ¿Por qué tendríamos que esperar hasta que alguien realizara la actividad para registrar el precio? A esta restricción se le llama anomalía de inserción. No podemos insertar un hecho de una entidad hasta que tengamos un hecho adicional acerca de otra entidad.

La relación en la figura 5-3 se puede usar para algunas aplicaciones, pero esto obviamente representa problemas. Podemos eliminar tanto las anomalías como la inserción de anomalías dividiendo la relación ACTIVIDAD en dos, cada una relacionada con un tema diferente. Por ejemplo, podemos poner los atributos EID y Actividad en una relación (a la nueva relación la llamaremos ESTUD-ACT-para denotar actividad de estudiante) y colocar los atributos de Actividad y Cuota en otra relación llamada COSTO-ACT (para costo de actividad). La figura 5-5 contiene los mismos datos de muestra almacenados en esas dos nuevas relaciones.

Ahora, si borramos Estudiante 100 de ACT-ESTUD, no perdemos el hecho de que esquiar cuesta $200. Además, podemos agregar buceo y sus cuotas a la relación COSTO-ACT, aun antes de cualquier inscripción. Por lo tanto, han quedado suprimidas las anomalías de eliminación e inserción.

Sin embargo, separar una relación en dos tiene una desventaja. Suponga que un estudiante trata de inscribirse en una actividad que no existe. Por ejemplo, Estudiante 250 quiere registrarse en ráquetbol. Podemos insertar este nuevo tuple en ACT-ESTUD (el renglón puede contener 250, Ráquetbol), pero ¿debemos hacerlo? ¿ Un estudiante podrá registrarse en una actividad que no esté en la relación COSTO-ACT? Dicho de otra manera, ¿el sistema podrá prevenir de algún modo que se agreguen renglones de estudiantes si el valor de ACTIVIDAD no está en la tabla de COSTO-ACT? La respuesta a esta pregunta se encuentra en los requerimientos del usuario. Si la acción estuviera prohibida, esta restricción (un tipo de regla del negocio) se debería documentar como parte del esquema de diseño. Posteriormente, en la puesta en marcha, la restricción se-

➤ FIGURA 5-5

División de ACTIVIDAD en dos relaciones

ESTUD-ACT (EID, Actividad)
Llave: EID

EID	Actividad
100	Esquí
150	Natación
175	Squash
200	Natación

COSTO-ACT (Actividad, Cuota)
Llave: Actividad

Actividad	Cuota
Esquí	200
Natación	50
Squash	50

rá definida en el DBMS si el producto en uso proporciona la verificación de restricciones; de lo contrario, dicha restricción debe imponerse mediante programas de aplicación.

Suponga que el usuario especifica que las actividades pueden existir antes de que cualquier estudiante se inscriba en ellas, pero que no se puede inscribir en una actividad que no tenga una cuota asignada (esto es, en ninguna actividad que no se encuentre en la tabla COSTO-ACT). Podemos documentar esta restricción en alguna de las varias formas durante el diseño de la base de datos: Actividad en ESTUD-ACT es un subconjunto de Actividad en COSTO-ACT, o ESTUD-ACT [Actividad] es un subconjunto de COSTO-ACT [Actividad], o ESTUD-ACT [Actividad] $\subseteq$ COSTO-ACT [Actividad].

De acuerdo con esta notación, los corchetes [ ] denotan una columna de datos que se extrae de una relación. Estas expresiones simplemente significan que el valor en el atributo Actividad de ESTUD-ACT debe existir en el atributo Actividad de COSTO-ACT. También significa que antes de permitir que Actividad sea introducida en ESTUD-ACT, debemos comprobar y asegurarnos que ya esté presente en COSTO-ACT. Restricciones como esta se llaman **restricciones de integridad referencial,** o **restricciones de interrelación.**

Esencia de la normalización

Las anomalías en la relación ACTIVIDAD en la figura 5-3 se pueden establecer de la siguiente forma intuitiva: los problemas ocurren porque ACTIVIDAD contiene hechos acerca de dos temas diferentes:

- Los estudiantes que participan en cada actividad
- Cuánto cuesta cada actividad

Cuando abrimos un nuevo renglón, debemos agregar datos de dos temas al mismo tiempo, y cuando eliminamos un renglón, por fuerza tenemos que suprimir datos de dos temas al mismo tiempo.

¿Recuerda a su maestro de español de segundo de secundaria? Él o ella le decían que un párrafo podía tener un solo tema o idea. Si el párrafo tenía más de un tema, le enseñaban cómo dividirlo en dos o más párrafos, de tal forma que cada uno tuviera sólo un tema. Instrucciones similares se aplican a las relaciones. Cada relación normalizada tiene sólo un tema. Cualquier relación que tenga dos o más temas debe dividirse en dos o más relaciones, que tengan cada una un tema. Este proceso es la esencia de la normalización. Cuando encontramos una relación con anomalías de modificación las eliminamos separando la relación en dos o más, de tal forma que cada una contenga sólo un tema.

Sin embargo, cada vez que dividimos una relación podemos crear restricciones de integridad referencial. Por lo tanto, recuerde verificar estas restricciones cada vez que divida una relación en dos o más.

En el resto de este capítulo aprenderá diversas reglas acerca de la normalización. Todas se refieren a instancias especiales de los procesos que acabamos de describir.

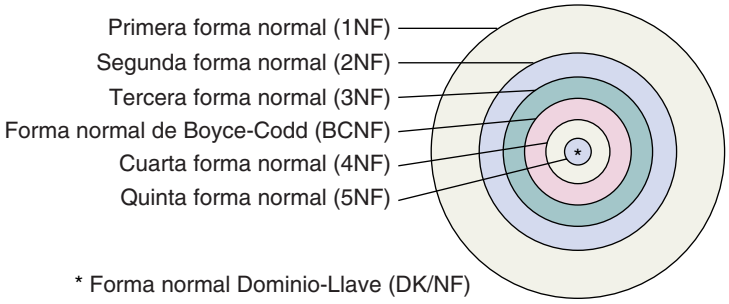
Clases de relaciones

Las relaciones se pueden clasificar por tipos de anomalías de modificación a las cuales son vulnerables. En la década de 1970, los teóricos relacionales investigaron acerca de estos tipos. Cuando alguien encontraba una anomalía, la clasificaba y pensaba en una manera de prevenirla. Cada vez que esto sucedía se mejoraban los criterios de diseño de las relaciones. Estas clases de relaciones, así como las técnicas para prevenir anomalías, se denominan **formas normales.** Dependiendo de su estructura, una relación puede estar en la primera forma normal, en la segunda forma normal, o en alguna otra.

En un trabajo posterior a su artículo de 1970, Codd y otros definieron las primera, segunda y tercera formas normales (1NF, 2NF, 3NF). Más tarde se especificó la forma normal de Boyce-Codd (BCNF), y después se definieron la cuarta y la quinta formas. Como se muestra en la figura 5-6, estas formas normales están anidadas. Esto es, una

➤ FIGURA 5-6

Relación de formas normales



relación en la segunda forma normal está también en primera forma normal, y una relación en 5NF (quinta forma normal) está asimismo en 4NF, BCNF, 3NF, 2NF y 1NF.

Estas formas normales fueron útiles, aunque tenían una seria limitación: ninguna teoría garantizaba que cualquiera de ellas podría eliminar todas las anomalías; cada forma podría quitar sólo algunos defectos. Sin embargo, esto cambió cuando en 1981 R. Fagin definió una nueva forma normal llamada **forma normal dominio-llave (DK/NF, por sus siglas en inglés).** En un importante artículo, Fagin mostró que una relación en DK/NF está libre de todas las anomalías de modificación, sin importar de qué tipo sean.[4] También enseñó que cualquier relación libre de anomalías de modificación debe estar en la forma DK/NF.

Hasta que se definió DK/NF era necesario que los teóricos de bases de datos relacionales buscaran más y más anomalías y más y más formas normales. Sin embargo, la prueba de Fagin simplificó la situación. Si se puede poner una relación en DK/NF, entonces se puede asegurar que no habrá anomalías. El truco es saber cómo poner las relaciones en DK/NF.

➤ DE LA PRIMERA A LA QUINTA FORMAS NORMALES

Cualquier tabla de datos que cumpla con la manera en que se define una relación se dice que está en la **primera forma normal.** Recuerde que para que una tabla sea una relación, necesita cumplir lo siguiente: las celdas de la tabla deben tener un solo valor y no se permiten ni grupos ni arreglos repetidos como valores. Todas las entradas en cualquier columna (atributos) deben ser de la misma clase. Cada columna debe tener un nombre único, pero el orden de las columnas en la tabla no es importante. Finalmente, dos renglones no pueden ser idénticos en la tabla y su orden no es significativo.

La relación en la figura 5-3 está en la primera forma normal. Sin embargo, como hemos visto, las relaciones en la primera forma normal pueden tener anomalías de modificación. Para eliminarlas, separamos la relación en dos o más relaciones. Cuando lo hacemos, las nuevas relaciones están en otra forma normal —lo cual justamente depende de las anomalías que hemos eliminado, así como de aquellas a las que son vulnerables las nuevas relaciones.

SEGUNDA FORMA NORMAL

Para entender la segunda forma normal considere la relación ACTIVIDADES en la figura 5-4; tiene anomalías de modificación similares a las que examinamos anteriormente. Si eliminamos o borramos el tuple para Estudiante 175, perderemos el hecho de que squash cuesta $50. Asimismo, no podemos registrar una actividad hasta que un estudiante se inscriba en ella. De esta forma, la relación sufre ambas anomalías: eliminación e inserción.

[4] R. Fagin "A Normal Form for Relational Databases That Is Based on Domains and Keys", *ACM Transactions on Database Systems*, septiembre de 1981, pp. 387-415.

El problema con esta relación es que tiene una dependencia que involucra sólo una parte de la llave, que es la combinación (EID, Actividad); pero la relación contiene una dependencia, Actividad → Cuota. El determinante de esta dependencia (Actividad) es sólo parte de la llave (EID, Actividad). En este caso, decimos que, en la tabla, Cuota es *parcialmente dependiente* de la llave. No debe haber anomalías de modificación si Cuota es dependiente de toda la llave. Para eliminar las anomalías debemos separar las relaciones en dos.

Este ejemplo conduce a la definición de la segunda forma normal: *una relación se encuentra en la segunda forma normal si todos los atributos que no son llaves son dependientes de todos los atributos de la llave.* De acuerdo con esta definición, si la relación tiene un solo atributo como su llave, entonces automáticamente está en la segunda forma normal. Debido a que la llave es sólo un atributo, en forma predeterminada cada atributo que no es llave depende de *todos* los atributos de la llave; no puede haber dependencia parcial. Por lo tanto, la segunda forma normal sólo se aplica a relaciones que tienen llaves compuestas.

ACTIVIDADES se puede descomponer para formar dos relaciones en la segunda forma normal. Las relaciones son las mismas que las de la figura 5-5, a saber: ACT-ESTUD y COSTO-ACT. Sabemos que las nuevas relaciones están en la segunda forma normal porque ambas tienen llaves de un solo atributo.

Tercera forma normal

Las relaciones en la segunda forma normal también pueden tener anomalías. Considere la relación VIVIENDA de la figura 5-7(a). La llave es EID y las dependencias funcionales son EID → Edificio y Edificio → Cuota. Estas dependencias surgen porque cada estudiante vive en un solo edificio y ese edificio tiene sólo una cuota. Cada uno de los que viven en Randolph Hall, por ejemplo, paga $3200 por trimestre.

FIGURA 5-7

Eliminación de la dependencia transitiva: (a) relación con dependencia transitiva, y (b) relaciones que eliminan la dependencia transitiva

VIVIENDA (EID, Edificio, Cuota)
Llave: EID
Dependencias funcionales: Edificio → Cuota
EID → Edificio → Cuota

EID	Edificio	Cuota
100	Randolph	3200
150	Ingersoll	3100
200	Randolph	3200
250	Pitkin	3100
300	Randolph	3200

(a)

VIVIENDA-ESTUD (EID, Edificio)
Llave: EID

EID	Edificio
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

EDIF-CUOTA (Edificio, Cuota)
Llave: Edificio

Edificio	Cuota
Randolph	3200
Ingersoll	3100
Pitkin	3100

(b)

Puesto que EID determina Edificio y Edificio determina Cuota, entonces indirectamente EID → Cuota. A un arreglo de dependencias funcionales como a éste se le llama **dependencia transitiva,** ya que EID determina Cuota a través del atributo Edificio.

La llave de VIVIENDA es EID, el cual es un atributo individual y, por lo tanto, la relación está en la segunda forma normal (tanto Edificio como Cuota están determinados por EID). A pesar de esto, VIVIENDA tiene anomalías debido a la dependencia transitiva.

¿Que pasaría si eliminamos el segundo tuple que se muestra en la figura 5-7(a)? Perderíamos no sólo el hecho de que Estudiante 150 vive en Ingersoll Hall, sino también que cuesta $3100 vivir ahí. Ésta es una anomalía de eliminación. ¿Cómo podemos registrar el hecho de que Cuota para Carrigg Hall es de $3500? No podemos, hasta que un estudiante decida mudarse ahí. Ésta es una anomalía de inserción.

Para eliminar las anomalías de una relación en la segunda forma normal debe suprimirse la dependencia transitiva, lo que conduce a una definición de la tercera forma normal: *una relación se encuentra en la tercera forma normal si está en la segunda forma normal y no tiene dependencias transitivas*.

La relación VIVIENDA se puede dividir en dos relaciones en la tercera forma normal. Esto se ha hecho para las relaciones VIVIENDA-ESTUD (EID, Edificio) y EDIF-CUOTA (Edificio, Cuota) en la figura 5-7(b).

La relación ACTIVIDAD en la figura 5-3 también tiene una dependencia transitiva. En ACTIVIDAD, EID determina Actividad y Actividad determina Cuota. Por lo tanto, ACTIVIDAD no está en la tercera forma normal. Descomponiendo ACTIVIDAD en las relaciones ACT-ESTUD (Actividad, EID) y COST-ACT (Cuota, Actividad) se eliminan las anomalías.

Forma normal Boyce-Codd

Por desgracia, incluso las relaciones en la tercera forma normal pueden tener anomalías. Considere la relación ASESOR en la figura 5-8(a). Suponga que los requerimientos que sustentan esta relación son que un estudiante (EID) puede tener una o más especialidades (Especialidad), una Especialidad puede tener varios miembros de la facultad (Fnombre) como asesores, y un miembro de la facultad (Fnombre) asesora en una sola área de especialidad. También suponga que dos miembros de la facultad no pueden tener el mismo nombre.

Debido a que los estudiantes pueden tener varias Especialidades, EID no determina la Especialidad. Además, ya que los estudiantes pueden tener varios asesores, EID tampoco determina Fnombre. Entonces, EID, por sí mismo, no puede ser una llave.

La combinación (EID, Especialidad) determina Fnombre y la combinación (EID, Fnombre) determina la Especialidad. De ahí que cualquiera de las combinaciones pueda ser una llave. Dos o más atributos o un conjunto de atributos que puedan ser una llave se denominan **llaves candidatas.** Cualquiera de las candidatas que se seleccione como *la* llave se llama **llave primaria.**

Además de las llaves candidatas, hay otra dependencia funcional que debemos considerar: Fnombre determina la Especialidad (cualquier miembro de la facultad asesora en sólo una especialidad. Por lo tanto, al proporcionar Fnombre podemos determinar la Especialidad). Así, Fnombre es un determinante.

Por definición, ASESOR está en la primera forma normal y también en la segunda, puesto que no tiene un atributo llave (todos los atributos cuando menos son parte de una llave). Asimismo, está en la tercera forma normal porque no tiene una dependencia transitiva. A pesar de esto, tiene anomalías de modificación.

Suponga que el Estudiante 300 abandona la escuela. Si borramos el tuple Estudiante 300 se perderá el hecho de que Perls asesora en psicología. Ésta es una anomalía de eliminación. De igual forma, ¿cómo podemos almacenar el hecho de que Keynes asesora en economía? No podemos, hasta que un estudiante se especialice en economía. Ésta es una anomalía de inserción.

Situaciones como la anterior conducen a la definición de la forma normal Boyce-Codd (BCNF): *una relación está en BCNF si cada determinante es una llave candidata*. ASESOR no está en BCNF porque el determinante, Fnombre, no es una llave candidata.

➤ FIGURA 5-8

Forma Normal Boyce-Codd: (a) relación en la tercera forma normal, pero no en la forma normal Boyce-Cood, y (b) relaciones en la forma normal Boyce-Codd

ASESOR (EID, Especialidad, Fnombre)

Llave (primaria): (EID, Especialidad)
Llave (candidata): (EID, Fnombre)

Dependencias funcionales:
Fnombre → Especialidad

EID	Especialidad	Fnombre
100	Matemáticas	Cauchy
150	Psicología	Jung
200	Matemáticas	Riemann
250	Matemáticas	Cauchy
300	Psicología	Perls
300	Matemáticas	Riemann

(a)

ASESOR-ESTUD(EID-Fnombre)
Llave: EID,Fnombre

EID	Fnombre
100	Cauchy
150	Jung
200	Riemann
250	Cauchy
300	Perls
300	Riemann

ASESOR-MATERIA (Fnombre, Materia)
Llave: Fnombre

Fnombre	Materia
Cauchy	Matemáticas
Jung	Psicología
Riemann	Matemáticas
Perls	Psicología

(b)

Al igual que con los otros ejemplos, ASESOR puede desglosarse en dos relaciones que no tengan anomalías. Por ejemplo, las relaciones ASESOR-ESTUD (EID, Fnombre) y ASESOR-MATERIA (Fnombre, Materia) no tienen anomalías.

Las relaciones en BCNF no tienen anomalías con respecto a dependencias funcionales y esto parece eliminar las anomalías de modificación. Sin embargo, pronto se descubrió que las anomalías pueden surgir de otras situaciones diferentes a las dependencias funcionales.

Cuarta forma normal

Considere la relación ESTUDIANTE de la figura 5-9, la cual muestra la relación entre estudiantes, especialidades y actividades. Suponga que los estudiantes pueden inscribirse en varias especialidades diferentes y participar en diversas actividades también diferentes. En un caso así, la única llave es la combinación de atributos (EID, Especialidad, Actividad). La Estudiante 100 se especializa en música y contabilidad y también participa en natación y tenis. El Estudiante 150 se especializa sólo en matemáticas y participa en carreras.

¿Cuál es la relación entre EID y Especialidad? No es una dependencia funcional porque los estudiantes pueden tener varias especialidades. Un solo valor de EID puede tener varios valores de Especialidad, y un solo valor de EID puede tener varios valores de Actividad.

Esta dependencia de atributos se llama **dependencia multivaluada.** Las dependencias multivaluadas conducen a anomalías de modificación. Para empezar, observe la redundancia de los datos en la figura 5-9. La Estudiante 100 tiene cuatro registros; cada uno muestra una de sus especialidades pareadas con una de sus actividades. Si los datos fueran almacenados con menos renglones —por ejemplo, sólo con dos tuples,

➤ FIGURA 5-9

Relaciones con dependencias multivaluadas

ESTUDIANTE (EID, Especialidad, Actividad)
Llave: (EID, Especialidad, Actividad)

Dependencias multivaluadas: EID →→ Especialidad
EID →→ Actividad

EID	Especialidad	Actividad
100	Música	Natación
100	Contabilidad	Natación
100	Música	Tenis
100	Contabilidad	Tenis
150	Matemáticas	Carreras

uno para música y natación y otro para contabilidad y tenis— las implicaciones podrían ser engañosas. *Parecería* que la Estudiante 100 nadaba sólo cuando tenía la especialidad de música, y jugaba tenis sólo cuando tenía la especialidad de contabilidad. Pero esta interpretación no es lógica. Sus especialidades y actividades son completamente independientes de cualquier otra. Para evitar esta conclusión engañosa almacenamos todas las combinaciones de especialidad y actividades.

Suponga que debido a que la Estudiante 100 decide inscribirse en esquí, agregamos el tuple [100, MÚSICA, ESQUÍ], como se muestra en la figura 5-10(a). La relación en este punto implica que la estudiante 100 esquía cuando toma la especialidad de música, pero no cuando toma la especialidad de contabilidad. Con el fin de mantener la consistencia de los datos, debemos agregar también el renglón [100, CONTABILIDAD, ESQUÍ] como en la figura 5-10(b). Ésta es una anomalía de actualización —hay que hacer demasiadas actualizaciones para realizar un simple cambio en los datos.

En general, hay una dependencia multivaluada cuando una relación tiene cuando menos tres atributos, dos de los cuales son valores múltiples, y sus valores dependen sólo de un tercer atributo. En otras palabras, en la relación R (A, B, C) existe una depen-

➤ FIGURA 5-10

Relaciones ESTUDIANTE con anomalías de inserción: (a) inserción de un solo tuple, y (b) inserción de dos tuples

ESTUDIANTE (EID, Especialidad, Actividad)
Llave: (EID, Especialidad, Actividad)

EID	Especialidad	Actividad
100	Música	Esquí
100	Música	Natación
100	Contabilidad	Natación
100	Música	Tenis
100	Contabilidad	Tenis
150	Matemáticas	Carreras

(a)

EID	Especialidad	Actividad
100	Música	Esquí
100	Contabilidad	Esquí
100	Música	Natación
100	Contabilidad	Natación
100	Música	Tenis
100	Contabilidad	Tenis
150	Matemáticas	Carreras

(b)

➤ FIGURA 5-11

Eliminación de una dependencia multivaluada

ESPECIALIDAD-ESTUD (EID, Especialidad)
Llave: (EID, Especialidad)

EID	Especialidad
100	Música
100	Contabilidad
150	Matemáticas

ACT-ESTUD (EID, Actividad)
Llave: (EID, Actividad)

EID	Actividad
100	Esquí
100	Natación
100	Tenis
150	Carreras

dencia multivaluada si A determina valores múltiples de B; A determina valores múltiples de C, y B y C son independientes entre sí. Como vimos en el ejemplo anterior, EID determina valores múltiples de Especialidad y también valores múltiples de Actividad, pero Especialidad y Actividad son independientes una de otra.

Nuevamente nos remitiremos a la figura 5-9. Observe cómo se escriben las dependencias multivaluadas: EID $\rightarrow\!\!\rightarrow$ Especialidad y EID $\rightarrow\!\!\rightarrow$ Actividad. Esto se lee: "EID multidetermina Especialidad y EID multidetermina Actividad". Esta relación está en BCNF (2-NF porque todo esto es llave; 3NF porque no tiene dependencias transitivas, y BCNF porque no tiene determinantes que no son llaves). Sin embargo, como podemos ver, esto tiene anomalías: si un estudiante agrega otra especialidad debemos ingresar un tuple para ésta y parearlo con cada una de las actividades del estudiante. Lo mismo se aplica si un estudiante se inscribe en una nueva actividad. Si abandona una especialidad debemos eliminar cada uno de los registros que contienen esa especialidad. Si participa en cuatro actividades, debe haber cuatro tuples que contengan la especialidad que ha dejado y deberán borrarse.

Para evitar estas anomalías tenemos que eliminar las dependencias multivaluadas. Esto se hace construyendo dos relaciones, cada una almacena datos para uno solo de los atributos multivalor. Las relaciones resultantes no tienen anomalías. Éstas son ESPECIALIDAD-ESTUD (EID, Especialidad) y ACT-ESTUD (EID, Actividad), como se muestra en la figura 5-11.

A partir de las observaciones anteriores, definimos la cuarta forma normal de la siguiente manera: *una relación se encuentra en la cuarta forma normal si está en BCNF y no tiene dependencias multivaluadas*. Después de analizar la forma dominio-llave en este capítulo, describiremos las dependencias multivaluadas en otra forma más intuitiva.

Quinta forma normal

La quinta forma normal se refiere a dependencias que son más raras. Esto tiene que ver con relaciones que pueden dividirse en subrelaciones, como lo hemos hecho, pero que no se pueden reconstruir. La condición bajo la cual se origina esta situación no tiene un significado claro, intuitivamente hablando. No sabemos cuáles son los resultados de estas dependencias, ni si tienen alguna consecuencia práctica. Para más información acerca de la quinta forma normal consulte el trabajo de Date que citamos al principio de este capítulo.

➤ FORMA NORMAL DOMINIO-LLAVE

Cada una de las formas normales ya analizadas las identificaron aquellos investigadores que encontraron anomalías con algunas relaciones, las cuales estaban en una forma normal más baja: observaron que las anomalías de modificación en las relaciones de la segunda forma normal conducen a la definición de la tercera forma normal, y así sucesivamente. Aunque cada forma normal resuelve alguno de los problemas que han sido identificados con la anterior, nadie puede saber qué problemas aún no han sido detecta-

dos. En cada etapa se ha ido progresando mediante un diseño bien estructurado de bases de datos, pero nadie puede garantizar que no se puedan encontrar más anomalías. En esta sección estudiaremos una forma normal, la cual garantiza que no habrá anomalías de ningún tipo. Cuando planteamos las relaciones de esta manera, sabemos que no pueden ocurrir incluso las anomalías más raras asociadas con la quinta forma normal.

En 1981 Fagin publicó un importante artículo en el que definió la forma normal dominio-llave (DK/NF).[5] Mostró que una relación en DK/NF no tiene anomalías de modificación y, además, que una relación sin anomalías de modificación debe estar en DK/NF. Este hallazgo establece un límite en la definición de las formas normales, y por lo tanto no es necesaria una forma normal más alta, al menos con el fin de eliminar anomalías de modificación.

Igualmente importante es que la DK/NF sólo implica los conceptos de llave y dominio, los cuales son fundamentales y clave para los profesionales de bases de datos. Se apoyan en los productos DBMS (o cuando menos deberían hacerlo). En cierto sentido, el trabajo de Fagin formaliza y justifica lo que muchos profesionales creían intuitivamente, pero no eran capaces de expresar en forma precisa.

Definición

El concepto DK/NF es muy simple: una relación está en DK/NF si cada restricción en la relación es una consecuencia lógica de la definición de llaves y dominios. Ponga mucha atención en los términos importantes: restricción, llave y dominio.

La **restricción** pretende ser muy extensa. Fagin la define como cualquier regla que gobierna valores estáticos de atributos, la cual es lo suficientemente precisa como para que podamos verificar si es o no verdadera. Las reglas de edición, las restricciones de intrarrelación e interrelación, las dependencias funcionales y la dependencia multivaluada son ejemplos de estas restricciones. Fagin excluye expresamente las restricciones correspondientes a cambios en los valores de los datos, o las restricciones que dependan del tiempo. Por ejemplo, la regla "el salario de los vendedores en el periodo actual no puede ser menor al que recibían en la etapa anterior" está excluida de la definición de restricción de Fagin. Exceptuando las restricciones dependientes del tiempo, la definición de Fagin es extensa e incluyente.

Una **llave** es un identificador único de un tuple, como ya lo hemos establecido. El tercer término importante en la definición de DK/NF es el **dominio.** En el capítulo 4 establecimos que el dominio es una descripción de los valores permitidos para un atributo. Consta de dos partes: una descripción física y una semántica o lógica. La descripción física constituye un conjunto de valores que el atributo puede tener, y la lógica se refiere al significado del atributo. La prueba de Fagin se aplica a ambas partes.

De manera informal, una relación está en DK/NF si al cumplir las restricciones de la llave y del dominio da como resultado que todas las restricciones se cumplan. Además, debido a que las relaciones en DK/NF no pueden tener anomalías de modificación, el DBMS puede prohibirlas imponiendo las restricciones de dominio y de llave.

Por desgracia, no se conoce algún algoritmo para convertir una relación a DK/NF; incluso no se sabe cuáles relaciones se pueden convertir a DK/NF. Encontrar o diseñar las relaciones DK/NF es más un arte que una ciencia.

A pesar de lo anterior, en el mundo práctico del diseño de bases de datos, la DK/NF es extremadamente útil para el diseño. Si podemos definir relaciones en una forma en que las restricciones sobre ellas sean consecuencias lógicas de dominio y llaves, entonces no habrá anomalías de modificación. En muchos diseños este objetivo se puede cumplir. Cuando no es así, las restricciones se deben construir dentro de la lógica de los programas de aplicación que procesan la base de datos. Veremos más acerca de esto en el presente capítulo y en el capítulo 10.

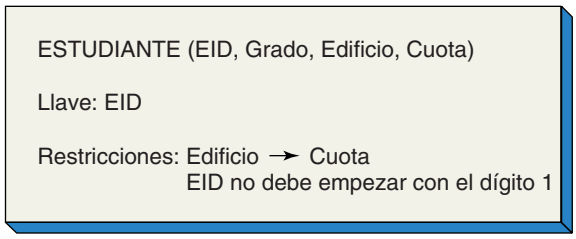
Los siguientes ejemplos ilustran la DK/NF.

---

[5] *Ibid.*

➤ FIGURA 5-12

Ejemplo 1 de DK/NF



EJEMPLO 1 DE LA FORMA NORMAL DOMINIO-LLAVE

Considere la relación ESTUDIANTE en la figura 5-12, la cual contiene atributos EID, Grado, Edificio y Cuota. Edificio es el lugar donde los estudiantes viven y Cuota es la cantidad que pagan por vivir en ese edificio.

EID determina funcionalmente a los otros tres atributos; por lo tanto, EID es una llave. Suponga que también sabemos, con base en la definición de requerimientos, que Edificio → Cuota y que los EID no deben empezar con 1. Si podemos expresar esas restricciones como consecuencias lógicas de dominio y definiciones de llaves, podemos estar seguros que, de acuerdo con el teorema de Fagin, no habrá anomalías de modificación. En el presente ejemplo, esto será fácil.

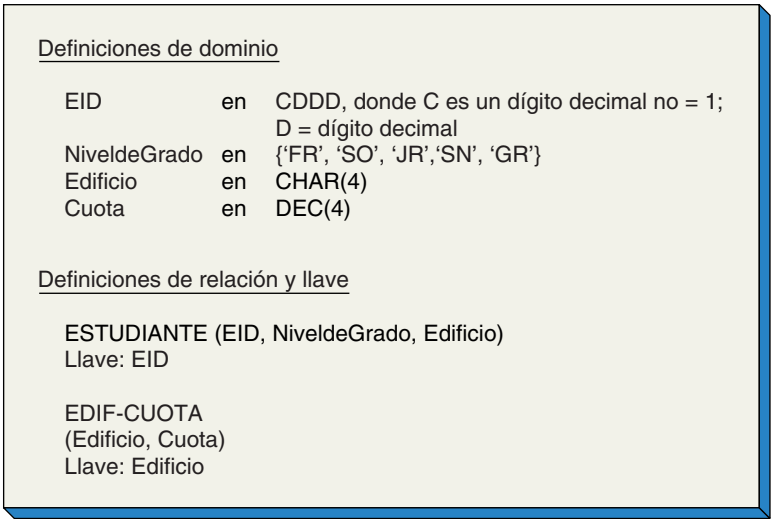
Para imponer la restricción de que los números de estudiantes no empiecen con 1, sólo definimos el dominio para los números de estudiantes que incorporen esta restricción (figura 5-13). Imponer la restricción de dominio garantiza que se cumplirá.

A continuación necesitamos hacer de la dependencia funcional Edificio → Cuota una consecuencia lógica de las llaves. Si Edificio fuera un atributo de llave, Edificio → Cuota podría ser una consecuencia lógica de una llave. Por lo tanto, la cuestión es cómo convertir Edificio en una llave. No puede haber una llave en ESTUDIANTE porque más de uno viven en el mismo edificio, pero puede ser una llave de su propia relación. Así, definimos la relación EDIF-CUOTA con Edificio y Cuota como sus atributos. Edificio es la llave de esta relación. Una vez definida esta nueva relación, podemos eliminar Cuota de ESTUDIANTE. El dominio final y las definiciones de relación para este ejemplo aparecen en la figura 5-13.

Este es el mismo resultado que obtenemos cuando convertimos una relación de una 2NF a una 3NF para eliminar las dependencias transitivas. Sin embargo, en este caso el proceso fue simple y el resultado más sólido. Resultó más sencillo porque no necesitamos saber que eliminábamos una dependencia transitiva. Simplemente tuvimos

➤ FIGURA 5-13

Definición de dominio-llave del ejemplo 1



➤ FIGURA 5-14

Ejemplo 2 de DK/NF

PROFESOR (FID, Fnombre, Clase, EID, Snombre)
Llave: (FID, Clase, EID)
Restricciones: FID → Fnombre
Fnombre → FID
FID →→ Clase | EID
Fnombre →→ Clase | EID
EID → FID
EID → Fnombre
EID → Enombre
FID debe empezar con 1; EID no debe empezar con 1

que encontrar maneras creativas de hacer que todas las restricciones fueran consecuencias lógicas de las definiciones de dominio y llave. El resultado fue más sólido porque cuando convertimos la relación a 3NF sólo sabíamos que existían menos anomalías que cuando estaba en 2NF. Para convertir la relación a DK/NF sabemos que ésta no tiene anomalías de modificación.

Ejemplo 2 de la forma normal de dominio-llave

El siguiente ejemplo, más complicado, involucra la relación de la figura 5-14. La relación PROFESOR contiene datos acerca de los profesores, la materia que enseñan y los alumnos que asesoran. FID y Fnombre únicamente identifican a un profesor, y EID sólo a un estudiante, pero Enombre no necesariamente identifica a EID. Los profesores pueden impartir varias materias y asesorar a varios estudiantes, pero un estudiante es asesorado sólo por un profesor. Los FID empiezan con un 1, pero los EID no deben empezar con un 1.

Estas declaraciones se pueden expresar con mayor precisión mediante la dependencia funcional y los valores múltiples que se muestran en la figura 5-14. FID y Fnombre se determinan funcionalmente entre sí (en esencia, son equivalentes). FID y Fnombre multideterminan a Materia y a EID; EID determina funcionalmente a FID y a Fnombre, y EID, a Enombre.

En ejemplos más complicados como este es útil considerar DK/NF desde un punto de vista más intuitivo. Recuerde que la esencia de la normalización es que cada relación puede tener sólo un tema. Si lo consideramos desde esta perspectiva, hay tres temas en PROFESOR. Uno es la correspondencia entre FID y Fnombre; otro se refiere a las clases que el profesor imparte, y el tercero al número de identificación, nombre y asesor de un estudiante determinado.

La figura 5-15 muestra tres relaciones que reflejan esos temas. La relación FACULTAD representa la equivalencia de FID y Fnombre. FID es la llave y Fnombre es una llave alternativa cuyo significado es que ambos atributos son únicos en la relación. Debido a que ambos son llaves, la dependencia funcional FID → Fnombre y Fnombre → FID son consecuencias lógicas de llaves.

La relación PREPARACIÓN contiene la correspondencia de facultad y clases; muestra las materias que un profesor está preparado para impartir. La llave es la combinación (Fnombre, Materia). Se requieren ambos atributos en la llave porque un profesor puede impartir varias materias y una materia puede ser impartida por varios profesores. Por último, ESTUDIANTE representa el nombre del estudiante y del asesor para un EID en particular. Observe que cada una de estas relaciones tiene un tema individual. Éstas expresan todas las restricciones de la figura 5-14 como una consecuencia lógica de las definiciones de dominio y llaves. Estas relaciones están, por lo tanto, en DK/NF.

Observe que separando el tema PREPARACIÓN del tema ESTUDIANTE se han eliminado las dependencias multivaluadas. Cuando examinamos la cuarta forma normal, encontramos que con el fin de eliminar dependencias multivaluadas tuvimos que

➤ FIGURA 5-15

Definición dominio-llave del ejemplo 2

Definiciones de dominio

FID	en	CDDD, C = 1; D = dígito decimal
Enombre	en	CHAR(30)
Clase	en	CHAR(10)
EID	en	CDDD, C es dígito decimal, no = 1; D = dígito decimal
Enombre	en	CHAR(30)

Definiciones de relación y llave

FACULTAD (FID, Fnombre)
Llave (primaria): FID
Llave (candidata): Fnombre

PREPARACIÓN (Fnombre, Clase)
Llave: Fnombre, Clase

ESTUDIANTE (EID, Enombre, Fnombre)
Llave: EID

separar los atributos multivaluados en diferentes relaciones. Nuestro planteamiento aquí es separar una relación con varios temas en varias relaciones que tengan cada una un tema. Al hacer esto eliminamos la dependencia multivaluada. De hecho, llegamos a la misma solución usando ambos planteamientos.

Ejemplo 3 de la forma normal dominio-llave

El siguiente ejemplo se refiere a una situación que no fue tratada por ninguna de las otras formas normales, pero que en la práctica ocurre con frecuencia. Esta relación tiene una restricción entre los valores de los datos dentro de un tuple, la cual no es una dependencia funcional ni una dependencia multivaluada.

Considere las restricciones en la relación ESTUD-ASESOR en la figura 5-16. Contiene información acerca de un estudiante y de su asesor, o asesora. EID determina Enombre, FID, Fnombre y EstadodeGradodeFacultad y por lo tanto es la llave. FID y Fnombre identifican a un miembro de una sola facultad y son equivalentes entre sí, como en el ejemplo 2. Tanto FID como Fnombre determinan el EstadodeGradodeFacultad. Por úl-

➤ FIGURA 5-16

Ejemplo 3 de DK/NF

ESTUD-ASESOR (EID, Enombre, FID, Fnombre, EstadodeGradodeFacultad)

Llave: EID

Restricciones:
FID → Fnombre
Fnombre → FID
FID y Fnombre → EstadodeGradodeFacultad
Sólo los miembros de la facultad para graduados pueden asesorar a los estudiantes
FID empieza con 1
EID no debe empezar con 1
EID de estudiantes graduados empieza con 9
EstadodeGradodeFacultad = { 0 para no graduados de la facultad; 1 para graduados de la facultad

timo, el nuevo tipo de restricción es que sólo a los miembros de la facultad de graduados se les permite asesorar a los estudiantes.

Las restricciones de dominio son que EID no debe comenzar con 1, sino con 9 para estudiantes de la facultad para graduados, FID debe comenzar con 1, y EstadodeGradodeFacultad es 0 para la facultad de no graduados y 1 para la facultad de graduados. Con estas definiciones de dominio, la restricción de que los estudiantes deben ser asesorados por los miembros de la facultad para graduados se puede expresar como una restricción en los valores de un renglón. Específicamente, si el EID comienza con 9, el valor de EstadodeGradodeFacultad debe ser 1.

Para poner esta relación en DK/NF procederemos como en el ejemplo 2. ¿Cuáles son los temas básicos de esta relación? Hay uno concerniente al personal de la facultad, que relaciona FID, Fnombre, y el EstadodeGradodeFacultad. Puesto que FID y Fnombre determinan el EstadodeGradodeFacultad, ambos atributos pueden ser la llave y esta relación está en DK/NF (véase la figura 5-17).

Ahora considere los datos con respecto a estudiantes y asesores. Aunque en principio puede parecer que sólo hay un tema, el de asesoría, la restricción de que sólo los miembros de la facultad para graduados pueden asesorar a estudiantes implica otra cosa. En realidad, hay dos temas: asesoramiento a los graduados y asesoría a los no graduados. Así, la figura 5-17 contiene una relación G-ASE (Graduado-Asesor) para estudiantes graduados, y una relación NG-ASE para no graduados. Veamos las definiciones de dominio: EGID comienza con 9; Gfnombre es el Fnombre de un tuple de FACULTAD con EstadodeGradodeFacultad igual a 1; y ENGID no debe empezar con 1 o 9. Todas las restricciones descritas en la figura 5-16 están implícitas en la llave y las definiciones de dominio en la figura 5-17. Estas relaciones están en DK/NF y no tienen anomalías de modificación.

Para resumir, en la figura 5-18 se enumeran las formas normales y se definen las características de cada una.

➤ FIGURA 5-17

Definición de dominio/Clave del ejemplo 3

Definición de dominio

FID	en	CDDD, donde C = 1; D = dígito decimal
Fnombre	en	CHAR (30)
EstadodeGradodeFacultad	en	[0, 1]
EGID	en	CDDD, donde C = 9; D = dígito decimal; estudiante graduado
ENGID	en	CDDD, DONDE C ≠ 1 y C ≠ 9; D = dígito decimal; estudiante no graduado
Enombre	en	CHAR (30)

Definiciones adicionales de dominio

Gfnombre	en	{Fnombre de FACULTAD, donde EstadodeGradodeFacultad = 1}

Definiciones de relación y llave

FACULTAD (**FID**, Fnombre, EstadodeGradodeFacultad)
Llave: **FID** o Fnombre

G-ASESOR (**EGID**, Enombre, Gfnombre)
Llave: **EGID**

NG-ASESOR (**ENGID**, Enombre, Fnombre)
Llave: **ENGID**

➤ FIGURA 5-18

Resumen de Formas Normales

Forma	*Características que la definen*
1NF	Cualquier relación.
2NF	Todos los atributos que no son llaves dependen por completo de cada llave.
3NF	No hay dependencias transitivas.
BCNF	Cada determinante es una llave candidata.
4NF	No hay dependencias multivaluadas.
5NF	No se describió en este análisis.
DK/NF	Todas las restricciones son consecuencias lógicas de los dominios y de las llaves.

➤ SÍNTESIS DE RELACIONES

En la sección anterior propusimos un diseño relacional a partir de un perspectiva analítica. Las preguntas que hicimos con respecto a una relación determinada fueron: ¿está en la forma correcta? ¿Tiene anomalías de modificación? Abordaremos el diseño relacional desde una perspectiva diferente: la sintética. A partir de dicha perspectiva nos preguntamos: "Considerando un conjunto de atributos con ciertas dependencias funcionales, ¿qué relaciones debemos formar?"

Primero, observemos que dos atributos, digamos A y B, pueden estar relacionados de tres formas:

1. Se determinan entre sí:
A → B y B → A
Por lo tanto, A y B tienen una relación de atributos uno a uno.

2. Uno determina al otro.
A → B, pero B no → A
Así, A y B tienen una relación muchos a uno.

3. No están relacionados funcionalmente.
A no → B y B no → A
Entonces, A y B tienen una relación de atributos muchos a muchos.

RELACIONES DE ATRIBUTOS UNO A UNO

Si A determina a B, y B determina a A, los valores de los atributos tienen una relación uno a uno. Esto debe ser porque si A determina a B, la relación entre A y B es de muchos a uno. Sin embargo, también es cierto que si B determina a A la relación entre B y A debe ser de muchos a uno. Para que ambos enunciados sean verdad al mismo tiempo, la relación entre A y B debe ser realmente uno a uno (lo cual es un caso especial de muchos a uno), y la relación entre B y A es también realmente de uno a uno. Por lo tanto, la relación es uno a uno.

Este caso se ilustra mediante FID y Fnombre en los ejemplos 2 y 3 de la sección anterior, referente a la forma normal dominio-llave. Cada uno de estos atributos únicamente identifica a una persona de la facultad. Consecuentemente, un valor de FID corresponde exactamente a un valor de Fnombre y viceversa.

Se pueden esbozar tres enunciados equivalentes a partir del ejemplo de FID y Fnombre:

- Si dos atributos se determinan funcionalmente entre sí, la relación de sus valores de datos es uno a uno
- Si dos atributos identifican en forma única a la misma cosa (entidad u objeto), la relación de sus valores es uno a uno
- Si dos atributos tienen una relación uno a uno, se determinan funcionalmente entre sí

Cuando se crea una base de datos con atributos que tienen una relación uno a uno, ambos atributos deben aparecer juntos cuando menos en una relación. Otros atributos que son determinados funcionalmente por éstos (un atributo que es determinado funcionalmente por uno de ellos también lo es por los otros) pueden residir en la misma relación.

Considere FACULTAD (FID, Fnombre, EstadodeGradodeFacultad) en el ejemplo 3 de la sección anterior. FID y Fnombre se determinan entre sí. EstadodeGradodeFacultad también puede aparecer en esta relación porque está determinado por FID y Fnombre. Los atributos que no están determinados funcionalmente por éstos no pueden aparecer en una relación con ellos. Considere las relaciones FACULTAD y PREPARACIÓN en el ejemplo 2; tanto FID como Fnombre aparecen en FACULTAD, pero Clase (a partir de PREPARACIÓN) puede no ocurrir. Clase puede tener múltiples valores para un miembro de la facultad, de tal forma que Clase no depende de FID o Fnombre. Si agregamos Clase a la relación FACULTAD, la llave de FACULTAD necesitará ser cualquiera (FID, Clase) o (Fnombre, Clase). Sin embargo, en este caso FACULTAD no estaría en DK/NF porque las dependencias entre FID y Fnombre no estarían lógicamente involucradas en ninguna de las llaves posibles.

Estos enunciados se resumen en la primera columna de la figura 5-19, y las reglas de definición del registro se enumeran en la figura 5-20. Si A y B tienen una relación 1:1 pueden residir en la misma relación, digamos R. A determina a B y B determina a A. La llave de la relación puede ser A o B. Se puede agregar un nuevo atributo, C, a R, si A o B determinan funcionalmente a C.

Los atributos que tienen una relación uno a uno deben existir juntos con el fin de establecer su equivalencia (por ejemplo, FID de 198 se refiere al profesor Heart). Sin embargo, por lo general no es conveniente que aparezcan juntos en más de una relación, ya que esto provoca una duplicación de datos innecesaria. Con frecuencia uno o ambos de los atributos aparecen en otras relaciones. En el ejemplo 2, Fnombre ocurre tanto en PREPARACIÓN como en ESTUDIANTE. Aunque será posible colocar Fnombre en PREPARACIÓN y FID en ESTUDIANTE, por lo general es una práctica errónea, porque cuando los atributos se parean en esta forma se debe seleccionar a uno para que represente el par en todas las otras relaciones. En el ejemplo 2 se seleccionó Fnombre.

Relaciones de atributos muchos a uno

Si el atributo A determina a B, pero B no determina a A, la relación entre los valores de los datos es de muchos a uno. En la relación del asesor en el ejemplo 2, EID determina a FID. Varios estudiantes (EID) son asesorados por un miembro de la facultad (FID), pero cada estudiante es asesorado sólo por un miembro de la facultad. Ésta, entonces, es una relación de muchos a uno.

Para una relación que esté en DK/NF todas las restricciones deben estar incluidas en llaves, y por lo tanto cada determinante debe ser una llave. Si A, B y C están en la misma relación, y si A determina a B, entonces A debe ser la llave (lo cual significa que

➤ FIGURA 5-19

Resumen de los tres tipos de relaciones de atributos

	Tipo de relación del atributo		
	Uno a uno	Muchos a uno	Muchos a muchos
Definición de la relación*	R(A,B)	S(C,D)	T(E,F)
Dependencias	A → B B → A	C → D D ↛ C	E ↛ F F ↛ E
Llave	A o B	C	(E,F)
Regla para agregar otro atributo	A o B → C	C → E	(E,F) → G

* Las letras usadas en estas definiciones de relaciones son iguales a las que se usan en la figura 5-20.

➤ FIGURA 5-20

Resumen de las reglas para construir relaciones

Con referencia a las relaciones de atributo uno a uno

- Los atributos que tienen una relación uno a uno deben aparecer juntos cuando menos en una relación. Llame a la relación *R* y a los atributos *A* y *B*
- *A* o *B* deben ser la llave de *R*
- Se puede agregar un atributo a *R* si está determinado funcionalmente por *A* o *B*
- Un atributo que no está determinado funcionalmente por *A* o *B* no puede agregarse a *R*
- *A* y *B* deben aparecer juntos en *R*, pero no deberán aparecer juntos en otras relaciones
- *A* o *B* deben usarse consistentemente para representar el par en las relaciones diferentes a *R*

Con referencia a las relaciones de atributos muchos a uno

- Los atributos que tienen una relación muchos a uno pueden existir juntos en una relación. Suponga que *C* determina a *D* en la relación *S*
- *C* debe ser la llave de *S*
- Puede agregarse un atributo a *S* si está determinado por *C*
- Un atributo que no está determinado por *C* no puede agregarse a *S*

Con referencia a las relaciones de atributos muchos a muchos

- Los atributos que tienen una relación muchos a muchos pueden existir juntos en una relación. Suponga que dos de estos atributos, *E* y *F*, residen juntos en la relación *T*
- La llave de T debe ser (*E*, *F*)
- Se puede agregar un atributo a *T* si está determinado por la combinación (*E*, *F*)
- No se puede agregar un atributo a *T* si no está determinado por la combinación (*E*, *F*)
- Si se agrega un nuevo atributo, *G*, se expande la llave a (*E, F, G*), entonces el tema de la relación ha sido cambiado. *G* no pertenece a *T* o el nombre de *T* se debe cambiar para reflejar el nuevo tema

también determina a C). Si por el contrario (A, B), determina a C, entonces (A, B) debe ser la llave. En este último caso, no se permite otra dependencia funcional, como por ejemplo que A determine a B.

Usted puede aplicar estos enunciados al diseño de bases de datos en la siguiente forma: cuando construya una relación, si A determina a B los otros atributos individuales que puede agregar a la relación también deben estar determinados por A. Por ejemplo, suponga que tiene que poner EID y Edificio juntos en una relación llamada ESTUDIANTE. Puede agregar cualquier otro atributo determinado por EID, por ejemplo Enombre, a esta relación. Pero si el atributo Cuota está determinado por Edificio, usted no puede añadirlo a esta relación. Sólo podrá agregar Cuota si EID → Cuota.

Estos enunciados están resumidos en la columna central de la figura 5-19. Si C y D tienen una relación N:1, pueden residir juntos en una relación, digamos S. C determinará a D, pero D no determinará a C. La llave de S será C. Otro atributo, E, se podrá agregar a S sólo si C determina a E.

Relaciones de atributos muchos a muchos

Si A no determina a B y B no determina a A, la relación entre sus valores de datos es muchos a muchos. En el ejemplo 2, Fnombre y Clase tienen una relación muchos a muchos. Un profesor imparte muchas clases y una clase la imparten muchos profesores. En una relación muchos a muchos ambos atributos deben ser una llave de la relación. Por ejemplo, la llave de PREPARACIÓN en el ejemplo 2 es la combinación (Fnombre, Clase).

Cuando se construyen relaciones que tienen múltiples atributos como llaves, puede sumar nuevos atributos que sean funcionalmente dependientes de todos los atributos de la llave. CantidaddeVecesImpartida es funcionalmente dependiente de ambos (Fnombre, Clase) y se puede agregar a la relación. Sin embargo, OficinadeFacultad no puede añadirse porque sólo sería dependiente de Fnombre y no de Clase. Si se necesita almacenar OficinadeFacultad en la base de datos, debe adjuntarse en la relación concerniente a facultad, no en la que se refiere a preparaciones.

Estos enunciados se resumen en la columna derecha de la figura 5-19. Si E y F tienen una relación M:N, E no determina a F, y F no determina a E. Tanto E como F se pueden poner en la relación T y, si se hace así, la llave de T podrá ser la llave compuesta (E, F). Se puede agregar un nuevo atributo, G, a T si está determinado por (E, F). No se puede agregar a T si sólo está determinado por E o F.

Considere un ejemplo similar. Suponga que agregamos NúmerodeSalón a PREPARACIÓN. ¿NúmerodeSalón está determinado funcionalmente por la llave de PREPARACIÓN (Fnombre, Clase)? Lo más probable es que no, porque un profesor puede impartir determinada clase en varios salones diferentes.

La composición (Fnombre, Clase) y NúmerodeSalón tiene una relación M:N. Puesto que es así, las reglas en la figura 5-19 se pueden aplicar, pero con E representando (Fnombre, Clase) y F representando NúmerodeSalón. Ahora podemos integrar una nueva relación, T, con los atributos Fnombre, Clase y NúmerodeSalón. La llave es entonces (Fnombre, Clase, NúmerodeSalón). En este caso, hemos creado una nueva relación con un nuevo tema. Considere la relación T, que contiene nombres de las facultades, clases y cantidad de salones. Por lo tanto, el tema de esta relación no es PREPARACIÓN, sino QUIÉN-QUÉ-DÓNDE-HA-IMPARTIDO.

Cambiar el tema puede ser o no apropiado. Si NúmerodeSalón es importante, no se necesita cambiar el tema. En este caso, PREPARACIÓN es la relación equivocada y QUIÉN-QUÉ-DÓNDE-HA-IMPARTIDO es un tema más adecuado.

Por otra parte, dependiendo de los requerimientos del usuario, PREPARACIÓN puede ser completamente adecuado así como está. Si es el caso, entonces NúmerodeSalón pertenece a la base de datos y podrá ser localizado en una relación diferente —tal vez NÚMERO-SECCIÓN, SECCIÓN-CLASE o alguna combinación similar.

➤ DEPENDENCIAS MULTIVALUADAS, ITERACIÓN 2

El análisis de las relaciones de valores de atributos muchos a muchos puede hacer que el concepto de dependencias multivaluadas sea más fácil de comprender. El problema con la relación ESTUDIANTE (EID, Especialidad, Actividad) en la figura 5-9 es que tiene *dos* relaciones diferentes muchos a muchos —una entre EID y Especialidad y la otra entre EID y Actividad—. Queda claro que varias especialidades de estudiantes no tienen nada que ver con sus diferentes actividades. Sin embargo, poner ambas relaciones muchos a muchos en una sola hace que parezca como si existiera alguna asociación.

Especialidad y Actividad son independientes y no habría problema si un estudiante tuviera sólo una de cada una. EID determinaría funcionalmente Especialidad y Actividad, y la relación estaría en DK/NF. En este caso, tanto las relaciones entre Especialidad y EID como entre Actividad y EID serían muchos a una.

Otra forma de percibir la dificultad es examinar la llave (EID, Especialidad, Actividad). Puesto que ESTUDIANTE tiene relaciones muchos a muchos, todos los atributos tienen que estar en la llave. Ahora, ¿qué tema representa esta llave? Podemos decir que la combinación de los estudios y las actividades de los estudiantes. Pero esto no es un hecho singular, es plural. Un renglón de esta correspondencia sólo describe parte de la combinación, y para obtener la imagen completa necesitamos todos los renglones de un estudiante específico. *En general, un renglón puede tener todos los datos del tema de la relación*. Por ejemplo, un renglón de Cliente podría tener todos los datos que queremos acerca de un cliente en particular.

Considere PREPARACIÓN en el ejemplo 2 en la sección acerca de la forma normal dominio-llave. La llave es (Fnombre, Clase). Esto quiere decir que un profesor en particular está preparado para impartir determinada clase. Sólo necesitamos un renglón para obtener toda la información (la relación puede incluir CantidaddeVecesImpartida, CalificaciónPromediodelCurso, etc.) que tenemos con respecto a la combinación de ese profesor y esa clase. Consultar más renglones no proporcionará información adicional.

Como sabe, la solución al problema de restricción a dependencias multivaluadas es dividir la relación en dos, cada una con un solo tema. ESTUD-ESPECIALIDAD muestra la combinación de un estudiante y una especialidad. Todo lo que sabemos acerca de la combinación está en un solo renglón y no obtendremos información adicional si examinamos más renglones.

➤ OPTIMIZACIÓN

En este capítulo estudiamos los conceptos de normalización y mostramos cómo crear tablas que estén en DK/NF. El procedimiento que empleamos por lo general es el adecuado, pero a veces el resultado de la normalización no justifica el costo. Por último, veremos dos maneras en las que lo anterior puede suceder.

DESNORMALIZACIÓN

Como mencionamos, las relaciones normalizadas evitan anomalías de modificación, y en este sentido son preferibles a las relaciones sin normalizar. Sin embargo, si la valoramos bajo otros criterios, la normalización a veces no vale la pena.

Considere esta relación:

CLIENTE (NúmerodeCliente, NombredelCliente, Ciudad, Estado CódigoPostal)

donde NúmerodeCliente es la llave.

Esta relación no está en DK/NF porque contiene la dependencia funcional CódigoPostal → (Ciudad, Estado), lo cual no está implícito en la llave NúmerodeCliente. Por lo tanto, hay una restricción no implícita por la definición de llaves.

Esta relación se puede transformar en las siguientes relaciones DK/NF:

CLIENTE (NúmerodeCliente, NombredelCliente, CódigoPostal)

donde la llave es NúmerodeCliente

CÓDIGOS (CódigoPostal, Ciudad, Estado)

donde la llave es CódigoPostal.

Estas dos tablas están en la forma normal dominio-llave, pero es probable que no representen el mejor diseño. La tabla sin normalizar probablemente es mejor porque sería más fácil de procesar y las desventajas de duplicar los datos de Ciudad y Estado no serían muy importantes.

Para otro ejemplo de desnormalización considere la relación

COLEGIO (NombredelColegio, Decano, AsistentedeDecano)

y suponga que un colegio tiene un decano y de uno a tres asistentes de decano. En este caso, la llave de la tabla es (NombredelColegio, AsistentedeDecano). Esta tabla no está en la forma normal dominio-llave porque la restricción NombredelColegio → Decano no es una consecuencia lógica de las llaves de la tabla.

COLEGIO se puede normalizar en las relaciones:

DECANO (NombredelColegio, Decano)

y

ASISTENTE-DECANO (NombredelColegio, AsistentedeDecano)

Pero ahora cada vez que una aplicación de la base de datos necesite obtener datos del colegio, deberá leer cuando menos dos renglones de datos, o posiblemente cuatro. Una alternativa para este diseño es colocar los tres AsistentedeDecano en la tabla de COLEGIO, cada uno en un atributo por separado. La tabla sería entonces:

COLEGIO1 (NombredelColegio, Decano, AsistentedeDecano1, AsistentedeDecano2, AsistentedeDecano3)

COLEGIO1 está en DK/NF porque todos sus atributos son funcionalmente dependientes de la llave NombredelColegio. Pero algo se ha perdido. Para ver qué es, suponga que quiere determinar los nombres de los COLEGIOs que tienen una asistente de decano llamada 'Mary Abernathy'. Para hacer esto, tendría que buscar este valor en cada una de las tres columnas de AsistentedeDecano. En su consulta aparecería algo similar a lo siguiente:[6]

```
SELECT      NombredelColegio
FROM        COLEGIO1
WHERE       AsistentedeDecano1 = 'Mary Abernathy' o
            AsistentedeDecano2 = 'Mary Abernathy' o
            AsistentedeDecano3 = 'Mary Abernathy'
```

Usando el diseño normalizado con ASISTENTE-DECANO, sólo se necesitaría establecer:

```
SELECT      NombredelColegio
FROM        ASISTENTE-DECANO
WHERE       AsistentedeDecano = 'Mary Abernathy'
```

En este ejemplo hay tres soluciones posibles, cada una con ventajas y desventajas. Elegir entre ellas es un arte, no hay una regla estricta que establezca cómo seleccionarlas. La mejor opción depende de las características de procesamiento de las aplicaciones que use la base de datos.

En resumen, a veces las relaciones no son normalizadas a propósito, o son normalizadas y después se desnormalizan. Con frecuencia se hace esto para mejorar el desempeño. Siempre que los datos de dos tablas individuales se deben combinar, los DBMS tienen que ejecutar trabajo extra. En la mayoría de los casos se requieren cuando menos dos lecturas en lugar de una.

Redundancia controlada

Una de las ventajas de las relaciones normalizadas es que la duplicación de datos se minimiza (sólo los valores de la llave aparecen en más de una relación). Sin embargo, por razones de desempeño a veces es apropiado duplicar intencionalmente los datos. Considere, por ejemplo, una aplicación de pedido que accese a la tabla ARTÍCULO, la cual tiene las siguientes columnas:

NúmerodelaPieza
NombredelaPieza
ColordelaPieza
DescripcióndelaPieza
FotografíadelaPieza
CantidaddeManodeObra
CantidaddelPedido
PrecioEstándar

[6] Estos enunciados son ejemplos de SQL, un lenguaje relacional que analizaremos con detalle en el capítulo 9. Por el momento, sólo pensaremos en ellos de manera intuitiva; en el presente capítulo aprenderá acerca de su formato.

CostoEstándar
NombredelComprador

Suponga que NúmerodelaPieza es la llave y que la tabla está en DK/NF. También suponga que el atributo DescripcióndelaPieza es un campo memo potencialmente largo, y FotografíadelaPieza es una columna binaria de cuando menos 256K bytes de longitud.

La aplicación de procesamiento de pedidos necesitará accesar a esta tabla para obtener NombredelaPieza, ColordelaPieza, PrecioEstándar y CantidaddeManodeObra. Suponga que no se necesita DescripcióndelaPieza o FotografíadelaPieza. Dependiendo de las características del DBMS en uso, es posible que la presencia de estas dos grandes columnas haga que el proceso sea muy lento. Si es el caso, los diseñadores de la base de datos pueden decidir duplicar algunos de los datos en una segunda tabla que contenga sólo los necesarios para el procesamiento del pedido. Ellos pueden definir una tabla como PEDIDOARTÍCULO (NúmerodelaPieza, NombredelaPieza, ColordelaPieza, PrecioEstándar, CantidaddeManodeObra) que se use sólo para la aplicación del procesamiento del pedido.

En este caso, los diseñadores están creando un potencial para prevenir problemas serios de integridad de datos. Necesitarán desarrollar controles programáticos y manuales para asegurar que tales problemas no ocurran. Sólo crearán ese diseño si consideran que el incremento del desempeño vale la pena el costo de los controles y el riesgo de los problemas de integridad.

Otra razón para controlar la redundancia es crear tablas que se usan sólo con el propósito de elaborar reportes y apoyar la toma de decisiones. Trataremos estos temas en el capítulo 17.

➤ RESUMEN

El modelo relacional es importante por dos razones: se puede usar para expresar diseños de bases de datos independientes de un DBMS, y es la base para una categoría importante de productos DBMS. La normalización se puede usar como lineamiento para verificar la pertinencia y exactitud de las relaciones.

Una relación es una tabla en dos dimensiones que tiene entradas de un solo valor. Todas las entradas en una columna determinada son de la misma clase; las columnas tienen un nombre único, y el orden de éstas no es importante. Las columnas también se llaman atributos. No existen dos renglones idénticos en la tabla y el orden de los renglones no es importante. Los renglones también se llaman tuples. Los términos *tabla, archivo* y *relación* son sinónimos; *columna, campo* y *atributo* son sinónimos, y *renglón*, *registro* y *tuple* también son sinónimos.

Una dependencia funcional es una relación entre atributos. Es funcionalmente dependiente de X si su valor determina el de Y. Un determinante es un grupo de uno o más atributos en el lado izquierdo de una dependencia funcional. Por ejemplo, si X determina a Y, entonces X es el determinante. Una llave es un grupo de uno o más atributos que únicamente identifican un tuple. Cada relación tiene cuando menos una llave; debido a que cada renglón es único; en el caso más extremo la llave es el conjunto de todos los atributos en la relación. Aunque una llave es siempre única, el determinante en una dependencia funcional no necesita serlo. Si los atributos son o no llaves, y si éstas son o no atributos no lo determina un conjunto abstracto de reglas, sino la semántica del usuario.

Cuando se actualizan, algunas relaciones experimentan consecuencias no deseables llamadas anomalías de modificación. Ocurre una anomalía de eliminación cuando se suprime un renglón y se pierde información de dos o más entidades. Ocurre una anomalía de inserción cuando la estructura relacional obliga a la adición de hechos sobre dos entidades al mismo tiempo. Las anomalías pueden ser eliminadas dividiendo la relación en dos o más relaciones.

Existen muchos tipos de anomalías de modificación. Las relaciones se pueden clasificar por los tipos de anomalías que eliminan. Estas clasificaciones se denominan formas normales.

Por definición, cada relación está en la primera forma normal. Está en la segunda forma normal si todos los atributos que no son llaves dependen de la llave completa. Una relación está en la tercera forma normal cuando se encuentra en la segunda forma normal y no tiene dependencias transitivas. Una relación está en la forma normal Boyce-Codd si cada determinante es una llave candidata. Cuando está en la cuarta forma normal es cuando se encuentra en la forma normal Boyce-Codd y no tiene dependencias multivaluadas. La definición de la quinta forma normal es intuitivamente oscura y por lo tanto no la establecemos.

Una relación está en la forma normal dominio-llave si cada restricción en la relación es una consecuencia lógica de la definición de dominios y llaves. Una restricción es cualquier restricción de los valores estáticos de los atributos cuya veracidad puede ser evaluada. Como lo establecimos, los dominios tienen una parte física y una semántica; sin embargo, en el contexto de DK/NF los dominios sólo se refieren a la descripción física.

Una manera informal de expresar la DK/NF es decir que cada relación debe tener un tema individual. Por ejemplo, esto puede referirse a PROFESORes o ESTUDIANTEs, pero no a PROFESORes y ESTUDIANTEs al mismo tiempo.

La normalización es un proceso de análisis de relaciones. Es posible construir relaciones mediante un proceso de síntesis considerando las relaciones entre los atributos. Si dos atributos se determinan funcionalmente entre sí, tienen una relación uno a uno. Si uno determina funcionalmente al otro, pero no a la inversa, los atributos tienen una relación uno a muchos. Si ninguno de los atributos determina a otro, entonces tienen una relación muchos a muchos. Estos hechos se pueden usar cuando se construyen relaciones, como se resume en la figura 5-20.

En algunos casos la normalización no es pertinente. Si una tabla se separa en dos o más se crean restricciones de integridad referencial. Si el costo del procesamiento extra de las dos tablas y sus restricciones integrales es más alto que el beneficio de evitar anomalías de modificación, entonces no se recomienda la normalización. Además, en algunos casos es preferible crear columnas repetidas en lugar de usar las técnicas de normalización estándar, y en otros casos se usa la redundancia controlada para mejorar el desempeño.

➤ PREGUNTAS DEL GRUPO I

5.1 ¿Qué restricciones se deben colocar en una tabla para que sea considerada una relación?

5.2 Defina los siguientes términos: *relación*, *tuple*, *atributo*, *archivo*, *registro*, *campo*, *tabla*, *renglón*, *columna*.

5.3 Defina *dependencia funcional*. Dé un ejemplo sobre dos atributos que tengan dependencia funcional, y otro de dos atributos que no tengan dependencia funcional.

5.4 Si EID determina funcionalmente a Actividad, ¿significa que sólo puede existir un valor de EID en la relación? ¿Por qué?

5.5 Defina *determinante*.

5.6 Proporcione un ejemplo de una relación que tenga una dependencia funcional en la cual el determinante posea dos o más atributos.

5.7 Defina *llave*.

5.8 Si EID es la llave de una relación, ¿es un determinante? ¿Puede un valor determinado de EID ocurrir más de una vez en la relación?

5.9 ¿Qué es una anomalía de eliminación? Dé un ejemplo diferente al que se menciona en este texto.

5.10 ¿Qué es una anomalía de inserción? Mencione un ejemplo diferente al de este texto.

5.11 Explique la relación de las siguientes formas normales: primera, segunda, tercera, Boyce-Codd, cuarta, quinta y de dominio-llave.

5.12 Defina la *segunda forma normal*. Mencione un ejemplo sobre la relación en 1NF, pero no en 2NF. Transforme la relación en relaciones en la forma 2NF.

5.13 Defina la *tercera forma normal*. Proporcione un ejemplo de la relación en 2NF, pero no en 3NF. Transforme la relación en relaciones en la forma 3NF.

5.14 Defina la forma *BCNF*. Dé un ejemplo sobre la relación en 3NF, pero no en BCNF. Transforme la relación en relaciones en la forma BCNF.

5.15 Defina la *dependencia multivaluada*. Mencione un ejemplo.

5.16 ¿Por qué las dependencias multivaluadas no son un problema en las relaciones con sólo dos atributos?

5.17 Defina la *cuarta forma normal*. Mencione un ejemplo de la relación en BCNF, pero no en 4NF. Transforme la relación en relaciones en 4NF.

5.18 Defina la *forma normal dominio-llave*. ¿Por qué es importante?

5.19 Transforme la siguiente relación a DK/NF. Establezca las suposiciones apropiadas acerca de dependencias funcionales y dominios.

EQUIPO (Fabricante, Modelo, FechadeAdquisición, NombredelComprador, TeléfonodelComprador, LocalizacióndelaPlanta, Ciudad, Estado, CódigoPostal)

5.20 Transforme la siguiente relación a DK/NF. Establezca las suposiciones apropiadas acerca de las dependencias funcionales y los dominios.

FACTURA (Número, NombredelCliente, NúmerodeCliente, DirecciódelCliente, NúmerodelArtículo, PreciodelArtículo, CantidaddeArtículo, NúmerodeVendedor, NombredelVendedor, Subtotal, Impuesto, AdeudoTotal)

5.21 Conteste otra vez la pregunta 5.20, pero agregue el atributo EstadodelosImpuestosdelCliente (0 si no está exento, 1 si está exento). También agregue la restricción de que no habrá impuesto si EstadodelosImpuestosdelCliente = 1.

5.22 Dé un ejemplo, diferente al de este texto, en el cual juzgue que la normalización no vale la pena. Muestre las relaciones y justifique su diseño.

5.23 Explique dos situaciones en las que los diseñadores de las bases de datos intencionalmente deciden crear datos duplicados. ¿Cuál es el riesgo de tal diseño?

➤ PREGUNTAS DEL GRUPO II

5.24 Considere la siguiente definición de relación y los datos de muestra:

Relación PROYECTO

ProyectoID	NombredelEmpleado	SalariodelEmpleado
100A	Jones	64K
100A	Smith	51K
100B	Smith	51K
200A	Jones	64K
200B	Jones	64K
200C	Parks	28K
200C	Smith	51K
200D	Parks	28K

Proyecto(ProyectoID, NombredelEmpleado, SalariodelEmpleado)
Donde ProyectoID es el nombre de un proyecto de trabajo

NombredelEmpleado es el nombre de un empleado que trabaja en el proyecto

SalariodelEmpleado es el salario del empleado cuyo nombre es NombredelEmpleado

Suponga que todas las dependencias funcionales y restricciones son evidentes en estos datos, ¿cuál de los siguientes enunciados es cierto?

a. ProyectoID → NombredelEmpleado
b. ProyectoID → SalariodelEmpleado
c. (ProyectoID, NombredelEmpleado) → SalariodelEmpleado
d. NombredelEmpleado → SalariodelEmpleado
e. SalariodelEmpleado → ProyectoID
f. SalariodelEmpleado → (ProyectoID, NombredelEmpleado)

Conteste las siguientes preguntas:

g. ¿Cuál es la llave de PROYECTO?
h. ¿Todos los atributos que no son llaves (si los hay) dependen por completo de las llaves?
i. ¿En cuál forma normal está PROYECTO?
j. Describa dos anomalías de modificación que tiene PROYECTO
k. ¿ProyectoID es un determinante?
l. ¿NombredelEmpleado es un determinante?
m. ¿(ProyectoID, NombredelEmpleado) es un determinante?
n. ¿SalariodelEmpleado es un determinante?
o. ¿Tiene esta relación una dependencia transitiva? Si es así, ¿cuál es?
p. Rediseñe esta relación para eliminar las anomalías de modificación

5.25 Considere la siguiente definición de relación y datos muestra:

Relación PROYECTO-HORAS

NombredelEmpleado	ProyectoID	TareasID	Teléfono	HorasTotales
Don	100A	B-1	12345	12
Don	100A	P-1	12345	12
Don	200B	B-1	12345	12
Don	200B	P-1	12345	12
Pam	100A	C-1	67890	26
Pam	200A	C-1	67890	26
Pam	200D	C-1	67890	26

HORAS-PROYECTO (NombredelEmpleado, ProyectoID, TareasID, Teléfono, HorasTotales) Donde NombredelEmpleado es el nombre de un empleado

ProyectoID es el nombre de un proyecto

TareasID es el nombre estándar de la tarea

Teléfono es el número telefónico del empleado

HorasTotales son las horas que ha trabajado el empleado en este proyecto

Suponga que todas las dependencias funcionales y restricciones son evidentes en estos datos, ¿cuáles de los siguientes enunciados son verdaderos?

a. NombredelEmpleado → ProyectoID
b. NombredelEmpleado → ProyectoID
c. NombredelEmpleado → TareasID

d. NombredelEmpleado →→ TareasID
e. NombredelEmpleado → Teléfono
f. NombredelEmpleado → HorasTotales
g. (NombredelEmpleado, ProyectoID) → HorasTotales
h. (NombredelEmpleado, Teléfono → TareasID
i. ProyectoID → TareasID
j. TareasID → ProyectoID

Conteste las siguientes preguntas:

k. Enumere todos los determinantes
l. ¿Esta relación contiene una dependencia transitiva? Si es así, ¿cuál es?
m. ¿Esta relación contiene una dependencia multivalor? Si es así, ¿cuáles son los atributos no relacionados?
n. Describa la anomalía de eliminación que presenta esta relación
o. ¿Cuántos temas contiene esta relación?
p. Rediseñe esta relación para eliminar las anomalías de modificación. ¿Cuántas relaciones usó? ¿Cuántos temas contiene cada una de sus nuevas relaciones?

5.26 Considere la siguiente definición de dominio, relación y llave:

Definiciones de dominio

NombredelEmpleado	en	CHAR(20)
NúmeroTelefónico	en	DEC(5)
NombredelEquipo	en	CHAR(10)
Localización	en	CHAR(7)
Costo	en	CURRENCY (MONEDA)
Fecha	en	YYMMDD (AAMMDD)
Hora	en	HHMM donde HH es el valor entre 00 y 23, y MM entre 00 y 59

Definiciones de relación, llave y restricción.

EMPLEADO (NombredelEmpleado, NúmeroTelefónico)
- Llave: NombredelEmpleado
- Restricciones: NombredelEmpleado → NúmeroTelefónico

EQUIPO (NombredelEquipo, Localización, Costo)
- Llave: NombredelEquipo
- Restricciones: NombredelEquipo → Localización
  NombredelEquipo → Costo

CITA (Fecha, Hora, NombredelEquipo, NombredelEmpleado)
- Llave: (Fecha, Hora, NombredelEquipo)
- Restricciones: (Fecha, Hora, NombredelEquipo) → NombredelEmpleado

a. Modifique las definiciones para agregar esta restricción: un empleado no puede anotarse en más de una cita
b. Defina horario nocturno para referirse a las horas entre las 21:00 y las 5:00. Agregue un atributo Tipo Empleado cuyo valor sea 1 si el empleado trabaja durante la noche. Cambie este diseño para forzar la restricción de que sólo los empleados que trabajan en la noche pueden programar citas nocturnas

➤ PREGUNTAS DEL PROYECTO FIREDUP

FiredUp alquila a un grupo de diseñadores de bases de datos (¡quienes deben estar muy motivados!) para que creen las siguientes relaciones con el fin de que dicha base sirva para hacer el seguimiento de sus estufas, reparaciones y datos de los clientes. Vea el proyecto al final de los capítulos 1 al 3 si necesita revisar las necesidades de los clientes. Para cada una de las siguientes relaciones, especifique llaves candidatas, dependencias funcionales, dependencias multivaluadas (si las hay). Justifique estas especificaciones a menos que sean muy obvias. Considerando sus especificaciones acerca de las llaves, ¿qué forma normal tiene cada relación? Transforme cada una en dos o más relaciones que estén en la forma normal dominio-llave. Indique la llave primaria de cada tabla, llaves candidatas, llaves externas y especifique cualquier restricción de integridad referencial.

Para contestar estas preguntas, suponga lo siguiente:

- El tipo de estufa y la versión determinan la capacidad del tanque
- Una estufa puede ser reparada muchas veces, pero nunca más de una en un día determinado
- Cada estufa tiene su propia factura de reparación
- Una estufa puede estar registrada con diferentes usuarios, pero nunca al mismo tiempo
- Una estufa tiene muchos componentes y cada uno se puede usar en varias estufas. Por lo tanto, FiredUp conserva los registros acerca de los tipos de partes o refacciones, tales como la *válvula del quemador*, y no acerca de las partes, como por ejemplo las válvulas del quemador número 41734 fabricado el 12 de diciembre del 2001

A. PRODUCTO1 (NúmerodeSerie, Tipo, NúmerodeVersión, CapacidaddelTanque, FechadeFabricación, InicialesdelInspector)

B. PRODUCTO2 (NúmerodeSerie, Tipo, CapacidaddelTanque, FechadeReparación, NúmerodeFacturadeReparación, CostodeReparación)

C. REPARACIÓN1 (NúmerodeFacturadeReparación, FechadeReparación, CostodeReparación, NombredelReparador, NúmeroTelefónicodelReparador)

D. REPARACIÓN2 (NúmerodeFacturadeReparación, FechadeReparación, CostodeReparación, NombredelReparador, NúmeroTelefónicodelReparador, NúmerodeSerie, Tipo, CapacidaddelTanque)

E. REPARACIÓN3 (FechadeReparación, CostodeReparación, NúmerodeSerie, FechadeFabricación)

F. ESTUFA1 (NúmerodeSerie, NúmerodeFacturadeReparación, NúmerodelComponente)

G. ESTUFA2 (NúmerodeSerie, NúmerodeFacturadeReparación, PropietarioRegistradoID)

Suponga que es necesario registrar al propietario de la estufa, aun cuando nunca haya sido reparada.

H. Considerando los supuestos de este caso, las relaciones y los atributos en los puntos A-G, y su conocimiento acerca de negocios pequeños, construya un conjunto de relaciones dominio-llave para FiredUp. Indique llaves primarias, llaves externas y restricciones interrelacionadas.

CAPÍTULO 6

Diseño de bases de datos utilizando modelos de entidad-relación

En el capítulo 3 analizamos las especificaciones de modelos de datos usando el modelo de entidad-relación, y en el capítulo 5 estudiamos el modelo relacional y la normalización. En el presente capítulo reunimos estos temas para ilustrar la transformación de los requerimientos de los usuarios expresados en modelos de entidad-relación en el diseño de bases de datos relacionales. Estos diseños son independientes de cualquier DBMS en particular.

Este capítulo tiene tres secciones: en la primera mostramos técnicas para transformar modelos de datos de entidad-relación en diseños relacionales. Como podrá ver, la normalización desempeña un papel importante; en la segunda sección se aplican estas técnicas para transformar cuatro estructuras de datos comunes en las aplicaciones de bases de datos; por último, este capítulo concluye con un análisis de llaves sustitutas y de valores nulos.

➤ TRANSFORMACIÓN DE LOS MODELOS ENTIDAD-RELACIÓN EN DISEÑOS DE BASES DE DATOS RELACIONALES

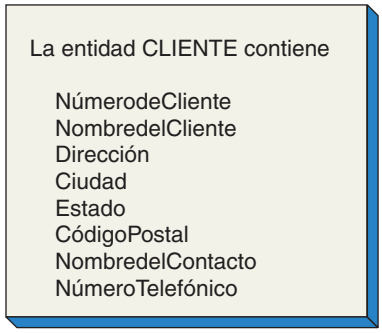
De acuerdo con el modelo entidad-relación, las cosas que los usuarios quieren registrar se representan mediante **entidades,** y las relaciones entre esas entidades se representan mediante **relaciones** definidas en forma explícita. Esta sección describe cómo transformar esas entidades y relaciones en términos del modelo relacional.

Representación de entidades con el modelo relacional

La representación de entidades por medio de un modelo relacional es directa. Comenzamos definiendo una relación para cada entidad. El nombre de la relación es el de la

➤ FIGURA 6-1

Representación de una entidad con una relación: (a) la entidad CLIENTE, y (b) la relación que representa la entidad CLIENTE



(a)

CLIENTE (NúmerodeCliente, NombredelCliente, Dirección, Ciudad, Estado, CódigoPostal, NombredelContacto, NúmeroTelefónico)

(b)

entidad y los atributos de la relación son los de la entidad. Entonces, examinamos cada relación de acuerdo con el criterio de normalización que vimos en el capítulo 5. Puede o no ser necesario cambiar este diseño inicial.

El ejemplo en la figura 6-1(a) es la entidad que se muestra en la figura 3-1. La entidad CLIENTE contiene los siguientes atributos: NúmerodeCliente, NombredelCliente, Dirección, Ciudad, Estado, CódigoPostal, NombredelContacto y NúmeroTelefónico. Para representar esta entidad con una relación, definimos una relación para la entidad y colocamos los atributos como columnas en la relación. Si sabemos qué atributo del modelo de datos identifica esta entidad, éste se convertirá en la llave de la relación. En caso contrario, debemos preguntar a los usuarios o investigar de otra manera los requerimientos para determinar qué atributo o atributos pueden definir una entidad. En este caso, suponemos que la llave es NúmerodeCliente. En esta figura, al igual que en otras, las llaves de las relaciones están subrayadas.

EL PAPEL DE LA NORMALIZACIÓN. Durante la fase de requerimientos, la única disposición que tiene una entidad es que es importante para el usuario. No se intenta determinar si ésta se ajusta o no a cualquiera de los criterios de normalización que se analizaron en el capítulo 5. Por lo tanto, cuando ya se definió la relación para una entidad, se deberá examinar de acuerdo al criterio de normalización.

Considere, por ejemplo, la relación CLIENTE en la figura 6-1(b). ¿Está en la forma normal dominio-llave (DK/NF)? Para saberlo, necesitamos conocer las restricciones de esta relación. Sin una descripción completa de los requisitos implícitos no conocemos todas las restricciones, como por ejemplo las de dominio. Pero podemos descubrir algunos de los requisitos a partir de los nombres de los atributos y de nuestro conocimiento sobre la naturaleza del negocio.

Primero, NúmerodeCliente determina todos los otros atributos, ya que los únicos valores de NombredelCliente, Dirección, Ciudad, Estado, CódigoPostal, NombredelContacto y NúmeroTelefónico se pueden determinar a partir de un valor específico de NúmerodeCliente. Sin embargo, hay restricciones que surgen de las otras dependencias funcionales. CódigoPostal determina Ciudad y Estado, y NombredelContacto determina NúmeroTelefónico. Para crear un conjunto de relaciones en DK/NF es necesario hacer que estas dependencias funcionales adicionales sean una consecuencia lógica de los dominios y las llaves, y esto lo podemos hacer mediante la definición de las tres relaciones que se muestran en la figura 6-2. Observe que la llave de CLIENTE es NúmerodeCliente, la de CÓDIGO-POSTAL-TABLA es CódigoPostal, y la llave de CONTACTO es NombredelContacto. Observe también las restricciones de integridad referencial.

El diseño en la figura 6-2 está en DK/NF y no hay anomalías de modificación. Esto es, podemos añadir nuevos códigos postales y contactos sin tener que agregar un clien-

➤ FIGURA 6-2

Representación de la entidad Cliente con relaciones en la forma normal dominio/llave

CLIENTE (NúmerodeCliente, Dirección, CódigoPostal, NombredelContacto)

CÓDIGO-POSTAL-TABLA (CódigoPostal, Ciudad, Estado)

CONTACTO (NombredelContacto, NúmeroTelefónico)

Restricciones de integridad referencial:

El CódigoPostal en CLIENTE debe existir en CódigoPostal en CÓDIGO-POSTAL-TABLA
NombredelContacto en CLIENTE debe existir en NombredelContacto en CONTACTO

te con el nuevo código postal o contacto. Además, cuando se elimina el último cliente en un código postal no se pierden la ciudad y el estado. Pero, como señalamos al final del capítulo 5, la mayoría de los profesionales consideran que este diseño es demasiado puro; el hecho de dividir CódigoPostal, Ciudad y Estado haría que este diseño fuera difícil de usar. Por lo tanto, probablemente resultaría mejor dejar CódigoPostal, Ciudad y Estado en la relación CLIENTE.

¿Qué podemos decir acerca de CONTACTO? Si la relación entre un contacto y una compañía es 1:1, entonces se gana poco con colocar los datos del contacto en su propia tabla. La relación en la figura 6-1(b) es aceptable. Si la relación no es 1:1, entonces CONTACTO estaría en una entidad, con la relación apropiada en CLIENTE (la cual podría ser N:1 o 1:N) y revisando el modelo E-R adecuadamente.

En otros ejemplos es preferible el diseño DK/NF. Considere la entidad COMISIÓN-VENTAS en la figura 6-3(a). Si intentamos representar esta entidad con una relación, como se muestra en la figura 6-3(b), el resultado sería un desorden de atributos con muchas anomalías potenciales de modificación.

Esta relación obviamente contiene más de un tema. Si la examinamos veremos que contiene uno sobre vendedor, otro acerca de ventas durante algún periodo y otro tema relativo a los cheques de las comisiones de ventas. Las relaciones en DK/NF que representan esta entidad se muestran en la figura 6-3(c). Intuitivamente, este diseño parece superior al de la figura 6-3(b); es más directo y se ajusta mejor.

Para resumir el análisis, cuando se representa una entidad con el modelo relacional, el primer paso es construir una relación que tenga todos los atributos de las entidades como columnas. Después, la relación se examina en función del criterio de normalización. En muchas instancias el diseño se puede mejorar desarrollando conjuntos de relaciones en DK/NF.

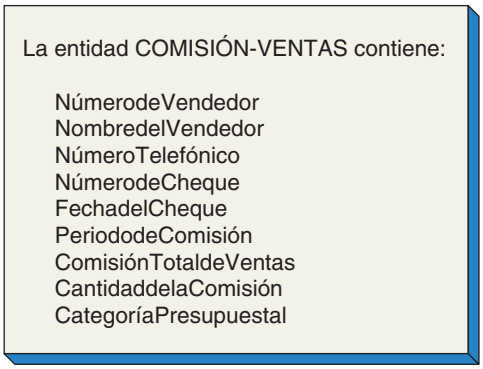
Sin embargo, no siempre se prefieren las relaciones DK/NF. Si son artificiales y difíciles de trabajar, puede convenir un diseño que no sea DK/NF. El desempeño también puede ser un factor. Tener acceso a dos o tres relaciones para obtener los datos necesarios acerca de un cliente puede implicar una cantidad de tiempo prohibitiva.

Sin considerar nuestra decisión acerca de normalizar o no, deberíamos examinar cada relación de la(s) entidad(es) en función del criterio de normalización. Así que, si vamos a "pecar", sería mejor tomar una decisión consciente y bien informada. En el proceso, también aprenderemos los tipos de modificación de anomalías a los cuales son vulnerables las relaciones.

REPRESENTACIÓN DE ENTIDADES DÉBILES. Las entidades débiles requieren un trato especial cuando se crea el diseño relacional. Recordemos que la existencia de una entidad débil depende de otra. Si la entidad débil tiene una existencia dependiente, pero no depende de un identificador, se puede representar usando las técnicas descritas en la última sección. La dependencia necesita registrarse en el diseño relacional para que ninguna aplicación provoque una entidad débil sin un padre apropiado (la entidad de la cual depende la entidad débil). Además, se deben implementar reglas del ne-

➤ FIGURA 6-3

Entidad con la normalización apropiada: (a) entidad COMISIÓN-VENTAS, (b) representación de COMISIÓN-VENTAS con una sola relación, y (c) representación de COMISIÓN-VENTAS con las relaciones forma normal dominio/llave



(a)

COMISIÓN-VENTAS (NúmerodeVendedor, NombredelVendedor, NúmeroTelefónico, NúmerodeCheque, FechadelCheque, PeriododeComisión, ComisiónTotaldeVentas, CantidaddelaComisión, CategoríaPresupuestal)

Dependencias funcionales:
NúmerodeCheque es llave
NúmerodeVendedor determina el NombredelVendedor, NúmeroTelefónico, CategoríaPresupuestal
(NúmerodeVendedor, PeriododeComisión) determina ComisiónTotaldeVentas, CantidaddelaComisión

(b)

VENDEDOR (NúmerodeVendedor, NombredelVendedor, NúmeroTelefónico, CategoríaPresupuestal)
VENDEDOR (NúmerodeVendedor, PeriododeComisión, ComisiónTotaldeVentas, CantidaddelaComisión)
CHEQUE-COMISIÓN (NúmerodeCheque, FechadelCheque, NúmerodeVendedor, PeriododelaComisión)

Restricción de integridad referencial:
NúmerodeVendedor en VENTAS debe existir en NúmerodeVendedor en VENDEDOR
(NúmerodeVendedor, PeriododeComisión) en CHEQUE-COMISIÓN debe existir en (NúmerodeVendedor, PeriododeComisión) en VENTAS

(c)

gocio para que cuando se suprima el padre, la entidad débil también quede eliminada. Estas reglas se deben escribir en el diseño relacional.

La situación anterior es un poco diferente si la entidad débil también es dependiente de un identificador. En la figura 6-4(a), ARTÍCULO-LÍNEA es una entidad débil dependiente de un identificador. Es débil porque su existencia lógica depende de FACTURA, y es dependiente de un identificador porque su identificador contiene el de FACTURA.

Cuando se crea una relación para una entidad dependiente de un identificador, debemos asegurarnos que tanto la llave del padre como la de la entidad aparezcan en la relación. Por ejemplo, considere qué pasaría si simplemente establecemos una relación para ARTÍCULO-LÍNEA y no incluimos la llave de FACTURA. Una relación como ésta se muestra en la figura 6-4(b). ¿Cuál es la llave de esta relación? Debido a que ARTÍCULO-LÍNEA es dependiente de un identificador no tiene una llave completa y, de hecho, esta relación podría tener renglones duplicados. (Lo anterior podría suceder si dos facturas tienen igual cantidad del mismo artículo en la misma línea.)

De esta forma, para una entidad débil dependiente de un identificador es necesario agregar la llave de la entidad padre a la relación de la entidad débil, y este atributo agregado sería parte de la llave de la entidad débil. En la figura 6-4(c) hemos agregado

➤ FIGURA 6-4

Representación relacional de una entidad débil: (a) ejemplo de entidad débil, (b) representación de la relación ARTÍCULO-LÍNEA con la llave incorrecta, y (c) relación ARTÍCULO-LÍNEA con la llave correcta

(a)

ARTÍCULO-LÍNEA (NúmerodeLínea, Cantidad, NúmerodeArtículo, Descripción, Precio, PrecioExtra)

(b)

ARTÍCULO-LÍNEA (NúmerodeFactura, NúmerodeLínea, Cantidad, NúmerodeArtículo, Descripción, Precio, PrecioExtra)

(c)

NúmerodeFactura, la llave de FACTURA, a los atributos en ARTÍCULO-LÍNEA. La llave de ARTÍCULO-LÍNEA es la compuesta {NúmerodeFactura, NúmerodeLínea}.

Representación de relaciones TIENE-UN

Hay dos tipos de relaciones en el modelo E-R: las relaciones TIENE-UN entre entidades de tipos lógicos diferentes, y las relaciones ES-UN entre entidades que son subtipos de un tipo lógico común. En esta sección abordaremos las relaciones TIENE-UN; después analizaremos las relaciones ES-UN. Hay tres tipos de relaciones TIENE-UN: uno a uno, uno a muchos y muchos a muchos.

REPRESENTACIÓN DE RELACIONES UNO A UNO. La forma más simple de relación binaria es uno a uno (1:1), en la que una entidad de un tipo está relacionada únicamente con una entidad de otro tipo. En el ejemplo de EMPLEADO y AUTO, suponga que un empleado tiene asignado un automóvil y un automóvil está asignado a un empleado. En la figura 6-5 se muestra un diagrama E-R para esta relación.

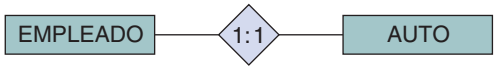
La representación de una relación 1:1 con el modelo relacional es directa. Primero, cada entidad se representa con una relación y entonces la llave de una de las relaciones se coloca en la otra. En la figura 6-6(a), la llave de EMPLEADO se almacena en AUTO y en la figura 6-6(b) la llave AUTO se almacena en EMPLEADO.

Cuando la llave de la relación se almacena en una segunda relación, se llama **llave externa.** En la figura 6-6(a), NúmerodeEmpleado es una llave externa en AUTO, y en la figura 6-6(b) NúmerodeLicencia es una llave externa en EMPLEADO. En esta figura, las llaves externas se muestran en cursivas, pero algunas veces puede ver que las llaves externas se representan mediante una línea punteada. Aunque, en otros casos, las llaves externas no se señalan de ninguna manera en especial. En este texto, cuando hay peligro de confusión, se muestran las llaves externas en cursivas, pero la mayor parte del tiempo no tienen ninguna señal.

Para una relación 1:1, la llave de cualesquiera de las tablas se puede colocar como una llave externa en la otra tabla. En la figura 6-6(a) la llave externa NúmerodeLicencia se coloca en EMPLEADO. Con este diseño podemos navegar de EMPLEADO a AUTO, o de AUTO a EMPLEADO. En el primer caso tenemos un empleado y queremos saber acerca del automóvil que le fue asignado. Para obtener los datos del empleado utilizamos NúmerodeEmpleado para obtener los renglones que le corresponden, en EMPLEADO. De ahí obtenemos el NúmerodeLicencia del automóvil que tiene asignado. Entonces utilizamos este número para buscar los datos del automóvil en AUTO.

➤ FIGURA 6-5

Ejemplo de una relación 1:1



➤ FIGURA 6-6

Alternativas para la representación de relaciones 1:1: (a) colocación de la llave AUTO en EMPLEADO, y (b) colocación de la llave EMPLEADO en AUTO

EMPLEADO (NúmerodeEmpleado, NombredelEmpleado, NúmeroTelefónico. . . *NúmerodeLicencia*)

AUTO (NúmerodeLicencia, NúmerodeSerie, Color, Marca, Modelo. . .)

La integridad referencial contiene:
NúmerodeLicencia en EMPLEADO debe existir en NúmerodeLicencia en AUTO

(a)

EMPLEADO (NúmerodeEmpleado, NombredelEmpleado, NúmeroTelefónico. . .)

AUTO (NúmerodeLicencia, NúmerodeSerie, Color, Marca, Modelo. . . *NúmerodeEmpleado*)

Restricción de integridad referencial:
NúmerodeEmpleado en AUTO debe existir en NúmerodeEmpleado en EMPLEADO

(b)

Ahora consideremos la otra dirección. Suponga que tenemos un automóvil y deseamos saber a qué empleado le fue asignado. Usando el diseño en la figura 6-6(a) accedemos a la tabla EMPLEADO y buscamos el renglón que tiene el número de licencia. Los datos del empleado a quien le fue asignado el automóvil aparecen en ese renglón.

Realizamos acciones similares para dirigirnos a cualquier otra dirección para el diseño alternativo, en el cual la llave externa de NúmerodeEmpleado se coloca en AUTO. Usando este diseño, para cambiar de EMPLEADO a AUTO, accedemos directamente a la relación AUTO y buscamos el renglón en AUTO que tiene el número de empleado como valor de NúmerodeEmpleado. Para ir de AUTO a EMPLEADO buscamos el renglón en AUTO con un NúmerodeLicencia determinado. De este renglón extraemos el NúmerodeEmpleado y lo utilizamos para tener acceso a los datos del empleado en EMPLEADO. Aquí estamos usando el término *buscar* para decir que "encontramos un renglón que da un valor de una de sus columnas". Más adelante, cuando analicemos los modelos DBMS le mostraremos cómo se hace.

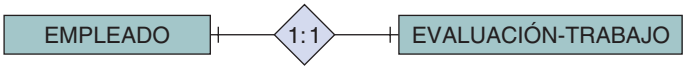
Aunque los dos diseños en la figura 6-6 son equivalentes conceptualmente hablando, pueden ser diferentes en cuanto a su funcionamiento. Por ejemplo, si una consulta en una dirección es más común que una consulta en otra, podemos inclinarnos por un diseño. También, si el producto DBMS es mucho más rápido en búsquedas de llaves primarias que en búsquedas de las llaves externas, también podemos preferir un diseño al otro.

RELACIONES 1:1 CUESTIONABLES. La figura 6-7 muestra otra relación 1:1, en la cual cada EMPLEADO tiene una EVALUACIÓN-TRABAJO y cada EVALUACIÓN-TRABAJO corresponde a un empleado en particular. Observe que las marcas verticales en la línea significan que la relación es obligatoria en ambas direcciones. Cuando la relación es 1:1 y es obligatoria en ambas direcciones, es común que los registros describan aspectos diferentes de la misma entidad, especialmente si, como en el caso en la figura 6-7, ambas tienen la misma llave. Cuando esto ocurre, por lo general los registros se deben combinar en una relación. Hay que aprender a considerar con reservas estas relaciones obligatorias 1:1.

Sin embargo, a veces la separación de una entidad en dos relaciones se puede justificar. Una justificación es el desempeño. Por ejemplo, suponga que los datos de EVALUACIÓN-TRABAJO son extensos y se usan con menos frecuencia que los otros datos del empleado. En estas circunstancias podemos almacenar apropiadamente EVALUA-

➤ FIGURA 6-7

Relación 1:1 sospechosa



CIÓN-TRABAJO en una tabla por separado, así las solicitudes más comunes para datos del empleado no evaluadas se podrán procesar rápidamente.

Una mayor seguridad sería la segunda justificación para separar en dos a una entidad lógica. Si el DBMS no tiene seguridad en el nivel de dato-artículo, se pueden necesitar los datos de EVALUACIÓN-TRABAJO para la separación, con el fin de evitar que los usuarios no autorizados tengan acceso a ellos. También puede ser preferible colocar EVALUACIÓN-TRABAJO en una tabla por separado y almacenarla en un disco al que puedan tener acceso sólo ciertos usuarios.

La conclusión de este análisis es que no todas las relaciones 1:1 son cuestionables, sino sólo aquellas que parecen describir diferentes aspectos de la misma entidad. Por ejemplo, la relación obligatoria 1:1 entre EMPLEADO y AUTO es bastante apropiada, porque cada relación describe una cuestión lógicamente diferente.

REPRESENTACIÓN DE RELACIONES UNO A MUCHOS. El segundo tipo de relación binaria es de uno a muchos (1:N), en la cual una entidad de un tipo se puede relacionar con muchas entidades de otro tipo. La figura 6-8 es un diagrama E-R de una relación uno a muchos entre profesores y estudiantes. En esta relación, PROFESOR se relaciona con muchos ESTUDIANTEs que él o ella asesoran. Como se estableció en el capítulo 3, el óvalo significa que la relación entre PROFESOR y ESTUDIANTE es opcional; por lo tanto, un profesor no necesita tener asesorados. La barra que cruza la línea hasta el otro extremo significa que un renglón ESTUDIANTE debe corresponder a un renglón PROFESOR.

Los términos **padre** e **hijo** algunas veces se aplican a las relaciones 1:N. La relación padre está en *el lado uno* de la relación, y la relación hijo está en el *lado muchos*. En la figura 6-8(a) PROFESOR es la entidad padre y ESTUDIANTE es la entidad hijo.

La figura 6-8 muestra otras dos relaciones uno a muchos. En la figura 6-8(b), una entidad DORMITORIO corresponde a muchas entidades ESTUDIANTEs, pero una entidad ESTUDIANTE corresponde a un DORMITORIO. Además, un dormitorio no tiene que tener estudiantes asignados, ni se requiere que un estudiante viva en un dormitorio.

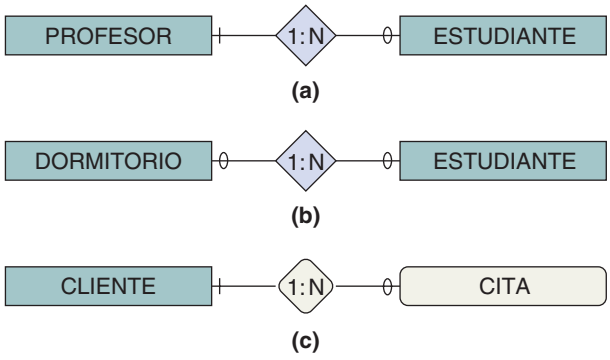
En la figura 6-8(c) un CLIENTE está relacionado con muchas entidades CITA, y una CITA en particular corresponde únicamente a un CLIENTE. Además, un CLIENTE puede tener o no una CITA, pero cada CITA debe corresponder a un CLIENTE.

Representar relaciones 1:N es simple y directo. Primero, cada entidad se representa mediante una relación, y después la llave de la relación que representa a la entidad padre se coloca en la relación que representa a la entidad hijo. De esta forma, para representar la relación ASESORADO de la figura 6-8(a), colocamos la llave de PROFESOR, NombredelProfesor, en la relación ESTUDIANTE, como se muestra en la figura 6-9.

La figura 6-9 es un ejemplo de lo que a veces se denomina **diagrama de estructura de datos,** en el cual las relaciones se muestran mediante rectángulos con líneas que representan relaciones, y los atributos de la llave están subrayados. Un tenedor, o pata de gallo, en una línea de relación indica una relación de muchos.

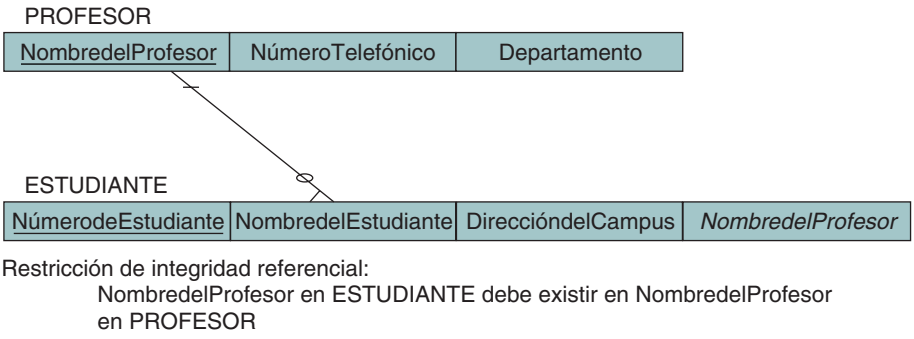
➤ FIGURA 6-8

Ejemplos de relaciones uno a muchos: (a) relación 1:N opcional a obligatoria, (b) relación 1:N opcional a opcional, y (c) relación 1:N con entidad débil



➤ FIGURA 6-9

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



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

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

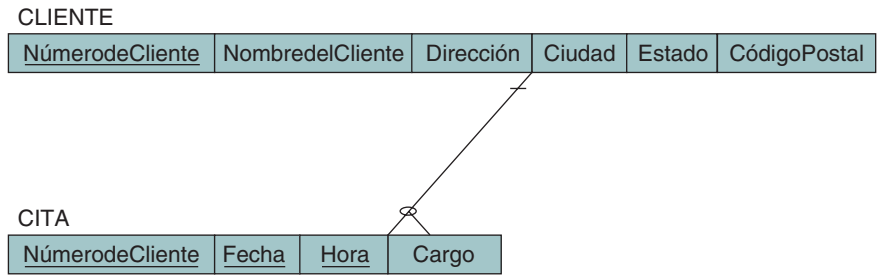
Compare esta situación con una representación de relaciones 1:1. En ambos casos almacenamos la llave de una relación como una llave externa en la segunda relación. Sin embargo, en una relación 1:1 no importa qué llave se mueva a la segunda relación. Pero sí tiene importancia en una relación 1:N. *La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

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

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

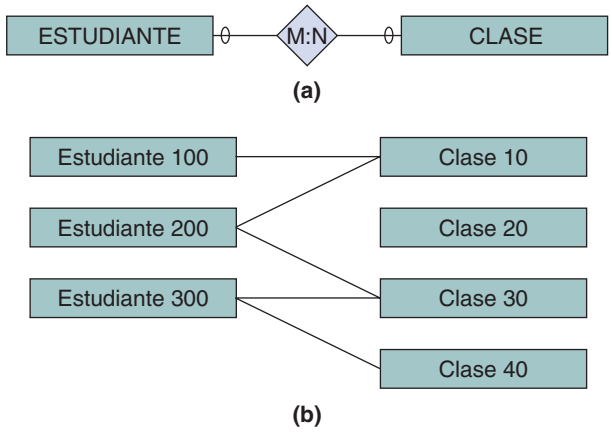
➤ FIGURA 6-10

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



➤ FIGURA 6-11

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



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

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

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

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

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

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

FIGURA 6-12

Representación incorrecta de una relación M:N

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

ESTUDIANTE

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

CLASE

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

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

FIGURA 6-13

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

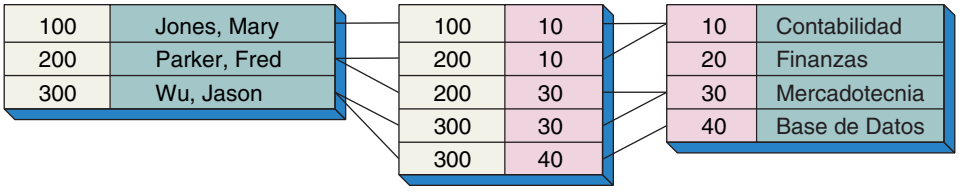
ESTUDIANTE (NúmerodeEstudiante, NombredelEstudiante)

CLASE (NúmerodeClase, NombredelaClase)

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

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

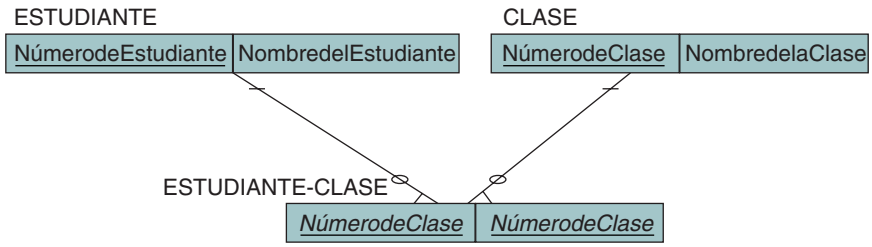
(a)



(b)

➤ FIGURA 6-14

Diagrama de estructura de datos para la relación ESTUDIANTE con CLASE



Representación de relaciones recursivas

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

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

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

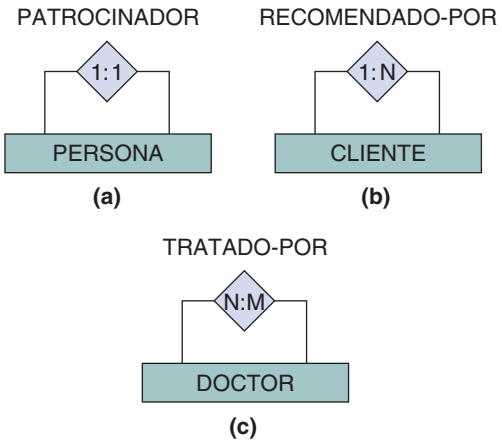
Esta técnica es idéntica a la de las relaciones no recursivas 1:1, excepto que los renglones de hijo y padre se ubican en la misma relación. Puede pensar el proceso de la forma siguiente: imagine que la relación es entre dos relaciones diferentes. Determine dónde van las llaves y después combine ambas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

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

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

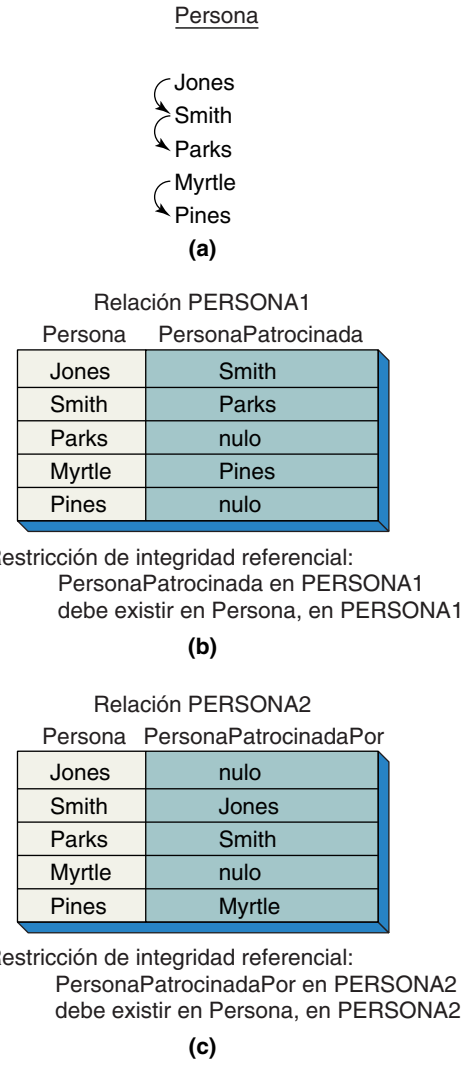
➤ FIGURA 6-15

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



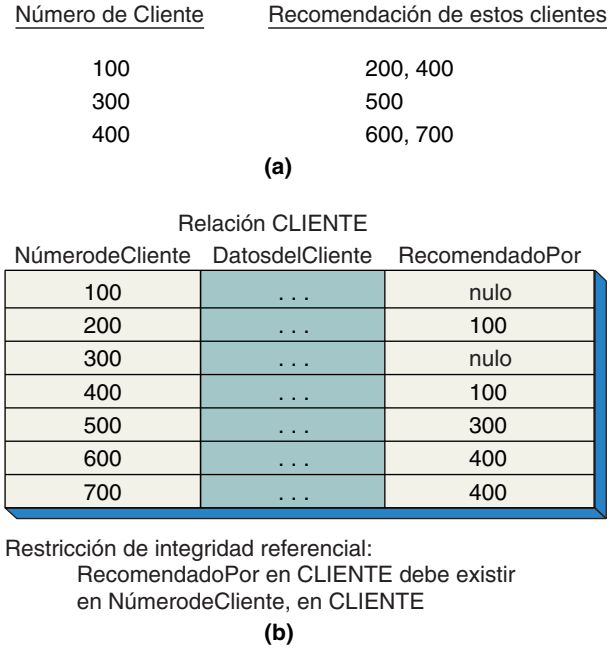
➤ FIGURA 6-16

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



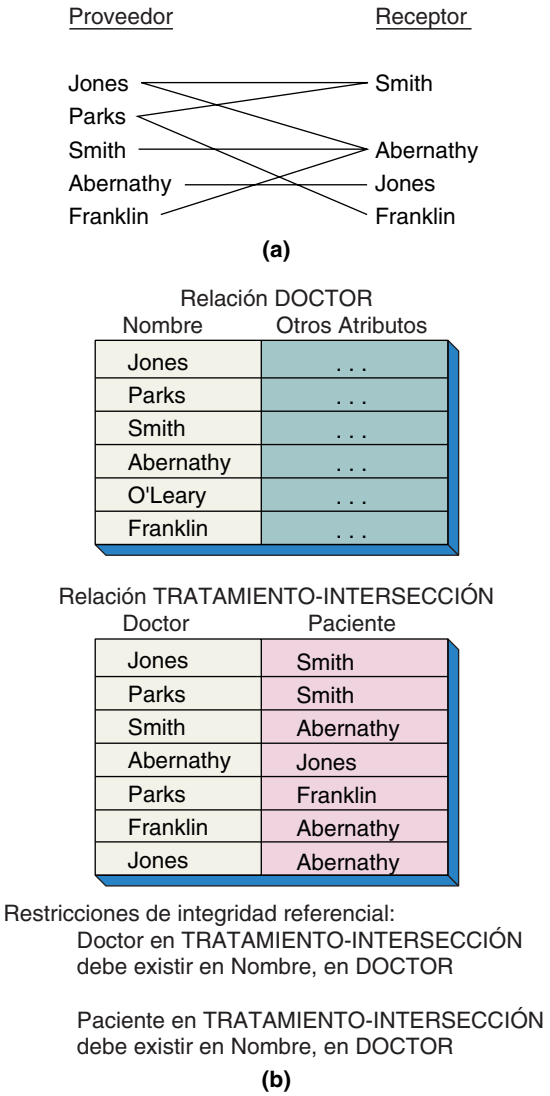
➤ FIGURA 6-17

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



➤ FIGURA 6-18

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



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

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

Representación de relaciones ternarias y de orden superior

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

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

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

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

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

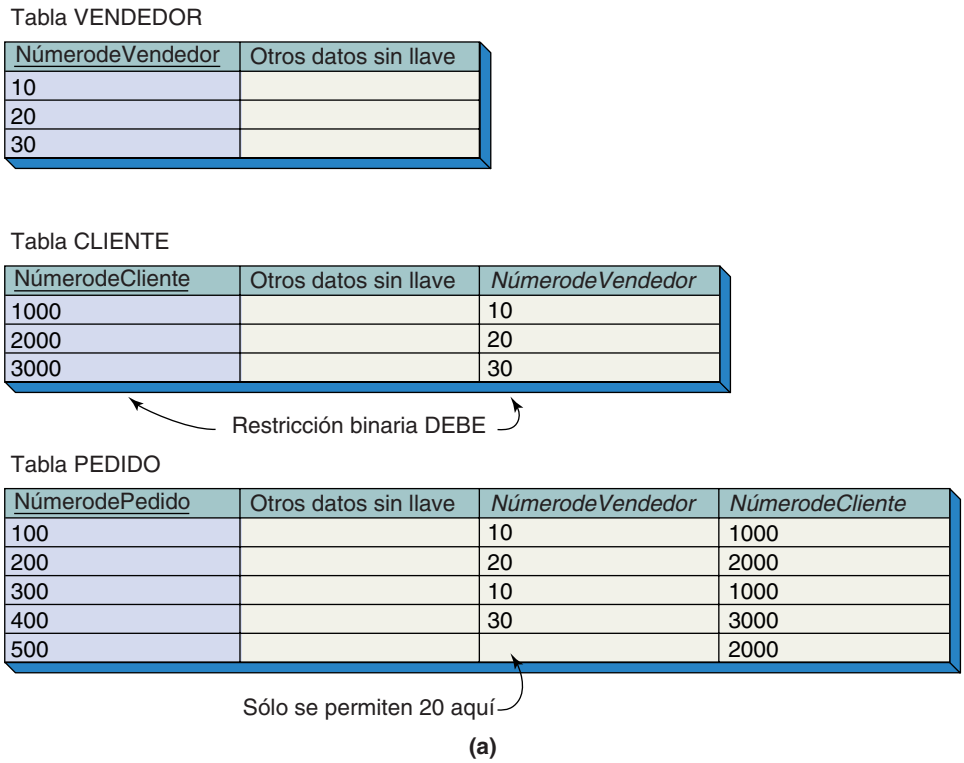
PEDIDO (NúmerodePedido, atributos sin datos llave, *NúmerodeCliente, NúmerodeVendedor)*
CLIENTE (NúmerodeCliente, atributos sin datos llave, *NúmerodeVendedor)*
VENDEDOR (NúmerodeVendedor, atributos sin datos llave)

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

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

➤ FIGURA 6-19

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



➤ FIGURA 6-19

(Continuación)

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

Tabla MEDICAMENTO

NúmerodeMedicamento	Otros datos sin llave
10	
20	
30	
45	
70	
90	

Tabla ALERGIA

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

Restricción binaria NO DEBE

Tabla RECETA

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

No pueden aparecer aquí ni 20 ni 45

(b)

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

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

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

Representación de relaciones ES-UN (subtipos)

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

➤ FIGURA 6-19

(Continuación)

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

Tabla REPARACIÓN

NúmerodeReparación	Otros datos sin llave
10	
20	
30	
40	

Tabla TAREA

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

Restricción binaria DEBE CUBRIR

Tabla AUTO-REPARACIÓN

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

aquí debe aparecer el 2002

(c)

CLIENTE-PERSONA: Dirección, NúmerodeSeguroSocial

CLIENTE-SOCIEDAD: NombredelSocioAdministrador, Dirección, NúmerodeIdentificaciónFiscal

CLIENTE-CORPORACIÓN: PersonadeContacto, NúmeroTelefónico, NúmerodeIdentificaciónFiscal

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

CLIENTE (NúmerodeCliente, NombredelCliente, CantidadAdeudada)

CLIENTE-PERSONA (NúmerodeCliente, Dirección, NúmerodeSeguroSocial)

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

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

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

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

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

➤ EJEMPLO DE DISEÑO

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

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

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

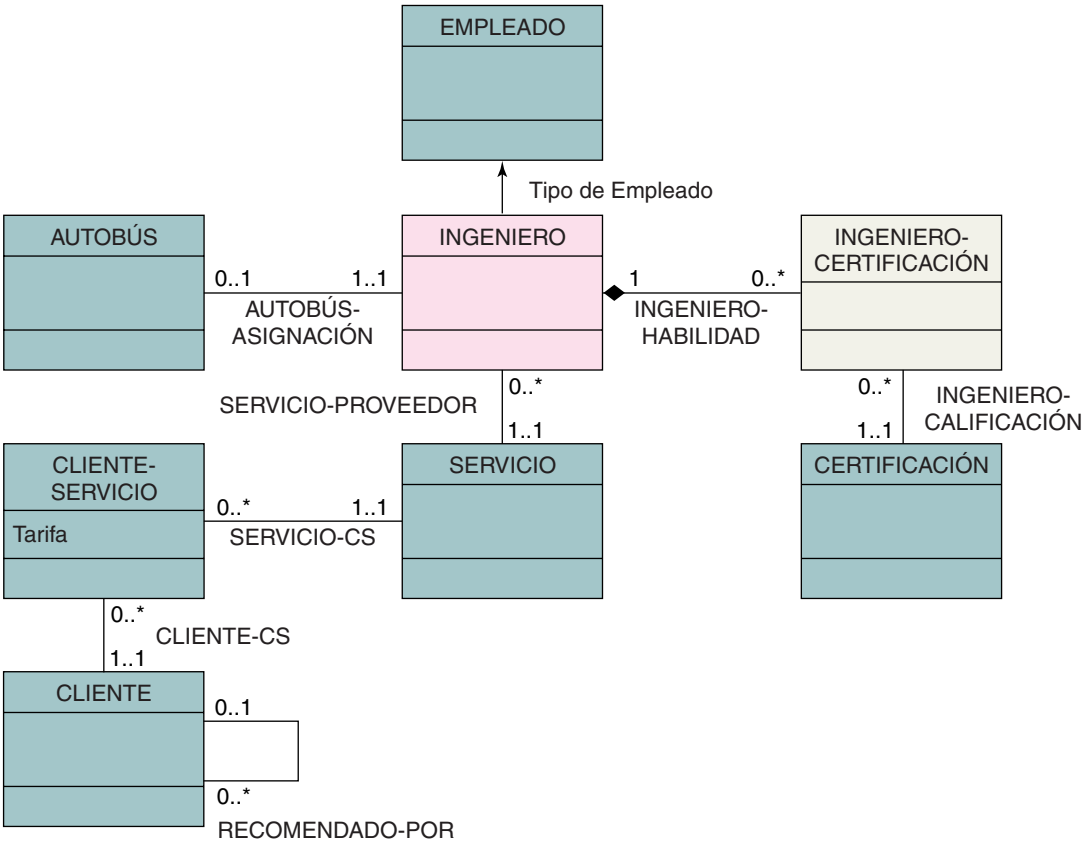
La relación entre EMPLEADO e INGENIERO ya está representada porque las relaciones tienen la misma llave, NúmerodeEmpleado. INGENIERO y AUTOBÚS tienen una relación 1:1 y así pueden estar relacionados por la colocación de la llave de una en la otra. Debido a que un autobús debe ser asignado a un empleado,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

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

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

➤ FIGURA 6-20

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



(a)

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

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

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

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

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

CLIENTE-SERVICIO (NúmerodeFactura, NúmerodeCliente, Tarifa)

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

CERTIFICACIÓN (NombredeCertificación, otros atributos sin llave de CERTIFICACIÓN)

(b)

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

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

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

➤ ÁRBOLES, REDES Y LISTAS DE MATERIALES

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

ÁRBOLES

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

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

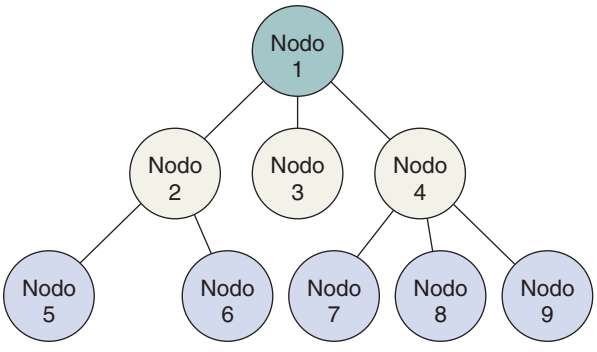
Los descendientes de un nodo se llaman **hijos.** En general, no hay un límite en el número de hijos que puede tener un nodo. El nodo 2 tiene dos hijos, los nodos 5 y 6; el nodo 3 no tiene hijo; y el nodo 4 tiene tres hijos, los nodos 7, 8 y 9. Los nodos que tienen el mismo padre se llaman **gemelos,**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

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

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

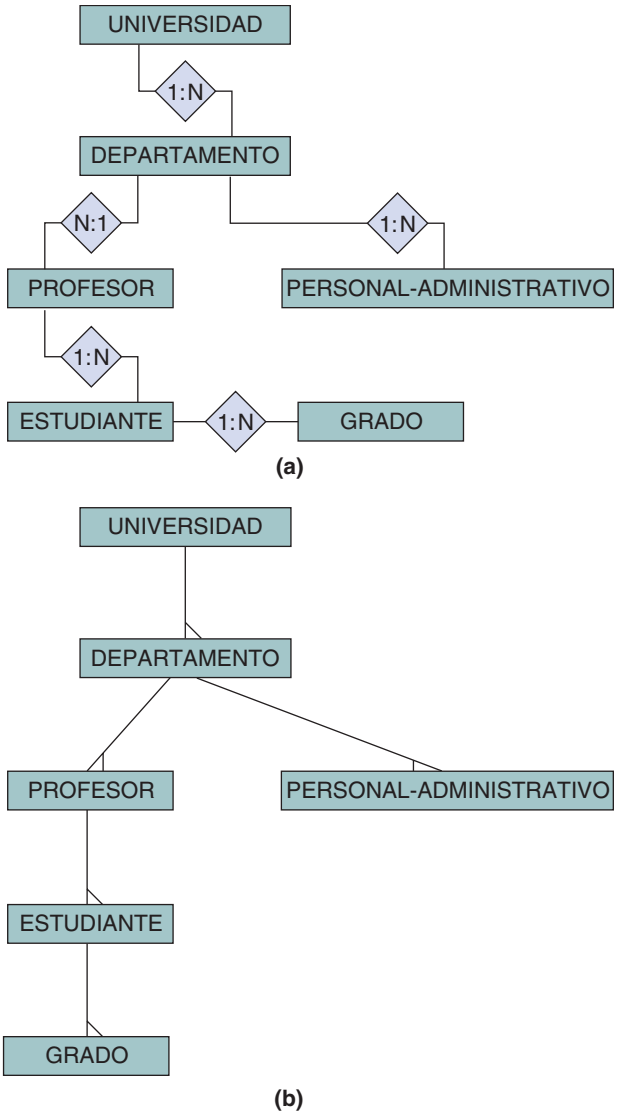
➤ FIGURA 6-21

Ejemplo de un árbol



➤ FIGURA 6-22

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



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

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

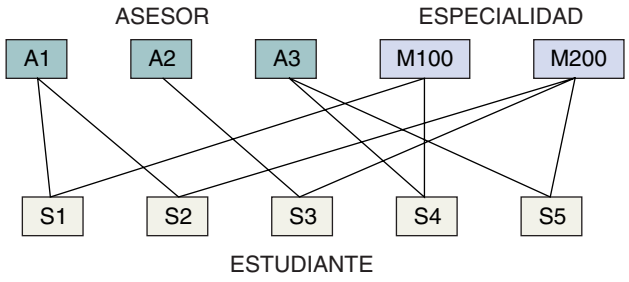
Redes simples

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

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

➤ FIGURA 6-23

Ejemplo de una red simple



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

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

Redes complejas

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

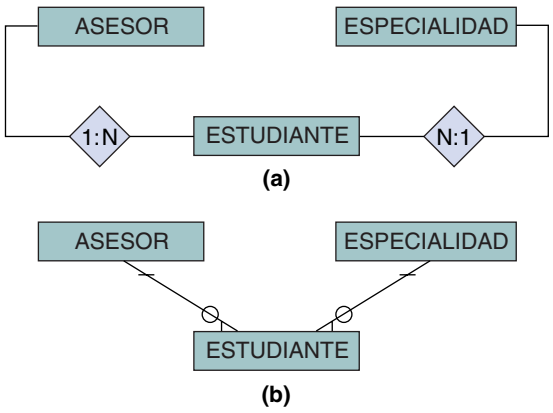
Como analizamos, las relaciones M:N no tienen representación directa en el modelo relacional. En consecuencia, antes de que se pueda almacenar esta estructura en forma relacional debemos definir una relación de intersección. En la figura 6-25(b) la relación de intersección es Proveedor-Parte.

Lista de materiales

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

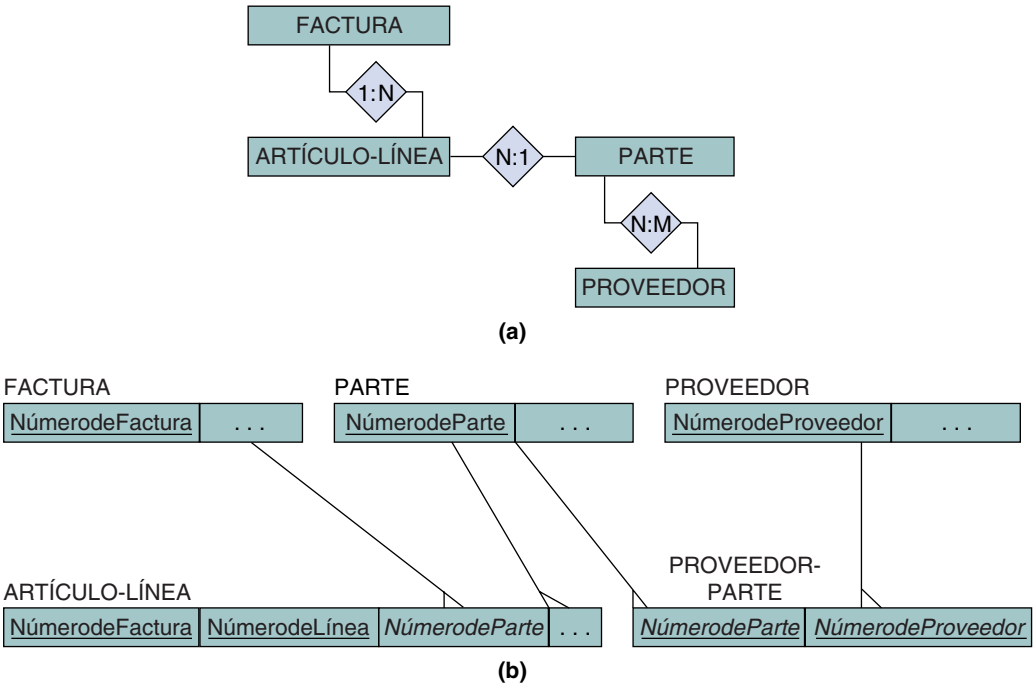
➤ FIGURA 6-24

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



➤ FIGURA 6-25

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

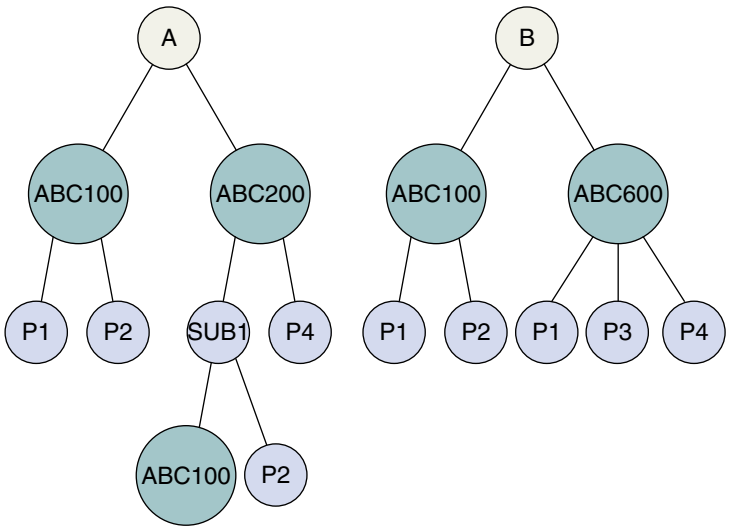


La figura 6-26 es un ejemplo de una lista de materiales, la cual muestra las partes que constituyen los productos. Cuando se observa desde el punto de vista de un producto específico, por ejemplo el producto A, esta estructura de datos es una jerarquía. Debido a que se puede usar una parte en más de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

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

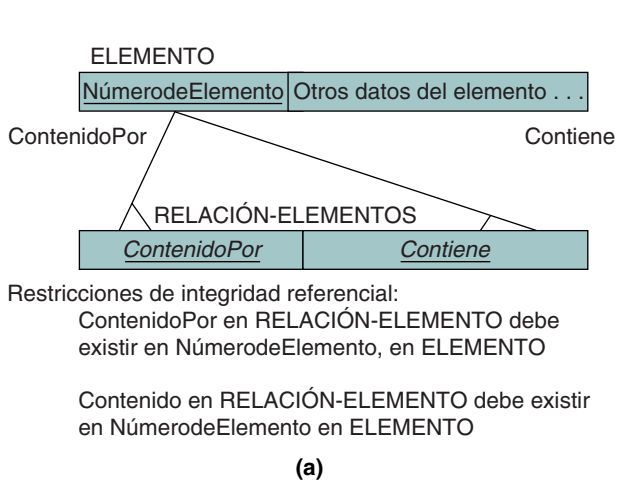
➤ FIGURA 6-26

Ejemplo de una lista de materiales



➤ FIGURA 6-27

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



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

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

(b)

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

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

Llaves sustitutas

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

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

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

MARISCO (NombredelMarisco, Tamaño, Color, Descripción)

y

PLAYA (NombredelaPlaya, Tipo, OrientacióndelaOla, TemperaturadelaCorrientedeAgua)

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

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

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

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

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

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

y

MARISCO_PLAYA_INT (MariscoID, PlayaID)

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

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

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

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

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

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

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

➤ FIGURA 6-28

Ejemplo de llave confusa

Relación ASESOR

Nombre	Departamento
Johnson	Contabilidad
Eldridge	Leyes
Sesiones	Leyes

Relación ESTUDIANTE

Nombre	Especialidad	NombredelAsesor
Franklin	Leyes	Eldridge
Jefferson	Contabilidad	Johnson
Washington	Leyes	Sesiones
Lincoln	Leyes	Sesiones

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

Relación ASESOR

Nombre	Departamento
Johnson	Contabilidad
Eldridge	Leyes
Johnson	Leyes

Relación ESTUDIANTE

Nombre	Especialidad	NombredelAsesor
Franklin	Leyes	Eldridge
Jefferson	Contabilidad	Johnson
Washington	Leyes	Johnson
Lincoln	Leyes	Johnson

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

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

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

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

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

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

➤ FIGURA 6-29

Ejemplo de una llave no confusa con llaves sustitutas

Relación ASESOR

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

Relación ESTUDIANTE

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

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

Relación ASESOR

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

Relación ESTUDIANTE

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

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

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

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

Valores nulos

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

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

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

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

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

➤ RESUMEN

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

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

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

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

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

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

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

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

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

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

➤ PREGUNTAS DEL GRUPO I

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

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

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

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

6.5 Defina *llave externa* y dé un ejemplo.

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

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

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

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

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

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

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

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

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

Ejemplifique sus respuestas usando diagramas de estructura de datos.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.35 Describa tres formas diferentes de evitar los valores nulos.

6.36 ¿Cuándo no representan un problema los valores nulos?

➤ PREGUNTAS DEL GRUPO II

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

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

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

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

➤ PROYECTOS

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

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

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

➤ PREGUNTAS DEL PROYECTO FIREDUP

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

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

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

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

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

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

CAPÍTULO 7

Diseño de bases de datos con modelos de objeto semántico

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

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

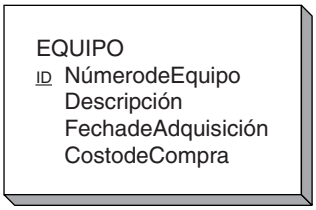
En el capítulo 4 abordamos el modelo de datos de objeto semántico y definimos siete tipos de objetos semánticos. En esta sección presentamos métodos para transformar cada uno de éstos en relaciones. Cuando se trabaja con objetos semánticos, los problemas de normalización son menos comunes que con los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

OBJETOS SIMPLES

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

➤ FIGURA 7-1

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



(a)

EQUIPO (NúmerodeEquipo, Descripción, FechadeAdquisición, CostodeCompra)

(b)

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

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

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

Objetos compuestos

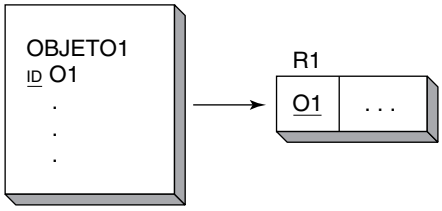
Un objeto compuesto es aquel que tiene uno o más atributos simples multivaluados, o grupo de atributos, pero que no tiene atributos de objeto. La figura 7-3(a) muestra un ejemplo de objeto compuesto: CUENTA-HOTEL. Para representarlo se crea una relación para el objeto base, CUENTA-HOTEL, y otra para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

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

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

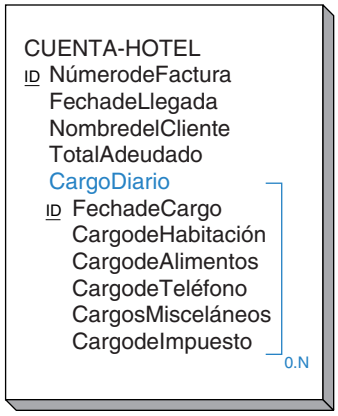
➤ FIGURA 7-2

Transformación general de objeto simple en una relación



➤ FIGURA 7-3

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



(a)

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

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

Restricción de integridad referencial:

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

(b)

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

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

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

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

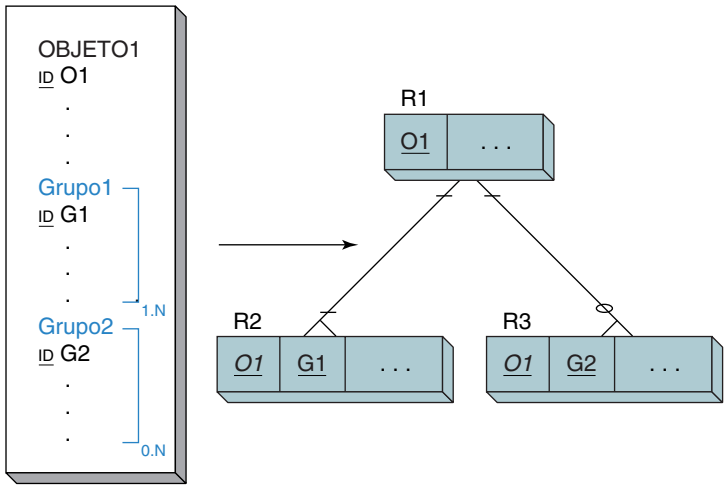
Objetos combinados

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

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

➤ FIGURA 7-4

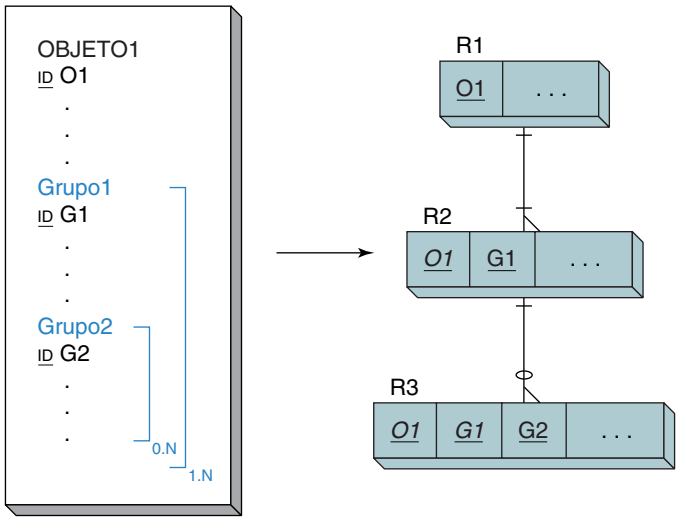
Transformación general de objetos compuestos: (a) objeto compuesto con grupos separados, y (b) objeto compuesto con grupos anidados



Restricciones de integridad referencial:

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

(a)



Restricciones de integridad referencial:

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

(b)

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

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

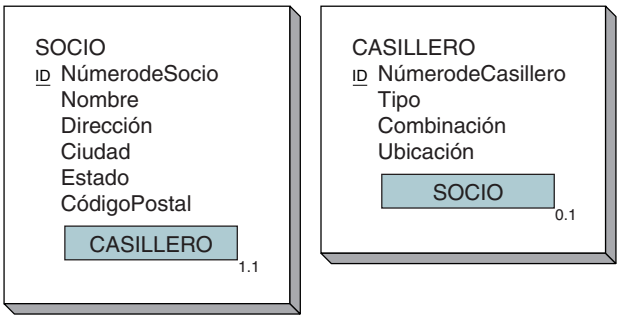
➤ FIGURA 7-5

Cuatro tipos de objetos compuestos

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

➤ FIGURA 7-6

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



(a)

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

CASILLERO (NúmerodeCasillero, Tipo, Combinación, Ubicación)

Restricción de integridad referencial:

NúmerodeCasillero en SOCIO debe existir en
NúmerodeCasillero en CASILLERO

(b)

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

Representación uno a uno de objetos combinados

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

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

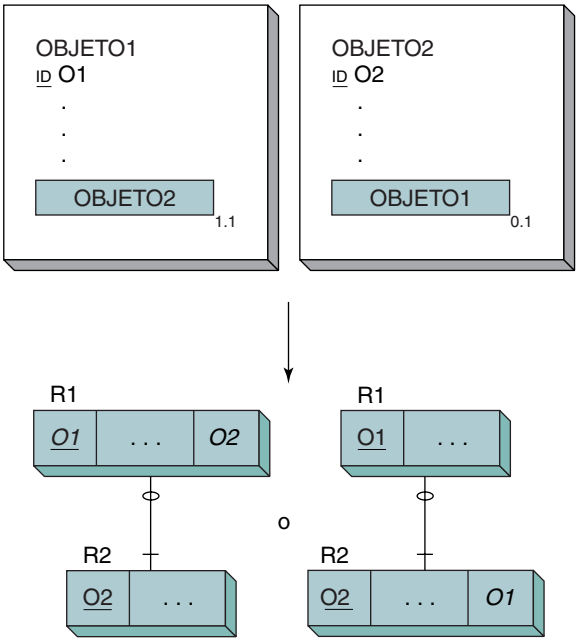
Representación de relaciones uno a muchos y muchos a uno

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

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

➤ FIGURA 7-7

Transformación general de objetos combinados 1:1



Restricción de integridad referencial:

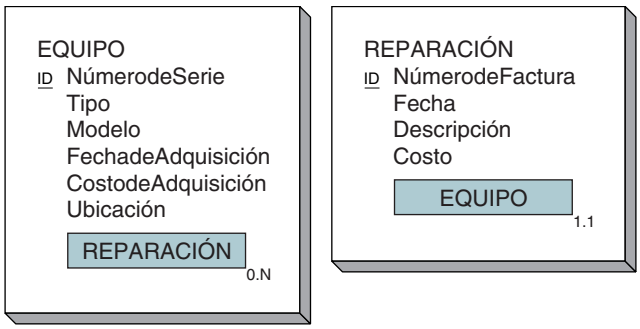
O2 en R1 debe existir en O2 en R2

Restricción de integridad referencial:

O1 en R2 debe existir en O1 en R1

➤ FIGURA 7-8

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



(a)

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

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

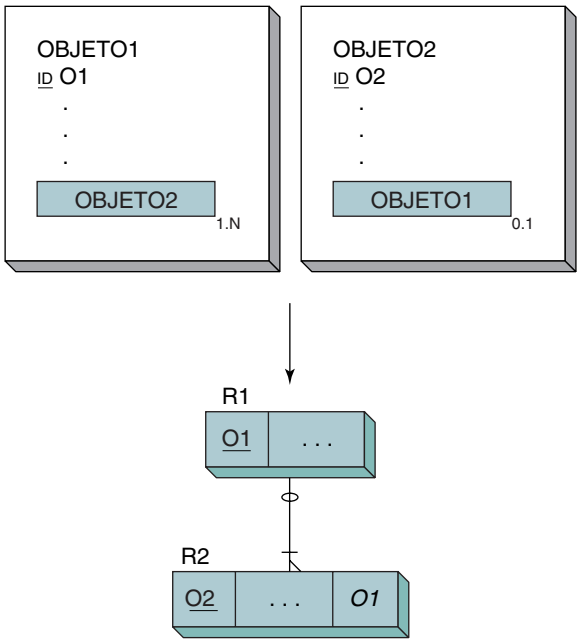
Restricción de integridad referencial:

NúmerodeSerie en REPARACIÓN debe existir en NúmerodeSerie en EQUIPO

(b)

➤ FIGURA 7-9

Transformación general de objetos combinados 1:N



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

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

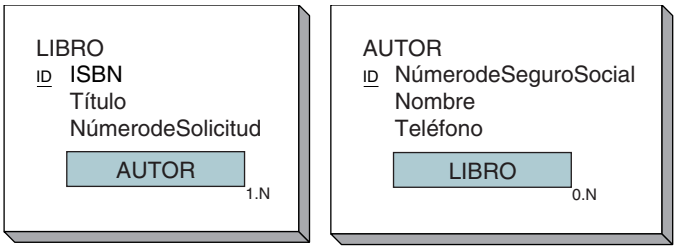
Las cardinalidades mínimas en cualquier caso están determinadas por las cardinalidades mínimas de los atributos del objeto. En la figura 7-9 el OBJETO1 requiere cuando menos un OBJETO2, pero OBJETO2 no necesariamente requiere un OBJETO1. Estas cardinalidades se muestran en el diagrama de estructura de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

Representación de relaciones muchos a muchos

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

➤ FIGURA 7-10

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



(a)

LIBRO (ISBN, Título, NúmerodeSolicitud)

AUTOR (NúmerodeSeguroSocial, Nombre, Teléfono)

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

Restricciones de integridad referencial:

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

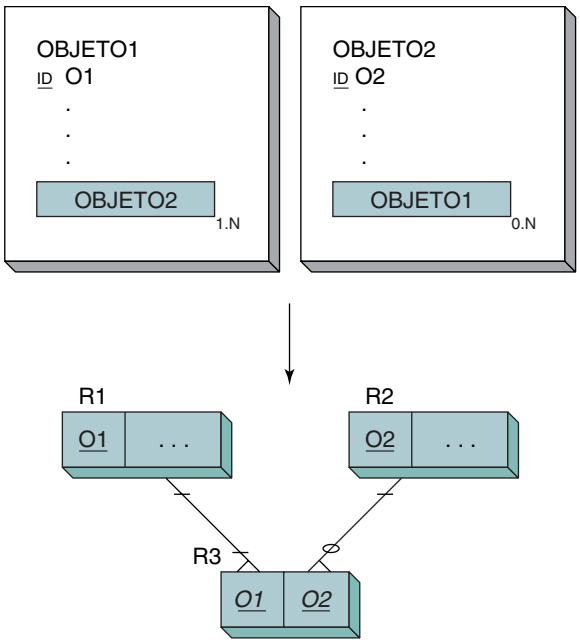
NúmerodeSeguroSocial en LIBRO-AUTOR-INT debe existir en
NúmerodeSeguroSocial en AUTOR

(b)

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

➤ FIGURA 7-11

Transformación general de objetos combinados M:N en relaciones



Restricciones de integridad referencial:

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

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

Objetos híbridos

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

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

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

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

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

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

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

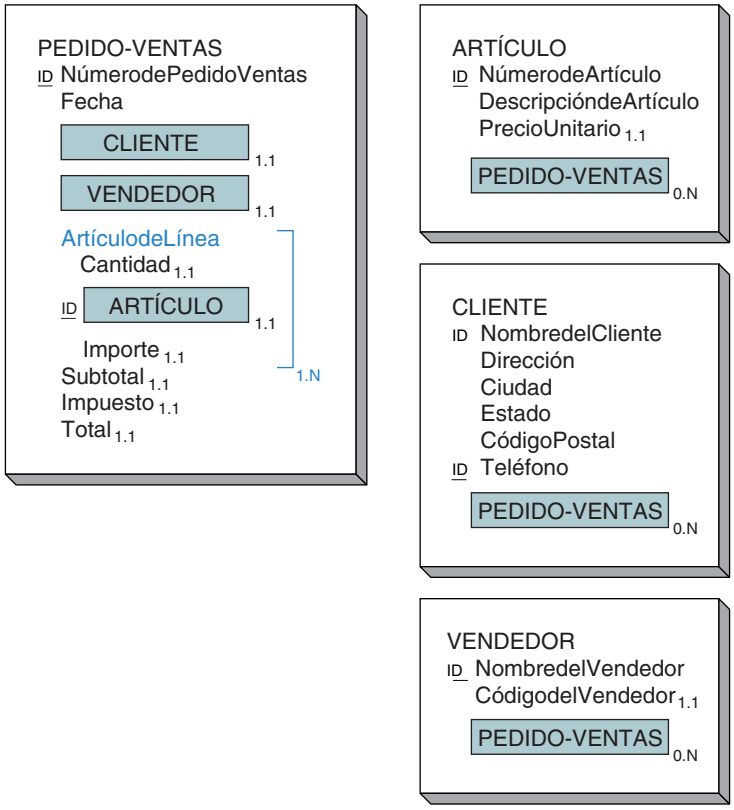
En la figura 7-14 creamos una relación R3 para OBJETO3 y otra, R-G2, para Grupo2. La relación entre R1 y R-G2 es 1:N, así que se coloca la llave de R1 (que es O1) en R-G2. La relación entre R3 y R-G2 es también 1:N, por lo cual se coloca la llave de R3 (que es O3) en R-G2.

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

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

➤ FIGURA 7-12

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



(a)

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

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

VENDEDOR (NombredelVendedor, CódigodelVendedor)

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

ARTÍCULO (NúmerodeArtículo, DescripciónArtículo, PrecioUnitario).

Restricciones de integridad referencial:

NombredelVendedor en PEDIDO-VENTAS debe existir en NombredelVendedor en VENDEDOR

Teléfono en PEDIDO-VENTAS debe existir en Teléfono en CLIENTE

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

NúmerodeArtículo en ARTÍCULO-LÍNEA debe existir en NúmerodeArtículo en ARTÍCULO

(b)

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

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

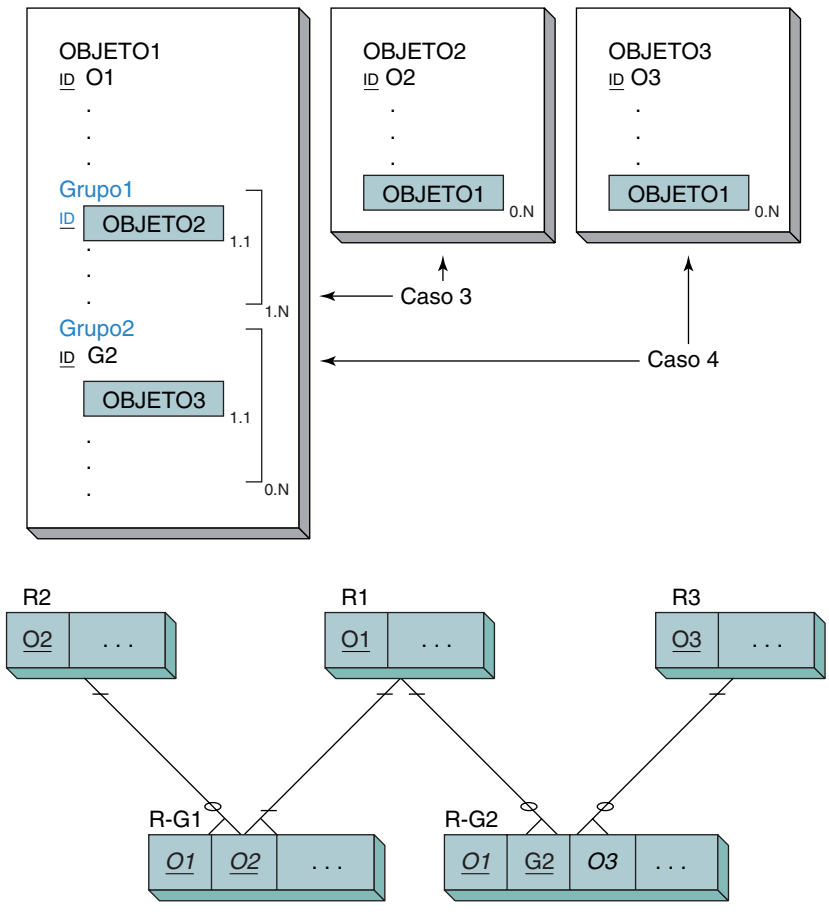
➤ FIGURA 7-13

Cuatro casos de cardinalidad de objetos híbridos

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

➤ FIGURA 7-14

Transformación general de objetos híbridos en relaciones



Restricciones de integridad referencial:

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

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

Objetos de asociación

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

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

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

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

➤ FIGURA 7-15

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

(a)

AVIÓN (NúmerodeAvión, Tipo, Horas)

PILOTO (NúmerodePiloto, Nombre, Teléfono)

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

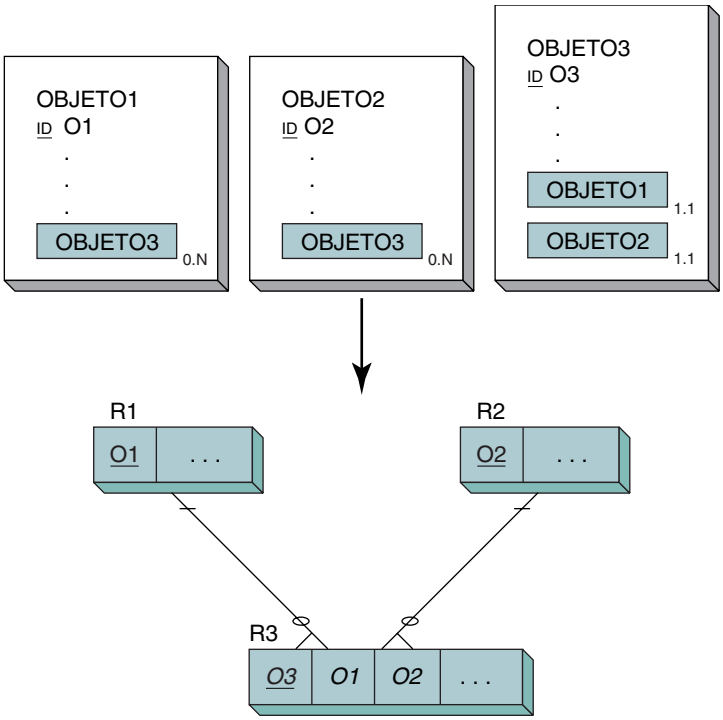
Restricción de integridad referencial:

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

(b)

➤ FIGURA 7-16

Transformación general de los objetos de asociación dentro de las relaciones



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

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

Objetos padre y objetos subtipo

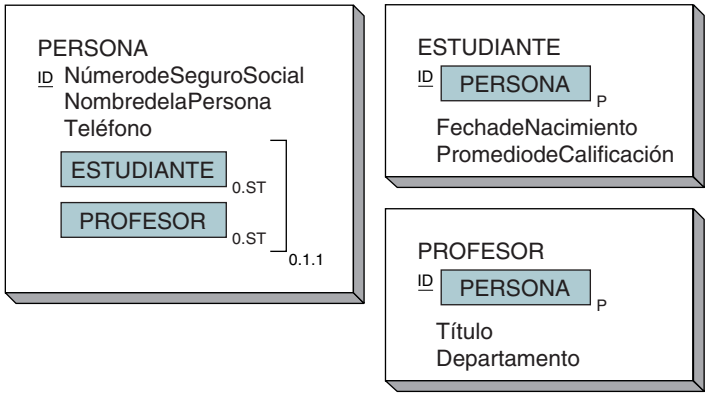
Los objetos padre (también llamados *supertipo*) y los objetos subtipo se representan de manera similar a las entidades padre y subtipo. Definimos una relación por el objeto padre y uno por cada objeto de subtipo. La llave de cada una de estas relaciones es la llave del padre.

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

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

➤ FIGURA 7-17

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



(a)

PERSONA (NúmerodeSeguroSocial, NombredelaPersona, Teléfono)

ESTUDIANTE (NúmerodeSeguroSocial, FechadeNacimiento, PromediodeCalificación)

PROFESOR (NúmerodeSeguroSocial, Título, Departamento)

Restricciones de integridad referencial:

NúmerodeSeguroSocial en ESTUDIANTE debe existir en NúmerodeSeguroSocial en PERSONA

NúmerodeSeguroSocial en PROFESOR debe existir en NúmerodeSeguroSocial en PERSONA

(b)

PERSONA1 (NúmerodeSeguroSocial, NombredelaPersona, Teléfono, TipodePersona)

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

(c)

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

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

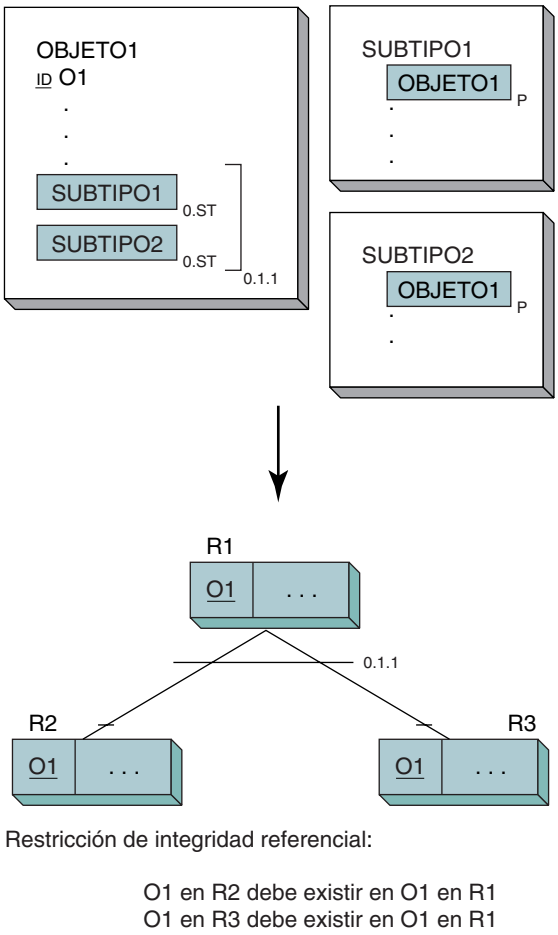
Una segunda posibilidad se aprecia en la relación PERSONA2, a la cual se han agregado dos atributos: uno para TipodeEstudiante y otro para TipodePersona. Cada uno de los atributos es una variable booleana; los valores permitidos son verdaderos o falsos. Observe que, como sucede en este caso, si una persona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

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

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

➤ FIGURA 7-18

Transformación general de los objetos padre-subtipo en relaciones



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

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

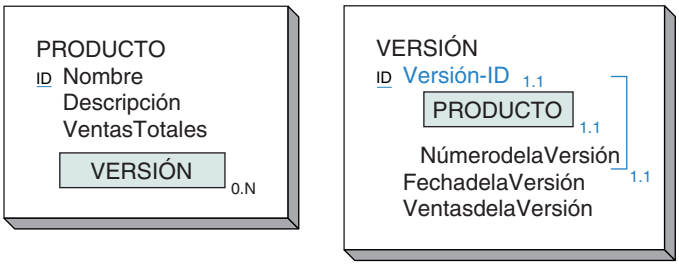
Objetos arquetipo-versión

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

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

➤ FIGURA 7-19

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



(a)

PRODUCTO (Nombre, Descripción, VentasTotales)

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

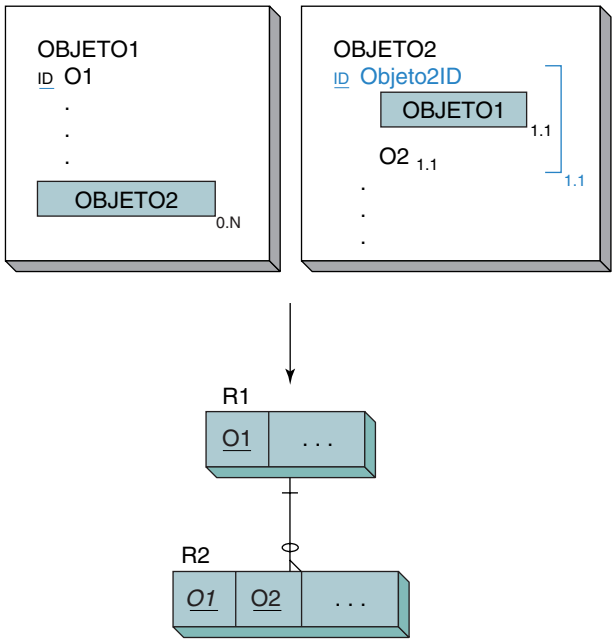
Restricción de integridad referencial:

Nombre en VERSIÓN debe existir en Nombre en PRODUCTO

(b)

➤ FIGURA 7-20

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



Restricción de integridad referencial:

O1 en R2 debe existir en O1 en R2

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

➤ OBJETOS DE MUESTRA

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

Forma de suscripción

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

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

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

Descripción del producto

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

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

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

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

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

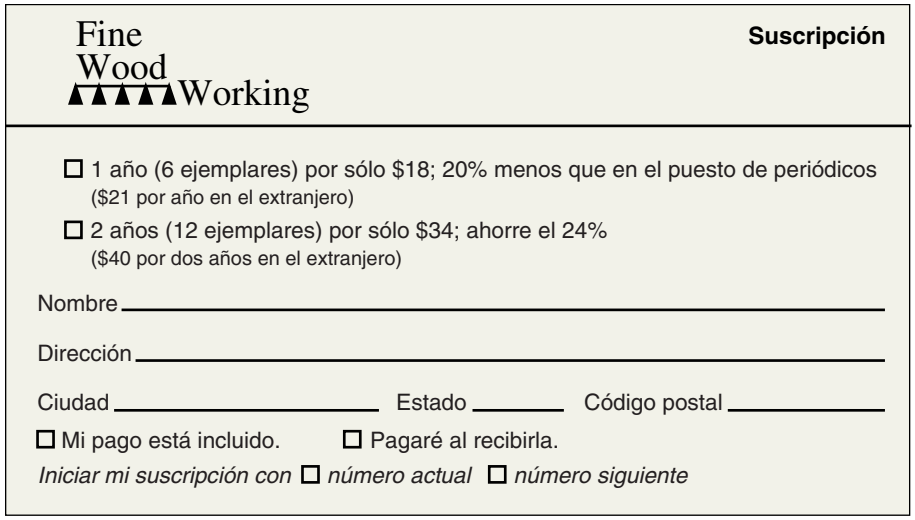
Citatorio de amonestación vial

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

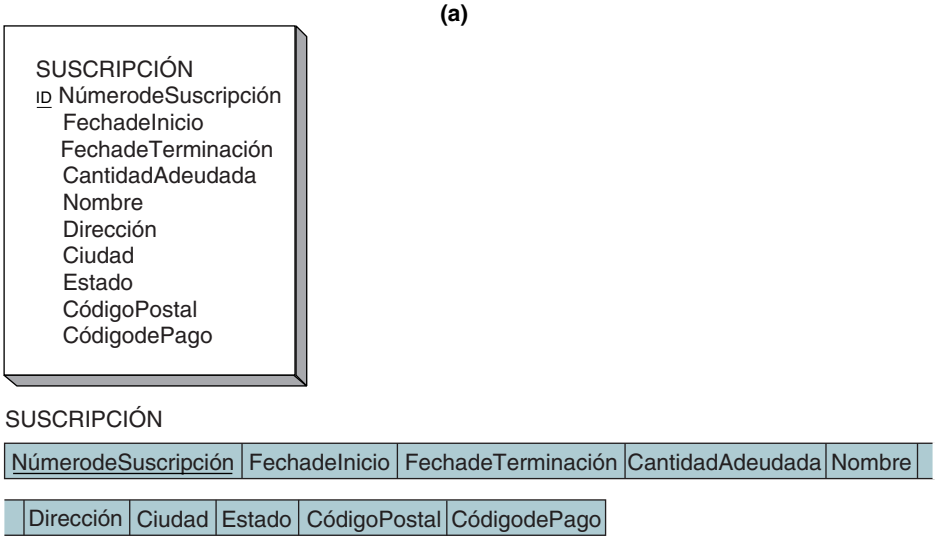
La figura 7-23(b) es una manera de ilustrar los objetos implícitos del citatorio de amonestación vial. Aunque no podemos estar seguros sólo a partir de una forma, hay ciertos indicios que nos llevan a creer que el conductor, el vehículo y el oficial son ob-

➤ FIGURA 7-21

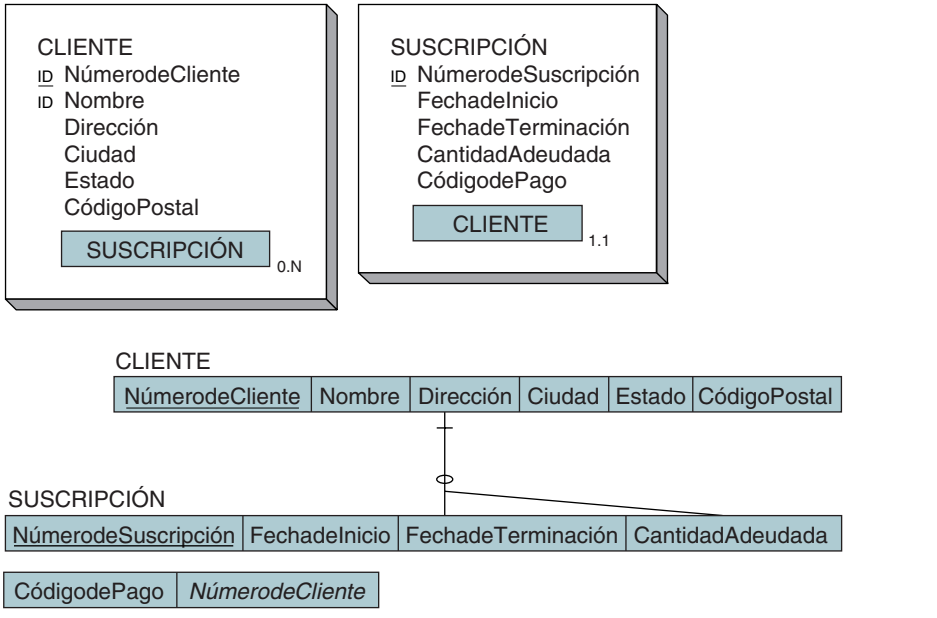
Representación alternativa de suscripción: (a) forma de pedido de suscripción, (b) suscripción modelada como un objeto, y (c) suscripción modelada como dos objetos



(a)



(b)



(c)

➤ FIGURA 7-22

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

Kellogg's®
RICE KRISPIES®

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

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

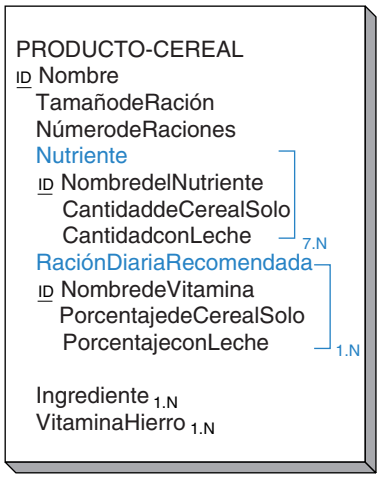
PORCENTAJE DE CANTIDADES DIARIAS QUE RECOMIENDA EL MINISTERIO DE AGRICULTURA DE ESTADOS UNIDOS DE AMÉRICA

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

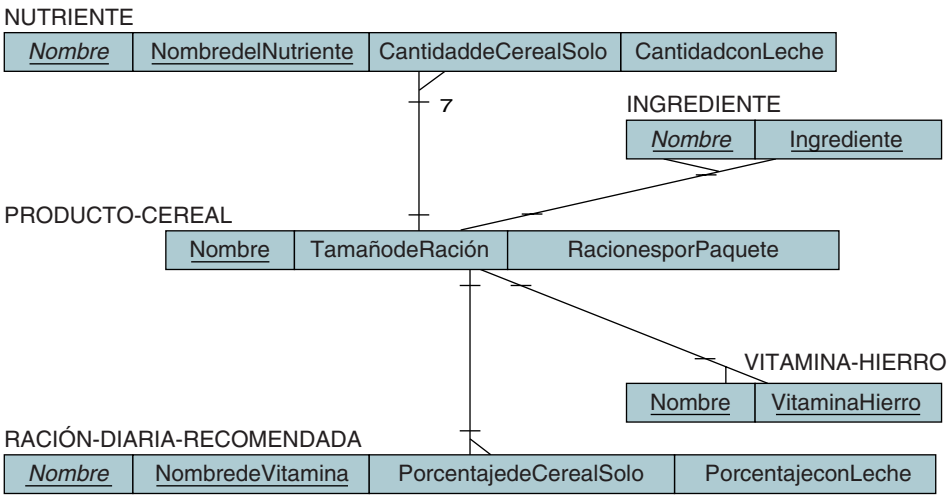
* LA LECHE ENTERA PROPORCIONA 30 CALORÍAS MÁS, 4 g DE GRASA Y 15 mg DE COLESTEROL.
** CONTIENE MENOS DEL 2% DE LA RACIÓN DIARIA RECOMENDADA.

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

(a)



(b)



(c)

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

➤ FIGURA 7-23

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


AVISO DE INFRACCIÓN. PATRULLA DEL ESTADO DE WASHINGTON

NOMBRE Kroenke (APELLIDO) David M (NOMBRE)

DIRECCIÓN 5053 88 Ave SE

CIUDAD Mecer Island ESTADO Wa CÓDIGO POSTAL 98040

LICENCIA DE CONDUCIR 00000 | ESTADO Wa | M F | FECHA DE NACIMIENTO 2/27 46 | ESTATURA 6 | PESO 165 | OJOS Cafés

NÚMERO DE MOTOR AAA000 | ESTADO Wa | COLOR | AÑO 90 | MARCA Saab | TIPO 900

NIV

REGISTRO

PROPIETARIO

DIRECCION

FECHA DE LA INFRACCIÓN MES 11 DÍA 7 AÑO 2001 HORA: 935 | DISTRITO 2 | DESTACAMENTO 17

LUGAR 17 MILLAS E DE Enumclaw SOBRE SR410

VIOLACIONES

Escribía mientras manejaba

FIRMA DEL OFICIAL S Scott NÚMERO DE PERSONAL 850

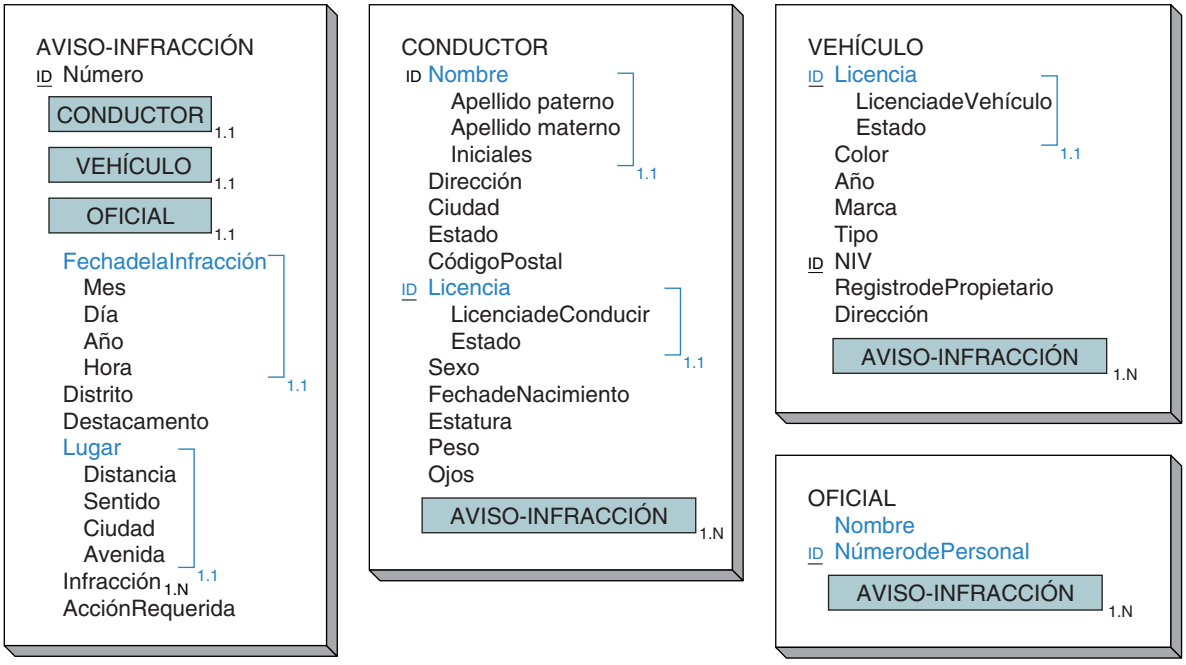
[X] Ésta es una amonestación; no se requieren acciones posteriores.

[ ] Se autoriza el traslado para reparar el vehículo. No se autoriza la reparación en esta vía.

[ ] CORRIJA LA(S) VIOLACIÓN(ES) DE INMEDIATO. Regrese firmada esta tarjeta como prueba de cumplimiento dentro de 15-30 días (si se marca este recuadro).

X FIRMA DEL CONDUCTOR

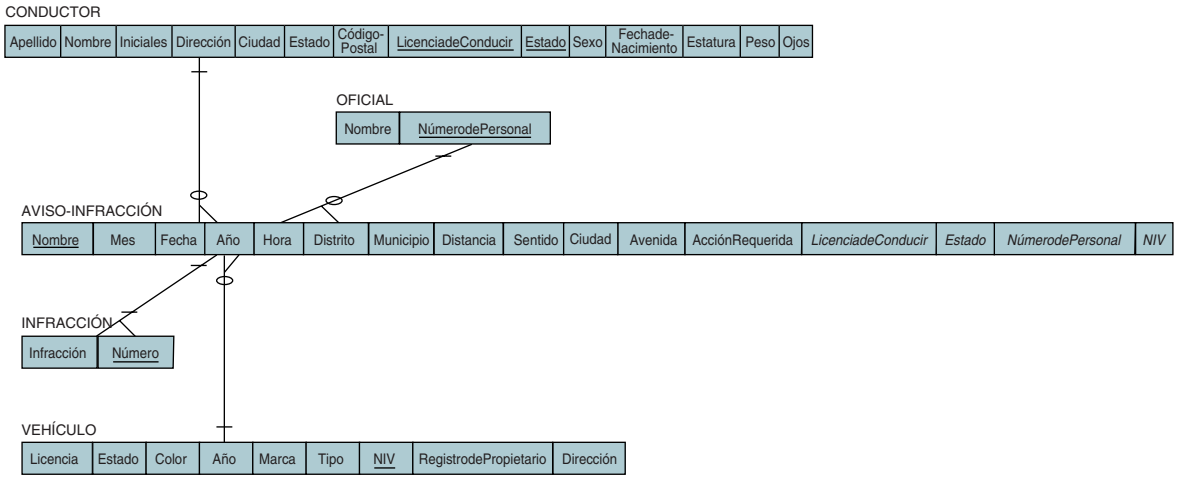
(a)



(b)

FIGURA 7-23

(Continuación)



(c)

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

RESUMEN

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

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

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

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

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

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

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

➤ PREGUNTAS DEL GRUPO I

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

➤ PREGUNTAS DEL GRUPO II

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

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

➤ FIGURA 7-24

Reportes para la pregunta 7.25

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

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

(a)

LISTA DE PROVEEDORES
Fecha: 06/30/2001

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

(b)

➤ FIGURA 7-25

Reporte para la pregunta 7.26

Amor sin barreras
Basada en una idea de Jerome Robbins

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

Dirección y coreografía de la producción original por JEROME ROBBINS

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

PUNTOS CULMINANTES DE TODA LA GRABACIÓN

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

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

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

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

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

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

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

d. Transforme los diagramas de objetos en diagramas de relaciones

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

➤ PROYECTOS

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

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

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

➤ PREGUNTAS DEL PROYECTO FIREDUP

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

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

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

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

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

PARTE IV

IMPLEMENTACIÓN DE BASES DE DATOS CON EL MODELO RELACIONAL

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

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

CAPÍTULO 8

Fundamentos de la implementación relacional

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

➤ DEFINICIÓN DE DATOS RELACIONALES

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

Revisión de terminología

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

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

➤ FIGURA 8-1

Ejemplo de una estructura de la relación PACIENTE

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

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

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

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

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

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

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

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

➤ FIGURA 8-2

Resumen de terminología relacional

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

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

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

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

Implementación de una base de datos relacional

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

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

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

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

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

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

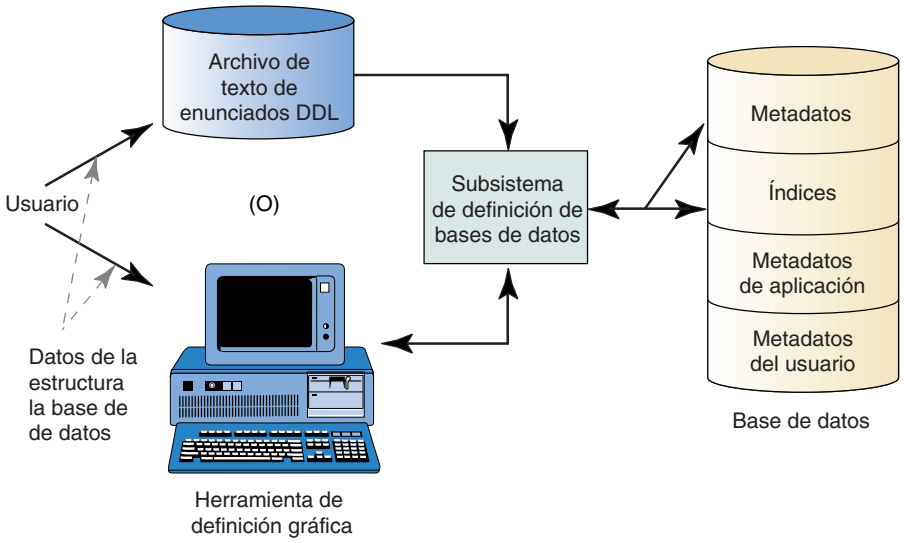
➤ FIGURA 8-3

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

```
CREATE SCHEMA MÉDICO

CREATE TABLE PACIENTE

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

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

CREATE TABLE MÉDICO

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

  PRIMARY KEY (NombredelMédico)
 )
```

➤ FIGURA 8-4

Proceso para la definición de la base de datos



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

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

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

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

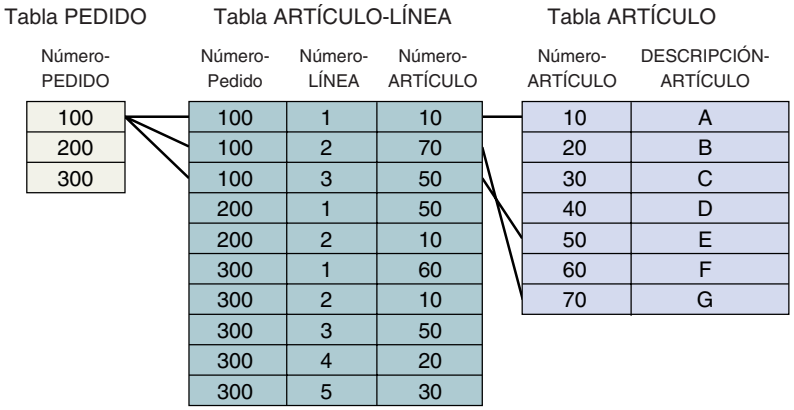
Ahora suponga que una organización procesa en conjunto muchos pedidos y tiene un disco grande y rápido y otro pequeño y lento. El programador debe determinar cuál es el mejor lugar para colocar los datos. Una posibilidad es que el desempeño mejore si la tabla ARTÍCULO se almacena en el disco más grande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

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

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

➤ FIGURA 8-5

Datos de ejemplo de tres tablas que representan un PEDIDO



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

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

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

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

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

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

➤ MANEJO DE DATOS RELACIONALES

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

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

Categorías de lenguajes de manipulación de datos relacionales

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

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

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

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

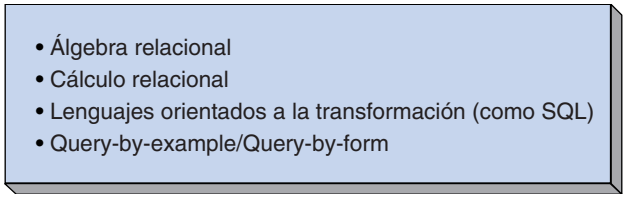
Aunque el cálculo relacional es difícil de entender y usar, su característica respecto a que no está basado en procedimientos es muy conveniente. Por lo tanto, los diseñadores DBMS buscaron otras técnicas no basadas en procedimientos, lo cual condujo a la tercera y cuarta categorías de DML relacionales.

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

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

➤ FIGURA 8-6

Cuatro categorías de DML relacional



➤ FIGURA 8-7

Ejemplo de una forma de pantalla tabular predeterminada

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

Interfaces DML con el DBMS

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

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

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

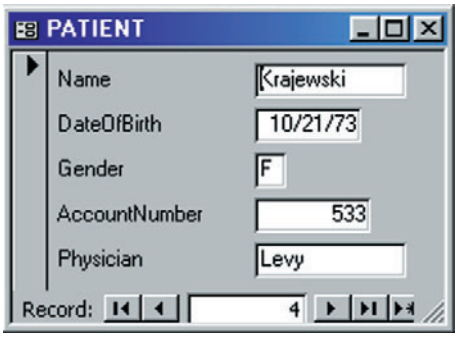
INTERFAZ DE LENGUAJE CONSULTA-ACTUALIZACIÓN. El segundo tipo de interfaz para una base de datos es a través de un **lenguaje de consulta-actualización,** o simplemente **lenguaje de consulta** (aunque la mayoría de estos lenguajes realizan tanto la consulta como la actualización, por lo general se les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

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

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

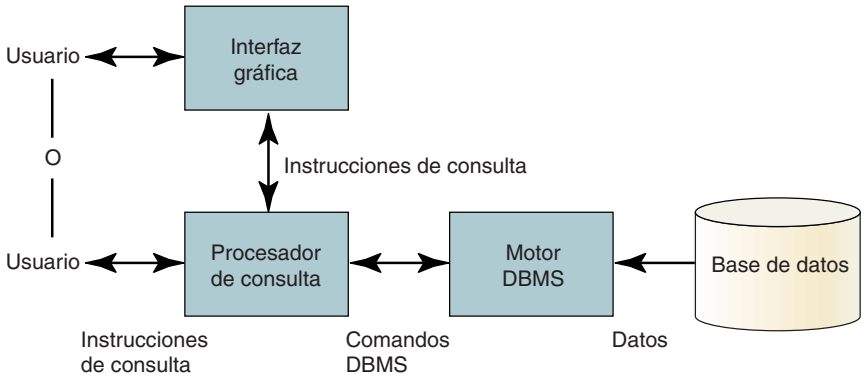
➤ FIGURA 8-8

Ejemplo de una forma de pantalla de un renglón individual predeterminado



➤ FIGURA 8-9

Programas involucrados en el procesamiento de consultas



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

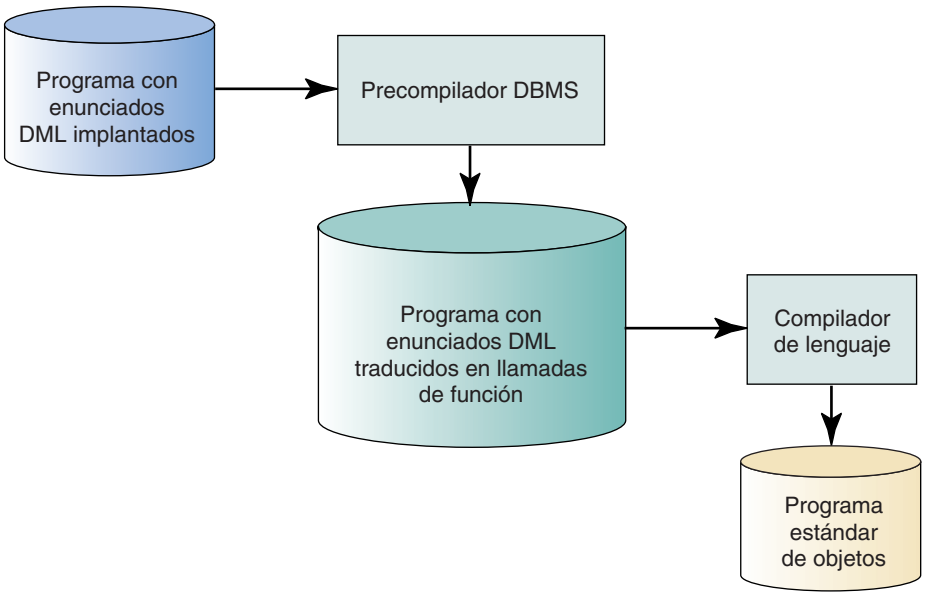


FIGURA 8-10

Programa de procesamiento con enunciados DML implantados

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

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

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

➤ ÁLGEBRA RELACIONAL

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

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

OPERADORES RELACIONALES

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

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

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

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

➤ FIGURA 8-11

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

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

(a)

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

(b)

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

(c)

➤ FIGURA 8-12

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

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

(a)

Número	Nombre	Interés
105	ANDERSON	ADMINISTRACIÓN
123	JONES	HISTORIA

(b)

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

(c)

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

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

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

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

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

➤ FIGURA 8-13

Relación JUNIOR menos ESTUDIANTE-HONOR

Enum	Nombre	Especialidad
158	PARKS	MATEMÁTICAS
271	SMITH	HISTORIA

➤ FIGURA 8-14

Relaciones de intersección de JUNIOR y ESTUDIANTE-HONOR

Enum o Número	Nombre	Especialidad o Interés
123	JONES	HISTORIA

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

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

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

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

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

➤ FIGURA 8-15

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

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

(a)

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

(b)

➤ FIGURA 8-16

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

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

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

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

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

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

➤ FIGURA 8-17

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

Nombre	Especialidad
JONES	HISTORIA
PARKS	MATEMÁTICAS
ANDERSON	ADMINISTRACIÓN
SMITH	HISTORIA

(a)

Especialidad	Grado
HISTORIA	JR
MATEMÁTICAS	GR
ADMINISTRACIÓN	SN

(b)

➤ FIGURA 8-18

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

EID	Nombre	Especialidad	Grado
158	PARKS	MATEMÁTICAS	GR

(a)

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

(b)

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

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

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

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

➤ FIGURA 8-19

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

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

(a)

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

(b)

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

(c)

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

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

Expresión de consultas en álgebra relacional

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

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

➤ FIGURA 8-20

Resumen de operaciones de álgebra relacional

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

➤ FIGURA 8-21

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

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

(a)

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

(b)

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

(c)

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

   ESTUDIANTE [Nombre]

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

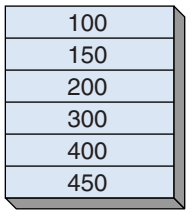
JONES
PARKS
BAKER
GLASS
RUSSELL
RYE

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

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

   INSCRIPCIÓN [NúmerodeEstudiante]

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

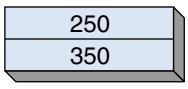


   Otra vez se omitieron los tuples duplicados.

3. ¿Cuáles son los números de estudiante de todos los que no se han inscrito en una clase?

   ESTUDIANTE[EID] - INSCRIPCIÓN [NúmerodeEstudiante]

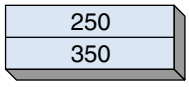
   Esta expresión encuentra la diferencia en la proyección de las dos relaciones: ESTUDIANTE [EID] tiene los números de todos los estudiantes, e INSCRIPCIÓN [NúmerodeEstudiante] tiene los números de todos que están inscritos en una clase. La diferencia es el número de los que no están inscritos. El resultado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



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

   INSCRIPCIÓN WHERE NombredeClase = 'BD445' [NúmerodeEstudiante]

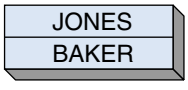
   Esta expresión selecciona los tuples apropiados y luego los proyecta en el atributo NúmerodeEstudiante. El resultado es



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

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

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



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

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

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

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

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

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

BA200	M-F9

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

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

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

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

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

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

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

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

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

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

➤ RESUMEN

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

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

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

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

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

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

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

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

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

➤ PREGUNTAS DEL GRUPO I

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

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

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

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

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

8.6 Mencione tres razones para usar índices.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

8.24 Mencione un ejemplo del producto de VENDEDOR y PEDIDO.

8.25 Muestre un ejemplo de:

VENDEDOR [Nombre, Salario]
VENDEDOR [Salario]

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

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

➤ FIGURA 8-22

Datos de muestra para las preguntas 8.24 a 8.28

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

VENDEDOR

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

PEDIDO

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

CLIENTE

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

8.28 Muestre las expresiones del álgebra relacional para

a. Los nombres de todos los vendedores

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

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

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

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

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

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

CAPÍTULO 9

Lenguaje de consulta estructurado

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

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

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

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

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

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

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

➤ CONSULTA DE UNA SOLA TABLA

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

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

➤ FIGURA 9-1

Relaciones usadas para los ejemplos SQL

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

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

➤ FIGURA 9-2

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

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

(a)

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

(b)

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

(c)

Proyecciones utilizando SQL

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

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

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

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

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

Considere otro ejemplo:

```
SELECT      Especialidad
FROM        ESTUDIANTE
```

El resultado de esta operación es el siguiente:

HISTORIA
CONTABILIDAD
MATEMÁTICAS
HISTORIA
CONTABILIDAD
MATEMÁTICAS
CONTABILIDAD
HISTORIA

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

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

```
SELECT      DISTINCT Especialidad
FROM        ESTUDIANTE
```

El resultado de esta operación es la relación:

HISTORIA
CONTABILIDAD
MATEMÁTICAS

Selecciones usando SQL

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

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

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

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

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

200	BAKER	MATEMÁTICAS	GR
350	RUSSELL	MATEMÁTICAS	JR

Podemos combinar la selección y la proyección como sigue:

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

El resultado es:

BAKER	GR
RUSSELL	JR

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

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

se obtiene lo siguiente:

BAKER	GR

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

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

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

PARKS
BAKER
BAKER
RUSSELL
RYE

La expresión:

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

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

JONES
GLASS
JONES

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

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

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

se obtendrá el siguiente resultado:

BAKER	MATEMÁTICAS
GLASS	HISTORIA
BAKER	CONTABILIDAD

Esta expresión equivale a:

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

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

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

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

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

JONES	GR
BAKER	GR
RUSSELL	JR
RYE	FR

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

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

el resultado es:

JONES
PARKS
GLASS
JONES

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

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

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

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

Ordenamiento

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

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

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

BAKER	CONTABILIDAD	SN
PARKS	CONTABILIDAD	SO
RYE	CONTABILIDAD	FR

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

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

El resultado es:

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

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

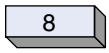
Funciones SQL interconstruidas

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

La expresión de consulta:

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

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



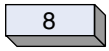
Considere las expresiones:

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

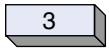
y

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

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



y



respectivamente.

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

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

no está permitido.

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

Funciones interconstruidas y de agrupamiento

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

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

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

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

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

El resultado de esta expresión es:

HISTORIA	3
CONTABILIDAD	3
MATEMÁTICAS	2

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

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

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

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

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

HISTORIA	3
CONTABILIDAD	3

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

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

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

HISTORIA	450

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

➤ CONSULTAS DE TABLAS MÚLTIPLES

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

RECUPERACIÓN USANDO UNA SUBCONSULTA

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

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

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

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

El resultado de esta operación es:

100
200

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

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

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

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

JONES
BAKER

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

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

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

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

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

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

Esto da como resultado:

100
200
300

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

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

El resultado es:

JONES
BAKER
BAKER

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

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

JOIN CON SQL

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

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

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

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

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

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

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

400	BF410

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

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

El resultado de esta operación es:

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

Comparación del join y de la subconsulta SQL

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

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

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

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

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

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

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

o como un join:

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

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

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

El resultado es:

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

Outer join

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

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

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

El resultado es:

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

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

➤ EXISTS Y NOT EXISTS

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

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

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

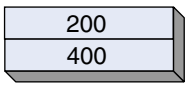
El significado de la expresión subconsulta es: Encontrar dos renglones en INSCRIPCIÓN que tengan el mismo número de estudiante, pero diferentes nombres de clases (lo cual significa que el estudiante está tomando más de una clase). Si existen dos de estos renglones, entonces el valor lógico de EXISTS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

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

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

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

El resultado de esta consulta es:



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

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

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

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

➤ CAMBIO DE DATOS

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

Inserción de datos

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

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

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

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

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

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

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

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

Eliminación de datos

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

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

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

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

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

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

Modificación de datos

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

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

y

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

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

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

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

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

➤ RESUMEN

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

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

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

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

➤ PREGUNTAS DEL GRUPO I

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

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

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

9.1 Muestre los salarios de todos los vendedores.

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

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

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

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

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

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

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

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

9.10 Calcule el número de pedidos.

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

9.12 Calcule el porcentaje promedio de cuota para los vendedores.

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

9.14 Calcule el número de pedidos de cada vendedor.

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

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

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

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

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

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

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

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

➤ FIGURA 9-3

Datos de muestra para las preguntas del grupo 1

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

VENDEDOR

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

PEDIDO

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

CLIENTE

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

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

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

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

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

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

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

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

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

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

➤ PREGUNTAS DEL GRUPO II

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

a. Liste todas las columnas de proveedores

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

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

d. Liste NiveldeReordenamiento y cuente todos los productos

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

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

➤ PREGUNTAS DEL PROYECTO FIREDUP

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

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

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

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

B. Liste las versiones de todas las estufas.

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

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

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

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

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

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

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

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

K. Cuente todas las estufas.

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

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

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

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

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

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

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

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

CAPÍTULO 10

Diseño de aplicaciones de bases de datos

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

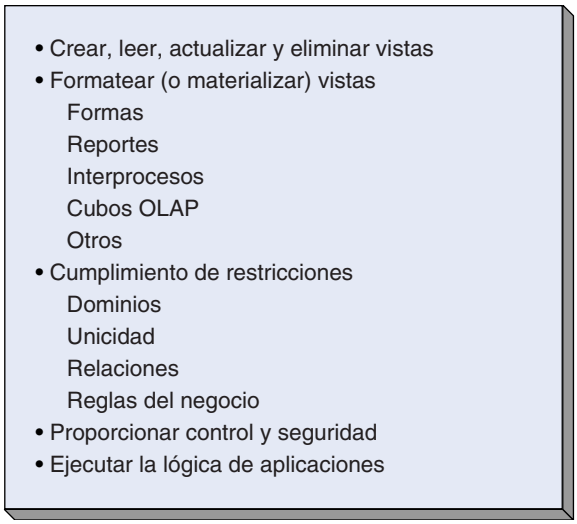
Antes de comenzar conviene una observación acerca de la terminología. En las primeras etapas del procesamiento de bases de datos era muy fácil encontrar un nexo entre el DBMS y la aplicación —las aplicaciones eran programas independientes que invocaban al DBMS—. Hoy, especialmente con los productos de escritorio del DBMS tales como Microsoft Access, esa conexión se ha vuelto confusa. Para evitar malos entendidos en este capítulo, supondremos que todas las formas, reportes, menúes y cualquier código de programa que esté contenido en cualesquiera de ellos, son parte de la aplicación de la base de datos. Además, los programas que llaman al DBMS son también parte de la aplicación. Cualquier estructura, regla o restricción que sea parte de la tabla, así como las definiciones de la relación, las maneja el DBMS y son parte de la base de datos. Así, una regla que se coloca en determinada columna de una tabla durante la definición de ésta se vuelve obligatoria mediante el DBMS. La misma regla, puesta en una forma de control, se considera obligatoria mediante la aplicación de la base de datos. A medida que avancemos podremos ver la importancia de esta diferencia.

➤ FUNCIONES DE UNA APLICACIÓN DE BASE DE DATOS

La figura 10-1 lista las funciones de una aplicación de base de datos. La primera es procesar vistas de datos. Hay cuatro funciones de procesamiento básico: crear, leer, actualizar y eliminar. En inglés a veces a estas funciones se les denomina mediante el acrónimo **CRUD.** Así, la primera tarea de una aplicación de base de datos es para vistas CRUD.

➤ FIGURA 10-1

Funciones de una aplicación de base de datos



Una vista de aplicación es más que un renglón de una tabla y más que el resultado de un enunciado SQL. Con frecuencia la construcción de una vista de aplicación requiere dos o más enunciados SQL, como se verá en la siguiente sección.

Una segunda tarea de la aplicación es **formatear** o **materializar** las vistas que están siendo procesadas. Observe que existe una diferencia entre los datos contenidos (la vista) y la apariencia de ese contenido (el formato o materialización). Una vista determinada por lo general tiene diferentes formatos. La distinción entre contenido y forma es especialmente importante en las aplicaciones de bases de datos que usan la tecnología Internet.

Hay varios tipos de materialización de vistas. Las formas y reportes son los dos más comunes; sin embargo, otros también son importantes. Una materialización del interproceso se usa para enviar una vista de un servidor a otro, o de una aplicación a otra. El formato de tales materializaciones se determina mediante una interfaz estándar como Microsoft COM, o un protocolo como XML, que presentaremos en el capítulo 14. El intercambio electrónico de datos es un buen ejemplo de su uso. Otros tipos de materialización son más especializadas. En el capítulo 17 verá el papel que desempeñan los cubos OLAP. El lenguaje natural es otro tipo de ejemplo.

El cumplimiento de restricciones es una tercera función. Éstas pueden ser estructurales, por ejemplo que los datos de valores requeridos se ajusten a las especificaciones de dominio, con lo que se asegura la unicidad y se imponen restricciones a la relación. Otras restricciones implican reglas del negocio, tales como: "Ningún vendedor puede tratar con ningún cliente cuya dirección asentada en el contrato esté fuera de la región que tiene asignada."

La cuarta función de una aplicación de bases de datos es proporcionar mecanismos de seguridad y control. La aplicación funcionará en conjunto con el sistema operativo y el DBMS para aumentar la seguridad que proporcionan los nombres de los usuarios y las contraseñas (passwords). Los menúes y estructuras similares determinan qué y cuándo pueden llevar a cabo determinadas acciones los usuarios.

La última tarea de una aplicación de base de datos es ejecutar la lógica del negocio. Por ejemplo, en una aplicación de captura de pedidos, cuando un cliente ordena cinco ejemplares de un libro, la aplicación necesita reducir automáticamente la cantidad de libros en el almacén. Si no hay ejemplares suficientes en inventario, o si la cantidad disponible es menor al punto de reorden, también se necesita llevar a cabo otra acción.

Las últimas dos funciones se abordan detalladamente en las clases de desarrollo de sistemas, por lo tanto casi no las abordaremos. Sin embargo, las primeras tres funciones son específicas de las aplicaciones de base de datos, y por ende centraremos nuestra atención en ellas.

Antes de aprender mayores detalles acerca de dichas funciones considere las necesidades para una base de datos y la aplicación con respecto a la galería View Ridge.

➤ CASO DE APLICACIÓN: GALERÍA VIEW RIDGE

La galería View Ridge es una pequeña empresa que vende arte contemporáneo, incluyendo litografías, pinturas originales y fotografías. Todas las litografías y fotografías están firmadas y numeradas y la mayoría de las piezas cuestan entre $1000 y $25000. View Ridge ha estado en el negocio durante 27 años; tiene un solo propietario que le dedica tiempo completo, así como tres vendedores y dos trabajadores que elaboran marcos, cuelgan los cuadros en la galería y preparan las obras de arte para su envío.

View Ridge realiza exposiciones y otros eventos con el fin de atraer clientes. Las obras de arte también se exhiben en compañías locales, restaurantes y otros lugares públicos. View Ridge es dueña de todo el material que vende; no tiene obras de arte a consignación.

Requerimientos para la aplicación

Los requerimientos para la aplicación View Ridge se resumen en la figura 10-2. En primer lugar, tanto el dueño como los vendedores desean dar seguimiento a sus clientes, así como a los intereses de éstos en cuanto a la compra de arte. Los vendedores necesitan saber a quién contactar cuando llega nuevo material, y tener toda esa información para crear cartas personalizadas y comunicarse verbalmente con sus clientes.

Además, la base de datos debe registrar las compras de arte de los clientes para que los vendedores dediquen más tiempo a los compradores más activos. En ocasiones también necesitan usar los registros de adquisiciones para ubicar las obras de arte, porque a veces la galería compra nuevamente algunas obras que son difíciles de encontrar y las revende. La aplicación de la base de datos debe tener un formato para agregar las nuevas obras que compra la galería.

View Ridge quiere que su aplicación de base de datos proporcione una lista de los artistas y los trabajos que se han exhibido en la galería. Al dueño también le gustaría poder determinar con qué rapidez se vende el trabajo de un artista y cuáles son los márgenes de utilidad, y la aplicación de base de datos debe mostrar un inventario actual en una página Web a la que los clientes puedan tener acceso a través de Internet. Por último, la galería View Ridge quiere que la base de datos produzca reportes que aminoren la carga de trabajo del contador de tiempo parcial que está contratado.

Diseño de la base de datos

La figura 10-3(a) muestra la estructura de los objetos que se requieren para manejar la base de datos de la galería. CLIENTE (CUSTOMER) y ARTISTA (ARTIST) son objetos compuestos y TRABAJO (WORK) es un objeto híbrido cuyo único identificador es el grupo {ARTISTA, Título y Copia} {ARTIST, Title y Copy}. El grupo multivaluado Transacción (Transaction) representa las adquisiciones y las ventas. Considerando que determinada obra puede pasar por la galería varias veces (debido a las adquisiciones y a

➤ FIGURA 10-2

Resumen de los requerimientos para las aplicaciones de la base de datos de la galería View Ridge

Dar seguimiento a los clientes y a sus intereses de compra.

Registrar las compras de material artístico de los clientes.

Registrar las compras.

Listar los artistas y las obras que se han exhibido en la galería.

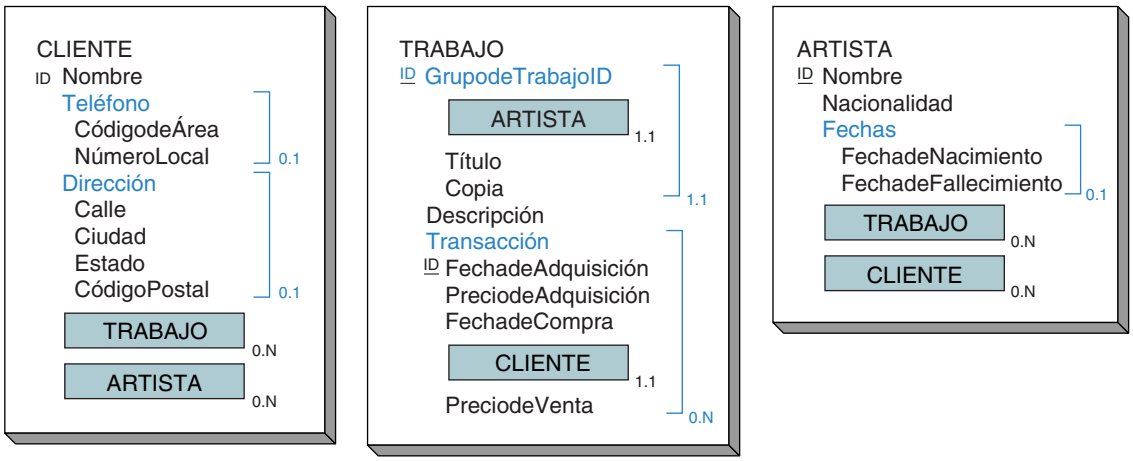
Reportar con qué rapidez se vende el trabajo de un artista y con qué margen de utilidad.

Mostrar el inventario actual en una página Web.

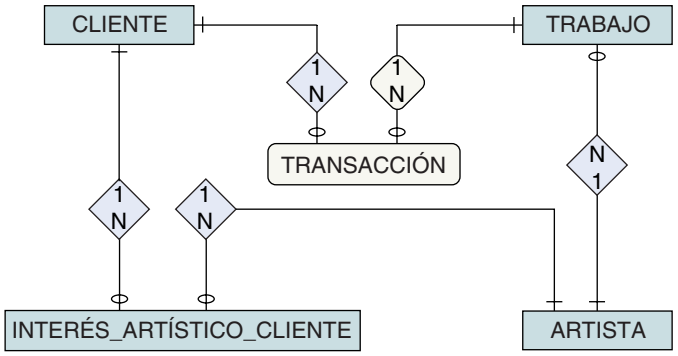
Listar reportes que empleará el contador de la galería.

➤ FIGURA 10-3

Diseño de la base de datos de la galería View Ridge: (a) objetos semánticos, (b) diagrama E-R, (c) diseño relacional, (d) diseño relacional con llaves sustitutas, (e) diagrama de relación de Access



(a)



(b)

CLIENTE (NúmerodeCliente, Nombre, CódigodeÁrea, NúmeroLocal, Calle, Ciudad, Estado, CódigoPostal)

TRABAJO (NombredelArtista, Título, Copia, Descripción)

TRANSACCIÓN (*NombredelArtista*, *Título*, *Copia*, FechadeAdquisición, PreciodeAdquisición, FechadeCompra, *NúmerodeCliente*, PreciodeVenta)

ARTISTA (NombredelArtista, Nacionalidad, FechadeNacimiento, FechadeFallecimiento)

INTERÉS_ARTÍSTICO_CLIENTE (NúmerodeCliente, NombredelArtista)

(c)

los intercambios), Transacción tiene valores múltiples. TRABAJO es un objeto híbrido porque contiene el atributo objeto CLIENTE en el grupo Transacción multivaluado (o multivalor).

View Ridge quiere registrar los intereses de sus clientes. En particular, desea conocer los artistas en los cuales un cliente en particular está interesado, y aquellos a los que les interesa un artista en especial. Estos requisitos se logran colocando los atributos multivaluados ARTISTA en CLIENTE, y CLIENTE en ARTISTA. En la figura 10-3(b) se muestra un diagrama E-R.

➤ FIGURA 10-3

(Continuación)

CLIENTE (ClienteID, Nombre, CódigodeÁrea, NúmeroLocal, Calle, Ciudad, Estado, CódigoPostal)
CUSTOMER (CustomerID, Name, AreaCode, LocalNumber, Street, City, State, Zip)

TRABAJO (TrabajoID, *ArtistaID,* Título, Copia, Descripción)
WORK (WorkID, *ArtistID*, Title, Copy, Description)

TRANSACCIÓN (TransacciónID, *TrabajoID,* FechadeAdquisición, PreciodeAdquisición, FechadeCompra, *ClienteID,* PreciodeVenta)
TRANSACTION (TransactionID, *WorkID*, DateAcquired, AcquisitionPrice, PurchaseDate, *CustomerID*, SalesPrice)

ARTISTA (ArtistaID, NombredelArtista, Nacionalidad, FechadeNacimiento, FechadeFallecimiento)
ARTIST (ArtistID, ArtistName, Nationality, Birthdate, DeceasedDate)

INTERÉS_ARTÍSTICO_CLIENTE (*ClienteID, ArtistaID*)
CUSTOMER_ARTIST_INT (*CustomerID*, *ArtistID*)

Restricciones de integridad referencial:

ArtistaID en TRABAJO debe existir en ArtistaID en ARTISTA

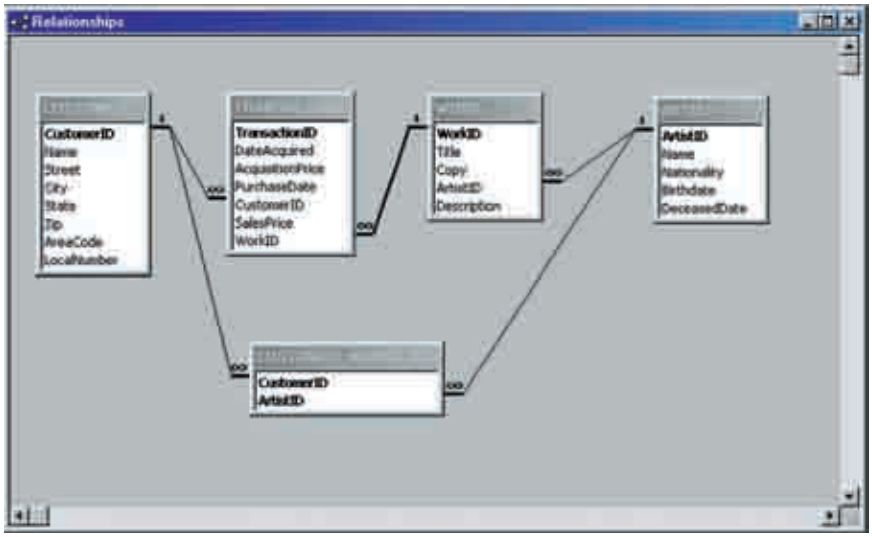
TrabajoID en TRANSACCIÓN debe existir en TrabajoID en TRABAJO

ClienteID en TRANSACCIÓN debe existir en ClienteID en CLIENTE

ClienteID en INTERÉS_ARTÍSTICO_CLIENTE debe existir en ClienteID en CLIENTE

ArtistaID en INTERÉS_ARTÍSTICO_CLIENTE debe existir en ArtistaID en ARTISTA

(d)



(e)

La representación relacional de los objetos de View Ridge se muestra en la figura 10-3(c). Puesto que el objeto CLIENTE no tiene un identificador único, se debe crear uno para usarlo como llave. Aquí hemos agregado un número de identificación: NúmerodeCliente. La cardinalidad máxima de los grupos Teléfono y Dirección es 1, así que los atributos en esos grupos se pueden colocar en la tabla CLIENTE; de hecho, no aparecen en la tabla como grupos, sino que la información del grupo se usa después para construir formas.

La llave de TRABAJO consiste en la llave de ARTISTA más Título y Copia. El único atributo sin la llave de TRABAJO es Descripción. Puesto que el atributo Transacción tie-

ne valores múltiples, debe creársele una tabla. Su llave es la del objeto en el que está contenida (NombredelArtista, Título, Copia) más la llave del grupo, la cual es FechadeAdquisición. Observe también que NúmerodeCliente se toma como llave externa en TRANSACCIÓN.

El objeto ARTISTA se representa mediante una tabla independiente, y NombredelArtista se puede usar como la llave de la tabla porque los nombres de los artistas están modelados para ser únicos. La tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE se debe crear para desarrollar la relación M:N entre CLIENTE y ARTISTA.

Debido al número de llaves de texto, y en particular a la gran llave compuesta de TRANSACCIÓN, este diseño puede mejorar sustancialmente si se reemplazan llaves de datos con llaves sustitutas. Esto se ha hecho en el diseño que se muestra en la figura 10-3(d). Los nombres de las llaves sustitutas se construyen anexando las letras ID al nombre de la tabla.

El diseño de la figura 10-3(d) es mejor porque hay menos datos duplicados. No es necesario copiar las columnas NombredelArtista, Título y Copia en la tabla Transacción. Debido a que puede haber muchas transacciones para un trabajo determinado, estos ahorros son valiosos. La figura 10-3(e) muestra un diagrama de relación para el diseño de la llave sustituta.

➤ CREAR, LEER, ACTUALIZAR Y BORRAR INSTANCIAS DE VISTAS

Una **vista** es una lista estructurada de elementos de datos de entidades u objetos semánticos definidos en el modelo. Un ejemplo de una vista es aquél que contiene datos para una entidad u objeto semántico. La figura 10-4(a) muestra un ejemplo de vista para la base de datos de la galería en la figura 10-3. Esta vista muestra datos de un cliente, sus transacciones e intereses artísticos. En esta vista existen potencialmente muchas TRANSACCIONes para cada CLIENTE, y cada TRANSACCIÓN tiene un TRABAJO. También, pueden existir muchos nombres de ARTISTA por cada CLIENTE (los artistas en los cuales el cliente está interesado).

Observe que ARTISTA.Nombre está repetido en esta vista. La primera vez es el nombre de un artista cuyo trabajo ha comprado el CLIENTE. La segunda, es el nombre del artista que le interesa al CLIENTE. Por ejemplo, en la figura 10-4(b) un cliente ha comprado trabajos de Tobey y de Miró y está interesado no sólo en el trabajo de estos dos artistas, sino también en el de Dennis Frings. En este caso, Tobey y Miró aparecen dos veces en la vista, una como un valor de TRABAJO.ARTISTA.Nombre y otra como un valor de ARTISTA.Nombre.

De nuevo, una vista es una lista estructurada de atributos. Debido a que está estructurada, los atributos pueden encontrarse más de una vez. También observe que una vista es sólo una lista de valores de datos. Puede ser formateada o materializada de muchas formas diferentes —como forma, reporte, u otro tipo de materialización.

Ahora considere las acciones CRUD que se pueden ejecutar en una vista. Comenzamos con Leer.

Lectura de ejemplos de vistas

Para leer una vista la aplicación debe ejecutar uno o más enunciados SQL para obtener valores de datos y luego poner los valores resultantes en la estructura de la vista. La vista de Cliente en la figura 10-4 contiene datos en dos caminos de acceso: uno a través de la tabla TRANSACCIÓN, y el otro mediante la tabla INTERÉS_ARTÍSTICO_CLIENTE. La estructura de los enunciados SQL es tal que sólo se puede seguir un camino a través del esquema con un enunciado SQL independiente. Así, esta vista requerirá un enunciado SQL para cada camino. Para el primero el siguiente enunciado SQL obtendrá los datos necesarios para el nombre del cliente "Jackson, Mary":

```
SELECT    CLIENTE.ClienteID, CLIENTE.Nombre,
          CLIENTE.CódigodeÁrea, CLIENTE.NúmeroLocal,
```

➤ FIGURA 10-4

Vista de Cliente: (a) lista estructurada de atributos, y (b) datos de muestra

CLIENTE.Nombre
CLIENTE.CódigodeÁrea
CLIENTE.NúmeroLocal
    TRANSACCIÓN.FechadeCompra
    TRANSACCIÓN.PreciodeVenta . . .
        TRABAJO.ARTISTA.Nombre
        TRABAJO.Título
        TRABAJO.Copia
    ARTISTA.Nombre . . .

(a) Lista estructurada de atributos

Jackson, Elizabeth		
206		
284-6783		
	7/15/94	
	4,300	
		Mark Tobey
		Lithograph One
		10/75
	2/5/99	
	2,500	
		Juan Miró
		Poster 14
		5/250
	11/22/97	
	17,850	
		Juan Miró
		Awakening Child
		1/1
	Juan Miró	
	Mark Tobey	
	Dennis Frings	

(b) Datos de muestra

```
        ARTISTA.Nombre, TRABAJO.Título, TRABAJO.Copia,
        TRANSACCIÓN.FechadeCompra, TRANSACCIÓN.PreciodeVenta
FROM    CLIENTE, TRANSACCIÓN, TRABAJO, ARTISTA
WHERE   CLIENTE.ClienteID = TRANSACCIÓN.ClienteID
  AND   TRABAJO.TrabajoID = TRANSACCIÓN.TrabajoID
  AND   ARTISTA.ArtistaID = TRABAJO.ArtistaID
  AND   CLIENTE.Nombre = 'Jackson, Mary'
```

Repasemos este enunciado SQL mientras revisamos el diagrama de relación en la figura 10-3(e). Los tres joins son necesarios para obtener datos a través de las tres relaciones en la parte superior del diagrama.

En el contexto del desarrollo de la aplicación, al resultado de un enunciado SQL a veces se le llama **recordset.** Para Microsoft, este término significa una relación con un empaquetador de programación de objetos. Un recordset tiene tanto métodos como

propiedades. *Open* es un ejemplo de método del recordset, *Cursor Type* es un ejemplo de una propiedad del recordset. Aprenderá más acerca de esto en el capítulo 14.

Para obtener los intereses de un cliente en determinado artista, se requiere un enunciado SQL para seguir el camino a través de INTERÉS_ARTÍSTICO_CLIENTE. El enunciado SQL para este camino es:

```
SELECT    CLIENTE.ClienteID, ARTISTA.Nombre
FROM      CLIENTE, INTERÉS_ARTÍSTICO_CLIENTE, ARTISTA
WHERE     CLIENTE.ClienteID = INTERÉS_ARTÍSTICO_CLIENTE.ClienteID
  AND     INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ARTISTA.ArtistaID
  AND     CLIENTE.Nombre = 'Jackson, Mary'
```

Debido a que CLIENTE.Nombre no es único, posiblemente los dos recordset de estos enunciados recuperarán datos de más de un cliente. Por lo tanto, la aplicación necesita tener lógica para examinar los valores de ClienteID en el recordset y asociar los renglones correctos.

Después de ejecutar estos dos enunciados, la aplicación tiene todos los datos necesarios para construir uno o más ejemplos de la vista que se presenta en la figura 10-3. Cómo se hace esto depende del lenguaje que se utilice. En COBOL la información se coloca en estructuras definidas en el Data Division. En Visual Basic la información se puede colocar en una estructura de datos o una serie de arreglos. En C++ y Java la información se coloca en los objetos. En realidad esos puntos no nos preocupan; lo que importa es que usted logre tener una idea completa de cómo debe ejecutar la aplicación uno o más enunciados SQL para llenar la estructura de la vista de datos. Véase el capítulo 16 para tener un ejemplo de Java.

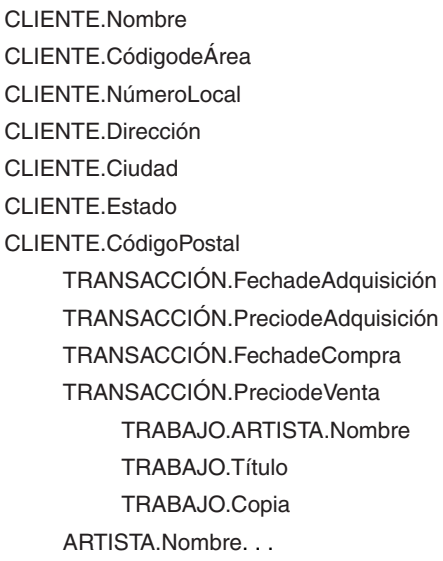
Creación de instancias de una vista

Para crear una instancia de una vista, la aplicación primero debe obtener los nuevos valores de datos y relaciones. Esto se hace mejor mediante un formato de entrada de información, pero también recibe datos de otros programas, y en otras maneras. Una vez que la aplicación tiene valores de datos, entonces ejecuta los enunciados SQL para almacenar la información en la base de datos.

Considere la vista Nuevo Cliente en la figura 10-5; se usa cuando un nuevo cliente compra una pintura. Contiene información acerca de él, de la transacción de compra y de sus múltiples intereses en obras de arte. Esta vista difiere de la de la figura 10-4 porque tiene más datos de clientes y también permite sólo una transacción. Sin embargo,

➤ FIGURA 10-5

Nueva vista de Cliente



puede haber valores múltiples ARTISTA.Nombre que registren los nuevos intereses del cliente.

Suponga que los valores de los datos para esta vista se localizan en una estructura de programa llamada NuevoCliente; además suponga que podemos accesar a los valores en la estructura anexando los caracteres NuevoCliente a los nombres en la estructura. Así, NuevoCliente.CLIENTE.Nombre tiene acceso a Nombre del CLIENTE en la estructura NuevoCliente.

Para crear esta vista en la base de datos debemos almacenar los nuevos datos del cliente en CLIENTE, los nuevos datos de transacción en TRANSACCIÓN y crear un renglón en la tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE para cada uno de los artistas que le interesan al cliente.

El siguiente enunciado SQL almacenará los nuevos datos del cliente:

```
INSERT    INTO CLIENTE
          (CLIENTE.Nombre,
          CLIENTE.CódigodeÁrea,
          CLIENTE.NúmeroLocal,
          CLIENTE.Dirección,
          CLIENTE.Ciudad,
          CLIENTE.Estado,
          CLIENTE.CódigoPostal)
VALUES    (NuevoCliente.CLIENTE.Nombre,
          NuevoCliente.CLIENTE.CódigodeÁrea,
          NuevoCliente.CLIENTE.NúmeroLocal,
          NuevoCliente.CLIENTE.Dirección,
          NuevoCliente.CLIENTE.Ciudad,
          NuevoCliente.CLIENTE.Estado,
          NuevoCliente.CLIENTE.CódigoPostal)
```

Suponga que cuando se crea el nuevo renglón el DBMS asigna el valor de la llave sustituta CLIENTE.ClienteID. Necesitaremos los valores de esta llave para terminar la creación de la nueva instancia, así que la aplicación necesitará obtenerla. Una manera de hacerlo es ejecutar el siguiente SELECT de SQL:

```
SELECT    CLIENTE.ClienteID, CLIENTE.CódigodeÁrea,
          CLIENTE.NúmeroLocal
FROM      CLIENTE
WHERE     CLIENTE.Nombre = NuevoCliente.CLIENTE.Nombre
```

Debido a que CLIENTE.Nombre no es único, puede aparecer más de un renglón en el recordset. En este caso, lo correcto sería identificarlo examinando la información de número de teléfono. Suponga que esto se ha realizado si es necesario y que el valor correcto está colocado en la estructura del programa como NuevoCliente.CLIENTE.ClienteID.

Un enunciado INSERT también será utilizado para almacenar el nuevo registro TRANSACCIÓN. Sin embargo, en este caso los valores de las llaves externas TRANSACCIÓN.TrabajoID y TRANSACCIÓN.ClienteID tendrán que ser proporcionados. Ya hemos mostrado cómo obtener el valor de ClienteID, así que lo único que falta es obtener el valor de TrabajoID. El siguiente SQL hará esto:

```
SELECT    TRABAJO.TrabajoID
FROM      TRABAJO,ARTISTA
WHERE     TRABAJO.ArtistaID = ARTISTA.ArtistaID
  AND     ARTISTA.Nombre = NuevoCliente.TRABAJO.ARTISTA.Nombre
  AND     TRABAJO.Título = NuevoCliente.TRABAJO.Título
  AND     TRABAJO.Copia = NuevoCliente.TRABAJO.Copia
```

Suponga que el valor de la llave sustituta que se ha regresado se almacena como NuevoCliente.TRABAJO.TrabajoID.

Se puede ejecutar el siguiente SQL para agregar el nuevo renglón TRANSACCIÓN:

```
INSERT      INTO TRANSACCIÓN
                (TRANSACCIÓN.TrabajoID,
                TRANSACCIÓN.FechadeAdquisición,
                TRANSACCIÓN.PreciodeAdquisición,
                TRANSACCIÓN.FechadeCompra,
                TRANSACCIÓN.ClienteID,
                TRANSACCIÓN.PreciodeVenta)
            VALUES
                (NuevoCliente.TRABAJO.TrabajoID,
                NuevoCliente.TRANSACCIÓN.FechadeAdquisición,
                NuevoCliente.TRANSACCIÓN.PreciodeAquisición,
                NuevoCliente.TRANSACCIÓN.FechadeCompra,
                NuevoCliente.CLIENTE.ClienteID,
                NuevoCliente.TRANSACCIÓN.PreciodeVenta)
```

Ahora todo lo que falta es crear renglones para la tabla de intersección INTERÉS_ARTÍSTICO_CLIENTE. Para hacerlo, necesitamos obtener el ArtistaID de cada artista que le interesa al cliente, y luego crear un nuevo renglón en la tabla de intersección. El siguiente seudocódigo ilustra la lógica:

```
Para cada NuevoCliente.ARTISTA.Nombre:
        SELECT      ARTISTA.ArtistaID
        FROM        ARTISTA
        WHERE       ARTISTA.Nombre = NuevoCliente.ARTISTA.Nombre
        INSERT       INTO INTERÉS_ARTÍSTICO_CLIENTE
                    (ClienteID, ArtistaID)
                    VALUES (NuevoCliente.CLIENTE.ClienteID,
                    ARTISTA.ArtistaID)
El siguiente NuevoCliente.ARTISTA.Nombre
```

En este punto, la vista de Nuevo Cliente se ha almacenado en la base de datos. Por supuesto, una aplicación completa incluye la lógica para captar los errores que ha regresado el DBMS y procesarlos. Por ejemplo, la aplicación debe manejar los casos de TRABAJO que no existen en la base de datos, y los de un ARTISTA que tampoco tiene registrado.

Actualización de instancias de vistas

La tercera acción fundamental para realizar una vista es la actualización. Cuando se actualiza una vista hay tres tipos de cambio que son posibles: uno es un cambio simple de valores, como por ejemplo un cliente que cambia de número telefónico. Otra es un cambio a una relación; por ejemplo cuando un cliente ya no tiene interés en determinado artista. Un tercer tipo de actualización requeriría agregar uno o más nuevos renglones; eso ocurriría en nuestro ejemplo cuando un cliente hiciera una nueva compra.

El primer tipo de actualización se puede lograr con enunciados UPDATE SQL. Por ejemplo, suponga que un programa tiene una estructura llamada ActualizaciónCliente

(UpdateCust), y que ésta a su vez tiene ClienteID, CódigodeÁrea y NúmeroLocal. El siguiente SQL actualizará los nuevos valores:

```
UPDATE   CLIENTE
SET      CLIENTE.CódigodeÁrea = ActualizaciónCliente.CódigodeÁrea
         CLIENTE.NúmeroLocal = ActualizaciónCliente.NúmeroLocal
WHERE    CLIENTE.ClienteID = ActualizaciónCliente.ClienteID
```

Los cambios a relaciones también son directos. Si la relación es uno a muchos, entonces sólo se necesita actualizar el valor de la llave externa conforme al nuevo valor. Por ejemplo, suponga que la relación de DEPARTAMENTO con EMPLEADO es 1:N. Entonces Depto# (u otra llave) será almacenado como una llave externa en EMPLEADO. Para mover un EMPLEADO a un nuevo DEPTO, la aplicación sólo necesita cambiar el valor Depto# por uno nuevo.

Si la relación es muchos a muchos, entonces la llave externa en la tabla de intersección necesitará ser modificada. Por ejemplo, si en la galería un cliente ya no se interesa por Mark Tobey sino por Dennis Frings, entonces el renglón de intersección que representa la conexión con Mark Tobey necesita ser cambiado para que apunte a Dennis Frings.

Suponga que ActualizaciónCliente.ClienteID tiene el ID del cliente, ActualizaciónCliente.ViejoArtistaID tiene el ID del renglón de Mark Tobey en ARTISTA, y ActualizaciónCliente.NuevoArtistaID tiene el ID del renglón de Dennis Frings en ARTISTA. El siguiente SQL hará el cambio necesario:

```
UPDATE   INTERÉS_ARTÍSTICO_CLIENTE
SET      INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ActualizaciónCliente.NuevoArtistaID
WHERE    INTERÉS_ARTÍSTICO_CLIENTE.ClienteID = ActualizaciónCliente.ClienteID
  AND    INTERÉS_ARTÍSTICO_CLIENTE.ArtistaID = ActualizaciónCliente.ViejoArtistaID
```

En una relación muchos a muchos también es posible eliminar una conexión sin reemplazarla, y crear una nueva sin desaparecer una vieja. Para borrar una conexión sólo eliminaríamos el renglón apropiado en la tabla de intersección. Para crear una agregaríamos un nuevo renglón en la tabla de intersección.

El tercer tipo de actualización requiere aumentar un nuevo renglón en una o más tablas. Si, por ejemplo, un cliente hace una nueva compra, entonces se necesitará crear otro renglón en la tabla de TRANSACCIÓN. Esto se puede realizar de la misma manera en que un nuevo renglón de TRANSACCIÓN se agrega cuando se crea una nueva instancia de vista.

Borrado de instancias de vista

Borrar una vista implica eliminar renglones de las tablas que la integran. La clave está en saber cuánto borrar. Por ejemplo, suponga que la galería quiere eliminar información de un cliente cuyo nombre es “Jones, Mary”. Obviamente, es necesario eliminar el renglón “Jones, Mary” en CLIENTE. Todo lo de los renglones de intersección en INTERÉS_ARTÍSTICO_CLIENTE que corresponden a esa clienta también deberá ser eliminado. ¿Pero qué hay acerca de los renglones en la tabla de TRANSACCIÓN? Esta tabla contiene ClienteID y si su renglón es eliminado todos los renglones en TRANSACCIÓN que tienen su valor de ClienteID tendrán datos inválidos.

Las respuestas a preguntas como esta surgen a partir del modelo de datos. En el caso del modelo E-R, todas las entidades débiles son eliminadas si la entidad de la que dependen también lo es. De otra manera, ningún renglón adicional de la tabla es suprimido. En el caso del modelo de objeto semántico todos los datos dentro de un objeto

son eliminados (puede haber muchas tablas en un objeto si éste tiene atributos multivaluados), pero no se suprime ningún dato en un objeto diferente. En adición a estas reglas, no se debe permitir ninguna eliminación si puede causar una violación a las cardinalidades de la relación. Tocaremos este tema ahora, y más adelante lo trataremos con más profundidad.

Considere el modelo en la figura 10-3. Cuando se borra un cliente, el renglón en CLIENTE es eliminado junto con todos sus renglones en INTERÉS_ARTÍSTICO_CLIENTE correspondientes. Los datos de TRANSACCIÓN no serán eliminados porque residen en un objeto diferente llamado TRABAJO. Sin embargo, observe que la cardinalidad mínima de CLIENTE en TRANSACCIÓN es 1. Por lo tanto, si un CLIENTE específico está ligado a una TRANSACCIÓN, entonces éste se requiere y su eliminación no debe ser permitida.

También podemos concluir de la figura 10-3 que si una vista de TRABAJO es eliminada, entonces el renglón TRABAJO y todos los renglones de TRANSACCIÓN relacionados se borrarán.

Por último, si se suprime una vista de ARTISTA, entonces los renglones ARTISTA e INTERÉS_ARTÍSTICO_CLIENTE se borrarán. Además, si hay algún objeto TRABAJO que esté ligado a ese ARTISTA, entonces la eliminación no podrá ser permitida.

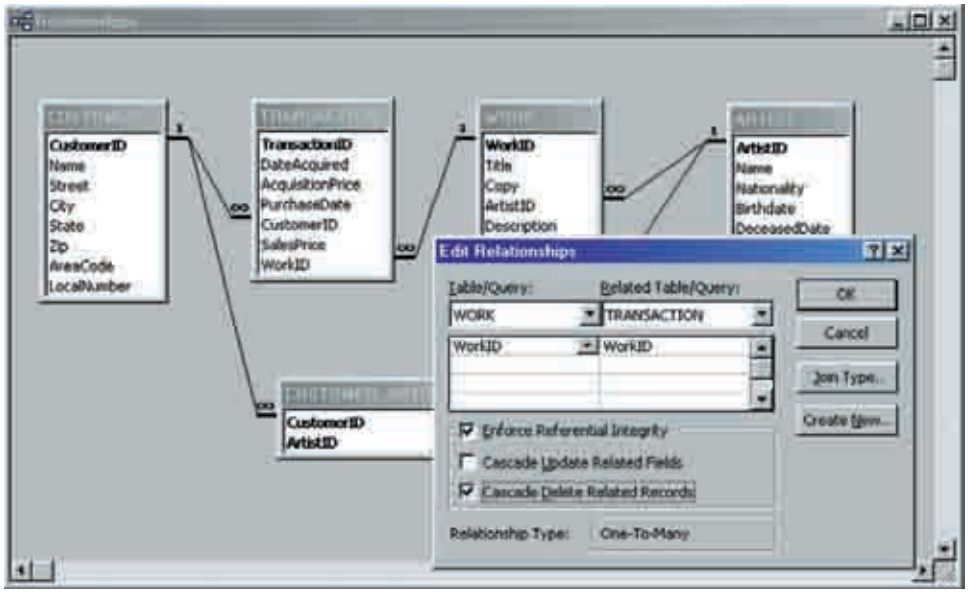
Por lo general algunos productos DBMS soportan la eliminación de los renglones dependientes bajo el término **eliminaciones en cascada.** La figura 10-6 muestra la caja de diálogo de la relación para Access. Observe que ha sido verificada la caja de registros de la relación de eliminación en cascada. Esto significa que cuando un renglón WORK (TRABAJO) es suprimido, Access automáticamente elimina cualquier renglón relacionado con TRANSACTION (TRANSACCIÓN).

Como se estableció, la cardinalidad de la relación desempeña un papel importante en la decisión de eliminar o no los renglones en la vista. Regresaremos a este tema más adelante cuando abordemos cómo forzar o hacer cumplir restricciones.

En esta sección estudiamos las acciones que se deben seguir cuando se crean, leen, actualizan y eliminan instancias de vistas. Algunas de las acciones aquí descritas se pueden realizar automáticamente mediante los productos DBMS. Por ejemplo, Access tiene asistentes (wizards) que generarán un formato con Cliente y un camino autónomo —ya sea para los intereses relacionados con Transacciones o con Artista—. Los usuarios de este formato pueden crear, leer, actualizar y eliminar instancias de un Cliente y datos en uno de los caminos. Sin embargo, los asistentes de Access no generarán un formato que apoye ambos caminos en la vista de Cliente en la figura 10-3. Para que lo anterior sea posible, un programador necesitará escribir el código de un programa.

➤ FIGURA 10-6

Especificación de eliminaciones en cascada



➤ DISEÑO DE FORMAS

Como se mostró en la figura 10-1, la segunda función principal de una aplicación de la base de datos es generar materializaciones de vistas. En este capítulo analizaremos materializaciones de formatos y reportes. En el capítulo 14 abordaremos materializaciones de interproceso usando XML, y en el capítulo 17 estudiaremos materializaciones de cubos OLAP.

Una **forma** es un desplegado en pantalla que se usa para introducir y editar datos. Las formas de sólo lectura también se pueden usar para reportar datos, pero en la mayoría de los casos, cuando los programadores hablan de formas, se refieren a las que se usan para el ingreso y la edición de datos.

Algunas formas son fáciles de usar y dan como resultado pocos errores en el ingreso de datos. Otras parecen difíciles de manejar, artificiales, y es difícil usarlas sin que se creen errores. En esta sección analizamos e ilustramos varios principios sobre el diseño correcto de formas.

La estructura de la forma debe reflejar la estructura de la vista

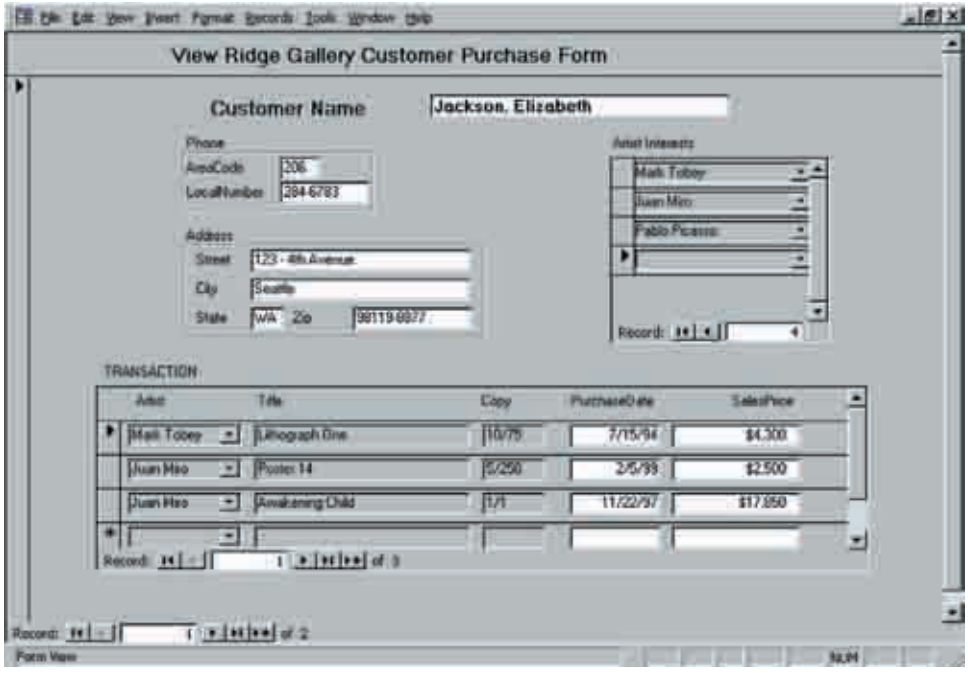
Primero, para que parezca natural y sea fácil de usar, *la estructura de una forma debe reflejar la estructura de la vista que materializa*. Observe la forma de la figura 10-7, la cual es una materialización de la vista Cliente en la figura 10-4(a).

La estructura de esta forma refleja la de la vista Cliente (Customer). Una sección de la forma tiene los datos básicos del cliente tales como Nombre, Teléfono y Dirección (Name, Phone and Address). La segunda sección muestra los intereses del cliente en Artista (Artist interests). Por último, la tercera sección lista las transacciones de compra del cliente. Los usuarios encontrarán esta forma fácil de usar debido a que los atributos están agrupados de manera tal que reflejan su comprensión sobre la estructura de los datos del cliente.

En general, cuando se diseña una forma toda la información de una relación independiente se debe colocar en una sección contigua de la forma. Suponiendo que la base de datos está en DK/NF, cada relación debe pertenecer a un tema específico, y el

➤ FIGURA 10-7

Materialización de una forma de vista de Cliente en la figura 10-4(a)



usuario esperaría encontrar en un lugar los datos de ese tema. Por lo tanto, en la forma hay clientes, interés del artista y secciones de transacción.

Hay una excepción a esta regla: los atributos en la relación base de la vista (aquí, la relación CLIENTE) a veces no están colocados en forma contigua. Suponga, por ejemplo, que CLIENTE tiene un atributo independiente llamado TotaldeCompras. Si siguiéramos esta regla pondríamos TotaldeCompras en la primera sección de la forma. Pero para los usuarios sería más claro poner ese atributo al final, después de que se hayan listado todas las compras.

La forma en la figura 10-7 no es la única aceptable de esta vista. Por ejemplo, TRANSACTION se podría colocar antes de Artist Interests. Los datos básicos de CUSTOMER podrían ser reacomodados para que tuvieran una apariencia más horizontal. CustomerName, AreaCode y LocalNumber se podrían colocar en una columna, y Street, City, State y Zip en otra. Todas estas alternativas permiten la estructura de la forma para que refleje la estructura de los objetos implícitos.

La semántica de la información debe ser gráficamente obvia

Otra característica de una forma bien diseñada es que la semántica de los datos es gráficamente obvia. Considere la sección de datos básicos del cliente de la forma CUSTOMER en la figura 10-7. Observe que existen rectángulos alrededor de AreaCode y LocalNumber y alrededor de Street, City, State y Zip.

Para entender por qué se hace esto, remitámonos nuevamente a la figura 10-3(a). El objeto semántico CUSTOMER (Cliente) tiene un atributo de grupo llamado Teléfono y otro Dirección. Puesto que ambos tienen una cardinalidad máxima de 1, en realidad no se requieren —desde el punto de vista del diseño relacional—. De hecho, no aparecen en las relaciones en la figura 10-3(c) o (d). El único propósito de estos atributos de grupo es asociar semánticamente CódigodeÁrea con NúmeroLocal, y Calle, Ciudad, Estado y CódigoPostal con algún otro.

El propósito de los rectángulos en la forma Customer Purchase Form es aclarar gráficamente estas asociaciones. La mayoría de los usuarios están de acuerdo con este arreglo. Debido a que saben que el número de teléfono consta de CódigodeÁrea y NúmeroLocal, les parece que la asociación gráfica de ambos es muy apropiada.

La estructura de la forma debe fomentar la acción apropiada

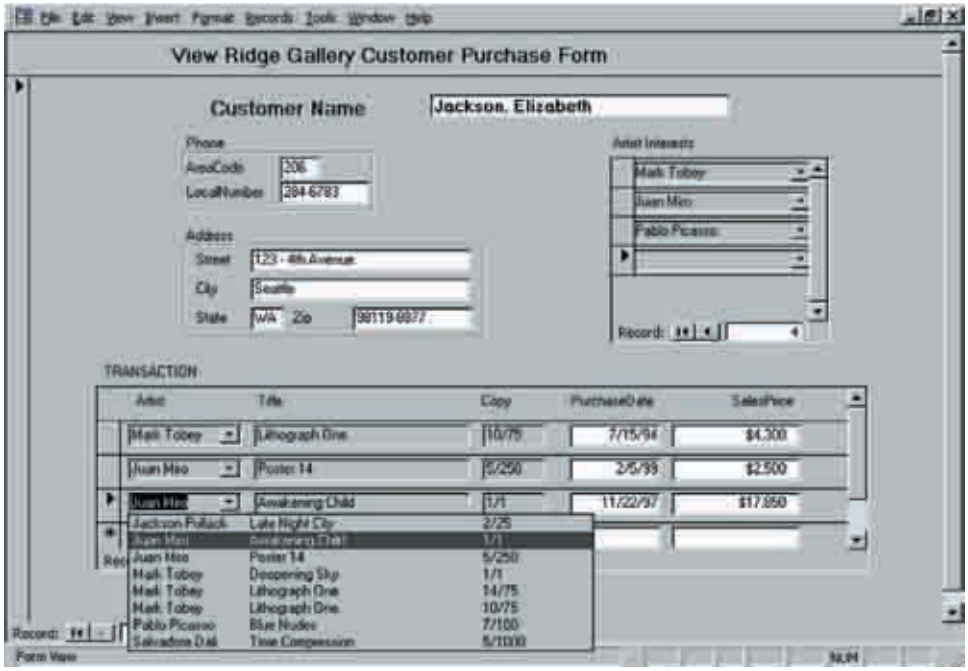
La estructura de las formas debe facilitar la ejecución de acciones apropiadas y dificultar la ejecución de acciones erróneas. Por ejemplo, el campo para ingresar State (Estado) en la forma de la figura 10-7 es pequeño. Obviamente, se supone que el usuario introduce sólo una abreviación de dos dígitos; un diseño más apropiado sólo permitiría ingresar dos dígitos; pero el mejor diseño presentaría los nombres de los estados en una lista desplegable y permitiría que el usuario seleccionara.

En la figura 10-7 algunos campos son blancos y otros grises. Las formas en esta aplicación han sido diseñadas para que el usuario sólo pueda ingresar datos en los espacios en blanco; los elementos de los datos en gris no pueden ser modificados. Así, el usuario no puede escribir en los campos Title o Copy en la sección TRANSACTION. En lugar de eso, para seleccionar o cambiar una obra, el usuario hace “clic” en la caja que está abajo de Artist. Cuando hace clic en la flecha, se despliega una lista de obras, como se muestra en la figura 10-8. Si el usuario quiere actualizar las obras o los datos de un artista, o agregar o eliminar datos de esa obra o dicho artista, debe utilizar una forma diferente.

La razón de este diseño es que cuando los usuarios están ingresando nuevas transacciones para los clientes, sólo deberían agregar los datos de la transacción; a saber, PurchaseDate (FechadeCompra) y SalesPrice (PreciodeVenta). Permitirles cambiar datos en ARTIST o WORK abriría la puerta a cambios accidentales o erróneos, que obstaculizaría la venta de una obra.

➤ FIGURA 10-8

Uso de la caja de lista desplegable



Un diseño similar se usa para Artist Interests. Los usuarios sólo pueden elegir de una lista. Sin embargo, el caso no es tan inflexible. Los vendedores podrían querer registrar que un cliente se interesa en un artista que no forma parte de la base de datos de la galería. Si es así, la forma cambiaría para permitir que los usuarios agreguen otros nombres a la lista. Como justificación a este diseño de la figura 10-7, el dueño de la galería quizá quiera registrar únicamente a los artistas de interés con los que trabaja o a los cuales representará. Probablemente quiera tener control de qué nombres pueden aparecer.

Formas en un ambiente GUI

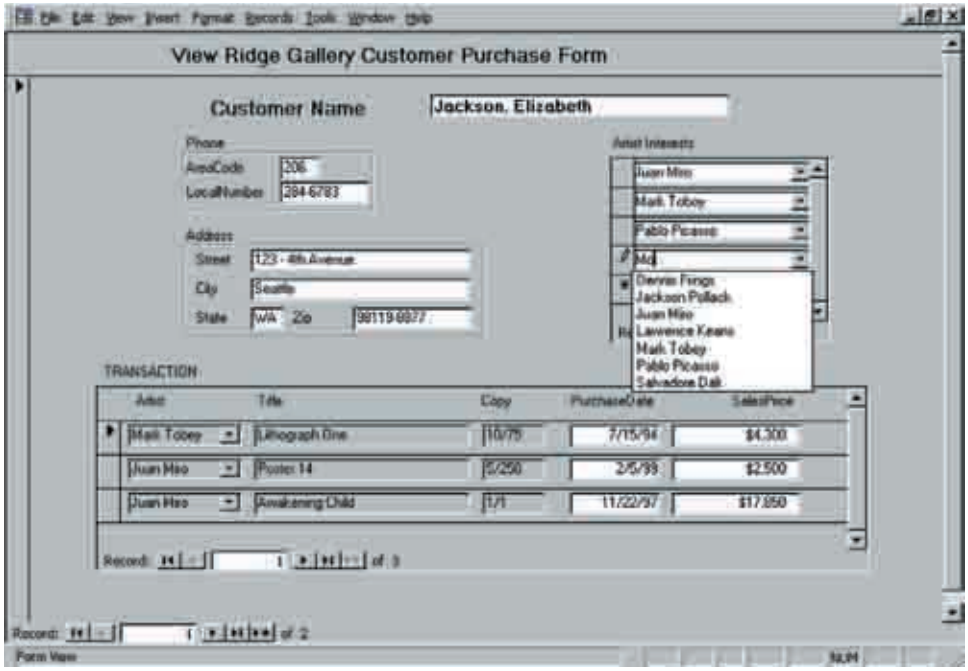
Hay varias características peculiares de los sistemas GUI que pueden facilitar enormemente el uso de las aplicaciones de la base de datos.

CAJAS DE LISTAS DESPLEGABLES. Una **caja de lista desplegable** es un control GUI que presenta una lista de elementos entre los cuales puede elegir el usuario. Una propiedad del control determina si la lista es fija o si los usuarios pueden agregarle algo. La figura 10-9 muestra una versión diferente de Customer Purchase Form (Forma de Compra del Cliente) en la cual los usuarios pueden agregar Nombres de Artistas que no formen parte de la lista. Tras bambalinas, la aplicación se almacena en un nuevo renglón de ARTIST en la base de datos. Por supuesto, para que esto funcione sólo ArtistName (NombredelArtista) puede ser un atributo requerido. Si hubiera otros campos requeridos, el DBMS rechazaría la inserción del nuevo renglón.

Las cajas de listas tienen muchas ventajas sobre las cajas de entrada de datos. Primero, para la gente es más fácil *reconocer* que *recopilar*. Por ejemplo, en la forma en la figura 10-8, es más fácil elegir de la lista el nombre de un artista, que recordar a todos los que forman parte de la base de datos. También es más fácil reconocer un nombre que escribirlo en forma correcta. Finalmente, si la caja de lista está preparada para desplegar sólo los valores presentes en la base de datos, el usuario no puede introducir errores de mecanografía. Por desgracia, el DBMS considerará como dos artistas diferentes a Juan Miró (con un espacio entre la *n* y la *M*) y a Juan Miró (con dos espacios entre la *n* y la *M*). Las cajas de listas fijas desplegables evitan estos errores.

➤ FIGURA 10-9

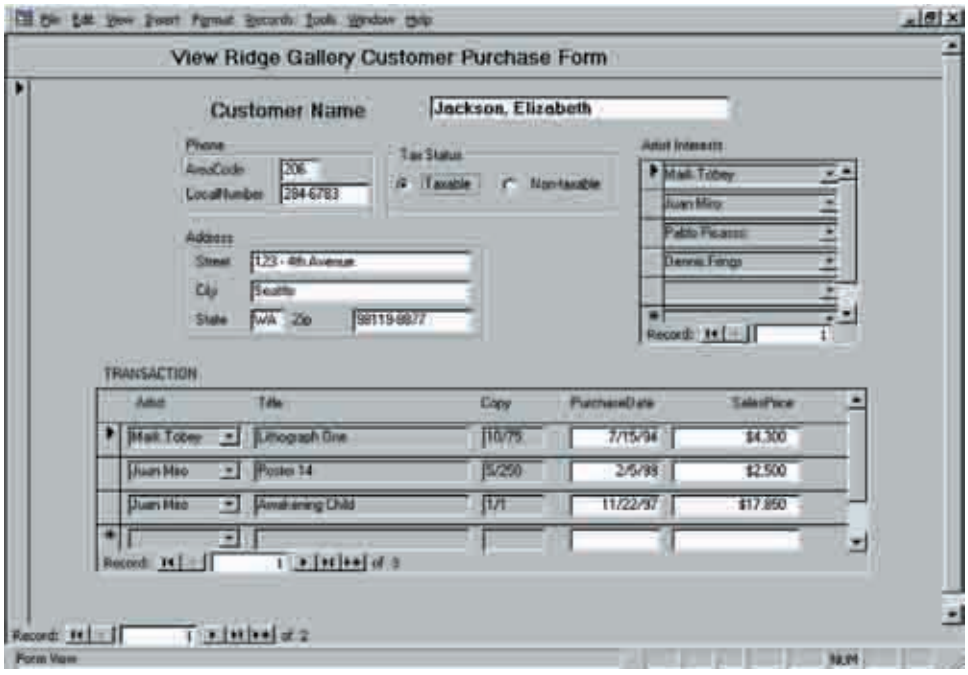
Caja de lista que permite nuevos valores



BOTONES DE OPCIONES Y GRUPOS. Un **botón de opción,** o un **botón de radio,** es un dispositivo de desplegado que permite a los usuarios seleccionar una condición alternativa o estado a partir de una lista de posibilidades. Por ejemplo, suponga que la galería decide que necesita registrar el estado fiscal de sus clientes. Suponga que éste tiene dos posibilidades:Taxable and Non-taxable (gravable y no gravable), y que ambas son mutuamente excluyentes. Se podría usar un grupo de opción como el que se muestra en la figura 10-10 para recopilar esos datos. Debido a la forma en la que el grupo de opción funciona, si el usuario selecciona Taxable, entonces Non-taxable automática-

➤ FIGURA 10-10

Uso de botones de grupo y de opción



mente deja de ser una selección. Si el usuario opta por Nogravable, entonces Gravable se elimina automáticamente.

Detrás de la forma, el programa de aplicación debe almacenar información en una columna de la tabla que representa el botón de radio seleccionado. Para este ejemplo, la columna se llama TaxStatus (EstadoFiscal). Se usa una de las dos maneras posibles para almacenar los datos del botón de opción. Una es almacenar un número entero del 1 al número de botones. En este ejemplo, uno de los valores 1 o 2 sería almacenado. La segunda opción es almacenar un valor de texto que describa la opción elegida. Las posibilidades son Sí o No.

CAJAS DE VERIFICACIÓN (CHECK BOXES). Las **cajas de verificación** son similares a los botones de opción, excepto que las alternativas en un grupo de cajas de verificación no son mutuamente excluyentes —se puede elegir más de una alternativa—. Suponga, por ejemplo, que la galería quiere registrar el tipo, o los tipos, de material artístico que les interesan a los clientes. Los posibles tipos son Lithographs, Oils, Pastels, and Photographs (Litografías, Óleos, Pasteles y Fotografías), y se muestran en una serie de cajas de verificación en la versión de la forma CUSTOMER en la figura 10-11. El usuario selecciona o marca las cajas apropiadas.

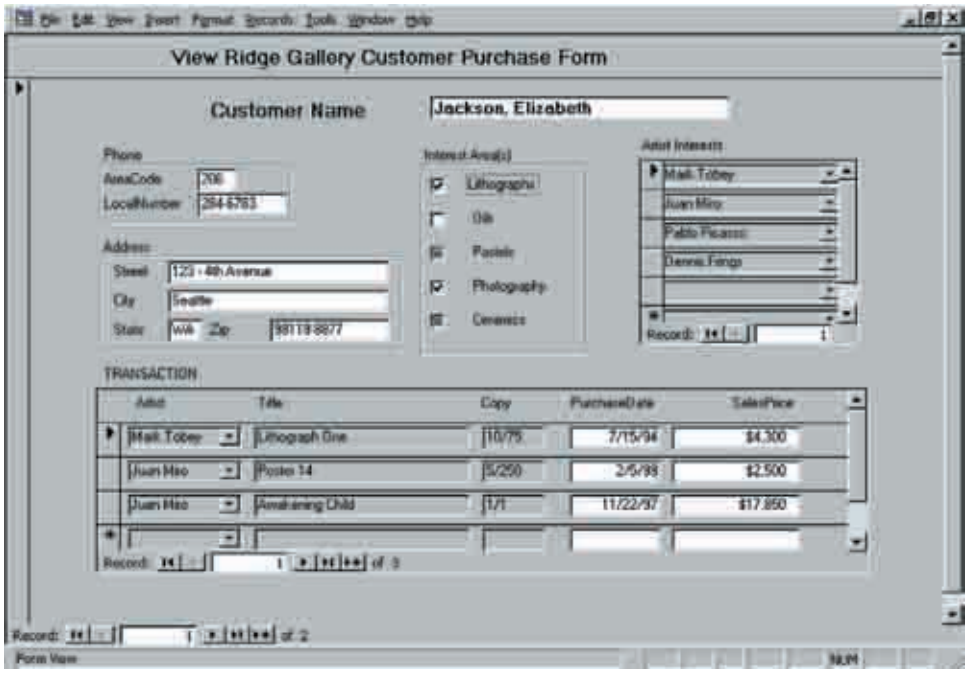
Hay varias formas de representar cajas de verificación en las relaciones. Una manera común y sencilla es definir una columna de valores booleanos para cada elemento en el grupo de cajas de verificación. El valor de cada una de estas columnas es binario; esto es, puede ser 1 o 0 para representar Sí o No. Hay otras posibilidades tales como codificar bits en un byte, aunque éstos no son importantes para nuestro análisis.

En la figura 10-11 observe los cuadros grises en Pastels y Ceramics. Ese gris significa que los valores de esas opciones son nulos. Los usuarios pueden interpretarlos como que este cliente está interesado en Lithographs y Photography y no en Oils. Su interés en Pastels y Ceramics es ambiguo (véase el análisis de nulos en el capítulo 6). Si es importante registrar el estado nulo, entonces el diseño de la base de datos debe permitir tres valores para cada caja de verificación: verificado, no verificado y nulo. Aunque el análisis también corresponde a los grupos con botones de opción.

FORMAS WEB. Las formas usadas en el explorador son GUI y todos los comentarios anteriores también se aplican. Sin embargo, las formas Web tienen una característica que las de Windows no tienen: hiperconexiones, las cuales son vínculos especial-

➤ FIGURA 10-11

Uso de cajas de verificación



mente útiles para materializar conexiones con otros objetos semánticos o entidades fuertes. Por ejemplo, en la forma de la figura 10-7, las conexiones con artistas de interés se pueden reemplazar por hiperconexiones para que cuando el usuario haga clic en éstas sea llevado a una forma ARTIST con los datos apropiados. Se puede hacer lo mismo con los enlaces a WORKs.

Movimiento del cursor y teclas de uso exclusivo

Otro aspecto sobre el diseño de formas es la acción del cursor, el cual se debe mover a través de la forma de manera fácil y natural. Esto generalmente significa que el cursor sigue el patrón de procesamiento del usuario conforme éste lee los documentos de ingreso de los datos fuente. Si se usan formas para introducir datos por teléfono, el cursor debe controlar el ritmo de la conversación. En este caso, este movimiento progresará de tal manera que el cliente lo encuentre natural y conveniente.

El movimiento del cursor es especialmente importante durante y después de una condición de excepción. Suponga que usando la forma en la figura 10-7 se comete un error —quizás al introducir un código inválido de estado—. La forma se debe procesar para que el cursor se mueva a una posición lógica. Por ejemplo, la aplicación puede desplegar una caja de lista de valores de estado disponibles y colocar al cursor en una posición lógica en la lista —quizás en el primer estado que empieza con la primera letra, la cual introdujo el usuario—. Cuando se selecciona el State (estado), el cursor debe regresar a Zip (CódigoPostal), el siguiente espacio adecuado de la forma.

Las acciones de las teclas de propósito especial, tales como ESC y las llaves de función, deberían ser *consistentes* y *exclusivas* en su uso. Si se usa ESC para salir de formas, se le debe usar únicamente para ese propósito (excepto cuando se trate de acciones que sean lógicamente equivalentes para salir de las formas). Las acciones de las teclas deben ser consistentes en toda la aplicación. Esto es, si se usa ESC para salir de una forma, también se debe usar para salir de todas las formas. Si se usa Ctrl-D para borrar datos en una forma, se debe usar para borrar datos en todas las formas. De otro modo, los hábitos que se adquieren en una parte de la aplicación se deben olvidar o aprender nuevamente en otras partes de la aplicación. Esto, además de ser un desperdicio de tiempo, es frustrante, exasperante y ocasiona errores. Aunque estos comentarios pueden parecer obvios, el cuidado con este tipo de detalles es lo que hace a una forma fácil y conveniente de usar.

➤ DISEÑO DE REPORTES

El tema del diseño de reportes, más que el diseño de formas, se ha analizado ampliamente en textos de desarrollo de aplicaciones. No duplicaremos, ni siquiera trataremos de resumir esos análisis aquí, sino más bien observaremos varios conceptos directamente relacionados con la noción de un reporte, tal como una materialización de la vista de una base de datos.

Estructura de reportes

Los principios para el diseño de un reporte eficaz son similares a los de diseño de formas. Igual que con las formas, *la estructura de un reporte debe reflejar la estructura de la vista implícita*. Esto significa que los datos de una tabla generalmente deben localizarse en un grupo contiguo al reporte. Al igual que las formas, una excepción a esta regla es que la relación básica de la vista (por ejemplo, la relación CLIENTE de la vista Cliente) puede estar por separado en el reporte. Los grupos de atributos, como teléfono, también se deben localizar juntos y distinguirse de alguna manera.

La figura 10-12 muestra un ejemplo de reporte de la galería View Ridge que lista los datos para cada obra de arte y las transacciones para cada una; calcula el margen total de utilidad por trabajo y por artista, así como el gran total.

➤ FIGURA 10-12

Reporte de lista de ventas

Lista de ventas
15-Nov-01

NombredelArtista	Título		Copia		
Dennis Frings	**South Toward Emerald Sea**		**106/195**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
4/17/1986	$750.00	5/3/1986	Heller, Max	$1,000.00	$250.00
3/15/2000	$1,200.00	5/11/2000	Jackson, Elizabeth	$1,800.00	$600.00
			MargenTotal de South Toward Emerald Sea, Copia 106/195		**$850.00**
			Margen total de utilidad en el caso de Dennis Frings		**$850.00**
Mark Tobey	**Patterns III**		**27/95**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
7/3/1971	$7,500.00	9/11/1971	Cooper, Tom	$10,000.00	$2,500.00
1/4/1986	$11,500.00	3/18/1986	Jackson, Elizabeth	$15,000.00	$3,500.00
9/11/1999	$17,000.00	10/17/1999	Cooper, Tom	$21,000.00	$4,000.00
			Margen Total para Patrones III, Copia 27/95		**$10,000.00**
Mark Tobey	**Rhythm**		**2/75**		
Fechade-Adquisición	**Preciode-Adquisición**	**Fecha-deVenta**	**Vendido a**	**Preciode-Venta**	**MargenTotal-deUtilidad**
4/8/2001	$17,000.00	7/14/2001	Heller, Max	$27,000.00	$10,000.00
			Margen total de utilidad en el caso de Rhythm, Copia 2/75		**$10,000.00**
			Margen total de utilidad en el caso de Mark Tobey		**$20,000.00**
			Gran Total:		**$20,850.00**

La estructura del reporte en la figura 10-12 refleja la estructura del objeto TRABAJO. La sección para cada obra comienza con el nombre de ésta, el cual incluye al artista, el título y la copia. Sigue una sección de renglones de repetición que muestra las transacciones de la obra. Dentro de cada sección el nombre del cliente se ha extraído de la tabla CLIENTE.

Hay que estar conscientes de que con la mayoría de los editores de reportes es difícil construir un reporte que siga más de un camino de valores múltiples a través del esquema de la base de datos. El reporte en la figura 10-12 es una materialización de una vista de ARTISTA que sigue el camino desde ARTISTA hasta TRABAJO y luego a TRANSACCIÓN. El diagrama de relación (relationship) de la figura 10-3(e) muestra otra pista —que va de la tabla INTERÉS_ARTÍSTICO_CLIENTE (CUSTOMER_ARTIST_INT) a la de CLIENTEs (CUSTOMER) que están interesados en un artista específico—. Con la mayoría de los editores de reportes será difícil construir un reporte que muestre ambos caminos.

Los reportes con frecuencia tienen atributos de cálculo de datos que no son parte de la vista implícita ni están almacenados en la base de datos. El reporte en la figura 10-12 tiene cálculos para MargenTotaldeUtilidad, margen total de utilidad por Trabajo, margen total de utilidad por Artista y Gran Total. Todos estos valores se calculan al momento en que se produce el reporte.

Si bien estos valores calculados se podrían almacenar en la base de datos, hacerlo casi nunca es una buena idea porque los valores usados para el cálculo pueden cambiar.

Si, por ejemplo, un usuario modifica el PreciodeVenta de una transacción en particular y no calcula nuevamente el MargenTotaldeUtilidad y los totales basados en éste, los valores almacenados serán erróneos. Sin embargo, si se hacen todos los cálculos necesarios mientras se procesan las transacciones actualizadas probablemente dará como resultado un procesamiento inaceptablemente lento. Por lo tanto, los totales como este por lo general se calculan mejor sobre la marcha. Así, las fórmulas para calcularlos se consideran parte de la materialización del reporte y no parte de la vista implícita.

Objetos implícitos

Considere la solicitud: "Imprimir todos los ARTISTAs ordenados por margen total". A primera vista, parece ser una solicitud de impresión de un reporte acerca del objeto ARTISTA. Sin embargo, las palabras *ordenados por*, indican que se debe considerar más de un ARTISTA. De hecho, esta solicitud no está basada en el objeto ARTISTA, sino en el objeto CONJUNTO DE TODOS LOS ARTISTAs. El reporte en la figura 10-12 muestra los datos de múltiples ARTISTAs y de hecho está basado en el objeto CONJUNTO DE TODOS LOS ARTISTAs, y no en el objeto ARTISTA.

La mente humana cambia con tanta rapidez del objeto *OBJETO-A* al objeto *CONJUNTO DE TODOS LOS OBJETOS-A* que normalmente ni siquiera sabemos que ha ocurrido un cambio. Sin embargo, cuando se desarrollan las aplicaciones de la base de datos es importante observar este cambio, porque la aplicación necesita comportarse en forma diferente cuando esto sucede. Considere tres maneras en las que un ordenamiento puede cambiar la naturaleza del objeto base: (1) ordenar por identificador de objeto; (2) ordenar por columnas no identificadoras, y columnas no objeto; y (3) ordenar por atributos contenidos en los atributos de objeto.

ORDENAMIENTO POR IDENTIFICADOR DE OBJETO. Si el reporte se va a ordenar por un atributo que es un identificador de objeto, el objeto verdadero es un conjunto de esos objetos. Así, un reporte de ARTISTA ordenado por NombredelArtista se refiere al objeto CONJUNTO DE TODOS LOS ARTISTAS. Para la mayoría de los productos de escritura de reportes DBMS, un reporte sobre el objeto CONJUNTO DE TODOS LOS X no es más difícil de producir que un reporte acerca del objeto X. Sin embargo, es importante saber que ha ocurrido un cambio en el tipo de objetos.

ORDENAMIENTO POR COLUMNAS NO IDENTIFICADORAS, Y COLUMNAS NO OBJETO. Cuando un usuario necesita un reporte ordenado por un atributo que es un identificador del objeto, es muy probable que en su mente el objeto verdadero sea un tipo de objeto totalmente diferente. Por ejemplo, el usuario desea producir un reporte acerca de ARTISTA ordenado por Nacionalidad. Dicho reporte en realidad es una materialización del objeto NACIONALIDAD, y no del objeto ARTISTA. De manera similar, si el usuario solicita un reporte sobre CLIENTE ordenado por CódigodeÁrea, el reporte en realidad está basado en un objeto llamado REGIÓN-TELEFÓNICA, o algún objeto similar. La figura 10-13 muestra un ejemplo del objeto REGIÓN-TELEFÓNICA.

Los objetos tales como NACIONALIDAD y REGIÓN-TELEFÓNICA son **objetos implícitos;** esto es, su existencia se puede inferir por el hecho de que el usuario solicitó tal reporte. Si tiene sentido para el usuario solicitar algo como un valor de ordenamiento, entonces ese algo debe ser un objeto en su mente, sin importar que esté modelado en la base de datos o no.

➤ FIGURA 10-13

Objeto REGIÓN TELEFÓNICA

ORDENAMIENTO POR ATRIBUTOS CONTENIDOS EN ATRIBUTOS OBJETO. La tercera manera en la cual se pueden ordenar los reportes es por atributos contenidos en los atributos del objeto. Por ejemplo, el usuario podría solicitar un reporte acerca de los TRABAJOs ordenados por FechadeNacimiento de ARTISTA, el cual es un atributo del objeto TRABAJO, y la Fecha de Nacimiento es un atributo contenido en ARTISTA. En este caso, el usuario en realidad está solicitando un reporte acerca de un objeto implícito (digamos, TIEMPO o PERIODO ARTÍSTICO) que contiene muchos ARTISTAS, cada uno de los cuales incluye muchos TRABAJOs en la galería.

Comprender este cambio en los objetos puede facilitar la tarea de desarrollar el reporte. Proceder como si TRABAJO fuera el objeto básico del reporte lo haría lógicamente artificial. Si TRABAJO se considera la base, se deben crear las materializaciones de los objetos TRABAJO que incluyen ARTISTA y FechadeNacimiento para todos los objetos TRABAJO, almacenados en el disco, y luego ordenarlos por FechadeNacimiento. Por otra parte, si este reporte se refiere a un objeto implícito que contiene ARTISTA, el cual a su vez abarca TRABAJO, entonces se pueden crear los objetos ARTISTA y ordenarlos por FechadeNacimiento, y tratar a los objetos TRABAJO como renglones de valores múltiples en cada objeto ARTISTA.

➤ CUMPLIMIENTO DE RESTRICCIONES

Como se observa en la figura 10-1, la tercera función principal de una aplicación de la base de datos es el cumplimiento de restricciones. En muchos casos, el DBMS es más capaz de imponer restricciones que la aplicación, y ésta no es estrictamente una función del programa de aplicación. Sin embargo, nuestra preocupación es describir los tipos de restricciones y cómo se pueden cumplir, sin importar el código de la aplicación o el DBMS que las haga cumplir.

En primer lugar, la razón de que el DBMS sea con frecuencia el mejor lugar para hacer cumplir restricciones es que constituye el punto central a través del cual deben pasar todos los cambios de información. Sin tomar en cuenta la fuente de un cambio de datos (forma, otro programa, importación de un bloque de datos) el DBMS tendrá la oportunidad de examinar y rechazar el cambio, si es necesario. Además, ciertas reglas (la unicidad es una de ellas) pueden requerir la revisión de todos los renglones de la tabla; el DBMS tiene más capacidad para ejecutar esta función que una aplicación. Además, si el DBMS impone una regla, entonces se necesita codificarla sólo una vez. Si las aplicaciones imponen una regla, entonces se necesitará codificar cada nueva aplicación. Esto es una pérdida de tiempo y genera la posibilidad de que los programas de aplicación impongan reglas de manera inconsistente.

Cabe aclarar que no todos los productos DBMS tienen las características y funciones necesarias para hacer cumplir restricciones. A veces es mucho más difícil escribir e instalar el código de aplicación de restricciones en productos DBMS que en un código de aplicación. Además, en algunas arquitecturas (el servidor de transacción de Microsoft, o Microsoft Transaction Server, es una de ellas) el procesamiento es más eficiente si se elimina la verificación de restricciones del servidor de datos. También, hay algunas restricciones que forman parte de la aplicación. Por ejemplo, un usuario de una forma en particular no se puede permitir ciertas acciones o introducir ciertos datos. En este caso, la aplicación tiene más capacidad para imponer la restricción.

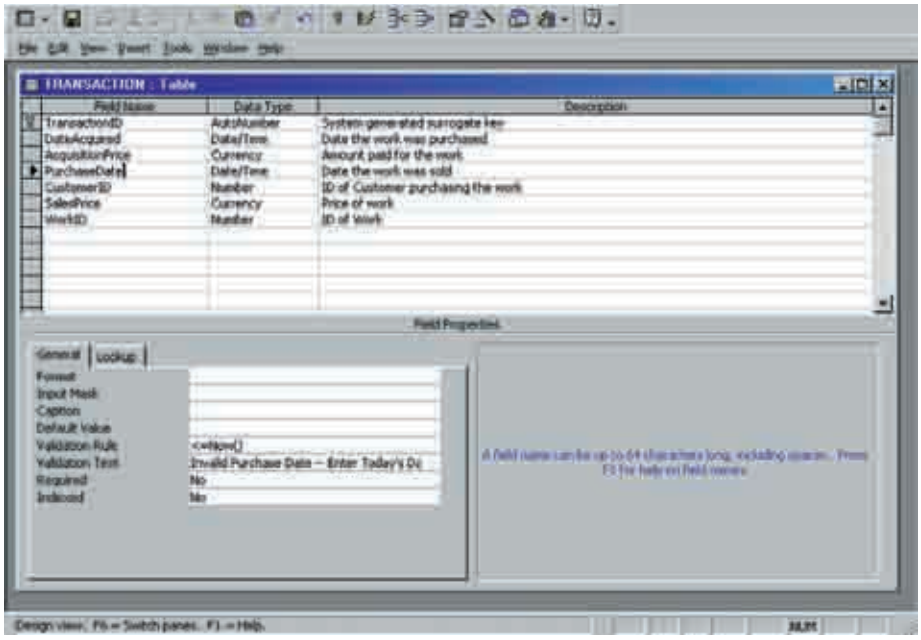
La meta que usted tiene es entender los tipos de restricciones y los medios por los cuales se aplican. Cuando sea posible, hay que imponerlas en el DBMS, cuando no, en la aplicación. Ahora consideraremos cuatro tipos de restricción: dominio, unicidad, integridad referencial y reglas del negocio.

RESTRICCIONES DE DOMINIO

Como establecimos en el el capítulo 4, un dominio es el conjunto de valores que un atributo puede tener. Las definiciones de dominio tienen un componente semántico

➤ FIGURA 10-14

Creación de una restricción de dominio con Access



(el nombre de un artista) y uno físico (texto 25, alfabético). En general, no es posible imponer el componente semántico mediante un proceso automatizado. La industria de la computación aún no está en una etapa de desarrollo tal en que los procesos automatizados puedan determinar que 3M es el nombre de una compañía y no el de un artista. Por lo tanto, actualmente estamos principalmente preocupados por la aplicación de la parte física de una definición de dominio.

Las restricciones de dominio surgen del modelo de datos, como se describió en el capítulo 4. La especificidad de las restricciones de dominio varían ampliamente. En la base de datos de la galería View Ridge, Artista.Nombre puede ser cualquier conjunto de 25 o menos caracteres de texto. Sin embargo, un dominio como el de Copia, es más específico. El formato usado por la galería es nn/mm, donde nn es un número particular de copias y mm es el número total de copias hechas. Así, 5/100 significa la quinta copia de 100. Claramente, un valor como 105/100 no tiene sentido. Por lo tanto, se debe desarrollar el código de validación para asegurar que se siga este formato.

La figura 10-14 muestra un ejemplo de restricción de dominio en Access. La restricción es que PurchaseDate (Fecha de compra) debe ser menor o igual a la fecha del reloj de la computadora en la hora y la fecha en que se introdujo la compra. Observe el renglón etiquetado Validation Rule (regla de validación). La expresión < = Now ( ) define la regla. Si el usuario intenta introducir datos que violen esta regla, se genera una caja de mensaje usando el texto que se introdujo en el siguiente renglón (texto de validación, o Validation Text). La figura 10-15 muestra qué ocurre cuando el usuario intenta ingresar la fecha 10/15/99 cuando la computadora tiene una fecha anterior. En ese caso Access rechaza los datos.

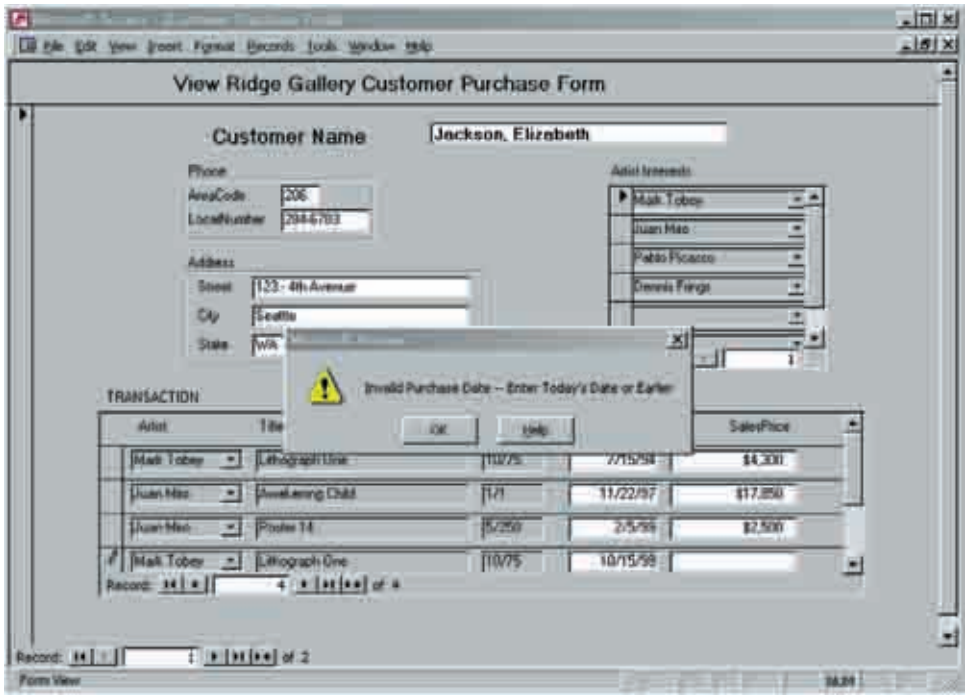
Otro tipo de restricción es si los valores son o no requeridos. Estrictamente hablando, el requerimiento de que se proporcione un valor es una restricción de la tabla, más que una restricción de dominio. Un dominio es un conjunto de valores; si un valor se requiere o no es una pregunta que surge cuando el dominio aparece en una tabla.

La modelación de datos debe indicar si se requieren los valores del atributo. En el modelo de objeto semántico si la cardinalidad mínima de un atributo es 1, entonces se requiere ese atributo. Para imponer esta restricción con Access todo lo que se necesita es establecer la propiedad Required de una columna en Yes. Se hizo lo mismo con Name en la definición de la tabla ARTIST en la figura 10-16. Como verá, se usan diferentes medios con otros productos DBMS.

Las restricciones de valor que se requieren son importantes porque eliminan la posibilidad de valores nulos. Otra forma de hacer esto es definir un valor inicial que el

➤ FIGURA 10-15

Resultado de violar una regla de validación



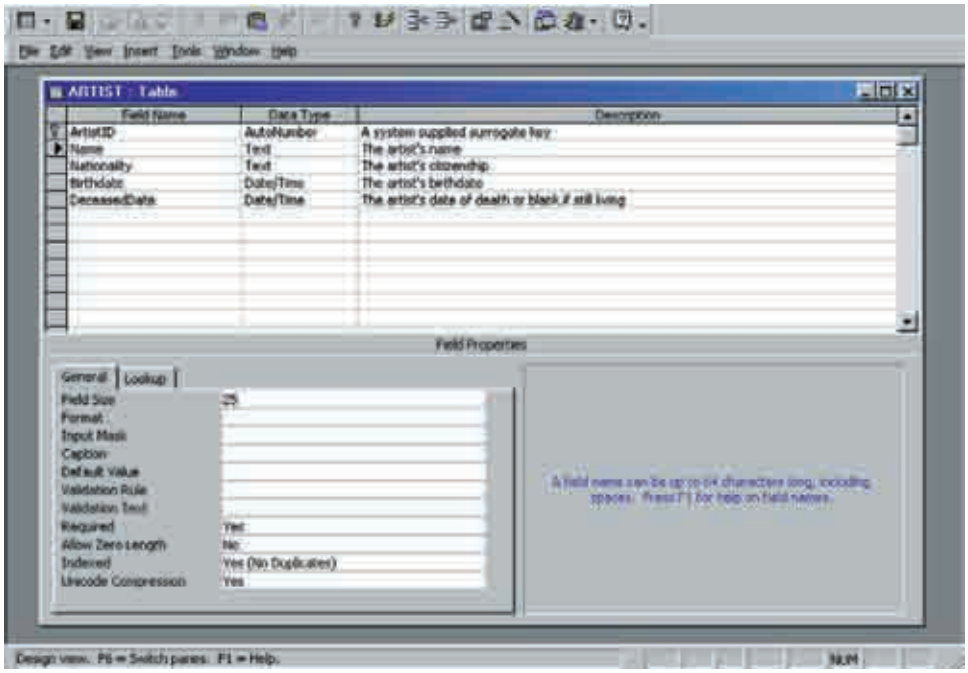
DBMS asigne a la columna cuando se cree el renglón. Con Access esto se hace colocando un valor o expresión en la propiedad Default Value (Valor Predeterminado). Véanse la figura 10-16 y el análisis de los valores nulos en el capítulo 6.

Restricciones de unicidad

La unicidad es el segundo tipo de restricción. Como se estableció, las restricciones de este tipo las aplica mejor el DBMS debido a que puede crear estructuras de datos para efectuar más rápido la verificación de la unicidad. Véase el apéndice A para obtener una descripción de la manera en que se pueden usar los índices con este propósito.

➤ FIGURA 10-16

Definición de la unicidad para Nombre en ARTISTA (Name in ARTIST)



La figura 10-16 muestra cómo Name en ARTIST puede ser definido como único en Access. Observe que Indizar se ha puesto en Yes (No Duplicates). Con esa precisión Access se asegurará de que no se dupliquen los nombres de los artistas que ingresen desde cualquier fuente.

Restricciones de relación

Hay dos tipos de restricciones de relación: **integridad referencial** y **cardinalidad de relación.** La integridad referencial consiste en las restricciones a los valores de las llaves externas. Por ejemplo, las restricciones que surgen de la normalización cuando una relación individual se divide en dos, y eso ocurre entre entidades fuertes y débiles. Las restricciones de cardinalidad de relación ocurren debido a los valores de cardinalidad máxima y mínima en las relaciones. En esta sección consideramos ambos tipos.

RESTRICCIONES DE INTEGRIDAD REFERENCIAL. Todas las restricciones de integridad referencial son restricciones a valores de llaves externas. Considere la primera restricción de integridad referencial en la figura 10-3(d): ArtistaID en TRABAJO debe existir en ArtistaID en ARTISTA. ArtistaID en TRABAJO es una llave externa y la restricción sólo significa que su valor debe concordar con un valor en ArtistaID en la tabla ARTISTA, como ya se explicó.

Para que no exista violación alguna a la composición de la llave externa es necesario que se cumplan las siguientes condiciones: que todas las aplicaciones estén codificadas en forma correcta, que todas las transacciones se procesen adecuadamente y que ningún usuario borre por accidente un artista sin haber eliminado lo que se relaciona con TRABAJO (el administrador debe borrar cualquier TRANSACCIÓN relacionada, conforme a lo que se estipula en la segunda condición y en eliminar todo CLIENTE relacionado, según lo que se establece en la tercera condición). Después de todo, la base de datos comienza sin problemas de integridad; cuando éstos ocurren es que fueron introducidos.

Lo anterior, por supuesto, es esperar demasiado. Los errores suceden; por lo tanto, los productos DBMS proporcionan la facilidad de que estas restricciones de la llave externa las pueda definir e imponer el DBMS. Esto significa que el DBMS no permitirá ninguna inserción, eliminación, o modificación de un valor de llave externa que ocasione una violación a la restricción de una llave externa. Por ejemplo, observe la figura 10-20, donde la marca frente a la opción Enforce Referential Integrity (Imponer Integridad Referencial) significa que se le ha pedido a Access que cumpla la restricción de que EmployeeName (NombredeEmpleado) en:

EMPLOYEE_PhoneNumber (EMPLEADO_NúmerodeTeléfono) exista en Name (Nombre) en EMPLOYEE (EMPLEADO)

Facilidades similares existen para todos los productos DBMS. Además, los productos del servidor DBMS tales como los ORACLE y SQL Server proporcionan utilidades que pueden estar corriendo periódicamente para asegurar que ninguna restricción de llave externa haya sido permitida en la base de datos debido a transacciones parcialmente terminadas, datos importados, u otros misterios del universo. Estas utilidades corren normalmente como parte del plan de mantenimiento de la base de datos (véase el capítulo 11).

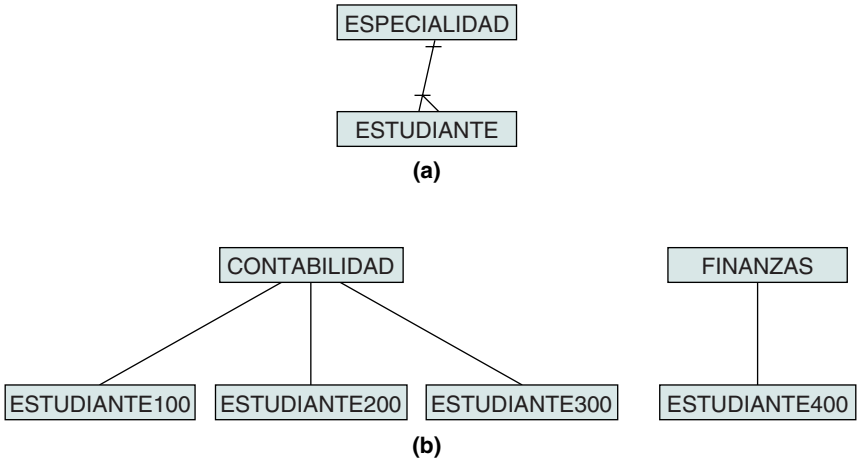
RESTRICCIONES DE CARDINALIDAD DE LA RELACIÓN. Las restricciones de cardinalidad de la relación surgen del establecimiento de la cardinalidad en los atributos de conexión del objeto. Por ejemplo, en la figura 10-3(a) la conexión CLIENTE en TRABAJO. La transacción tiene cardinalidad de 1.1; por lo tanto, una Transacción debe tener una conexión con CLIENTE.

En general, estas restricciones surgen de dos fuentes: establecimiento de no cero en la cardinalidad mínima, o asignaciones de cardinalidad máxima que no son 1 ni N. Así, las cardinalidades de 1.1, 1.N, 2.N ocasionarán que las restricciones de cardinalidad se presenten, como lo harán las cardinalidades de 0.3, 1.4, y 2.4. Con una excepción, estas restricciones se deben imponer mediante un código de aplicación.

La excepción son las cardinalidades 1.1 en el lado hijo de relaciones 1:N. En este caso, la restricción se puede imponer haciendo obligatoria la llave externa. Así, la res-

➤ FIGURA 10-17

Ejemplo de una restricción obligatoria-a-obligatoria: (a) muestra de relación obligatoria-a-obligatoria, y (b) muestra de datos para ésta



tricción 1.1 de CLIENTE en TRABAJO.Transacción se puede imponer haciendo ClienteID obligatoria en la tabla TRANSACCIÓN.

Además de esta salvedad, el programador de la base de datos debe escribir un código para imponer restricciones de cardinalidad. Dicho código se puede colocar en los procedimientos almacenados (stored procedures) para que la invoque el DBMS cuando se hagan cambios de relación, se puede poner en programas de aplicación, o llamarlo durante ciertos eventos de forma, tal como AntesdelaActualización (BeforeUpdate), que analizaremos más adelante.

Para simplificar el análisis sólo consideraremos restricciones 1.1 y 1.N. La lógica para las otras restricciones es una extensión directa de la que se presenta aquí.

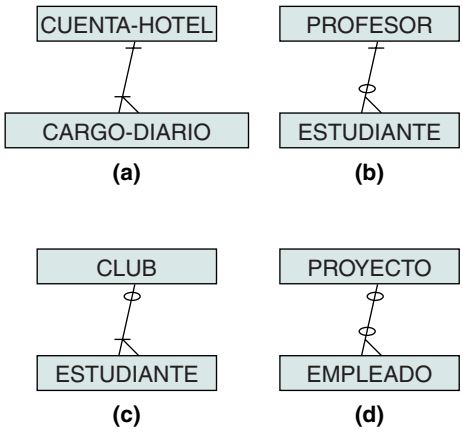
La figura 10-17 describe la relación entre las relaciones ESPECIALIDAD y ESTUDIANTE. Como se muestra en la figura 10-17(a), una ESPECIALIDAD debe tener cuando menos un ESTUDIANTE y un ESTUDIANTE debe tener exactamente una ESPECIALIDAD. Cuando los usuarios actualizan cualquiera de estas relaciones, se debe invocar el código para imponer la restricción. Por ejemplo, en la figura 10-17(b) si un usuario intenta eliminar el renglón de Estudiante 400, el código debe rechazar esa solicitud. Si ésta fuera permitida, el renglón para FINANZAS no tendría un renglón hijo, y la restricción obligatoria sería violada. De manera similar, una nueva ESPECIALIDAD, digamos BIOLOGÍA, no se puede agregar hasta que haya un estudiante que se esté especializando en esa materia.

Un renglón que existe en forma inadecuada sin un padre o hijo con frecuencia se llama **fragmento,** y a los renglones hijos que existen sin un padre obligatorio a veces se les llama **huérfanos.** Una de las funciones de un programa de aplicación es evitar la creación de fragmentos y huérfanos.

Los medios para evitar los fragmentos dependen del tipo de restricción. La figura 10-18 muestra ejemplos de las cuatro restricciones posibles en las relaciones 1:N: obli-

➤ FIGURA 10-18

Ejemplo de los cuatro tipos de restricciones: (a) restricción obligatoria a obligatoria (M-M); (b) restricción obligatoria a opcional (M-O); (c) restricción opcional a obligatoria (O-M), y (d) restricción opcional a opcional (O-O)



gatoria a obligatoria (M-M); obligatoria a opcional (M-O); opcional a obligatorio (O-M), y opcional a opcional (O-O). Estas restricciones se muestran en las relaciones uno a muchos, pero los mismos cuatro tipos se aplican también a otras.

Se pueden violar las restricciones cuando hay cambios en los atributos de la llave. Por ejemplo, en la figura 10-17(b), cambiar la especialidad de Estudiante 300 de CONTABILIDAD a FINANZAS reasigna a ese estudiante al departamento de finanzas. Aunque esto produce un cambio de padre, no viola la restricción.

Sin embargo, dicha violación existirá si la especialidad de estudiante 400 se cambia a CONTABILIDAD. Cuando se hace esto, FINANZAS ya no tiene ningún estudiante, y también se viola la restricción M-M entre ESPECIALIDAD y ESTUDIANTE.

La figura 10-19 presenta reglas para evitar fragmentos en cada uno de estos tipos de restricciones. La figura 10-19(a) muestra acciones sobre el renglón padre, y la figura 10-19(b), acciones en los renglones hijos. Como estas figuras lo indican, las posibles acciones son insertar nuevos renglones, modificar los datos llave y eliminar renglones. La

➤ FIGURA 10-19

Reglas para evitar fragmentos: (a) condiciones para permitir cambios en los registros padres, y (b) condiciones para permitir cambios en los registros hijos

Acción propuesta en padre

Tipo de relación	*Insertar*	*Modificar (llave)*	*Eliminar*
M-M	Crear cuando menos un hijo	Cambiar llaves coincidentes de todos los hijos	Eliminar todos los hijos o reasignar todos los hijos
M-O	OK	Cambiar llaves coincidentes de todos los hijos	Eliminar todos los hijos o reasignar todos los hijos
O-M	Insertar nuevo hijo o Existe hijo apropiado	Cambiar la llave de cuando menos un hijo o Existe hijo apropiado	OK
O-O	OK	OK	OK

(a)

Acción propuesta en hijo

Tipo de relación	*Insertar*	*Modificar (llave)*	*Eliminar*
M-M	Existe padre o Crear padre	Existe padre con un nuevo valor (o crear uno) y Existe hermano	Existe hermano
M-O	Existe padre o Crear padre	Existe padre con un nuevo valor o Crear padre	OK
O-M	OK	Existe hermano	Existe hermano
O-O	OK	OK	OK

(b)

figura 10-19 lista las reglas para relaciones uno a muchos; las reglas para relaciones uno a uno son similares.

RESTRICCIONES PARA ACTUALIZAR LOS RENGLONES PADRE. El primer renglón de la figura 10-19(a) se refiere a restricciones M-M. Sólo se puede insertar un nuevo renglón padre si al mismo tiempo se creó cuando menos un renglón hijo, lo cual se puede hacer insertando un nuevo renglón hijo o reasignando un hijo de un padre diferente (no obstante, esta ultima acción puede ocasionar en sí misma una violación a la restricción).

En una relación M-M sólo se permite un cambio de la llave de un padre si también se modifican los nuevos valores correspondientes a la llave externa en los renglones hijo (es posible reasignar todos los hijos a otro padre y después crear cuando menos un nuevo hijo para el padre, pero esto se hace muy rara vez). Así, se permite cambiar la factura en CUENTA-HOTEL siempre y cuando también se cambie Factura en todos los renglones CARGO-DIARIO. Observe que si se usan las llaves sustitutas esta acción nunca ocurrirá.

Por último, un padre de una relación M-M se puede eliminar siempre y cuando todos los hijos también sean eliminados o reasignados.

En lo que se refiere a las restricciones M-O, se puede agregar un nuevo padre sin ninguna restricción, debido a que no necesitan tener hijos. Para la relación en la figura 10-18(b) se puede agregar un nuevo renglón PROFESOR sin restricción. Sin embargo, sólo se permite un cambio en el valor de la llave padre si los valores correspondientes en los renglones hijos también se cambian. Si un PROFESOR en la relación de la figura 10-18(b) cambia su llave, también se debe alterar el valor del Asesor en todos los renglones de los estudiantes que asesora el profesor.

Finalmente, en una relación con una restricción M-O, el renglón padre se puede suprimir sólo si todos los hijos son eliminados o reasignados. Para la relación PROFESOR-ESTUDIANTE, probablemente todos los renglones de estudiante serían reasignados.

Para las restricciones O-M, sólo puede insertarse un padre si al mismo tiempo se agrega cuando menos un hijo, o si éste ya existe. Por ejemplo, para la relación O-M entre CLUB y ESTUDIANTE de la figura 10-18(c) se puede agregar un nuevo club sólo si se puede crear un renglón ESTUDIANTE (ya sea agregando un nuevo estudiante, o cambiando el valor de CLUB en un ESTUDIANTE). Alternativamente puede existir un renglón adecuado de estudiante.

De manera similar, la llave del padre en una relación O-M sólo se puede cambiar si se crea un hijo o si ya existe un renglón hijo adecuado. Esto es, el Club de Esquí puede cambiar su nombre por el de Buceo sólo si cuando menos un esquiador está dispuesto a unirse a Buceo, o si un estudiante se ha inscrito ya en Buceo. No hay restricciones para la eliminación de un renglón padre en una relación O-M.

En la figura 10-18(d) se muestra el último tipo de restricción de relación: O-O. No existe ninguna restricción a las actualizaciones en los renglones de una relación O-O. Los renglones PROYECTO y EMPLEADO se pueden actualizar conforme sea necesario.

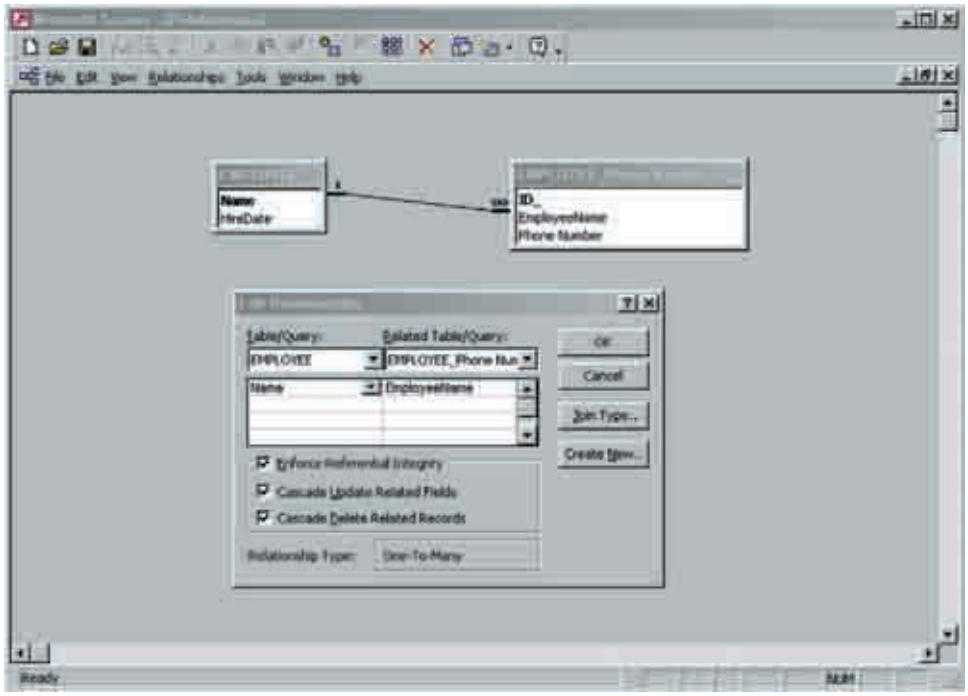
RESTRICCIONES A LA ACTUALIZACIÓN DE RENGLONES HIJO. Las reglas para evitar fragmentos cuando se actualizan renglones hijo se muestran en la figura 10-19(b) y son similares a las de la figura 10-19(a). La única diferencia notable es que, en varios casos, los renglones hijo se pueden modificar o eliminar siempre y cuando existan renglones hermano. Por ejemplo, en una restricción M-M, un renglón hijo se puede eliminar siempre y cuando existan hermanos. (¡El último hijo nunca se va de casa!) Para la restricción M-M de la figura 10-18(a), se puede eliminar un renglón particular CARGO-DIARIO siempre y cuando quede cuando menos uno de ellos.

Con excepción de las consideraciones relacionadas con los hermanos, las reglas para evitar fragmentos cuando se procesan renglones hijo son similares a las de los padres. Asegúrese de entender cada enunciado de la figura 10-19(b).

USO DEL DBMS PARA IMPONER RESTRICCIONES DE CARDINALIDAD. Considerando el análisis anterior es útil tomar en cuenta las restricciones en la figura 10-19 en el contexto de las facilidades de definición de la relación en Access. Suponga que

➤ FIGURA 10-20

Propiedades de relación en Access



una base de datos tiene un objeto EMPLEADO con múltiples números de teléfono. Para mostrar la relevancia de las columnas al centro de la figura 10-19 suponga que la llave de EMPLOYEE (EMPLEADO) no es sustituta, sino EMPLOYEE.Name (EMPLEADO.Nombre).

La figura 10-20 muestra la ventana de definición de relación para este ejemplo. La marca en la caja Enforce Referential Integrity (Imponer Integridad Referencial) indica que Access no permitirá crear un nuevo renglón EMPLOYEE_PhoneNum, a menos que el valor de EmployeeName ya esté presente en un renglón en Employee. Éste es el significado de integridad referencial, como se analizó previamente. La segunda marca, en Cascade Update Related Fields (Campos relacionados de actualización en cascada), significa que si se cambia el nombre de un empleado en la tabla Employee, entonces ese cambio se propagará a todos los renglones relacionadas con EMPLOYEE_PhoneNum. Esto impone las reglas en las columnas de enmedio de la figura 10-19. (Nuevamente, esto no sería necesario si la llave sustituta se usara en su lugar.)

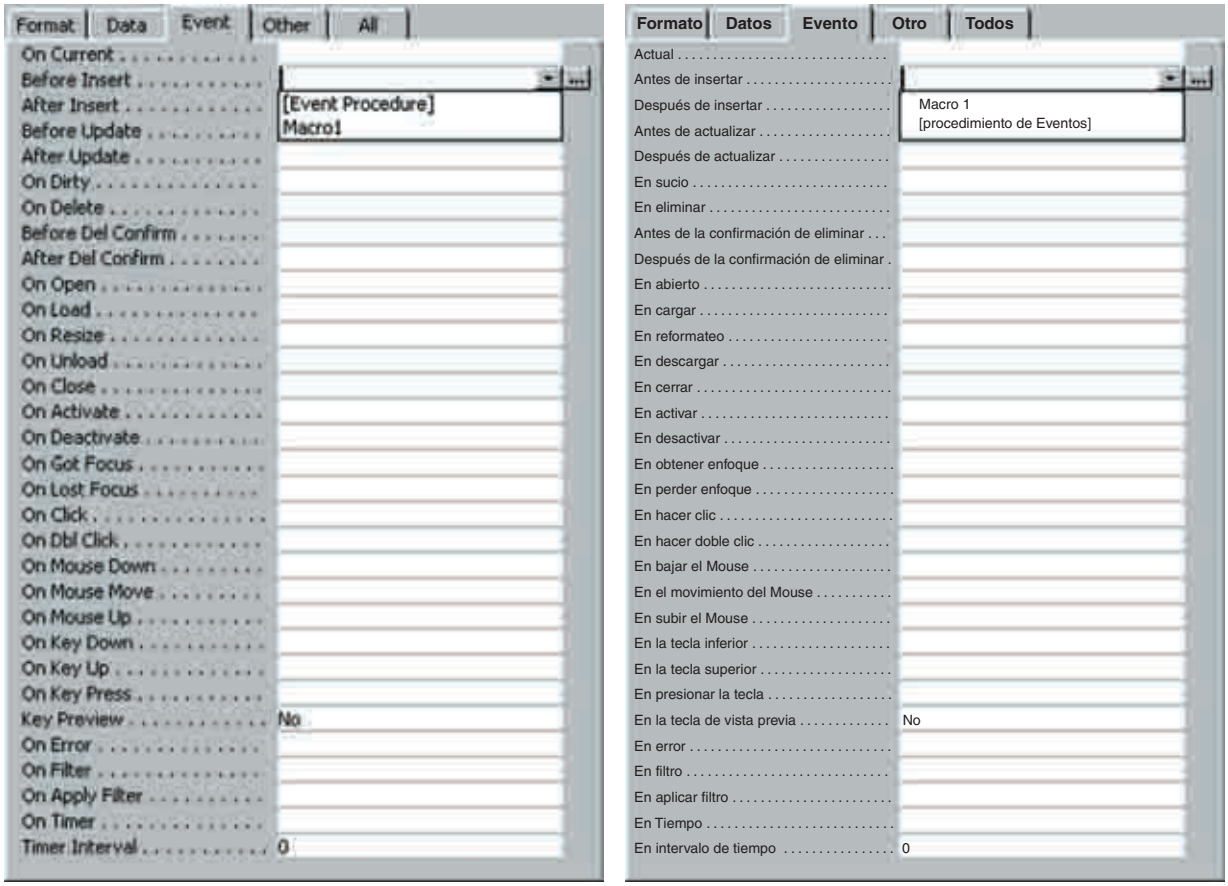
La última marca, en Cascade Delete Related Records (Registros relacionados de eliminación en cascada), significa que cuando se borra un renglón EMPLOYEE se eliminarán también todos los que están conectados con EMPLOYEE_PhoneNum. Esto es similar a lo que se analizó anteriormente en la sección de eliminación de instancias de vistas.

Esta característica de Access direcciona las reglas en la segunda y tercera columnas de la figura 10-19(a) y las de la primera y segunda columnas de la figura 10-19(b). No direcciona las reglas en la primera columna de la figura 10-19(a), tampoco en la tercera columna de la figura 10-19(b). Esas reglas necesitan ser codificadas en procedimientos almacenados (stored procedures) o en la aplicación.

Restricciones de las reglas del negocio

Las restricciones de las reglas del negocio son específicas para la lógica y los requerimientos de una aplicación determinada. Surgen de los procedimientos y las políticas que existen en la empresa los cuales usará la aplicación de la base de datos. Ejemplos de reglas del negocio en una aplicación de ventas son:

➤ FIGURA 10-21 *Eventos para formas de Access*



- Ningún cheque de comisión puede exceder el 50% del total de la "bolsa de comisión"
- No se generará ninguna cancelación de pedido si el valor total de los artículos es menor a $200
- Los costos de envío no se le cobran a los clientes distinguidos
- Los vendedores no pueden crear pedidos para ellos mismos
- Para que un empleado sea gerente de ventas primero debe ser vendedor

Debido a que las reglas del negocio son dependientes de la aplicación, no existen características genéricas de los productos DBMS para imponerlas. Más bien, los productos DBMS proporcionan medios para insertar un código antes o después de los eventos más importantes. La figura 10-21 muestra una lista de los eventos que pueden ser capturados en las formas de Access. En esta figura, el programador está en el proceso de añadir lógica al evento Antes de insertar (Before Insert). La lógica puede tener una de varias formas: un procedimiento del evento en el lenguaje de expresión de Access, o de Visual Basic (u otro lenguaje de programación), o un macro de Access. Todos los datos en la forma y en la base de datos son accesibles para el procedimiento del evento o macro. Así, se pueden imponer cualesquiera de las reglas de la lista de viñetas mediante los acontecimientos que se capturan.

Con los productos servidores DBMS tales como ORACLE y SQL Server, se usa un medio similar. La lógica puede ser codificada en disparadores (trigger), los cuales son procedimientos almacenados (stored procedures) que se invocan cuando tienen lugar eventos en la base de datos. Los eventos que se pueden capturar son similares a los que se muestran en la figura 10-21.

➤ SEGURIDAD Y CONTROL

La cuarta función principal de una aplicación de base de datos listada en la figura 10-1 es proporcionar mecanismos de seguridad y control. El objetivo es crear aplicaciones en las que sólo los usuarios autorizados puedan realizar actividades apropiadas en el tiempo correcto.

SEGURIDAD

La mayoría de los productos DBMS proporcionan nombre del usuario y contraseña de seguridad. Una vez que el usuario se registra, se puede limitar el acceso a ciertas formas, reportes, tablas e incluso a columnas de tablas. Esto es tan apropiado y útil como se quiera. Sin embargo, no ayuda a limitar los datos que los usuarios pueden ver.

Por ejemplo, en la aplicación de la base de datos de recursos humanos cada empleado sólo debería poder ver sus propios registros. Ciertos empleados de recursos humanos deberían poder consultar alguna información acerca de todos los empleados, y los gerentes de recursos humanos deberían poder ver toda la información acerca de todos los empleados.

Limitar el acceso a ciertas formas y reportes no ayuda. Cada empleado necesita ver la Forma del Empleado; la restricción necesita ser que el empleado sólo pueda ver la información de la forma que le pertenece (con las excepciones señaladas). Algunas veces usted escuchará los términos **seguridad horizontal** y **seguridad vertical.** Para entenderlos, piense en una tabla. La seguridad vertical limitaría el acceso a ciertas columnas, pero se podrían ver todos los renglones. La seguridad horizontal limitaría el acceso a ciertos renglones, pero se podrían ver todas las columnas.

Las aplicaciones que limitan a los usuarios a ciertas formas, reportes, tablas, o columnas proporcionan seguridad vertical. Las que limitan a los usuarios a ciertos datos en formas, reportes, tablas o columnas proporcionan seguridad horizontal. Los nombres de usuarios y las contraseñas se pueden usar con facilidad para proporcionar seguridad vertical. La seguridad horizontal generalmente requiere que el programador escriba un código de aplicación.

Por ejemplo, para proporcionar seguridad horizontal en la aplicación de empleado, el código de aplicación obtendría el nombre del usuario del sistema de seguridad del DBMS y limitaría el acceso a los renglones que contienen el nombre, o que están ligadas a renglones con ese nombre a través de joins. Una manera de hacer esto es anexar el nombre del usuario como una cláusula WHERE en los enunciados SQL.

Debido a que cada situación es diferente, no podemos decir más. Sólo esté consciente de que cuando los productos DBMS dicen que apoyan la seguridad, con frecuencia sólo significa que tienen seguridad vertical vía el nombre del usuario y la contraseña.

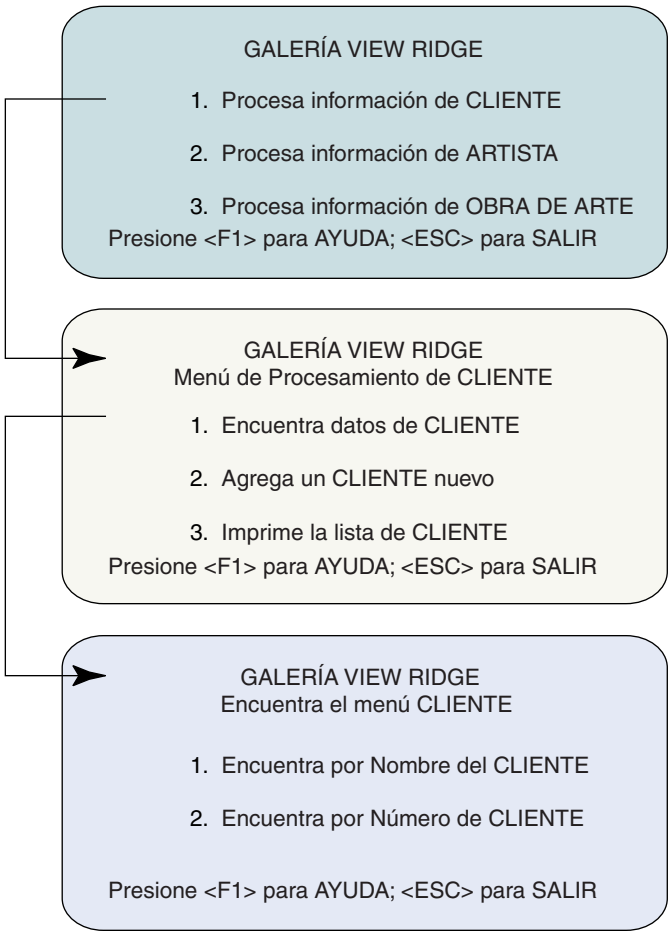
CONTROL

La mayoría de las aplicaciones de base de datos proporcionan control a través de menúes. La figura 10-22(a) muestra un sistema de menúes para una aplicación pre-GUI, y la figura 10-22(b) muestra los mismos menúes para una aplicación GUI.

Los menúes en la figura 10-22 son estáticos. El control más eficaz se puede proporcionar cambiando dinámicamente el contenido del menú conforme el usuario cambia el contexto. Puede ver menúes de ese tipo en Access cuando la barra de herramientas cambia, dependiendo de si usted está en las herramientas de definición de la tabla, de definición de la forma, o de definición del reporte. Los programadores de aplicación de la base de datos pueden usar una estrategia similar para cambiar las opciones de menú, dependiendo de la forma o reporte que un usuario esté viendo, e incluso de la acción que el usuario esté llevando a cabo en la forma. Con el uso de Access, un programador puede cambiar opciones de menú cuando captura eventos como los que se muestran en la figura 10-21, y reestructurar drásticamente las opciones del menú.

➤ FIGURA 10-22

Jerarquía de menúes de la galería View Ridge: (a) sin usar una interfaz de usuario gráfica, y (b) usando una interfaz de usuario gráfica



(a)

(b)

Un tipo diferente de control es el que se refiere a las transacciones. En el capítulo 11 aprenderá los medios de control del procesamiento multiusuarios para que las acciones de uno no tengan consecuencias inapropiadas en las acciones de otro. Una parte clave del control multiusuarios es identificar las fronteras de trabajo que se deben completar como unidad, a veces se les llama **fronteras de transacción.** Por ejemplo, la serie de enunciados SQL para crear la vista que se muestra al principio de este capítulo se necesita completar como una sola transacción.

No anticiparemos el análisis en ese capítulo, sólo diremos que especificar las fronteras de la transacción es trabajo de la aplicación. Por lo general esto se lleva a cabo mediante la ejecución de un enunciado como BEGIN TRANSACTION al comienzo de una unidad de trabajo y END TRANSACTION cuando el trabajo concluye.

Establecer fronteras es fácil si las vistas del usuario están bien diseñadas: las instrucciones de la aplicación BEGIN TRANSACTION al inicio de la vista y END TRANSACTION al final de la vista. Continuaremos con este análisis en el capítulo 12.

➤ LÓGICA DE LA APLICACIÓN

La última función de la aplicación de una base de datos que se lista en la figura 10-1 es ejecutar la lógica de la aplicación. Este tema por lo general se analiza en las clases de desarrollo de sistemas y en diversos textos, y por eso aquí diremos muy poco al respecto.

Las necesidades de la lógica de aplicación surgen de los requerimientos de los sistemas. En un sistema de registro de pedidos la lógica de la aplicación se refiere a cómo llevar inventarios de almacén, cómo encargarse de las existencias insuficientes en inventario, cómo programar un pedido cancelado y actividades similares. En una aplicación de base de datos basada en una forma como la de la galería, el código para avalar esa lógica se relaciona con eventos como los que se muestran en la figura 10-21.

Para otras aplicaciones de la base de datos, donde las formas se materializan mediante la aplicación, en lugar de que sea por medio del DBMS (común en las aplicaciones de macrocomputadoras), la aplicación procesa los datos en una forma similar a la del procesamiento de archivos. La lógica se codifica en un renglón dentro de la aplicación conforme recibe los datos y los pone en el DBMS. Algunos programas de aplicación reciben información de otros; en este caso, también la lógica de la aplicación está contenida en línea.

Así, los medios por los cuales la lógica de la aplicación se implanta en las aplicaciones de la base de datos depende tanto de la lógica como del medio ambiente. Se usan diferentes medios para las aplicaciones de escritorio, servidor del cliente, macrocomputadora, y tecnología de Internet. Usted ha visto cómo capturar hechos en las aplicaciones de escritorio. En los siguientes capítulos consideraremos otros medios.

➤ RESUMEN

Las cinco funciones principales de la aplicación de una base de datos son: (1) crear, leer, actualizar y eliminar las vistas (CRUD), (2) materializar o formatear vistas, (3) imponer restricciones, (4) proporcionar mecanismos de seguridad y control, y (5) ejecutar la lógica del negocio.

Una vista es una lista estructurada de elementos de datos de las entidades u objetos semánticos en el modelo de datos. Una instancia de vista es una que está llena de datos. Debido a que las vistas están estructuradas, un elemento de datos puede aparecer más de una vez. Para leer una vista se emiten uno o más enunciados SQL con el fin de obtener valores de datos. Se requerirá más de un enunciado SQL si la vista incluye dos o más caminos de relaciones a través del esquema. Un recordset es una relación con un empaquetador del objeto de programación.

Crear una vista requiere almacenar uno o más nuevos renglones en tablas, y posiblemente crear o cambiar valores de llaves externas, así como establecer relaciones. Hay tres tipos de actualizaciones: cambiar los datos existentes, cambiar las relaciones, y crear nuevos renglones para los atributos de multivaluados. Eliminar una vista implica borrar uno o más renglones y ajustar las llaves externas. El problema cuando se elimina es saber qué tanto se debe descartar. Hay que suprimir las entidades débiles si la entidad de la que dependen se elimina. También hay que quitar los atributos multivaluado dentro de un objeto semántico. Con algunos productos DBMS las relaciones se pueden marcar para la eliminación en cascada, de tal forma que el DBMS remueva los renglones dependientes que sea apropiado.

Una forma es una pantalla desplegada que se usa para introducir datos y editarlos. Los principios del diseño de formas son que la estructura de ésta debe reflejar la de la vista, la semántica de los datos debe ser gráficamente evidente, y la estructura de la forma debe fomentar la acción apropiada. Las cajas de lista desplegables, los botones de opción y las cajas de verificación se pueden usar para aumentar la utilidad de las formas.

Los reportes también se deben diseñar de tal forma que su estructura refleje la de la vista que materializan. El ordenamiento de los reportes con frecuencia implica la

existencia de otros objetos. Con la mayoría de los editores de reportes es difícil construir reportes que sigan más de una pista multivaluada a través del esquema. Los reportes con frecuencia calculan atributos de datos; por lo general es mejor no almacenar esos atributos en la base de datos.

Las restricciones se pueden imponer ya sea mediante el DBMS o a través de un programa de aplicación. En la mayoría de los casos es mucho mejor para el DBMS aplicarlas cuando sea posible, principalmente debido a que el DBMS es un punto central a través del cual pasan todos los cambios de datos. En algunos casos, el DBMS no tiene las características para imponer restricciones; sin embargo, se deben imponer mediante la aplicación. Las restricciones de dominio imponen la parte física de las definiciones de dominio. Los valores requeridos son otro tipo de restricción. Tener un valor requerido evita la ambigüedad de los valores nulos. Las restricciones de unicidad se aplican mejor con el DBMS; por lo general se hace construyendo índices.

Hay dos tipos de restricciones de relación: integridad referencial y cardinalidad de relación. Las restricciones de integridad referencial se aplican mejor por medio del DBMS. Las restricciones de cardinalidad de relación surgen del establecimiento de la cardinalidad en los vínculos del objeto, de cualquier establecimiento de cardinalidad mínima que no sea cero, o de una cardinalidad máxima que no sea 1 ni N. Excepto en el caso de la cardinalidad 1.1 en el lado de los hijos de las relaciones 1:N, estas restricciones se deben imponer mediante un código de aplicación. Las reglas para la aplicación de restricciones en las relaciones 1:N se resumen en la figura 10-19. Las restricciones de la regla de negocio mediante un código de aplicación que se invoca a través de eventos capturados, de disparadores (triggers), o de programas de aplicación en línea.

La mayoría de los productos DBMS proporcionan el nombre del usuario y la contraseña de seguridad. Esto se usa para proporcionar seguridad vertical; la seguridad horizontal se debe proveer mediante el código de aplicación. La mayoría de las aplicaciones ofrecen control a través de menúes. El mejor control ocurre cuando se cambian los menúes conforme cambia el contexto del usuario. Los programas de aplicación desempeñan un papel importante en la definición de las fronteras de transacción. La lógica de la aplicación se codifica y se invoca capturando eventos, así como a través de otros medios que serán explicados en los siguientes capítulos.

➤ PREGUNTAS DEL GRUPO I

10.1 Enumere las cinco funciones principales de una aplicación de la base de datos.

10.2 Explique el significado del acrónimo CRUD.

10.3 Defina el término *vista* como se utilizó en este capítulo.

10.4 ¿Qué es una instancia de vista?

10.5 Explique en qué difiere una vista de una materialización.

10.6 ¿Un atributo puede aparecer más de una vez en una vista? ¿Por qué?

10.7 ¿Bajo qué condiciones se puede leer una vista con un enunciado SQL?

10.8 ¿En qué casos leer una vista requiere más de un enunciado SQL?

10.9 Explique los dos caminos que existen en la vista de Cliente en la figura 10-4.

10.10 Defina el término *recordset*.

10.11 Describa en términos generales el trabajo que se requiere cuando se crea una instancia de vista.

10.12 ¿Cómo se crean las nuevas relaciones cuando se desarrolla una instancia de vista?

10.13 ¿Qué técnica se puede usar para obtener el valor de una llave sustituta cuando se insertan nuevos renglones en una tabla?

10.14 Liste los tres tipos de cambio que pueden ocurrir cuando se actualiza una instancia de vista.

10.15 Explique cómo cambiar las relaciones 1:N, y las relaciones N:M.

10.16 ¿Cuál es la dificultad principal cuando se escribe un código para eliminar una instancia de vista?

10.17 ¿Cómo puede ayudar un modelo E-R a determinar qué tanto se debe eliminar?

10.18 ¿Cómo puede ayudar un objeto semántico a determinar qué tanto se debe suprimir?

10.19 ¿Qué son las eliminaciones en cascada y por qué son importantes?

10.20 Explique el enunciado "la estructura de la forma debería reflejar la estructura de la vista".

10.21 ¿Cómo se pueden diseñar las formas para hacer la semántica de la información gráficamente evidente?

10.22 ¿Cómo se pueden diseñar las formas para fomentar la acción apropiada?

10.23 Explique el papel de las cajas de listas desplegables, los grupos de opción y las cajas de verificación en el diseño de formas.

10.24 ¿Qué limitación existe para la materialización de reportes de las vistas?

10.25 Explique por qué los valores calculados en reportes por lo regular no deberían almacenarse en la base de datos.

10.26 Explique cómo el requerimiento de objetos de reporte ordenados por un valor cambia el objeto implícito del reporte.

10.27 ¿Por qué las restricciones normalmente deberían ser impuestas por el DBMS y no mediante un programa particular de formas, reportes o aplicación?

10.28 ¿Por qué a veces las restricciones se imponen en programas de aplicación?

10.29 Dé un ejemplo sobre una restricción de dominio y explique cómo se puede imponer con Access.

10.30 Describa la ambigüedad que surge cuando los valores son nulos, así como dos maneras para que éstos valores puedan ser eliminados.

10.31 ¿Por qué el DBMS es el que normalmente debería imponer restricciones de unicidad?

10.32 Describa las dos fuentes de restricciones de cardinalidad.

10.33 Nombre dos tipos de restricción de relación.

10.34 ¿Cuál es la mejor manera de imponer restricciones sobre valores de llave externa?

10.35 ¿Cómo se puede imponer una restricción de cardinalidad 1.1 en el lado de los hijos de una relación 1:N?

10.36 Defina *fragmento* y *huérfano*.

10.37 Explique las entradas en la primera columna de la figura 10-19(a).

10.38 Explique por qué la columna central de la figura 10-19(a) es innecesaria cuando se usan las llaves sustitutas.

10.39 Explique las entradas en la tercera columna de la figura 10-19(a).

10.40 Explique las entradas en la primera columna de la figura 10-19(b).

10.41 Explique las entradas en la tercera columna de la figura 10-19(b).

10.42 Explique por qué la primera columna en la figura 10-19(a) y la tercera columna en la figura 10-19(b) no están impuestas mediante las propiedades de relación de Access que se muestran en la figura 10-20.

10.43 Dé un ejemplo sobre restricción de una regla del negocio que se podría aplicar al modelo de datos de la figura 10-3. Explique cómo se puede imponer esta restricción al capturar un evento.

10.44 Defina seguridad *horizontal* y *vertical*.

10.45 ¿Qué tipo de seguridad se avala con el nombre del usuario y la contraseña?

10.46 ¿Qué tipo de seguridad se debe apoyar mediante un código de aplicación?

10.47 Explique por qué los menúes dinámicos son mejores que los estáticos.

10.48 ¿Cómo es la lógica de negocios conectada a una base de datos cuando se usa Access?

➤ PREGUNTAS DEL GRUPO II

Las preguntas 10.49 a la 10.52 pertenecen a la siguiente vista de Artista, la cual está basada en el modelo de datos de la figura 10-3.

ARTISTA.Nombre
ARTISTA.Nacionalidad
TRANSACCIÓN.FechadeCompra
TRANSACCIÓN.PreciodeVenta. . .
CLIENTE.Nombre
CLIENTE.Teléfono.CódigodeÁrea
CLIENTE.Teléfono.NúmeroLocal
CLIENTE.Nombre. . .

Las elipsis (. . .) se refieren a estructuras que se pueden repetir.

10.49 Codifique los enunciados SQL para leer la instancia de "Mark Tobey" en esta vista.

10.50 Codifique los enunciados SQL para crear una nueva instancia de esta vista. Suponga que usted tiene datos para ARTISTA, una TRANSACCIÓN y muchos CLIENTE.Nombre(s) para la segunda instancia de CLIENTE.Nombre. Suponga que estos datos se localizan en una estructura llamada NuevoArtista. Use sintaxis similar a la del texto.

10.51 Codifique los enunciados SQL para actualizar esta vista como sigue:

a. Cambie la ortografía de Mark Tobey a Mark Toby

b. Cree una nueva Transacción para Mark Toby. Suponga que tiene los datos necesarios de la transacción, obra y cliente en una estructura llamada NuevaTransacción

c. Agregue nuevos clientes interesados en Mark Toby. Suponga que se encuentran almacenados en un conjunto al que usted puede tener acceso con el enunciado "Para cada NuevoCliente.Nombre"

10.52 Codifique los enunciados SQL para eliminar el renglón de Mark Toby y todos los renglones TRABAJO y TRANSACCIÓN relacionados.

➤ PROYECTOS

A. Usando Access, cree la base de datos que se muestra en la figura 10-3. Cree una forma para la vista de Artista que se muestra en la pregunta 10.49. Justifique el diseño de su forma usando los principios de este capítulo. Sugerencia: puede utilizar un asistente (wizard) para crear uno de los subformularios, pero necesitará agregar el segundo de forma manual. También, agregue las cajas de conjunto manualmente después de que haya creado las formas para la subforma.

B. Termine el Proyecto A al final de los capítulos 3 y 4, si aún no lo ha hecho.

1. Liste y describa el propósito de las tres vistas, las tres formas y los tres reportes que considere serán necesarios para esta aplicación.
2. Muestre la estructura de un menú GUI desplegable para esta aplicación. Usando su modelo, diseñe una de las formas para introducir nuevas propiedades. Expli-

que qué tipo de control (caja de texto, lista desplegable) se usa para cada campo. Justifique la estructura usando los conceptos que presentamos en este capítulo.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Lea el proyecto FiredUp al final del capítulo 9. Use las cuatro tablas que se describen ahí para resolver lo siguiente:

A. Construya las siguientes vistas. Use la figura 10-4 como ejemplo.

1. Construya una vista que comience con ESTUFA y contenga todas las tablas y los datos. Llame a la vista ESTUFA_VISTA.
2. Construya una vista que inicie en CLIENTE e incluya todas las tablas y los datos. Llame a la vista CLIENTE_VISTA.
3. Construya una vista que dé comienzo en REGISTRO y contenga todas las tablas, excepto ESTUFA_REPARACIÓN. Llame a la vista REGISTRO_VISTA.
4. Construya una vista que inicie con ESTUFA_REPARACIÓN y contenga todas las tablas y los datos. Llame a la vista ESTUFA_REPARACIÓN_VISTA.

B. Construya enunciados SQL para procesar vistas como sigue. Use el comienzo de SQL de la página 262 como ejemplo.

1. Muestre los enunciados SQL necesarios para leer ESTUFA_VISTA. Suponga que inicia con un NúmerodeSerie en particular.
2. Muestre los enunciados SQL que se necesitan para construir una nueva instancia de REGISTRO_VISTA. Suponga que los datos de estufa que se necesitan ya están en la base de datos, pero falta la información necesaria del cliente.
3. Muestre los enunciados SQL para construir una nueva instancia de ESTUFA_REPARACIÓN. Suponga que los datos de ESTUFA están en la base de datos, pero falta de información de CLIENTE. Registre la ESTUFA mientras anota la reparación.
4. Muestre los enunciados SQL para eliminar todos los registros referentes a una estufa en particular. Use la vista de ESTUFA.

PARTE V

PROCESAMIENTO DE BASES DE DATOS MULTIUSUARIO

Los tres capítulos que conforman la parte V describen puntos importantes y problemas de las bases de datos multiusuario y muestran respuestas y soluciones mediante dos productos DBMS (Database management system, por sus siglas en inglés) populares. En el capítulo 11 describimos la administración de la base de datos, así como las principales tareas y técnicas para administrar una base de datos multiusuario. Los dos capítulos siguientes ilustran la implementación de estos conceptos, con el uso de Oracle 8i en el capítulo 12, y con el SQL Server 2000 en el capítulo 13.

CAPÍTULO 11

Administración de bases de datos multiusuario

Si bien es cierto que las bases de datos multiusuario son de gran valor para las organizaciones que las crean y las usan, también es verdad que les plantean problemas difíciles. Para unas, la base de datos multiusuario es complicada de diseñar y desarrollar debido a que maneja muchas consultas de usuarios al mismo tiempo. Además, los requerimientos cambian con el tiempo y dichos cambios necesitan otros cambios en la estructura de la base de datos. Estos cambios de estructura deben ser cuidadosamente planeados y controlados para que un cambio hecho para un grupo no cause problemas en otro. Además, cuando los usuarios procesan una base de datos con frecuencia se necesitan controles especiales para asegurar que el trabajo de un usuario no afecte inapropiadamente el de otro. Como verá, este tema es importante y complicado.

En organizaciones grandes se necesita definir e imponer los derechos de procesamiento y las responsabilidades. Por ejemplo, ¿qué sucede cuando un empleado deja la empresa? ¿Cuándo se pueden eliminar sus registros? Si se trata de la elaboración de la nómina, tendrá que ser después del último periodo de pago; en el caso del reporte trimestral, al final del periodo; con respecto a los impuestos, cuando termine el año, y así sucesivamente. Es claro que ningún departamento puede decidir en forma unilateral cuándo eliminar la información. Se pueden hacer comentarios similares con respecto a la inserción y al cambio de valores de los datos. Por esta y otras razones es necesario desarrollar sistemas de seguridad para permitir que sólo los usuarios autorizados lleven a cabo acciones permitidas dentro de horarios también establecidos.

Las bases de datos se han convertido en las componentes claves de operaciones organizacionales, e incluso en componentes claves de la plusvalía de la empresa. Por desgracia, las bases de datos fallan y ocurren desastres. Así, los planes de respaldo y recuperación, las técnicas y los procedimientos eficaces son esenciales.

Por último, conforme pasa el tiempo se necesitará cambiar el mismo DBMS para mejorar su rendimiento e incorporar nuevas características y versiones a medida que se realicen los cambios en el sistema operativo. Todo esto requiere de una administración cuidadosa.

Con el fin de asegurarse que estos problemas están identificados y son resueltos, la mayoría de las empresas cuentan con una oficina de administración de base de datos.

Comenzaremos por mencionar las tareas de dicha oficina, y después describiremos la combinación de software, y las prácticas y los procedimientos manuales que se usan para desempeñar estas tareas. En los dos capítulos siguientes analizaremos las características y funciones de Oracle 8i y del SQL Server 2000, respectivamente, para tratar estos puntos.

➤ ADMINISTRACIÓN DE LA BASE DE DATOS

En la industria se usan los términos **administración de datos** y **administración de la base de datos.** En algunos casos los términos se consideran sinónimos; en otros, tienen diferentes significados. En este libro usamos el término *administración de datos* para referirnos a una función que se aplica a toda la empresa. El término *administración de la base de datos* se refiere a la función que es específica para una base de datos en particular, incluyendo las aplicaciones que la procesan. Este capítulo aborda la administración de la base de datos. En el capítulo 17 se analizará la administración de datos.

Las bases de datos varían considerablemente en tamaño y alcance, desde una base de datos personal de usuario único hasta una gran base de datos interorganizacional como el sistema de reservaciones de una aerolínea. Todas esas bases de datos requieren administración, aunque las tareas a realizar varían en cuanto a su complejidad. Por ejemplo, en una base de datos personal el usuario sigue procedimientos simples para respaldar su información y conservar registros mínimos con fines documentales. En este caso, esa persona que usa la base de datos ejecuta las funciones del DBA (Database Administration, por sus siglas en inglés) aun cuando probablemente no esté consciente de que lo hace.

Para aplicaciones de bases de datos multiusuario, la administración se vuelve más importante y difícil. Por lo tanto, usualmente tiene reconocimiento formal. Para algunas aplicaciones, esta función se le asigna a una o dos personas de tiempo parcial. En el caso de grandes bases de datos de Internet o Intranet, con frecuencia las responsabilidades de la administración de la base de datos llevan mucho tiempo y varían mucho como para que sean manejadas incluso por una sola persona de tiempo completo. Mantener en funcionamiento una base de datos con docenas o centenas de usuarios requiere mucho tiempo, así como conocimientos técnicos y técnicas diplomáticas que usualmente se manejan en una oficina de administración de base de datos. Al jefe de la oficina con frecuencia se le conoce como **administrador de la base de datos,** en este caso, el acrónimo **DBA** se refiere tanto a la oficina como a la administración.

La responsabilidad del DBA es facilitar el desarrollo y el uso de la base de datos. Por lo general, esto significa balancear las metas conflictivas de protección de la base de datos y maximizar su disponibilidad y beneficio para los usuarios. La DBA es responsable del desarrollo, operación y mantenimiento de la base de datos y de sus aplicaciones. En la figura 11-1 se muestran las tareas específicas. Consideraremos cada una de éstas en las siguientes secciones.

➤ FIGURA 11-1

Resumen de las tareas de la administración de la base de datos

> Administración de la estructura de la base de datos

> Control de los procesos repetitivos

> Administración de los derechos y responsabilidades del procesamiento

> Desarrollo de la seguridad de la base de datos

> Servicios de recuperación de la base de datos

> Administración del DBMS

> Mantenimiento del repositorio de datos

Administración de la estructura de la base de datos

La administración de la estructura de la base de datos incluye la participación en el diseño inicial y su implementación, así como controlar y administrar los cambios en ésta. Lo ideal es que el DBA participe en el desarrollo de la base de datos y sus aplicaciones, colabore en el estudio de los requisitos y en la evaluación de las alternativas, incluyendo el DBMS que será usado, y ayude a diseñar la estructura de la base de datos. En el caso de aplicaciones para grandes empresas u organizaciones, el administrador de la base de datos por lo general supervisa el trabajo del personal técnico que diseña la base.

Como se describió en el capítulo 8, crear la base de datos implica varias tareas diferentes. Primero se crea la base de datos y se asigna el espacio para ésta y sus registros. Después se generan las tablas, se crean los índices y se escriben los procedimientos almacenados y los disparadores. En los dos capítulos siguientes analizaremos ejemplos de todo esto. Una vez que se crean las estructuras de la base de datos, se ingresan éstos. La mayoría de los fabricantes de DBMS proporcionan utilidades para insertar grandes cantidades de datos.

CONTROL DE CONFIGURACIÓN. Después de que se implementan la base de datos y sus aplicaciones, los cambios en los requerimientos son inevitables. A partir de éstos pueden surgir nuevas necesidades, desde cambios en el medio ambiente del negocio, hasta cambios en sus normas y políticas, y así sucesivamente. Cuando los cambios en los requerimientos afectan la estructura de la base de datos, debe tenerse mucho cuidado porque pocas veces involucran sólo una aplicación.

Por lo tanto, una administración eficaz de base de datos debe incluir procedimientos y políticas por medio de las cuales los usuarios puedan registrar sus necesidades de cambios, y absolutamente todos tengan oportunidad de analizar sus repercusiones y se pueda tomar una decisión en conjunto con respecto a implementar o no los cambios propuestos.

Debido al tamaño y complejidad de las bases de datos y sus aplicaciones, a veces los cambios tienen resultados inesperados. Así, la DBA debe estar preparada para reparar la base de datos y reunir información suficiente para diagnosticar y corregir el problema que ocasionó el daño. La base de datos es más susceptible a fallar después de un cambio en su estructura.

DOCUMENTACIÓN. La responsabilidad final de la DBA en cuanto a la administración de la estructura de la base de datos es la documentación. Es muy importante saber qué cambios se hicieron, y cómo y cuándo fueron hechos. Un cambio en la estructura de la base de datos podría ocasionar un error que no se manifieste hasta seis meses después; sin la documentación apropiada del cambio, diagnosticar el problema será casi imposible. Se puede requerir repetir la ejecución de trabajos para identificar el punto en el que ciertos síntomas aparecen primero, y por esta razón es importante conservar un registro de los procedimientos de prueba y de las pruebas que se hicieron para verificar un cambio. Si los procedimientos de prueba son estándar, y si se usan formas de prueba y métodos para mantener registros, documentar los resultados de las pruebas no llevará mucho tiempo.

Aunque conservar la documentación es tedioso y poco satisfactorio, el esfuerzo se recompensa cuando sucede un desastre y la documentación es la diferencia entre resolver o no un problema mayor (y costoso). Actualmente están surgiendo varios productos que facilitan elaborar la documentación. Por ejemplo, se pueden usar muchas herramientas CASE para documentar diseños lógicos de la base de datos. Se puede usar software de control de versiones para registrar los cambios. Los diccionarios de datos proporcionan reportes y otros productos para leer e interpretar las estructuras de la base de datos.

Otra razón para documentar con cuidado los cambios en la estructura de la base de datos es tener datos históricos apropiados. Si, por ejemplo, mercadotecnia quiere analizar las ventas de tres años que han estado en los archivos durante dos años, será necesario saber qué estructura se usaba en el momento en que se registraron los datos la última vez. Se pueden usar los registros que muestran los cambios en la estructura pa-

➤ FIGURA 11-2

Resumen de las responsabilidades del DBA para la administración de la estructura de la base de datos

Participación en la base de datos y en el desarrollo de la aplicación
- Ayudar en la etapa de requerimientos y en la evaluación de las alternativas
- Desempeñar un papel activo en el diseño de la base de datos y la creación

Facilitar cambios en la estructura de la base de datos
- Buscar soluciones para toda la comunidad
- Evaluar el impacto en todos los usuarios
- Proporcionar un foro de control de configuración
- Estar preparados para los problemas después de que se hacen los cambios
- Conservar la documentación

ra contestar esa pregunta. Surge una situación similar cuando se debe usar una copia de los datos de respaldo de hace seis años para reparar una base de datos dañada (aunque esto no debería suceder, a veces pasa). Se puede usar la copia de respaldo para reconstruir la base de datos al estado en que estaba al momento del respaldo. Después se hacen los cambios estructurales y las transacciones en orden cronológico para restablecer la base de datos a su estado actual. La figura 11-2 resume las responsabilidades del DBA en cuanto a la administración de la estructura de la base de datos.

➤ CONTROL DE CONCURRENCIA

Se toman medidas de control de concurrencia para asegurar que el trabajo de un usuario no influya negativamente en el de otro. En algunos casos, estas medidas aseguran que un usuario obtendrá el mismo resultado cuando procesa con otros usuarios, que el que obtendría si estuviera procesando solo. En otros casos, esto significa que su trabajo se ve influenciado por el de otros, pero de manera anticipada.

Por ejemplo, en un sistema de registro de pedidos, un usuario debería poder ingresar un pedido y obtener el mismo resultado sin importar que haya otro usuario, o cientos de éstos. Por otra parte, un usuario que está imprimiendo un reporte del estado actual del inventario quizás quiera obtener los cambios en el procesamiento de datos de otros usuarios, a pesar de que exista el peligro de que estos cambios sean abortados más tarde.

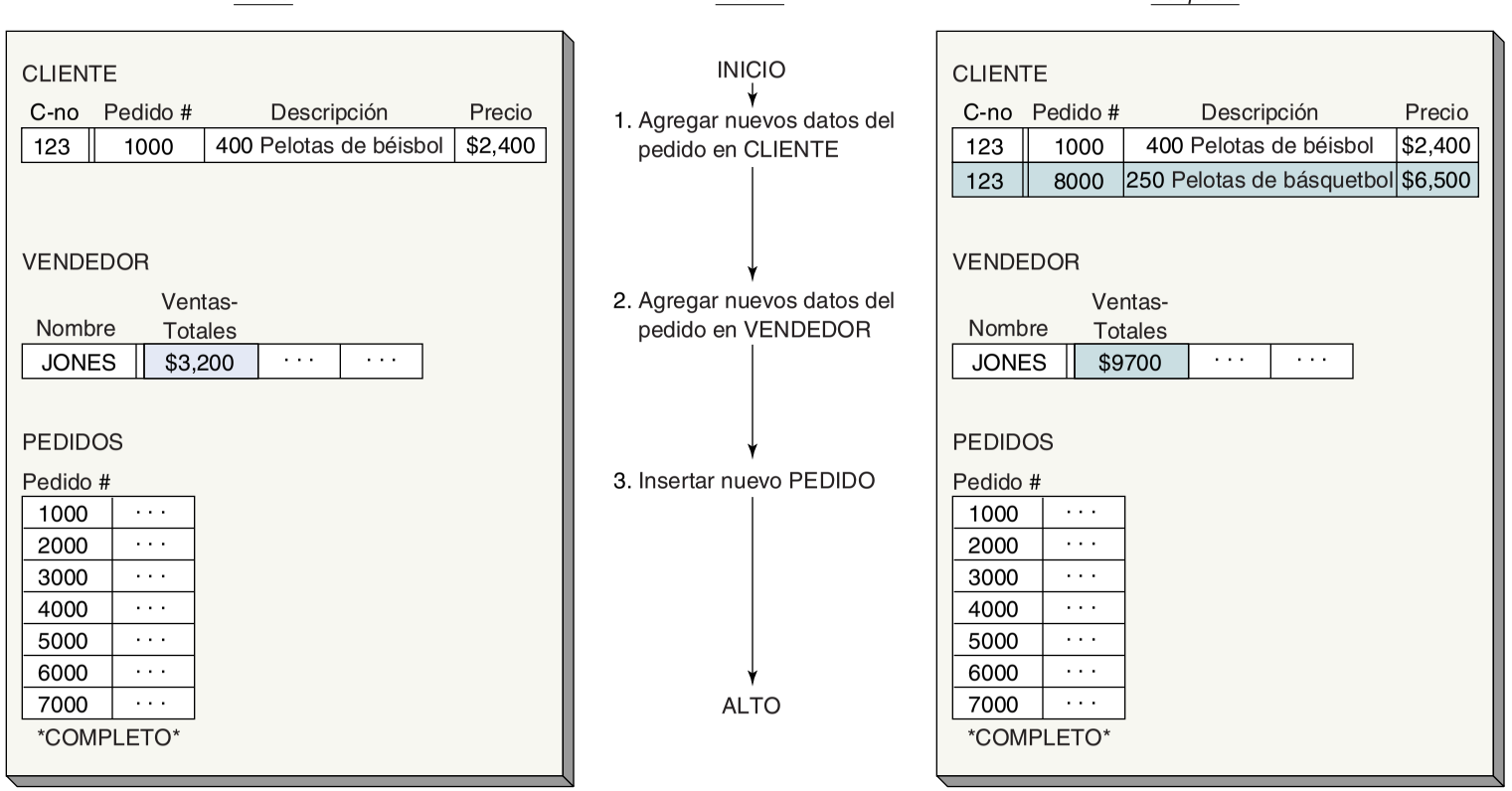
Por desgracia, ninguna técnica o mecanismo de control de concurrencias es ideal para todas las circunstancias. Todas implican decidir entre varias alternativas. Por ejemplo, un usuario puede lograr un control de concurrencia muy estricto al aplicar locks a la base de datos completa, pero mientras esté trabajando con ésta nadie más podrá hacer nada. Ésta es una protección muy estricta, pero a un precio muy alto. Como verá, hay otras medidas disponibles que son más difíciles de programar o aplicar, pero que permiten un rendimiento más eficaz. Incluso hay otras medidas que maximizan el rendimiento eficaz, pero con un nivel bajo de control de concurrencia. Cuando se diseñan aplicaciones de base de datos multiusuarios es necesario elegir una de estas alternativas.

La necesidad de transacciones atómicas

En la mayoría de las aplicaciones de la base de datos los usuarios transmiten su trabajo en forma de **transacciones,** que también se conocen como **unidades lógicas de trabajo** (LUWs, Logical Units of Work, por sus siglas en inglés). Una transacción (o LUW) es una serie de acciones que se llevarán a cabo en la base de datos, de tal manera que todas se ejecuten con éxito, o que ninguna se realice por completo; en ese caso la base de datos permanecerá sin cambios. Algunas veces esta transacción se llama **atómica,** puesto que se ejecuta como una unidad.

➤ FIGURA 11-3

Comparación de resultados de la aplicación de acciones seriales contra una transacción de pasos múltiples: (a) dos de tres actividades que concluyeron con éxito, lo que originó anomalías de la base de datos, y (b) no se realizó ningún cambio debido a que la transacción total no tuvo éxito

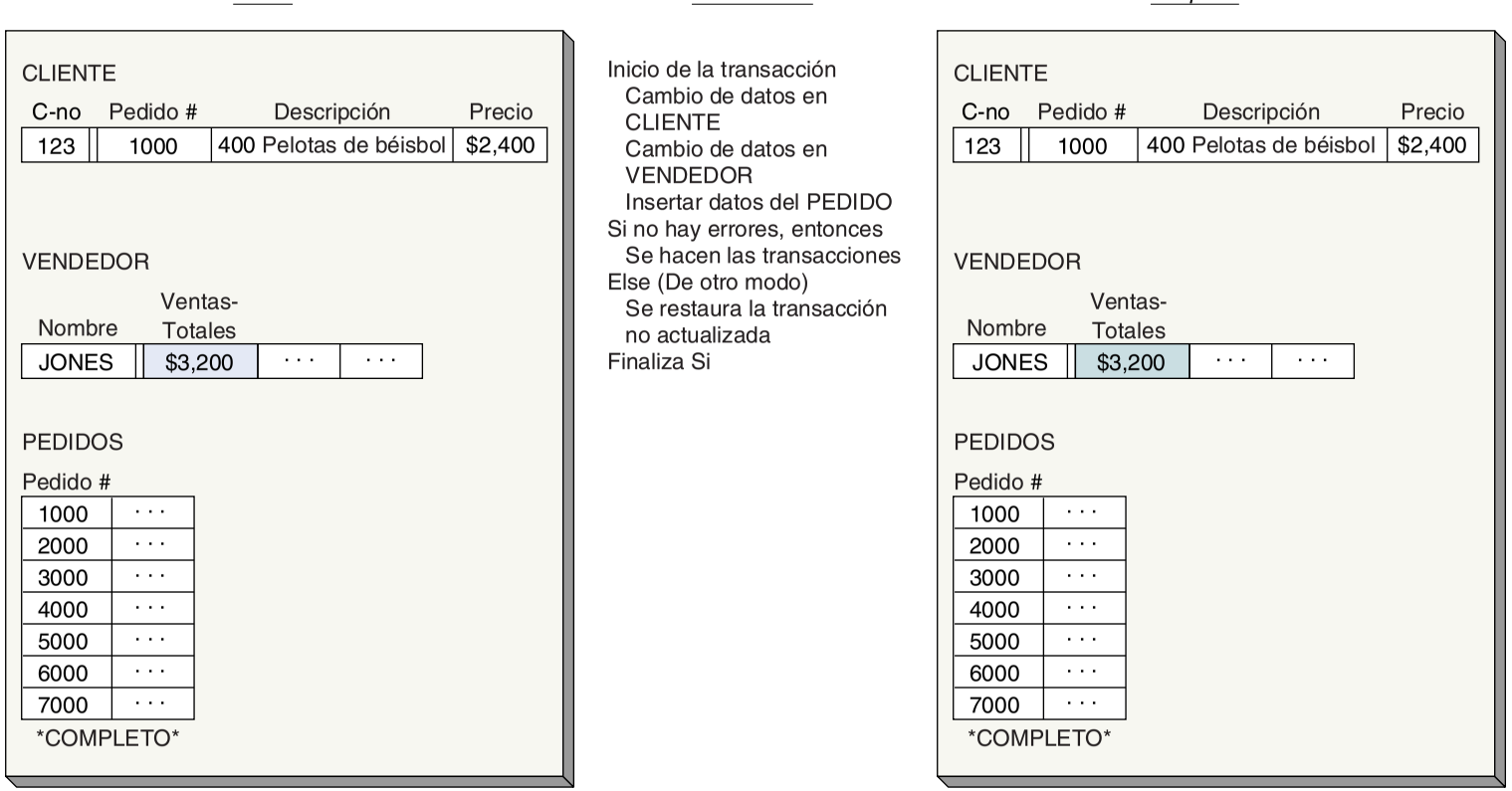


(a)

(*continúa*)

➤ FIGURA 11-3

(Continuación)



(b)

Considere la siguiente secuencia de acciones de la base de datos que puede ocurrir cuando se registra un pedido nuevo:

1. Cambia el registro del cliente y aumenta Cantidad que Adeuda.
2. Cambia el registro del vendedor y aumenta Comisión que se Debe.
3. Ingresa el nuevo registro de pedido en la base de datos.

Suponga que falla el último paso, quizás porque no hay suficiente espacio en el archivo. Imagine la confusión que habría si se hicieran los dos primeros cambios, pero no el tercero. Al cliente se le cobraría por un pedido que nunca recibió y un vendedor recibiría la comisión por un pedido que nunca fue enviado. Obviamente, se necesita considerar estas tres acciones en conjunto —ya sea que se realicen todas o que ninguna se lleve a cabo.

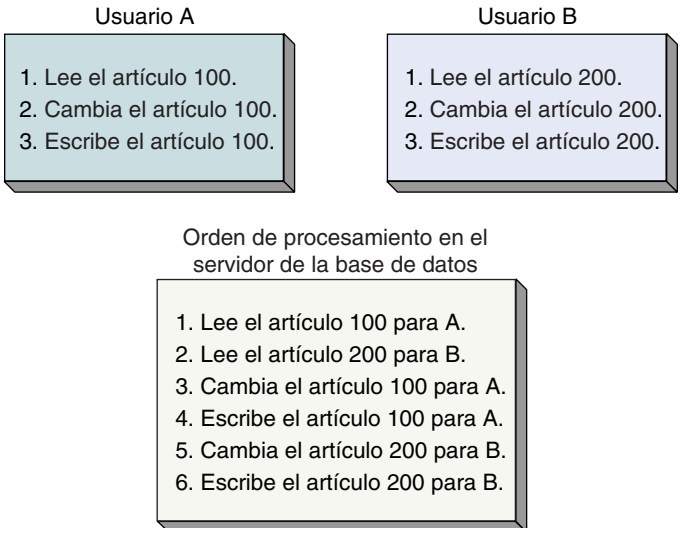
En la figura 11-3 se comparan los resultados de la realización de estas actividades como una serie de pasos independientes (figura 11-3[a]) y como una transacción atómica (figura 11-3[b]). Observe que cuando se realizan los pasos atómicamente y falla uno, no se realiza ningún cambio en la base de datos. También note que las instrucciones Iniciar transacción (Start Transaction), Confirmar transacción (Commit Transaction), o Deshacer transacción (Rollback Transaction) los debe emitir el programa de aplicación para que se marquen los límites de la lógica de transacción. La forma particular de estas instrucciones varía de un producto DBMS a otro.

PROCESAMIENTO DE TRANSACCIONES CONCURRENTES. Cuando se han procesado dos transacciones en una base de datos al mismo tiempo, se les llama *transacciones concurrentes*. Aun cuando al usuario le pueda parecer que las transacciones concurrentes se han procesado en forma simultánea, esto no puede ser verdad, ya que la CPU de la máquina que está procesando la base de datos sólo puede ejecutar una instrucción a la vez. Por lo general las transacciones están mezcladas, lo que significa que el sistema operativo alterna los servicios de la CPU entre las tareas para que alguna parte de cada una de ellas se realice en un intervalo determinado. Este cambio entre las tareas se realiza tan rápidamente que dos personas usando browsers, y sentadas lado a lado procesando la misma base de datos, quizás crean que sus transacciones fueron completadas simultáneamente, pero, en realidad, las dos están mezcladas.

La figura 11-4 muestra dos transacciones concurrentes. La del usuario A lee el artículo 100, lo cambia, y lo rescribe en la base de datos. La transacción del usuario B lleva a cabo las mismas acciones, pero en el artículo 200. La CPU procesa lo del usuario A hasta que encuentra una interrupción de I/O o alguna otra causa de retraso. El sistema operativo cambia el control al usuario B. La CPU procesa ahora lo del usuario B hasta que encuentra una interrupción; en este punto el sistema operativo pasa el control de regreso al usuario A. Para los usuarios, el procesamiento parece simultáneo, pero en realidad está mezclado, o es concurrente.

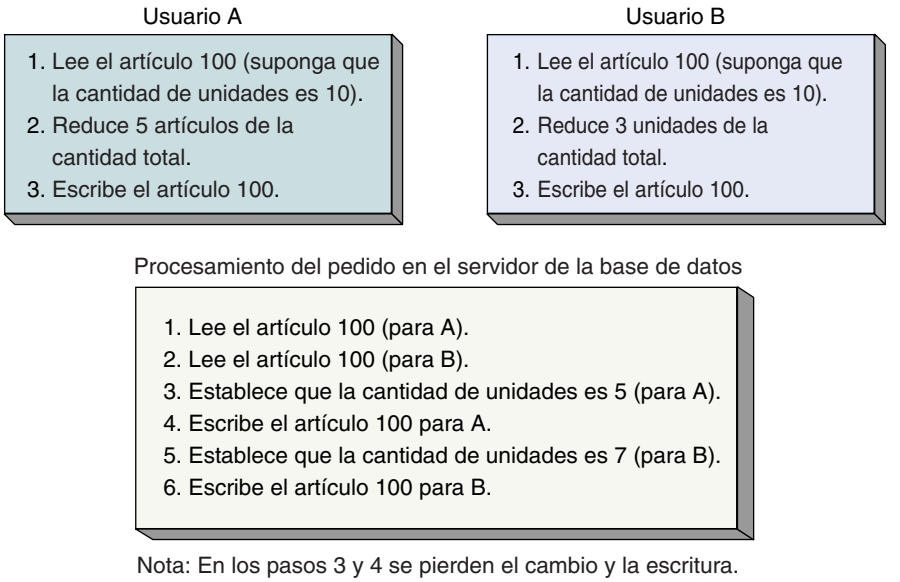
➤ FIGURA 11-4

Ejemplo de procesamiento concurrente de las tareas de dos usuarios



➤ FIGURA 11-5

Problema de pérdida de una actualización



PROBLEMA DE PÉRDIDAS EN LA ACTUALIZACIÓN. El procesamiento concurrente en la figura 11-4 no tiene problemas porque dos usuarios están procesando datos diferentes. Pero supongamos que ambos quieren procesar el artículo 100. Por ejemplo, el usuario A quiere ordenar cinco unidades del artículo 100, y el B quiere ordenar tres unidades del mismo artículo.

La figura 11-5 ilustra el problema. El usuario A lee el registro del artículo 100 en el área de trabajo de un usuario. De acuerdo con el registro, existen 10 unidades en inventario. El usuario B lee el registro del artículo 100 en otra área de trabajo. Nuevamente, de acuerdo con el registro existen 10 unidades en inventario. Ahora el usuario A toma cinco, disminuye la cantidad de elementos en su área de trabajo a cinco y rescribe el registro para el artículo 100. Luego el usuario B toma tres, disminuye la cantidad en su área de trabajo a siete, y rescribe el registro para el elemento 100. La base de datos ahora muestra, incorrectamente, que existen siete elementos en el inventario del elemento 100. Revisemos: comenzamos con 10 en inventario, el usuario A toma cinco, el usuario B toma tres y la base de datos muestra que hay siete en inventario. Obviamente, hay un problema.

Ambos usuarios tuvieron datos que eran correctos al momento de obtenerlos. Pero cuando el usuario B leyó el registro, el A ya tenía una copia que estaba por actualizarse. Esta situación se llama **problema de pérdida de la actualización** o **problema concurrente en la actualización.** Existe otro problema similar, llamado **problema de la lectura inconsistente.** En éste, el usuario A lee información que ha sido procesada por una parte de la transacción del usuario B. Como resultado, el A lee incorrectamente los datos.

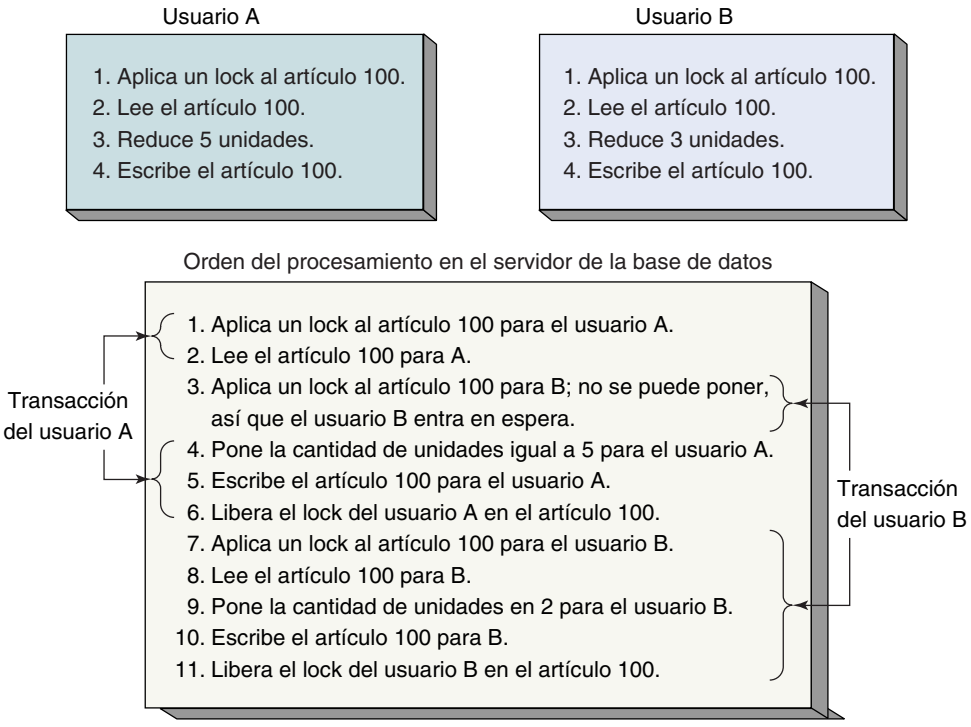
Una solución para las inconsistencias causadas por procesamiento concurrente es evitar que aplicaciones múltiples obtengan copias de un mismo registro cuando va a tener cambios. A esto se le denomina **lock (Bloqueo) de recursos** (resource locking).

Lock de recursos

Una manera de evitar problemas de procesamiento concurrente es anular cualquier posibilidad de compartir información mediante el lock de los datos que se recuperan para la actualización. La figura 11-6 muestra el orden del procesamiento usando un comando de **lock (Bloqueo).** Debido al bloqueo, la transacción del usuario B debe esperar hasta que el A haya terminado con los datos del artículo 100. Usando esta estrategia, el B puede leer el registro del elemento 100 sólo después de que el usuario A ha terminado la modificación. En este caso, la cantidad final del artículo almacenado en la base de datos es dos. (Comenzamos con 10; A tomó cinco y B tomó tres, por lo tanto quedan dos.)

➤ FIGURA 11-6

Procesamiento concurrente con locks (Bloqueos) explícitos



TERMINOLOGÍA DE LOCKS. Los locks se pueden poner ya sea en forma automática, a través del DBMS, o mediante una instrucción enviada al DBMS desde el programa de aplicación, o a solicitud del usuario. Los locks que impone el DBMS se llaman **locks implícitos;** los que se ponen mediante una instrucción se llaman **locks explícitos.**

En el ejemplo anterior los locks se aplicaron en los renglones de datos. Sin embargo, no se aplican todos los locks en este nivel. Algunos productos DBMS aplican locks por página, otros, por tabla, y otros más, a toda la base de datos. Al tamaño de un lock se le llama **granularidad del bloqueo.** Los locks con una gran granularidad son fáciles de administrar en el DBMS pero con frecuencia ocasionan conflictos. Los locks con granularidad pequeña son difíciles de administrar (hay muchos más detalles que debe atender el DBMS para conservar los registros y verificarlos), pero los conflictos son menos comunes.

Los locks también varían en su tipo. Un **lock exclusivo** impide el acceso a cualquier tipo de artículo. Ninguna otra transacción puede leer o cambiar los datos. Un **lock compartido** impide que se hagan cambios al elemento de datos, pero permite la lectura. Es decir, otras transacciones pueden leer de ese dato siempre y cuando no intenten modificarlo.

TRANSACCIONES SERIALIZABLES. Cuando dos o más transacciones se procesan al mismo tiempo, los resultados en la base de datos deben ser lógicamente consistentes con los resultados de las transacciones que hayan sido procesadas de manera serial arbitraria. A un esquema para el procesamiento concurrente de transacciones se le llama **serializable.**

La seriabilidad se puede alcanzar por diferentes medios. Una manera es procesar la transacción usando un **lock de dos fases.** Con esta estrategia, a las transacciones se les permite tener locks cuando sea necesario, pero una vez que se libera el primer lock, no se puede obtener otro. Las transacciones tienen una **fase de crecimiento,** en la que los locks se pueden obtener, y una **fase de liberación** en la que se liberan los locks.

Un caso especial de lock de dos fases se usa en varios productos DBMS. Con éste, los locks se obtienen durante la transacción, pero no se libera ninguno hasta que se emite la instrucción COMMIT o ROLLBACK. Esta estrategia es más restrictiva que los requerimientos del lock de dos fases, pero es más fácil de implementar.

En general, las fronteras de una transacción deberían corresponder a la definición de la vista de la base de datos que está en proceso. Siguiendo con la estrategia de dos fases, los renglones de cada relación en la vista se bloquean cuando es necesario. Se hacen los cambios, pero no se confirman los datos en la base hasta que se ha procesado toda la vista. En este punto, se hacen los cambios en la base de datos real y se liberan todos los locks.

Considere una transacción de registro de pedidos que implique un objeto CLIENTE-PEDIDO, el cual se construye a partir de los datos de las tablas CLIENTE, VENDEDOR y PEDIDO. Esto asegura que la base de datos no sufrirá ninguna anomalía debido a la concurrencia, la transacción de registro de pedidos aplica locks en CLIENTE, VENDEDOR y PEDIDO cuando es necesario, hace todos los cambios de la base de datos y luego libera todos sus locks.

DEADLOCK (BLOQUEO MORTAL). Aunque los locks resuelven un problema, introducen otro. Considere lo que puede pasar cuando dos usuarios quieren ordenar dos artículos del inventario. Suponga que el usuario A quiere ordenar papel y si puede obtenerlo quiere ordenar algunos lápices. Entonces suponga que el usuario B quiere ordenar algunos lápices, y si puede obtenerlos entonces quiere algo de papel. El orden del procesamiento podría ser el que se muestra en la figura 11-7.

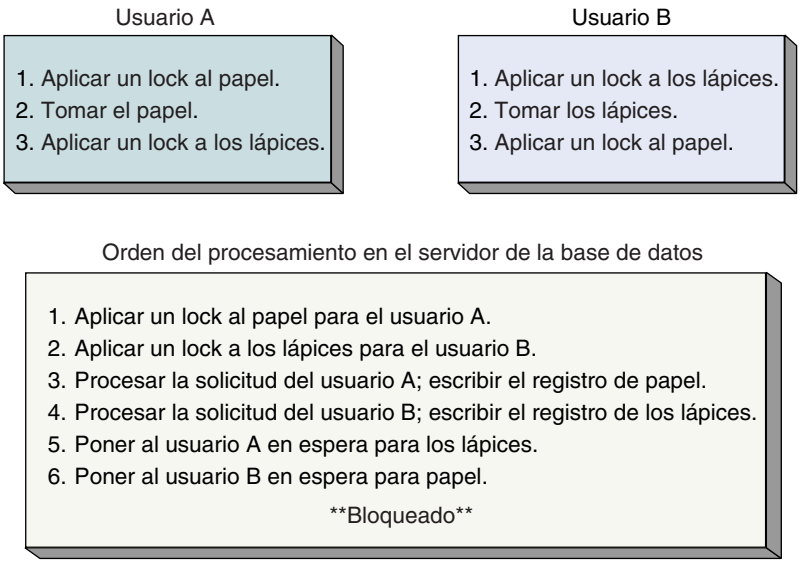
En esta figura, los usuarios A y B están atrapados en una condición llamada **deadlock** o **abrazo mortal.** Cada uno está esperando un recurso que el otro bloqueó. Hay dos maneras comunes de resolver este problema: evitar que ocurra el deadlock, o permitirlo y luego romperlo.

El deadlock se puede evitar de varias maneras. Una consiste en permitir que los usuarios emitan sólo una solicitud de lock a la vez. En esencia, los usuarios deben bloquear al instante todos los recursos que requieren. Si el usuario A en la ilustración bloquea al principio los registros de papel y lápices el deadlock nunca tendrá lugar. Una segunda manera de evitar el deadlock es requerir que todos los programas de aplicación bloqueen los recursos en el mismo orden. Incluso si no se bloquean todas las aplicaciones en ese orden, el deadlock se reducirá a los que lo hacen. Esta filosofía podría extenderse a un estándar de programación organizacional tal como: "Siempre que se procesen renglones de las tablas en una relación padre-hijo, bloquear al padre antes que a los renglones hijo". Esto cuando menos reducirá la probabilidad del deadlock y podría eliminarlo por completo.

Casi todos los DBMS tienen algoritmos para detectar el deadlock. Cuando éste ocurre la solución más común es revertir una de las transacciones para eliminar dead-

➤ FIGURA 11-7

Deadlock



lock sus cambios en la base de datos. En los dos capítulos siguientes usted verá variantes con Oracle y con el SQL Server.

Bloqueo optimista contra bloqueo pesimista

Los locks se pueden invocar en dos estilos básicos. Con el **lock optimista** se supone que no ocurrirá ningún conflicto. Los datos se leen, se procesan las transacciones y las actualizaciones y después se realiza una verificación para ver si ocurre algún conflicto; en caso contrario se termina la transacción. Si hay un conflicto ésta se repite hasta que se procese sin ningún conflicto. El **bloqueo pesimista** supone que ocurrirá el conflicto. Primero se aplican los locks, después se procesa la transacción y luego se liberan los locks.

La figura 11-8 muestra un ejemplo de cada estilo para una transacción que está restando cinco a la cantidad del renglón de lápiz en PRODUCTO. La figura 11-8(a) muestra un lock optimista. Primero se leen los datos y se almacena el valor real de Cantidad de lápices en la variable ViejaCantidad. Después la transacción se procesa y, suponiendo que todo está bien, se aplica un lock en PRODUCTO. El lock podría ser sólo para el renglón lápiz, o podría tener un nivel más grande de granularidad. En cualquiera de los casos se emite una instrucción SQL para actualizar el renglón lápiz con una condición WHERE con lo cual el valor actual de Cantidad sea igual a ViejaCantidad. Si otra transacción no ha cambiado la Cantidad del renglón lápiz, entonces este UPDATE (ACTUALIZACIÓN) tendrá éxito. Si otra transacción ha cambiado la Cantidad del renglón lápiz, El UPDATE fallará y será necesario repetir la transacción.

La figura 11-8(b) muestra la lógica para la misma transacción usando el lock pesimista. Aquí, se aplica un lock en PRODUCTO (con algún nivel de granularidad) antes de iniciar cualquier trabajo. Los valores se leen, la transacción se procesa, ocurre el UPDATE y PRODUCTO se desbloquea.

La ventaja del lock optimista es que se obtiene sólo después de que la transacción ha sido procesada. Así, el lock se mantiene durante menos tiempo que con el lock pesimista. Si la transacción es complicada, o si el cliente es lento (debido a los retrasos en la transmisión, o a que esté realizando otro trabajo, consiguiendo una taza de café, o que apague sin salirse del explorador, o browser), el lock se mantendrá durante un tiempo considerablemente menor. Esta ventaja será aún más importante si la granularidad del lock es grande; digamos, la tabla PRODUCTO.

La desventaja del lock optimista es que si existe mucha actividad en el renglón lápiz, la transacción se puede repetir muchas veces. Así, las transacciones que implican mucha actividad en un renglón determinado son muy poco adecuadas para el lock optimista.

En general, Internet es un sitio salvaje y denso, y a los usuarios les gusta tomar decisiones inesperadas, como por ejemplo abandonar las transacciones a la mitad. Así que a menos que los usuarios de Internet hayan sido precalificados (por ejemplo, al inscribirse en línea en un plan de corretaje de compra), el lock optimista es una mejor opción. Sin embargo, en intranet, la decisión es más difícil. Probablemente el lock optimista es el más indicado a menos que algunas características de la aplicación causen actividad sustancial en algunos renglones en particular, o si los requisitos de la aplicación hacen el reprocesamiento de transacciones particularmente indeseable.

Declarar las características del lock

Como usted puede ver, el control de concurrencias es un tema complicado; algunas decisiones acerca de los tipos y estrategias de los locks se han tenido que tomar a base de prueba y error. Debido a ésta y a otras razones, los programas de aplicación de base de datos, por lo general, no emiten locks explícitos, sino que marcan las fronteras de la transacción y después establecen el tipo de comportamiento de bloqueo que desean use el

➤ FIGURA 11-8

Lock optimista contra pesimista: (a) lock optimista; (b) lock pesimista

```
SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

ViejaCantidad = PRODUCTO.Cantidad

Set NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar transacción – llevar a cabo acción de excepción si NuevaCantidad < 0, etc.

Suponer que todo está BIEN:}

LOCK        PRODUCTO {con algún nivel de granularidad}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'
      AND   PRODUCTO.Cantidad = ViejaCantidad

UNLOCK      PRODUCTO

{comprobar para ver si la actualización tuvo éxito; de lo contrario, repetir la transacción}
```

(a)

```
LOCK        PRODUCTO {con algún nivel de granularidad}

SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

Set NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar transacción – llevar a cabo acción de excepción si NuevaCantidad < 0, etc.

Suponer que todo está BIEN:}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'

UNLOCK      PRODUCTO

{no se necesita comprobar si la actualización tuvo éxito}
```

(b)

DBMS. De esta manera, si se necesita cambiar el comportamiento del bloqueo, no se necesita rescribir la aplicación para colocar locks en diferentes lugares en la transacción. En lugar de eso, se cambia la declaración del lock.

La figura 11-9 muestra la transacción de lápiz con las fronteras de la transacción marcadas con INICIAR TRANSACCIÓN, COMMIT TRANSACCIÓN, ROLLBACK TRANSACCIÓN (BEGIN TRANSACTION, COMMIT TRANSACTION y ROLLBACK TRANSACTION). Estas fronteras son la información esencial que el DBMS necesita para aplicar las diferentes estrategias de bloqueo. Si el programador ahora declara (a través de un parámetro del sistema, o por medios similares) que quiere que el lock sea optimista, el

➤ FIGURA 11-9

Establecimiento de las fronteras de transacción

```
INICIAR TRANSACCIÓN:

SELECT      PRODUCTO.Nombre, PRODUCTO.Cantidad
FROM        PRODUCTO
WHERE       PRODUCTO.Nombre = 'Lápiz'

ViejaCantidad = PRODUCTO.Cantidad

SET NuevaCantidad = PRODUCTO.Cantidad – 5

{procesar parte de la transacción – llevar a cabo la acción de excepción si NuevaCantidad < 0, etc.}

UPDATE      PRODUCTO
SET         PRODUCTO.Cantidad = NuevaCantidad
WHERE       PRODUCTO.Nombre = 'Lápiz'

{continuar procesando transacción}...

Si la transacción ha terminado normalmente      THEN (ENTONCES)

      COMMIT TRANSACCIÓN

ELSE (DE OTRO MODO)

      ROLLBACK TRANSACCIÓN

END IF (TERMINA SI)

Continuar procesando otras acciones que no son parte de esta transacción...
```

DBMS establecerá implícitamente los locks en los lugares correctos para ese tipo de bloqueo. Si el programador más tarde cambia las tácticas y solicita el bloqueo pesimista, el DBMS configurará implícitamente los locks en un lugar diferente.

Transacciones consistentes

Algunas veces usted verá el acrónimo ACID aplicado a las transacciones. Una **transacción ACID** (**a**tomic, **c**onsistent, **i**nsolated and **d**urable, por sus siglas en inglés) es aquella que es atómica, consistente, aislada y durable. Atómica y durable son fáciles de definir. Como acaba de leer, una transacción atómica es aquella en la que todas las acciones de la base de datos pueden ocurrir, o también ninguna. Una transacción durable es aquella para la que todos los cambios confirmados son permanentes. El DBMS no eliminará esos cambios, incluso en el caso de fracasar. Si la transacción es durable, el DBMS proporcionará las facilidades para recuperar los cambios de todas las acciones confirmadas cuando sea necesario.

Sin embargo, los términos *consistente* y *aislada* no son tan definitivos como *atómica* y *durable*. Considere la siguiente instrucción de actualización de SQL:

```
UPDATE    CLIENTE
SET       CódigodeÁrea = '425'
WHERE     CódigoPostal = '98050'
```

Suponga que hay 500, 000 renglones en la tabla CLIENTE y que 500 tienen CódigoPostal igual a '98050'. Le tomará algún tiempo al DBMS encontrar los 500 renglones. Durante ese tiempo, ¿otras transacciones permitirán actualizar los campos de CódigodeÁrea o CódigoPostal de CLIENTE? Si la instrucción de SQL es consistente, estas actualizaciones estarán prohibidas. La actualización se aplicará para establecer que los

renglones como éstos existen en el momento en que el enunciado de SQL inició. Esta consistencia se llama **consistencia del nivel de instrucción.**

Ahora considere una transacción que contenga dos instrucciones de actualización de SQL:

```
BEGIN TRANSACTION
UPDATE    CLIENTE
SET       CódigodeÁrea = '425'
WHERE     CódigoPostal = '98050'
{otra transacción en funcionamiento}
UPDATE    CLIENTE
SET       Descuento = 0.05
WHERE     CódigodeÁrea = '425'
{otra transacción en funcionamiento}
COMMIT    TRANSACTION
```

En este contexto, ¿qué significa consistente? La consistencia del nivel de instrucción quiere decir que cada instrucción procesa independientemente renglones consistentes, pero que los cambios de otros usuarios de estos renglones se pueden permitir durante el intervalo entre las dos instrucciones SQL. El **nivel de consistencia de la transacción** significa que todos los renglones impactados por cualquiera de las instrucciones SQL son protegidos de cambios durante la transacción completa. Observe, sin embargo, que para algunas implementaciones de la consistencia del nivel de transacción, una transacción no verá sus propios cambios. En este ejemplo, la segunda instrucción SQL puede no ver los cambios en los renglones derivados de la primera instrucción SQL.

Así, cuando usted escuche el término *consistente*, ponga más atención para determinar a qué tipo de consistencia se refiere. También tenga cuidado con la trampa potencial de consistencia del nivel de transacción.

La situación es más complicada para el término *aislada*, el cual consideraremos a continuación.

Nivel de aislamiento de la transacción

Los locks evitan que los procesos concurrentes ocasionen la pérdida de actualizaciones; pero hay otro tipo de problemas que no evitan. Específicamente, ocurre una **lectura sucia** cuando una transacción lee un registro cambiado que no ha sido registrado en la base de datos. Por ejemplo, esto puede ocurrir si una transacción lee un renglón cambiado por una segunda transacción no confirmada, la cual más tarde aborta.

Las **lecturas no repetibles** ocurren cuando una transacción relee datos que fueron leídos con anterioridad y encuentra modificaciones o eliminaciones ocasionadas por una transacción confirmada. Por último, las **lecturas fantasma** tienen lugar cuando una transacción relee los datos y encuentra que se insertaron nuevos renglones como resultado de una transacción confirmada en la lectura anterior.

El estándar ANSI SQL de 1992 define cuatro **niveles de aislamiento,** que especifican cuál de estos problemas se permite que ocurra. El objetivo es que el programador de la aplicación pueda declarar el tipo de nivel de aislamiento que quiere y entonces tener la administración de los locks del DBMS, así como lograr ese nivel de aislamiento.

Como se muestra en la figura 11-10, la lectura en un nivel de aislamiento no confirmado permite que ocurran lecturas sucias, lecturas no repetibles y lecturas fantasmas. Con aislamiento de lecturas confirmadas, las lecturas sucias están prohibidas. El nivel de aislamiento de lecturas repetibles prohíbe tanto las lecturas sucias como las no repetibles. El nivel de aislamiento serializable no permitirá que ocurra ninguna de las tres.

Por lo general, el nivel más restrictivo, de menor rendimiento efectivo, depende mucho de la carga de trabajo y de cómo estén escritos los programas de aplicación. Además, no todos los productos DBMS usan todos estos niveles. Los productos también varían en la manera en la cual se manejan y en cuanto a la carga que ponen en el

➤ FIGURA 11-10

Resumen de los niveles de aislamiento

		Nivel de aislamiento			
		Lectura no confirmada	**Lectura confirmada**	**Lectura repetible**	**Serializable**
Tipo de problema	**Lectura sucia**	Posible	No posible	No posible	No posible
	Lectura no repetible	Posible	Posible	No posible	No posible
	Lectura fantasma	Posible	Posible	Posible	No posible

programador de la aplicación. En los dos siguientes capítulos aprenderá cómo Oracle y el SQL Server manejan los niveles de aislamiento.

Tipo de cursor

Un cursor es un apuntador a un conjunto de renglones. Los cursores generalmente se definen usando las instrucciones SELECT. Por ejemplo, la instrucción:

```
DECLARE CURSOR TransCursor AS
SELECT    *
FROM      TRANSACCIÓN
WHERE     PreciodeCompra > '10000'
```

define un cursor llamado TransCursor que opera sobre el conjunto de renglones que indica la instrucción SELECT. Cuando un programa de aplicación abre un cursor y lee el primer renglón, se dice que está "apuntando al primer renglón".

Una transacción abre varios cursores —ya sea en secuencia o al mismo tiempo—. Además, se pueden abrir dos o más cursores en la misma tabla; ya sea directamente en ésta o a través de la vista SQL en la tabla. Debido a que los cursores requieren mucha memoria, el hecho de tener muchos abiertos al mismo tiempo —por ejemplo, para mil transacciones concurrentes—, puede consumir mucha memoria y tiempo del CPU. Una manera de reducir los gastos del cursor es definir cursores de capacidad reducida y utilizarlos cuando no se necesite uno de capacidad completa.

La figura 11-11 lista cuatro tipos de cursores usados en el ambiente de Windows 2000 (los tipos de cursores para otros sistemas son similares). El cursor más simple es sólo hacia delante (forward only). Con éste, la aplicación sólo se puede mover hacia adelante a través del conjunto de registros. Los cambios hechos por otros cursores en esta transacción y por otras transacciones serán visibles sólo si están en los renglones arriba del cursor.

Los tres tipos siguientes de cursor se llaman **cursores deslizables** puesto que la aplicación puede deslizarse hacia adelante y hacia atrás del conjunto de registros (recordset). Un cursor estático procesa una "fotografía" de la relación que se tomó cuando se abrió el cursor. Los cambios que se hacen usando este cursor son visibles para éste; pero los cambios de cualquier otra fuente no son visibles.

Los cursores keyset combinan algunas características de los cursores estáticos con otras de los dinámicos. Cuando se abre el cursor se guarda un valor de la llave primaria para cada renglón en el conjunto de registros (recordset). Cuando la aplicación posiciona el cursor sobre un renglón, el DBMS usa el valor de la llave para leer el valor actual del renglón. Si la aplicación emite una actualización sobre un renglón que ha sido eliminado por un cursor diferente en esta transacción, o por una transacción diferente, el DBMS crea un nuevo renglón con el valor de la llave anterior y pone los valores de la actualización en el renglón nuevo (suponga que están presentes todos los campos requeridos). Las inserciones de nuevos renglones que hacen otros cursores en esta transac-

➤ FIGURA 11-11

Resumen de tipos de cursor

Tipo de cursor	Descripción	Comentarios
Sólo hacia delante (forward only)	La aplicación sólo puede moverse hacia adelante a través del conjunto de registros.	Los cambios hechos por otros cursores en esta transacción o en otra estarán visibles sólo si se localizan en líneas adelante del cursor.
Estático	La aplicación ve los datos como si éste fuera el momento en que se abrió el cursor.	Los cambios hechos por este cursor son visibles. Las cambios de otras fuentes no lo son. Puede deslizarse hacia adelante y hacia atrás.
Keyset	Cuando se abre el cursor se salva el valor de la llave primaria para cada renglón en el conjunto de registros. Cuando la aplicación accede a un renglón, la llave se usa para buscar los valores actuales para el renglón.	Las actualizaciones de cualquier fuente son visibles. Las inserciones de fuentes fuera de este cursor no son visibles (no hay una llave para estas en el keyset). Las inserciones de este cursor aparecen al final del conjunto de registros (recordset). Las eliminaciones de cualquier fuente están visibles. Los cambios en el orden de los renglones no son visibles. Si el nivel de aislamiento es de lectura sucia, entonces las actualizaciones confirmadas y las eliminaciones están visibles; de otra manera, sólo las actualizaciones confirmadas y las eliminaciones son visibles.
Dinámico	Los cambios de cualquier tipo y fuente están visibles.	Todas las inserciones, actualizaciones, eliminaciones y cambios en el orden del conjunto de registros (recordset) están visibles. Si el nivel de aislamiento es de lectura sucia, entonces los cambios no confirmados están visibles; de otra manera, sólo se ven los confirmados.

ción, u otras transacciones, no están visibles para un cursor de keyset. A menos que el nivel de aislamiento de la transacción sea una lectura sucia, sólo las actualizaciones confirmadas y las eliminaciones serán visibles para el cursor.

Un cursor dinámico es un cursor plenamente caracterizado. Todas las inserciones, actualizaciones, eliminaciones, cambios en un renglón están visibles para un cursor dinámico. Al igual que con los cursores de keyset, a menos que el nivel de aislamiento de la transacción sea una lectura sucia, sólo los cambios confirmados estarán visibles.

La cantidad de gastos y el procesamiento que se requieren para sostener un cursor es diferente para cada tipo. En general, el costo aumenta conforme nos movemos hacia abajo de los tipos de cursor de la figura 11-11. Por lo tanto, con el fin de mejorar el rendimiento del DBMS, el programador de la aplicación sólo debería crear cursores que sean lo suficientemente poderosos como para hacer el trabajo. También es muy importante entender cómo un DBMS en particular implementa cursores, y si éstos están localizados en el servidor o en el cliente. En algunos casos, sería mejor poner un cursor dinámico a disposición del cliente, que tener un cursor estático en el servidor. No se puede establecer ninguna regla general debido a que el desempeño depende de la implementación que usa el producto DBMS, así como de los requerimientos de la aplicación.

Advertencia: si no se especifica el nivel de aislamiento de una transacción, o no se especifica el tipo de cursores que abre, el DBMS usará el nivel y los tipos predeterminados. Estas predeterminaciones pueden ser perfectas para su aplicación, pero también

terribles. Así, aunque se pueden ignorar estos asuntos, no se evitarán sus consecuencias. Aprenda las capacidades de su producto DBMS y úselas con sabiduría.

➤ SEGURIDAD DE LA BASE DE DATOS

El objetivo de la seguridad de la base de datos es asegurar que sólo los usuarios autorizados puedan desempeñar actividades permitidas en momentos también establecidos. Este objetivo es difícil de alcanzar y para lograrlo el equipo programador de la base de datos debe, durante la fase de especificación de requerimientos del proyecto, determinar los derechos y responsabilidades de procesamiento de todos los usuarios. Estos requerimientos de seguridad se pueden imponer usando las características de seguridad del DBMS y las adiciones a las características que se describen en los programas de aplicación.

DERECHOS Y RESPONSABILIDADES DEL PROCESAMIENTO

Considere, por ejemplo, las necesidades de la galería View Ridge que analizamos en el capítulo 10. Existen tres tipos de usuarios: el personal de ventas, el administrativo y los administradores del sistema. Al personal de ventas se le permite ingresar un nuevo cliente y los datos de la transacción, cambiar la información del cliente y solicitar cualquier información. No se les permite meter nuevos artistas o la información del trabajo. Tampoco pueden eliminar cualquier dato.

El personal administrativo tiene acceso a todos los permisos del personal de ventas, además pueden introducir nuevos artistas y datos de las obras de arte, así como modificar los datos de las transacciones. Aunque el personal administrativo está autorizado para eliminar datos, no tienen ese permiso en esta aplicación. Esta restricción tiene como fin evitar la posibilidad de perder información accidentalmente.

Al administrador del sistema se le permite el acceso irrestricto a los datos. Puede crear, actualizar, leer y eliminar cualquier información de la base de datos. También puede otorgar derechos de procesamiento a otros usuarios y cambiar la estructura de los elementos de la base de datos tales como tablas, índices, procedimientos almacenados y cosas por el estilo. La figura 11-12 resume estos requerimientos.

➤ FIGURA 11-12

Derechos del procesamiento en la galería View Ridge

	Cliente	**Transacción**	**Trabajo**	**Artista**
Personal de ventas	Insertar, Cambiar, Consultar	Insertar, Consultar	Consultar	Consultar
Personal administrativo	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar	Insertar, Cambiar, Consultar
Administrador del sistema	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura	Insertar, Cambiar, Consultar, Eliminar, Otorgar derechos, Modificar estructura

Los permisos en esta tabla están por tipos de usuarios, o **grupos de usuarios,** y no por personas. Esto es normal, pero no es obligatorio. Sería posible decir, por ejemplo, que el usuario identificado como "Benjamin Franklin" tiene ciertos derechos de procesamiento. También observe que cuando se usan los grupos es necesario tener un medio para asignar usuarios a los grupos. Cuando "Mary Smith" se registra en la computadora, se necesitan algunos medios para determinar a qué grupo o grupos pertenece. Analizaremos esto con mayor detalle en la siguiente sección.

En este análisis hemos usado la frase *derechos y responsabilidades de procesamiento*; como lo indica dicha frase, las responsabilidades van unidas a los derechos de procesamiento. Si, por ejemplo, el administrador del sistema elimina los datos de la transacción, tiene la responsabilidad de asegurarse que esto no afecte el funcionamiento de la galería, la contabilidad, etcétera.

Las responsabilidades de procesamiento no se pueden imponer a través del DBMS o de las aplicaciones de la base de datos. Éstas son codificadas en procedimientos manuales y se les explican a los usuarios durante la capacitación para operar el sistema. Éstos son temas de un texto de desarrollo de sistemas, por lo tanto no los abordaremos más aquí, sólo reiteramos que las *responsabilidades* van unidas a los *derechos*. Estas responsabilidades deben ser documentadas e impuestas.

De acuerdo con la figura 11-1, el DBA tiene la tarea de administrar los derechos y responsabilidades del procesamiento. Como es obvio, esos derechos y responsabilidades cambiarán con el tiempo. Conforme se use la base de datos, y se realicen cambios a las aplicaciones y la estructura del DBMS, surgirá la necesidad de derechos y responsabilidades nuevos o diferentes. El DBA es un punto focal para el análisis de estos cambios y su implementación.

Una vez que los derechos del procesamiento han sido definidos, se pueden implementar en muchos ámbitos: sistema operativo, servidor Web, el DBMS y la aplicación. En las dos secciones siguientes consideraremos al DBMS y la aplicación. Los otros están más allá del ámbito de este texto.

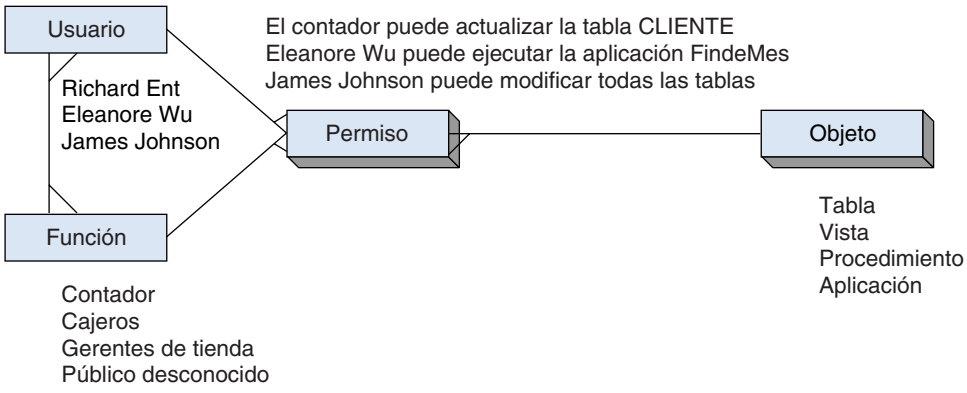
Seguridad DBMS

La terminología, las características y funciones de la seguridad del DBMS dependen del producto DBMS que se use. Básicamente, todos proporcionan dispositivos que limitan ciertas acciones u objetos a determinados usuarios. En la figura 11-13 se muestra un modelo general de seguridad DBMS. Se puede asignar a un usuario una o más funciones y una de éstas puede tener uno o más usuarios. Tanto los usuarios como las funciones tienen muchos permisos. Los objetos (usados en un sentido genérico) tienen muchos permisos asignados. Cada permiso pertenece a un usuario o función y a un objeto.

Cuando un usuario se registra en la base de datos, el DBMS limita sus acciones a los permisos que le fueron conferidos, y a los permisos para las funciones que se le asignaron. Determinar si alguien realmente es quien dice ser, en general, es una tarea difícil. Todos los productos comerciales usan alguna versión del nombre del usuario y la

➤ FIGURA 11-13

Un modelo de seguridad DBMS



contraseña, aun cuando esta seguridad se anula con facilidad si los usuarios no tienen cuidado con sus contraseñas.

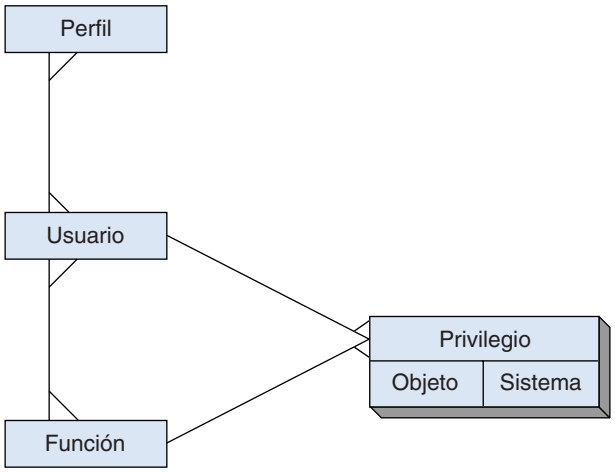
Los usuarios pueden introducir su nombre y contraseña, o, en algunas aplicaciones, el nombre y la contraseña se introducen en el nombre del usuario. Por ejemplo, el nombre y contraseña del usuario de Windows 2000 se puede pasar directamente a SQL Server. En otros casos, un programa de aplicación proporciona el nombre y la contraseña.

Las aplicaciones de Internet por lo general definen a un grupo como "Público desconocido" y le asignan usuarios anónimos cuando se inscriben. De esta manera, compañías como Dell Computer no necesitan introducir a cada usuario en su sistema de seguridad por nombre y contraseña.

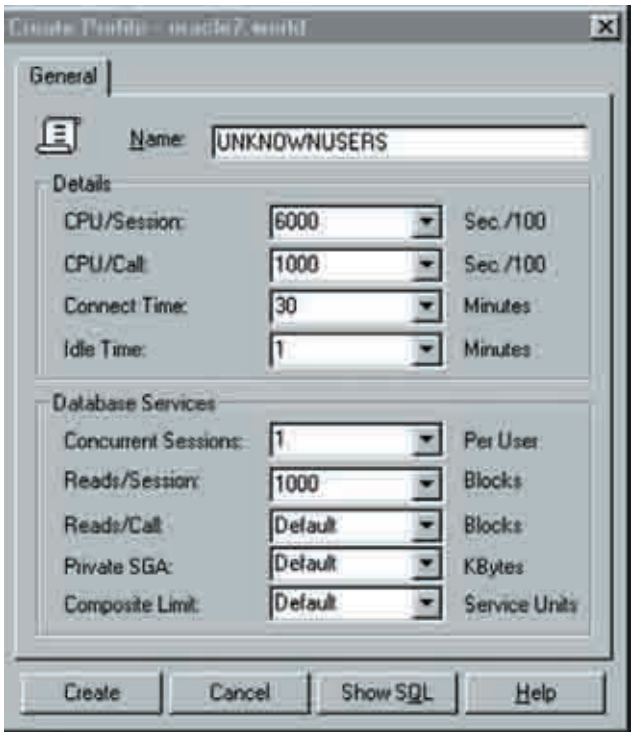
Los modelos de los sistemas de seguridad que usa Oracle y SQL Server se ilustran en las figuras 11-14 y 11-15. Como puede ver, ambos son variantes del modelo general

➤ FIGURA 11-14

Seguridad en Oracle: (a) modelo de seguridad de Oracle, (b) definición del perfil de Oracle, (c) usuario, funciones y perfiles de Oracle



(a) Modelo de seguridad de Oracle

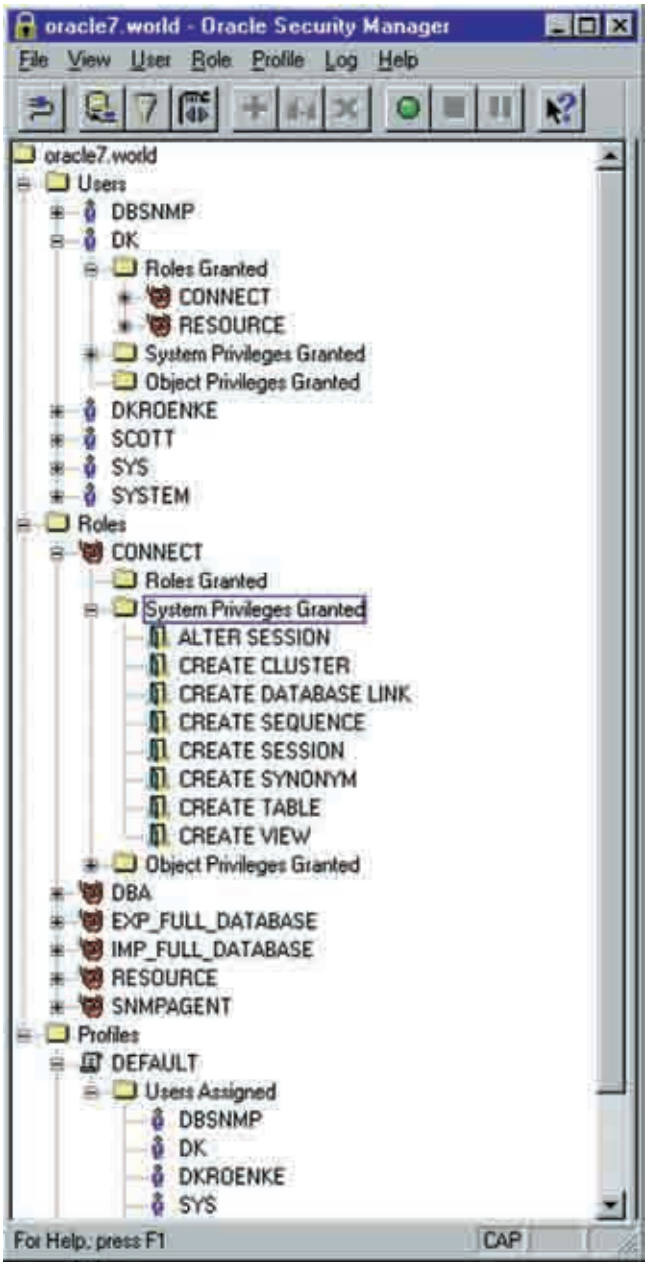


(b)

(continúa)

➤ FIGURA 11-14

(Continuación)



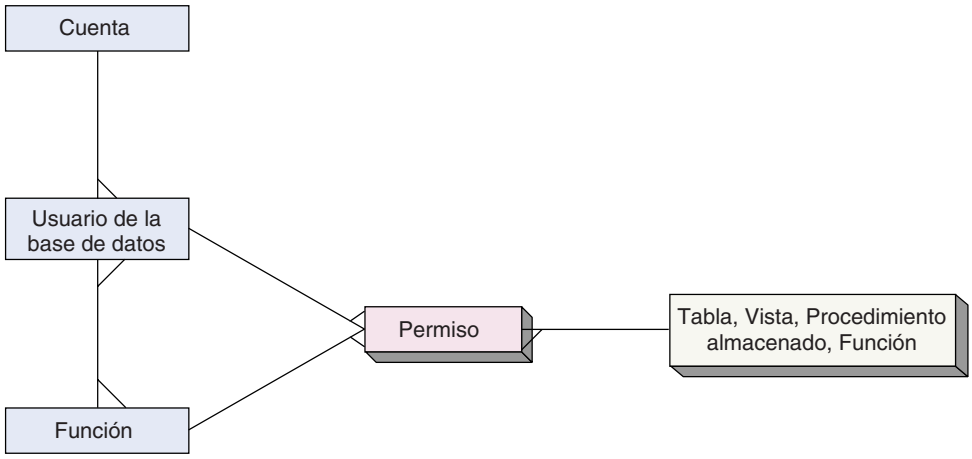
(c)

de la figura 11-13. Considerando el modelo de Oracle en la figura 11-14(a), los usuarios tienen un perfil, que es una especificación de los recursos de los sistemas que se pueden usar. En la figura 11-14(b) se muestra un ejemplo de la definición de perfil. A los usuarios se les pueden asignar muchas funciones y una función se asigna a muchos usuarios. Cada usuario y función tienen muchos privilegios, de los cuales hay dos subtipos: los del objeto que se refieren a las acciones que se pueden llevar a cabo sobre los objetos de la base de datos como tablas, vistas, e índices; y los privilegios de los sistemas que se refieren a las acciones usando los comandos de Oracle.

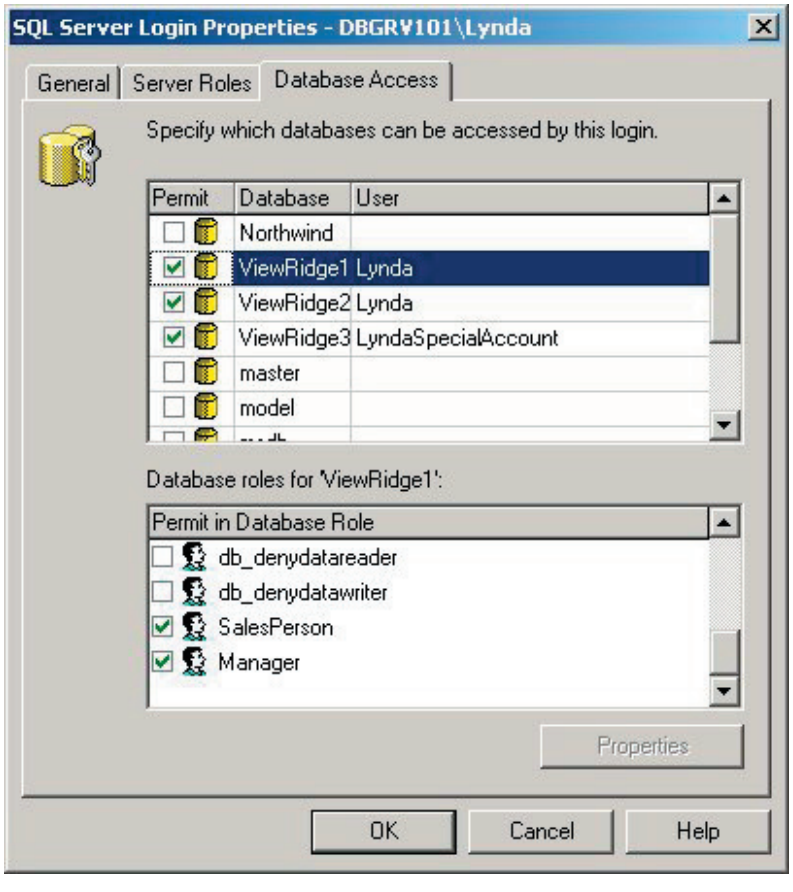
En la figura 11-14(c) se ejemplifica este sistema de seguridad. Observe que al usuario DK se le han asignado funciones para CONNECT y RESOURCE. Si vemos más abajo en la lista, a la función CONNECT se le han otorgado privilegios del sistema como

➤ FIGURA 11-15

Seguridad del SQL Server: (a) un modelo de seguridad del SQL Server, (b) ejemplo de entrada al SQL Server, (c) Ejemplo de la función del SQL Server y (d) ejemplo del usuario del SQL Server



(a)



(b)

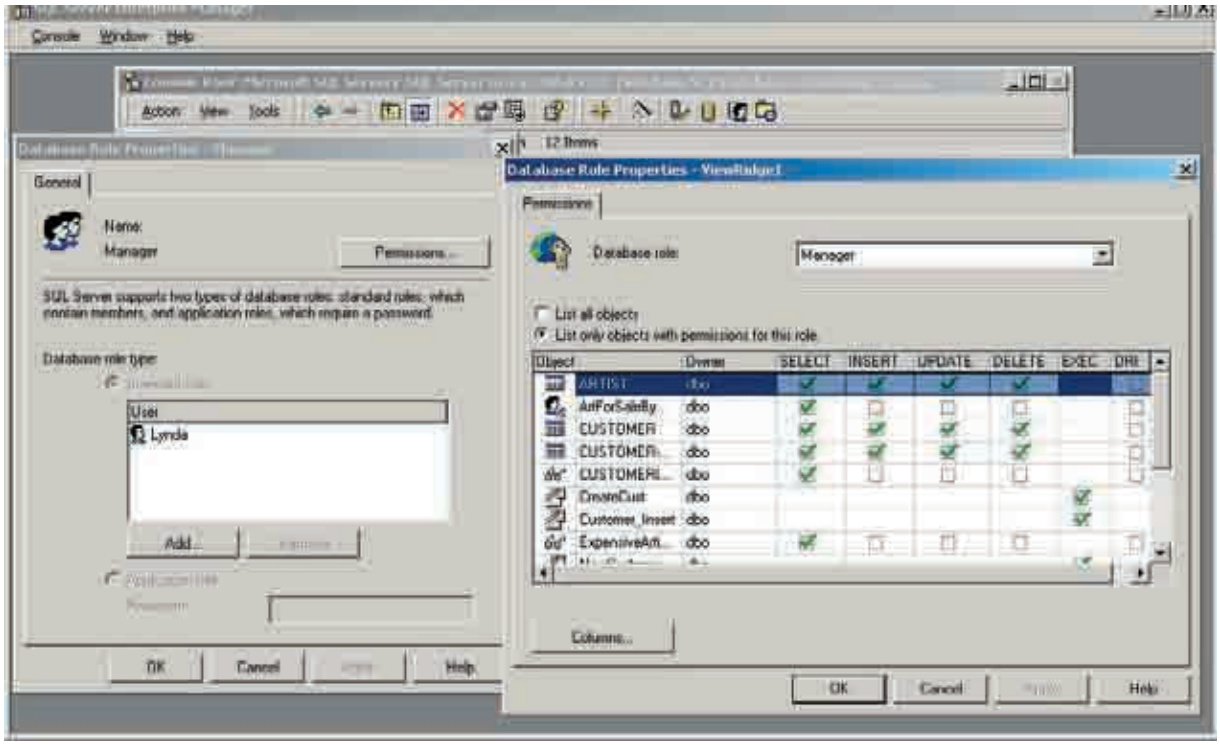
(*continúa*)

ALTER SESSION, CREATE CLUSTER y otros, incluyendo CREATE TABLE. Si vemos más abajo en la lista, el usuario DK ha sido asignado a DEFAULT.

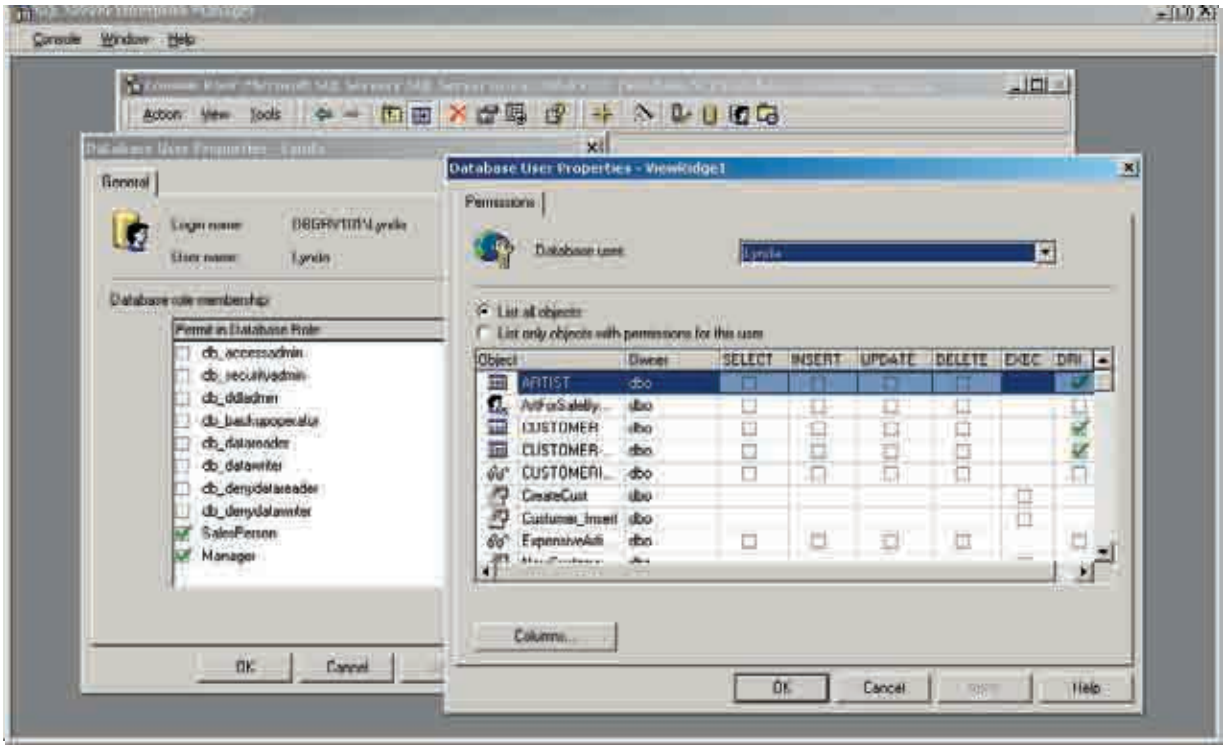
En la figura 11-15(a) se ilustra el esquema de seguridad para SQL Server. Una cuenta (el nombre y la contraseña usadas cuando se registra en SQL Server o en Windows) se puede asociar con una o más combinaciones de base de datos/usuario. En la figura 11-15(b), por ejemplo, al nombre de la cuenta de Lynda en el dominio DBGRV101 se

➤ FIGURA 11-15

(Continuación)



(c)



(d)

le ha permitido el acceso a la base de datos ViewRidge1 con el nombre de usuario Lynda, para ViewRidge2 con el nombre de usuario Lynda y a ViewRidge3 con el nombre del usuario LyndaSpecialAccount. Como se muestra en la figura 11-15(a), se puede asociar más de una entrada con una combinación determinada de base de datos/usuario.

La figura 11-15(c) muestra los permisos de una función llamada Manager (Gerente) en la base de datos ViewRidge1. Un rol puede tener uno o más usuarios, pero en este caso sólo tiene uno: Lynda. Los permisos de la función Manager se muestran con una marca verde en la caja de permisos. EXEC se refiere al permiso de ejecutar un procedimiento almacenado y DRI, al permiso para crear, cambiar o eliminar restricciones declarativas de integridad referencial, tales como CustomerID en CUSTOMER-ARTIST-INT que debe ser un subconjunto de CustomerID en CUSTOMER.EXEC; sólo es aplicable a objetos de procedimiento almacenado, y DRI sólo se aplica a tablas y vistas.

Una base de datos/usuario tiene todos los permisos de las funciones a las cuales está asignado el usuario, así como también cualquier permiso que sólo se le asigna a él. La figura 11-15(d) muestra que al usuario ViewRidge1/Lynda se le han asignado las funciones SalesPerson y Manager, y tendrá todos los permisos de éstas. Además, se le ha permitido crear, modificar y eliminar restricciones DRI en las tablas ARTIST, CUSTOMER y CUSTOMER-ARTIST-INT (el nombre de esta última tabla está incompleto en esta figura).

Seguridad de la aplicación

Si bien los productos DBMS, como Oracle y SQL Server, proporcionan capacidades de seguridad sustanciales de la base de datos, por su misma naturaleza son genéricos. Si la aplicación requiere medidas de seguridad específicas tales como: "Ningún usuario puede ver un renglón de una tabla, o un join de una tabla, que tenga el nombre de otro empleado" los dispositivos del DBMS no serán adecuados. En este caso, se debe aumentar la seguridad del sistema mediante algunas características en la aplicación de la base de datos.

Por ejemplo, como verá en el capítulo 14, la seguridad en las aplicaciones de Internet con frecuencia las proporciona la computadora del servidor Web. Ejecutar la seguridad de la aplicación en este servidor significa que la información de seguridad no necesita ser transmitida a la red.

Para entenderlo mejor, suponga que escribe una aplicación de tal manera que cuando los usuarios aprieten un botón en particular en una página del explorador, se envíe la siguiente solicitud al servidor Web y luego al DBMS:

```
SELECT    *
FROM      EMPLEADO
```

Esta instrucción, por supuesto, regresará a todos los renglones EMPLEADO. Si la seguridad de la aplicación limita a los empleados para que sólo tengan acceso a sus propios datos, entonces un servidor Web podría agregar la cláusula que se muestra abajo de esta consulta:

```
SELECT    *
FROM      EMPLEADO
WHERE EMPLEADO.Nombre = ´<%=SESIÓN("EmpleadoNombre")%>´
```

Como aprenderá en los capítulos 15 y 16, una expresión como ésta ocasionaría que el servidor Web llenara los nombres de los empleados en la cláusula WHERE. Para un usuario inscrito con el nombre 'Benjamin Franklin', la instrucción que resulta de esta expresión es:

```
SELECT    *
FROM      EMPLEADO
WHERE EMPLEADO.Nombre = "Benjamin Franklin"
```

Debido a que el nombre lo inserta un programa en el servidor Web, el usuario del explorador no sabrá qué está ocurriendo y no podrá interferir aunque lo intente.

Este procesamiento de seguridad se puede llevar a cabo en un servidor Web como se muestra aquí, pero también se puede hacer dentro de los mismos programas de aplicación, o se puede escribir como procedimientos almacenados (stored procedures) o disparadores (triggers) para que los ejecute el DBMS en los momentos apropiados.

Esta idea se puede extender mediante el almacenamiento adicional de datos en una base de datos de seguridad al que se tenga acceso a través del servidor Web, o mediante procedimientos almacenados y disparadores. Por ejemplo, esa base de datos de seguridad podría contener las identidades de los usuarios asociadas con los valores adicionales de las cláusulas WHERE. Por ejemplo, suponga que los usuarios en el departamento de personal pueden acceder sólo a sus propios datos. Los predicados para las cláusulas apropiadas WHERE se podrían almacenar en la base de datos de seguridad, que las lea el programa de aplicación, y se anexen a las instrucciones SELECT de SQL cuando sea necesario.

Existen muchas otras posibilidades para extender la seguridad DBMS con el procesamiento de la aplicación. Sin embargo, en general, es mejor que primero use usted las características de seguridad del DBMS. Sólo si son inadecuadas para los requerimientos, deberá agregarlas con el código de la aplicación. Mientras más cercana esté la aplicación de seguridad a la información, menos probabilidades existen de que ocurra alguna infiltración. Asimismo, usar las características de seguridad del DBMS es más rápido, barato y probablemente se obtienen resultados de mayor calidad que desarrollando las propias.

➤ RECUPERACIÓN DE LA BASE DE DATOS

Los sistemas de la computadora pueden fallar. Los programas tienen defectos; los procedimientos humanos conllevan errores, y las personas se equivocan. Todas estas fallas pueden ocurrir y de hecho ocurren en las aplicaciones de la base de datos. Puesto que una base de datos la comparten muchas personas, y que con frecuencia esto es un elemento clave de las operaciones de la organización, es importante recuperarla lo más rápido posible.

Hay varios problemas que conviene señalar. Primero, desde el punto de vista del negocio, las operaciones deben continuar. Por ejemplo, los pedidos del cliente, las transacciones financieras y las listas de empaque deben llenarse en forma manual. Más tarde, cuando se opera de nuevo la aplicación de la base de datos, se pueden introducir datos nuevos. Segundo, el personal de operaciones de la computadora debe restaurar el sistema para su uso tan rápido como sea posible para que funcione como antes del desperfecto. Tercero, los usuarios deben saber qué hacer cuando el sistema vuelva a estar disponible. Quizá sea necesario ingresar otra vez datos y los usuarios deben saber qué tanto necesitan regresar.

Cuando ocurren fallas es imposible arreglar el problema y reanudar el procesamiento. Incluso si no se perdieron datos durante la falla (lo que nos hace suponer que todos los tipos de memoria son no volátiles, lo que constituye una suposición irreal), el tiempo y el horario del procesamiento de la computadora son muy complejos para ser recreados en forma exacta. Se requieren enormes cantidades de datos y de procesamiento para que el sistema operativo pueda reiniciar el procesamiento exactamente en donde se interrumpió. Es imposible revertir el reloj y poner todos los electrones en la misma configuración donde estaban en el momento de la falla. Hay dos vías posibles: recuperar a través del reprocesamiento, y recuperar a través de revertir/avanzar (rollback/rollforward).

RECUPERACIÓN MEDIANTE REPROCESAMIENTO.

Puesto que el procesamiento no se puede reanudar en un punto preciso, la mejor alternativa es regresar a un punto conocido y reprocesar la carga de trabajo desde ahí. La forma más simple de este tipo de recuperación es hacer periódicamente una copia de la base de datos **(guar-**

dar la base de datos) y conservar un registro de todas las transacciones que se han procesado desde que se guardó. Así, cuando hay una falla, el personal puede restaurar la base de datos que se guardó y reprocesar todas las transacciones.

Por desgracia, esta estrategia simple no es factible. Primero, las transacciones del reprocesamiento tardan el mismo tiempo que si se procesan por primera vez. Si la computadora se programa con carga muy pesada, puede ser que el sistema nunca se ponga al corriente.

Segundo, cuando las transacciones se procesan concurrentemente, los sucesos son asincrónicos. Las ligeras variaciones en la actividad humana —por ejemplo un usuario que inserta un disco flexible lentamente, o uno que lee un mensaje de correo electrónico antes de responder a una aplicación inmediata— pueden cambiar el orden de la ejecución de las transacciones concurrentes. Por lo tanto, mientras que el cliente A viaja en el último asiento de un vuelo durante el procesamiento original, el cliente B puede ir en el último asiento durante el reprocesamiento. Por estas razones, normalmente no es el reprocesamiento una forma viable de recuperación a partir de una falla en sistemas de procesamiento concurrente.

RECUPERACIÓN MEDIANTE ROLLBACK/ROLLFORWARD. Una segunda opción es hacer una copia de la base de datos (guardar la base de datos) periódicamente y llevar un registro de los cambios que se hicieron y guardaron en la base de datos con las transacciones. Así, cuando hay una falla, se puede optar por uno de los dos métodos. Si se usa el primer método, llamado **rollforward,** la base de datos se restaura usando la información que ha sido guardada y todas las transacciones válidas se aplican nuevamente (no estamos reprocesando las transacciones, ni los programas de aplicación están implicados en el avance. En lugar de eso, los cambios procesados, como se grabaron en el registro, se aplican nuevamente).

El segundo método es **rollback,** con el cual deshacemos los cambios que se realizaron con las transacciones erróneas, o parcialmente procesadas. De esta manera, las transacciones válidas que estaban en proceso en el momento de la falla se reinician.

Ambos métodos requieren que se mantenga un **registro** de las transacciones (log). Éste contiene los cambios de datos en orden cronológico. Las transacciones se deberán escribir en el log antes de que se apliquen a la base de datos. De ese modo, si el sistema se colapsa entre el momento en que una transacción se registra y el momento en que se aplica, entonces en el peor de los casos existirá un registro de una transacción no aplicada. Si, por otro lado, las transacciones se aplicaron antes de que fueran registradas, sería posible (aunque también indeseable) cambiar la base de datos, pero sin tener el registro del cambio. Si esto sucede, un usuario no precavido podría ingresar una transacción ya terminada.

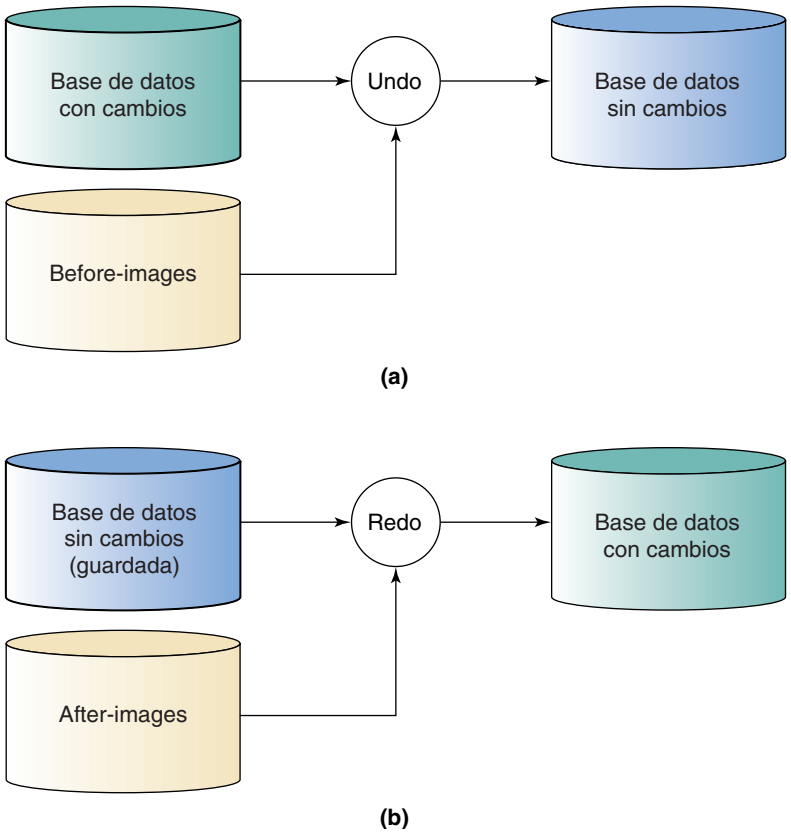
Cuando ocurre una falla, el log se usa tanto para rehacer (redo) como para deshacer (undo) las transacciones, como se muestra en la figura 11-16. Para deshacer (undo) una transacción, el log debe contener una copia de cada registro de la base de datos (o página) antes de que ésta haya cambiado. Dichos registros se llaman **before-images.** Una transacción se deshace aplicando las before-images de todos sus cambios en la base de datos.

Para rehacer (redo) una transacción el log debe contener una copia de cada registro de la base de datos (o página) después de que cambió. Estos registros se llaman **after-images.** Una transacción se rehace aplicando las after-images de todos sus cambios en la base de datos. Los posibles elementos de datos de un log de transacción se muestran en la figura 11-17(a).

Para este ejemplo de log, cada transacción tiene un nombre único con fines de identificación. Además, todas las imágenes de una transacción determinada están encadenadas mediante apuntadores. Un apuntador apunta al cambio previo hecho por la transacción (reverse pointer) y el otro apunta al siguiente cambio (forward pointer). Un cero en el campo del apuntador significa que es el final de la lista. El subsistema de recuperación DBMS usa estos apuntadores para localizar todos los registros para una transacción en particular. La figura 11-17(b) muestra un ejemplo de los encadenamientos de los registros del log.

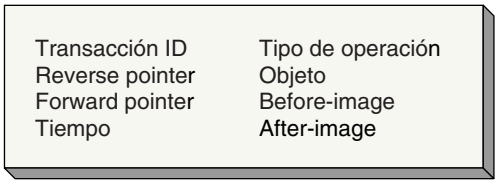
➤ FIGURA 11-16

Transacciones deshacer y rehacer: (a) se eliminan los cambios en la base de datos (rollback), y (b) se reaplican los cambios en la base de datos (rollforward)



➤ FIGURA 11-17

Log de una transacción: (a) elementos de datos en un log, y (b) ejemplo del log para tres transacciones



Número de registro relativo

1	OT1	0	2	11:42	INICIO			
2	OT1	1	4	11:43	MODIFICAR	CLIENTE 100	(valor viejo)	(valor nuevo)
3	OT2	0	8	11:46	INICIO			
4	OT1	2	5	11:47	MODIFICAR	SP AA	(valor viejo)	(valor nuevo)
5	OT1	4	7	11:47	INSERTAR	PEDIDO 11		(valor nuevo)
6	CT1	0	9	11:48	INICIO			
7	OT1	5	0	11:49	CONFIRMAR			
8	OT2	3	0	11:50	CONFIRMAR			
9	CT1	6	10	11:51	MODIFICAR	SP BB	(valor viejo)	(valor nuevo)
10	CT1	9	0	11:51	CONFIRMAR			

(b)

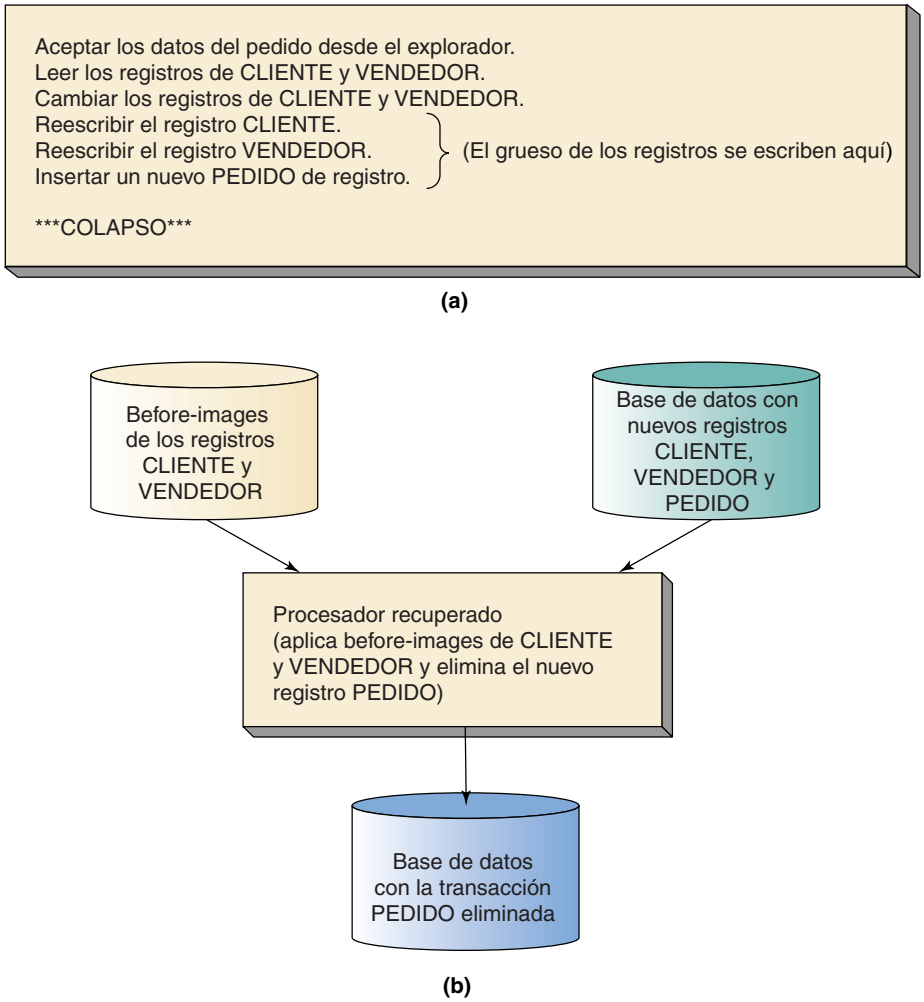
Otros elementos de datos en el log son: el tipo de operación (START señala el comienzo de una transacción y COMMIT termina una transacción, liberando así todos los locks que había); el objeto que actuó, por ejemplo el tipo e identificador de registro; y, por último, las before-images y las after-images.

Si consideramos un registro con before-images y con after-images, las acciones undo y redo son directas. Para deshacer la transacción en la figura 11-18 el procesador de recuperación simplemente remplaza cada registro cambiado con su before-image. Cuando se han restaurado todas las before-images se deshace la transacción. Para rehacer una transacción el procesador de recuperación inicia con la versión de la base de datos en el momento en que comenzó la transacción y se aplica a todas las after-images. Como se indicó, esta acción supone que hay una versión anterior de la base de datos desde una base de datos guardada.

Restaurar una base de datos a su versión más reciente y reaplicar todas las transacciones puede requerir un procesamiento considerable. Para reducir el retraso los productos DBMS algunas veces usan puntos de verificación. Un **checkpoint** es un punto de sincronización entre la base de datos y el log de transacciones. Para ejecutar un checkpoint el DBMS rechaza nuevas peticiones, finaliza las peticiones de procesamiento pendientes y escribe sus buffers en un disco. El DBMS espera hasta que el sistema operativo le notifica que todos los requerimientos escritos pendientes en la base de datos y en el log se han terminado con éxito. En este punto, el log y la base de datos están sincronizados. Así, un log del checkpoint se escribe en el log. Más tarde, la base de datos se puede recuperar a partir del checkpoint y de las after-images para las transacciones que comenzaron después de que fue necesario aplicar el checkpoint.

➤ FIGURA 11-18

Ejemplo de una estrategia de recuperación: (a) la transacción PEDIDO, y (b) procesamiento de recuperación para deshacer un registro de PEDIDO



Los checkpoints son operaciones baratas y es factible usar tres o cuatro (o más) por hora. De esta manera, no se necesitan más de 15 o 20 minutos de procesamiento para la recuperación. La mayoría de los productos DBMS se verifican a sí mismos automáticamente, lo que hace innecesaria la intervención humana.

En los dos capítulos siguientes verá ejemplos específicos de técnicas de respaldo y recuperación para Oracle y SQL Server. Por ahora, sólo necesita entender las ideas básicas y darse cuenta de que es responsabilidad del DBA asegurarse que los planes de respaldo y recuperación se lleven a cabo, que la base de datos se guarde, y que los registros se hayan generado como se requirió.

Administración del DBMS

Además de encargarse del manejo de datos y de la estructura de la base de datos, el DBA tiene que administrar el mismo DBMS. Debe recopilar y analizar las estadísticas con respecto al desempeño de los sistemas e identificar las áreas potenciales de problema. Tenga en cuenta que la base de datos está sirviendo a muchos grupos de usuarios. El DBA necesita investigar todas las quejas con respecto al tiempo de respuesta de los sistemas, la exactitud de los datos, la facilidad de su uso, etcétera. Si es necesario hacer cambios, el DBA debe planearlos e implementarlos.

El DBA debe monitorear periódicamente la actividad de los usuarios en la base de datos. Los productos DBMS incluyen características para recolectar y hacer reportes estadísticos. Por ejemplo, algunos de éstos pueden indicar qué usuarios han estado activos, qué archivos y quizás cuáles elementos de los datos se han usado, y qué métodos de acceso se han empleado. Los porcentajes de error y los tipos de fallas también se pueden capturar y reportar. El DBA analiza esta información para determinar si se necesita un cambio en el diseño de la base de datos para mejorar el desempeño, o para facilitar las tareas de los usuarios. Si es necesario el cambio, el DBA se asegurará de que se lleve a cabo.

El DBA debe analizar las estadísticas en cuanto a tiempo de funcionamiento y desempeño de la base de datos. Cuando se identifique un problema de desempeño (ya sea mediante un reporte o una queja del usuario), el DBA deberá determinar si procede una modificación de la estructura de la base de datos o del sistema. Algunos ejemplos de posibles modificaciones a la estructura son el establecimiento de nuevas llaves, la purga de datos, la eliminación de llaves y la asignación de nuevas relaciones entre los objetos.

Cuando el vendedor del DBMS comienza a anunciar las características de un nuevo producto, el DBA debe considerarlas a la luz de las necesidades de toda la comunidad de usuarios. Si decide incorporar las nuevas características, tendrá que notificarlo a los programadores y capacitarlos. Por consiguiente, el DBA debe administrar y controlar los cambios en el DBMS, así como en la estructura de la base de datos.

Otros cambios en el sistema de los cuales el DBA es responsable varían dependiendo del producto DBMS, así como del software y hardware en uso. Por ejemplo, los cambios en el software (tal como el sistema operativo o el Servidor Web) pueden significar que algunas características del DBMS, las funciones, o los parámetros se deben cambiar. Por lo tanto, el DBA también debe adaptar el producto DBMS al software en uso.

Las opciones del DBMS (tales como los niveles de aislamiento de las transacciones) se eligen inicialmente cuando se sabe muy poco acerca de cómo se desempeñará el sistema en el ambiente del usuario. Por lo tanto, la experiencia operacional y el análisis de desempeño en un periodo determinado pueden revelar si son necesarios los cambios. Aun cuando el desempeño parezca aceptable, el DBA quizá quiera modificar las opciones y observar el efecto en el desempeño. Este proceso se llama *afinación,* u *optimización,* del sistema. La figura 11-19 resume las responsabilidades del DBA en cuanto a la administración del producto DBMS.

➤ FIGURA 11-19

Resumen de las responsabilidades del DBA en la administración del DBMS

- Generar reportes sobre el rendimiento de las aplicaciones de la base de datos
- Investigar las quejas de los usuarios acerca del desempeño
- Evaluar la necesidad de cambios en la estructura de la base de datos o en el diseño de la aplicación
- Modificar la estructura de la base de datos
- Evaluar e implementar las nuevas características del DBMS
- Afinar u optimizar el DBMS

Mantenimiento del repositorio de datos

Considere una aplicación de la base de datos en Internet grande y activa, como las que usan las compañías comerciales a través del correo electrónico; por ejemplo, una aplicación que usa una compañía que vende música en Internet. Este sistema puede implicar información de varias bases de datos diferentes, docenas de diferentes páginas Web, y cientos o miles de usuarios.

Suponga que la compañía que está usando esta aplicación decide expandir su línea de productos para incluir la venta de artículos deportivos. El administrador de esta compañía podría pedir al DBA que desarrolle un estimado de tiempo y de otros recursos que se requieran para modificar la aplicación de la base de datos con el fin de agregar esta nueva línea de productos.

Para que el DBA responda a este requerimiento necesita los metadatos exactos de la base, de las aplicaciones y sus componentes, de los usuarios y sus derechos y privilegios, y con respecto a otros elementos del sistema. La base de datos contiene algo de estos metadatos en las tablas del sistema, pero no son adecuados para responder las preguntas que ha planteado el administrador general. El DBA necesita metadatos adicionales con respecto a los objetos COM y ActiveX, los procedimientos de escritura y las funciones, las páginas ASP, hojas de estilo, definición de tipos de documentos y cosas por el estilo. Además, si bien es cierto que los mecanismos de seguridad del DBMS hacen documentos de usuarios, grupos y privilegios, lo realizan de una forma sumamente estructurada y con frecuencia inconveniente.

Por todos estos motivos, muchas organizaciones desarrollan y mantienen **repositorios de datos,** los cuales son conjuntos de metadatos acerca de la base, aplicaciones de ésta, páginas Web, usuarios y otros componentes de la aplicación. El repositorio puede ser virtual porque se compone de metadatos de muchas fuentes diferentes: el DBMS, software de control de versiones, las bibliotecas de código, la generación de páginas Web, las herramientas de edición, etc. Asimismo, el repositorio de datos puede ser un producto integrado de la herramienta CASE, o de otras compañías tales como Microsoft u Oracle.

De cualquier manera, el momento en que el DBA debe pensar cómo construir un tipo de repositorio es mucho antes de que el administrador haga preguntas. De hecho, el repositorio se debe construir conforme se desarrolle el sistema y debería considerarse como parte importante de los productos a entregar del sistema. De lo contrario, el DBA estará siempre tratando de mantener las aplicaciones que están en uso, adaptándolas a las nuevas necesidades y de algún modo juntando los metadatos para formar un repositorio.

Los mejores repositorios son **activos,** son parte del proceso de desarrollo del sistema, de tal manera que los metadatos se originan automáticamente conforme se crean los componentes del sistema. Menos deseables, pero más eficaces, son los **repositorios pasivos** que se llenan sólo cuando alguien se toma el tiempo de generar los metadatos necesarios y colocarlos en el repositorio.

Internet ha creado enormes oportunidades para que los negocios expandan su base de clientes y aumenten sus ventas y ganancias. Las bases de datos y las aplicaciones

de la base que manejan estas compañías son un elemento esencial de su éxito. Por desgracia existirán organizaciones cuyo crecimiento se obstaculice por su falta de habilidad para aumentar sus aplicaciones o adaptarlas a las necesidades de cambio. Con frecuencia es más fácil construir un nuevo sistema que adaptar uno que ya existe; ciertamente construir un nuevo sistema que se integre a uno viejo mientras lo remplaza puede ser muy difícil.

➤ RESUMEN

Las bases de datos multiusuarios plantean problemas difíciles para las organizaciones que las crean y las usan, y la mayoría han creado una oficina de administración de la base de datos para asegurarse de que esos problemas se resuelvan. En este texto, el término *administrador de la base de datos* se refiere a la persona u oficina involucrada en una sola base de datos. El término *administrador de datos* se usa para describir una función similar que tiene que ver con todos los datos de los activos de una organización. En el capítulo 17 se analizará la administración de datos. Las funciones principales del administrador de la base de datos se enumeran en la figura 11-1.

El DBA se involucra en el desarrollo inicial de las estructuras de la base de datos y tiene el control de la configuración cuando se necesite hacer cambios. Documentar de manera precisa la estructura y las modificaciones que se hagan es una función importante del DBA.

La meta del control de concurrencia es asegurar que el trabajo de un usuario no interfiera con el de otro usuario. Ninguna técnica para el control de concurrencia es ideal para todas las circunstancias. Se necesita hacer acuerdos entre el nivel de protección y el rendimiento eficaz. Una transacción o unidad lógica de trabajo es una serie de acciones que se efectuaron en la base de datos, la cual tiene lugar como una unidad atómica; ya sea que todas o ninguna ocurren. La actividad de las transacciones concurrentes está soportada en el servidor de la base de datos. En algunos casos, se pueden perder las actualizaciones si no se controlan las transacciones concurrentes. Otro problema de concurrencia es el de las lecturas inconsistentes.

Para evitar problemas de concurrencia se bloquean los elementos de la base de datos. Los locks implícitos se ponen a través del DBMS; los explícitos, mediante el programa de aplicación. El tamaño del recurso bloqueado se llama granularidad del lock. Un lock excluyente prohíbe a otros usuarios leer el recurso bloqueado; un lock compartido permite a otros usuarios leer el recurso bloqueado, pero no pueden actualizarlo.

A dos transacciones que funcionan coincidentemente y generan resultados que son consistentes con los que se obtendrán por separado se les llama transacciones serializables. El lock de dos fases, en el cual los locks entran en una fase creciente y se liberan en una fase decreciente, es un esquema para la serialización. Un caso especial de lock de dos fases es adquirir locks durante la transacción, pero no liberar ninguno hasta que ésta termine.

El deadlock, o abrazo mortal, tiene lugar cuando dos transacciones están cada una esperando un recurso que la otra transacción conserva. El deadlock se puede evitar requiriendo que las transacciones tengan todos los locks al mismo tiempo; una vez que esto ocurre, la única manera para remediarlo es abortar una de las transacciones (y regresar al trabajo parcialmente terminado). El locking optimista supone que no ocurrirá ningún conflicto de transacción y enfrenta las consecuencias cuando las hay. El locking pesimista supone que el conflicto ocurrirá, así que se evita con anticipación mediante locks. En general, se prefiere el lock optimista en Internet y en muchas aplicaciones de intranet.

La mayoría de las aplicaciones no declaran explícitamente los locks. En lugar de eso, señalan los límites de la transacción con instrucciones de transacción BEGIN, COMMIT y ROLLBACK e indican el comportamiento concurrente que quieren. Entonces el DBMS bloquea la aplicación, lo que dará como resultado el comportamiento que se requiere.

Una transacción ACID es atómica, consistente, aislada y durable. Durable significa que los cambios de la base de datos son permanentes. Consistente puede significar ya sea a nivel de instrucción, o consistencia a nivel de transacción. Con la consistencia a nivel de transacción, una transacción puede no ver sus propios cambios. El estándar SQL de 1992 define cuatro niveles de aislamiento de transacción: leer sin confirmar, leer confirmando, leer repetible, y serializable. Las características de cada una están resumidas en la figura 11-10.

Un cursor es un apuntador a un conjunto de registros. Sobresalen cuatro tipos: rollforward, estático, keyset, y dinámico. Los programadores deben seleccionar los niveles de aislamiento y tipos de cursores que son apropiados para la carga de trabajo de su aplicación y para el producto DBMS que se están usando.

El objetivo de la seguridad de la base de datos es asegurar que sólo los usuarios autorizados puedan ejecutar actividades permitidas en momentos específicos. Para desarrollar una seguridad eficaz de la base de datos, se deben establecer los derechos de procesamiento y las responsabilidades de todos los usuarios.

Los productos DBMS proporcionan dispositivos de seguridad. La mayoría implican la declaración de los usuarios, objetos que deben ser protegidos y permisos o privilegios de los objetos. Casi todos los productos DBMS usan alguna forma de seguridad, tal como nombre del usuario y contraseña. La seguridad del DBMS se puede aumentar mediante la seguridad de la aplicación.

Si el sistema falla, se debe restaurar esa base de datos a un estado en que se pueda usar lo más pronto posible. Las transacciones en proceso en el momento de la falla se deben reaplicar o reiniciar. Mientras que en algunos casos la recuperación se puede llevar a cabo mediante el reprocesamiento, casi siempre se prefiere el uso de logs, rollback y rollforward. Los ckeckpoints se pueden consultar para reducir la cantidad de trabajo que se debe repetir después de una falla.

Además de estas tareas. El DBA administra el mismo producto DBMS, midiendo el desempeño de la aplicación de la base de datos y evaluando las necesidades de cambio en la estructura, o afinando el desempeño del DBMS. El DBA también se debe asegurar que los nuevos dispositivos del DBMS sean evaluados y usados correctamente. Por último, el DBA es responsable de conservar el repositorio de la base de datos.

➤ PREGUNTAS DEL GRUPO I

11.1 Describa brevemente cinco problemas difíciles para empresas que crean y usan bases de datos multiusuarios.

11.2 Explique la diferencia entre un administrador de base de datos y un administrador de datos.

11.3 Enumere siete tareas importantes de un DBA.

11.4 Resuma las responsabilidades del DBA en cuanto a la administración de la estructura de la base de datos.

11.5 ¿Qué es el control de configuración? ¿Por qué es necesario?

11.6 Explique el significado de la palabra *inapropiadamente* en la frase “que el trabajo de un usuario no afecte inapropiadamente el de otro”.

11.7 Explique las inconveniencias del control de concurrencia.

11.8 Defina una transacción atómica y explique por qué es importante la atomicidad.

11.9 Explique la diferencia entre transacciones concurrentes y simultáneas. ¿Cuántas CPU se requieren para las transacciones simultáneas?

11.10 Dé un ejemplo, diferente al de este texto, sobre el problema de actualización con pérdida.

11.11 Explique la diferencia entre un lock explícito y uno implícito.

11.12 ¿Qué es la granularidad del lock?

11.13 Explique la diferencia entre un lock excluyente y uno compartido.

11.14 Explique el lock de dos fases.

11.15 ¿De qué manera se relaciona la liberación de todos los locks al final de la transacción con el lock de dos fases?

11.16 En general, ¿cómo se deberían definir las fronteras de una transacción?

11.17 ¿Qué es el deadlock? ¿Cómo se puede evitar? ¿Cómo se puede solucionar cuando ocurre?

11.18 Explique las diferencias entre lock optimista y pesimista.

11.19 Explique los beneficios de señalar fronteras de transacción, declarar las características del lock, y dejar que el DBMS los ponga.

11.20 Explique el uso de los enunciados BEGIN, COMMIT Y ROLLBACK TRANSACTION.

11.21 Explique el significado de la expresión: transacción ACID.

11.22 Describa consistencia a nivel de instrucción.

11.23 Describa consistencia a nivel de transacción.

11.24 ¿Cuál es el propósito de los niveles de aislamiento de la transacción?

11.25 Explique el nivel de aislamiento de la lectura no confirmada. Exponga un ejemplo de su uso.

11.26 Explique el nivel de aislamiento de la lectura confirmada. Dé un ejemplo sobre su uso.

11.27 Explique el nivel de aislamiento de la lectura repetible. Proporcione un ejemplo de su uso.

11.28 Explique el nivel de aislamiento serializable. Mencione un ejemplo de su uso.

11.29 Explique el término *cursor*.

11.30 Explique por qué una transacción puede tener muchos cursores. También, ¿cómo es posible que una transacción pueda tener más de un cursor en una tabla determinada?

11.31 ¿Cuál es la ventaja de usar diferentes tipos de cursores?

11.32 Explique los cursores forward only. Dé un ejemplo de su uso.

11.33 Explique los cursores estáticos. Proporcione un ejemplo de su uso.

11.34 Explique los cursores de keyset. Mencione un ejemplo de su uso.

11.35 Explique los cursores dinámicos. Dé un ejemplo de su uso.

11.36 ¿Qué sucede si usted no establece en el DBMS el nivel de aislamiento de la transacción y el tipo de cursor? ¿Esto es bueno o malo?

11.37 Explique la necesidad de definir los derechos y las responsabilidades del procesamiento. ¿Cómo se imponen estas responsabilidades?

11.38 Explique las relaciones de USUARIOS, GRUPOS, PERMISOS, Y OBJETOS para un sistema de seguridad de una base de datos genérica.

11.39 Describa las ventajas y desventajas de la seguridad que proporciona el DBMS.

11.40 Describa las ventajas y desventajas de seguridad que proporciona la aplicación.

11.41 Explique cómo se puede recuperar una base de datos a través del reprocesamiento. ¿Por qué en general no es factible?

11.42 Defina *rollback* y *rollforward*.

11.43 ¿Por qué es importante escribir en el registro antes de cambiar los valores de la base de datos?

11.44 Describa el proceso rollback. ¿Bajo qué condiciones se debe usar?

11.45 Describa el proceso rollforward. ¿Bajo qué condiciones se debe usar?

11.46 ¿Cuál es la ventaja de establecer checkpoints de una base de datos?

11.47 Resuma las responsabilidades del DBA en la administración del DBMS.

11.48 ¿Qué es un repositorio de datos? ¿Un repositorio de datos pasivo? ¿Un repositorio de datos activo?

11.49 Explique por qué es importante un repositorio de datos. ¿Qué puede suceder si no hay uno?

➤ PREGUNTAS DEL GRUPO II

11.50 Visite www.microsoft.com y busque información acerca de los niveles de aislamiento y tipos de cursores. Por ahora, ignore la información acerca de RDS, ADO, ODBC, Y OLE/DB y concéntrese en las características y funciones del SQL Server. Compare y contraste esas capacidades con las que describimos.

11.51 Visite www.oracle.com y busque información acerca de los niveles de aislamiento y tipos de cursores. Ignore la información acerca de ODBC y concéntrese en las características y funciones de Oracle. Compare y contraste esas capacidades con las que describimos en este texto.

11.52 Describa las ventajas y desventajas de la seguridad del nombre del usuario y la contraseña. ¿De qué manera los usuarios pueden descuidar sus identidades? ¿Cómo puede comprometer la seguridad de la base de datos ese descuido? ¿Qué pasos se podrían seguir para reducir el riesgo de esos problemas?

11.53 Busque la Web de las herramientas CASE que proporcionan repositorios y productos para repositorio. Decida cuál es el mejor y enumere sus funciones y características principales. Explique cómo se podrían usar las funciones y características para ayudar al DBA de una compañía comercial que se anuncia en Internet a que agregue nuevas líneas de productos a su negocio.

➤ PROYECTO

A. Considere la vista Cliente de la figura 10-4.

1. Suponga que usted está desarrollando una aplicación para crear nuevas instancias de esta vista. ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores involucrados y recomiende un tipo de cursor para cada uno.
2. Suponga que está desarrollando una aplicación para modificar los valores (sólo valores, no relaciones) de esta vista. ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores y recomiende un tipo de cursor para cada uno.
3. Suponga que está desarrollando una aplicación para modificar tanto los valores de datos como las relaciones en esta vista. ¿Cómo cambia su respuesta a la pregunta A.2?
4. Suponga que está desarrollando una aplicación para eliminar instancias de clientes inactivos (definidos como clientes que nunca han comprado arte). ¿Qué nivel de aislamiento de transacción usaría? Nombre los cursores implicados y recomiende un tipo de cursor para cada uno.

➤ PREGUNTAS DEL PROYECTO FIREDUP

A. Suponga que la compañía FiredUp lo ha contratado como un consultor de base de datos para desarrollar su base de datos operacional, considerando las cuatro tablas descritas al final del capítulo 9. Suponga que el personal de FiredUp son los dos dueños, un administrador de oficina, un técnico en reparaciones y dos empleados de producción. El administrador de la oficina procesa todas las formas de registro. El técnico en reparaciones introduce la información de reparación, y los empleados de producción, los datos de las estufas que han producido. Prepare un memorándum de tres a cinco páginas, dirigido a la gerencia de FiredUp, con los siguientes puntos:

1. La necesidad de contar con un administrador de la base de datos en la compañía FiredUp.
2. Su recomendación sobre quién debe servir como administrador de la base de datos. Suponga que FiredUp no es lo suficientemente grande como para contratar un administrador de la base de datos de tiempo completo.
3. Usando la figura 11-1 como guía, describa la naturaleza de las actividades de la administración de la base de datos de FiredUp. Como consultor audaz, tenga en cuenta que puede recomendarse a sí mismo para desempeñar algunas de las funciones del DBA.

B. Para los empleados que se mencionan en la pregunta A, defina usuarios, grupos y permisos de información en las cuatro tablas que se describen al final del capítulo 9. Utilice el esquema de seguridad que se muestra en la figura 11-13 como ejemplo. Nuevamente, no olvide incluirse usted mismo.

C. Considere REGISTRO_VISTA y ESTUFA_REPARACIÓN_VISTA definidas al final del capítulo 10.

1. Mencione un ejemplo de una lectura sucia, una lectura no repetible y una lectura fantasma cuando se procesa REGISTRO_VISTA.
2. ¿Qué medidas de control de concurrencia cree que serían apropiadas para una transacción que actualice REGISTRO_VISTA? ¿Qué medidas de control de concurrencia piensa usted que serían apropiadas para una transacción que elimine la información de REGISTRO_VISTA? Establezca sus hipótesis.
3. Responda la pregunta 2 en el caso de CLIENTE_VISTA. Justifique cualquier diferencia de su respuesta con la pregunta 2.
4. ¿Qué nivel de aislamiento usaría para una transacción que imprime un reporte listando todas las reparaciones de estufas durante un periodo determinado? ¿Qué tipo de cursor usaría?

CAPÍTULO 12

Administración de bases de datos con Oracle

Oracle es un DBMS (Database Management System, por sus siglas en inglés) poderoso y robusto que funciona en muchos sistemas operativos diferentes, incluyendo Windows 98, Windows 2000, diversas variantes de UNIX, diferentes sistemas operativos de macrocomputadoras, y Linux. Éste es el DBMS más popular del mundo y tiene una larga historia de desarrollo y uso. Oracle muestra al programador mucha de su tecnología y consecuentemente puede afinarse y ajustarse de diversas maneras.

Sin embargo, esto quiere decir que Oracle puede ser difícil de instalar y hay mucho que aprender. Una muestra de la amplitud de Oracle es que uno de sus referentes más populares, *Oracle 8i, The Complete Reference*, escrito por Loney y Koch, tiene más de 1,300 páginas y no abarca todo. Además, puede ser necesario modificar las técnicas que funcionan con una versión de Oracle en un sistema operativo, cuando se esté trabajando con una versión en un sistema operativo diferente. Tendrá que ser paciente con Oracle y consigo mismo y estar consciente de que no dominará este tema de la noche a la mañana.

Existen muchas configuraciones de la serie Oracle. Para empezar, hay dos versiones diferentes del motor DBMS de Oracle: Oracle personal y Oracle empresarial. Además, existen Formas y Reportes y también el Diseñador Oracle, así como un sistema central de herramientas para la publicación de las bases de datos de Oracle en la Web. Agregue a lo anterior la necesidad que tienen de operar en muchos sistemas operativos diferentes, en redes mediante diversos protocolos de comunicación, y verá que aprenderlo es considerablemente complejo.

Oracle SQL Plus es un programa de utililería para el procesamiento de SQL y la creación de componentes tales como los procedimientos almacenados (stored procedures) y los disparadores (triggers). Asimismo, es un componente constante a lo largo de todas estas configuraciones de productos. Por lo anterior concentraremos toda la atención en éste. El último, el PL/SQL, es un lenguaje de programación que agrega construcciones de programación al lenguaje SQL. Emplearemos el PL/SQL para crear tablas, restricciones, relaciones, vistas, procedimientos almacenados (stored procedures) y disparadores (triggers).

Este capítulo utiliza el ejemplo View Ridge del capítulo 10, y el análisis puede ser burdamente paralelo al de la administración de la base de datos del capítulo 11.

➤ INSTALACIÓN DE ORACLE

Para aprovechar completamente este capítulo es conveniente tener instalado Oracle en su computadora. Conforme avancemos, repita las acciones que se describen. Puede tener dos versiones de la edición personal: Windows 98 o Windows 2000. Puede utilizar la versión que tenga. Será suficiente con el Oracle básico.

➤ CREACIÓN DE UNA BASE DE DATOS DE ORACLE

Existen tres formas de crear una base de datos de Oracle: mediante el asistente de configuración de la base de datos, a través de los procedimientos de creación de la base de datos proporcionada por Oracle, o por medio de la instrucción SQL CREATE DATABASE (CREAR UNA BASE DE DATOS). El asistente de configuración de la base de datos de Oracle es por mucho el más fácil, por lo tanto es el que debe usar.

Puede encontrar el asistente de configuración de la base de datos en uno de los directorios cuando se instaló Oracle. Dependiendo de su sistema operativo, lo encontrará haciendo clic en Start/Programs/Oracle-OraHome81/Database Administration, o algo similar. Probablemente sus directorios no se llamen exactamente igual; búsquelo a través de los directorios debajo de Start/Programs (Inicio/Programas) para encontrar el asistente de configuración de la base de datos.

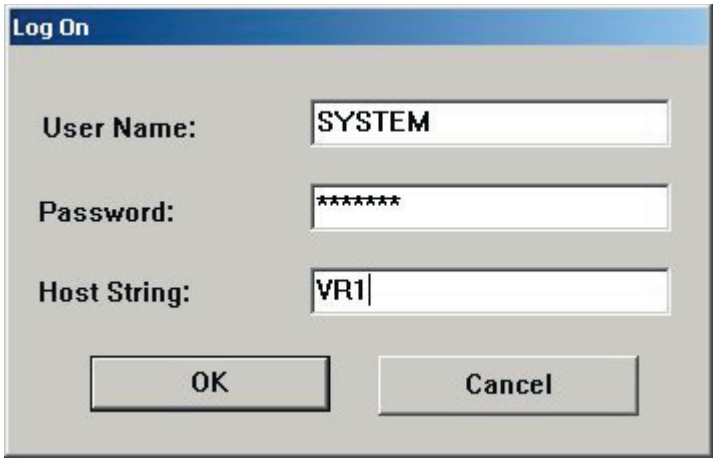
Cuando inicie el asistente de configuración verá la figura 12-1. Haga clic en *Create a database* y seleccione en la forma usual; elija copiar los archivos de la base de datos del CD; después seleccione un nombre para su base. En este capítulo se utiliza el nombre de la base de datos VR1. Haga clic en Finish (Terminar) y se creará su base de datos.

➤ FIGURA 12-1

Creación de una base de datos de Oracle

➤ FIGURA 12-2

Cuenta Oracle



Por acción predeterminada, Oracle crea tres cuentas en su nueva base de datos: *INTERNAL,* con contraseña *ORACLE; SYS,* con contraseña *CHANGE _ON_ INSTALL;* y *SYSTEM,* con contraseña *MANAGER.* (Los nombres y contraseñas de las cuentas son insensibles a mayúsculas y minúsculas.)

Oracle creará archivos predeterminados para sus datos y para logs de actividad de datos. El programador tiene gran control sobre el modo y el lugar donde se crean estos archivos, pero aquí se omitirá el análisis del tema. Usted podrá investigar más cuando haya aprendido los fundamentos de Oracle.

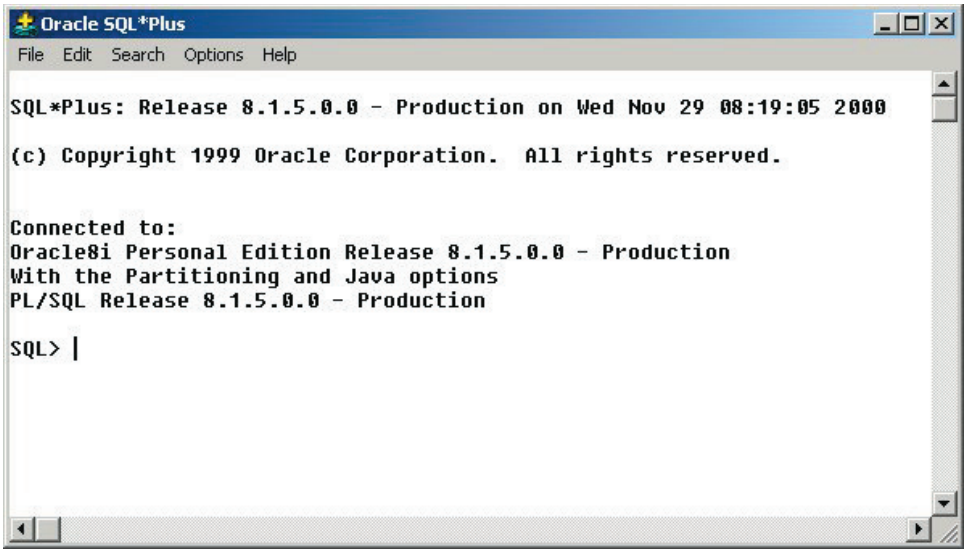
Uso de SQL Plus

Para utilizar SQL Plus encuentre el ícono correspondiente debajo de los menúes Start/Programs y haga clic. Ingrese a su base de datos mediante la cuenta SYSTEM e introduzca el nombre de su base de datos debajo del *Host String*, como se muestra en la figura 12-2. (Este procedimiento puede ser diferente si está empleando una versión de Oracle instalada por alguien más. Si es el caso, compruébelo con su administrador de base de datos.) Haga clic en OK y verá una ventana similar a la de la figura 12-3.

Entre sus muchas funciones, SQL Plus es un editor de textos. El trabajo con Oracle será más fácil si aprende un poco acerca de este editor. Primero, cuando escriba en SQL Plus lo que tecleó se colocará dentro de una memoria intermedia (buffer). Cuando pre-

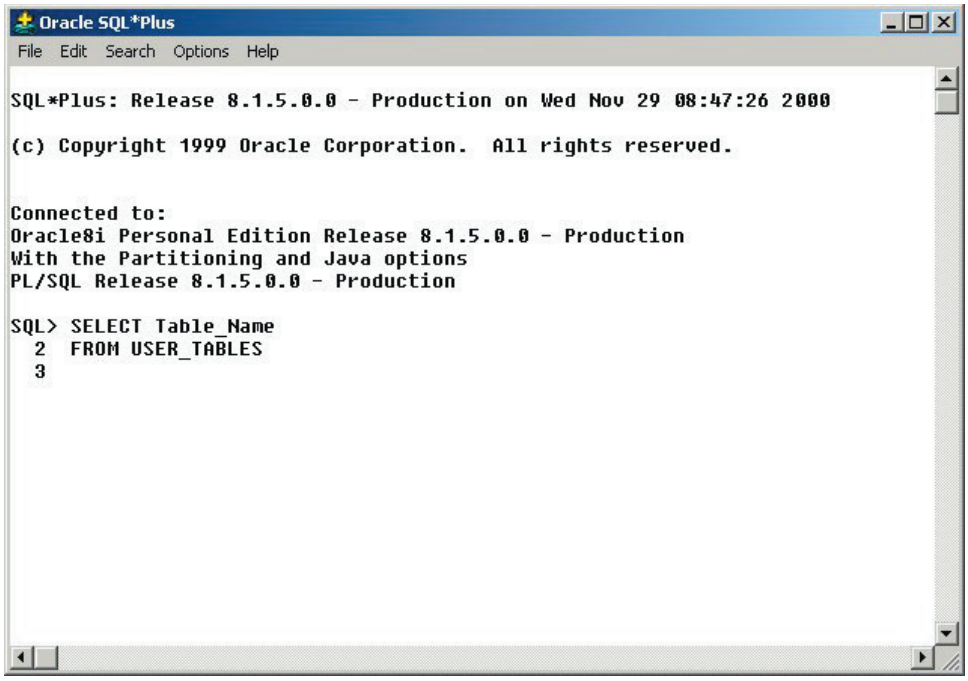
➤ FIGURA 12-3

Apuntador SQL Plus



➤ FIGURA 12-4

Memoria intermedia (buffer) de líneas múltiples de SQL Plus

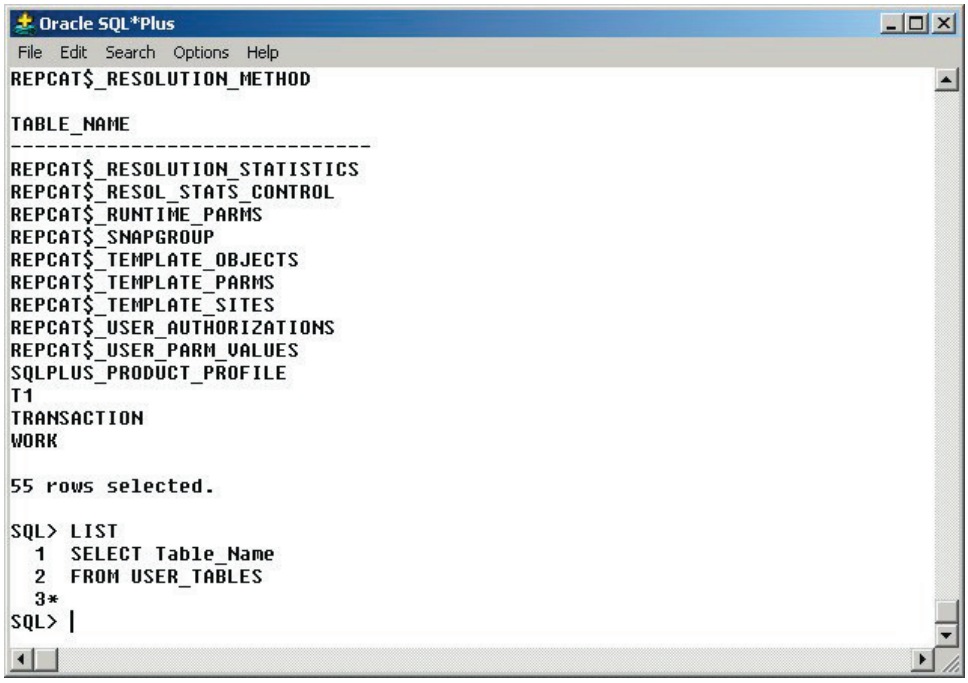


sione Enter, SQL Plus guardará lo que apenas escribió dentro de un renglón en la memoria intermedia (buffer), e irá a un nuevo renglón, pero no terminará ni ejecutará la instrucción.

MEMORIA INTERMEDIA DE SQL PLUS. Por ejemplo, en la figura 12-4 el usuario introdujo dos renglones en una instrucción SQL. Si es necesario, puede agregar más renglones. Cuando escriba un punto y coma y presione {Enter}, SQL Plus finalizará la instrucción y la ejecutará. Inténtelo, pero ignore los resultados, porque posteriormente los analizaremos.

➤ FIGURA 12-5

Utilización de LIST



Para ver el contenido de la memoria intermedia escriba LIST, como se muestra en la figura 12-5. El renglón que aparece con un asterisco, en este caso es el 3, es el renglón actual. Usted puede cambiar el renglón actual mediante la introducción de LIST seguido por un número de renglón, tal como LIST 1. En ese punto, el renglón 1 es el actual.

Para cambiar el contenido del renglón actual introduzca el cambio change */astring/bstring/*, donde astring es la cadena que puede cambiar y bstring es la que quiere reemplazar. En la figura 12-6 el usuario ha introducido

```
Change/Table_Name/*/
```

Esta expresión reemplazará la cadena 'Table_Name' con la cadena '*'.

Ahora, cuando el usuario escriba list la expresión en el renglón 1 de la memoria intermedia se ha cambiado de SELECT Table_Name a SELECT*. Escriba LIST para ver la instrucción completa. Introduzca la diagonal inclinada hacia la derecha (/), seguida de {Enter} y se ejecutará la instrucción en la memoria intermedia (buffer).

Antes de continuar debe saber que las instrucciones, nombres de las columnas, nombres de las tablas, nombres de las vistas, y todos los otros elementos de la base de datos de Oracle son insensibles a mayúsculas y minúsculas. Como se muestra en la figura 12-6, *LIST* es lo mismo que *list*. La única vez que las mayúsculas y las minúsculas importan es cuando están entre comillas de las cadenas de caracteres. Así,

```
SELECT * from ARTIST;
```

y

```
select * FROM artist;
```

son idénticas. Pero

```
SELECT * FROM ARTIST WHERE Name='Miró';
```

y

```
SELECT * FROM ARTIST WHERE Name='MIRÓ';
```

son diferentes. Las mayúsculas y minúsculas importan cuando están entre comillas y se utilizan para los valores de datos.

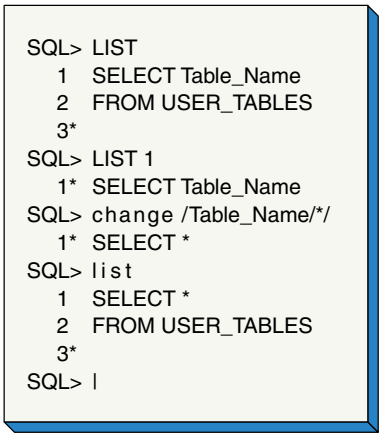
Asimismo, existe una diferencia entre el punto y coma y la diagonal inclinada hacia la derecha (/). El punto y coma termina una instrucción SQL; la diagonal inclinada hacia la derecha le ordena a Oracle ejecutar cualquier instrucción que esté en la memoria intermedia. Si sólo hay una instrucción y no existe ninguna ambigüedad sobre lo que se desea, Oracle tratará al punto y coma y a la diagonal como lo mismo. Por consiguiente, con

```
Select * from USER_TABLES;
```

el punto y coma termina la instrucción y hace que Oracle la ejecute. En este punto, escriba / y se ejecutará de nuevo dicha instrucción.

➤ FIGURA 12-6

Cambio de un renglón de la memoria intermedia (buffer)



USO DE UN EDITOR DE TEXTOS. Esta facilidad es útil para hacer cambios pequeños, pero no es viable para la edición de expresiones más grandes tales como los procedimientos almacenados (stored procedures). Para dicho propósito, puede instalar SQL Plus para conectarse a su editor de textos. Sin embargo, antes de realizarlo deberá crear un directorio para su código y señalar a SQL Plus hacia tal directorio.

Primero, salga de SQL Plus escribiendo Exit en el apuntador SQL>. Ahora, cree un directorio para su código de Oracle; c:\MyDirectory\OracleCode. Encuentre el icono de SQL Plus en su computadora, haga clic derecho en él para ver las propiedades, e introduzca el nombre de su nuevo directorio en el cuadro de texto Start In (Inicio). Haga clic en OK. Reinicie SQL Plus.

Haga clic en el elemento Edit (Edición) en el menú de la ventana del SQL Plus y después seleccione Editor/Define Editor. Ahí, puede introducir el nombre de su editor. El bloc de notas se ofrece como la opción predeterminada y eso será suficiente para varios propósitos, así que haga clic en OK.

Hasta este punto usted ha definido el bloc de notas como su editor predeterminado para SQL Plus y lo ha establecido para señalar su directorio. Cada vez que escriba *Edit*, SQL Plus invocará el bloc de notas (o su editor, si ha seleccionado uno diferente). Ahora puede crear, guardar y editar archivos de código en ese directorio. Por ejemplo, vuelva a introducir las instrucciones:

```
SELECT Table_Name
FROM USER_TABLES;
```

Después de que aparezcan los resultados, escriba *Edit*. SQL Plus traerá el bloc de notas con el contenido de la memoria intermedia (buffer). Utilice Save As (Guardar como) para dar a este archivo un nuevo nombre: EX1.txt. Cierre el bloc y regresará a SQL Plus. Para editar el archivo que acaba de crear escriba *Edit EX1.txt* e ingresará a su editor con dicho archivo. Cuando salga de su editor y regrese a SQL Plus, se almacenará EX1.txt en la memoria intermedia de SQL Plus. Para hacer que se ejecute el contenido de la memoria intermedia, introduzca la tecla diagonal inclinada hacia la derecha (/).

A propósito, la extensión predeterminada del archivo opcional para SQL Plus es *.sql*. Si nombra a un archivo *EX1.sql* entonces simplemente puede introducir *Edit EX1* y SQL Plus agregará la extensión por usted.

Ahora que tenemos este conocimiento sobre SQL Plus podemos investigar algunas de las características de Oracle. En la siguiente sección usaremos el ejemplo de la galería View Ridge que presentamos en el capítulo 10 y crearemos la llave sustituta del esquema de la base de datos de la figura 10-3(d).

Creación de tablas

La sintaxis básica para la instrucción CREATE TABLE SQL se ilustra en la figura 12-7. Consiste en la expresión CREATE TABLE, seguida del nombre de la tabla, de una lista de los nombres de columnas, tipos y propiedades entre paréntesis. La instrucción termina con un punto y coma, igual que todas las instrucciones de Oracle SQL.

La figura 12-7 muestra dos maneras diferentes de crear una llave primaria. La estructura de la tabla CLIENTE se crea sin ninguna llave primaria y después se define una mediante la siguiente instrucción: ALTER TABLE. El nombre de la restricción de la llave primaria que se emplea aquí es CustomerPK. Un segundo medio para la creación de una llave primaria es establecerla como una propiedad de una columna en la instrucción CREATE TABLE. Esto se hace para la tabla ARTIST en la figura 12-7. (En este capítulo, el texto que usted teclee en su editor de textos aparecerá como texto plano en el formato de la figura 12-7. El despliegue de vistas de pantalla de SQL Plus aparecerá con recuadro como las de la figura 12-6.)

Si la tabla tiene una llave primaria compuesta, entonces sólo se puede utilizar el método de creación de la llave primaria ALTER TABLE, debido a que la sintaxis de CREATE TABLE sólo permite que una columna tenga la propiedad de la LLAVE. Por lo tanto, la llave primaria para la tabla CUSTOMER_ARTIST_INT (con Oracle utilice guiones bajos en lugar de guiones), la cual es una llave compuesta, debe definirse mediante una instrucción ALTER, como se muestra.

➤ FIGURA 12-7

Creación de las tablas CUSTOMER, ARTIST y CUSTOMER_ARTIST_INT

```
CREATE TABLE CUSTOMER(
        CustomerID          int             NOT NULL,
        Name                varchar(25)     NOT NULL,
        Street              varchar(30)     NULL,
        City                varchar(35)     NULL,
        State               varchar(2)      NULL,
        Zip                 varchar(5)      NULL,
        Area_Code           varchar(3)      NULL,
        Phone_Number        varchar(8)      NULL);

ALTER TABLE CUSTOMER
 ADD CONSTRAINT CustomerPK PRIMARY KEY ( CustomerID );

CREATE INDEX CustomerNameIndex ON CUSTOMER(Name);

CREATE TABLE ARTIST(
        ArtistID            int             PRIMARY KEY,
        Name                varchar(25)     NOT NULL,
        Nationality         varchar(30)     NULL,
        Birthdate           date            NULL,
        DeceasedDate        date            NULL);

CREATE UNIQUE INDEX ArtistNameIndex ON ARTIST(Name);


CREATE TABLE CUSTOMER_ARTIST_INT(
        ArtistID        int             NOT NULL,
        CustomerID      int             NOT NULL);

ALTER TABLE CUSTOMER)_ARTIST_INT
 ADD CONSTRAINT CustomerArtistPK PRIMARY KEY  (  ArtistID, CustomerID );
```

Para crear estas tablas abra el bloc de notas y escríbalas en él. Guárdelas con el nombre de archivo: Create1.sql. Ahora, inicie SQL Plus e ingrese al

```
Start Create1
```

apuntador de SQL. (Si utilizó una extensión de archivo diferente a .sql deberá agregarla; por ejemplo, Start Create1.txt.)

SQL Plus abrirá el archivo Create1.sql y leerá SQL en la memoria intermedia (buffer). Sin embargo, no ejecutará estas instrucciones hasta que presione el botón de la diagonal inclinada hacia la derecha (/). Ahora hágalo y se crearán sus tablas.

Si hay un problema, escriba

```
Edit Create1
```

y su editor se abrirá con su archivo. Haga los cambios necesarios, guárdelos y salga de su editor. SQL Plus tendrá en su memoria intermedia las instrucciones cambiadas; para ejecutarlas, introduzca la diagonal.

Si desea que sus instrucciones se ejecuten automáticamente al final del archivo, coloque la diagonal después del punto y coma de su última instrucción. Cuando escriba Start *filename,* se ejecutará automáticamente la instrucción en su archivo.

Se puede obtener la estructura de una tabla mediante la instrucción DESCRIBE o DESC. Introduzca *DESC CUSTOMER* y se desplegará la estructura de la tabla. Sus tablas deberán aparecer como se muestra en la figura 12-8. Observe que el tipo de datos estándar de SQL varchar fue cambiado a varchar 2; Oracle prefiere ese nombre. Asimismo, las llaves sustitutas, que se definieron como del tipo estándar int, se cambiaron a un número de longitud 38. En la figura 12-9 se muestra una lista parcial de los tipos de datos de Oracle.

LLAVES SUSTITUTAS MEDIANTE EL USO DE SECUENCIAS. Una secuencia es un objeto que genera una serie secuencial de números únicos. A menudo, las secuencias se utilizan para proporcionar valores a las llaves sustitutas. Las siguientes instrucciones definen una secuencia llamada CustID (CLIENTEID) que comienza en 1000 y aumenta de uno en uno cada vez que se utiliza.

➤ FIGURA 12-8

Uso de la instrucción Describe (desc)

```
SQL> desc customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMERID                                NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(25)
 STREET                                             VARCHAR2(30)
 CITY                                               VARCHAR2(35)
 STATE                                              VARCHAR2(2)
 ZIP                                                VARCHAR2(5)
 AREA_CODE                                          VARCHAR2(3)
 PHONE_NUMBER                                       VARCHAR2(8)

SQL> desc artist
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ARTISTID                                  NOT NULL NUMBER(38)
 NAME                                      NOT NULL VARCHAR2(25)
 NATIONALITY                                        VARCHAR2(30)
 BIRTHDATE                                          DATE
 DECEASEDDATE                                       DATE

SQL> desc customer_artist_int;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ARTISTID                                  NOT NULL NUMBER(38)
 CUSTOMERID                                NOT NULL NUMBER(38)
```

```
Create Sequence CustID Increment by 1 start with 1000;
```

Hay dos métodos de secuencias que nos interesan. El método NextVal da el valor siguiente en una secuencia, y el método CurrVal proporciona el valor actual en una secuencia. Por consiguiente, CustID.NextVal asigna el valor siguiente de la secuencia de CustID. Se puede insertar un reglón dentro de CUSTOMER mediante el uso de una secuencia como sigue:

```
INSERT INTO CUSTOMER
    (CustomerID, Name, Area_Code, Phone_Number)
    VALUES
    (CustID.NextVal, 'Mary Jones', '350', '555–1234);
```

➤ FIGURA 12-9

Tipo de datos de Oracle comúnmente usados

Tipo de datos	Descripción
BLOB	Objeto binario grande. Con más de 4 gigabytes de longitud.
CHAR (n)	Campo de carácter de longitud fija con longitud n. Máximo 2000.
DATE	Campo de 7 bytes que contiene fecha y hora.
INT	Número entero de longitud 38.
NUMBER(n,d)	Campo numérico de longitud n, d lugares a la derecha del punto decimal.
VARCHAR(n) o VARCHAR2(n)	Campo de carácter de longitud variable superior a n caracteres de longitud. Máximo valor de n = 4,000.

Se creará un reglón de CUSTOMER con el siguiente valor en la secuencia, como el valor para CustomerID. Una vez que se haya ejecutado esta instrucción puede acceder al renglón que acaba de crear con el método CurrVal como sigue:

```
SELECT    *
FROM      CUSTOMER
WHERE     CustomerID = CustID.CurrVal
```

Aquí, CustID.CurrVal regresa al valor actual de la secuencia, el cual es el valor que apenas se utilizó.

Por desgracia, el uso de secuencias no garantiza valores válidos de la llave sustituta. El primer problema es que un programador puede usar una secuencia definida para cualquier propósito. Si se utiliza una secuencia para propósitos diferentes al de la llave sustituta faltarán algunos valores. Un segundo problema, más serio, es que no existe nada en el esquema que impida a alguien emitir una instrucción Insert que no utilice la secuencia. Por lo tanto, Oracle aceptará

```
INSERT INTO CUSTOMER
    (CustomerID, Name, Area_Code, Phone_Number)
    VALUES
    (350, 'Mary Jones', '350', '555–1234);
```

Cuando esto se realiza probablemente se presenten valores duplicados de una llave sustituta. En este caso, Oracle no permitirá la inserción del duplicado, puesto que CustomerID está definida como una llave primaria. Sin embargo, esto aún significa que el código puede necesitar estar escrito para mostrar este error. Por último, es posible que alguien accidentalmente haya empleado la secuencia equivocada cuando estuvo insertando en la tabla. A pesar de estos posibles problemas, las secuencias son la mejor manera de trabajar con las llaves sustitutas en Oracle.

Se utilizarán las siguientes secuencias dentro de la base de datos View Ridge. Y ahora se crearán mediante SQL Plus.

```
Crear Secuencia CustID aumentando en 1 iniciando con 1000;
Crear Secuencia ArtistID aumentando en 1 iniciando con 1;
Crear Secuencia WorkID aumentando en 1 iniciando con 500;
Crear Secuencia TransID aumentando en 1 iniciando con 100;
```

INTRODUCCIÓN DE DATOS. Ahora se pueden utilizar estas secuencias para agregar datos. La figura 12-10 muestra un archivo de instrucciones de inserción creadas en

➤ FIGURA 12-10

Inserción de datos en las Tablas ARTIST (ARTISTA) y CUSTOMER (CLIENTE)

```
INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
  (ArtistID.NextVal, 'Tobey', 'US');
INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
  (ArtistID.NextVal, 'Miro', 'Spanish');
INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
  (ArtistID.NextVal, 'Frings', 'US');
INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
  (ArtistID.NextVal, 'Foster', 'English');
INSERT INTO ARTIST (ArtistID, Name, Nationality) Values
  (ArtistID.NextVal, 'van Vronkin', 'US');

INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
  (CustID.NextVal, 'Jeffrey Janes', '206', '555-1234');
INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
  (CustID.NextVal, 'David Smith', '206', '555-4434');
INSERT INTO CUSTOMER (CustomerID, Name, Area_Code, Phone_Number) Values
  (CustID.NextVal, 'Tiffany Twilight', '360', '555-1040');
```

el bloc de notas. Escríbalas en su editor, coloque una diagonal al final del archivo y guárdelo con el nombre *ACIns.sql*. Introduzca

```
Start ACIns;
```

y se ejecutarán las instrucciones en ACIns. Sus datos deberán aparecer como se muestra en la figura 12-11.

INSTRUCCIONES DROP Y ALTER. Puede utilizar la instrucción drop para eliminar estructuras de la base de datos. Por ejemplo, las instrucciones

```
DROP TABLE MYTABLE;
DROP SEQUENCE MySequence;
```

eliminarán la tabla MYTABLE y la secuencia MySequence, respectivamente. Se perderán todos los datos de la tabla MYTABLE.

Puede eliminar una columna con la instrucción ALTER como sigue:

```
ALTER TABLE MYTABLE DROP COLUMN MyColumn;
```

Mostraremos otros usos para DROP y ALTER conforme avance el análisis.

Creación de relaciones

Con Oracle se crean relaciones entre las tablas al definir las restricciones de las llaves externas. Por ejemplo, las siguientes instrucciones SQL definirán las relaciones entre CUSTOMER y CUSTOMER_ARTIST_INT y entre ARTIST y CUSTOMER_ARTIST_INT:

```
ALTER TABLE CUSTOMER_ARTIST_INT ADD CONSTRAINT ArtistIntFK
FOREIGN KEY(ArtistID) REFERENCES ARTIST ON DELETE CASCADE;

ALTER TABLE CUSTOMER_ARTIST_INT ADD CONSTRAINT CustomerIntFK
FOREIGN KEY(CustomerID) REFERENCES CUSTOMER ON DELETE CASCADE;
```

Los nombres de las restricciones son ArtistInFK y CustomerIntFK. Estos nombres no tienen ningún significado particular para Oracle; el programador puede elegirlos. Observe que sólo se especifica la columna de la llave externa de la tabla hija. Oracle supone que la llave externa se relacionará con la llave primaria de la tabla padre, y de esta manera no es necesario nombrar la columna de la llave primaria. Asimismo, la frase ON DELETE CASCADE especifica que los renglones hijos deben eliminarse cuando se suprime un reglón padre. El término *cascade* se emplea debido a las eliminaciones en cascada de la tabla padre a la tabla hija.

➤ FIGURA 12-11

Tablas ARTIST y CUSTOMER después de la inserción

```
SQL> SELECT ArtistID, Name, Nationality FROM ARTIST;

 ARTISTID  NAME          NATIONALITY
---------- ------------- ---------------
        2  Miro          Spanish
        3  Frings        US
        4  Foster        English
        5  van Vronkin   US
        1  Tobey         US

SQL> SELECT CustomerID, Name, Area_Code, Phone_Number FROM CUSTOMER;

CUSTOMERID NAME          ARE PHONE_NU
---------- ------------- --- --------
      1001 David Smith   206 555-4434
      1002 Tiffany Twilight 360 555-1040
      1000 Jeffrey Janes 206 555-1234
```

Introduzca estas instrucciones dentro de SQL Plus y a continuación los diversos renglones de intersección de las tablas. Si después elimina un cliente o un artista, se eliminará también cualquier intersección de los renglones de las tablas para dicho cliente o artista.

La figura 12-12 muestra la creación de instrucciones necesarias para crear las tablas WORK (TRABAJO) y TRANSACTION (TRANSACCIÓN). Observe que no existe ninguna cláusula ON DELETE CASCADE en las especificaciones de las restricciones de la llave externa. De este modo, la restricción ArtistFK prohibirá la eliminación de los renglones ARTISTA que tengan filas de la tabla hija TRABAJO. WorkFK y CustomerFK funcionan de manera similar.

También puede emplear la instrucción ALTER (ALTERAR) para eliminar una restricción. La instrucción:

```
ALTER TABLE MYTABLE DROP CONTRAINT MyConstraint;
```

eliminará la restricción MyConstraint de la tabla MYTABLE.

En la figura 12-12, la notación numérica (7,2) significa un número con siete dígitos, dos de los cuales están a la derecha del punto decimal. Por otra parte, Número(7) significa que se está definiendo un número entero de siete dígitos.

Creación de índices

Los índices se crean para imponer unicidad en las columnas, con el fin de facilitar la clasificación y permitir la rápida recuperación de los valores de las columnas, las cuales se utilizan frecuentemente con condiciones iguales en donde las cláusulas WHERE son buenas candidatas para los índices. La cláusula igual puede ser una condición simple en una cláusula WHERE, o puede presentarse en un join. Ambas se muestran en las siguientes instrucciones:

```
SELECT *
FROM MYTABLE
WHERE Column1=100;
```

y

```
SELECT *
FROM MYTABLE1, MYTABLE2
WHERE MYTABLE1.Column1=MYTABLE2.Column2;
```

➤ FIGURA 12-12

Creación de las tablas WORK (TRABAJO) y TRANSACTION (TRANSACCIÓN)

```
CREATE TABLE      WORK (
        WorkID              int             PRIMARY KEY,
        Description         varchar(1000)   NULL,
        Title               varchar(25)     NOT NULL,
        Copy                varchar(8)      NOT NULL,
        ArtistID            int             NOT NULL);

ALTER TABLE WORK ADD CONSTRAINT ArtistFK
FOREIGN KEY (ArtistID) REFERENCES ARTIST;

CREATE TABLE      TRANSACTION (
        TransactionID       int             PRIMARY KEY,
        DateAcquired        date            NOT NULL,
        AcquisitionPrice    number(7, 2)    NULL,
        PurchaseDate        date            NULL,
        SalesPrice          number(7, 2)    NULL,
        CustomerID          int             NULL,
        WorkID              int             NOT NULL);

ALTER TABLE TRANSACTION ADD CONSTRAINT WorkFK
FOREIGN KEY (WorkID) REFERENCES WORK;

ALTER TABLE TRANSACTION ADD CONSTRAINT CustomerFK
FOREIGN KEY (CustomerID) REFERENCES CUSTOMER;
```

Si se ejecutan instrucciones como estas frecuentemente, entonces las columnas 1 y 2 son buenas candidatas para los índices.

La siguiente instrucción crea un índice en el nombre de la columna de la tabla CUSTOMER.

```
CREATE INDEX CustNameIdx ON CUSTOMER(Name);
```

El índice se llamará CustNameIdx. De nuevo, el nombre no tiene ninguna importancia para Oracle. Con el fin de crear un índice único agregue la palabra clave UNIQUE antes de la palabra clave INDEX. Por ejemplo, para garantizar que no se está agregando dos veces a la tabla TRABAJO, podemos crear un índice único en (Título, Copia, Artista ID) como sigue:

```
CREATE UNIQUE INDEX WorkUniqueIndex ON WORK(Title, Copy, ArtistID);
```

Cambio de la estructura de tabla

Después de que se ha creado una tabla se puede modificar su estructura mediante la instrucción ALTER TABLE (ALTERAR TABLA). Sin embargo, sea cuidadoso porque al hacerlo se pueden perder datos.

La adición o eliminación de una columna es directa, como se muestra a continuación:

```
ALTER TABLE MYTABLE ADD C1 NUMBER(4);
ALTER TABLE MYTABLE DROP COLUMN C1;
```

La primera instrucción agrega una nueva columna llamada C1 y le da un tipo de datos numérico con una longitud de cuatro caracteres. La segunda instrucción elimina la columna que se acaba de agregar. Advierta que se omite la palabra llave *column* cuando se agrega una columna nueva.

Cuando edite estas instrucciones recibirá como respuesta el mensaje breve "Table altered" ("Tabla alterada"). Para asegurarse de que se hicieron los cambios deseados, utilice la instrucción DESCRIBE para ver la estructura de la tabla.

RESTRICCIONES A LAS MODIFICACIONES DE LAS COLUMNAS DE LA TABLA.

Puede eliminar una columna en cualquier momento; sin embargo, todos los datos se perderán cuando lo haga. También puede agregar una columna cuando sea, siempre y cuando sea una columna NULL (NULA).

Para agregar una columna NO NULA (NOT NULL), primero agréguela a la tabla como NULL, después llene cada fila de la columna nueva con datos y cambie su estructura a NOT NULL mediante la cláusula de modificación. Por ejemplo, suponga que acaba de agregar la columna C1 a la tabla T1. Después de que haya llenado la C1 en cada fila de la T1 puede editar lo siguiente:

```
ALTER TABLE T1 MODIFY C1 NOT NULL;
```

Ahora se requerirán los valores para la columna C1.

Cuando modifique una columna puede aumentar el número de caracteres en las columnas de caracteres, o el número de dígitos en las columnas numéricas. También puede aumentar o disminuir el número de lugares decimales cuando lo desee. Si los valores de todos los renglones de una columna determinada son NULL (NULOS), puede disminuir la anchura de los datos de caracteres y numéricos, y cambiar el tipo de datos de la columna.

Restricciones de verificación (Check)

Las restricciones de verificación se utilizan para restringir los valores que las columnas puedan tener. Las restricciones de verificación deben escribirse para que evalúen la veracidad o la falsedad de los valores, y para que sólo puedan evaluarse mediante las constantes y los valores del renglón actual de la tabla en la cual están definidos. Las restricciones de verificación no pueden contener ni subconsultas ni secuencias, ni pueden referirse a la función SysDate.

Las restricciones de verificación pueden crearse junto con la tabla o se pueden agregar posteriormente. Un ejemplo de lo anterior es

```
CREATE TABLE MYTABLE(
    Name VARCHAR(50)        NOT NULL,
    State CHAR(2)           NULL CHECK (State IN ('CA', 'CO', 'NY')));
```

Con esta definición los únicos valores permitidos para State (Estado) son 'CA', 'CO' y 'NY'.

Un ejemplo de una restricción de VERIFICACIÓN agregada con ALTER (ALTERAR) es

```
ALTER TABLE TRANSACTION ADD CONSTRAINT DateCheck CHECK
es decir DateAcquired <= PurchaseDate); (Fecha de adquisición <= Fecha de compra)
```

Para este trabajo, tanto DateAcquired como PurchaseDate deben residir en la tabla TRANSACCIÓN.

Las restricciones también pueden eliminarse con la instrucción ALTER TABLE (ALTERAR TABLA):

```
ALTER TABLE TRANSACTION DROP CONSTRAINT DateCheck;
```

Uso de ALTER TABLE y de las restricciones de verificación

Para modificar la definición de la tabla ARTIST podemos utilizar lo que recién analizamos. En la figura 12-7 se establecieron el tipo de datos de las columnas BirthDate (FechadeNacimiento) y DeceasedDate (FechadeFallecimiento) para Date (Fecha). Suponga que a los usuarios de esta base de datos no les importa almacenar una fecha en particular para estas columnas, sino que sólo quieren registrar el año de nacimiento y el año de muerte del artista. Suponga también que ninguno de los artistas de la galería nació o murió antes de 1400 o después de 2100.

Puesto que aún no se ha almacenado ningún dato en estas columnas, ambas son NULL y podemos cambiar sus tipos de datos sin eliminarlas. Para realizarlo, se envían las siguientes instrucciones mediante SQL Plus:

```
ALTER TABLE ARTIST MODIFY FechadeNacimiento(4);
ALTER TABLE ARTIST MODIFY FechadeFallecimiento(4);
```

Ahora, las dos instrucciones siguientes establecerán los límites en los valores de las fechas de nacimiento y fallecimiento:

```
ALTER TABLE ARTIST ADD CONSTRAINT BDLimit CHECK (FechadeNacimiento
    BETWEEN 1400 AND 2100);
ALTER TABLE ARTIST ADD CONSTRAINT DDLimit CHECK (FechadeFallecimiento
    BETWEEN 1400 AND 2100);
```

Considere las siguientes instrucciones de actualización:

```
UPDATE ARTIST SET BirthDate = 1870 WHERE Name = 'Miro';
UPDATE ARTIST SET BirthDate = 1270 WHERE Name = 'Tobey';
```

La primera actualización se procesará normalmente, pero la segunda provocará una violación de la verificación y no se procesará. Pruébelo para ver los resultados.

Vistas

Como se mencionó en el capítulo 10, las vistas en SQL son diferentes a lo que se ha llamado vistas de base de datos. Una vista en SQL es el resultado de una expresión de SQL que usa selección, proyección y join. Por ejemplo, una vista puede hacer el join de los renglones ARTIST, WORK y TRANS, seleccionar ciertas columnas de tales renglones y aplicar las condiciones WHERE a la relación resultante.

➤ FIGURA 12-13

Creación de una vista

```
SQL> CREATE VIEW V1 AS
  2  SELECT Name
  3  FROM CUSTOMER;

View created.

SQL> SELECT * FROM V1;

NAME
-----------------
David Smith
Tiffany Twilight
Jeffrey Janes
```

La limitación más importante de las vistas SQL es que no pueden contener más que una trayectoria de valores múltiples. Por ejemplo, no es posible representar una vista que tenga los datos CUSTOMER, TRANS y CUSTOMER_ARTIST_INT con una vista SQL (como se hizo en la figura 10-4). Analizaremos nuevamente este tema cuando abordemos XML en el capítulo 14.

A pesar de esta limitación, las vistas SQL son útiles. Uno de sus usos consiste en proporcionar mayor seguridad. Para hacerlo, defina una vista que tenga todas las columnas de una tabla, pero confiera a los usuarios de la nueva vista permisos de seguridad diferentes a los de los usuarios de la tabla, o de los usuarios de otras vistas de la tabla. Otro propósito es definir una vista de una tabla que omita ciertas columnas o ciertos renglones. Por ejemplo, se puede definir una vista sobre una tabla EMPLOYEE (EMPLEADO) que omita la columna Salario (Salary), así como todos los datos para los ejecutivos. Sin embargo, el tercer propósito de una vista es juntar las tablas para que los usuarios de una vista no necesiten ejecutar el join sino simplemente consultar la vista. Asimismo, este uso de una vista oculta a los usuarios de la estructura de la tabla, que es lo que algunas veces se desea.

CREACIÓN DE VISTAS. La figura 12-13 muestra la creación y el uso de una vista simple. La vista V1 se define sólo con el nombre de la columna de la tabla CUSTOMER (CLIENTE). Cuando el usuario selecciona todas las columnas de la vista (con *), sólo se presenta el nombre de la columna. No se muestra ninguna de las otras columnas en CUSTOMER, ya que no están presentes en la vista.

En la figura 12-14(a) se muestra una vista más complicada, la cual junta tres tablas unidas y después aplica una condición sobre PreciodeAdquisición y ClienteID. Algunas veces a las vistas como éstas se les llama **vistas de join** debido a que se basan en joins. Advierta que la instrucción seleccionada en la figura 12-14(b) no contiene ni el join ni la cláusula where. Éstas se aplicaron tras bambalinas cuando se construía la vista.

En general, es posible actualizar los datos de una vista que está basada en una tabla. Si no quiere hacerlo, puede crear una vista sólo de lectura al agregar la expresión *with read only* al final de la definición de la vista. De esta manera,

➤ FIGURA 12-14

Ejemplo de un join de vistas: (a) creación de una vista

```
CREATE VIEW ExpensiveArt AS

SELECT    Name, Copy, Title
FROM      ARTIST, WORK, TRANSACTION
WHERE     ARTIST.ArtistID        =     WORK.ArtistID AND
          WORK.WorkID            =     TRANSACTION.WorkID AND
          AcquisitionPrice       >      10000 AND
          CustomerID IS NULL;
```

➤ FIGURA 12-14
(Continuación)

(b) selección de la vista

```
SQL> select * from expensiveart;

NAME                 COPY       TITLE
-------------------- ---------- --------------------
Tobey                4/40       Mystic Fabric
Tobey                4/40       Mystic Fabric
Miro                 79/122     Mi Vida
```

```
CREATE VIEW V1 AS
    SELECT *
    FROM ARTIST
    WITH READ ONLY;
```

creará una vista sólo de lectura.

La instrucción de SQL UPDATE (ACTUALIZAR) se puede utilizar algunas veces para modificar los valores mediante una vista, pero sólo bajo circunstancias especiales, y también si el cambio únicamente afecta a una tabla.

En general, para realizar actualizaciones mediante una vista, no se puede utilizar la instrucción UPDATE (ACTUALIZAR). En vez de eso, debe escribir un tipo especial de disparador denominado disparador INSTEAD OF (EN LUGAR DE). En la siguiente sección se considerarán dichos disparadores (triggers).

➤ LÓGICA DE LA APLICACIÓN

Existen muchas maneras de procesar la base de datos Oracle desde una aplicación. Una de ellas es crear un código de aplicación en C++, C#, Java, Visual Basic, o en algún otro lenguaje de programación y llamar a los programas de Oracle. Como se mencionó en el capítulo 8, esto se puede realizar a través de las bibliotecas originales de Oracle o mediante los estándares industriales de ODBC o de JDBC. Estos últimos se ilustrarán en los capítulos 15 y 16.

Otra manera de procesar una base de datos de Oracle es escribir los procedimientos en PL/SQL, los cuales pueden guardarse como archivos y ejecutarse mediante la instrucción START en SQL Plus; pueden guardarse en la base de datos como procedimientos almacenados o también como disparadores que se activarán cuando se presenten determinados sucesos de la base de datos. Cada uno se considerará en su momento.

PROCESAMIENTO DE ARCHIVOS DE PL/SQL

Si el usuario de la base de datos tiene acceso a SQL Plus, puede guardar la instrucción PL/SQL en archivos y procesarlos directamente mediante la instrucción START. Un archivo que tuviese la instrucción

```
SELECT    *
FROM      ExpensiveArt;
/
```

podría almacenarse bajo el nombre ToSell.sql. Entonces, el usuario podría abrir SQL Plus y escribir la instrucción

```
Start ToSell;
```

Entonces se desplegarían los datos de la vista ExpensiveArt (arte costoso).

Sin embargo, en realidad la mayoría de los usuarios de negocios no tienen acceso a SQL Plus y si lo tuvieran probablemente no les agradaría. Este estilo de procesamien-

to lo utilizan los administradores y programadores de bases de datos para automatizar las tareas rutinarias de la administración de la base de datos.

Procedimientos almacenados (Stored procedures)

Un procedimiento almacenado es un programa PL/SQL o Java que está guardado dentro de la base de datos. Los procedimientos almacenados son programas; pueden tener parámetros, invocar otros procedimientos y funciones, pueden devolver valores y originar errores. Casi nunca se pueden invocar los procedimientos almacenados. En los capítulos 15 y 16 hay ejemplos de la invocación de procedimientos almacenados mediante la tecnología de Internet. Aquí consideraremos dos ejemplos.

PROCEDIMIENTO ALMACENADO Customer_Insert. Suponga que la galería View Ridge quiere agregar un nuevo cliente a su base de datos y registrar los intereses de éste en los artistas. En particular, la galería desea registrar el nombre y los datos telefónicos del cliente y después ponerlo en contacto con todos los artistas de una nacionalidad específica.

La figura 12-15 muestra un procedimiento almacenado que llevará a cabo esta tarea. El procedimiento, llamado Customer_Insert, recibe cuatro parámetros: nombrenuevo, códigopostalnuevo, númerotelefóniconuevo y nacionalidaddelartista. La palabra clave *IN* significa que éstos son parámetros de entrada. La palabra clave *OUT* representa un parámetro de salida, y la palabra clave IN OUT denota un parámetro utilizado tanto de entrada como de salida. Observe que este tipo de datos es acorde al parámetro, pero no a su longitud; Oracle la determinará a partir del contexto.

Las variables se declaran después de la palabra clave *AS*. En la instrucción mostrada SELECT se define un **cursor variable** llamado *artistcursor*. Este cursor se utilizará para procesar todos los renglones para un artista de la nacionalidad que se ingresó originalmente.

Se verifica la primera sección del procedimiento para determinar si ya existen los datos del cliente. Si es así, sólo se imprime un mensaje de salida mediante el paquete de Oracle, DBMS_OUTPUT.

Antes de continuar con este procedimiento, advierta que este mensaje sólo se puede ver si se invoca el procedimiento desde SQL Plus. Si se invoca de una manera diferente, por ejemplo, sobre Internet mediante un explorador, entonces el mensaje no aparecerá. El programador necesitará utilizar un parámetro de salida u originar una excepción. Sin embargo, estos temas están fuera del alcance de este análisis. Aunque no se muestra, la sintaxis para la impresión de una cadena y de un valor variable es: DBMS_OUTPUT.PUT_LINE ('string'//variable).

Adicionalmente, para ver tales mensajes, ejecute lo siguiente:

```
Set serveroutput on;
```

Si no está recibiendo la salida de sus procedimientos cuando utiliza SQL Plus, probablemente se deba a que no ha ejecutado esta instrucción.

El resto del procedimiento en la figura 12-15 inserta los datos del cliente nuevo y los relaciona mediante todos los artistas de la nacionalidad específica. Observe el uso de la construcción especial PL/SQL *FOR artist IN artistcursor*. Esta construcción realiza diversas tareas. Abre el cursor y va al primer renglón. Después se repite en todos los renglones en el cursor y cuando ya no hay más, transfiere el control a la siguiente instrucción después de FOR. Asimismo, note que el valor de ArtistID del renglón actual del cursor puede ingresarse con la sintaxis artist.ArtistID, donde artist es el nombre de la variable en la instrucción FOR y *no* el nombre del cursor.

Una vez que haya escrito este procedimiento, primero deberá compilarlo y almacenarlo en la base de datos. Codifique el procedimiento mediante su editor y guárdelo bajo un nombre, por ejemplo, SP_CI.sql. Si incluye una diagonal como su última línea, entonces el procedimiento se compilará y almacenará cuando ingrese la instrucción:

```
Start SP_CI
```

➤ FIGURA 12-15

El procedimiento Customer_Insert

```
CREATE OR REPLACE PROCEDURE Customer_Insert
        (
        newname                 IN      char,
        newareacode             IN      char,
        newphone                IN      char,
        artistnationality       IN      char
        )

AS

        rowcount integer(4);

        CURSOR          artistcursor IS
                        SELECT ArtistID
                        FROM ARTIST
                        WHERE Nationality=artistnationality;

BEGIN

        SELECT          Count (*) INTO rowcount
        FROM            CUSTOMER
        WHERE           Name=newname AND Area_Code=newareacode AND
                        Phone_Number = newphone;

        IF rowcount > 0 THEN
            BEGIN
                DBMS_OUTPUT.PUT_LINE ('Customer Already Exists -- No Action Taken');
                RETURN;
            END;
        END IF;

        INSERT INTO CUSTOMER
                (CustomerID, Name, Area_Code, Phone_Number)
                VALUES
                (CustID.NextVal, newname, newareacode, newphone);

        FOR artist IN artistcursor
                LOOP
                        INSERT INTO CUSTOMER_ARTIST_INT
                        (CustomerID, ArtistID)
                        VALUES
                        (CustID.CurrVal, artist.ArtistID);
                END LOOP;

        DBMS_OUTPUT.PUT_LINE ('New Customer Successfully Added') ;
END;
/
```

Si ha cometido una equivocación, puede haber errores de compilación. Lamentablemente, SQL Plus no se los mostrará automáticamente, sino que le dará el mensaje "Warning: Procedure created with compilation errors" ("Advertencia: Procedimiento creado con errores de compilación"). Para ver los errores, introduzca

```
Show errors;
```

Si no hay ningún error de sintaxis recibirá el mensaje "Procedure created" ("Procedimiento creado"). Ahora puede invocar el procedimiento con la instrucción Execute o Exec, como se muestra a continuación:

```
Exec Customer_Insert ('Selma Warning', '206', '555-0099', 'US');
```

Si tiene errores en tiempo de ejecución, los números de líneas reportados diferirán de los que ve en su editor de texto. Puede ajustarlos para que se adapten al suyo, sin em-

bargo, el proceso es demasiado complicado para describirlo aquí. Para los procedimiento simples que se harán, sólo trabaje sobre el asunto. Sin embargo, no suponga que los números de línea coinciden.

PROCEDIMIENTO ALMACENADO NewCustomerWithTransaction (Cliente NuevoconTransacción). En el capítulo 10 se describieron las instrucciones de SQL necesarias para la creación, lectura, modificación y eliminación de la vista de la base de datos mostrada en la figura 10-4. En la figura 12-16 se muestra un procedimiento de almacenado de Oracle para implementar la creación de la vista lógica que se describe en el capítulo 10.

La lógica de este procedimiento, llamado NewCustomerWithTransaction, se muestra a continuación. Primero cree los renglones del cliente nuevo y después busque los renglones de TRANSACTION para el trabajo comprado que tengan valores nulos para Customer ID. Dicha búsqueda involucra el join de las tablas ARTIST, WORK y TRANSACTION, puesto que el nombre del artista se almacena en ARTIST y el título y la copia de la obra o trabajo se almacenan en WORK. Si sólo se encuentra un reglón, actualice ahí Customer ID (ClienteID), SalesPrice (PreciodeVenta), y PurchaseDate (FechadeCompra). Luego, inserte un reglón en la tabla de intersección para registrar el interés del cliente en el artista. De lo contrario, no haga cambios en la base de datos.

Como se muestra, el procedimiento NewCustomerWithTransaction acepta los parámetros que tengan los datos del cliente y la fecha de compra. Posteriormente, se declaran diversas variables y un cursor. Éste define el join de las tablas ARTIST, WORK y TRANSACTION; selecciona a TransactionID (IDdelatransacción) y a ARTIST.artistID de los renglones que se asocian con el ingreso del artista y los datos de trabajo, y esto tiene un valor nulo para CustomerID.

El procedimiento verifica si los datos de entrada del cliente existen realmente en la base de datos. Si no, inserta los datos del cliente nuevo. En PL/SQL no existe ninguna instrucción BEGIN TRANSACTION[1]; la primera acción de la base de datos inicia automáticamente una transacción. Aquí, la inserción de los datos del cliente comenzará una nueva transacción.

También observe que los comentarios están entre /* y */. Éstos pueden extenderse sobre líneas múltiples, y si comienza un comentario con /* y olvida terminarlo con */, entonces su programa entero será tomado como un comentario.

Después de que se insertan los datos del cliente, se procesa el TransCursor. La variable rowcount se utiliza para contar los renglones, el valor de TransactionID se almacena en *tid*, y el de ArtistID, en *aid*. Observe que el operador de la asignación en Oracle es := . De esta manera, tid := trans. TransactionID se utiliza para asignar el valor de trans.TransactionID a tid variable.

Según esta lógica, si sólo se encuentra un reglón calificado, entonces *tid* y *aid* tendrán los valores que se necesitan para continuar. Si no se encuentra ninguno, o se encuentra más de un reglón calificado, entonces se abortará la transacción, pero no se utilizarán ni *tid* ni *aid*.

Podríamos utilizar Count(*) para contar los renglones calificados, y entonces, si Count(*) = 1, se ejecuta otra instrucción SQL para obtener los valores requeridos de *tid* y *aid*. La lógica en la figura 12-16 guarda esta segunda instrucción SQL.

Si RowCount es mayor que 1 o igual a cero, entonces se genera un mensaje de error y se invierte la transacción para remover la inserción anterior a CLIENTE. Si RowCount es igual a 1, entonces se actualiza el renglón apropiado de TRANSACTION. Observe el uso de la función SysDate para almacenar el dato actual. Finalmente, se inserta un reglón de intersección para este cliente y el artista del trabajo comprado (*aid)*.

Este procedimiento almacenado que se guardó puede invocarse con una instrucción como la siguiente:

```
Exec NewCustomerWithTransaction ('Susan Wu', '206', '555-1000', 'Miro', 'Mi Vida',
'79/122', '65000');
```

[1] ¡Tenga cuidado aquí! En esta sección se está utilizando la palabra *TRANSACTION* (TRANSACCIÓN) de dos maneras: como el nombre de una de las tablas de View Ridge y como el nombre de un grupo de instrucciones que se ejecutarán automáticamente.

➤ FIGURA 12-16

El procedimiento de la transacción NewCustomerWith

```
CREATE OR REPLACE PROCEDURE NewCustomerWithTransaction
        (
        newname IN char,
        newareacode IN char,
        newphone IN char,
        artistname IN char,
        worktitle IN char,
        workcopy IN char,
        price IN number
        )

AS

        rowcount integer (2) ;
        tid     int;
        aid     int;

            CURSOR      transcursor IS
              SELECT    TransactionID, ARTIST.ArtistID
              FROM      ARTIST, WORK, TRANSACTION
              WHERE     Name=artistname AND Title=worktitle AND Copy=workcopy AND
                        TRANSACTION.CustomerID IS NULL AND
                        ARTIST.ArtistID = WORK.ArtistID AND
                        WORK.WorkID = TRANSACTION.WorkID;

BEGIN

        /* Does Customer Already exist? */

        SELECT    Count (*) INTO rowcount
        FROM      CUSTOMER
        WHERE     Name=newname AND Area_Code=newareacode AND Phone_Number = newphone;

        IF rowcount > 0 THEN
            BEGIN
                DBMS_OUTPUT.PUT_LINE ('Customer Already Exists -- No Action Taken');
                RETURN;
            END;
        END IF;

        /* Customer not exist, add new customer data */
        INSERT INTO CUSTOMER
                (CustomerID, Name, Area_Code, Phone_Number)
                VALUES
                (CustID.NextVal, newname, newareacode, newphone);

        /* Look for one and only one available TRANSACTION row. */
        rowcount := 0;
        FOR trans In transcursor
                LOOP
                        tid := trans.TransactionID;
                        aid := trans.ArtistID;
                        rowcount := rowcount + 1;
                END LOOP;

        IF rowcount > 1 Then
            BEGIN
                /* Too many available rows -- undo with message and return */
                ROLLBACK;
                DBMS_OUTPUT.PUT_LINE ('Invalid Artist/Work/Transaction data -- No Action Taken');
                RETURN;
            END;
        END IF;
```

(*continúa*)

➤ FIGURA 12-16

(Continuación)

```
        IF rowcount = 0 Then
            BEGIN
                /* No available row exists -- undo with message and return */
                ROLLBACK;
                DBMS_OUTPUT.PUT_LINE ('No available transaction row -- No Action Taken');
                RETURN;
            END;
        END IF;

        /* Exactly one exists -- use it with tid obtained from transcursor above */
        DBMS_OUTPUT.PUT_LINE (tid);
        UPDATE TRANSACTION
                SET         CustomerID = CustID.Currval, Salesprice = price, PurchaseDate = SysDate
                WHERE       TransactionID = tid;
        DBMS_OUTPUT.PUT_LINE ('Customer created and transaction data updated') ;

        /* Now create interest in this artist for this customer */
        /* Use aid from transcursor above and CurrVal of sequence*/

        INSERT INTO CUSTOMER_ARTIST_INT (ArtistID, CustomerID)
                        VALUES (aid, CustID.CurrVal);
END;
/
```

Las siguientes instrucciones SQL desplegarán los datos después de la actualización:

```
SELECT    CUSTOMER.Name, Copy, Title, ARTIST.Name
FROM      CUSTOMER, TRANSACTION, WORK, ARTIST
WHERE     CUSTOMER.CustomerID = TRANSACTION.CustomerID AND
          TRANSACTION.WorkID = WORK.WorkID Y
          WORK.ArtistID = ARTIST.ArtistID;
```

y

```
SELECT    CUSTOMER.Name, ARTIST.Name
FROM      CUSTOMER, CUSTOMER_ARTIST_INT, ARTIST
WHERE     CUSTOMER.CustomerID = CUSTOMER_ARTIST_INT.CustomerID
          AND
          ARTIST.ArtistID = CUSTOMER_ARTIST_INT.ArtistID;
```

Observe que se requieren dos instrucciones SQL debido a que esta vista es una base de datos que tiene dos trayectorias de valores múltiples. Esto no puede representarse con una instrucción SQL (o vista SQL).

Los resultados de estas preguntas se muestran en la figura 12-17. Los datos de Susan Wu están incluidos correctamente en los datos de TRANSACTION y CUSTOMER _ARTIST_INT.

Disparadores (Triggers)

Los disparadores de Oracle son procedimientos PL/SQL o Java que se invocan cuando se presenta una actividad específica de la base de datos. Oracle maneja diversos tipos de disparadores; algunos se activan en las órdenes SQL que crean estructuras nuevas de la base de datos como tablas, algunos se activan sólo una vez en el nivel de la tabla cuando se hace un cambio en los renglones de una tabla, y otras sólo una vez para cada renglón que se cambia en una tabla. A estos últimos se les llama **disparadores de renglones** y los abordaremos en esta sección. Oracle maneja los disparadores de renglones BEFORE (ANTES), AFTER (DESPUÉS), y INSTEAD OF (EN VEZ DE). Considere un ejemplo de cada uno.

➤ FIGURA 12-17

Datos después de la inserción de Customer

```
SQL> SELECT CUSTOMER.Name, Copy, Title, ARTIST.Name
  2  FROM CUSTOMER, TRANSACTION, WORK, ARTIST
  3  WHERE CUSTOMER.CustomerID = TRANSACTION.CustomerID AND
  4        TRANSACTION.WorkID = WORK.WorkID AND
  5        WORK.ArtistID = ARTIST.ArtistID;

NAME                 COPY     TITLE          NAME
-------------------- -------- -------------- ----------
Tiffany Twilight     4/40     Mystic Fabric  Tobey
David Smith          4/40     Mystic Fabric  Tobey
Jeffrey Janes        5/40     Mystic Fabric  Tobey
Tiffany Twilight     79/122   Mi Vida        Miro
Susan Wu             79/122   Mi Vida        Miro

SQL> SELECT CUSTOMER.Name, ARTIST.Name
  2  FROM CUSTOMER, CUSTOMER_ARTIST_INT, ARTIST
  3  WHERE CUSTOMER.CustomerID = CUSTOMER_ARTIST_INT.CustomerID
  4        AND
  5        ARTIST.ArtistID = CUSTOMER_ARTIST_INT.ArtistID;

NAME                 NAME
-------------------- ----------
Susan Wu             Miro
Fred Smathers        Frings
Selma Warning        Frings
Fred Smathers        van Vronkin
Selma Warning        van Vronkin
Fred Smathers        Tobey
Selma Warning        Tobey

7 rows selected.
```

EJEMPLO DEL DISPARADOR BEFORE (ANTES). La figura 12-18 muestra un DISPARADOR BEFORE, el cual se utiliza para establecer un valor en un reglón anterior a la actualización. Para utilizar este disparador primero se crea una columna en TRANSACTION llamada AskingPrice (PrecioConsultado). Usted puede crearla con la siguiente instrucción:

```
ALTER TABLE TRANSACTION ADD AskingPrice Number(7, 2);
```

En la figura 12-18 el disparador New_Price (Precio_nuevo) se define con la frase "Before Insert or Update of AcquisitionPrice ON TRANSACTION" ("Antes de la inserción o actualización del precio de adquisición EN TRANSACCIÓN"). Dicha frase, aunque es correcta, contiene una ambigüedad; es decir, el disparador se activará "ANTES de (cualquier) inserción a TRANSACTION o ANTES de una actualización EN TRANSACTION".

➤ FIGURA 12-18

DISPARADOR BEFORE New_Price

```
CREATE OR REPLACE TRIGGER New_Price
        BEFORE INSERT OR UPDATE of AcquisitionPrice ON TRANSACTION
        FOR EACH ROW
                /* set AskingPrice before insert or update */
        BEGIN
                :new.AskingPrice := :new.AcquisitionPrice * 2;
        END;
```

Así, cualquier inserción activará el disparador y una actualización de AcquisitionPrice también lo activará.

Las actualizaciones de otras columnas no activarán el disparador. La frase "FOR EACH ROW " ("PARA CADA RENGLÓN") se identifica como un disparador de renglones. La lógica es simple: calcula el valor nuevo de AskingPrice como dos veces el valor nuevo de AcquisitionPrice (Preciodeadquisición).

El prefijo *:new* sólo está disponible para los disparadores (triggers). Se refiere al valor nuevo para una columna de la instrucción insertar o actualizar. De este modo, :new.AcquisitionPrice se refiere al valor nuevo de AcquisitionPrice (como lo proporciona el usuario). Para las actualizaciones y eliminaciones, existe también un prefijo *:old* el cual se refiere al valor de columnas anteriores a la instrucción de actualizar o eliminar.

Para crear este disparador escríbalo mediante su editor y guárdelo en un archivo; por ejemplo, Trig1.sql. Después, escriba *start Trig1* en SQL Plus y el disparador se compilará. Si no existe ningún error de compilación el disparador se almacenará en la base de datos y estará activo. Actualice la columna de AcquisitionPrice y luego seleccione los renglones que ya actualizó. El disparador habrá establecido el AskingPrice (PrecioConsultado).

Si tiene errores de compilación los podrá ver si introduce *Show Error* (*Mostrar errores*), como se hizo anteriormente con los procedimientos almacenados.

EJEMPLO DEL DISPARADOR AFTER. La figura 12-19 muestra un ejemplo de un DISPARADOR AFTER (DESPUÉS). La lógica de este disparador es la siguiente: View Ridge define un trabajo para venta como cualquiera que tenga un reglón TRANSACTION con un valor nulo de CustomerID. El propósito de este disparador es asegurar que existe dicho renglón de TRANSACTION cada vez que se agregue un trabajo a la base de datos. Este disparador, que se activa al crear un reglón de WORK, verifica la tabla

➤ FIGURA 12-19

DISPARADOR AFTER (DESPUÉS) On_Work_Insert

```
CREATE OR REPLACE TRIGGER On_WORK_Insert
      AFTER INSERT ON WORK

FOR EACH ROW

DECLARE
        rowcount        integer(2) ;

BEGIN

        /* Count Available Rows */

        SELECT      Count (*) INTO rowcount
        FROM        TRANSACTION
        WHERE       CustomerID IS NULL AND WorkID=:new.WorkID;

        IF rowcount > 0 Then /* Row exists do nothing */

            DBMS_OUTPUT.PUT_LINE ('Suitable transaction row exists; nothing done.');
            RETURN;

        ELSE /* Need to add new row */

            INSERT INTO TRANSACTION (TransactionID, DateAcquired, WorkID)
            VALUES (TransID.NextVal, SysDate, :new.WorkID);

        END IF;

END;
/
```

➤ FIGURA 12-20

Vista de CustomerPurchases (ComprasdelCliente): (a) definición de la vista, y (b) datos de la vista

```
CREATE VIEW CustomerPurchases AS

SELECT    CUSTOMER.Name CustName, Copy, Title, ARTIST.Name ArtistName
FROM      CUSTOMER, TRANSACTION, WORK, ARTIST
WHERE     CUSTOMER.CustomerID = TRANSACTION.CustomerID AND
          TRANSACTION.WorkID = WORK.WorkID AND
          WORK.ArtistID = ARTIST.ArtistID;
```

(a)

```
SQL> SELECT * FROM CustomerPurchases;
```

CUSTNAME	COPY	TITLE	ARTISTNAME
Tiffany Twilight	4/40	Mystic Fabric	Tobey
David Smith	4/40	Mystic Fabric	Tobey
Jeffrey Janes	5/40	Mystic Fabric	Tobey
Tiffany Twilight	79/122	Mi Vida	Miro
Susan Wu	79/122	Mi Vida	Miro

(b)

TRANSACTION y no hace nada si encuentra un reglón conveniente de TRANSACTION. En caso contrario, crea un nuevo renglón en esa tabla.

La lógica es directa. Se cuentan los renglones convenientes y si es mayor que cero, no se hace nada. Si la cuenta es cero, se crea un nuevo renglón de TRANSACTION con los datos apropiados. Observe cómo se utiliza el prefijo *:new* en la parte VALUES (VALORES) de la instrucción INSERT (INSERTAR).

EJEMPLO DEL DISPARADOR INSTEAD OF (EN VEZ DE). Los DISPARADORES INSTEAD OF se utilizan para actualizar las vistas. Considere la vista de la figura 12-20(a). Junta cuatro tablas y, como una vista de join, no se puede actualizar a menos que se aplique el DISPARADOR INSTEAD OF.

Observe la frase SELECT en esta definición de la vista. Especifica sinónimos para CUSTOMER.Name (sinónimo de CustName) y ARTIST.Name (sinónimo ArtistName). Si lo anterior no se hiciera, esta vista tendría dos columnas llamadas Name, lo cual está prohibido. La figura 12-20(b) muestra cómo se utilizan los sinónimos cuando se procesa esta vista. Observe los títulos de las columnas CUSTNAME y ARTISTNAME.

Examine los datos en la figura 12-20(b); considere lo que debe suceder cuando el usuario intenta actualizar la columna Title (Título). El título "Mystic Fabric" aparece en tres filas, a pesar de la tabla fundamental WORK (TRABAJO) sólo tiene dos renglones con este valor. Cuando el usuario actualiza Title (título), ¿qué debe hacer Oracle? ¿Debe actualizar los renglones de WORK que usan la vista? ¿Debe crear nuevos renglones en WORK y conectarlos con los renglones en esta vista? ¿Debe crear algo más? No es posible escribir un código generalizado en el DBMS para manejar correctamente cada posibilidad.

Los DISPARADORES INSTEAD OF permiten que el programador especifique las acciones de aplicación única cuando se intenta actualizar una vista. La figura 12-21 muestra un disparador de dicho tipo, el cual procesa las actualizaciones en la columna Title (Título) de la vista de CustomerPurchases (ComprasdelCliente). Debido a que los DISPARADORES INSTEAD OF no pueden tener la frase UPDATE OF, como se utilizó en el ejemplo del DISPARADOR BEFORE, se tiene que escribir code (codificar) para determinar si se ha cambiado la columna Title.

Si el usuario está actualizando Title, entonces se realiza una actualización apropiada a la tabla WORK. Advierta que el disparador activa una actualización de la vista Cus-

➤ FIGURA 12-21

DISPARADOR INSTEAD OF Title_Update

```
CREATE OR REPLACE TRIGGER Title_Update
        INSTEAD OF UPDATE ON CustomerPurchases

        FOR EACH ROW

        BEGIN
                /* Do nothing unless update is on title column */


                IF :new.Title = :old.Title THEN
                      RETURN;
                END IF;


                UPDATE          WORK
                SET             Title = :new.Title
                WHERE           Title = :old.Title;

        END;
```

tomerPurchases (ComprasdelCliente) pero la actualización se hace en WORK, una de las tablas fundamentales de la vista. Esta acción es exactamente para lo que están diseñados los disparadores.

Este disparador debe generar resultados sorprendentes. Si el usuario introduce

```
UPDATE CustomerPurchases SET Title='aa', Copy='1/3' WHERE Title='bb';
```

se actualizará la columna Title, pero no le hará nada a la columna Copy. Sería mejor practicar el envío de un mensaje de alerta al usuario.

CONTROL DE ANOMALÍAS. Este análisis ha omitido el estudio del control de anomalías PL/SQL, lo cual es lamentable debido a que es importante y útil. Hay demasiado por hacer. Sin embargo, si en el futuro usted va a programar en PL/SQL, asegúrese de aprender este importante tema. Se puede utilizar en todos los tipos de programación PL/SQL, pero es especialmente útil en los DISPARADORES BEFORE e INSTEAD OF para la cancelación de las actualizaciones pendientes. Las anomalías son necesarias en Oracle debido a que las transacciones no se pueden invertir en los disparadores. La excepción se puede utilizar para generar mensajes de error y de alerta. Asimismo, mantienen mejor informado a Oracle sobre las acciones que ha realizado el disparador.

Por ejemplo, el disparador en la figura 12-21 tiene una característica singular. Si introduce

```
UPDATE CustomerPurchases SET Copy='5/5' WHERE Title='Mystic Fabric;
```

el disparador no actualizará ningún reglón. Sin embargo, Oracle reportará que se actualizarán todos los renglones en la vista que tenga un título igual a "Mystic Fabric". Esto se debe a que se transfirieron todos esos renglones al disparador y Oracle no sabe qué originó y qué no originó una actualización. Sin embargo, si usted escribe code (codificar) en este disparador para activar un evento de anomalía, Oracle sabrá que no se actualizó el renglón y señalará una cuenta correcta de los renglones actualizados.

➤ DICCIONARIO DE DATOS

Oracle mantiene un amplio diccionario de datos de metadatos que describe estructuras de tablas, secuencias, vistas, índices, restricciones, columnas, procedimientos almacenados y otros. También contiene el código original de los procedimientos, funciones, y disparadores, así como muchas cosas más.

El diccionario incluye metadatos sobre sí mismo en la tabla DICT, la cual puede consultar para conocer más sobre el contenido del diccionario de datos, pero tenga en

➤ FIGURA 12-22

Algunas tablas útiles en el diccionario de datos de Oracle

Nombre de la tabla	Contenido
DICT	Metadatos del diccionario de datos.
USER_CATALOG	Lista de tablas, vistas, secuencias y otras estructuras correspondientes al usuario.
USER_TABLES	Estructuras de las tablas del usuario.
USER_TAB_COLUMNS	Una hija de USER_TABLES. Contiene datos sobre las columnas de las tablas. Su sinónimo es COLS.
USER_VIEWS	Vistas del usuario.
USER_CONTRAINTS	Restricciones del usuario.
USER_CONS_COLUMNS	Una hija de USER_CONSTRAINTS. Contiene columnas en las restricciones.
USER_TRIGGERS	Contiene metadatos del disparador. Consulta Trigger_Name, Trigger_Type, y Trigger_Event. Atención: Trigger_Body no proporciona una lista útil.
USER_SOURCE	Se utiliza para obtener el texto del procedimiento MY TRIGGER, SELECT Text FROM USER_SOURCE WHERE Name = 'MY TRIGGER' AND Type = 'PROCEDURE'

cuenta que se trata de una tabla grande. Por ejemplo, se desplegarán más de 800 renglones si consulta los nombres de todas las tablas en el diccionario de datos.

Suponga que desea saber qué tablas contiene el diccionario de datos con respecto a las tablas del usuario o del sistema. La siguiente consulta arroja este resultado:

```
SELECT    Table_Name, Comments
FROM      DICT
WHERE     Table_Name LIKE ('%TABLES%');
```

Aproximadamente se devolverán 25 renglones. Uno de éstos se llama USER_TABLES. Para desplegar las columnas de dicha tabla, introduzca

```
DESC      USER_TABLES;
```

Usted puede emplear esta estrategia de consulta y descripción para obtener los metadatos del diccionario sobre los objetos y estructuras que desea. La figura 12-22 lista muchas de las vistas y sus propósitos. Las tablas USER_SOURCE y USER_TRIGGERS son útiles cuando desea saber qué código de la fuente se almacena actualmente en la base de datos para los procedimientos y disparadores.

A estas alturas, ya debe saber bastante sobre SQL como para navegar por sí mismo en el diccionario. Tenga en cuenta que Oracle almacena todos los nombres en letras mayúsculas. Si busca un disparador llamado On_Customer_Insert, busque ON_CUSTOMER_INSERT.

➤ CONTROL DE CONCURRENCIA

Oracle maneja tres diferentes niveles de aislamiento de la transacción y, además, permite que las aplicaciones coloquen locks explícitos. Sin embargo, el lock explícito no es recomendable, ya que puede interferir con el comportamiento predeterminado del bloqueo de Oracle y también porque aumenta la probabilidad de que se interrumpa la transacción.

Antes de analizar los niveles de aislamiento de la transacción necesitamos resumir la manera en que Oracle procesa los cambios de la base de datos. Oracle mantiene un **número de cambio del sistema** (SCN, por sus siglas en inglés), el cual es un valor grande de la base de datos que Oracle aumenta cada vez que se realizan cambios en ésta. Cuando se cambia un reglón, se almacena el valor actual del SCN junto con el renglón. Al mismo tiempo, se coloca la imagen anterior del renglón en un **segmento de rollback,** el cual es una memoria intermedia (buffer) que genera Oracle con el fin de deshacer y hacer el log de transacciones. La imagen anterior incluye el SCN que estaba en el renglón anterior al cambio. Después de terminar la actualización, Oracle incrementa el SCN.

Cuando una aplicación despliega una instrucción SQL, por decir algo,

```
UPDATE    MYTABLE
SET       MyColumn1='NewValue'
WHERE     MyColumn2='Something';
```

se registra el valor del SCN que estaba vigente en el momento en que se inició la instrucción. Llame a este valor la Instrucción SCN. Mientras se procesa la consulta, en este caso mientras se buscan los renglones con MyColumn2 = 'Something', Oracle solamente seleccionará los renglones que hayan hecho cambios con un valor SCN menor que o igual a la instrucción SCN. Cuando encuentra un reglón con un cambio efectuado y con un valor del SCN mayor que la instrucción SCN, busca en el segmento del rollback para encontrar una versión anterior del renglón. Busca los segmentos del rollback hasta que encuentra una versión del renglón con un cambio hecho que tenga un SCN menor que la instrucción SCN.

De esta manera, las instrucciones SQL siempre leen un conjunto de valores consistentes —aquellos que fueron confirmados al mismo tiempo o antes de que se iniciara la instrucción—. Como verá, a veces esta estrategia se amplía para aplicarse a las transacciones. En tal caso, todas las instrucciones en una transacción leen los renglones con un valor SCN menor al SCN que estaba vigente cuando inició la transacción.

Observe que Oracle sólo lee los cambios ya confirmados. Por lo tanto, no son posibles las lecturas sucias.

Oracle maneja los siguientes niveles de aislamiento de transacciones, Read Commited (de lectura comprometida), Serializable, y Read Only (Sólo Lectura). Los dos primeros se definen en la norma ANSI de 1992; Read Only es exclusivo de Oracle. La figura 12-23 resume estos niveles de aislamiento.

➤ FIGURA 12-23

Resumen de las facilidades para control de concurrencia de Oracle

Aislamiento de transacción de lectura comprometida	Es el nivel de aislamiento predeterminado de Oracle. No se permiten lecturas sucias, pero las repetidas pueden producir datos diferentes. Son posibles los fantasmas. Cada instrucción lee datos consistentes. Cuando se bloquean para actualizaciones, las instrucciones se revierten (rolled back) y se reinician cada vez que es necesario. Se detecta el deadlock y se hace rollback a una de las instrucciones bloqueadas.
Aislamiento de transacción serializable	No se permiten las lecturas sucias, las lecturas repetidas producen los mismos resultados y no se permiten los fantasmas. Todas las instrucciones en la transacción leen datos consistentes. El error "Cannot serialize" ("No se puede serializar") se presenta cuando una transacción intenta actualizar o eliminar un reglón con un cambio que se hizo en los datos y que se presentó después de que inició la transacción. También se presenta cuando las transacciones o instrucciones de bloqueo realizan sus cambios, o cuando se revierte (rollback) la transacción debido a un deadlock. Necesitan escribirse los programas de aplicación para controlar la excepción "Cannot serialize".
Aislamiento de transacción Read Only	Todas las instrucciones leen datos consistentes. No se permite ninguna inserción, actualización o eliminación.
Locks explícitos	No se recomiendan.

Nivel del aislamiento de transacción lectura comprometida

Recuerde, como se mencionó en el capítulo 11, que con el nivel de aislamiento de lectura comprometida no se permiten las lecturas sucias; sin embargo, las lecturas no son repetibles y son posibles los fantasmas. La lectura comprometida es el nivel predeterminado de aislamiento de la transacción de Oracle, puesto que Oracle nunca lee cambios de datos que no se realizaron (uncommited).

Con el aislamiento de lectura comprometida cada instrucción es consistente, pero dos instrucciones diferentes en la misma transacción pueden leer datos inconsistentes. Como se definió en el capítulo anterior, este nivel es igual a la consistencia de nivel instrucción. Si se requiere consistencia de nivel de la transacción, entonces se debe utilizar el aislamiento serializable. Sin embargo, no confunda la consistencia de la instrucción con el problema de actualización perdida. Oracle prohíbe las actualizaciones perdidas debido a que nunca lee datos sucios.

Debido a la manera en que utiliza SCN, Oracle nunca necesita colocar locks de lectura. Sin embargo, cuando tenga que cambiar o eliminar un reglón Oracle colocará un bloqueo exclusivo en éste antes de realizar el cambio o la eliminación. Si otra transacción tiene un lock exclusivo en el renglón la instrucción esperará. Si se revierte la transacción del bloqueo, procede el cambio o la eliminación.

Si ocurre que la transacción bloqueando llega a commit, entonces se da el nuevo valor de SCN a la instrucción, ésta (no la transacción) da rollback e inicia de nuevo. Cuando se da rollback a una instrucción, los cambios que se hicieron mediante ésta se eliminan usando los segmentos del rollback.

Debido a que se usan locks exclusivos se pueden presentar deadlocks. Cuando eso sucede, Oracle lo detecta mediante una gráfica de espera y da rollback a una de las instrucciones problemáticas.

Nivel de aislamiento de transacción serializable

Como aprendió en el capítulo 11, el aislamiento de transacción serializable no permite lecturas sucias, éstas siempre son repetitivas y no se permiten los fantasmas. Oracle maneja el aislamiento serializable de la transacción, pero para que se ejecute el programa de aplicación debe desempeñar una función.

Utilice la orden Set (establecer) para cambiar el nivel de aislamiento de la transacción dentro de una transacción. La siguiente instrucción establece el aislamiento serializable para la duración de la transacción:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Para cambiar el nivel de aislamiento de todas las transacciones de una sesión, utilice la instrucción ALTER:

```
ALTER SESSIONS SET ISOLATION_LEVEL SERIALIZABLE;
```

Cuando el nivel de aislamiento es serializable Oracle guarda el SCN al mismo tiempo que comienza la transacción. Llame a este valor Transacción SCN. Conforme procede la transacción, Oracle sólo lee los cambios realizados que tengan un valor SCN menor o igual a la Transacción SCN. Por consiguiente, las lecturas siempre son repetitivas y no pueden presentarse fantasmas.

En tanto que la transacción no intente actualizar o eliminar algún reglón que tenga un cambio realizado con un SCN mayor que la transacción SCN, ésta procederá normalmente. Sin embargo, si la transacción intenta actualizar o borrar tal renglón, Oracle desplegará el error “Cannot serialize” cuando se presente la actualización o eliminación. En ese momento, el programa de aplicación debe desempeñar un papel. Puede hacer los cambios realizados hasta ese punto, hacer rollback a la transacción entera, o llevar a cabo alguna otra acción. Cualquier programa que se ejecute bajo un aislamiento serializable debe incluir dicho código de manejo de anomalías.

Asimismo, cuando una transacción que se ejecuta bajo un aislamiento serializable intenta actualizar o eliminar un reglón que ha sido bloqueado exclusivamente por una transacción o instrucción distinta, la transacción queda en espera. Si después se hace rollback a la transacción o a la instrucción bloqueante, entonces la transacción puede continuar. Sin embargo, si se realiza la transacción del bloqueante, entonces Oracle generará el error "Cannot serialize" ("No se puede serializar") y la aplicación necesitará procesar dicha anomalía.

De manera similar, si se hace rollback a una transacción serializable debido a un deadlock, también se generará el error "Cannot serialize".

Aislamiento de transacción Sólo lectura

Con este nivel de aislamiento, la transacción sólo lee los renglones que tengan cambios realizados con un valor SCN menor que o igual a la transacción SCN. Si la transacción encuentra renglones con cambios que tengan un valor SCN mayor que la transacción SCN, Oracle se basará en los segmentos del rollback y reconstruirá el renglón como estaba antes de la transacción SCN. Con este nivel de aislamiento de la transacción no se permite ninguna inserción, actualización o eliminación.

Comentarios adicionales sobre Locking

La aplicación puede invocar locks explícitamente mediante la forma SELECT FOR UPDATE (SELECCIONAR PARA ACTUALIZAR) de la instrucción seleccionada. No se lo recomendamos y será mejor que no la utilice hasta que haya aprendido más sobre el locking de Oracle, del que hemos hablado.

Tras bambalinas, Oracle utiliza mucho una gran variedad de locks para proporcionar los niveles de aislamiento que mencionamos. Existe un lock compartido de renglones, así como diversos tipos de locks de tablas. Existen otros locks que se utilizan internamente en Oracle. Puede aprender más sobre ellos en la documentación de Oracle.

Para reducir la probabilidad del conflicto de locks, Oracle no promueve locks de un nivel a otro. Los locks de renglones siguen siendo locks de renglones, incluso si existen cientos de ellos o centenares de renglones de una tabla. Como verá en el siguiente capítulo, esta estrategia es diferente a la de SQL Server. La corporación Oracle afirma que el no promover los locks es una ventaja, especialmente si se considera el resto de la arquitectura de locks de Oracle.

➤ SEGURIDAD DE ORACLE

Como se describió en el capítulo 11, el administrador de la base de datos define a los usuarios mediante la creación de una cuenta del usuario con nombre y contraseña. A cada usuario se le asigna un perfil: una lista de los límites de recursos del sistema que se asignan a dicho usuario. Éstos incluyen el tiempo total de CPU que puede utilizar por sesión o por llamada de Oracle, así como otros límites similares.

Oracle define los privilegios del sistema y de los objetos. Los privilegios del sistema son los derechos a tareas generales tales como SELECT ANY TABLE (SELECCIONAR CUALQUIER TABLA), y UPDATE ANY TABLE (ACTUALIZAR CUALQUIER TABLA). Los privilegios de los objetos se refieren a las acciones en los elementos de construcción de la base de datos tales como tablas, vistas y secuencias. Una instrucción GRANT (CONCEDER) puede utilizarse para dar un privilegio a otro usuario.

Un rol de Oracle es un conjunto de privilegios y otros roles. Se le puede asignar un rol a muchos usuarios y un usuario puede tener muchos roles. Un ejemplo de un rol para la galería View Ridge es MANAGER (ADMINISTRADOR). Éste le daría todos los privilegios del sistema y de los objetos que se necesitan para desempeñar la función de la administración de View Ridge. De esta manera, se les otorgaría a ciertos empleados de View Ridge las cuentas de usuario que les otorgara dicho rol.

Usted puede aprender más sobre la seguridad de Oracle si busca en la documentación Usuario, Roles, o Privilegios.

➤ RESPALDO Y RECUPERACIÓN CON ORACLE

Oracle proporciona un sofisticado conjunto de facilidades y utilidades para el proceso de respaldo y de recuperación. Se pueden utilizar de distintas maneras para realizar el respaldo y la recuperación apropiados para las bases de datos; desde una pequeña base de datos para trabajo de grupo que se pueda respaldar en la noche cuando no se utilice, hasta grandes bases de datos interorganizacionales que operen las 24 horas del día, los 7 días de la semana (24/7) y nunca se puedan apagar.

Facilidades de recuperación de Oracle

Oracle mantiene tres tipos de archivos que son importantes para el respaldo y la recuperación. Los **archivos de datos** contienen datos del usuario y del sistema. Debido a la manera en que Oracle escribe en un disco las memorias intermedias de datos, en un momento arbitrario, los archivos de datos pueden contener cambios realizados y no realizados. Por supuesto Oracle procesa las transacciones de tal manera que los cambios no efectuados se realizan o finalmente se eliminan, pero una fotografía instantánea de los archivos de datos en cualquier momento arbitrario incluirá los cambios que no se hicieron. Por lo tanto, cuando Oracle se apague, o cuando se haga algún tipo de respaldo, deben limpiarse los archivos de datos de modo que sólo permanezcan en ellos los cambios realizados.

Los **archivos rehacer** (ReDo files) contienen logs de los cambios en la base de datos; son respaldos de los segmentos del rollback utilizados para el procesamiento concurrente. Los **archivos de control** (Control files) son pequeños archivos que describen nombre, contenido, y localizaciones de diversos archivos utilizados por Oracle. Con frecuencia Oracle actualiza los archivos de control y disponibles para que una base de datos sea operativa.

Existen dos tipos de archivos rehacer. Los archivos rehacer en línea (On Line ReDo files) se mantienen en el disco y contienen los segmentos del rollback de los cambios recientes de la base de datos. Los **archivos fuera de línea** o de **rehacer** (Offline o Archive ReDo files) son respaldos de los archivos rehacer en línea. Se almacenan por separado de los archivos rehacer en línea, y no necesariamente tienen que residir en disco. Oracle puede operar en el modo ARCHIVELOG o NOARCHIVELOG. Si se está ejecutando en el modo ARCHIVE, cuando los archivos rehacer en línea se llenan en su totalidad, se copian a los archivos Rehacer.

Los archivos de control y los archivos rehacer en línea son tan importantes que Oracle recomienda que se guarden dos copias activas de ellos, lo cual es un proceso de **multiplexaje** (multiplexing) en la terminología de Oracle.

Tipos de fallas

Las técnicas de recuperación de Oracle dependen del tipo de falla. Cuando ocurre una **falla en la aplicación,** por ejemplo, debido a errores en la lógica de la aplicación, Oracle hace rollback a los cambios no realizados que hizo mediante tal aplicación usando los segmentos de rollback en memoria y los archivos rehacer en línea, según sea necesario.

Otros tipos de recuperación de fallas son más complicados y dependen de qué tipo sean. Una **falla de concurrencia** se presenta cuando Oracle falla por sí mismo debido a un desperfecto en el sistema operativo o en el hardware de la computadora. Una **falla del medio** ocurre cuando Oracle no puede escribir en un archivo físico. Esto puede presentarse debido a la ruptura de la cabeza de un disco u otra falla en el disco, a que los dispositivos necesarios no están encendidos, o a la alteración de un archivo.

RECUPERACIÓN DE FALLAS DE UNA INSTANCIA. Cuando se reinicia Oracle después de una falla primero examina el archivo de control para encontrar dónde se

localiza el resto de los archivos. Después, confronta los logs de rehacer en línea contra los archivos de datos. Se adelanta a realizar todos los cambios en el log ReDo, los cuales no se alcanzaron a escribir en los archivos de datos al momento de la falla. En el proceso de roll forward se llenan los segmentos del rollback con los registros de las transacciones en el log ReDo.

Después del roll forward, los archivos de datos pueden contener cambios no realizados. Éstos pudieron haber estado en los archivos de datos en el momento en que se presentó la falla o quizá fueron introducidos mediante el proceso de roll forward. De cualquier manera, Oracle los elimina a través del rollback de dichos cambios no realizados, usando los segmentos del rollback que se crearon durante el roll forward. De tal modo, las transacciones no necesitan esperar a que concluya el rollback, puesto que todas las transacciones no realizadas se marcan como TERMINADAS (DEAD). Si se bloquea una nueva transacción mediante un cambio realizado por una transacción TERMINADA, el controlador destruye los locks que sostenían la transacción TERMINADA.

Los logs de registros ReDo almacenados no se utilizan para la recuperación. Por consiguiente, ésta puede realizarse en el modo ARCHIVELOG o en el modo NOARCHIVELOG.

RECUPERACIÓN DE LA FALLA DE MEDIO. Para recuperar una falla de medio, la base de datos se restablece desde un respaldo. Si la base de datos se ejecutaba en el modo NOARCHIVELOG, entonces no se podrá hacer nada. El registro rehacer en línea no será útil debido a que se referirá a los cambios realizados después de que se hizo el respaldo. La empresa debe encontrar otra manera de recuperar los cambios hechos a la base de datos. (Por cierto, sería un mal momento para comenzar a preocuparse por esto.)

Si Oracle funcionaba en el modo ARCHIVELOG, entonces se habrán copiado al archivo los logs rehacer en línea. Para su recuperación, se restaura la base de datos desde un respaldo y entonces a ésta se hace roll forward aplicando los archivos de log Rehacer. Después de que termina el proceso de roll forward, mediante el rollback se eliminan los cambios realizados por las transacciones no realizadas, como se describió anteriormente.

Hay dos clases de respaldos posibles. Un **respaldo consistente** es aquel en el cual se han eliminado de los archivos de datos todos los cambios no realizados. Se debe interrumpir la actividad de la base de datos, vaciar todas las memorias intermedias al disco, y eliminar los cambios efectuados por cualquier transacción no realizada. Obviamente, este tipo de respaldo no se puede llevar a cabo si la base de datos maneja operaciones 24/7.

Un **respaldo inconsistente** puede contener cambios no realizados. En cierta medida es un respaldo volátil, que se hace mientras Oracle procesa la base de datos. Para la recuperación, dichos respaldos pueden hacer consistentes los registros de log para realizar o para hacer rollback en todas las transacciones que estaban en proceso cuando se realizó el respaldo. Los respaldos inconsistentes se pueden hacer en algunas partes de la base de datos. Por ejemplo, en una aplicación 24/7 cada noche se puede respaldar un séptimo de la base de datos; después de una semana se habrá hecho una copia completa.

El control de recuperación de Oracle (RMAN, por sus siglas en inglés) es un programa de utilería que se emplea para crear respaldos y llevar a cabo recuperaciones. Se le puede ordenar al RMAN que cree una base de datos especial para recuperación, que contenga datos sobre los archivos y las operaciones de recuperación. Las especificaciones de este programa están más allá del alcance de este análisis.

➤ TEMAS NO ANALIZADOS EN ESTE CAPÍTULO

Existen diversas características importantes de Oracle que no analizamos en este capítulo. En una, Oracle utiliza las estructuras orientadas a objetos, y los programadores pueden utilizarlas para definir sus propios tipos de datos abstractos. Oracle se puede utilizar también para crear y procesar las bases de datos que son híbridos de las bases de datos tradicionales y de las de objetos. Dichos híbridos se llaman bases de datos de objetos relacionales y los describiremos en el capítulo 18.

Asimismo, la empresa Oracle maneja el procesamiento distribuido de bases de datos, en el cual la base se almacena en más de una computadora. Este tema se presentará en el capítulo 17. Además, hay diversas utilerías de Oracle que no se analizaron. El cargador de Oracle es un programa de utilerías para ingresar datos de gran volumen en una base de datos de Oracle. Se pueden emplear otras utilerías para medir y ajustar el funcionamiento de Oracle.

Sin embargo, hemos analizado las características y los temas más importantes de Oracle. Si ha aprendido estos conceptos, va bien en su camino de convertirse en un exitoso programador de Oracle.

➤ RESUMEN

Oracle es un DBMS poderoso y amplio que funciona en muchos sistemas operativos diferentes y tiene diversos productos. Este capítulo indica el uso de la utilería de SQL Plus de Oracle, que puede utilizarse para crear y procesar SQL y PL/SQL con todas las versiones de Oracle. PL/SQL es un lenguaje que agrega facilidades de programación al lenguaje SQL.

Usted puede crear una base de datos mediante el Asistente de configuración de bases de datos, usando los procedimientos de creación de base de datos que proporciona Oracle, y a través de la instrucción SQL CREATE DATABASE. El Asistente de configuración de bases de datos crea archivos predeterminados de bases de datos y de registros. SQL Plus tiene un editor de textos limitado que mantiene la instrucción actual en una memoria intermedia (buffer) de líneas múltiples. SQL Plus puede configurarse para invocar editores de textos tales como el Bloc de notas.

La instrucción SQL CREATE TABLE se utiliza para crear tablas de Oracle. Esta instrucción nombra la tabla nueva y crea columnas en ésta de acuerdo con una lista de nombres de columnas, tipos de datos, y propiedades que están entre paréntesis. En Oracle se pueden mantener las llaves sustitutas usando secuencias. Los objetos de secuencias tienen un método NextVal, que proporciona el valor siguiente de una secuencia, y un método CurrVal, el cual da el valor actual de una secuencia. Por desgracia, la implementación de llaves sustitutas con secuencias no garantiza que la tabla tendrá valores válidos de las llaves sustitutas.

Las tablas (y sus datos) pueden eliminarse de la base de datos con la instrucción DROP. Las columnas de las tablas se pueden suprimir con la instrucción ALTER TABLE. Las relaciones se declaran mediante la definición de las restricciones de las llaves externas mediante la instrucción ALTER TABLE . . . ADD CONSTRAINT . . . La eliminación de los renglones padre puede propagarse a las tablas con llaves externas dependientes, mediante la frase ON DELETE CASCADE.

Los índices se crean para dar unicidad y permitir la recuperación rápida de los valores de las columnas. Se crean con la instrucción CREATE INDEX. Se pueden agregar, modificar o eliminar columnas de una tabla con la instrucción ALTER. Las RESTRICCIONES DE VERIFICACIÓN se utilizan para restringir los valores que puedan tener las columnas. Las condiciones en dichas restricciones deben evaluar la veracidad o falsedad sólo mediante las constantes y los valores de las columnas en el renglón actual de la tabla en la que están definidos. Las restricciones se agregan y se eliminan con la instrucción ALTER TABLE.

Una vista SQL es resultado de una expresión SQL SELECT que utilice la selección, proyección, y join. Tales vistas pueden tener cuando mucho una trayectoria de valores múltiples. Las vistas se construyen con la instrucción CREATE VIEW. Las vistas de join se crean mediante un join y por lo general no se actualizan con las instrucciones INSERT, UPDATE y DELETE. Sin embargo, se pueden actualizar con los DISPARADORES (TRIGGERS) INSTEAD OF.

Las instrucciones PL/SQL se pueden guardar en archivos y procesar con SQL Plus. También se pueden registrar en la base de datos como procedimientos almacenados y ser invocadas desde otros programas PL/SQL, o desde programas de aplicación. En las figuras 12-15 y 12-16 se muestran ejemplos de procedimientos almacenados. Los disparadores (triggers) de Oracle son programas PL/SQL o Java que se invocan cuando se

presenta una actividad específica de la base de datos. En las figuras 12-18, 12-19, y 12-20, respectivamente, se muestran ejemplos de DISPARADORES (TRIGGERS) BEFORE, AFTER e INSTEAD OF.

Oracle mantiene un diccionario de datos de metadatos. Esos metadatos del diccionario se almacenan en la tabla DICT. Usted puede consultar esta tabla para determinar el contenido del diccionario.

Oracle tiene niveles de aislamiento de la transacción de lectura comprometida, serializable y de sólo lectura. Debido a la manera en que se procesan los valores SCN, Oracle nunca lee datos sucios. El aislamiento serializable puede presentarse, pero se debe escribir el programa de aplicación para procesar la anomalía "Cannot serialize". Las aplicaciones pueden colocar bloqueos explícitos mediante las instrucciones SELECT FOR UPDATE, pero esto no es recomendable.

Con Oracle, el DBA define usuarios y privilegios. Un rol es un grupo de privilegios. Se le pueden otorgar diversos roles a un usuario y un rol a muchos usuarios.

En la recuperación de Oracle se utilizan tres tipos de archivos: de datos, de rehacer en línea y fuera de línea y de control. Si se ejecuta en el modo ARCHIVELOG, Oracle registra todos los cambios realizados a la base de datos. Oracle puede recuperarse de una falla de aplicación y de una falla de prueba sin tener que usar el log de registros archivados. Sin embargo, los registros de archivo son necesarios para la recuperación de una falla de medio. Los respaldos pueden ser consistentes o inconsistentes. Un respaldo inconsistente puede hacerse consistente procesando un archivo de log.

➤ PREGUNTAS DEL GRUPO I

12.1 Describa las características generales de Oracle y de la serie de productos Oracle. Explique por qué es considerablemente complejo su dominio.

12.2 ¿Cuál es SQL Plus y qué propósito tiene?

12.3 Nombre tres maneras de crear una base de datos de Oracle. ¿Cuál es la más fácil?

12.4 Explique cómo se cambia un reglón en la memoria intermedia (buffer) de SQL Plus. Suponga que existen tres instrucciones en la memoria intermedia, la parte central se encuentra en la instrucción 3, y usted desea cambiar la segunda instrucción de CustID=1000 a CustomerID=1000.

12.5 ¿Cómo puede establecer el directorio predeterminado para que SQL lo utilice?

12.6 Muestre la instrucción SQL necesaria para crear una tabla llamada TI con las columnas C1, C2, y C3. Suponga que C1 es una llave sustituta; que C2 tiene datos de caracteres de una longitud máxima de 50, y que C3 contiene una fecha.

12.7 Muestre la instrucción necesaria para crear una secuencia que comience en 50 y aumente en 2. Nombre a esa secuencia T1Seq.

12.8 Muestre cómo insertar un reglón en la tabla T1 (pregunta 12.6) usando la secuencia creada en la pregunta 12.7.

12.9 Muestre una instrucción SQL para consultar el renglón creado en la pregunta 12.8.

12.10 Explique los problemas inherentes al uso de las secuencias para las columnas de la llave sustituta.

12.11 Muestre las instrucciones SQL para eliminar la tabla T1 y para eliminar SeqT1.

12.12 Muestre las instrucciones SQL para eliminar la columna C3 de la tabla T1.

12.13 Muestre las instrucciones de SQL para crear una relación entre las tablas T2 y T3. Suponga que la tabla T3 tiene una columna de una llave externa llamada FK1 que se relaciona con la tabla T2, y que las eliminaciones en la tabla T2 deben provocar las eliminaciones en la T3.

12.14 Responda la pregunta 12.13, pero no provoque eliminaciones.

12.15 Muestre cómo eliminar una relación con SQL.

12.16 Muestre las instrucciones SQL para crear un índice único en las columnas C2 y C3 de la tabla T1.

12.17 ¿Bajo qué circunstancias se deben utilizar los índices?

12.18 Muestre las declaraciones de SQL para agregar una columna nueva C4 a la tabla T1. Suponga que T1 tendrá valores corrientes de hasta $1 millón.

12.19 ¿Bajo qué condiciones se puede eliminar una columna en una tabla?

12.20 ¿Bajo qué condiciones puede usted agregar una columna a una tabla?

12.21 Explique cómo agregar una columna NOT NULL a una tabla.

12.22 ¿Bajo qué condiciones puede cambiar la extensión de un carácter o de una columna numérica*?*

12.23 ¿Bajo qué condiciones puede cambiar el tipo de datos de una columna?

12.24 Muestre cómo agregar una restricción para especificar que la columna C4 de la tabla T1 no puede ser menor que 1000.

12.25 Muestre cómo agregar una restricción para especificar que la columna C4 de la tabla T1 no puede ser menor que la columna C5 de la tabla T1.

12.26 Para la base de datos View Ridge que analizamos en este capítulo construya una vista que contenga Nombre, Ciudad, y Estado de un cliente. Llámele a esa vista CustView.

12.27 Para la base de datos View Ridge construya una vista que tenga el nombre del cliente y el nombre del artista para todas las obras de arte que compró el cliente.

12.28 Para la base de datos View Ridge construya una vista que contenga el nombre del cliente y el nombre del artista para todos los artistas en los que se interese el cliente. Explique la diferencia entre esta vista y la de la pregunta 12.27.

12.29 ¿Puede combinar las vistas las preguntas 12.27 y 12.28 en una sola? ¿Por qué? Explique.

12.30 ¿Cómo puede actualizar una vista de join mediante Oracle?

12.31 Cree un archivo de las instrucciones PL/SQL que describa la estructura de las tablas CUSTOMER, ARTIST, WORK, TRANSACTION, y CUSTOMER_ARTIST_INT. Almacene el archivo con el nombre VRTabs.sql y muestre cómo invocar el procedimiento PL/SQL mediante SQL Plus.

12.32 En un procedimiento PL/SQL, ¿qué significan las palabras clave IN, OUT e IN OUT?

12.33 ¿Qué se debe hacer para ver la salida generada por el paquete DBMS_OUTPUT de Oracle? ¿Qué límites existen en dicha salida?

12.34 Explique cómo funciona la instrucción PL/SQL FOR *variable* IN *cursorname*.

12.35 ¿Qué instrucción se utiliza para obtener los errores cuando se compilan los procedimientos almacenados (stored procedures) y los disparadores (triggers)?

12.36 ¿Cuál es la sintaxis de la instrucción BEGIN TRANSACTI0N en PL/SQL? ¿Cómo se inicia una transacción?

12.37 En el procedimiento de almacenado en la figura 12-16, ¿cómo se utilizan los valores de las variables tid y aid, si es que no existe ningún reglón adecuado para el renglón en la base de datos? ¿Cómo se utilizan si sólo existe un reglón adecuado de TRANSACCIÓN en la base de datos?

12.38 Explique el propósito de los disparadores BEFORE, AFTER, e INSTEAD OF.

12.39 Cuando una actualización está en proceso, ¿cómo puede el código del disparador obtener el valor de una columna, por ejemplo, C1, antes de que inicie la actualización? ¿Cómo puede el código del disparador obtener el valor en el cual está establecida la columna?

12.40 Explique por qué son necesarios los disparadores para las vistas de joins.

12.41 Explique qué sucedería si se eliminara la instrucción IF en el disparador INSTEAD OF en la figura 12-21.

12.42 Muestre una instrucción SQL para obtener los nombres de las tablas que el diccionario de los datos contiene sobre disparadores.

12.43 ¿Cuáles son los tres niveles de aislamiento que maneja Oracle?

12.44 Explique cómo utiliza Oracle el número de cambio del sistema al leer los datos actuales en un periodo determinado.

12.45 ¿Bajo qué circunstancias leerá Oracle datos sucios?

12.46 Explique cómo maneja Oracle los locks conflictivos cuando una transacción opera en el modo de aislamiento LECTURA COMPROMETIDA.

12.47 Muestre la instrucción SQL necesaria para establecer un nivel de aislamiento de la transacción llamado SERIALIZABLE para una sesión completa.

12.48 ¿Qué sucede cuando una transacción en el modo serializable intenta actualizar datos que han sido actualizados mediante diferentes transacciones? Suponga que el SCN es menor que el SCN de la transacción. Suponga que el SCN es mayor que el SCN de la transacción.

12.49 Describa tres circunstancias bajo las cuales una transacción podría recibir la excepción "Cannot serialize".

12.50 Explique cómo procesa Oracle el nivel de aislamiento de transacción de sólo lectura.

12.51 Explique el uso de usuario, privilegio, y función en la seguridad de Oracle.

12.52 ¿Cuáles son los tres tipos de archivos que son importantes para el procesamiento del respaldo y la recuperación con Oracle?

12.53 ¿Cuál es la diferencia entre los registros rehacer en línea y los archivos rehacer fuera de línea? ¿Cómo se utiliza cada tipo?

12.54 ¿Qué significa el multiplexaje en el contexto de la recuperación de Oracle?

12.55 Explique cómo se recupera Oracle de la falla de aplicación.

12.56 ¿Cuál es una falla de instancia y cómo se recupera Oracle de ella?

12.57 ¿Cuál es falla de medio y cómo se recupera Oracle de ella?

➤ PROYECTOS

Para los siguientes proyectos utilice Oracle con el fin de crear la base de datos de la galería View Ridge, como se describió en este capítulo. Considere la vista de la base de datos llamada Artist View que se muestra para las preguntas del grupo II al final del capítulo 10.

A. Escriba un procedimiento PL/SQL para leer la parte de esta vista de la tabla ARTIST. Despliegue los datos que lee. Acepte al nombre del artista como parámetro de entrada.

B. Escriba un procedimiento PL/SQL para leer las tablas ARTISTA, TRANSACCIÓN y CLIENTE (en TRANSACCIÓN) en esta vista. Despliegue los datos que lee mediante DBMS_OUTPUT. Acepte el nombre del artista como parámetro de entrada.

C. Escriba un procedimiento PL/SQL para leer todas las tablas de la vista. Despliegue los datos que lee mediante DBMS_OUTPUT. Acepte al nombre del artista como parámetro de entrada.

D. Escriba un procedimiento PL/SQL para asignar el interés que tiene un cliente nuevo en un artista. Suponga que el nombre del artista y el del cliente son entradas. Si el nombre del cliente no es único, despliegue un mensaje de error. Verifique si hay alguna duplicación en la tabla CUSTOMER_ARTIST_INT antes de insertar el renglón nuevo. Despliegue un mensaje de error si existe una duplicación.

E. Codifique un DISPARADOR BEFORE que compruebe las inserciones y las modificaciones en Nacionalidad del ARTISTA. Si el valor nuevo es 'Británico' cámbielo por 'Inglés'.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Utilice Oracle para crear una base de datos con las siguientes tablas:

CLIENTE (ClienteID), Nombre, Teléfono, CorreoElectrónico.
ESTUFA (NúmerodeSerie, tipo, Versión, FechadeManufactura)
REGISTRO (*ClienteID*, *NúmerodeSerie*, *Rfecha*)
ESTUFA-REPARACIÓN (NúmeroFacturaReparación, NúmerodeSerie, FechadeReparación, Descripción, Costo, *ClienteID*)

Suponga que las llaves primarias de CLIENTE, ESTUFA y ESTUFA-REPARACIÓN son llaves sustitutas y forme secuencias para cada una de ellas. Cree las relaciones para hacer cumplir las siguientes restricciones de integridad referencial:

- ClienteID de REGISTRO es un subconjunto de CLIENTEID de CLIENTE
- Número de serie de REGISTRO es un subconjunto de NúmerodeSerie de ESTUFA
- NúmerodeSerie de ESTUFA-REPARACIÓN es un subconjunto de NúmerodeSerie de ESTUFA
- ClienteID de ESTUFA-REPARACIÓN es un subconjunto de ClienteID de CLIENTE

No haga eliminaciones en cascada.

A. Llene sus tablas con ejemplos de datos y despliéguelos.

B. Cree un procedimiento de almacenado para registrar una estufa. El procedimiento recibe nombre, teléfono, correo electrónico, y número de serie de la estufa . Si el cliente ya existe en la base de datos (el nombre, el teléfono, y la dirección de correo electrónico concordarán), utilice el ClienteID de dicho cliente para el REGISTRO. De lo contrario, cree un nuevo renglón CLIENTE. Suponga que una estufa con el número de serie de entrada ya existe en la base de datos. De lo contrario, imprima un error y dé rollback a los cambios de la tabla CLIENTE. Codifique y compruebe su procedimiento.

C. Cree un procedimiento de almacenado para registrar una reparación de estufa. El procedimiento recibe nombre, teléfono, correo electrónico, número de serie de la estufa, descripción de la reparación, y costo. Suponga que se le proporciona el número de serie válido de la estufa; imprima un mensaje de error y si no es así no realice ningún cambio en la base de datos. Utilice un reglón de CLIENTE si concuerdan el nombre, el teléfono, y la dirección de correo electrónico; de lo contrario, cree un registro nuevo de CLIENTE. Suponga que debe crearse el renglón ESTUFA_REPARACIÓN. Si es necesario registre la estufa.

D. Cree una vista que contenga todos los datos de FiredUp para un cliente determinado. Denomine a esta vista Registro del cliente. Esta vista debe juntar los datos de CLIENTE, REGISTRO, ESTUFA y ESTUFA-REPARACIÓN. Escriba un procedimiento almacenado que acepte el nombre de un cliente y despliegue todos los datos para ese cliente en específico.

CAPÍTULO 13

Manejo de bases de datos con SQL Server 2000

Este capítulo describe las características y funciones básicas del SQL Server 2000 de Microsoft. El análisis usa el ejemplo de la Galería View Ridge, que se menciona en el capítulo 10, y lo compara con las tareas de administración de la base de datos de las que se habla en el capítulo 11. El enfoque y la orientación son similares a los de Oracle en el capítulo anterior. De hecho, debido a que el Instituto Nacional de Estándares Americanos (ANSI) ha estandarizado el lenguaje SQL, casi todo lo que aprenda acerca del procesamiento SQL con Oracle se relacionará también con SQL Server 2000. Por lo tanto, nos concentraremos más en las herramientas de diseño gráfico de SQL Server, que en las instrucciones SQL.

SQL Server es un producto amplio y complicado. En este capítulo lo podremos abordar sólo superficialmente. El objetivo es prepararlo a usted para que use el SQL Server en sus propios proyectos, y que adquiera las bases suficientes para que continúe aprendiendo por sí mismo, o en otras clases.

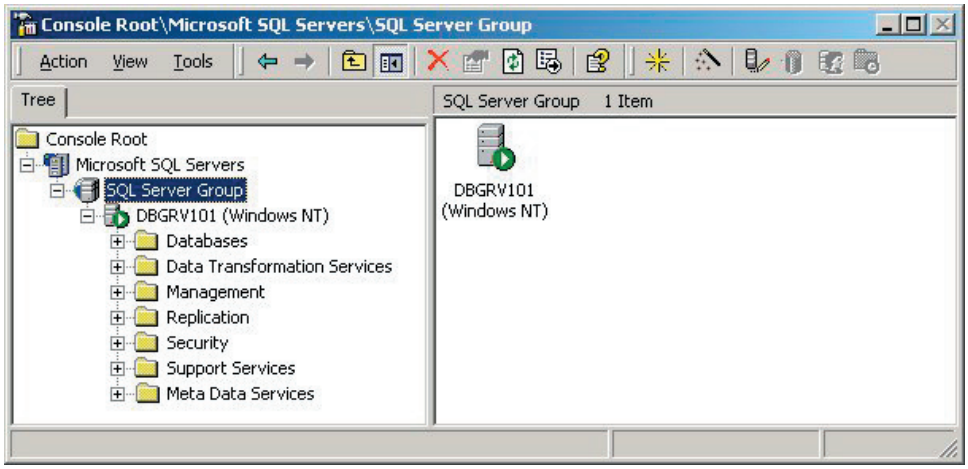
➤ INSTALACIÓN DE SQL SERVER 2000

Si compró la versión de este libro que tiene SQL Server, deberá instalarlo ahora. El CD de SQL Server que viene con este texto contiene una copia de evaluación que incluye una licencia válida por 120 días. Requiere Windows NT con Service Pack 5 o posterior, o Windows Profesional 2000, Servidor Windows 2000, Servidor Avanzado Windows 2000, o Windows 2000 Servidor Central de Datos. También necesita un mínimo de 64 MB de RAM y aproximadamente 250 MB de espacio en disco (es posible con menos, pero no es recomendable para nuestros propósitos).

Para instalar este software, registre su computadora con privilegios de Administración e inserte el CD-ROM. La instalación del programa deberá empezar automáticamente. De lo contario, haga doble clic en el ejecutable *autorun* en la parte superior. Haga clic en los componentes de SQL Server y después haga clic en el Instalador del Servidor de la base de datos. El resto del proceso es un programa de instalación común de Windows.

➤ FIGURA 13-1

Administrador empresarial SQL Server



Una vez que haya instalado el software podrá empezar a trabajar con SQL Server haciendo clic en Start/Programs/Microsoft SQL Server/Enterprise Manager. Después de que lo haya hecho, encuentre el icono etiquetado como Microsoft SQL Server en el lado izquierdo. Haga clic en el signo de más para abrirlo; abra SQL Server Group de la misma manera. A continuación verá el nombre de su computadora seguido por (Windows NT). Ábralo y verá desplegarse la pantalla que se muestra en la figura 13-1. En ésta, verá el nombre de la computadora que se usó para hacer dicha figura: DBGRV101.

➤ CREACIÓN DE UNA BASE DE DATOS CON SQL SERVER 2000

Para crear una nueva base de datos haga clic en la parte derecha de Databases y seleccione New Database (Nueva base de datos). Escriba el nombre de su base de datos (en este caso es View Ridge1) en la caja de texto Name (Nombre), como se muestra en la figura 13-2.

De manera preestablecida, SQL Server creará un archivo de datos y un archivo de log para cada base de datos. Puede crear múltiples archivos para ambos y asignar tablas específicas y logs a determinados archivos y grupos de archivos. Sin embargo, todo esto rebasa el ámbito de este análisis. Para aprender un poco más por su propia cuenta, haga clic en la parte derecha y seleccione Help (Ayuda). En el lado izquierdo del menú de Help busque en *File Groups* (Grupos de archivos) en la caja de búsqueda de texto para que inicie.

Por el momento emplee los tamaños predeterminados y los archivos que ofrece SQL Server. Como podrá ver, están disponibles con sólo hacer clic en Data Files en la pestaña (ficha) Transaction Log.

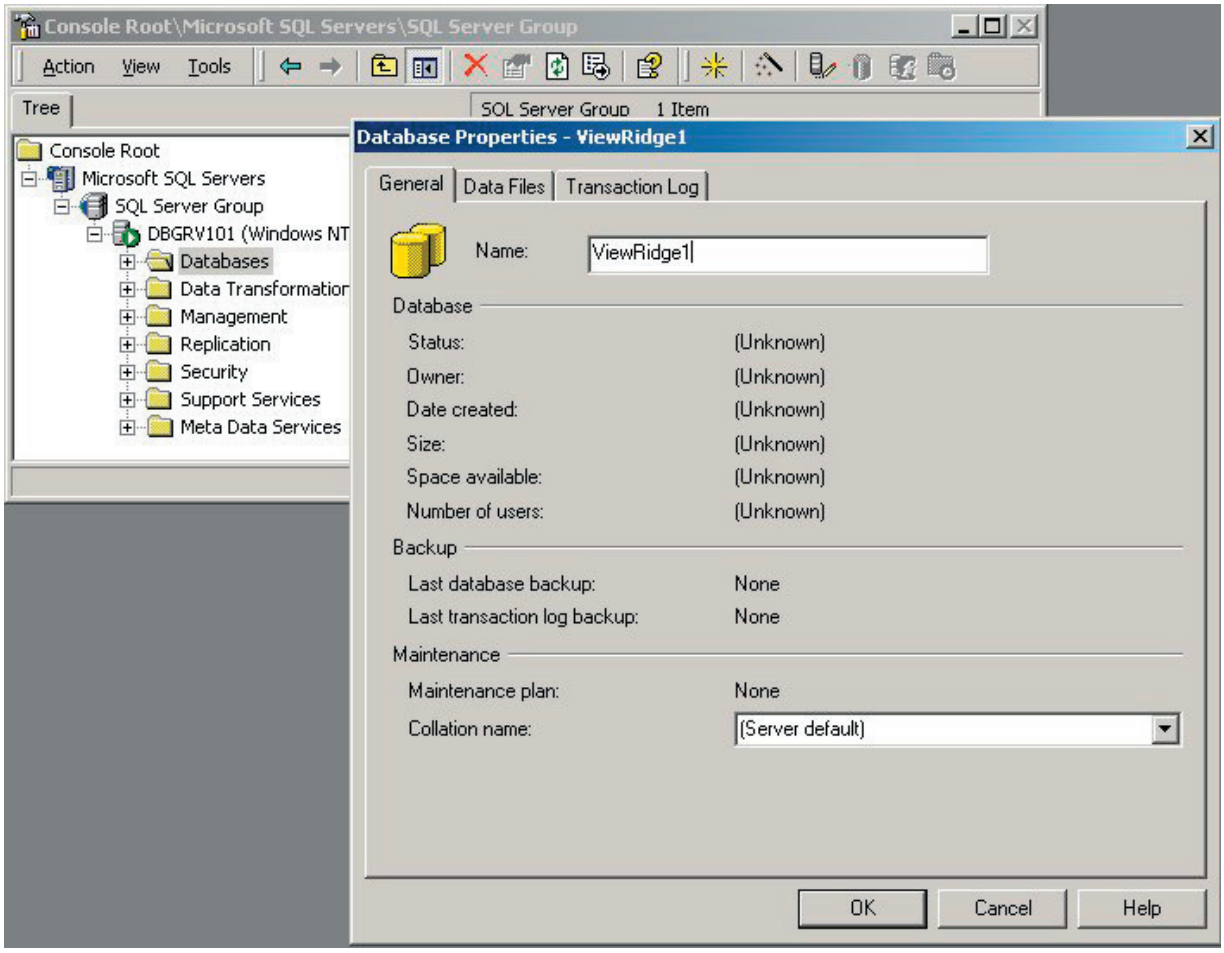
Una vez que haya creado su base de datos, abra la carpeta Databases, así como la que tiene el nombre de su base de datos. A continuación abra Tables (Tablas). Su pantalla deberá verse como se muestra en la figura 13-3, pero aún no tendrá ningunas tablas en uso. Todas las tablas que se muestran en su pantalla son tablas de sistema que SQL Server usa para manejar su base de datos. A propósito, *dbo* corresponde a la base de datos del propietario. Esto es, será suya si usted instaló SQL Server y creó esa base de datos.

CREACIÓN DE TABLAS

Hay dos maneras de crear y modificar tablas (y muchas estructuras de SQL Server para este propósito). La primera es escribir el código SQL usando las instrucciones CREATE o ALTER SQL, como lo hicimos en el capítulo anterior con Oracle. La segunda, es usar las facilidades gráficas de SQL Server. Debido a que los medios gráficos son diferentes a los que analizamos anteriormente, los usaremos la mayor parte del tiempo en este capítulo. Sin embargo, tenga en mente que las instrucciones SQL son la única forma de crear

➤ FIGURA 13-2

Creación de una base de datos con SQL Server



estructuras de bases de datos de manera programática, así que usted tiene que saber cómo usar ambas.

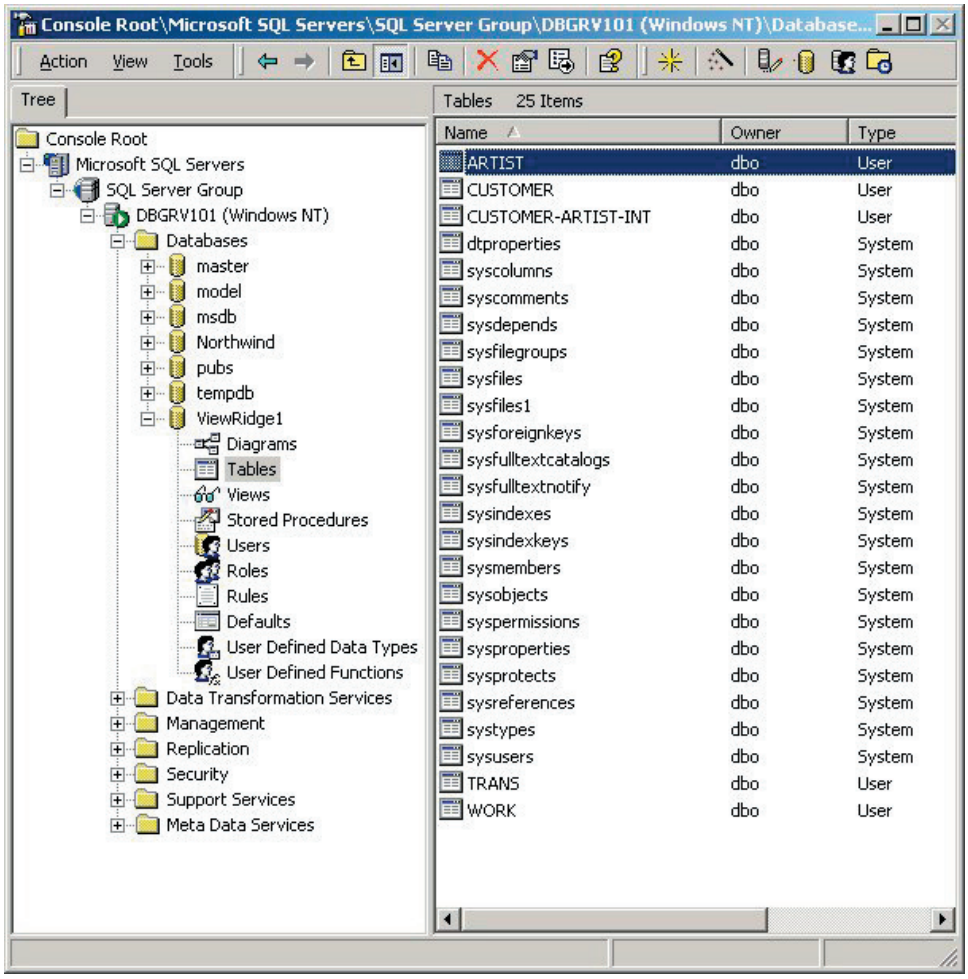
USO DE LAS INSTRUCCIONES DE CREACIÓN SQL. La figura 13-4 muestra unas instrucciones comunes (SQL CREATE TABLE). Como señalamos en el capítulo anterior, éstas siempre comienzan con CREATE TABLE, y después el nombre de la nueva tabla. A continuación se presenta una lista de las columnas de tabla encerradas entre paréntesis. Cada columna tiene un nombre, un tipo de datos y, por lo tanto, algunas propiedades especiales. Las descripciones de columna están separadas por comas, pero no hay coma después de la última columna.

En la figura 13-4 el nombre de la tabla es CUST (Cliente) y tiene cuatro columnas: CustomerID, Name (Nombre), AreaCode (CódigodeÁrea), y LocalNumber (NúmeroLocal). A la derecha de CustomerID está escrito *int* y es la primera llave de la tabla. Name tiene escrito char y es de 50 bytes de longitud. Name es NOT NULL, lo cual significa que no se permiten los valores nulos. Si no aparecen NULL o NOT NULL la columna será tomada como NULL.

El nombre está encerrado entre corchetes [Name]. Esto es necesario debido a que *Name* es una palabra reservada de SQL Server. Si no aparece entre corchetes, SQL Server tratará de interpretar *Name* como el nombre de una de sus construcciones. Por lo anterior, cada vez que usted esté usando una palabra reservada de SQL Server como un identificador del usuario, póngala entre corchetes. Si no está seguro de que una palabra ha sido reservada, póngala entre corchetes. No se corre ningún peligro con hacerlo así.

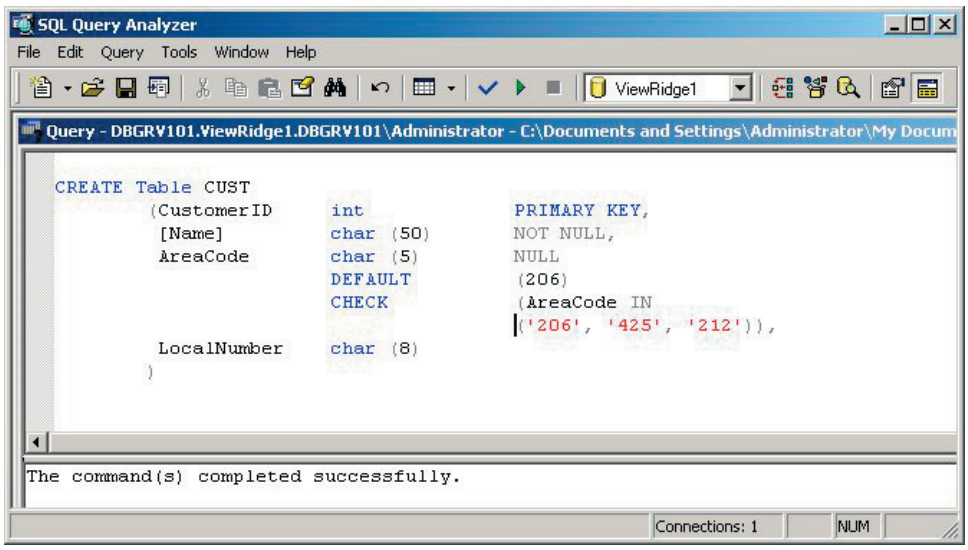
➤ FIGURA 13-3

Vista de las tablas en una base de datos



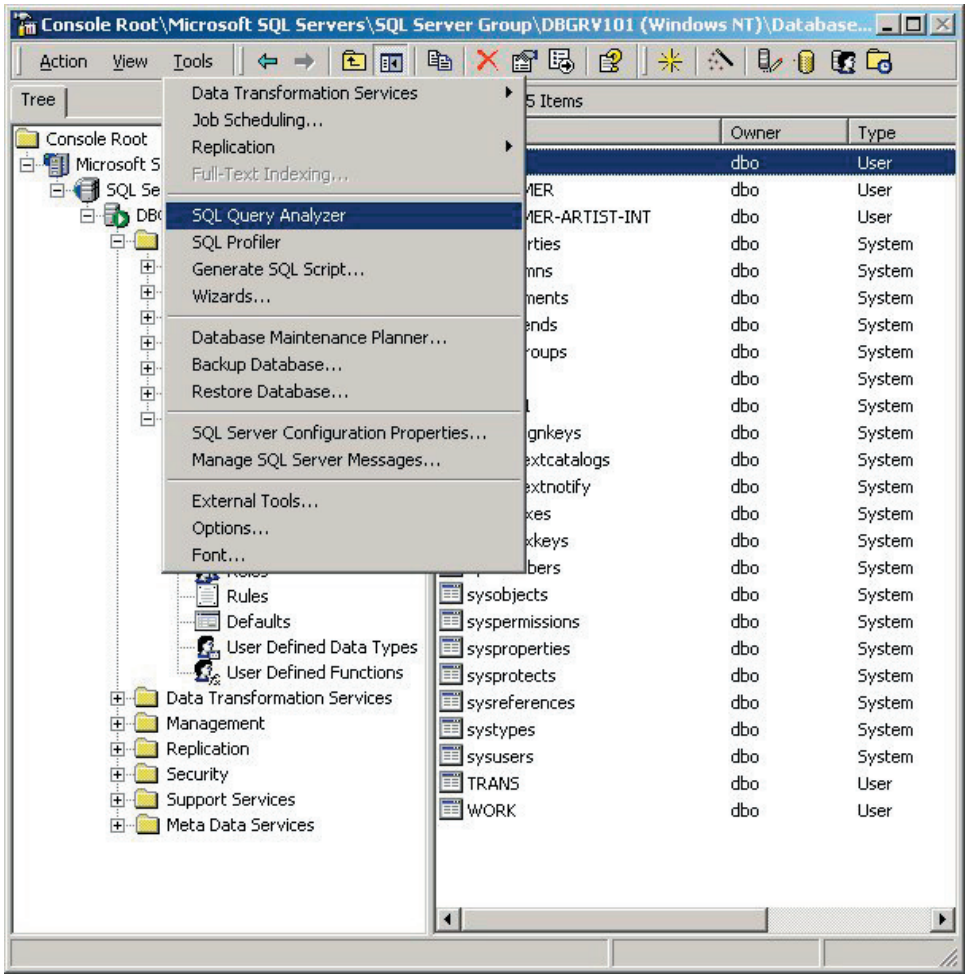
➤ FIGURA 13-4

Creación de una tabla con SQL en el Analizador de consultas SQL



➤ FIGURA 13-5

Llamado del Analizador de consultas



En este ejemplo, el valor predeterminado de AreaCode se define como (206). Además, la restricción CHECK limita los valores de CódigodeÁrea para estos listados. Los valores de CódigodeÁrea pueden ser NULL. El NúmeroLocal se define como carácter (8) y debido a que no se hace ninguna indicación, NULL o NOT NULL usará la predeterminación NULL.

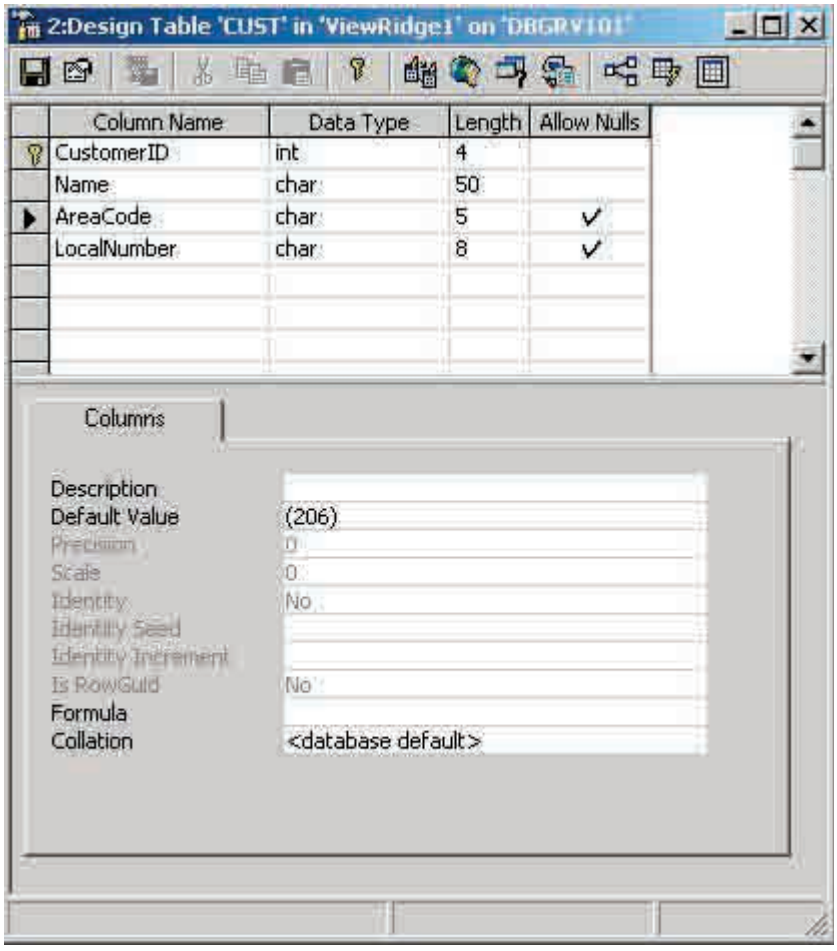
Hay varias formas de pasar esta instrucción CREATE a SQL Server. La más simple es usar el analizador de consultas. Para esto, haga clic en Tools (Herramientas) y seleccione SQL Query Analyzer (Analizador de consultas SQL), como se muestra en la figura 13-5. Teclee la instrucción CREATE Table dentro de la ventana y después haga clic en la marca de verificación. Si su instrucción tiene errores de sintaxis, se indicará en la parte inferior. Una vez que haya eliminado los errores de sintaxis, haga clic en el diamante de la derecha y se creará la tabla.

Para ver qué tiene la tabla, regrese a la ventana de Enterprise Manager; haga clic a la derecha de Tablas y seleccione Refresh (Regenerar). La nueva tabla debe aparecer en las listas a mano derecha. Haga clic a la derecha de la nueva tabla y seleccione Design Table (Diseñar tabla). Verá una imagen como la de la figura 13-6. Observe que el CódigodeÁrea predeterminado en realidad es (206) y que a AreaCode y a LocalNumber se les permite ser nulos. Además, la llave primaria (denotada por el símbolo llave) de la tabla es CustomerID.

CÓMO USAR LA VENTANA DE CREACIÓN DE UNA TABLA GRÁFICA. La segunda forma para crear una tabla es similar a la que se usó en Microsoft Access. Haga clic en el lado derecho en las tablas en Enterprise Manager y seleccione New Table. Aparecerá una forma en blanco en la que podrá escribir los nombres de las columnas y los tipos de datos. Para algunos tipos de datos puede colocar la longitud (los caracteres,

➤ FIGURA 13-6

Tabla de SQL Server creada por SQL en la figura 13-4



por ejemplo) pero para otras la longitud se determina mediante el Data Type (Tipo de datos) (por ejemplo, int, que significa entero).

En la figura 13-7, ArtistID (IDArtist) es una llave sustituta. Para hacer que SQL Server proporcione los valores para IDArtist, su Identity (Propiedad de identidad) se ha puesto como Yes (Sí). El valor de inicio se especifica mediante Identity Seed; la cantidad que se debe adicionar a la llave sustituta cuando se agrega un nuevo renglón se especifica mediante Identity Increment (Aumento de identidad). Aquí, ArtistID comenzará con 1 y aumentará de 1 en 1. Sólo una columna en una tabla puede ser una columna de identidad. SQL Server no tiene objetos de secuencia como los de Oracle.

Para hacer de esta columna la llave de la tabla, haga clic en cualquier lugar del renglón ArtistID para resaltarla y después haga clic en el símbolo llave en la ventana de diseño de la barra de menú. Para hacer una llave compuesta, resalte todas las columnas que comprendan la llave y después haga clic en el símbolo llave.

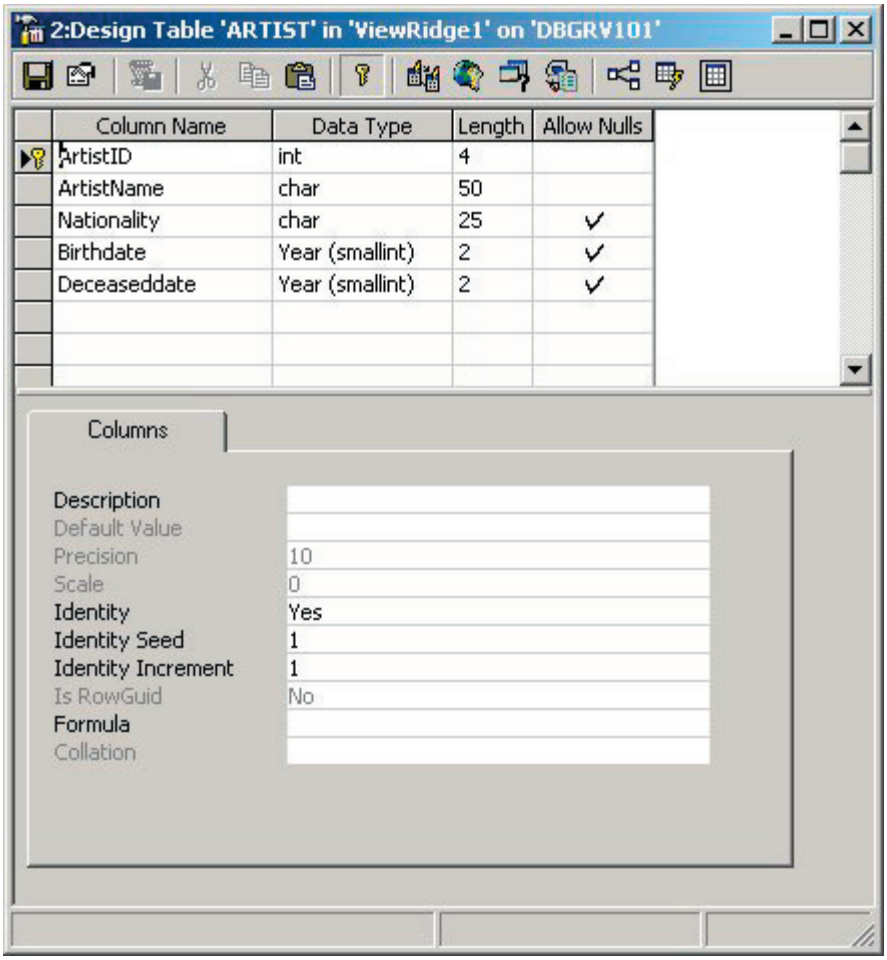
Los tipos de datos que proporciona el sistema para SQL Server 2000 se listan en la figura 13-8. Puede seleccionar cualquiera cuando defina una columna.

TIPOS DE DATOS DEFINIDOS POR EL USUARIO. SQL Server 2000 implementa los tipos de datos definidos por el usuario, lo cual se puede usar para representar los dominios. Por ejemplo, en la aplicación de View Ridge suponga que el dominio de FechadeNacimiento y FechadeFallecimiento no son datos verdaderos, pero que en su lugar hay años de cuatro dígitos.

Para implementar lo anterior con SQL Server 2000 se pueden establecer tipos de datos definidos por el usuario. Para hacerlo en Enterprise Manager (Administración de la empresa) haga clic en el lado derecho de los tipos de datos definidos por el usuario y seleccione tipos de datos definidos por un usuario nuevo. Aparecerá una caja de diálogo co-

➤ FIGURA 13-7

Creación de una tabla con la herramienta de diseño gráfico



mo la de la figura 13-9. Introduzca el nombre del nuevo tipo de datos en Name, la caja de texto y establezca el tipo de datos. Aquí *(Year),* se define y se le da el tipo *smallint.*

Una vez que ha definido el tipo de datos se muestra en la lista desplegable que aparece cuando hace clic en la columna DataTypes —como si fuera uno de los tipos de datos que proporciona el sistema—. En la figura 13-7 el tipo de datos Year se seleccionó para ambas columnas, Birthdate y Deceaseddate.

Los tipos de datos definidos por el usuario son aún más útiles cuando se combinan con reglas.

REGLAS. Las reglas son expresiones declarativas que limitan los valores de los datos. Su formato general es @variablename <rule expression> (@nombrevariable <expresión de la regla>). La figura 13-10 muestra la definición de la regla @year BETWEEN 1400 AND 2100. Puede usar cualquier predicado de expresión SQL (limítese a las expresiones que se encuentran en las cláusulas WHERE) como regla de expresión. El nombre de la variable no es importante. Puede definir una regla haciendo clic en el lado derecho en Rules debajo de Name en su base de datos y seleccionando New Rule.

Una expresión particularmente útil es @List IN (lista de valores). Por ejemplo, @State IN ('CA', 'OR', 'WA', 'AZ') limita una columna o los tipos de datos que define el usuario para esos valores.

La regla en la figura 13-10 está limitada al tipo de datos definido por el usuario (Year). Una vez que lo haya hecho, todas las columnas que usen el tipo de datos Year tendrán valores restringidos por esta regla.

Las reglas sirven para el mismo propósito que las cláusulas CHECK de las instrucciones CREATE y ALTER; puede usar cualquiera de las dos con SQL Server. Las cláusulas

➤ FIGURA 13-8

Tipos de datos comunes de SQL Server

Binario	**Binary**	Binario, Longitud de 0 a 8000 bytes.
Carácter	**Char**	Carácter, longitud de 0 a 8000 bytes.
Fecha	**Datetime**	Fecha 8 bytes. Intervalo de enero 1, 1753, a diciembre 31, 9999, con una exactitud de tres centésimas de segundo.
Decimal	**Decimal**	Decimal se puede establecer con precisión y a escala. Intervalo –10^38+1 hasta 10^38–1.
Flotante	**Float**	Número de punto flotante de 8-bytes. Intervalo de valores – 1.79E+308 hasta 1.79E+308.
Imagen	**Image**	Datos binarios de longitud variable. Longitud máxima 2,147,483,647 bytes.
Entero	**Int**	Entero de 4-bytes. Intervalo de valores de –2,147,483,648 hasta 2,147,483,647.
Monetario	**Money**	Monetario 8-bytes. Renglón de –922,337,203,685,477.5808 hasta +922,337,203,685,477.5807 con una exactitud de diez centésimas de unidad monetaria.
Numérico	**Numeric**	Igual que un decimal.
Real	**Real**	Punto de número flotante de 4-bytes. Gama de valores: –3.40E + 38 hasta 3.40E + 38.
FechaCorta	**Smalldatetime**	Fecha 4-bytes. Intervalo de enero 1, 1900, a junio 6, 2079, con una exactitud de un minuto.
EnteroCorto	**Smallint**	Entero de 2-bytes. Rango de –32,768 hasta 32,767.
MonetarioCorto	**Smallmoney**	Monetario de 4-bytes. Va de 214,748.3648 hasta +214,748.3647, con una exactitud de diez centésimas de unidad monetaria.
Texto	**Text**	Texto de longitud variable, longitud máxima 2,147,483,647 caracteres.
EnteroPequeño	**Tinyint**	Entero de 1-byte. Rango de valores de 0 hasta 255.
CaracVariable	**Varchar**	Carácter de longitud variable, longitud de 0 a 8000 bytes.

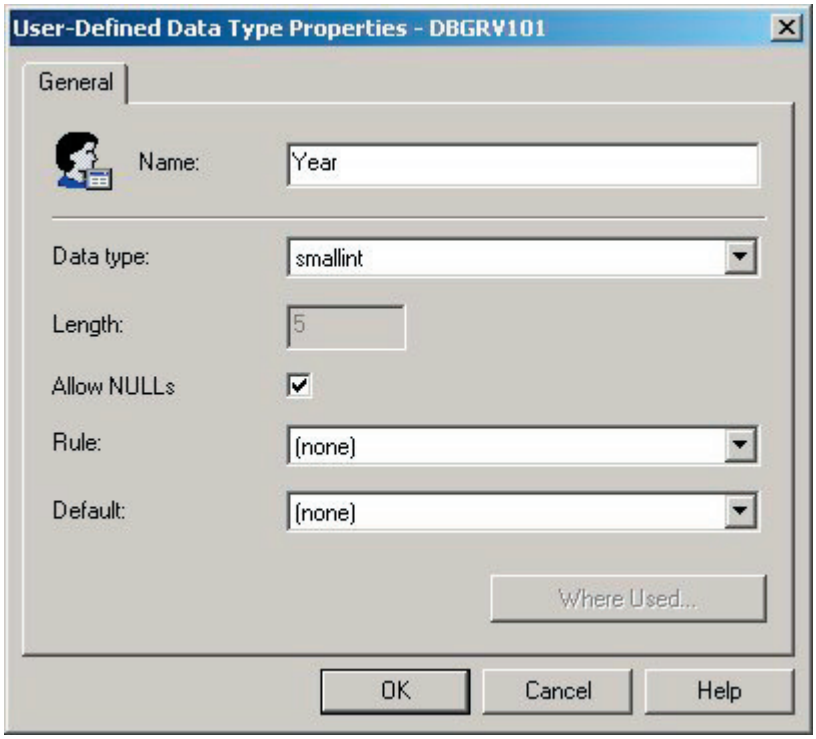
CHECK son parte de la norma ANSI SQL y, por lo tanto, son los preferidos de algunas organizaciones. Por otro lado, las reglas se pueden definir gráficamente.

CAMBIO DE LA ESTRUCTURA DE LA TABLA. La estructura de la tabla se puede cambiar de dos formas. Una es usar la instrucción ALTER SQL como lo hicimos en el capítulo anterior. La otra es hacer cambios usando la forma de diseño de la tabla (como en las figuras 13-6 y 13-7). La figura 13-11 muestra varias instrucciones ALTER para cambiar la estructura de la tabla CUST. La primera extiende la longitud de la columna Name de 50 a 100.

La segunda ALTER define una columna nueva, CustomerSince, usando el tipo de datos definido por el usuario al escribir *year*. Las dos instrucciones siguientes muestran una segunda manera de extender la longitud de una columna. Aquí, primero se elimina la columna y después se agrega con la nueva longitud. Sin embargo, a diferencia de la primera instrucción ALTER, se perderán datos porque la columna se elimina físicamente antes de ser creada nuevamente.

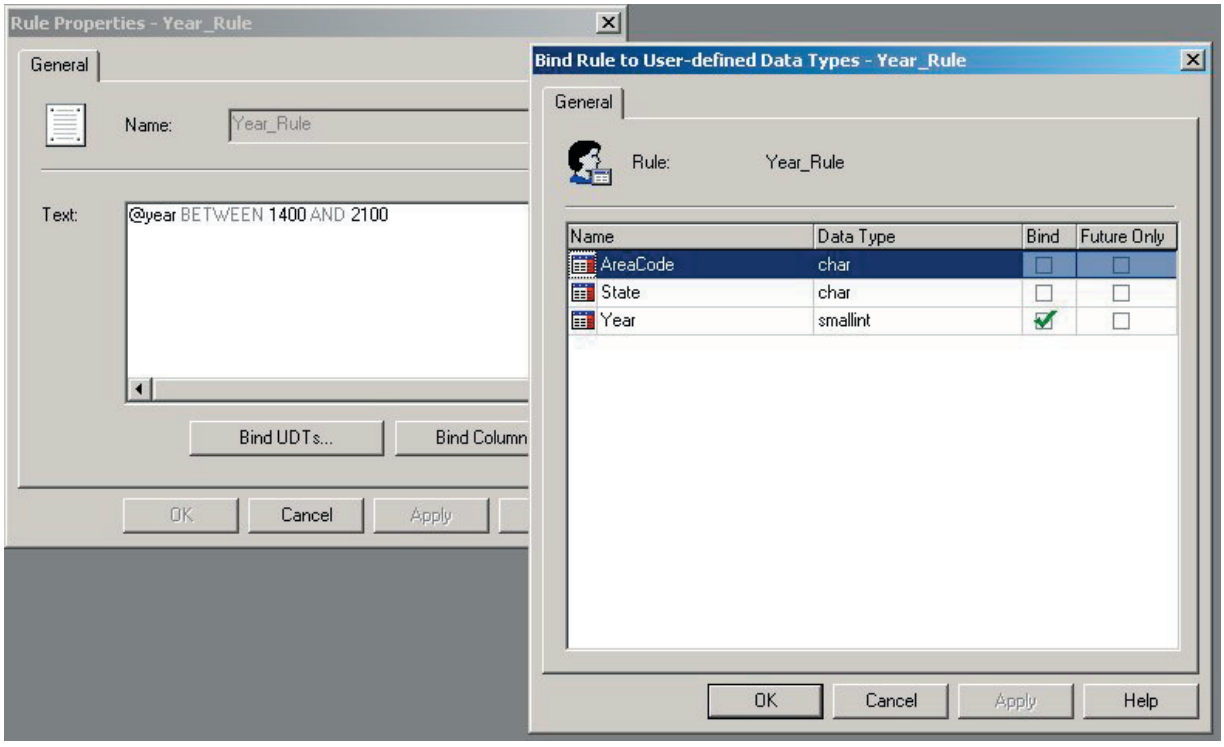
➤ FIGURA 13-9

Establecimiento de un tipo de datos definidos por el usuario



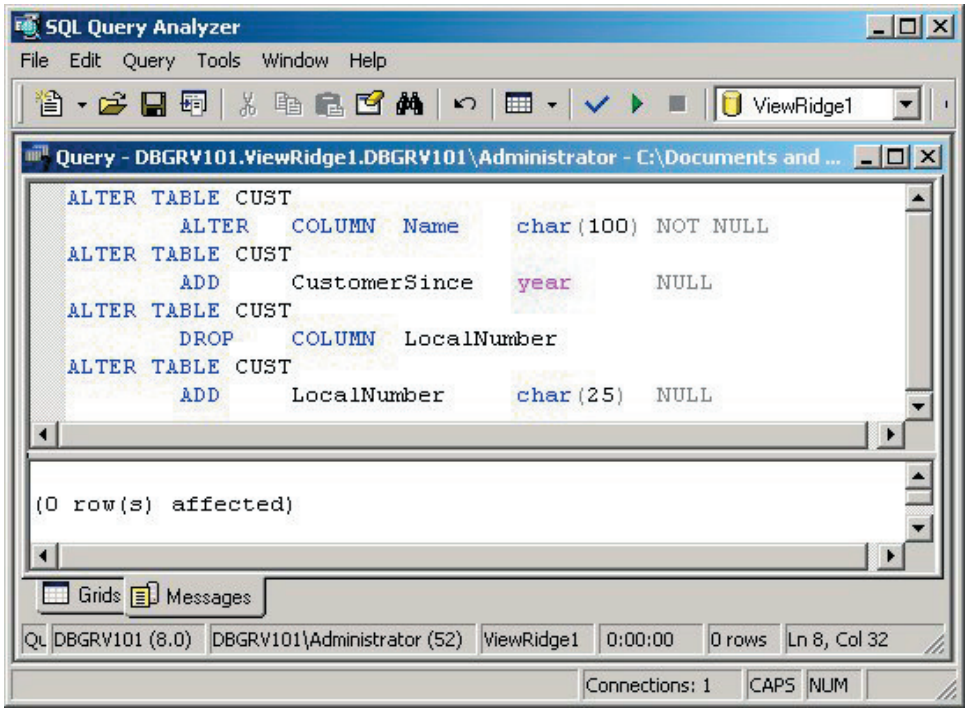
➤ FIGURA 13-10

Creación de una regla y su asociación con un tipo de datos definido por el usuario



➤ FIGURA 13-11

Cambio de la estructura de la tabla con la instrucción ALTER



Una segunda manera de hacer estos cambios es usar la ventana de la tabla de diseño gráfico, como se muestra en la figura 13-12. Sólo abra la ventana y haga los cambios que quiera a las columnas, tipos de datos y longitudes. SQL Server le hará saber si intentó hacer cambios sin apoyo.

Con SQL Server se puede reducir la longitud de char, varchar y tipos similares, incluso si la columna tiene datos. Sin embargo, hacerlo ocasionará que los datos se trunquen. En el proceso se pueden crear valores duplicados que violan las restricciones de unicidad. Esto ocasionará que el cambio falle y se puede crear un desorden. Por lo tanto, realice estas reducciones de tamaño con mucho cuidado.

Los comentarios son similares en cuanto al cambio de tipo de datos de una columna; puede hacerlo incluso si la columna tiene datos, pero con mucho cuidado porque puede perderlos. El cambio de un campo de texto numérico como Areacode a entero no es problema. Sin embargo, el cambio de un campo de texto tal como Name a entero, dará como resultado una pérdida de datos.

CREACIÓN DE LAS TABLAS DE EJEMPLO. Si está llevando a cabo las construcciones de este capítulo en su copia de SQL Server, ahora deberá crear las cinco tablas que se muestran en la figura 10-3(d). Puede eliminar la tabla CUST, ya no la usaremos. (Haga clic en el lado derecho del nombre de la tabla en la lista de tablas, y haga clic en eliminar.)

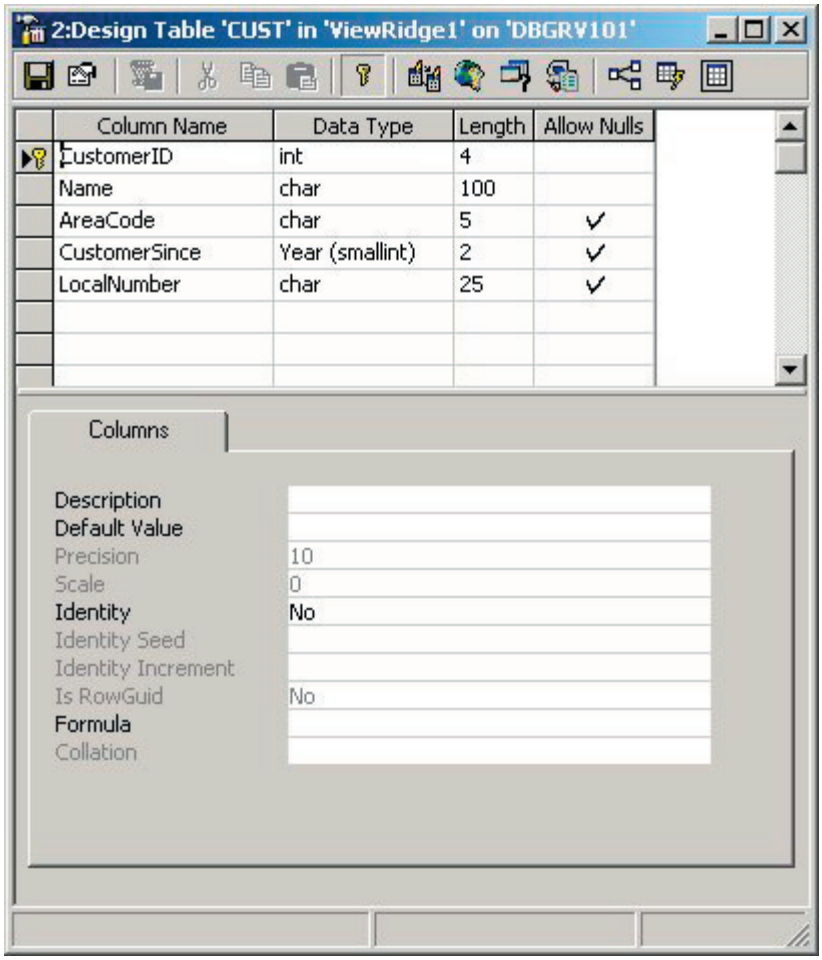
Nombre la tabla TRANSACTION como TRANS en lugar de TRANSACTION. El nombre *TRANSACTION* es tan especial para SQL Server que ningún procedimiento almacenado trabajará en una tabla llamada TRANSACTION, aun cuando lo haya encerrado entre paréntesis. Se complicará menos la existencia si lo nombra TRANS.

Use las llaves sustitutas para cada una de estas tablas. Como en el capítulo anterior, establezca los orígenes de identidad de forma diferente para cada tabla. Use un selector de identidad de 1 para ARTIST, 1000 para CUSTOMER, 500 para WORK y 100 para TRANS. Deje el incremento en 1 para cada uno de éstos. Establezca las llaves de tabla para resaltar la(s) columna(s) y haga clic en el símbolo llave.

Al igual que con Oracle, si está creando las tablas con las instrucciones SQL, cuando cree CUSTOMER_ARTIST_INT, no dé a cualquier columna la propiedad LLAVE PRI-

➤ FIGURA 13-12

Resultado de las instrucciones ALTER en la figura 13-11



MARIA. En lugar de eso, después de la instrucción CREATE TABLE cree una llave primaria con restricción

CONSTRAINT pk_restricción LLAVE PRIMARIA (CustomerID, ArtistID)

Esta instrucción establecerá una llave compuesta de (CustomerID, ArtistID) con el nombre pk_constraint. Por supuesto, puede usar el nombre de restricción (constraint) que usted quiera.

Definición de relaciones

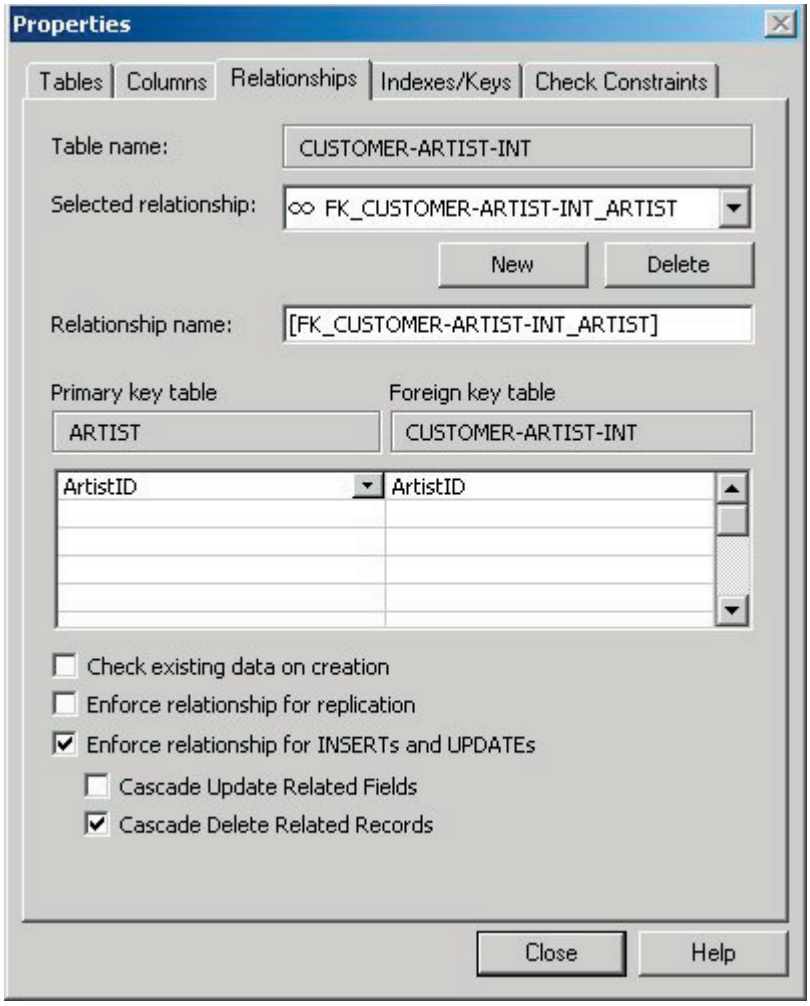
Como con otras estructuras, podemos definir las relaciones mediante la definición de llaves externas en las instrucciones ALTER TABLE, o de la creación de relaciones en un diagrama de base de datos. Lo hicimos en el capítulo anterior, así que usaremos un diagrama de base de datos.

Haga clic derecho en Diagrams (Diagramas) y elija un Nuevo diagrama de base de datos. Un asistente le conducirá desde el inicio durante todo el proceso de agregar tablas a su diagrama. Agregue al diagrama las cinco tablas de la figura 10-3(d).

Ahora, para crear una relación arrastre una llave primaria desde una tabla a una llave externa en la tabla que quiere relacionar. Por ejemplo, para crear la relación entre ARTIST y CUSTOMER_ARTIST_INT, arrastre ArtistID en ARTIST y suéltelo en la parte superior de ArtistID en CUSTOMER_ARTIST_INT. Esto creará la relación y la caja de diálogo de la figura 13-13 se desplegará. Observe que SQL Server proporciona un nombre preestablecido para la relación y muestra las columnas de la llave principal y externa. No ne-

➤ FIGURA 13-13

Establecimiento de las propiedades de la relación



cesita hacer nada de esto, pero debe pensar con cuidado acerca del control de la integridad referencial de las cajas que siguen.

CUMPLIMIENTO DE LA INTEGRIDAD REFERENCIAL. Recuerde que las restricciones de la integridad referencial se refieren a la presencia de valores de la llave en la tabla padre y en las tablas hijas. En particular, estas restricciones aseguran que un valor llave en una tabla hija exista en una tabla padre relacionada. Por ejemplo, ArtistID (ArtistaID) en CUSTOMER_ARTIST_INT (CLIENTE_ARTISTA_INT) debe existir en ArtistID (ArtistaID) en ARTIST (ARTISTA).

La primera caja de control en la figura 13-13 se refiere a la integridad referencial verificando los datos que existen. Puesto que estamos creando una nueva base de datos, esta caja no tiene importancia para nosotros. Sin embargo, si estuviésemos creando una relación de los datos existentes, tendríamos que verificar si queremos que SQL Server busque violaciones de integridad de los datos que ya existen. Si no, SQL Server sólo aplicará la restricción a los nuevos datos y a los que se modificaron después de esto.

Quizás no desee que SQL Server verifique la integridad de los datos si la base de datos es grande, debido al tiempo que esto llevaría. Además, probablemente no quiera verificar esta caja si no sabe cómo arreglar las violaciones que encuentre. Quizá no sepa cómo hacer una relación para los datos existentes.

La siguiente caja de control es Enforce, relationship for replication, la cual tiene que ver con procesamiento distribuido; por el momento no trataremos este tema.

➤ FIGURA 13-14

Resumen del cumplimiento de restricciones de integridad referencial en la base de datos de View Ridge

Relación	Relación impuesta	Eliminación en cascada
CLIENTE a TRANS	Sí	No
CLIENTE a CLIENTE-ARTISTA-INT	Sí	Sí
ARTISTA a TRABAJO	Sí	No
TRABAJO a TRANS	Sí	No
ARTISTA a CLIENTE-ARTISTA-INT	Sí	Sí

La tercera caja de verificación —Enforce relationship para INSERTs y UPDATEs— es importante para nuestros fines. Si esta caja no se verifica, SQL Server ignorará la relación durante las actualizaciones y las eliminaciones. Así que si la caja no se verifica, podríamos eliminar un ARTIST (ARTISTA) y posiblemente dejar los renglones CUSTOMER_ARTIST_INT con valores inválidos de ArtistID. Si no desea que suceda esto, será mejor que verifique esta caja.

Ahora, si verificamos cualquiera de las dos cajas siguientes, SQL Server simplemente anulará cualquier actualización o eliminación que pueda violar esta restricción de integridad. Usaremos esto para nuestro provecho durante un momento.

La relación que estamos creando se refiere a una tabla de intersección. Si se elimina un renglón artista, queremos que los renglones correspondientes en esta tabla intersección se eliminen también. En consecuencia, verificamos la caja de Cascade Delete Related Records. Para hacerlo, instruimos a Records SQL Server para que elimine todos los renglones de la tabla intersección relacionados cuando suprimimos un renglón en ARTIST. Esto tiene el mismo efecto que la cláusula SQL ON DELETE CASCADE (ELIMINACIÓN EN CASCADA) que se mostró en el capítulo 12.

Este diseño usa llaves sustitutas y, por lo tanto, no cambia los valores llaves que son posibles. Así, no necesitamos preocuparnos por Cascade Update Related Records. No es posible ninguna actualización a los valores llave.

Ahora podemos definir la relación entre CUSTOMER y CUSTOMER_ARTIST_INT (CLIENTE y CLIENTE_ARTISTA_INT) de manera similar. Verifique la relación impuesta y las cajas de eliminación en cascada.

Considere las siguientes relaciones entre ARTIST y WORK (TRABAJO), entre WORK (TRABAJO) y TRANS y entre CUSTOMER y TRANS. No queremos eliminar en cascada cualquiera de estas relaciones, sino anular cualquier eliminación que pueda causar una violación a la integridad referencial. Si, por ejemplo, alguien intentara eliminar un renglón en WORK (TRABAJO) que tiene una hija TRANS, querríamos evitarlo. Por lo anterior, en las propiedades de la relación de la caja de diálogo, revise la caja de verificación de relación impuesta, pero no examine la caja de verificación de eliminación en cascada. Las relaciones de esta base de datos se resumen en la figura 13-14.

Con las relaciones establecidas de esta manera la única forma de eliminar, por así decir, un renglón ARTIST es eliminar primero todos los renglones TRANS para cualquiera de los trabajos de ese artista, después todos los renglones de WOK para cualquiera de los trabajos de dicho artista, y a continuación eliminar el renglón ARTIST.

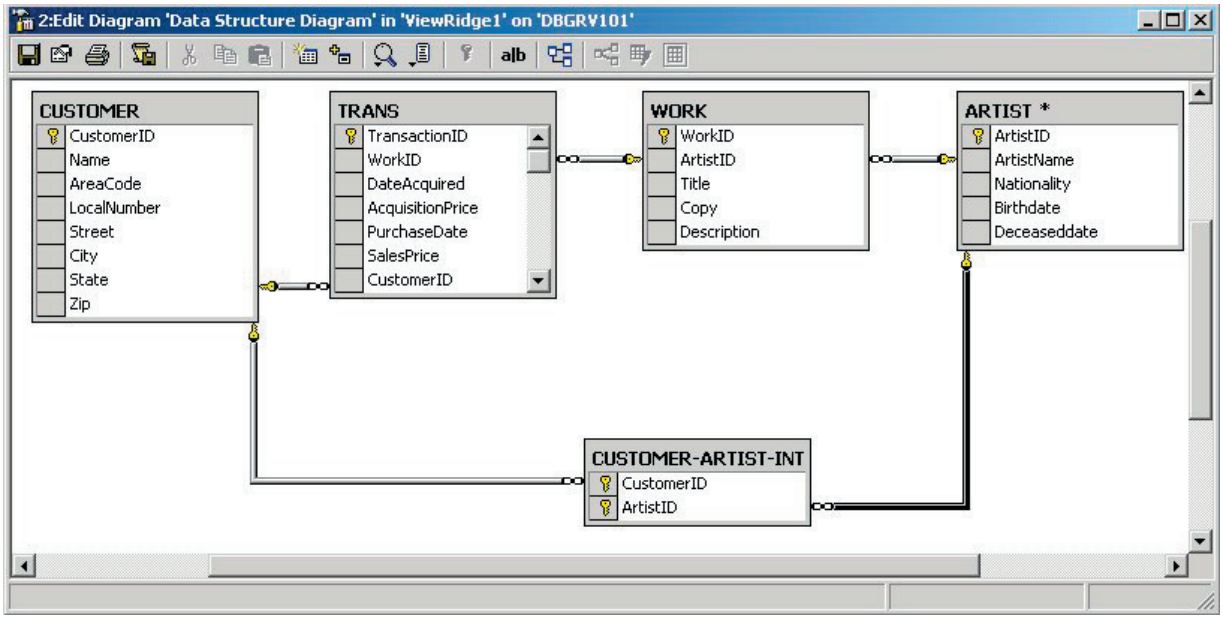
La relación del diagrama aparece en la figura 13-15. Puede determinar las propiedades de cualquier relación haciendo clic en el lado derecho en la línea que lo representa.

Vistas

Si aún no lo ha hecho, lea el análisis de las vistas del capítulo 12, páginas 341-342. Los comentarios generales que se hacen también se relacionan con las vistas expuestas en SQL Server.

➤ FIGURA 13-15

Diagrama de la base de datos para el esquema View Ridge

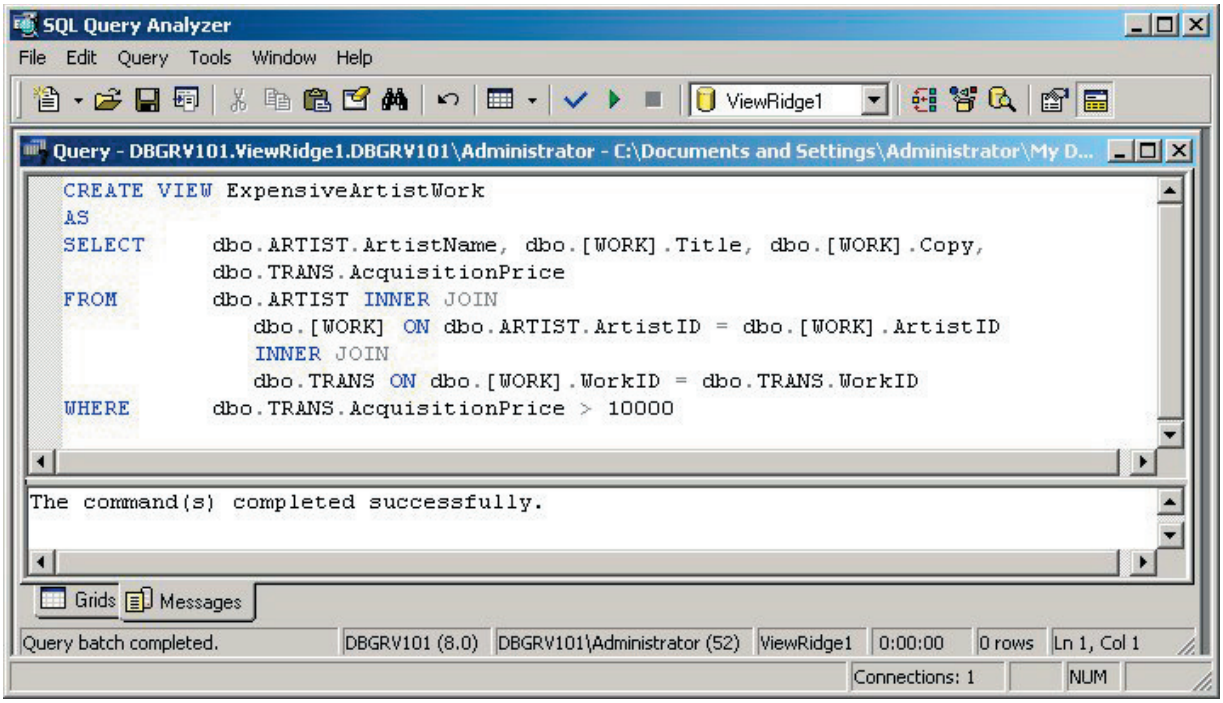


CREACIÓN DE VISTAS. Las vistas se pueden definir usando las instrucciones SQL, o los facilidades de diseño gráfico de SQL Server. Las figuras 13-16 y 13-17 muestran ambos planteamientos.

La vista de estas figuras se denomina ExpensiveArtistWork (ArtistaTrabajoCostoso). Junta los datos ARTIST, WORK y TRANS (ARTISTA, TRABAJO y TRANS) para todos

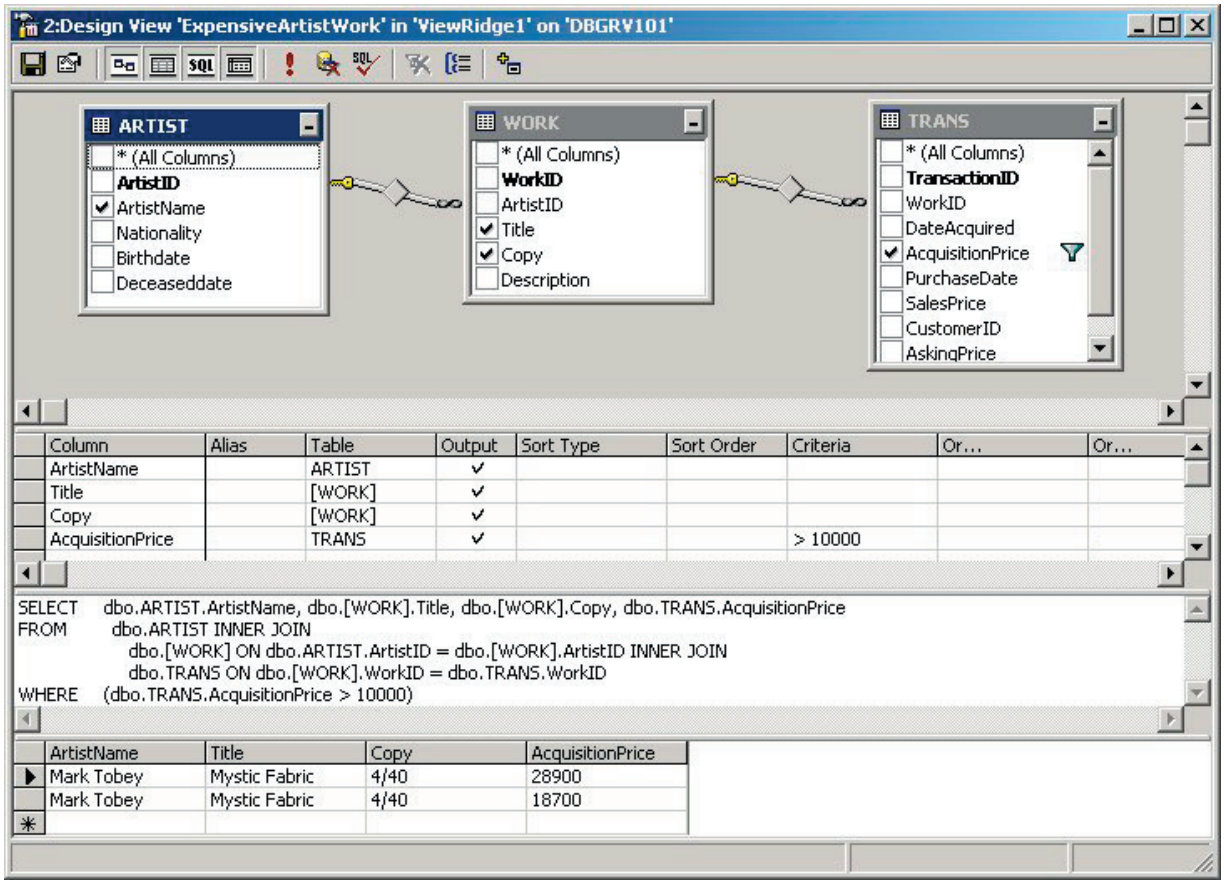
➤ FIGURA 13-16

Creación de una vista con instrucciones SQL



➤ FIGURA 13-17

Creación de una vista con la herramienta gráfica



los artistas, y después selecciona los renglones para los cuales el PreciodeAdquisición es mayor a 10000. Una vez que se ha creado esta vista, un usuario puede introducir el siguiente enunciado SQL para obtener los datos:

```
SELECT    *
FROM      ExpensiveArtistView
```

El usuario no necesitará realizar los join en las tablas implícitas, ni siquiera sabrá que este join se ha realizado.

A propósito, observe en la figura 13-16 que cada nombre de la tabla está calificado con el término *dbo*. Esto obedece a la siguiente razón: de manera predeterminada, cada nombre de tabla tiene un nombre de usuario anexo; si el nombre no aparece en una tabla de referencia, SQL Server anexa el nombre del usuario. Así, si el usuario John se inscribe en SQL Server y se remite a la tabla CUST, SQL Server buscará el nombre de la tabla John.CUST. Sin embargo, si el usuario John se inscribe en SQL Server y se remite a la tabla *dbo*.CUST, SQL Server buscará la tabla llamada dbo.CUST, no John.CUST.

La instrucción CREATE VIEW en la figura 13-16 anexa dbo a cada tabla de referencia para que otros usuarios aparte de dbo puedan usar esta instrucción. Cuando el usuario dbo ejecuta esta instrucción, de cualquier manera, todas las referencias dbo son superfluas debido a que las ha agregado SQL Server. Pero cuando el usuario John ejecuta esta instrucción, la presencia del prefijo dbo asegura que SQL Server se remita a las asignadas a dbo y no a las de John.

Puesto que no se ha agregado ningún nombre a ExpensiveArtistWork, cuando el dbo ejecute esta consulta creará un vista llamada dbo. ExpensiveArtistWork. Cuando John ejecute esta consulta, creará un vista llamada John.ExpensiveArtistWork. Las vistas serán diferentes. Tendrán los mismos datos puesto que se basan en las mismas tablas, pero habrá dos vistas diferentes con dos nombres diferentes.

Para crear una vista usando la herramienta gráfica, haga clic en el lado derecho en Views en el Enterprise Manager y después en Nueva Vista. Haga clic en el lado derecho en la parte superior derecha, en donde no hay nada y seleccione Add Table. Para crear la vista de la figura 13-17, agregue las tablas ARTIST (ARTISTA), WORK (TRABAJO), y TRANS. Verifique ArtistName, Title, Copy y AcquisitionPrice. En la pantallita del tercer lugar, agregue el criterio > 10000. Cuando lo haga, SQL Server construirá SQL en el cuarto lugar. Haga un clic en el signo de exclamación en la barra del menú y SQL Server poblará la vista con sus datos, como se muestra en el último lugar de la figura 13-17.

VISTAS ACTUALIZABLES. Algunas vistas se pueden usar para actualizar datos y otras no. El hecho de especificar las condiciones que hacen que una vista sea actualizable es complicado. En general, cualquier vista que se basa en una sola tabla se puede usar para actualizar y eliminar operaciones mientras no contenga funciones agregadas (como COUNT o MAX) o columnas derivadas en la cláusula SELECT. Esta vista también se puede usar para insertar datos, si contiene todas las columnas NOT NULL. De otra forma, no puede aceptar inserciones.

Las vistas que se forman sobre los join son más complicadas. Con SQL Server ninguna vista que tenga más de una tabla en sus cláusulas FROM puede aceptar eliminaciones. Las vistas de tablas múltiples pueden aceptar inserciones y actualizaciones mientras que éstas se hagan a una sola tabla dentro de la vista, y suponiendo nuevamente que la vista no tenga funciones agregadas o columnas derivadas en la cláusula SELECT.

Finalmente, las vistas no actualizables se pueden convertir en actualizables si quien programa crea un disparador (trigger) INSTEAD OF para la operación de actualización. Este disparador es un procedimiento TRANSACT-SQL que realiza la operación de inserción, actualización, o eliminación en SQL Server. Analizaremos esto más adelante, cuando estudiemos los disparadores.

Índices

Como se estableció, los índices son estructuras de datos especiales que se crean para mejorar el funcionamiento de la base de datos (véase el Apéndice A). SQL Server automáticamente crea un índice sobre todas las llaves primarias y sustitutas. El programador también puede hacer que SQL Server cree un índice sobre otras columnas que con frecuencia se usan en las cláusulas WHERE, o en las columnas que se usan para el ordenamiento de datos cuando se procesa secuencialmente una tabla de consultas y reportes.

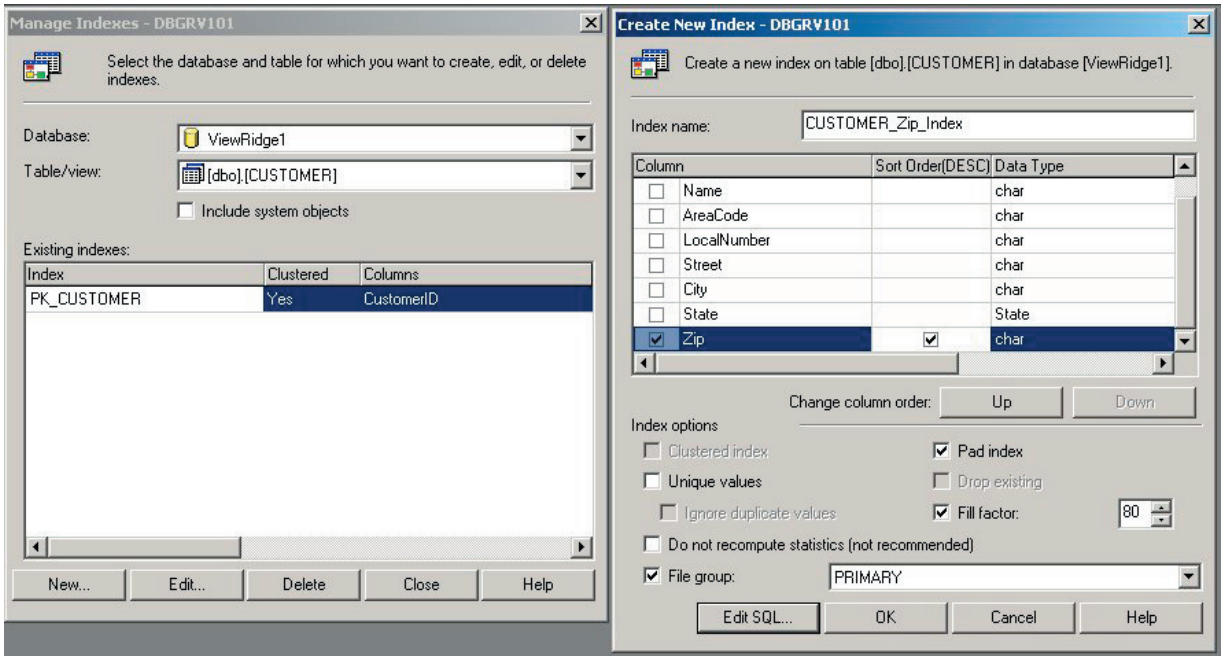
Para crear un índice, haga clic en el lado derecho en la tabla que tiene la columna que quiere indexar, haga clic en Todas las tareas y después haga clic en Administración de índices. Verá la caja de diálogo que se muestra en la parte izquierda de la figura 13-18. Haga clic en New (Nueva) y verá el diálogo de la caja a la derecha de esta figura. El programador está creando un índice sobre la columna CódigoPostal (Zip) de la tabla CUSTOMER. El índice, llamado CUSTOMER_Zip_Index, es reforzado, completado al 80%, y asignado al grupo de Archivos PRIMARY. El reforzamiento deja un espacio abierto para inserciones en todos los niveles del índice, excepto en la parte inferior. Llene las cantidades de los espacios vacíos en la parte inferior izquierda del índice. Véase el Apéndice A y la documentación del SQL Server para mayor información con respecto a estas opciones.

Haga clic en Edit SQL en esta caja de diálogo y verá la caja de diálogo de la figura 13-19, que muestra las instrucciones SQL que se pueden introducir mediante el analizador SQL para crear el índice.

SQL Server maneja dos clases de índices: agrupados y no agrupados. Con un índice agrupado, los datos se almacenan en la parte inferior del nivel del índice y en el mismo orden que el del índice. Con un índice no agrupado, el nivel de la parte inferior de

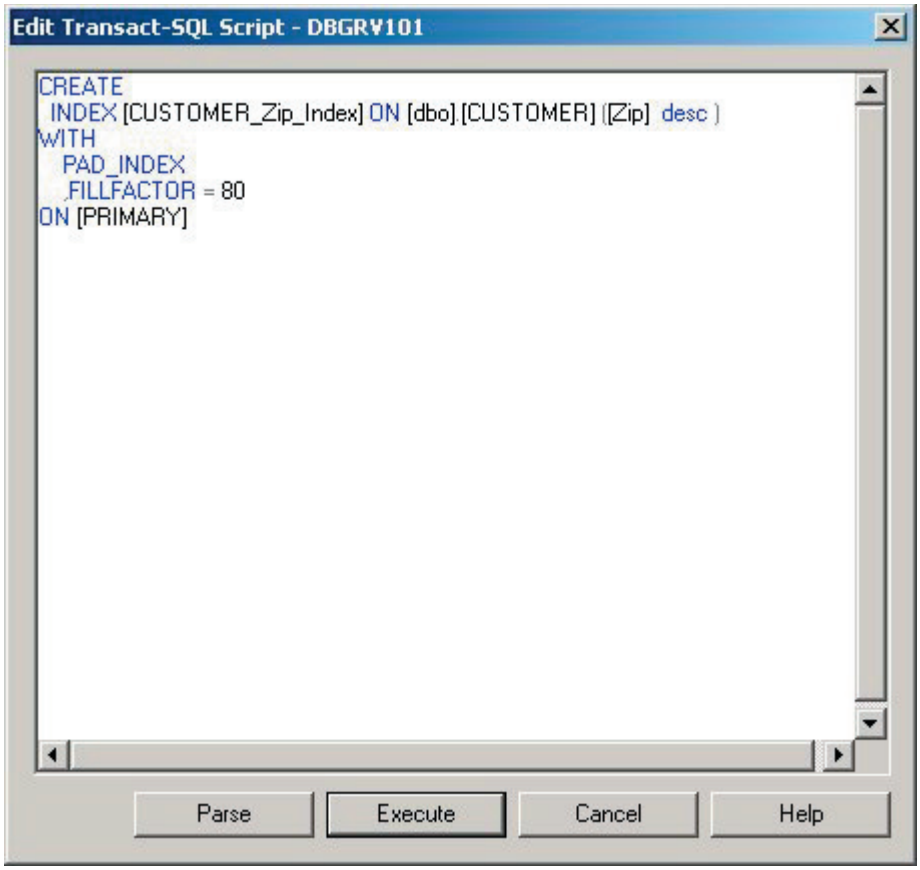
➤ FIGURA 13-18

Creación de un índice con la herramienta gráfica



➤ FIGURA 13-19

SQL para la creación de un índice



un índice no contiene datos, sino apuntadores a los datos. Debido a que los renglones se pueden almacenar sólo con un orden físico a la vez, únicamente se permite un índice agrupado por tabla. Los índices agrupados son más rápidos para la recuperación que los no agrupados. Normalmente también son más rápidos para la actualización, pero no si hay muchas actualizaciones en el mismo sitio en medio de la relación. Vea la documentación de SQL Server para tener mayores datos.

➤ LÓGICA DE LA APLICACIÓN

Hay muchas formas de procesar una base de datos de SQL Server. Ya habrá visto el uso del Analizador de Consultas (Query Analyzer) SQL para transmitir instrucciones SQL. Hemos usado ese dispositivo para crear estructuras de bases de datos, pero se puede emplear para procesar cualquier tipo de instrucción SQL. Otros medios para procesar la base de datos son escribir los programas de aplicación en lenguajes como Visual Basic, Java, C#, o C++ y usar los Objetos de Datos Activos, ODBS, o JDBC para procesar estructuras de bases de datos y su contenido. En el capítulo 15 mostraremos ejemplos de uso para el procesamiento de Internet.

Un tercer medio de tener acceso a la base de datos es escribir los procedimientos en TRANS ACT-SQL: el cual es un lenguaje que agrega elementos de programación tales como parámetros, variables, IF, WHILE y así sucesivamente, para las capacidades básicas de SQL. TRANSACT_SQL tiene la misma función para SQL Server que el PL/SQL para Oracle.

Puede escribir programas TRANSACT-SQL y enviarlos al DBMS a través del Analizador de Consultas SQL, vía procedimientos almacenados y a través de disparadores. Consideraremos cada uno de estos en esta sección. También puede invocar TRANSACT/SQL en los programas script para Web, como se verá en el capítulo 15.

CONSULTAS ALMACENADAS

Suponga que cada lunes en la mañana el administrador de View Ridge quiere obtener una lista de todos los trabajos que están a la venta. Una forma de hacerlo es escribir la consulta SQL que se muestra en la figura 13-20. Esta consulta lista el ArtistName, Title and Copy (NombredeArtista, Título y Copia) de cada trabajo para el cual el CustomerID en TRANS es nulo. Estos renglones representan los trabajos no vendidos.

Para evitar escribir esta consulta cada semana, el administrador puede escribirla una vez y guardarla en un archivo. Entonces, para ejecutarla, comenzará con el Analizador de Consultas SQL, abrirá el archivo consulta y la ejecutará.

Sin embargo, para hacer lo anterior se requiere que los usuarios tengan acceso al Analizador de Consulta SQL, un programa diferente, especialmente si están procesando en otra computadora separada de la que administra la base de datos. También somete al usuario a una complejidad considerable. En general, se puede escribir un programa de aplicación para que ejecute esta consulta y reporte los resultados al usuario en una forma que le resulte más amigable.

No obtante, este estilo de procesamiento con frecuencia lo usan los administradores de bases de datos. Desarrollan procedimientos en TRANSACT-SQL que realizan muchas tareas de administración de la base de datos; estos procedimientos se guardan en archivos y se usan posteriormente cuando se requieren.

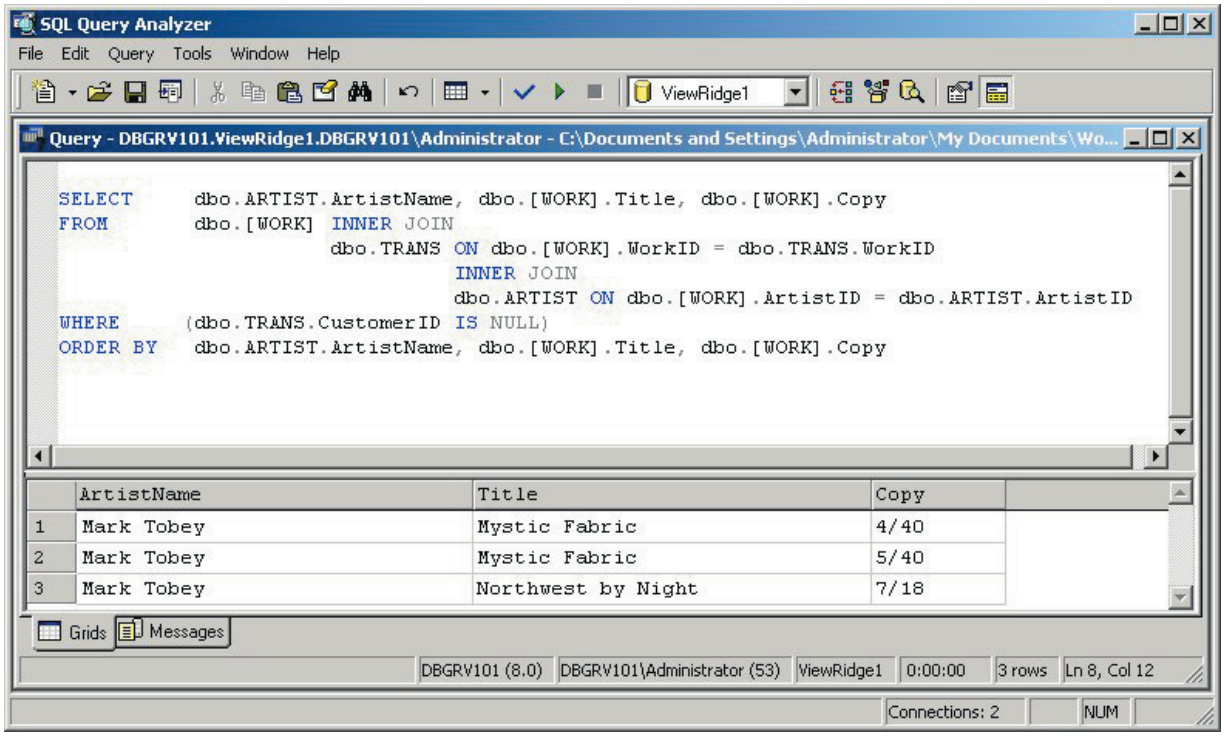
PROCEDIMIENTOS ALMACENADOS

Otro enfoque para el procesamiento de una base de datos de SQL Server se refiere a crear procedimientos TRANSACT-SQL y almacenarlos en cualquiera de las computadoras de los usuarios o en la base de datos. Estos procedimientos almacenados se pueden invocar por medio de su nombre y pueden pasar como valores de parámetros. Por ejemplo, la instrucción

```
InsertARTIST@ArtistName = `Matisse´
```

➤ FIGURA 13-20

Un ejemplo de consulta almacenada



invoca un procedimiento almacenado que se denomina InsertARTIST y le transmite el parámetro llamado @ArtistName con el valor "Matisse". Estos procedimientos pueden tener una lógica complicada, pueden procesar potencialmente muchas bases de datos, invocar otros procedimientos y funciones almacenados, etcétera.

Con el advenimiento de Internet cada vez es más común conservar los procedimientos almacenados en la base de datos antes de que se distribuyan a las computadoras de los usuarios. Entonces, el Servidor Web y el código del explorador invocan los procedimientos y pasan los parámetros mediante la conexión establecida usando los Datos de Objetos Activos o COM sobre HTTP. Ilustraremos este uso en el capítulo 15.

Sin embargo, aquí escribiremos los procedimientos almacenados dentro de SQL Server y después los probaremos usando el Analizador de Consultas SQL. Hacemos esto para ejemplificarlo de manera sencilla. En la realidad los procedimientos almacenados se invocarán mediante un programa de aplicación sobre una red de Internet, y no mediante el analizador de consultas SQL.

EL PROCEDIMIENTO ALMACENADO Customer_Insert (Cliente_Insertar).

La figura 13-21 ilustra un procedimiento almacenado que guarda datos de un cliente nuevo y lo conecta con todos los artistas que tienen una nacionalidad en particular. (Ésta es la misma lógica que se muestra para Cliente_Insertar en el capítulo anterior.) Se introducen cuatro parámetros al procedimiento: @NewName, @NewAreaCode, @NewPhone y @Nationality. Como se puede ver, los parámetros y las variables en TRANSACT-SQL están precedidas por el signo @. Los primeros tres parámetros son los datos del cliente nuevo y el cuarto es la nacionalidad del artista, en la cual está interesado el nuevo cliente.

La primera tarea que realiza este procedimiento almacenado es determinar si ya existe el cliente. Si el conteo de la primera instrucción SELECT es mayor que 0, entonces ya existe un renglón para ese cliente. En este caso, no se ha hecho nada y el procedimiento almacenado imprime un mensaje de error y sale. Por cierto, ese mensaje de error, estará

➤ FIGURA 13-21

Procedimiento Almacenado Cliente_Insertar

```
CREATE PROCEDURE Customer_Insert
        @NewName            char(50)
        @NewAreaCode        char (5)
        @NewPhone           char (8)
        @Nationality        char(25)
AS

DECLARE @Count      as smallint
DECLARE @Aid        as int
DECLARE @Cid        as int

/* Check to see if customer already exists */
SELECT @Count = Count (*)
FROM dbo.CUSTOMER
WHERE [Name]=@NewName AND AreaCode=@NewAreaCode AND LocalNumber=@NewPhone

IF @Count > 0
        BEGIN
                PRINT 'Customer Already Exists – No Action Taken'
                RETURN
        END

/* Add new Customer data */
INSERT INTO   dbo.CUSTOMER
              ([Name], AreaCode, LocalNumber)
              VALUES
              (@NewName, @NewAreaCode, @NewPhone)

/* Get new surrogate key value */
Select @Cid = CustomerID
FROM dbo.CUSTOMER
WHERE [Name]=@NewName AND AreaCode=@NewAreaCode AND LocalNumber=@NewPhone

/* Now create intersection record for each appropriate artist */
DECLARE Artist_Cursor CURSOR FOR
SELECT ArtistID
FROM dbo.ARTIST
WHERE Nationality=@Nationality

/* process each Artist of specified nationality */
OPEN Artist_Cursor
FETCH NEXT FROM Artist_Cursor INTO @Aid
WHILE @@FETCH_STATUS = 0
BEGIN
        INSERT INTO dbo.[CUSTOMER-ARTIST-INT]
                        (ArtistID, CustomerID)
                VALUES (@Aid, @Cid)
        FETCH NEXT FROM Artist_Cursor INTO @Aid
END

CLOSE Artist_Cursor
DEALLOCATE Artist_Cursor
GO
```

visible en el Analizador de Consulta, pero, en general, no estará visible para la aplicación de programas que invoquen este procedimiento. En lugar de eso, se necesitará usar un parámetro u otro dispositivo para regresar el mensaje de error al usuario mediante el programa de aplicación. Consideraremos este problema en los últimos capítulos.

A continuación, el procedimiento inserta los nuevos datos en dbo.CUSTOMER y después el nuevo valor de CustomerID se lee dentro de la variable @Cid.

Para crear los renglones de la tabla de intersección apropiados se abre un cursor en una instrucción SQL que obtiene todos los renglones ARTIST, donde Nacionalidad es igual al parámetro @Nationality. Entonces el cursor se procesa en el ciclo WHILE y los nuevos valores se insertan en la tabla intersección CUSTOMER_ARTIST_INT. La instrucción FETCH (BUSCAR) mueve el cursor al siguiente renglón.

La figura 13-22 muestra cómo invocar este procedimiento almacenado usando el Analizador de Consultas SQL para agregar un nuevo cliente que tenga interés en artistas norteamericanos. Los parámetros pasan con los valores que se muestran. La figura 13-23 muestra los datos de la base después de que se ha ejecutado el procedimiento almacenado. La clienta Lynda Johnson ha sido agregada a la tabla CUSTOMER y se le asignó CustomerID 1038. Observe que hay tres artistas norteamericanos y se ha insertado un renglón de cada uno de ellos en la tabla CUSTOMER_ARTIST_INT a CustomerID 1038.

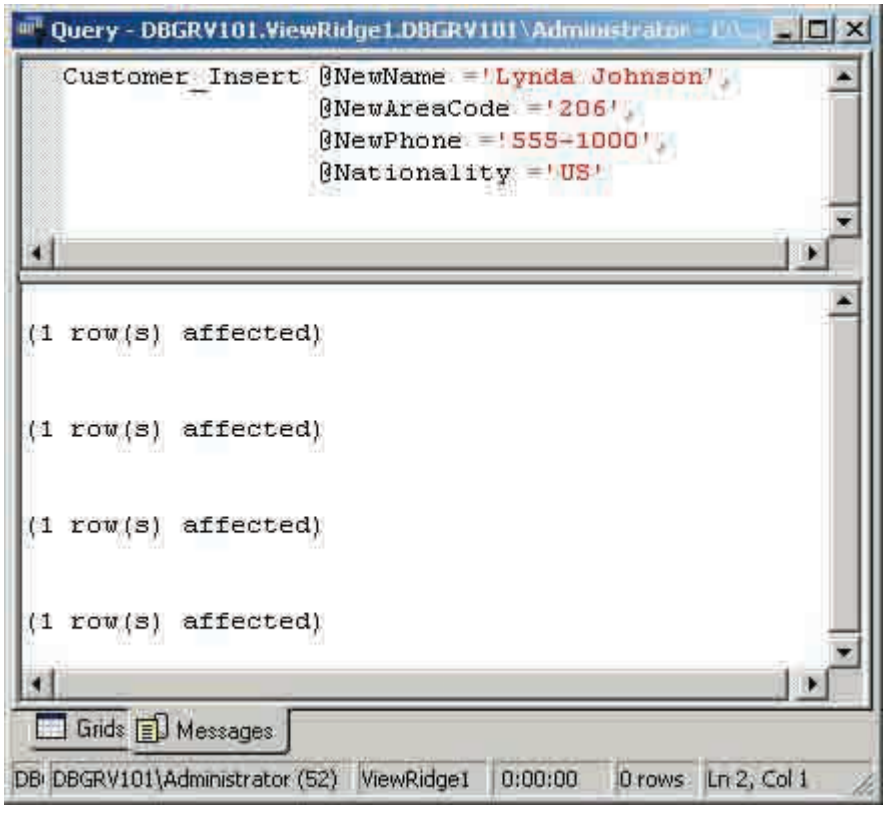
Para crear un procedimiento almacenado haga clic derecho en Stored Procedures (Procedimientos Almacenados) en una base de datos y seleccione Nuevo Procedimiento Almacenado. Para modificar un procedimiento almacenado haga clic derecho sobre el procedimiento en la lista de procedimientos y seleccione las propiedades.

EL PROCEDIMIENTO ALMACENADO NuevoClienteConTransacción. La figura 13-24, muestra un procedimiento almacenado que implementa la lógica para la creación de un seudocódigo de la vista View Ridge de la página 264. Este procedimiento recibe siete parámetros que tienen datos acerca del cliente nuevo y de sus compras.

La primera acción es revisar si ya existe el cliente; si existe, el procedimiento arroja un mensaje de error. Si el cliente no existe, entonces inicia una transacción. Recuerde que en el capítulo 11 dijimos que las transacciones aseguran que todas las actividades de la base de datos se confirmen atómicamente; todas o ninguna de las actualizaciones ocurrirá. La transacción comienza y se inserta el renglón del nuevo cliente. El nuevo valor de

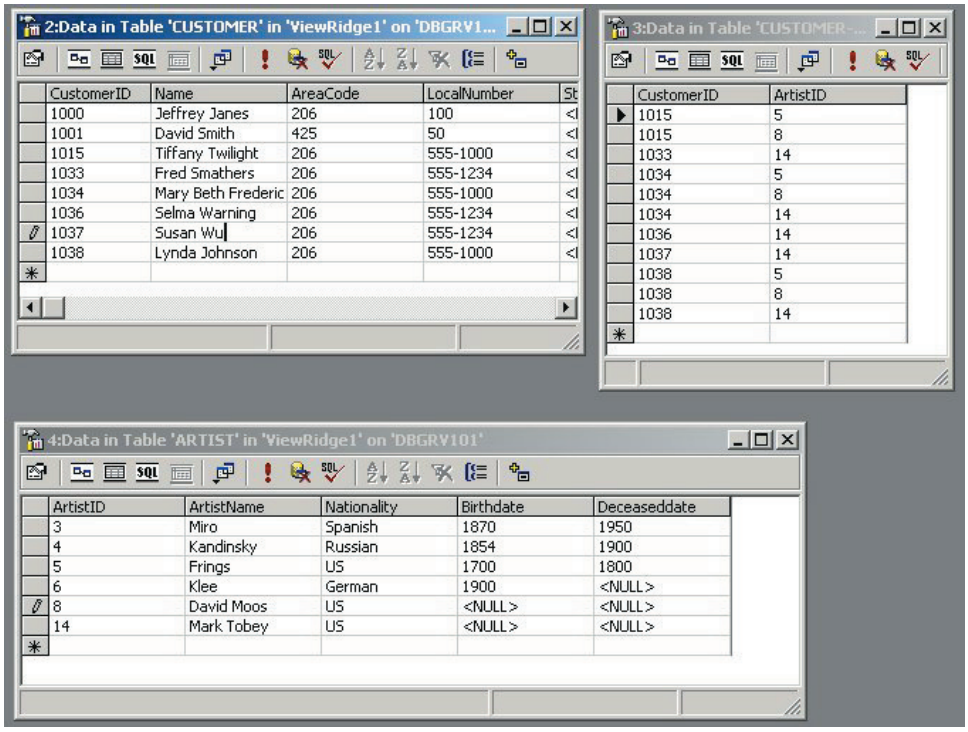
➤ FIGURA 13-22

Llamado del procedimiento Cliente_Insertar del analizador de consultas SQL



➤ FIGURA 13-23

Resultados del llamado al Procedimiento Almacenado de la figura 13-22



2:Data in Table 'CUSTOMER' in 'ViewRidge1' on 'DBGRV1...

CustomerID	Name	AreaCode	LocalNumber	St
1000	Jeffrey Janes	206	100	<I
1001	David Smith	425	50	<I
1015	Tiffany Twilight	206	555-1000	<I
1033	Fred Smathers	206	555-1234	<I
1034	Mary Beth Frederic	206	555-1000	<I
1036	Selma Warning	206	555-1234	<I
1037	Susan Wu	206	555-1234	<I
1038	Lynda Johnson	206	555-1000	<I

3:Data in Table 'CUSTOMER-...

CustomerID	ArtistID
1015	5
1015	8
1033	14
1034	5
1034	8
1034	14
1036	14
1037	14
1038	5
1038	8
1038	14

4:Data in Table 'ARTIST' in 'ViewRidge1' on 'DBGRV101'

ArtistID	ArtistName	Nationality	Birthdate	Deceaseddate
3	Miro	Spanish	1870	1950
4	Kandinsky	Russian	1854	1900
5	Frings	US	1700	1800
6	Klee	German	1900	<NULL>
8	David Moos	US	<NULL>	<NULL>
14	Mark Tobey	US	<NULL>	<NULL>

CUSTOMERID (ClienteID) se obtiene como se mostró anteriormente. Después, el procedimiento se revisa para determinar si ArtistID, WorkID y TransactionID (ArtistaID, TrabajoID y TransacciónID) son válidos. Si alguno es inválido la transacción se revierte (hace rollback).

Si todos son válidos, entonces una instrucción UPDATE actualiza PurchaseDate, Price y CustomerID (FechadeCompra, Precio y el ClienteID) en el renglón apropiado de TRANS. FechaDeCompra toma la fecha del sistema (mediante la función de sistema GETDATE(), es decir, OBTENER FECHA), SalesPrice (PreciodeVenta) se establece para el valor de @Price, y ClienteID, para el valor de @Cid. Por último, se agrega un renglón a CUSTOMER_ARTIST_INT (CLIENTE_ARTISTA_INT) para registrar los intereses del cliente en este artista. Si todo sucede de forma normal, entonces se confirma la transacción.

La figura 13-25 muestra la invocación de este procedimiento usando datos simples, y la figura 13-26 muestra los resultados en la base de datos. Al nuevo cliente se le asignó CustomerID 1040 y este ID se almacenó en la columna de llave externa CustomerID de TRANS, como se requiere. PurchaseDate y SalesPrice también se establecieron apropiadamente. Observe el nuevo renglón en la tabla de intersección que registra los intereses del cliente 1040 en artista 14, como se requiere.

DISPARADORES

Los disparadores (triggers) son procedimientos almacenados que se invocan cuando se procesan órdenes de inserción, actualización, o eliminación. Los disparadores **AFTER** se ejecutan después de que se ha procesado la orden; los disparadores **INSTEAD OF** se ejecutan en lugar de la instrucción que ocasionó que el disparador fuera invocado. Los disparadores AFTER se ejecutan después de que se ha procesado cualquier actualización o eliminación en cascada. SQL Server no maneja los disparadores BEFORE (ANTES), como sucede con Oracle.

DISPARADOR Nuevo_Precio. La figura 13-27 muestra un disparador que se usa para calcular el valor de la columna New Price (NuevoPrecio). La instrucción CREATE

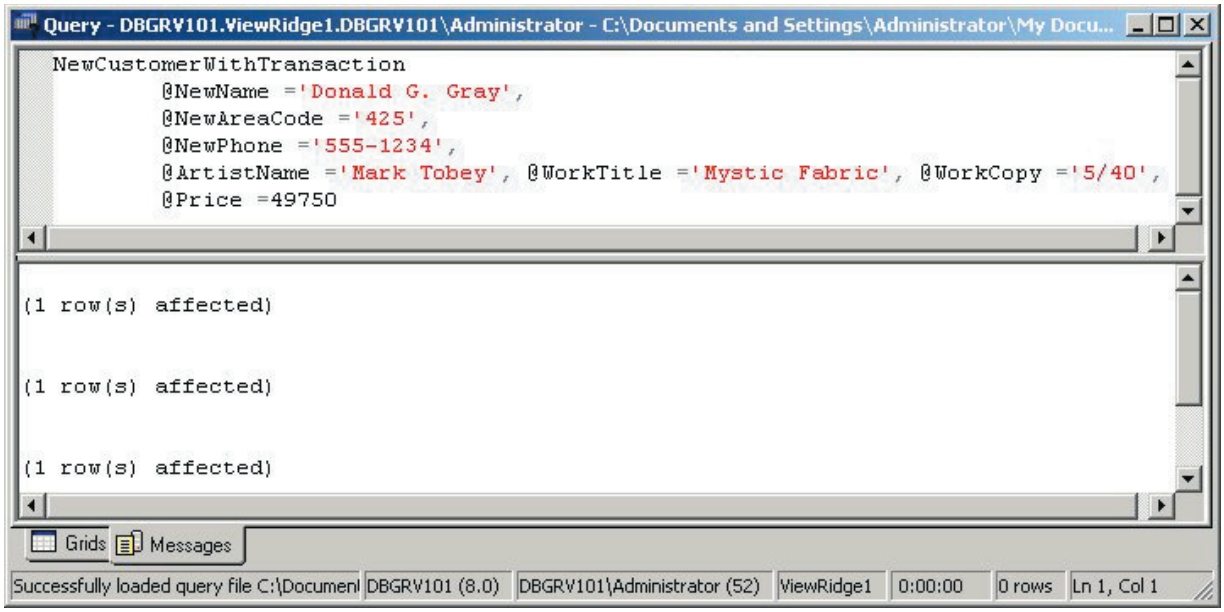
➤ FIGURA 13-24

NuevoCliente con el procedimiento almacenado Transacción

```
CREATE PROCEDURE NewCustomerWithTransaction
        @NewName char(50), @NewAreaCode char (3), @NewPhone char (8),
        @ArtistName char(50), @WorkTitle char(50), @WorkCopy char (10),
        @Price smallmoney
AS

DECLARE @Count as smallint
DECLARE @Aid as int
DECLARE @Cid as int
DECLARE @Wid as int
DECLARE @Tid as int

SELECT @Count = Count (*)
FROM dbo.CUSTOMER
WHERE [Name]=@NewName AND AreaCode=@NewAreaCode AND LocalNumber=@NewPhone

IF @Count > 0
        BEGIN PRINT 'Customer Already Exists – No Action Taken'
                RETURN
        END

BEGIN TRANSACTION /* Start transaction rollback everything if cannot complete it */

INSERT INTO dbo.CUSTOMER
        ([Name], AreaCode, LocalNumber)
        VALUES (@NewName, @NewAreaCode, @NewPhone)

Select @Cid = CustomerID
FROM dbo.CUSTOMER
WHERE [Name]=@NewName AND AreaCode=@NewAreaCode AND LocalNumber=@NewPhone

SELECT @Aid = ArtistID
FROM dbo.ARTIST
WHERE ArtistName=@ArtistName

If @Aid IS NULL  /* Invalid Artist ID */
        BEGIN
                Print 'Artist ID not valid'
                ROLLBACK
                RETURN
        END

SELECT @Wid = WorkID
FROM dbo.[WORK]
WHERE ArtistID = @Aid AND Title = @WorkTitle AND Copy = @WorkCopy
If @Wid IS NULL  /* Invalid Work ID */
        BEGIN
                Print 'Work ID not valid'
                ROLLBACK
                RETURN
        END

SELECT @Tid = TransactionID
FROM dbo.[TRANS]
WHERE WorkID=@Wid AND SalesPrice IS NULL
If @Tid IS NULL /*Invalid Transaction ID */
        BEGIN
                Print 'No valid transaction record'
                ROLLBACK
                RETURN
        END

UPDATE dbo.[TRANS] /* ALL is OK, update TRANS row */
SET PurchaseDate = GETDATE(), SalesPrice = @Price, CustomerID = @Cid
WHERE TransactionID=@Tid

INSERT INTO dbo.[CUSTOMER-ARTIST-INT] /* Create interest for this artist */
        (CustomerID, ArtistID)
        Values (@Cid, @Aid)

COMMIT
GO
```

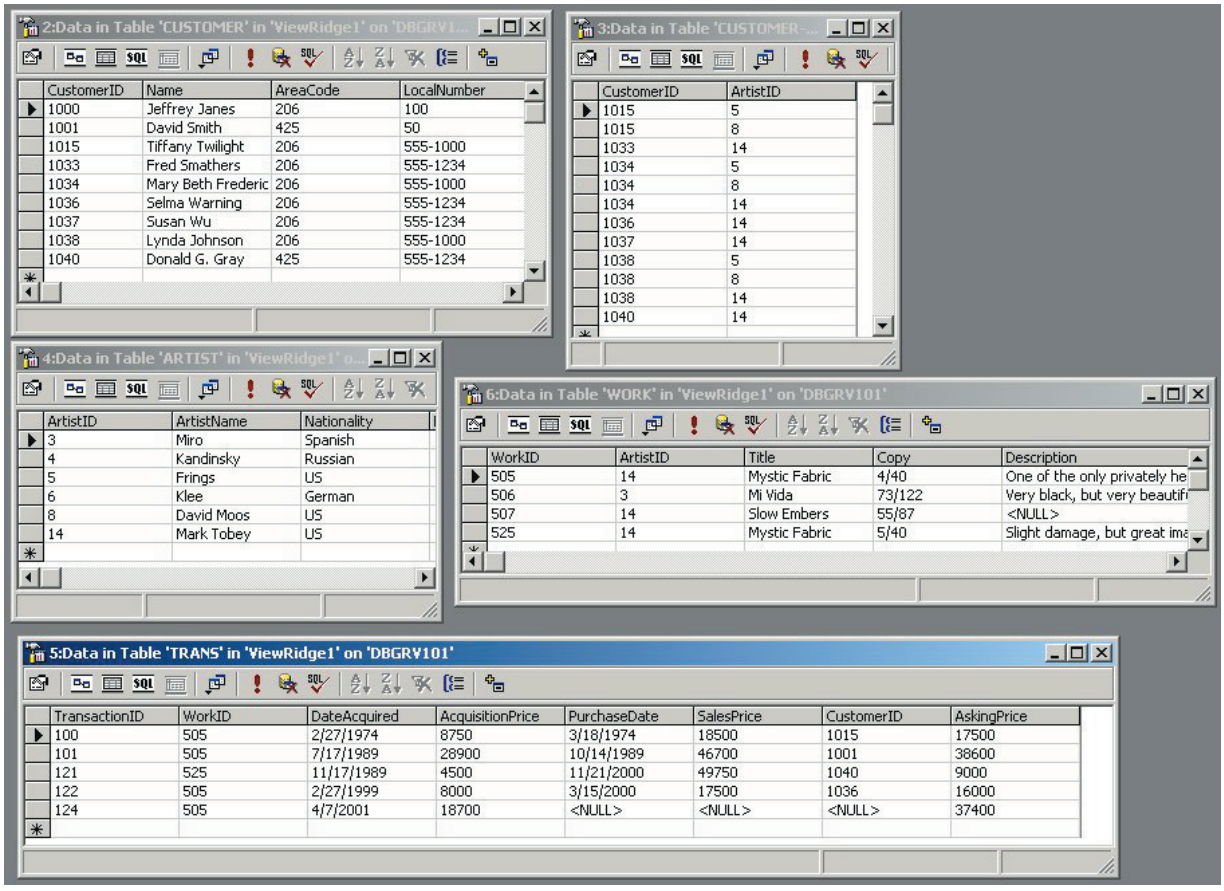
➤ FIGURA 13-25

Invocación del procedimiento almacenado de NuevoClienteconTransacción del Analizador de Consultas SQL



➤ FIGURA 13-26

Resultados de la llamada del Procedimiento Almacenado de la figura 13-25



➤ FIGURA 13-27

Disparador Nuevo_Precio

```
CREATE TRIGGER New_Price ON [dbo].[TRANS]
FOR INSERT, UPDATE
AS

DECLARE @NewPrice as smallmoney
DECLARE @id as int

IF NOT UPDATE (AcquisitionPrice) RETURN  /* Only concerned with new values or changes to Acquisition Price */

SELECT @NewPrice = AcquisitionPrice, @id = TransactionID
FROM inserted

Set @NewPrice = @NewPrice * 2 /* Could use a function here for more sophisticated aging of price, etc. */

/* Note: the following will cause a recursive call to this trigger.  It will terminate because this update does not
   involve AcquisitionPrice */

UPDATE dbo.TRANS
SET AskingPrice = @NewPrice
WHERE TransactionID =  @id
```

TRIGGER indica que éste se invoca para insertar y actualizar la tabla TRANS. El código de disparador primero determina si la actualización (aquí puede significar insertar o actualizar) involucra a la columna Acquisition Price (PreciodeAdquisición). En caso contrario, termina.

Si la inserción o la actualización involucran a la columna PreciodeAdquisición, entonces la siguiente instrucción emite un SELECT que obtiene el nuevo valor de PreciodeAdquisición y el valor de TransacciónID. Esta selección opera sobre una seudotabla que se mantiene mediante SQL Server. Esta tabla se llama *insertada* y contiene todos los datos insertados o cambiados. Existe otra tabla similar (no la usamos aquí) llamada *eliminada*, con los datos que existían antes de cualquier operación de borrado. Las tablas insertadas y eliminadas sólo están disponibles en el código de disparador. Son similares a los prefijos *:new* y *:old*, los cuales se usan en Oracle.

A continuación, el disparador calcula el nuevo valor del precio en la variable @Price. Este valor se usa para actualizar la columna PreguntaPrecio del renglón que se está actualizando. En este disparador, el UPDATE DE TRANS causará que dispare de manera repetitiva. Sin embargo, la segunda vez que dispare la actualización no afectará a PreciodeAdquisición y ese segundo disparo saldrá sin hacer otra actualización.

A propósito, ésta es una manera costosa de implementar una fórmula tan sencilla. Una solución mejor sería definir esta fórmula como una propiedad de la columna PreguntaPrecio en la definición de la tabla TRANS. Sin embargo, si el cálculo de PreguntaPrecio fuera más complicado, si por ejemplo implicara entrar a otras tablas o incluso a otras bases de datos, entonces sería necesario este disparador.

DISPARADOR On_WORK_Insert. La figura 13-28 muestra un uso más común para el código de disparador. Como se mencionó en el capítulo 10, el mismo trabajo artístico de la galería View Ridge puede ser vendido varias veces. Sin embargo, debido a que estamos usando llaves sustitutas, cada vez que se introduce un trabajo en la base de datos, ingresa como un renglón nuevo y aparecerá como un trabajo artístico diferente. Para evitar esto, se escribe el disparador On_WORK_Insert, con el fin de verificar si el trabajo nuevo ya había estado almacenado en la base de datos. Si es así, el renglón WORK que acaba de ser creado se elimina de la base de datos y se usa el viejo renglón.

Además, la galería quiere que cada WORK que está a la venta tenga un renglón TRANS, con un valor nulo para CustomerID. Si el trabajo no ha estado antes en la galería,

➤ FIGURA 13-28

Disparador On_WORK_Insert

```
CREATE TRIGGER On_WORK_Insert ON dbo.[WORK] FOR INSERT

AS

DECLARE @NewWorkID as int
DECLARE @Count as smallint
DECLARE @Aid as int
DECLARE @Title as char (50)
DECLARE @Copy as char (10)


SELECT      @NewWorkID = WorkID, @Aid = ArtistID, @Title = Title, @Copy = Copy
FROM        Inserted  /* this pseudo-table of inserted data is available to triggers */

SELECT      @Count = Count (*)
FROM        ViewRidge1.dbo.[WORK]
WHERE       ArtistID = @Aid AND Title = @Title AND Copy = @Copy

IF @Count > 2   /* Error – this work had 2  rows PRIOR to new insert */
      BEGIN
            ROLLBACK
            Print 'Error in work table data; more than one available row for this work.'
            RETURN
      END

IF @Count = 2  /* Means one was available prior to insert -- rollback new insert  and use existing row*/
      BEGIN
            ROLLBACK
            SELECT @NewWorkID = WorkID /* Get WorkID of existing row */
            FROM ViewRidge1.dbo.[WORK]
            WHERE ArtistID = @Aid AND Title = @Title AND Copy = @Copy
      END

SELECT @Count = Count(*) /* Check for available TRANS row */
FROM ViewRidge1.dbo.TRANS
WHERE @NewWorkID = WorkID AND CustomerID IS NULL

IF @Count > 0 RETURN /* One is Available */

INSERT INTO ViewRidge1.dbo.TRANS
(WorkID, DateAcquired)
VALUES (@NewWorkID, GETDATE())
```

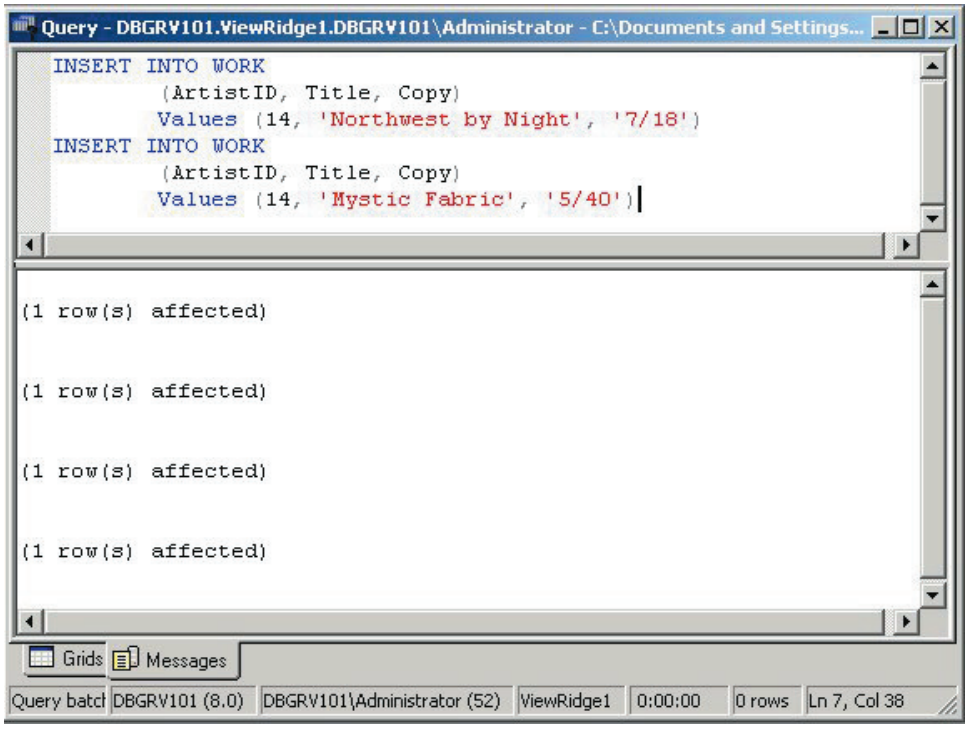
debe crearse un nuevo registro TRANS y conectarse al nuevo renglón WORK. Si el trabajo ha estado antes en la galería, y si no hay renglón TRANS para ese WORK con un CustomerID nulo, entonces debe crearse un nuevo renglón TRANS. Si ya existe ese renglón TRANS, entonces el trabajo nunca se vendió y se puede usar el renglón existente TRANS.

La primera tarea del procedimiento es obtener los nuevos valores WORK de la seudotabla *insertada*. Entonces, se emite una instrucción SELECT para contar los números de los renglones WORK que tiene determinado ArtistID Title y Copy. Si la cuenta es mayor a dos ha ocurrido un error. Debe haber cuando mucho dos renglones que tengan esos datos —el renglón original y el renglón nuevo—. Si hay más de dos, el disparador revierte la transacción, imprime un mensaje de error y termina.

Si hay exactamente dos renglones, entonces el trabajo ha estado en la galería antes y el disparador revierte la transacción para eliminar sólo el renglón WORK que acaba de ser insertado. A propósito, esta reversión (roolback) elimina cualquier cambio he-

➤ FIGURA 13-29

Inserción que invocará el disparador en la figura 13-28



cho por la transacción que ocasionó que este disparador también fuera invocado. Después de la reversión, el código del disparador obtiene WorkID para el otro renglón que tiene estos datos de trabajo.

A estas alturas, WorkID muestra el renglón viejo de un trabajo que ha estado antes en la galería, o muestra el nuevo renglón que se ha creado para un trabajo que es nuevo para la galería.

La única tarea restante es determinar si hay un renglón TRANS en la base de datos para este trabajo que tiene un valor nulo de CustomerID. Si es así, no es necesario crear nuevos renglones TRANS. En caso contrario, se inserta un nuevo renglón TRANS en la base de datos.

La figura 13-29 muestra la inserción de dos trabajos en la base de datos View Ridge. El primer trabajo nunca ha estado antes en la galería y el segundo sí. La figura 13-30 muestra las tablas WORK y TRANS después de que ambas han sido insertadas. Se agregó un nuevo renglón WORK para el trabajo "Northwest by Night", pero no se insertó un nuevo renglón para "Mystic Fabric" Copia "5/40" porque este trabajo ya estaba en la base de datos (podemos afirmar que ya existía porque el renglón TRANS con TransactionID de 131 estaba destinado a ésta).

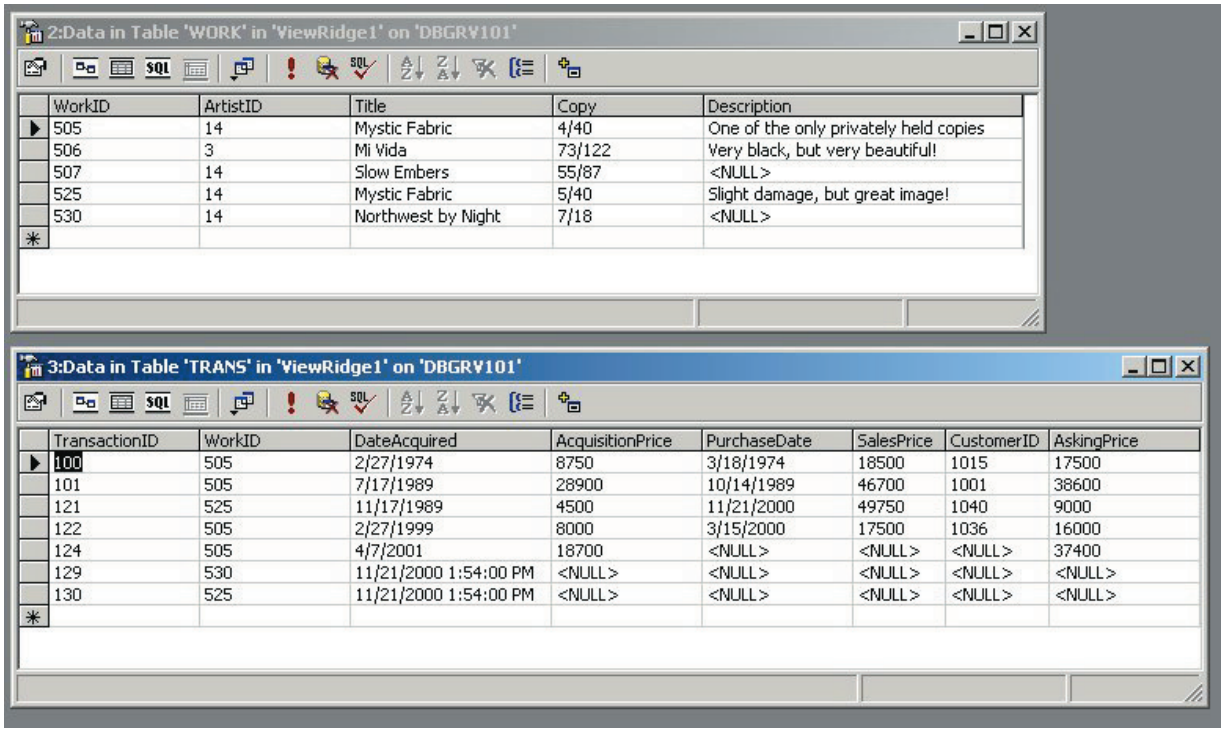
Se agregaron dos nuevos renglones a TRANS; se establecieron las llaves sustitutas en los renglones correctos de WORK. Observe que la función GETDATE proporciona ambas cosas, fecha y hora, en la columna DateAcquired. Aparentemente, los otros valores en DateAcquired se colocaron manualmente porque la hora no aparece en sus valores.

Se puede hacer una vista que de otra manera no es actualizable definiendo un disparador INSTEAD OF. En este caso, SQL Server no intentará ejecutar la operación de inserción, actualización, o eliminación, pero llamará al disparador. Esto es necesario porque no es posible que el proveedor del DBMS escriba un código de objetivo general, es decir, que inserte cualquier instancia de vista arbitraria. Sin embargo, conociendo la aplicación es posible escribir el código para hacerlo en el caso de vistas específicas.

Esta sección ha abordado el procedimiento de almacenamiento básico y la funcionalidad del disparador. Hay mucho que aprender, pero esto le dará las bases suficientes para empezar. En el capítulo 15 veremos cómo invocar los procedimientos de almacenamiento usando la tecnología de Internet.

➤ FIGURA 13-30

Resultados de las inserciones de la figura 13-29



2:Data in Table 'WORK' in 'ViewRidge1' on 'DBGR¥101'

WorkID	ArtistID	Title	Copy	Description
505	14	Mystic Fabric	4/40	One of the only privately held copies
506	3	Mi Vida	73/122	Very black, but very beautiful!
507	14	Slow Embers	55/87	<NULL>
525	14	Mystic Fabric	5/40	Slight damage, but great image!
530	14	Northwest by Night	7/18	<NULL>

3:Data in Table 'TRANS' in 'ViewRidge1' on 'DBGR¥101'

TransactionID	WorkID	DateAcquired	AcquisitionPrice	PurchaseDate	SalesPrice	CustomerID	AskingPrice
100	505	2/27/1974	8750	3/18/1974	18500	1015	17500
101	505	7/17/1989	28900	10/14/1989	46700	1001	38600
121	525	11/17/1989	4500	11/21/2000	49750	1040	9000
122	505	2/27/1999	8000	3/15/2000	17500	1036	16000
124	505	4/7/2001	18700	<NULL>	<NULL>	<NULL>	37400
129	530	11/21/2000 1:54:00 PM	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>
130	525	11/21/2000 1:54:00 PM	<NULL>	<NULL>	<NULL>	<NULL>	<NULL>

➤ CONTROL DE CONCURRENCIA

SQL Server 2000 proporciona un conjunto de posibilidades de procesamiento para el control de concurrencia. Hay muchas opciones disponibles, y el comportamiento resultante se determina mediante la interacción de tres factores: nivel de aislamiento de transacción, colocación de la concurrencia en el cursor, sugerencias para el locking proporcionadas en la cláusula SELECT. El comportamiento del locking también depende de que se procese o no el cursor como parte de una transacción, si la instrucción SELECT es parte de un cursor, y si los comandos de actualización ocurren dentro de la transacción o de forma independiente.

En esta sección sólo describiremos las bases. Vea la documentación de SQL Server 2000 para mayor información.

Con SQL Server los programadores no colocan directamente los locks, sino que establecen el comportamiento del control de concurrencia que desean y SQL Server determina dónde colocar los locks. Los locks se aplican en renglones, páginas, llaves, índices, tablas, e incluso en toda la base de datos. SQL Server determina qué nivel de bloqueo se usa y puede aumentar o disminuir un nivel mientras procesa. También determina cuándo colocar el lock y cuándo quitarlo, dependiendo de las declaraciones del programador.

NIVEL DE AISLAMIENTO DE TRANSACCIÓN

La figura 13-31 resume las opciones de control de concurrencia. El grado de establecimiento más amplio es el **nivel de aislamiento de transacción.** Las opciones se listan en un nivel ascendente de restricción en el primer renglón de la figura 13-31. Estas cuatro opciones fueron analizadas en el capítulo 11; son los niveles de las normas SQL-92. Observe que con SQL Server es posible hacer lecturas sucias colocando el nivel de

➤ FIGURA 13-31

Opciones de concurrencia con SQL Server 2000

Tipo	Alcance	Opciones
Nivel de aislamiento de transacción	Conexión Todas las transacciones	LECTURA NO CONFIRMADA LECTURA CONFIRMADA (predeterminada) LECTURA REPETIBLE SERIALIZABLE
Concurrencia de cursor	Cursor	SÓLO_LECTURA OPTIMISTA SCROLL_LOCK
Sugerencias de bloqueo	SELECT	LECTURACONFIRMADA LECTURANOCONFIRMADA LECTURAREPETIBLE SERIALIZABLE NO LOCK HOLDLOCK y otros...

aislamiento a LECTURANOCONFIRMADA. LECTURA CONFIRMADA es el nivel de aislamiento predeterminado.

El siguiente nivel más restrictivo es LECTURA REPETIBLE, lo cual significa que SQL Server coloca y mantiene los locks en todos los renglones que son de lectura. Quiere decir que ningún otro usuario puede cambiar o eliminar un renglón que ha sido leído hasta que la transacción se confirme o se aborte. Sin embargo, la relectura del cursor puede dar como resultado lecturas fantasmas.

El nivel de aislamiento más estricto es SERIALIZABLE. Con éste, SQL Server coloca un lock de rango en los renglones que han sido leídos. Esto asegura que los datos no leídos se puedan cambiar o eliminar, y que no se pueden insertar renglones nuevos en el rango que puedan ocasionar lecturas fantasmas. Este nivel es el más costoso de imponer y únicamente se debe usar cuando sea absolutamente necesario.

Un ejemplo de la instrucción TRANSACT-SQL es establecer el nivel de aislamiento a, por ejemplo, LECTURA REPETIBLE, lo cual sería:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Esta instrucción se podría activar en cualquier lugar en donde TRANSACT-SQL esté permitido, antes de cualquier otra actividad de la base de datos.

Concurrencia de cursor

La segunda forma en la que el programador puede declarar las características del locking es con la concurrencia del cursor. Las posibilidades son de sólo lectura, optimista y pesimista, y aquí se le llama SCROLL_LOCK. Como se describió en el capítulo 11, con lockings optimistas el locking no se obtiene hasta que el usuario actualiza los datos. En ese punto, si los datos han cambiado desde que se leyeron, la actualización es rechazada. Por supuesto, el programa de aplicación debe especificar qué hacer cuando ocurre dicho rechazo.

SCROLL_LOCK es una versión de bloqueo pesimista, con el cual se establece una actualización de un bloqueo en un renglón cuando éste es leído. Si el cursor se abre dentro de la transacción, el lock se mantiene hasta que ésta se confirme o se revierta. Si el cursor está fuera de la transacción, el lock se elimina cuando se lee el siguiente renglón. Recuerde que en el capítulo 11 se dijo que una actualización de un bloqueo bloqueará la actualización de otro más, pero no hará lo mismo con un bloqueo compartido. Por lo tanto, otras conexiones pueden leer el renglón con bloqueos compartidos.

La concurrencia del cursor predeterminada depende del tipo de cursor (vea el capítulo 11). Es de sólo lectura con cursores estáticos y de forward only, y es optimista en el caso de cursores dinámicos y de keyset.

La concurrencia del cursor se establece con la instrucción DECLARE CURSOR. Un ejemplo para declarar un cursor dinámico SCROLL_ROLL en todos los renglones de la tabla TRANS es:

```
DECLARE     MY_CURSOR CURSOR DYNAMIC SCROLL_LOCKS
FOR
        SELECT      *
        FROM        TRANS
```

Sugerencias de lockings

El comportamiento del locking se puede modificar más proporcionando sugerencias de locking en el parámetro WITH de la cláusula FROM en las instrucciones SELECT. La figura 13-31 lista varias sugerencias de locking disponibles con SQL Server. Las primeras cuatro anulan el nivel de aislamiento; las dos siguientes influyen en el tipo de lock usado.

Considere las siguientes instrucciones:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
DECLARE   MY_CURSOR CURSOR DYNAMIC SCROLL_LOCKS
FOR
        SELECT      *
        FROM        TRANS WITH READUNCOMMITTED NOLOCK
```

Sin las sugerencias de locking, el cursor MY_CURSOR tendría el aislamiento de LECTURA REPETIBLE y generaría locks de actualización en todos los renglones leídos. Los locks se mantendrían hasta que se confirmara la transacción. Con las sugerencias de locking el nivel de aislamiento para este cursor sería LECTURA NO CONFIRMADA. Además, la especificación de NOLOCKING cambia este cursor de DINÁMICO a SÓLO_LECTURA.

Veamos otro ejemplo:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
DECLARE   MY_CURSOR     CURSOR DYNAMIC SCROLL_LOCKS
FOR
        SELECT      *
        FROM        TRANS WITH HOLDLOCK
```

Aquí, la sugerencia de bloqueo causará que SQL Server mantenga los locks de actualización en todos los renglones leídos hasta que se confirme la transacción. El efecto es cambiar el nivel de aislamiento de transacción para este cursor de LECTURA REPETIBLE a SERIALIZABLE.

En general, al principiante se le aconseja no proporcionar sugerencias de locking sino hasta que se haya convertido en un experto, y establezca el nivel de aislamiento y la concurrencia del cursor a valores apropiados para sus transacciones y cursores, y que se queden así.

➤ SEGURIDAD

Analizamos la seguridad de SQL Server en el capítulo 11, así que no lo haremos aquí. A manera de repaso, diremos que una cuenta es una identificación que se usa para inscribirse en SQL Server, o para inscribirse en Windows 2000 y después pasar a SQL Server. Una cuenta está asociada con una o más bases de datos o combinaciones de usuarios. Un rol es un grupo de usuarios tales como vendedores o administradores. Los usuarios pueden tener de cero a muchos roles y tienen permisos. Las funciones tienen de cero a muchos usuarios y también tienen permisos.

Los permisos asociados con una cuenta determinada son todos los que tiene el usuario, los cuales están asociados con la cuenta más todos los permisos de todos los roles que tienen asignados dichos usuarios. A éstos se les puede dar un permiso para que se lo den otros usuarios, con la opción GRANT.

➤ RESPALDO Y RECUPERACIÓN

Cuando usted crea una base de datos en SQL Server, se crean datos y archivos de log. Como se explicó en el capítulo 11, estos archivos deberán ser respaldados periódicamente. Cuando se hace, es posible recuperar una base de datos que falla restaurándola a partir de una anterior almacenada y aplicando los cambios en el log.

Para recuperar una base de datos con SQL Server, ésta se restaura a partir del respaldo de una base de datos anterior y aplicando los cambios registrados en el log. Cuando se alcanza el final del log, los cambios que fallaron en la confirmación de alguna transacción se revierten (hacen rollback).

También es posible procesar el registro para un punto en particular en el tiempo o para una marca de transacción. Por ejemplo, la instrucción:

```
BEGIN TRANSACCIÓN NuevoCliente WHIT MARK
```

ocasionará una marca etiquetada NuevoCliente para que se coloque en el bloque cada vez que se ejecute esta transacción. Si se hace esto, el log se puede restaurar un punto justo antes o justo después de la primera marca NuevoCliente o de la primera marca NuevoCliente después de un punto particular en el tiempo. El log restaurado se puede usar para restaurar la base de datos. Sin embargo, estas marcas ocupan del espacio log y, por lo tanto, no deberían usarse si no hay una buena razón.

Tipos de respaldo

SQL Server maneja varios tipos de respaldo. Para verlos, abra el Administrador (Enterprise Manager) y las Bases de datos (Databases), y haga clic derecho en el nombre de la base de datos. Seleccione Todas las tareas (All Tasks) y después Respaldo de base de datos (Backup Database). Aparecerá el diálogo que se muestra en la figura 13-32. En este punto, se puede crear un respaldo de la base de datos completo o diferencial, crear un respaldo de un bloque de transacción, y crear un respaldo de archivos en particular y grupos de archivos.

Como su nombre lo indica, un respaldo completo hace una copia de toda la base de datos; un **respaldo diferencial** hace una copia de los cambios que se han hecho a la base de datos desde que concluyó el último respaldo. Esto significa que se debe hacer un respaldo completo antes del primer respaldo diferencial. Debido a que éstos son más rápidos, se puede optar por ellos con más frecuencia y se reduce el riesgo de perder datos. Por otra parte, los respaldos completos llevan un tiempo más largo, pero son un poco más sencillos de usar para la recuperación, como lo podrá ver.

El log de transacción también necesita ser respaldado periódicamente para asegurarse de que sus contenidos se preservan. Además, el log de transacción debe respaldarse antes de que se pueda usar para recuperar una base de datos.

Los respaldos se pueden hacer en un disco o en cinta. Cuando es posible, se deben hacer en otras facilidades diferentes a aquellas que almacenan la base de datos operacional y el log. Respaldarlos en facilidades removibles permite que los respaldos se almacenen físicamente en un lugar del que se puedan sacar del centro de datos. Esto es importante para la recuperación en caso de desastres causados por inundaciones, huracanes, etcétera.

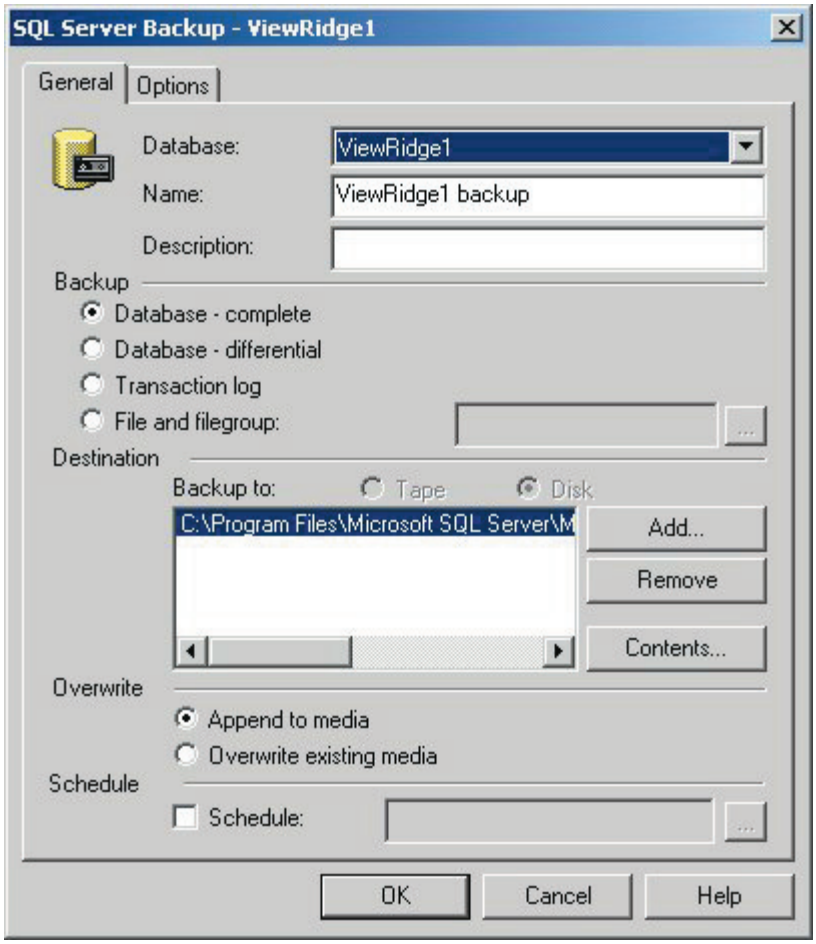
Modelos de recuperación del SQL Server

SQL Server maneja tres modelos de recuperación: simple, completo y bulk-logged. Puede elegirse el modelo de recuperación haciendo clic derecho en un nombre de base de datos en el Administrador Empresarial (Enterprise Manager) y seleccionando las propiedades. El modelo de recuperación se especifica en Options tab. La figura 13-33 muestra un ejemplo.

Con el modelo de recuperación simple no se hace el registro. La única manera de recuperar una base de datos es restaurarla a partir del último respaldo. Hay cambios,

➤ FIGURA 13-32

Diálogo para la creación de respaldos



puesto que los últimos respaldos se perdieron. El modelo de recuperación simple se puede usar para una base de datos que nunca cambia; por decir algo, que tiene los nombres y los lugares de los ocupantes de un cementerio lleno. Dicho de otra manera, la que se usa para el análisis de datos de sólo lectura que se copian de alguna otra base de datos transaccional.

Con la recuperación completa quedan registrados todos los cambios de la base de datos, y con el bulk-logged de la recuperación de la base de datos se registran todos los cambios, excepto los que ocasionan entradas grandes de archivos al log. Con la recuperación del bulk-logged los cambios de grandes textos y los elementos de datos gráficos no se graban en el registro; las acciones como CREATE INDEX no se registran y algunas otras acciones orientadas al volumen tampoco. Una empresa usaría la recuperación de bulk-logged si conservar un espacio considerable fuera importante y si los datos usados en las operaciones de volumen se guardan de alguna otra manera.

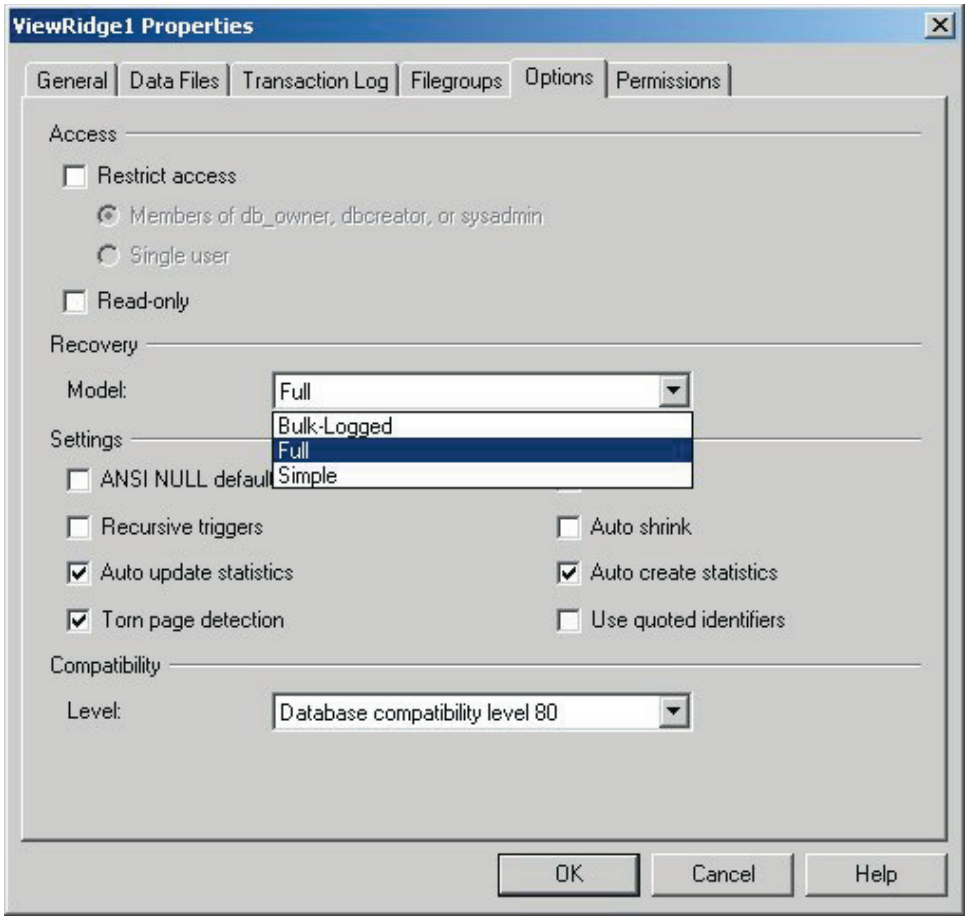
Restauración de una base de datos

Si la base de datos y los archivos de log han sido respaldados apropiadamente, la restauración de la base de datos será sencilla. Primero se respalda el log actual para que los cambios en el log más reciente estén disponibles. Después se da clic derecho en el nombre de la base de datos en el Enterprise Manager (Administrador); se selecciona All Tasks (Todas las tareas) y después Restore Database (Restaurar base de datos). Aparecerá el diálogo que se muestra en la figura 13-34.

En este ejemplo la base de datos View Ridge1 está siendo restaurada como base de datos View Ridge3. No es necesario cambiar el nombre de la base de datos. Aquí, la idea

➤ FIGURA 13-33

Recuperación de las opciones del modelo



es restaurarla con un nombre diferente, probar la base de datos restaurada, eliminar lo que quedó de base de datos anterior y después renombrar la base de datos recuperada View Ridge1.

Esta base de datos está siendo restaurada en un punto en la fecha 11/24/2000, 2:08:24 p.m. Si esta entrada se dejara en blanco, la base de datos se restauraría al final del registro.

Cuando se hace clic en OK, SQL Server inicia la restauración y notifica al DBA los progresos y el estado de cumplimiento.

Cabe aclarar que esta restauración se hizo porque en el proceso de preparación de este capítulo inadvertidamente transmití la siguiente actualización:

```
UPDATE    WORK
SET       Copy='99/2000'
```

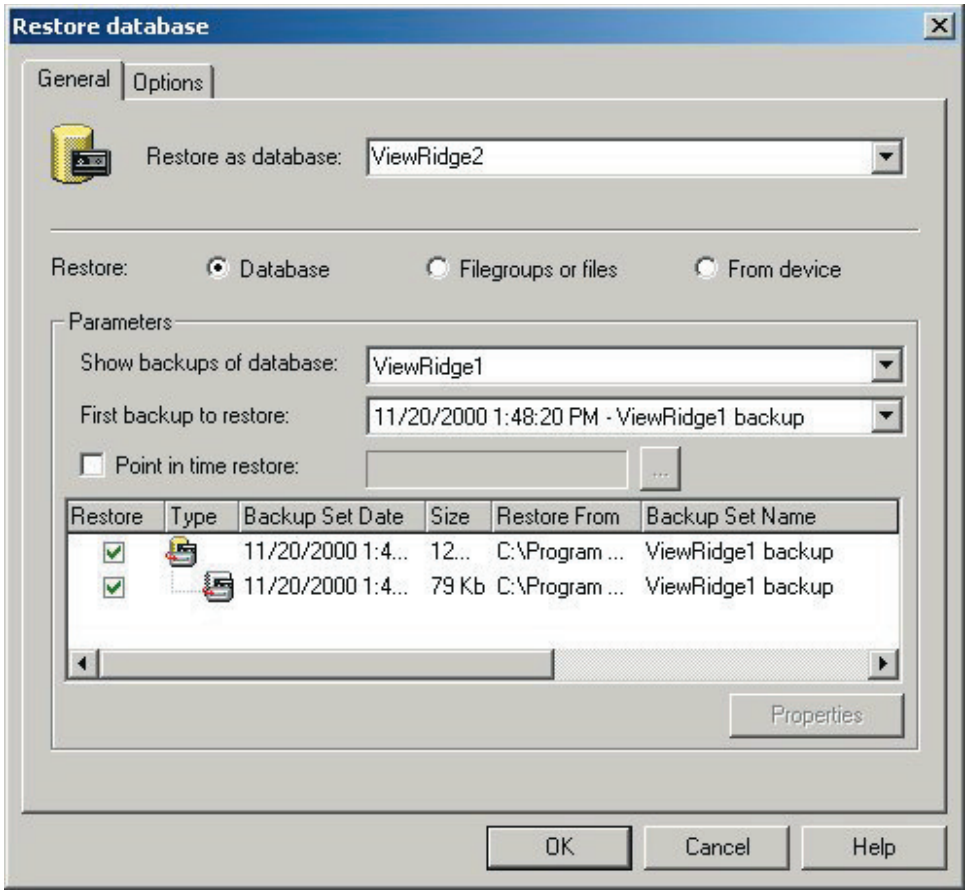
La consulta que quería transmitir era:

```
UPDATE    WORK
SET       Copy='99/2000'
WHERE     WorkID=530
```

Cometí este error justo después de las 2:30 p.m., así que lo restauré en un momento que fue anterior a éste, pero me di cuenta que fue después de otros cambios que yo quería conservar.

➤ FIGURA 13-34

Restauración de la base de datos en un punto en el tiempo



Plan de mantenimiento de la base de datos

Se puede crear un plan de mantenimiento de la base de datos para, entre otras tareas, facilitar la creación de la base de datos y el respaldo de logs. SQL Server proporciona un asistente para esta tarea. Para usarlo, dé clic derecho en el nombre de la base de datos, seleccione Todas las tareas, y seleccione Plan de mantenimiento. El asistente le guiará a través del proceso de programación de varias tareas. Algunas mantienen la base de datos por medio de la reorganización de índices y de otras actividades relacionadas. Sin embargo, lo importante aquí es que puede programar el respaldo automático de los datos de la base y de los registros.

➤ TEMAS QUE NO SE ANALIZARON EN ESTE CAPÍTULO

Hay varios temas importantes de SQL Server que están más allá del ámbito de este análisis. Uno de ellos es que SQL Server proporciona facilidades para medir la actividad y el desempeño de la base de datos. El DBA puede usarlas cuando afine la base de datos. Otra facilidad no descrita es la conexión de Access a SQL Server. Puede revisar la documentación de Access para obtener mayor información acerca de este tema.

SQL Server 2000 proporciona facilidades para el procesamiento de la bases de datos distribuidas (llamadas **replicación** en SQL Server). Estas facilidades usan el modelo de distribución suscripción-publicación que analizaremos en el capítulo 17. El procesamiento de distribución de la base de datos, si bien es cierto que es muy importante por sí mismo, está más allá del ámbito de este texto. Microsoft proporciona un servidor OLE DB llamado Administrador de Transacciones Distribuidas, que las coordina. Java

maneja Enterprise Java Beans con el mismo propósito. Abordaremos estos temas en los capítulos 15 y 16.

Por último, SQL Server tiene facilidades para el procesamiento de la base de datos con vistas en forma de documentos XML. Analizaremos el XML con detalle en el siguiente capítulo.

➤ RESUMEN

SQL Server se puede instalar en computadoras con Windows NT y Windows 2000. Hay dos formas de crear tablas, vistas, índices y otras estructuras de base de datos. Una es usar las herramientas de diseño gráfico, que son similares a las de Microsoft Access. La otra es escribir las instrucciones SQL para crear estructuras y transmitirlas a SQL Server mediante el Analizador de Consultas SQL. Los tipos de datos de SQL Server se listan en la figura 13-8.

SQL Server 2000 implementa los Tipos de Datos Definidos por el Usuario, los cuales se usan para implementar dominios. Una vez definidos, se pueden usar para la definición de columnas en la herramienta gráfica o en las instrucciones SQL. Las reglas se pueden definir y aplicar a las columnas, o a los Tipos de Datos Definidos por el Usuario. Si es esto último, las reglas serán impuestas en todas las columnas basadas en los tipos de datos del usuario.

Se puede cambiar la estructura de la tabla usando la herramienta gráfica o la instrucción SQL ALTER TABLE. Se pueden definir las relaciones en los diagramas de la base de datos o mediante la definición de llaves externas en las instrucciones SQL.

Algunas propiedades de las relaciones pertenecen a la integridad referencial. Si la relación impuesta se verifica, SQL Server no permitirá valores de llaves externas en las tablas hijas que no se acoplen a los valores de las llaves primarias de las tablas padre. Se confirman las actualizaciones en cascada y eliminaciones en cascada. Si el diseño de la base de datos usa llaves sustitutas, las actualizaciones en cascada nunca serán necesarias.

SQL Server mantiene las vistas SQL. Algunas se pueden usar para la actualización de datos y otras no. Cualquier vista basada en una sola tabla se actualiza, a menos que incluya funciones agregadas o columnas derivadas. Todas las columnas NOT NULL se deben presentar en la vista para que acepte las inserciones. Ninguna vista que tenga más de una tabla se puede usar para operaciones de eliminación. Las vistas de tablas múltiples pueden aceptar las inserciones y las actualizaciones, mientras que esas operaciones se apliquen a una sola tabla en la vista. Las vistas que no son actualizables de otra manera se pueden hacer mediante la definición de disparadores INSTEAD OF.

Los índices son estructuras especiales de datos que se usan para mejorar el desempeño. SQL Server crea automáticamente un índice en todas las llaves primarias y externas. Las vistas adicionales se pueden crear usando CREATE INDEX, o la herramienta gráfica de Administración de Índice. SQL Server mantiene índices agrupados y no agrupados.

SQL Server usa el lenguaje TRANSACT-SQL, que abarca las instrucciones básicas SQL con construcciones programadas como parámetros, variables y estructuras lógicas, tales como IF, WHILE, etc. Hay tres formas de procesar la aplicación de las estructuras lógicas con SQL Server para guardar consultas y procedimientos TRANSACT-SQL y se logran mediante el Analizador de Consultas SQL, para crear procedimientos almacenados e invocarlos mediante el programa de aplicación o el Analizador de Consultas SQL, y escribir disparadores que sean invocados por medio de SQL Server cuando ocurran determinadas acciones de la base de datos.

Los procedimientos almacenados se pueden guardar en las computadoras de los usuarios, o conservar dentro de la base de datos e invocarlos mediante programas de aplicación. Lo último es más común para aplicaciones de Internet, como lo aprenderá en el capítulo 15. En la figura 13-24 se muestra un procedimiento almacenado para la lógica de almacenar un nuevo cliente y una transacción. En las figuras 13-25 y 13-26 se muestran ejemplos de disparadores de inserción y de actualización.

Tres factores determinan el comportamiento del control de concurrencia de SQL Server: nivel de aislamiento de transacción, establecimiento de la concurrencia del cursor, y sugerencias de bloqueo proporcionadas en la cláusula SELECT. Estos factores se resumen en la figura 13-31. El comportamiento también cambia dependiendo de que las acciones ocurran o no en el contexto de las transacciones, de los cursores, o que suceda de manera independiente. Considerando estas declaraciones de comportamiento, SQL Server coloca locks por parte del programador. Los locks se pueden colocar en muchos niveles de granularidad y puede aumentar o disminuir conforme progrese el trabajo.

SQL Server mantiene el respaldo del log y los respaldos completo y diferencial de la base de datos. Hay tres modelos de recuperación disponibles: simple, completo y bulk-logged. Con la recuperación sencilla, no se hacen ni se aplican los logs. Los logs recuperan totalmente todas las operaciones de la base de datos y los aplican para la restauración. La recuperación bulk-logged registrada omite ciertas transacciones que de otra manera consumirían grandes cantidades de espacio en el log.

➤ PREGUNTAS DEL GRUPO I

13.1 Instale SQL Server 2000 y cree una base de datos llamada MEDIA. Use los valores predeterminados para los tamaños de archivos, nombres y ubicaciones.

13.2 Escriba una instrucción SQL para crear una tabla llamada FOTOGRAFÍA con las columnas Nombre, Descripción, FechadeToma y NombredeArchivo. Suponga que Nombre tiene char(20), Descripción es varchar(200), FechaTomada es fechacorta (smalldate) y NombreArchivo es char(45). También suponga que se requieren Nombre y FechadeToma. Use Nombre como llave primaria. Establezca el valor predeterminado de Descripción en ('Ninguno').

13.3 Use el Analizador de Consultas SQL para aplicar la instrucción SQL a la pregunta 13.2 para crear la tabla FOTOGRAFÍA en la base de datos MEDIA.

13.4 Abra la base de datos MEDIA usando el Administrador y la ventana de diseño de la base de datos de la tabla FOTOGRAFÍA. En esta ventana agregue una columna FotografíaID y establezca su identidad origen en 300 y el incremento de identidad en 25. Cambie la llave primaria de Nombre a FotografíaID.

13.5 Usando la herramienta de diseño gráfico de tabla, establezca el valor predeterminado de FechadeToma a la fecha del sistema.

13.6 Cree un tipo de datos definido por el usuario llamado Tema con tipo de datos de char(30). Agregue una columna a FOTOGRAFÍA llamada Asunto que esté basada en Tema.

13.7 Cree una regla llamada Sujetos_Válidos que define la siguiente serie de valores: {'Casa', 'Oficina', 'Familia', 'Recreación', 'Deportes', 'Mascotas'}. Una esta regla a los Tipos de Datos Definidos por el Usuario Sujeto.

13.8 Escriba instrucciones ALTER para:

a. Cambiar la longitud de Nombre a 50

b. Eliminar la columna FechadeToma

c. Agregar la columna TomadoPor a (40)caracteres

13.9 Transmita su respuesta a la pregunta 13.8 a SQL Server mediante el Analizador de Consultas SQL. Después, abra la ventana de diseño de tabla para verificar qué cambios ocurrieron correctamente.

13.10 Cree la tabla MOSTRAR-DIAPOSITIVA (MostrarID, Nombre, Descripción, Propósito). Suponga que MostrarID es una llave sustituta. Establezca el tipo de datos de Nombre y Descripción; sin embargo, considere lo que sea más apropiado. Establezca el tipo de datos de Propósito para Tema. Use una instrucción CREATE o la herramienta de diseño gráfico de tabla.

13.11 Cree la tabla MOSTRAR-FOTOGRAFÍA-INT como una tabla de intersección entre FOTOGRAFÍA y MOSTRAR-DIAPOSITIVA.

13.12 Cree relaciones apropiadas entre FOTOGRAFÍA y MOSTRAR-FOTOGRAFÍA-INT, y entre MOSTRAR-DIAPOSITIVA y MOSTRAR-FOTOGRAFÍA-INT. Establezca las propiedades de integridad referencial para anular cualquier eliminación de un renglón MOSTRAR-DIAPOSITIVA que tenga cualquiera de los renglones MOSTRAR-FOTOGRAFÍA-INT que lo relacionen con éste. Establezca las propiedades de la integridad referencial a eliminaciones en cascada cuando se elimina una FOTOGRAFÍA.

13.13 Explique cómo establecer la propiedad de Actualización en Cascada para las relaciones en la pregunta 13.12.

13.14 Escriba una instrucción SQL para crear una vista con el nombre EspectáculosPopulares que tenga MOSTRAR- DIAPOSITIVA.Nombre y FOTOGRAFÍA.Nombre para todos los espectáculos que tengan un Propósito de “Casa” o “Mascotas”. Ejecute esta instrucción usando el Analizador de Consultas SQL.

13.15 Abra la herramienta de diseño de vista y determine que EspectáculosPopulares se construyó correctamente. Modifique esta vista para incluir Descripción y NombredeArchivo.

13.16 ¿La instrucción SQL DELETE puede usarse con la vista de EspectáculosPopulares? ¿Por qué? Explique.

13.17 ¿En qué circunstancias se puede usar EspectáculosPopulares para inserciones y modificaciones?

13.18 Cree un índice sobre la columna Propósito. Para hacerlo use la herramienta de diseño gráfico de administrador de índices.

13.19 En la figura 13-21, ¿con qué fin se usa la variable @Cuenta?

13.20 ¿Por qué es necesaria que la instrucción SELECT que comienza con SELECT @Cid?

13.21 Explique cómo cambiaría el procedimiento almacenado en la figura 13-21 para poner en contacto al cliente con todos los artistas que: (a) hayan nacido antes de 1900, o (b) tengan un valor nulo de FechadeNacimiento.

13.22 Explique la finalidad de la transacción en la figura 13-24.

13.23 ¿Qué pasa si se introduce un valor incorrecto de Copia en el procedimiento almacenado de la figura 13-24?

13.24 Explique por qué el disparador de la figura 13-27 es repetitivo. ¿Qué puede impedirlo?

13.25 A partir de la figura 13-28, explique por qué un valor @Cuenta de 2 o mayor significa que se cometió un error antes de la invocación de On_WORK_Insert.

13.26 En la figura 13-28, ¿qué pasa si se ejecuta la instrucción ROLLBACK?

13.27 ¿Cuáles son los tres factores principales que influyen en el comportamiento de locking de SQL Server?

13.28 Explique el significado de cada uno de los cuatro niveles de aislamiento de transacción listados en la figura 13-31.

13.29 Explique el significado de cada uno de los valores de concurrencia del cursor listados en la figura 13-31.

13.30 ¿Cuál es la finalidad de las sugerencias de locking?

13.31 ¿Cuál es la diferencia entre respaldo completo y diferencial? ¿Bajo qué condiciones se prefieren los respaldos completos? ¿Cuándo se prefieren los respaldos diferenciales?

13.32 Explique las diferencias entre simple, completo y modelos de recuperación de bulk-logged. ¿En qué situaciones elegiría cada uno?

13.33 ¿Cuándo se necesita restaurar un punto en el tiempo?

➤ PROYECTOS

Para los siguientes proyectos use SQL Server para crear la base de datos de la Galería View Ridge, como se describió en este capítulo. Considere la vista de la base de datos llamada Vista de Artista que se muestra para el Grupo II de preguntas al final del capítulo 10.

A. Escriba un procedimiento Transacción/SQL para leer la parte de la tabla ARTISTA de esta vista. Despliegue los datos que leyó en el Analizador de Consultas como se mostró en el capítulo. Acepte el nombre del artista como un parámetro de entrada.

B. Escriba un procedimiento Transacción/SQL para leer ARTISTA, TRANSACCIÓN y CLIENTE (en TRANSACCIÓN) y la parte de la tabla de esta vista. Despliegue los datos que leyó mediante el Analizador de Consultas. Acepte el nombre del artista como parámetro de entrada.

C. Escriba un procedimiento de Transacción/SQL para leer todas las vistas de las tablas. Despliegue los datos que leyó mediante el Analizador de Consultas. Acepte el nombre del artista como parámetro de entrada.

D. Escriba un procedimiento Transacción/SQL para asignar el interés de un nuevo cliente en un artista. Suponga que se introducen los nombres del artista y del cliente. Si el nombre del cliente no es único, muestre un mensaje de error. Verifique el duplicado en la tabla CLIENTE_ARTISTA_INT antes de que inserte el nuevo renglón. Despliegue un mensaje de error si hay un duplicado.

E. Codifique un DISPARADOR AFTER que verifique las inserciones y modificaciones en Nacionalidad del ARTISTA. Si el valor nuevo se ha asignado como 'Británico' cámbielo a 'Inglés'.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Use SQL Server para crear una base de datos con las cuatro tablas siguientes:

CLIENTE (ClienteID, Nombre, Teléfono, Correo Electrónico)
ESTUFA (NúmerodeSerie, Tipo, Versión, FechadeFabricación)
REGISTRO (*ClienteID*, *NumerodeSerie*, Rfecha)
ESTUFA_REPARACIÓN (NumerodeFacturadeReparación, NúmerodeSerie, FechadeReparación, Descripción, Costo, *ClienteID)*

Suponga que las llaves primarias de CLIENTE, ESTUFA y ESTUFA_REPARACIÓN son llaves sustitutas. Cree relaciones para imponer las siguientes restricciones de integridad referencial:

- ClienteID de REGISTO es un subconjunto de ClienteID de CLIENTE
- NúmerodeSerie de REGISTRO es un subconjunto de NúmerodeSerie de ESTUFA
- NúmerodeSerie de ESTUFA_REPARACIÓN es un subconjunto de NúmerodeSerie de ESTUFA
- ClienteID de ESTUFA_REPARACIÓN es un subconjunto de ClienteID de CLIENTE

No realice eliminaciones en cascada.

A. Termine sus tablas con datos muestra y despliéguelos.

B. Cree un procedimiento almacenado para registrar una estufa. El procedimiento recibe el nombre del cliente, teléfono, correo electrónico y número de serie de la estufa. Si ya existe el cliente en la base de datos (iguale nombre, teléfono y correo electrónico),

use ClienteID de cliente para el REGISTRO. Además, cree un nuevo renglón CLIENTE para el cliente. Suponga que en la base de datos ya existe una estufa con el número de serie de entrada. De lo contrario, imprima un error y dé roll back (revierta) los cambios a la tabla CLIENTE. Codifique y pruebe su procedimiento.

C. Cree un procedimiento almacenado para registrar la reparación de una estufa. El procedimiento recibe el nombre del cliente, teléfono, correo electrónico, número de serie de la estufa, descripción de la reparación y costo. Suponga que está dando un número de serie válido de la estufa; imprima un mensaje de error y no haga cambios en la base de datos. Use un renglón CLIENTE que ya exista si coinciden nombre, teléfono y correo electrónico; de lo contrario, cree un nuevo registro CLIENTE. Suponga que se debe crear el renglón REPARACIÓN-ESTUFA. Si es necesario, registre la estufa.

D. Cree una vista que contenga todos los datos de FiredUp para un cliente específico. Nombre esta vista ClienteRegistro. Esta vista debe juntarse con los datos CLIENTE, REGISTRO, ESTUFA y REPARACIÓN-ESTUFA. Escriba un procedimiento almacenado que acepte un Nombre del cliente y despliegue todos los datos para el cliente específico.

PARTE VI

PROCESAMIENTO DE BASE DE DATOS EMPRESARIAL

La parte VI aborda las tecnologías empleadas para editar y usar bases de datos empresariales. Desde el capítulo 14 hasta el 16 se describe e ilustra el procesamiento de una base de datos empleando la tecnología de Internet.

Tales aplicaciones emplean tecnología desarrollada primero para Internet, pero actualmente se usan tanto en Internet como en intranets privadas. En el capítulo 14 se establecen las bases para describir los ambientes de red, arquitecturas de tres capas y *n* capas y XML. En el capítulo 15 se ilustra el empleo de la tecnología de Microsoft para publicar las aplicaciones de la base de datos empleando OLE DB, ADO y las páginas Active Server. El capítulo 16 ilustra el empleo de JDBC, Java y el servidor Java de páginas para editar estas aplicaciones. Por último, el capítulo 17 cierra la parte VI con un análisis de arquitecturas empresariales adicionales, aplicaciones OLAP y administración de datos.

Muchas de las tecnologías descritas en esta parte han evolucionado rápidamente. Consulte www.oracle.com para mayor información concerniente a las nuevas tecnologías de Oracle; www.microsoft.com, para obtener mayor información acerca de la nueva tecnología de Microsoft; www.w3.org para información reciente de XML y tecnologías relacionadas; y www.sun.com/java para la información más reciente sobre las tecnologías de Java. Estos sitios son sólo lugares para comenzar; también busque en la red porque con frecuencia aparecen nuevos sitios.

CAPÍTULO 14

Redes, arquitecturas multicapa y XML

Internet ha dado origen a las tecnologías que se emplean en la actualidad para publicar las aplicaciones de bases de datos, tanto en Internet como en intranets privadas. Comenzamos con un pequeño resumen de los ambientes de red, y después describimos las arquitecturas multicapa más comúnmente usadas para contar con estas aplicaciones. A continuación consideramos el papel de HTML y DHTML, y después concluimos el capítulo con un análisis del XML. Como aprendimos, el esquema XML es particularmente importante para el proceso de bases de datos.

➤ ENTORNOS DE RED

Una **red** es un conjunto de computadoras comunicadas entre sí que usan un protocolo estandarizado. Algunas redes son **públicas;** cualquiera puede utilizar la red pagando una cuota al proveedor que da el acceso(por ejemplo como estudiantes, que ingresan a una organización que ya ha pagado por tener acceso). Otras redes son **privadas,** así que sólo los usuarios que están preautorizados a conectarse a la red pueden tener acceso. La más extensa red pública es Internet y comenzaremos con ella.

INTERNET

Internet es una red pública de computadoras que se comunican empleando un protocolo de comunicación llamado **Programa de Control de la Transmisión/Protocolo de Internet (Transmission Control Program/Internet Protocol, TCP/IP).** Esta red fue creada por la Agencia de Proyectos de Investigación Avanzada (Advanced Research

Projects Agency, ARPA) de las fuerzas armadas de Estados Unidos, en la década de 1960. Inicialmente fue llamada ARPANET y conectó a los mayores centros de computación de instituciones militares, universidades y centros de investigación. Con el tiempo, más y más organizaciones se unieron a la red; dejó de ser útil únicamente para la investigación y se sumaron organizaciones netamente comerciales. Actualmente se llama Internet.

En 1989 Tim Berners-Lee, del Laboratorio Europeo de Física(CERN), comenzó a trabajar en un proyecto que capacitaría a los investigadores para compartir sus trabajos a través de Internet. Este proyecto guió al desarrollo del protocolo de transferencia de hipertexto **(Hypertext Transfer Protocol, HTTP),** basado en TCP/IP, que permite compartir documentos vinculados a otros publicados en redes TCP/IP.

Hay dos características del protocolo HTTP que son importantes para las aplicaciones de bases de datos: primero, HTTP está *orientado a peticiones*. Los servidores HTTP esperan que lleguen las peticiones; cuando esto sucede, el servidor HTTP realiza alguna acción y entonces, posiblemente, genera una respuesta. A diferencia de algunas aplicaciones MIS tradicionales, éstas no solicitan la actividad de aplicación.

Segunda, las aplicaciones HTTP no mantienen un estado. Los servidores HTTP reciben una pregunta, la procesan y luego olvidan cuál cliente la formuló. No se hace ningún intento para continuar una sesión o conversación con un cliente determinado. Esta característica de no mantener estados no presenta problemas para las aplicaciones de la red que entregan un contenido estático para desplegar páginas de material promocional y similares.

Sin embargo, *es* un problema para las aplicaciones de bases de datos, porque involucran el procesamiento de transacciones, que pueden requerir varios o muchos intercambios entre el cliente y el servidor. Por lo tanto, HTTP debe ser aumentado en capacidades de algún modo para proporcionar un ambiente orientado a estados para la base de datos y otras aplicaciones. Hace algunos años las aplicaciones de la red tuvieron por sí mismas que llevar a cabo una gimnasia algorítmica para mantener el estado. Hoy en día IIS con ASP, y Java servlets con JSP proporcionan mecanismos para mantener su orientación a estados, como se verá más adelante.

Intranet

El término *intranet* tiene varias y diferentes interpretaciones. Por lo general, el término significa red privada, local o red amplia (LAN o WAN), que emplea TCP/IP, HTML y tecnología relacionada con los exploradores en las computadoras de los clientes, y tecnologías de los servidores de la red en los servidores. Con menos frecuencia, el término se usa para significar cualquier LAN o WAN privadas que involucre clientes y servidores.

Hay dos características que diferencian a una intranet de Internet. Primero, las intranets son privadas. Ninguna está conectada a una red pública del todo, o están conectadas a una red pública mediante una **firewall,** una computadora que sirve como puerta de seguridad. Las computadoras firewall revisan y filtran la fuente y el destino del tráfico entre la intranet e Internet. Algunas firewalls operan para permitir sólo cierto tráfico entre ellas; otras operan como prohibición para cierto tráfico, y otras más funcionan en ambos modos.

Debido a que las intranets son privadas, la seguridad es menos preocupante. Esto no significa que no sea una preocupación, sino que las medidas que se requieren son menos elaboradas. En muchos casos las computadoras en una intranet son conocidas y manejadas por la misma organización que posee y mantiene la intranet. Las actividades no autorizadas y los problemas de seguridad son más fáciles de identificar y manejar que en una red pública.

La segunda característica principal que diferencia a una intranet es su velocidad; son rápidas, muchas operan en los rangos de 100000 kbs (miles de bits por segundo). Esto se compara con 56 kbs por usuario conectado a Internet mediante módem tradicional. Incluso los usuarios que han tenido acceso a DSL o a otras tecnologías de módem especializado operan a menos de 500 kbs. Por lo tanto, cualquiera que planee una aplicación de Internet necesita calcular los requerimientos de la transmisión, de acuerdo a la velocidad que tienen disponible los usuarios de las aplicaciones.

Debido a la diferencia de velocidad, pueden incluirse grandes mapas de bits, archivos de sonido y animaciones en las aplicaciones de intranet. Lo más importante en el mundo de la base de datos es que las respuestas a grandes consultas se puedan transmitir a las computadoras de los clientes. Además, debido al incremento de la velocidad, es posible bajar grandes archivos a las computadoras de los clientes. Esto significa, como se verá, que mucho del procesamiento de aplicación se puede dirigir al cliente en una intranet, en lugar de ser enviado a un cliente que está usando Internet.

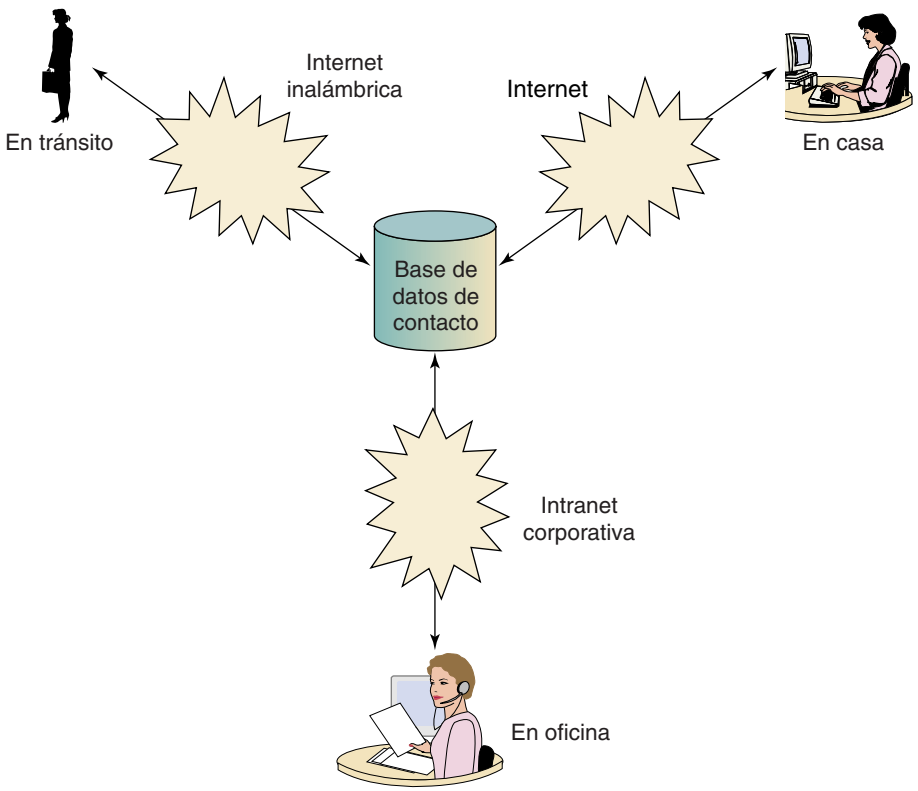
Acceso a las redes inalámbricas

Los usuarios de teléfonos inalámbricos, de asistentes digitales personales inalámbricos y de otros aparatos similares, quieren acceder a Internet y a las intranets organizacionales usando tecnología inalámbrica. Debido a que este tipo de aparatos tienen un tamaño de pantalla limitado, tableros pequeños y poca memoria, el HTML estándar, el XML y otros protocolos son inapropiados para el uso inalámbrico. El **protocolo de aplicación inalámbrica (WAP)** ha sido desarrollado para crear, a escala mundial, el acceso a la red a partir de aparatos inalámbricos. Además, una variante de XML, llamada **Lenguaje de generación inalámbrico** (WML), se ha destinado para desplegar páginas de la red en aparatos inalámbricos.

La figura 14-1 muestra el uso de tres tipos de red para contactar una aplicación administrativa. El usuario puede querer tener acceso a un nombre y teléfono desde su oficina, desde su hogar o desde un teléfono celular mientras se dirige hacia alguna parte. La base de datos a contactar puede estar localizada en su negocio, en su hogar o en un negocio que proporcione ese servicio. Al usuario no le interesa dónde esté localizada la base de datos, sino ponerse en contacto con una fuente de datos en cualquier momento. Más adelante analizaremos WML y WAP, pero antes estudiaremos XML.

➤ FIGURA 14-1

Necesidades de accesos múltiples a los datos de la red



➤ ARQUITECTURA MULTICAPA

La figura 14-2 muestra la arquitectura de tres capas, empleada en muchas aplicaciones tecnológicas de Internet para base de datos. Usaremos muchas siglas en esta figura y en las tres siguientes. No se preocupe si estos términos le resultan extraños. Explicaremos los servidores (como IIS) y los lenguajes de marcaje (como DHTML), y después describiremos la arquitectura general. La base de datos estándar (como ODBC) la explicaremos en los capítulos 15 y 16.

Los tres tipos de procesadores o capas se muestran en la figura 14-2. De derecha a izquierda están los servidores de datos, el servidor de la red y los exploradores o computadoras de los clientes. Como se muestra, cada una de estas capas puede ejecutar diferentes sistemas operativos. En este ejemplo el servidor de la base de datos ejecuta UNIX, el servidor de la red ejecuta Windows 2000 y los exploradores ejecutan el sistema operativo Macintosh. Éstos son sólo ejemplos. Probablemente, los exploradores operarían con una mezcla de Windows 98, UNIX, Macintosh y Windows 2000. El servidor de la Web y el servidor de la base de datos pueden ser UNIX o Windows 2000.

En la figura 14-2 se muestran ejemplos de productos: Oracle en el servidor de base de datos, Servidor de Información Internet de Microsoft (IIS) en el servidor de la red y en el navegador Netscape en los clientes. De nuevo, son sólo ejemplos. El servidor de datos podría ejecutar cualquier DBMS desde Access hasta DB2. Los servidores de la red más comunes son IIS, Servidor Netscape y Apache. Por lo general, los exploradores son navegadores Netscape y exploradores Internet de Microsoft.

Como se muestra, la interfaz entre el servidor de la red y el servidor de la base de datos transmite instrucciones SQL y datos relacionados. La interfaz entre el servidor de la Web y el explorador transmite paginas Web, código del cliente y datos. Diremos mucho más acerca de cada una de estas interfaces en éste y en los siguientes tres capítulos.

Las funciones de estas tres capas se resumen en la figura 14-3. El propósito del servidor de datos es ejecutar el DBMS para procesar las instrucciones SQL y ejecutar las tareas administrativas de la base de datos. Aquí, el DBMS está operando en su función tradicional de servidor de datos; las capacidades de las herramientas de aplicación que ha visto con Access no están en uso. El DBMS en el servidor de datos en la figura 14-3 no está creando formas, reportes o menús. Más bien, sólo es una máquina de datos que está recibiendo peticiones SQL y procesando renglones en las tablas, como se mencionó en capítulos anteriores.

La figura 14-3 muestra los estándares y los programas de aplicación de interfaces que se usan entre el servidor de datos y el servidor de la Web. Analizaremos ODBC, OLE DB y ADO en el capítulo 15, y JDBC en el capítulo 16. Por ahora, sólo pensaremos en los estándares como medios de transmisión de SQL y las relaciones entre el servidor de la red y el servidor de la base de datos.

➤ FIGURA 14-2

Arquitectura de tres capas

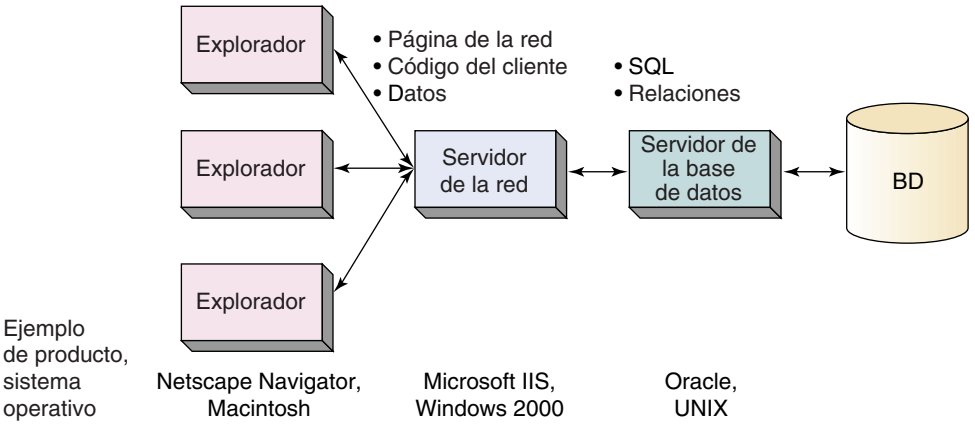
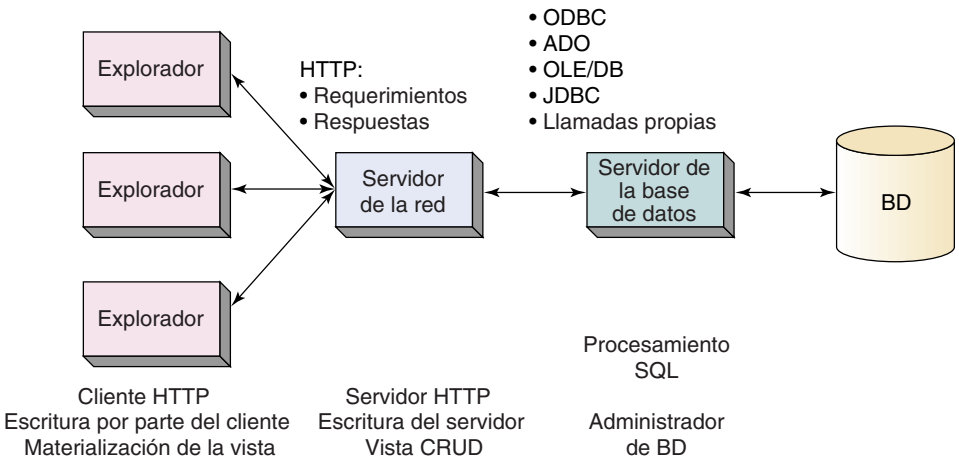


FIGURA 14-3

Funciones de las capas



El servidor de la red realiza tres funciones principales. Primero, es un servidor HTTP, lo que significa que procesa el protocolo HTTP, recibe requerimientos y genera respuestas en formato HTTP. El servidor de la red también mantiene ambientes para lenguajes script, así que los programadores pueden escribir código en lenguajes como VBScript y JavaScript y ejecutar el código en el servidor de la red. Por último, en aplicaciones de bases de datos, la tercera función del servidor de la red es crear, leer, actualizar y eliminar instancias de vistas, como vinos en el capítulo 10. Nuevamente hay que recordar la diferencia entre una relación y una vista; las vistas son construcciones de relaciones.

En las aplicaciones de bases de datos, el explorador tiene tres funciones que son análogas a las del servidor de la red. Primero, el explorador es un cliente HTTP que genera requerimientos con respecto a páginas y otras actividades. También alberga un medio ambiente de lenguaje script para ejecutar scripts en la máquina del cliente. Por último, el explorador materializa vistas, transformando HTML u otro lenguaje de marcaje dentro de una pantalla en la ventana del explorador del cliente.

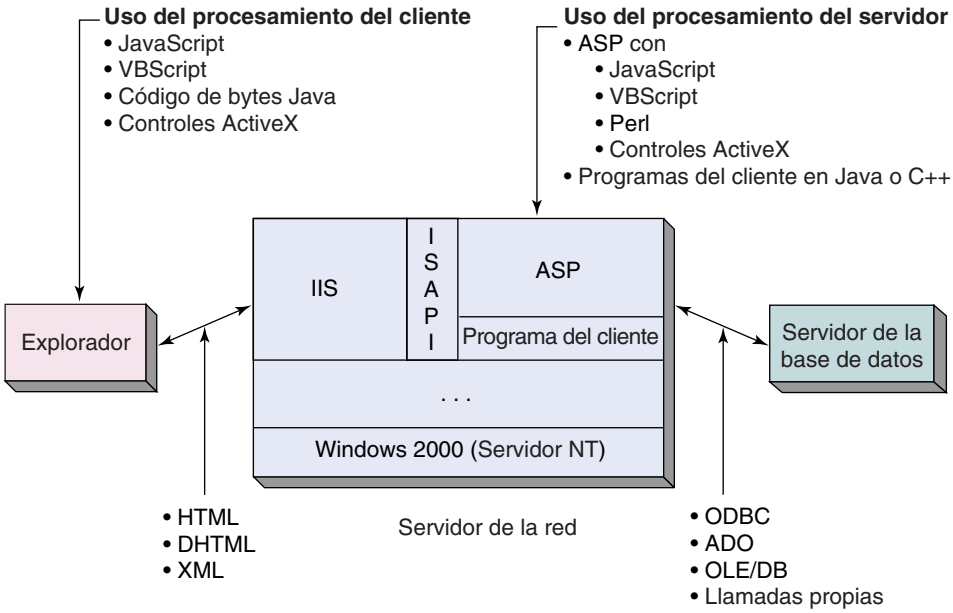
Esta arquitectura permite escribir en el servidor de la red y en la computadora del cliente. Una de las tareas importantes cuando se desarrolla una aplicación tecnológica de Internet es decidir qué trabajo será hecho y en qué máquina. Debido a que hay muchos exploradores y un solo servidor, es preferible colocar tanto código en el cliente como sea posible. Sin embargo, si éste necesita volver al servidor de la red para obtener datos para sus cálculos, esta ventaja se puede contrarrestar por el tiempo que se necesita para el requerimiento de datos y las respuestas completas. También se debe enviar al explorador el código del cliente en el mensaje HTTP. Grandes partes del código requerirán mayor tiempo para descargar. En la figura 14-3 se muestra una buena forma empírica: usar el código del cliente para las tareas de escritura que mantienen vistas de materialización. Use el código del servidor para las tareas de escritura que crean la vista CRUD. Asimismo, el código para cálculos inmediatos (Importe = Cantidad * Precio) por lo general se procesa en el explorador.

Ambiente de servidor WEB Windows 2000

La figura 14-4 muestra los estándares y lenguajes que son comunes cuando el servidor Web está ejecutando Windows 2000 como sistema operativo. En este caso, el servidor HTTP será casi siempre IIS, puesto que es parte del sistema operativo de Windows. IIS proporciona una interfaz llamada Interfaz de programa de aplicación del servidor Internet (ISAPI, por sus siglas en inglés), mediante el cual se pueden atrapar otros programas y procesar mensajes HTTP. El Procesador de servidor activo (ASP) es un programa que procesa todas las páginas de la red con el sufijo *.asp*. Cuando IIS recibe estas páginas, las manda al Procesador de servidor activo sobre la interfaz ISAPI. ASP procesa entonces la página y manda una respuesta al cliente mediante la interfaz ISAPI a IIS.

FIGURA 14-4

Estándares y lenguajes comunes con el servidor Web de Microsoft



ASP hospeda lenguajes script; por lo tanto, las páginas ASP pueden contener JavaScript, VBScript, Perl y otros estatutos de lenguaje script. Éstas se ejecutarán cuando se procese la página ASP. Además, los controles ActiveX se pueden incluir en la página y se invocarán también.

ASP no es el único programa que puede usar la interfaz ISAPI. C++ y los programadores de Java pueden crear programas para procesar mensajes HTTP en vez de usar la facilidad ASP.

Cuando se usa un servidor Web de Microsoft el estándar de acceso de datos será similar a ODBC, ADO u OLE DB, ya que estos estándares son apoyados y difundidos por Microsoft.

Ambiente de servidores de la red Unix y Linux

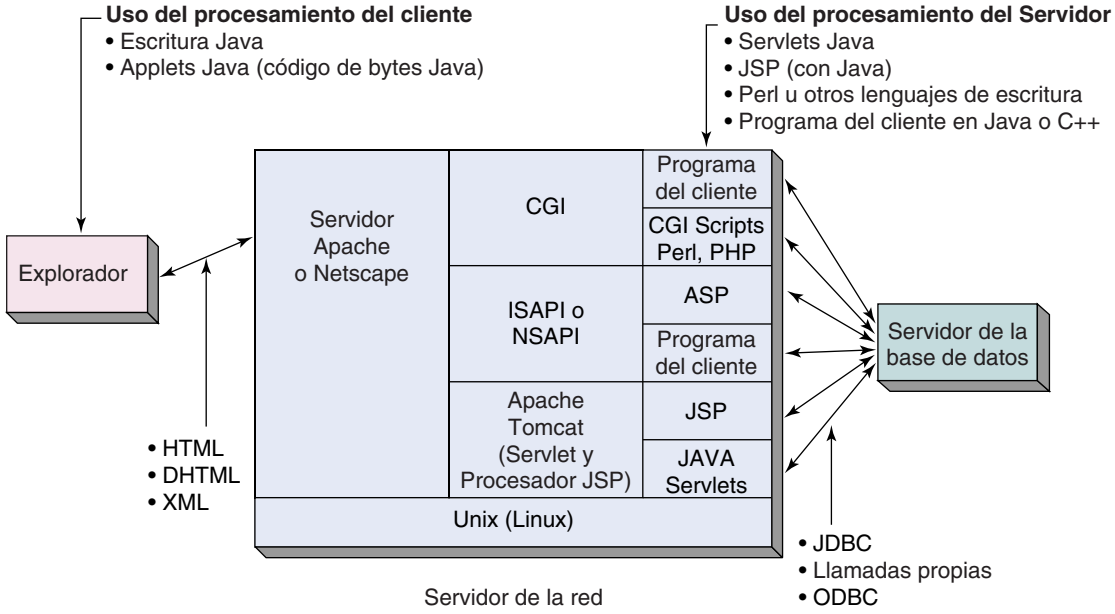
Cuando se tiene un servidor Web Unix o Linux (figura 14-5), IIS no se utiliza como servidor HTTP, sino que se usan productos como Apache y los servidores de Web Netscape. Apache es el servidor Web más comúnmente usado en el mundo, en parte porque es gratis. Se puede encontrar información acerca de Apache en www.apache.org.

En el pasado, la **interfaz de compuerta común** (common gateway interface, o CGI) fue la más usada para aplicaciones que procesaban requerimientos HTTP en servidores Apache y similares. El problema con CGI estriba en que una copia del programa procesado se carga dentro de la memoria de cada usuario. Si 100 usuarios están accediendo al mismo script CGI, entonces hay 100 copias de script cargadas en memoria. Esta situación llega a ser insostenible por la carga de trabajo en sitios Web comerciales. Existen otros problemas con CGI, como veremos en el capítulo 16.

Actualmente, para las aplicaciones de procesamiento de bases de datos, es más común usar los **servlets Java** o las páginas de **Java Server** (Java Server Pages JSP), o ambos. Estas tecnologías requieren software que implemente los servlets Java o las especificaciones de páginas del Java Server. El Apache básico no lo hace, pero la organización Apache ha facilitado el desarrollo de Apache Tomcat, un producto que implementa estas especificaciones. Tomcat, al igual que Apache, es gratis (www.jakarta.tomcat.org). Se puede ejecutar Tomcat como servidor Web autónomo para probar sus aplicaciones. También puede instalarse ejecutando en conjunto con Apache para aplicaciones Web de alto desempeño. El paquete de desarrollo para web de Java Server es un segundo producto gratuito que se puede usar en forma autónoma para probar los servlets y las

➤ FIGURA 14-5

Estándares y lenguajes comunes con el servidor Unix



páginas JSP. Hay otros productos también disponibles. Busque en Web por Java Servlet 2.2 y páginas de Java Server 1.1. Analizaremos el uso de estos productos con detalle en el capítulo 16.

Por otra parte, hay una diferencia importante entre las tecnologías ASP y JSP. Con ASP se puede ejecutar cualquier lenguaje script manejado como VBScript y JScript. Sin embargo, no se puede colocar directamente C++ o Java dentro de una página ASP. En contraste, con JSP toda la codificación se hace en Java; incluso si se escriben fragmentos de código, serán instrucciones de Java.

Apache y otros servidores Unix del Web también manejan ISAPI y hay versiones de ASP que pueden funcionar con estos productos. Por supuesto, si las páginas ASP usan ActiveX y otro componente Microsoft, no se ejecutarán correctamente (o cuando menos no por completo). Los servidores Netscape proporcionan su propia interfaz, NSAPI, que desempeña una función similar a la de ISAPI.

JDBC es más comúnmente usado cuando se accede a la base de datos desde los programas Java. También hay puentes de JDBC a ODBC que facilitan a los programas Java utilizar las unidades de disco ODBC. Además, los productos DBMS también se pueden llamar mediante bibliotecas nativas. Ni OLE DB ni ADO se implementan en Unix.

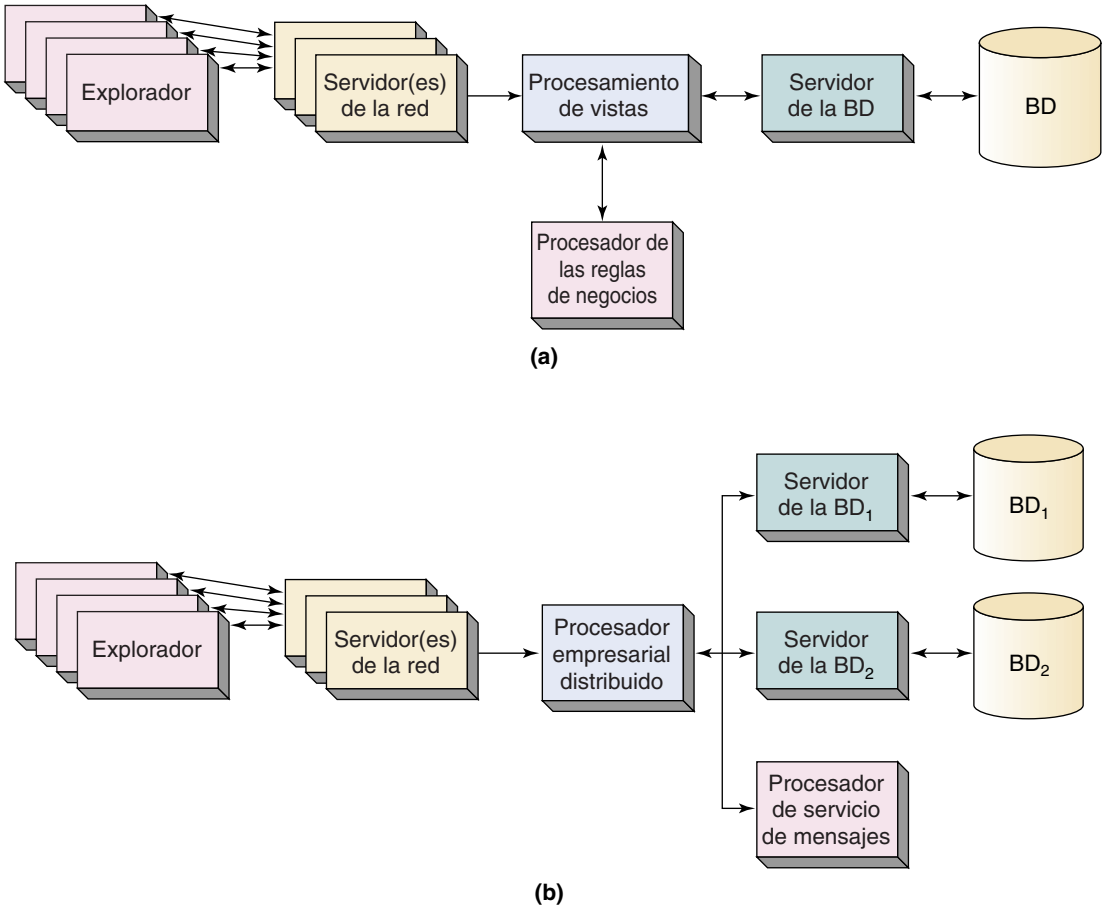
Anteriormente mencionamos que HTTP es un protocolo que no mantiene estados, lo que significa que el servidor procesa un requerimiento y después olvida al solicitante. Un procesamiento de este tipo es inaceptable para la mayoría de aplicaciones de base de datos, y para la habilidad de rastrear el estado de una sesión que se ha agregado a IIS para Windows 2000, o a Tomcat y procesadores servlet similares en el mundo Unix/ Linux. En los dos capítulos siguientes veremos cómo se hace esto.

Procesamiento multicapas

Para algunas aplicaciones, la arquitectura de tres capas pone una carga no razonable en el servidor de la Web. Recuerde que este servidor no sólo funciona como servidor HTTP respondiendo potencialmente a miles de requerimientos, sino que también crea y administra conexiones al servidor de la base de datos, genera y transmite requerimientos SQL, acumula resultados SQL dentro de las vistas, ejecuta lógica de negocios, e impone

➤ FIGURA 14-6

Ejemplos de arquitectura multicapas: (a) uso de procesadores múltiples para el procesamiento de vistas y reglas, y (b) uso de procesadores múltiples para el procesamiento distribuido



reglas del negocio. Después de todo esto, en el explorador formula una respuesta para el usuario.

Con una carga como ésta, el servidor de la red puede llegar a ser un cuello de botella. Una solución para este problema es balancear la carga de trabajo a través de muchos, quizá cientos, servidores de la red. Sin embargo, esta solución puede crear su propio conjunto de problemas, debido a que requiere que cada uno de los servidores de la red tenga todos los programas necesarios para realizar las funciones antes descritas. Esto puede llegar a ser un dolor de cabeza administrativo. Otra solución es utilizar el (los) servidor(es) Web sólo para las peticiones HTTP y el procesamiento de respuestas, y descargar el procesamiento de las vistas, el procesamiento de las reglas de negocios, etc., a otros servidores. Esta solución se muestra en la figura 14-6(a), donde las computadoras por separado procesan vistas e imponen reglas de negocios.

Un problema parecido ocurre cuando el procesamiento de una sola transacción está distribuido entre múltiples computadoras. Por ejemplo, en la figura 14-6(b) suponga que los datos de una orden residen en la base de datos BD_1 y los datos del cliente están en la base de datos BD_2. Además suponga que para un procesamiento de órdenes normal se necesita actualizar ambas bases de datos y enviar los mensajes a producción y los acuses de recibo antes de que la orden pueda ser confirmada. La solución que se muestra en la figura 14-6(b) es dedicar a un servidor que administre las transacciones distribuidas, mientras que otros servidores procesan las bases de datos y proporcionan servicios de mensajes.

Obviamente, hay muchas posibilidades. Conforme las organizaciones obtienen más experiencia en el procesamiento comercial de la red, surgen ciertos patrones de procesamiento multicapa. Por el momento, sólo sabemos que el procesamiento de la base de datos Web no se restringe a tres capas, y que puede haber más de un servidor compartiendo la carga de trabajo en cada capa.

➤ LENGUAJES DE MARCAJE HTML Y DHTML

Los lenguajes de marcaje se usan para especificar la apariencia y el comportamiento de una página de la red. Como se podrá ver, el primer lenguaje importante en la red fue HTML, que es un subconjunto de un lenguaje más complejo y poderoso llamado SGML. En esta sección consideraremos los lenguajes de marcaje dinámico HTML (DHTML) y XML. Sin embargo, antes de profundizar en esto conviene establecer un contexto histórico de los estándares o normas de la red.

Estándares de lenguajes de marcaje

HTML tuvo un éxito tremendo al fomentar el desarrollo de cientos de miles de sitios Web, e hizo de la comunicación Web una realidad para la computación pública en general. Sin embargo, con el tiempo fue obvio que la versión original de HTML tenía importantes desventajas, especialmente para el desarrollo de las aplicaciones de base de datos en Web. Conforme se trabajaba para superar estas desventajas, fue evidente que se requerían normas. De lo contrario, cada proveedor haría sus propias mejoras y pronto sólo ciertos exploradores podrían procesar ciertas páginas, lo que eliminaría una de las más importantes ventajas de la red.

En consecuencia, a principios de 1994 el Consorcio de la Red Mundial (Worl Wide Web Consortium, W3C) comenzó a promulgar estándares o normas para el HTML y otros lenguajes de marcaje. Estos estándares han llegado a ser muy importantes. Visite el sitio del consorcio de la red en www.w3.org y encontrará información útil, tutoriales, estándares de aceptación (o de recomendación), así como las normas que están en proceso de desarrollo.

La función más importante de los estándares será más clara si entiende que los proveedores de productos de software tienen amor-odio a relacionarse con ellos. Por una parte, les agradan los estándares porque proporcionan un orden en el mercado y establecen un conjunto básico de capacidades que los productos deben tener. Por otra, los rechazan porque pueden invalidar características, funciones y tecnologías en las cuales han hecho una inversión considerable.

Asimismo, los estándares ponen a los proveedores en un doble aprieto. Si únicamente desarrollan los estándares no hay razón para que los clientes prefieran sus productos a los de la competencia y los compren. Pero si les agregan características y funciones a los estándares, entonces serán criticados por no apegarse a las normas. Así, cuando Microsoft desarrolló DHTML, apoyó el estándar W3C llamado HTML 4.0. Sin embargo, también agregó características y funciones que van más allá de los estándares. Dependiendo de cuál use, esas características son maravillosas porque dan mayor funcionalidad; o son terribles porque las páginas que usan pueden no comportarse adecuadamente en todos los exploradores.

Por esta razón, cuando se agregan características no estándares a un producto, se hace de tal forma que los productos que no manejen esas características adicionales quedarán disminuidos en forma notoria. Por ejemplo, un esquema abrirá y cerrará dinámicamente en los exploradores que usen esa característica y permanecerán siempre abiertos en exploradores que no la sustenten.

Problemas con HTML

La versión original de HTML sufrió varios problemas. Para unos el contenido, la presentación y el formato de las páginas eran confusos; no hay manera de separar la defi-

nición del contenido de la presentación del formato. Esto significa que si se desea el mismo contenido en diferentes formas, se necesita crear dos copias completas de ese contenido, presentarlas y darles formato. Para las aplicaciones de base de datos también significa que la vista de los datos y la materialización de la vista están mezcladas de manera extremadamente confusa.

Otra desventaja era la carencia de definiciones de estilo. Por ejemplo, no era posible decir que todos los encabezados tendrían una fuente en particular, un tamaño y un énfasis. Por lo tanto, si un programador de la red deseba cambiar el formato de todo un nivel de encabezados, tendría que encontrar todos los niveles de los encabezados en la página (o en el sitio de la red) y después cambiar su formato uno por uno. Lo que se necesitaba era una forma de definir el estilo de un elemento de determinado formato.

Una tercera desventaja del HTML original era que no se podía acceder a los elementos de la página de la red a partir de scripts o de otros programas. Por ejemplo, no había manera de remitirse al elemento de un título de una página en un script para modificar dinámicamente su apariencia o su presentación. Tampoco se podía escribir un código para responder a eventos de Windows, tales como los movimientos del ratón y los clics. Por lo tanto, para cambiar la estructura o la presentación de una página el explorador necesitaba regresar al servidor para obtener una nueva página completa. Estos viajes redondos eran innecesarios, lentos y un desperdicio.

Por último, y lo más importante para nosotros, no había construcciones para facilitar el manejo del caché (memoria de acceso rápido) y el manejo de datos por parte del cliente. Siempre que un explorador necesitaba desplegar más datos, se tenía que hacer un viaje redondo al servidor. W3C desarrolló un nuevo estándar del HTML, llamado HTML 4.0, para definir características y funciones para superar estas y otras desventajas.

DHTML

DHTML es una implementación Microsoft de HTML 4.0. Incluye todos los estándares, características y funciones adicionales. El DHTML es empleado por Internet Explorer 4.0 de Microsoft (y versiones posteriores), y por IIS 4.0 (y versiones posteriores). El Navegador Netscape maneja únicamente HTML 4.0, que es una parte del DHTML; éste no usa todas las características que se agregaron a DHTML más allá de esos estándares. Por lo tanto, cuando se usa DHTML los programadores necesitan tener cuidado de mantenerse dentro del conjunto de características HTML 4.0, si sus usuarios verán la aplicación a través de un explorador que no sea de Microsoft.

Hay varias características claves de DHTML que superan las desventajas de las versiones anteriores. En primer lugar, DHTML proporciona un objeto modelo llamado el Modelo de Objeto Documento (DOM), el cual muestra todos los elementos de la página como objetos. Dichos objetos se pueden manejar con un script para cambiar los atributos de los elementos, e invocar métodos de los elementos. Así, se puede mostrar un esquema empacado y después abrirlo completamente cuando el usuario hace clic en él.

Gracias a DOM el contenido de la página, la presentación y el formato se pueden modificar a través de un programa sin regenerar la página desde el servidor. No sólo porque esto ahorra tiempo, sino porque el usuario no se confronta con páginas cambiantes y destellantes cada vez que cambia un encabezado, sino que únicamente se modifica la parte del texto que él cambió.

Otra característica clave de HTML 4.5 y de DHTML es que manejan las Hojas de Estilo en Cascada (CSS, por sus siglas en inglés). Esto crea formatos para definir los tipos de elementos en una página. El siguiente código DHTML fija el fondo y el color de un nivel y dos encabezados.

```
<STYLE TYPE="text/class">
<!—
H1 {font-family:Lucida; font-style:normal; color:black}
H2 {font-family:Lucida; font-style:normal; color:green}
—>
</STYLE>
```

En el documento, cuando en un nivel se encuentran uno o dos encabezados, aparecerán como definidos por el elemento de estilo. Así, en el siguiente:

```
<H2>
Éste es un ejemplo:
</H2>
```

nivel de los dos encabezados "Éste es un ejemplo:" aparecerá en verde, en el tipo de letra Lucida, normal. En este caso, ambos estilos se refieren a las etiquetas estándares HTML (H1 y H2). También es posible definir estilos para etiquetas definidas por el programador y después insertarlas en el documento cuando sea necesario.

Con DHTML es posible que un elemento tenga dos estilos en conflicto. Por ejemplo, un estilo de una hoja puede especificar un formato particular para todos los párrafos, pero en una página en particular se puede marcar un párrafo que tiene un estilo diferente. En este caso, el estilo del párrafo predominará en el estilo de la hoja. Como regla general, se usarán las marcas de estilo más cercanas al contenido. Esta característica es la que causa que las hojas de estilo se denominen hojas de estilo *en cascada*.

Las hojas de estilo pueden estar dentro de la página, o se pueden obtener de manera externa, de otros documentos que contengan las definiciones de estilo. Así, con DHTML el contenido y la materialización se pueden separar.

La combinación de las hojas de estilo y del Modelo de Objeto Documento (DOC, por sus siglas en inglés) significa que las páginas Web pueden cambiar sin regeneración de página desde el servidor. Por ejemplo, se puede cambiar el estilo de los encabezados de nivel dos conforme el ratón se mueva en los encabezados del nivel uno; incluso se pueden hacer cambios más drásticos.

Paralelamente al desarrollo de DHTML, Microsoft creó un conjunto de controles ObjetoX llamado **Servicios de Datos Remotos** (RDS, por sus siglas en inglés). Con estos controles un programador puede ocultar datos de la base de datos de los clientes, desplegarlos usando un control de datos, aceptar actualizaciones en el control de datos y enviar el procesamiento por lotes al servidor como actualización de la base de datos.

Sin embargo, en realidad RDS sólo es útil para vistas que constan de una sola tabla. La actualización de vistas más complejas puede requerir la ejecución de dos, tres o más instrucciones SQL, como vimos en el capítulo anterior. Esta clase de actualización de tablas múltiples no es posible con RDS. Para ello, los programadores usan ADO.

Aunque RDS en sí mismo no es tan importante para nosotros, sí nos interesa DHTML para los cambios dinámicos en la estructura de la página Web. Por ejemplo, con DHTML se puede consultar una vista en la base de datos, determinar en forma programática el número y los nombres de las columnas en la vista, y configurar dinámicamente una página de la red que despliegue esos datos.

DHTML y el HTML 4.0 corrigen muchas de las deficiencias del HTML original, pero conservan la misma estructura y carácter fundamentales. Ahora abordaremos una especificación que tiene una naturaleza fundamentalmente diferente.

➤ XML: LENGUAJES DE MARCAJE EXTENSIBLE

XML es uno de los desarrollos más importantes en los sistemas de información de los últimos 10 años. Por una parte, proporciona un estándar extensible para la materialización de documentos en páginas de la red. Incluso ha llegado a ser importante para el intercambio de datos; es particularmente útil para la transmisión de vistas de la base de datos. XML es también sencillo, cuando menos sus estructuras básicas lo son, y en consecuencia se usa para expresar muchos tipos de textos estandarizados. Por ejemplo, Tomcat, el procesador Java servlet para Apache, usa XML para sus archivos de configuración.

XML incluso se usa como estándar para ejecutar llamadas de procedimiento remoto. DCOM y CORBA compitieron durante años para ser el estándar, hasta que se definió el **Protocolo Access de Objeto Simple** (Simple Object Access Protocol, SOAP). SOAP es

simplemente un medio para transmitir procedimientos de llamadas expresados como pequeños documentos XML usando HTTP. Indudablemente en el futuro se encontrarán muchos usos más para XML.

XML es un tema tan amplio que amerita un libro. Aquí tendremos que restringirnos al uso de XML para la materialización de páginas de la red y expresar vistas de bases de datos. Vea el excelente material de www.w3.org y www.xml.org para mayor información. Puede dar un gran impulso a su profesión si aprende lo más posible acerca del XML.

XML como un lenguaje de marcaje

Como lenguaje de marcaje, XML es mucho más importante que HTML o DHTML. Hay varias razones para la superioridad de XML. Una de ellas se refiere a que los diseñadores de XML crearon una clara separación entre la estructura del documento, el contenido y la materialización. XML tiene dispositivos para usar cada uno, y la naturaleza de éstos es tal que no se pueden confundir, como sucede a veces con HTML.

Además, XML está estandarizado pero, como su nombre lo indica, los programadores pueden ampliar el estándar. Con XML no se está limitado a un conjunto fijo de elementos como <TÍTULO>, <H1>, <P>, sino que se pueden crear los propios.

Uno de los problemas con HTML y DHTML es que existe demasiada libertad. Considere el siguiente HTML:

```
<h2>Hola Mundo </h2>
```

Esta etiqueta <h2> se puede usar para marcar un nivel de dos encabezados en un bosquejo. Pero también se puede usar simplemente para hacer que “Hola Mundo” se despliegue con un estilo en particular. Debido a esta característica, no podemos confiar en etiquetas para indicar la verdadera estructura de una página HTML. El uso de la etiqueta es también arbitrario; <h2> puede significar un encabezado, o tal vez no signifique nada.

Como se verá, con XML la estructura de un documento se define formalmente. Si encontramos la etiqueta <calle>, sabemos exactamente a dónde pertenece y cómo se relaciona con la estructura de otras etiquetas. Así, los documentos XML representan exactamente la semántica de sus datos.

Declaraciones de tipo de documentos XML

La figura 14-7 contiene un documento muestra de XML. Observe que tiene dos secciones. La primera define la estructura del documento, se denomina declaración de tipo de documento o **DTD,** y la segunda son los datos del documento.

DTD empieza con la palabra DOCTYPE y especifica el nombre de este tipo de documento, que es el cliente (customer). Después, invoca el contenido del documento del cliente, que consta de dos grupos: nombre y dirección (name y address). El nombre del grupo consiste en dos elementos: Nombre y Apellidos (firstname, lastname), que se definen como #PCDATA, lo cual significa que son cadenas de caracteres de datos. A continuación se define el elemento dirección mediante cuatro elementos: calle, ciudad, estado y código postal (street, city, state y zip). Cada uno de éstos también se define como carácter de datos. El signo más (+) después de calle indica que se requiere un valor y que son posibles valores múltiples.

En la figura 14-7 se muestra el ejemplo de datos del cliente, de conformidad con el DTD; aquí se dice que este documento es un **documento XML de tipo válido.** Si no fuera acorde al DTD entonces sería un documento de **tipo no válido.** Los documentos que son de tipo no válido pueden ser perfectamente buenos para XML, sólo que no son exactamente válidos en casos de su tipo. Por ejemplo, si el documento de la figura 14-7 tiene dos elementos city (ciudad), éste aún sería un XML válido, pero de tipo no válido.

Aun cuando los DTD son casi siempre preferibles, no se requieren en documentos XML. Los documentos que no tienen DTD por definición son tipos no válidos, puesto que no se le puede validar con algún otro.

➤ FIGURA 14-7

Ejemplo de un documento XML

```
<!DOCTYPE customer [
 <!ELEMENT customer (name, address)>
        <!ELEMENT name (firstname, lastname)>
                <!ELEMENT firstname (#PCDATA)>
                <!ELEMENT lastname (#PCDATA)>
        <!ELEMENT address (street+, city, state, zip)>
                <!ELEMENT street (#PCDATA)>
                <!ELEMENT city (#PCDATA)>
                <!ELEMENT state (#PCDATA)>
                <!ELEMENT zip (#PCDATA)>
]>
 <customer>
        <name>
                <firstname>Michelle</firstname>
                <lastname>Correlli</lastname>
        </name>
        <address>
                <street>1824 East 7th Avenue</street>
                <street>Suite 700</street>
                <city>Memphis</city>
                <state>TN</state>
                <zip>32123-7788</zip>
        </address>
 </customer>
```

Los DTD no necesitan estar contenidos dentro del documento. La figura 14-8 muestra un documento customer (cliente) en el cual el DTD se obtiene del URL Http://www.somewhere.com/dtds/customer.dtd. La ventaja de almacenar el DTD externamente es que muchos documentos se pueden validar contra el mismo DTD.

El creador de un DTD es libre de escoger cualquier elemento que quiera. De ahí que los documentos XML se puedan extender, pero en una forma estandarizada y controlada. Como se verá, los DTD se pueden usar fácilmente para representar vistas de las bases de datos.

Materialización de documentos XML

Los documentos XML de la figura 14-7 muestran tanto estructuras de documentos como de contenidos. Sin embargo, nada en el documento indica cómo se materializa. Los diseñadores de XML crearon una separación clara entre estructura, contenido y formato.

Existen dos formas de materialización de documentos XML: hojas de estilo en cascada y XSLT. Analizaremos una por una.

MATERIALIZACIÓN XML CON HOJAS DE ESTILO EN CASCADA. En la sección de DHTML analizamos CSS (hojas de estilo en cascada); su uso para documentos XML es similar. Esto es, los estilos se definen para etiquetas y se aplican en forma de cascada. Considere el siguiente estilo de definición del documento en la figura 14-7:

```
<STYLE TYPE ="text/class">
<!—
customer   {font-family:Lucida; font-style:normal; color:black}
```

➤ FIGURA 14-8

Documento XML con un DTD externo

```
<!DOCTYPE customer SYSTEM "http://www.somewhere.com/dtds/customer.dtd>

 <customer>
       <name>
             <firstname>Michelle</firstname>
             <lastname>Correlli</lastname>
       </name>
       <address>
             <street>1824 East 7th Avenue</street>
             <street>Suite 700</street>
             <city>Memphis</city>
             <state>TN</state>
             <zip>32123-7788</zip>
       </address>
 </customer>
```

```
name      {font-family: Lucida; font-style: normal; color: green}
lastname  {font-family: Lucida; font-style: normal; color: red
—>
</STYLE>
```

De acuerdo con esta especificación de estilo, el color predeterminado para el cliente es negro. Sin embargo, el color para los nombres de los elementos pasa del negro al verde, y en los elementos del apellido va del verde al rojo. Por lo tanto, los estilos en cascada pasan de uno a otro.

Actualmente no hay un acuerdo estándar acerca de qué definiciones de estilo se establecen. Se podrían colocar en un documento o en un archivo externo y se colocaría alguna referencia en el documento. Los diferentes productos los implementan de diferentes maneras.

MATERIALIZACIÓN XML CON LENGUAJE DE ESTILO EXTENSIBLE: TRANSFORMACIONES (XSLT). El segundo medio de materialización de documentos XML es el uso de XSLT, o lenguaje de estilo extensible: transformaciones. XSLT es un lenguaje de transformación poderoso y muy amplio. Se puede usar para materializar los documentos XML en DHTML o HTML, así como para muchos otros propósitos. Una aplicación común de XSLT es transformar un documento XML con un formato dentro de un segundo documento XML en otro formato. Por ejemplo, una compañía puede usar XSLT para transformar un documento de pedido con sus propios formatos, en el documento equivalente del pedido XML en el formato de sus clientes. Hay muchas características y funciones de XSLT que estamos imposibilitados de analizar aquí. Vea www.w3.org para mayor información acerca del tema.

XSLT es un lenguaje de transformación declarativo. Es declarativo porque en lugar de especificar un procedimiento para materializar elementos de documentos, se crea un conjunto de reglas que gobiernan cómo se materializan dichos documentos. Es transformable porque transforma el documento de entrada dentro de otro documento.

➤ FIGURA 14-9

Documento XML
Lista de cliente

```
<?xml version='1.0'?>
<!DOCTYPE customerlist [
 <!ELEMENT customerlist (customer+)>
        <!ELEMENT customer (name, address) >
        <!ELEMENT name (firstname, lastname)>
        <!ELEMENT firstname (#PCDATA)>
 <!ELEMENT lastname (#PCDATA)>
<!ELEMENT address (street+, city, state, zip)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
]>

<?xml:stylesheet type="text/xsl" href="CustomerList.xsl" ?>

<customerlist>
 <customer>
  <name>
   <firstname>Michelle</firstname>
   <lastname>Correlli</lastname>
  </name>
  <address>
   <street>1824 East 7th Avenue</street>
   <street>Suite 700</street>
   <city>Memphis</city>
   <state>TN</state>
   <zip>32123-7788</zip>
  </address>
 </customer>

 <customer>
  <name>
   <firstname>Lynda</firstname>
   <lastname>Jaynes</lastname>
  </name>
  <address>
   <street>2 Elm Street</street>
   <city>New York City</city>
   <state>NY</state>
   <zip>02123-7445</zip>
  </address>
 </customer>
</customerlist>
```

En la figura 14-9 se muestra un documento XML customerlist (Lista de cliente) con los datos de dos clientes. Tiene un DTD y es un documento de tipo válido. La instrucción después del DTD invoca a la hoja de estilo XSL llamada CustomerList.xsl, que se muestra en la figura 14-10. El resultado del procesamiento de este documento XML aparece en la figura 14-11.

➤ FIGURA 14-10

Ejemplo de documento XSL

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:14pt;
                                        background-color:teal">

  <xsl:for-each select="customerlist/customer">
        <DIV STYLE="background-color:purple; color:white; padding:4px">
              <SPAN STYLE="font-weight:bold; color:white">
                    <xsl:value-of select="name/lastname"/></SPAN>
                    - <xsl:value-of select="name/firstname"/>
        </DIV>

        <xsl:for-each select="address/street">
          <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt;
                                        font-style:bold; color:blue">
                    <xsl:value-of select="node()"/>
          <DIV>
        <xsl:for-each>

        <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:12pt;
                        font-style:bold"><xsl:value-of select="address/city"/>,
                    <xsl:value-of select="address/zip"/>
        </DIV>

        <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:14pt; color:red">
                    <xsl:value-of select="address/zip"/>
        </DIV>
  <xsl:for-each>

  </body>
</HTML>
```

Esta hoja de estilo usa los dispositivos XSLT en el Microsoft Internet Explorer 5.0, que tiene un procesador XSLT al cual se puede acceder desde la hoja de estilo. Por ejemplo, en la figura 14-10 cualesquiera de los elementos que empiezan con <xls: están invocando al procesador XSLT de Internet Explorer 5.0.

La estructura de una hoja de estilo XSL es de la forma {*match*, *action*}. La idea es que el procesador XSLT buscará el acoplamiento (match) de algún elemento y cuando lo encuentre llevará a cabo la acción indicada. Por lo tanto, la primera instrucción xsl:

<xsl: for-each select = “customerlist/customer”/>

inicia la búsqueda de un elemento etiquetado customerlist (ListaCliente). Cuando lo encuentra comienza una segunda búsqueda de un elemento etiquetado del cliente (debe estar en customerlist). Si se encuentra algo semejante se toman las acciones que se indican en el ciclo que termina con </xls:for-each> (la tercera desde la parte inferior en la hoja de estilo).

Dentro del ciclo, se establecen los estilos para cada elemento en cliente dentro de ListaCliente. Observe la parte interna de cada ciclo que maneja la posibilidad de más de un elemento.

Los procesadores XSLT son orientados en contexto; cada instrucción se evalúa en el contexto en el que se ha realizado un acoplamiento. Así, la instrucción:

<xsl:value-of select= “name/lastname”>

opera en el contexto de ListaCliente/cliente semejante al que se ha hecho. No hay necesidad de codificar:

<xsl:select= "customerlist/customer/name/lastname">

debido a que el contexto ya se ha establecido para customerlist/customer. De hecho, si el select estaba codificado en esta segunda forma no se encontraría nada. De igual manera <xsl:select "lastname"/> no daría un resultado de acoplamiento porque apellido sólo existe en el contexto customerlist/customer/name, y no en el contexto customerlist/customer.

Esta orientación de contexto explica la necesidad de la instrucción:

<xsl:value-of select= "node0"/>

➤ FIGURA 14-11

Materialización con XSLT: a) resultado del procesamiento XSLT de la figura 14-9 con la figura 14-10

```
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:14pt;
      background-color:teal">
<DIV STYLE="background-color:brown; color:white; padding:4px">
<SPAN STYLE="font-weight:bold; color:white">Correlli</SPAN>
      - Michelle
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt; font-style:bold;
color:yellow">
1824 East 7th Avenue
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt; font-style:bold;
color:yellow">
Suite 700
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:12pt; font-style:bold">
Memphis, TN
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:14pt; color:blue">
32123-7788
</DIV>
<DIV STYLE="background-color:brown; color:white; padding:4px">
<SPAN STYLE="font-weight:bold; color:white">Jaynes</SPAN>
      - Lynda
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt; font-style:bold;
color:yellow">
2 Elm Street
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:12pt; font-style:bold">
New York City, NY
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:14pt; color:blue">
02123-7445
</DIV>
</BODY>
</HTML>
```

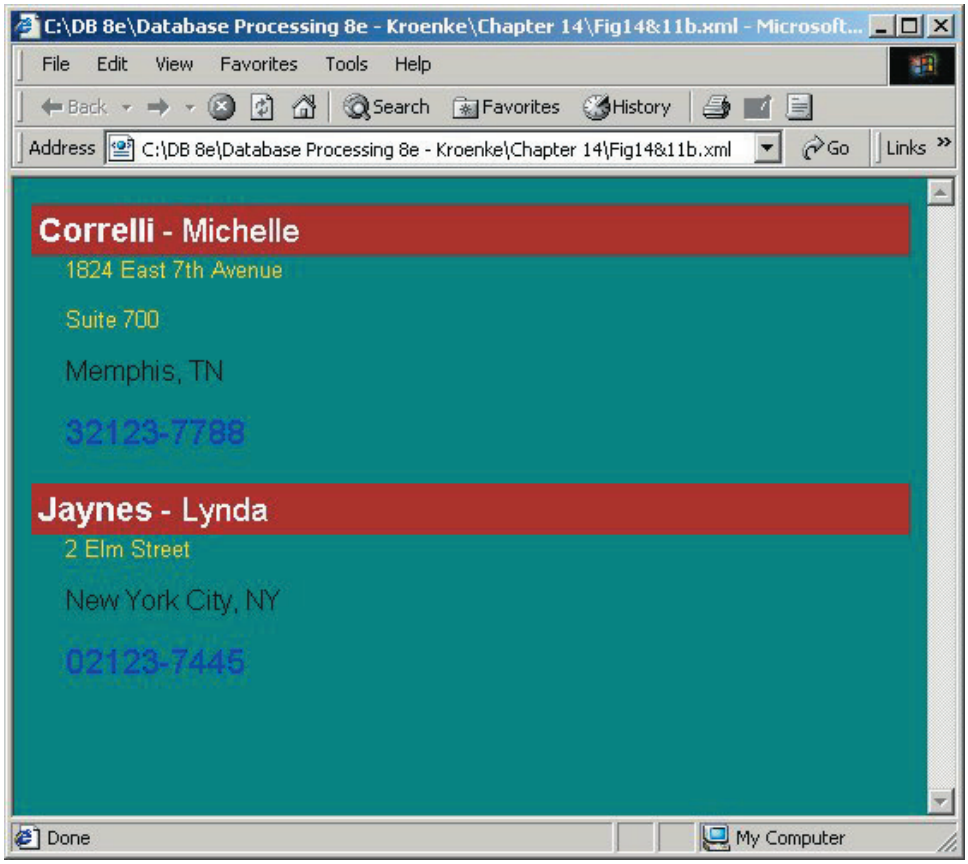
(a)

(*continúa*)

➤ FIGURA 14-11

(Continuación)

b) materialización del explorador



(b)

(en el centro de la hoja de estilo). El contexto de la localización de esta instrucción se ha puesto en customerlist/customer/address/street. Por lo tanto, el nodo actual es un elemento street y esta expresión indica que el valor de ese nodo se va a producir.

Observe también que se ha hecho una pequeña transformación mediante la hoja de estilo. El documento original tiene el nombre seguido del apellido, pero la salida tiene el apellido seguido por el nombre. El HTML generado mediante el analizador XSLT en Internet Explorer se muestra en la figura 14-11(a). Se puede ver en el documento XSL cómo se han seguido las reglas de producción en el documento XML de la figura 14-9. La materialización de HTML, como aparece en la ventana del explorador, se muestra en la figura 14-11(b).

Microsoft ha hecho actualizaciones en el analizador XSLT en Internet Explorer a medida que los estándares evolucionan. Los analizadores usados para generar la figura 14-11 eran recientes (de octubre de 2000). Visite el sitio de la red de Microsoft para obtener más información acerca de XSLT en cuanto a implementaciones más recientes de Internet Explorer.

Terminología y estándares XML

A medida que trabaje con tecnología XML encontrará gran abundancia de acrónimos y términos. En la figura 14-12 se listan la mayoría. Puede encontrar estándares, documentación y algunos tutoriales en los sitios de la red www.w3.org y www.xml.org.

Como se muestra en la figura 14-12, XSL ha tenido dos significados diferentes. En su origen era un método para transformar documentos XML en formatos complicados, como los que se requieren para las impresoras comerciales. En 1999 el Comité W3 decidió dividir el XSL en dos partes: la especificación XSLT para la transformación de documentos y el formato simple, y la especificación XSL-FO (objetos de formateo) para

➤ FIGURA 14-12

Resumen de la terminología XML

Estándar	Descripción
XML	Lenguaje de marcaje extensible. Un documento con lenguaje de marcaje que inicia lo siguiente:
XSL–primer significado	Lenguaje de estilo extensible. En su origen, una forma de transformar XML en estilos muy complicados. Se dividen en dos partes: XSLT para transformar la estructura de documentos y producir formatos simples, y XSL-FO para formatear documentos en estilos complicados y sofisticados.
XSL–significado actual	Hojas de estilo XSLT. Los documentos que proporcionan los pares (acoplamientos, acción) y otros datos para usar XSLT cuando se transforma un documento XML.
XSLT	Un programa (o proceso) que aplica hojas de estilo XSLT a un documento XML para producir un documento XML transformado.
XSL-FO	Lenguaje de estilo extensible. Objetos de formateo. La parte del lado izquierdo del XSL original después de que se eliminó el XSLT. Un estándar muy largo y complicado para formateo sofisticado que aún está en desarrollo.
X-Path	Sublenguaje dentro de XSLT que se ha usado para identificar partes del documento XML que será transformado. Se puede usar también para cálculos y manejo de cadenas. Parecido a XSLT.
XPointer	Un estándar para enlazar un documento a otro. Xpath tiene muchos elementos desde XPointer.
SAX	API simple(Interfaz de programas de aplicación) para XML. Un analizador basado en eventos que notifica al programa cuando se han encontrado elementos de un documento XML durante el análisis del documento.
DOM	Modelo de Objeto Documento. API que representa un documento XML como un árbol. Cada nodo del árbol representa una pieza del documento XML. Un programa puede acceder directamente y manejar un nodo de la representación DOM.
XQL	Un estándar para expresar consultas de la base de datos como documentos XML. La estructura de la consulta usa dispositivos XPath y el resultado de la consulta se representa en un formato XML. Está en desarrollo y puede ser importante en el futuro.
Namespace XML	Un estándar para ubicar la terminología para definir colecciones. X:Nombre se interpreta como el elemento Nombre definido en nombreespacio X. (namespace X) Y:Nombre se interpreta como el elemento nombre definido en nombreespacio Y. Es útil para generar términos no ambiguos.
Esquema XML	Un lenguaje XML dócil para restringir la estructura de un documento XML. Extiende y reemplaza los DTD. Está en desarrollo y es muy importante para el procesamiento de bases de datos.

formatos sofisticados. Actualmente el término XSL por sí mismo se refiere a documentos de hoja de estilo como el que se muestra en la figura 14-10.

XPath es un estándar para enrutar elementos dentro de documentos. En la figura 14-10 se usan expresiones como <xsl:value-of-select="name/lastname"> del estándar XPath para ubicar un elemento particular en el documento. XPath incluye conceptos de otro estándar denominado XPointer, que fue desarrollado para proporcionar un medio sofisticado para documentos que hacen referencia a elementos en otros documentos.

SAX y DOM se refieren a diferentes métodos de análisis de documentos XML. El proceso de análisis consiste en la lectura de un documento, separándolo por componentes y respondiendo a esos documentos de alguna manera, o quizás almacenándo-

los dentro de una base de datos. Los analizadores XML también validan documentos en función de los DTD o esquemas (vea la siguiente sección).

Para usar API SAX, un procesador XSLT (u otro programa que esté funcionando en un documento XML), invoque el analizador flexible SAX y tranfiérale el nombre del documento a analizar. El analizador procesa el documento e invoca objetos dentro del procesador XSLT cada vez que encuentra estructuras en particular. Por ejemplo, un analizador SAX podría llamar al analizador XSLT cuando encuentre un nuevo elemento, pasando el nombre de éste, su contenido y otros aspectos relevantes.

La API DOM funciona a partir de un paradigma diferente. Un analizador DOM flexible procesa el documento XML completo y le crea una representación de árbol. Cada elemento del documento es un nodo en el árbol. El procesador XSLT puede llamar entonces al analizador DOM para obtener elementos particulares usando XPath o un esquema de direccionamiento similar. DOM requiere que el documento completo sea procesado de una sola vez y puede requerir una cantidad de almacenamiento exagerada para documentos muy grandes. Si es así, SAX será la mejor opción. Por otro lado, si todo el contenido del documento necesita estar disponible para usarse de una sola vez, entonces DOM es la única opción.

XQL es un estándar emergente para expresar consultas en términos de un documento XML. Por ejemplo, ésta es una forma de representar una instrucción SQL en XML. Este estándar puede llegar a ser importante en el futuro.

Los dos últimos estándares de la figura 14-12 son muy importantes para los profesionales de las bases de datos porque, entre otros beneficios, facilitan la representación de dominios. Los estándares namespace XML especifican medios para declarar y usar grupos de definiciones. Suponga por ejemplo que un programador define Instrumento en un contexto de música. Dicho instrumento tiene atributos, como Manufactura, Modelo y Material. Un segundo programador también define instrumento, pero en un contexto de electrónica. Este tipo de instrumento tiene como atributos Manufactura, Modelo y Voltaje.

Podemos eliminar ambigüedades definiendo diferentes namespace para cada uno de estos usos. Definimos un namespace Música con las definiciones apropiadas de Manufactura, Modelo y Material para aplicaciones musicales. Definimos un namespace electrónico con Manufactura, Modelo y Voltaje.

Sin el manejo de namespace, la ambigüedad es probable. Un elemento llamado Modelo en un documento XML puede tener un valor de *violín,* o un valor de *regulador de voltaje*. Al especificar el modelo Música en un esquema DTD (siguiente sección), no habrá ambigüedades.

Usamos un namespace en la figura 14-10. La expresión <HTML xmins:xsl= “http...” define un namespace llamado xsl. Más adelante en el documento, expresiones como xsl:value-of select, estuvieron usando ese namespace e indicando que la expresión *value-of* está definido en el namespace xsl.

Para entender completamente la importancia del namespace, necesitamos direccionar el esquema estándar XML. Lo haremos en la siguiente sección.

Esquema XML

EL esquema XML[1] es un estándar para restringir documentos XML. Desempeña una función similar a la de los DTDs, pero mejora y expande la especificación DTD. El esquema XML es nuevo y cambiante, y los proveedores están desarrollando un analizador que pueda validar documentos XML en función del esquema XML. En el momento en que fue escrito este libro, Microsoft tenía una versión beta de un analizador de esquema XML para IE 5.0 en su sitio Web. Esta versión, o una similar, puede ser un

[1] El uso de la palabra esquema no es muy apropiado para nosotros; un esquema XML no tiene relación directa con un *esquema de la base de datos*. Piense en el término esquema en el sentido de un esquema XML. Ahora bien, es posible expresar el esquema de una base de datos en un documento XML. Dicho documento podría estar basado en un esquema XML. Si parece muy complicado por el momento, lo dejaremos así y entenderemos que los esquemas XML y los esquemas de las bases de datos son distintas cosas que se usan para diferentes propósitos.

producto comercial para cuando usted lea esto. También hay muchos otros vendedores, incluyendo Oracle, que han desarrollado un esquema XML validando analizadores. Busque en la red información reciente. La sintaxis que se usa en esta sección coincide con las Especificaciones del esquema W3 XML, de octubre de 2000.

Como se mencionó anteriormente, un documento que se ajusta a un DTD se denomina de tipo válido. De igual manera, un documento que se ajusta a un Esquema XML se llama **esquema válido.** Un documento XML puede estar perfectamente bien ajustado y no ser de tipo válido o de esquema válido.

La mejora más importante del esquema XML con respecto a los DTDs es que los esquemas XML son por sí mismos documentos XML. A diferencia de los DTDs, que tienen una sintaxis propia, se puede usar la misma sintaxis tanto para la definición del esquema XML como para los documentos XML. Esto significa que se puede emplear un validador de esquema XML para comprobar la validez de los esquemas XML que escriba. (Obviamente, hay el mismo problema del huevo y la gallina. El antepasado de todos los esquemas XML se localiza en www.w3.org y se tiene que escribir en formato DTD, pero ningún otro esquema necesita serlo.)

CONCEPTOS DEL ESQUEMA XML. La figura 14-13 ilustra importantes expresiones del esquema XML y su uso en un documento XML de ejemplo. En el esquema XML hay dos tipos de elementos: simple y complejo. (Estos términos se usan exactamente como los empleamos en el modelo de objetos semánticos en el capítulo 4.) Un elemento sencillo consta de un valor de contenido independiente. Los elementos price y purchaseDate de la figura 14-13 son sencillos.

Los elementos tienen un tipo de atributo que indica el dominio del contenido de elementos. Aquí, los precios surgen del dominio dt:decimal. Esta expresión significa que los valores de precio están restringidos por la definición *decimal* en el dt namespace. De manera similar, el tipo (dominio) de purchaseDate está establecido mediante la definición de *date* en el namespace dt. Diremos más acerca de namespace en breve. Por ahora, piense en éste como una referencia a una ubicación donde se documentan las definiciones de los tipos.

El tercer renglón de la figura 14-13 muestra un elemento de tipo complejo. Su nombre es Address y contiene una secuencia de los elementos {street, mailStop, city, state y

➤ FIGURA 14-13

Elementos del esquema XML de ejemplo

Definición del esquema	Parte del documento de ejemplo
<element name="price" type="dt:decimal"/>	<price>27.50</price>
<element name="purchaseDate" type="dt:date"/>	<purchaseDate>2001-12-10</purchaseDate>
<complex Type name="Address"/> <sequence> <element name="street" type="dt:string" maxoccurs="3"/> <element name="mailStop" type="dt:string" minoccurs="0"/> <element name="city" type="dt:string"/> <element name="state" type="dt:string"/> <element name="zip" type="dt:string"/> </sequence> <attribute name="dateLastChanged" type="dt:date"/> </complexType>	<Address dateLastChanged="2001-12-14"> <street>1824 Highland Ave</street> <street>Suite 400</street> <city>Seattle</city> <state>WA</state> <zip>98119</zip> </Address>
<schema targetNamespace xmlns="http://www.kumquat.com/xmldocs/dtd1" xmlns="http://www.kumquat.com/xmldocs/dtd1" xmlns:dt="http://www.w3.org/2000/10/XMLSchema"> <element name="customerView"> <element name="part" type="partType"/> <element name="part" type="partType"/> <element name="color" type="dt:string"/> </element> .... </schema>	<?xml version="1.0"?> <customerView xmlns="http://www.kumquat.com/xmldocs/dtd1"> … <part>12345</part> <color>red</color> … </customerView>

zip}. En el esquema XML la cardinalidad predeterminada de los elementos es 1.1, lo que significa que los valores son requeridos y que no se permite más de un valor. Estas predeterminaciones pueden anular los atributos de ocurrencias mínimas y ocurrencias máximas. Por ejemplo, en Address, street puede aparecer hasta tres veces (ocurrencias máximas = "3") y mailStop es opcional (ocurrencias mínimas = "0"). Un número de ocurrencias ilimitadas se define al poner ocurrenciasmáximas igual a "ilimitado".

A los de tipo complejo se les permite tener atributos, los cuales pueden ser valores asignados en el documento XML. La diferencia entre los atributos y el contenido de los elementos es que los atributos difieren de elementos contenidos en ellos e intentan llevar los metadatos con respecto a los elementos, más que un valor de contenido. Ejemplos típicos de atributos son atributos de unidad para un elemento cuantitativo, o un atributo de entidad federativa (estado) para un elemento de dirección. En la figura 14-13 Address tiene un atributo llamado dateLastChanged. Este atributo reside como "2001-12-10" en el fragmento del documento de ejemplo.

XML NAMESPACES. Los nombres de espacios XML son uno de los aspectos más confusos de la especificación de esquema XML. En parte porque fueron intencionalmente definidos para ser tan generales como fuera posible.

Consideremos las instrucciones en el elemento esquema en el último renglón de la figura 14-13. La primera expresión define el **targetNamespace** (nombreespacio objetivo). Esto identifica los namespaces que el esquema está creando. El objetivo del namespace es un conjunto de definiciones que resultan de un esquema actual; este namespace se usará para validar otros documentos XML. La expresión targetNamespace xmlns= "http://www.kumquat.com/xmldocs/dtd1" dice al analizador que este esquema se usa para generar un namespace con el nombre "http://www.kumquat.com/xmldocs/dtd1. Este nombre lo pueden invocar más tarde los documentos que están basados en este esquema. Se hizo para el documento clienteVista en la segunda columna del último renglón de la figura 14-13.

Además del namespace objetivo cada documento XML puede tener cero o un **namespace predeterminado** y cero para muchas **etiquetas namespace.** Estos namespaces son usados por el analizador mientras está interpretando el documento esquema en el proceso de creación de los namespaces objetivo. El namespace predeterminado está indicado con la palabra clave *xmlns*= seguido por el nombre del namespace entrecomillado. Un namespace etiquetado se indica mediante la palabra clave *xmlns* seguida por un punto y coma, por una etiqueta del namespace, y por un signo de igual, y después el nombre del namespace entre comillas. Así, xmlns:aviation="http://wwww.abc.com/dtd" crean un namespace etiquetado con el nombre de *aviation* y le da el nombre"http://wwww.abc.com/dtd".

En la figura 14-13, la instrucción xmlns="http://www.kumquat.com/xmldocs/dtd1" define el namespace predeterminado. A menos que se indique de otra manera, el analizador usará el namespace predeterminado cuando interprete el contenido del esquema. Observe que en este caso tanto el namespace objetivo como el namespace predeterminado son iguales a "http://www.kumquat.com/xmldocs/dtd1". Esta especificación le dice al analizador que vea dentro del esquema de documento actual las definiciones de cualquier elemento no etiquetado. Por otra parte, estos dos no siempre necesitan ser iguales; el namespace objetivo y el namespace predeterminado pueden ser diferentes.

En el último renglón de la figura 14-13 el namespace predeterminado se usa para definir la *parte* del elemento. Este tipo es partType ("parteTipo") sin etiqueta. Puesto que no hay etiqueta para partType, el analizador buscará la definición de partType en cualquier lugar en el esquema del documento actual.

La tercera instrucción que empieza con *xmlns:dt*, establece el nombre del namespace *dt* a "http://www.w3.org/2000/10/XMLSchema". Este nombre identifica un namespace que tiene tipos de elementos XML estándar. (Véase una lista de ellos en www.w3.org) Este namespace se usa más adelante en este esquema donde el color está determinado por el tipo *dt:cadena*. Esta expresión le dice al analizador que vaya al namespace indicado por la etiqueta dt para encontrar la definición de *cadena* (*string*). Se verá un mejor ejemplo del uso de namespaces múltiples en las siguientes secciones.

Ahora bien, aquí hay un truco. Los namespaces que hemos estado usando aparecen como dirección URL (Uniform Resource Locator). Sin embargo, éste no es un requisito. El único requisito es que los nombres de los namespaces sean únicos en cualquier parte en donde se use siempre el esquema XML. Si sabemos que no se puede usar ningún identificador BOTE, podemos definir un namespace, por ejemplo *navegar,* con la expresión xmlns:navegar="BOTE". Sin embargo, esta palabra no parece ser única.

Los URLs se usan como identificadores de namespace porque se pueden escoger de tal forma que sean únicos en el mundo. Por lo tanto, el identificador "http://www.prenhall.com//databaseprocessing8e/chapter14/figure14-13/dtd" es un identificador único y válido en todo el mundo. Alguien podría usarlo como un identificador namespace, pero Prentice Hall tendría los derechos y podría objetar; es como si alguien hiciera un uso no autorizado de un derecho de autor o marca registrada de Prentice Hall.

Ahora, para continuar con la parte truculenta, de acuerdo con la especificación W3, no hay requerimiento de que un documento de esquema en realidad se localice en el URL indicado. Por lo tanto, el URL sólo se usa para dar una identidad única al namespace y no por otra razón. La ubicación de los documentos actuales que se requieren para definir los namespace se le puede indicar al analizador en alguna otra forma —por ejemplo, quizás como parámetros que se dan al analizador cuando inicia—. La especificación no indica cómo se puede hacer esto.

Sin embargo, recuerde que las implementaciones específicas del Esquema XML puede usar los valores de los nombres namespace para localizar los documentos necesarios. Si es así, es la elección del que lo implementa; el estándar también permitiría otros mecanismos. A manera de recapitulación, diremos que los nombres de namespaces se parecen a los URL, pero no necesitan serlo. Sólo se requiere que sean únicos.

USO DE NAMESPACES MÚLTIPLES PARA RESOLVER AMBIGÜEDAD DE DOMINIO. Anteriormente, cuando analizamos la terminología XML y los estándares, presentamos el problema que ocurre cuando se usa una palabra sencilla para definir dos cosas diferentes. Por ejemplo, suponga que el diseñador de un esquema XML quiere definir instrumentomusical con los elementos {fuente, modelo, comentario} y definir dispositivoeléctrico con los elementos {fuente, modelo y comentario}. Los elementos instrumentomusical y dispositivoeléctrico tienen los mismos nombres, pero sus dominios son diferentes. La fuente de instrumentomusical son sólo compañías que hacen instrumentos musicales, mientras que la fuente de dispositivoeléctrico son únicamente compañías electrónicas. Comentario tiene una diferencia aún mayor. El comentario de instrumento musical se refiere al material del instrumento, —aleación, plata, latón y otros, mientras el comentario de dispositivoeléctrico es voltaje.

La figura 14-14 ilustra el uso de múltiples namespaces para resolver la ambigüedad de estos nombres. Se definen dos namespaces etiquetados: music y electronic (música y electrónica). El namespace música define estos elementos de una manera que abarca al mundo de la música; el namespace electrónica define estos elementos en una forma que abarca al mundo de la electrónica. El tipo complejo musicalInstrument incluye los elementos *source, model* y *comment* (*fuente*, *modelo* y *comentario)* como usted esperaría. Sin embargo, en estos elementos están dados los tipos de namespace musical, la fuente es music:manufacturer, modelo es music:model y comentario es music:material.

Una estrategia similar se usa para instrumentos eléctricos. Estos elementos se definen como tipos electronic:manufacturer, electronic:model y electronic:voltage.

El esquema XML estándar proporciona varias formas de limitar el dominio de definiciones de tipos. Los elementos se pueden restringir a ciertos valores, a ciertos rangos de números, ser constantes, tener valores predeterminados, y así sucesivamente. Por lo tanto, el elemento manufacturer en el namespace música se podría restringir por su tipo {Yamaha, Steinway, Horner} y después sólo se permitirán estos valores. Se pueden usar técnicas similares para todos los otros elementos. Estas técnicas están más allá del enfoque de este análisis; vea www.w3.org para mayor información.

➤ FIGURA 14-14

Ejemplo del uso del esquema XML con namespaces múltiples

Definición del esquema

```
<schema targetNamespace xmlns="http://www.mycompany.com/orders/dtd"
        xmlns="http://www.mycompany.com/orders/dtd"
        xmlns:dt="http://www.w3.org/2000/10/XMLSchema"
        xmlns:music="http://www.musiccompany.com/xml/dtd"
        xmlns:electronic="http://www.electricalcompany.com/xml/dtd">

 <complexType name="equipment">
  <complexType name="musicalInstrument" maxoccurs="unbounded">
   <sequence>
    <element name="source" type ="music:manufacturer"/>
    <element name="model" type ="music:model"/>
    <element name="comment" type ="music:material" minoccurs="0"/>
   </sequence>
  </complexType>

 <complexType name="electronicDevice"/>
   <sequence>
    <element name="source" type ="electronic:manufacturer"/>
    <element name="model" type ="electronic:model"/>
    <element name="comment" type ="electronic:voltage"/>
   </sequence>
  </complexType>
 </complexType>

</schema>
```

Ejemplo de documento

```
<?xml version="1.0"?>
<equipment xmlns="http://www.mycompany.com/orders/dtd>

  <musicalInstrument>
    <source>Yamaha</source>
    <model>Standard Flute</model>
    <comment>Silver</comment>
  </musicalInstrument>

  <musicalInstrument>
    <source>Yamaha</source>
    <model>Piano</model>
   </musicalInstrument>

  <electricalDevice>
    <source>Hewlett-Packard</source>
    <model>HP-2780Z</model>
    <comment>12</comment>
   </electricalDevice>

</equipment>
```

La figura 14-14 ilustra algunas otras ideas. A partir del análisis anterior usted debe ser capaz de explicar por qué este documento de ejemplo de esquema válido puede tener un musicalInstrument con elementos no materiales, y por qué se permiten dos musicalInstruments. También, ¿por qué no se permitiría un segundo dispositivo eléctrico para que éste sea un documento de esquema válido?

COMENTARIO EDITORIAL. Como hemos descrito, es muy importante el procesamiento de las bases de datos. Estas capacidades del Esquema XML proporcionan una solución mundial estandarizada, expandible, a un problema que ha atacado al procesamiento de bases de datos desde su inicio: el cumplimiento de dominios. La única solución hasta ahora era escribir código de aplicación, procedimientos almacenados, o disparadores para imponer las definiciones de dominio. Sin embargo, este código tiene un único propósito y se tiene que reescribir cada vez que un dominio cambia —por ejemplo, una nueva compañía que empieza a producir instrumentos musicales. Con el esquema XML el código se escribe una sola vez— en el programa que valida documentos XML en función de sus esquemas y después se definen los dominios en los datos, usando medios expandibles y estandarizados. Si un dominio cambia, la lista de dominios permitidos se actualiza en una lista enumerada. El programador humano (y, por lo tanto, propenso a cometer errores) que se requiere no tiene que ser altamente capacitado y caro.

Pasarán algunos años para que se desarrollen productos que cumplan este estándar, y a las empresas les llevará algunos años aprender a usarlos. (¡Aquí es donde usted interviene!) Pero, con el tiempo el esquema XML, o algo muy similar, será de muy amplio uso porque simplificará sensiblemente el desarrollo de la aplicación y su mantenimiento.

UN ESQUEMA XML PARA LA GALERÍA VIEW RIDGE CUSTOMERVIEW (VISTACLIENTE). La figura 14-15 muestra un Esquema XML para la vista del cliente de la Galería View Ridge (tomada de la figura 10-4). El elemento nameOfArtist (NombredelArtista) se define en la parte superior de este esquema; esto se hizo para que se pudiera usar dos veces en custView (VisitaCliente).

Este ejemplo muestra la utilidad de establecer igualdad entre Namespace y el namespace predeterminados. Cuando el analizador encuentra nameOfArtist, lo asigna

➤ FIGURA 14-15

Esquema XML para la vista del cliente de la galería View Ridge

Definición del Esquema

```
<schema targetNamespace xmlns="http://www.viewridge.com/customer"
        xmlns="http:// www.viewridge.com/customer"
        xmlns:dt="http://www.w3.org/2000/10/XMLSchema"

 <element name="nameOfArtist" type="dt.string"/>
 <complexType name="custView" maxoccurs="unbounded">
  <sequence>
   <complexType name="customer">
    <sequence>
     <element name="name" type ="dt:string"/>
     <element name="areaCode" type ="dt:string" minoccurs="0"/>
     <element name="localNumber" type ="dt:string"/>
     <complexType name="transaction" maxoccurs="unbounded">
      <sequence>
       <element name="purchaseDate" type="dt:date"/>
       <element name="salesPrice" type="dt:decimal"/>
       <complexType name="work">
        <sequence>
         <element name="artistName" type="nameOfArtist"/>
         <element name="workTitle" type="dt:string"/>
         <element name="workCopy" type="dt:string" minoccurs="0">
        </sequence>
       </complexType>
      </sequence>
     </complexType>
     <complexType name="artistInterest" maxoccurs="unbounded"/>
      <element name=artistName type="nameOfArtist"/>
     </complexType>
    </sequence>
   </complexType>
  </sequence>
 </complexType>
```

Documento ejemplo

```
<?xml version="1.0"?>
<custView xmins="http://www.viewridge.com/customer">

 <customer>
  <name>Jackson, Elizabeth</name>
  <areaCode>206</areaCode>
  <localNumber>989-4344</localNumber>
  <transaction>
   <purchaseDate>2002-12-10</purchaseDate>
   <salesPrice>4300.00</salesPrice>
   <work>
    <artistName>Juan Miro</artistName>
    <workTitle>Poster</workTitle>
    <workCopy>14/85</workCopy>
   </work>
  </transaction>
  <artistInterest>
   <artistName>Juan Miro</artistName>
   <artistName>Mark Tobey</artistName>
   <artistName>Dennis Frings</artistName>
  </artistInterest>
 </customer>
</customerview>
```

al objetivo namespace. Más tarde, cuando los dos elementos artistName sean procesados y se encuentre su tipo como nameOfArtist, sin ninguna etiqueta (como dt), el analizador verá al namespace predeterminado para encontrar nameOfArtist. Lo encontrará porque el objetivonamespace y el namespace predeterminados son iguales.

Definir un tipo de elemento como éste evita errores debido a que si éste se cambia, sólo se necesitará cambiarlo en un lugar y el cambio se propagará a los elementos que se definen en ahí. También las definiciones pueden ser larguísimas y definir múltiples atributos una vez que se guarde el trabajo.

Este ejemplo no muestra todo el poder de un Esquema XML. Por ejemplo, el dominio workCopy es nnn/mmm, donde nnn y mmm son enteros positivos y se requiere que sea nnn menor a mmm. Este dominio complicado se puede definir en una sintaxis de un Esquema XML y que el analizador lo imponga. Comentarios similares se aplican a las restricciones de areaCode, localNumber, purchaseDate y salesPrice.

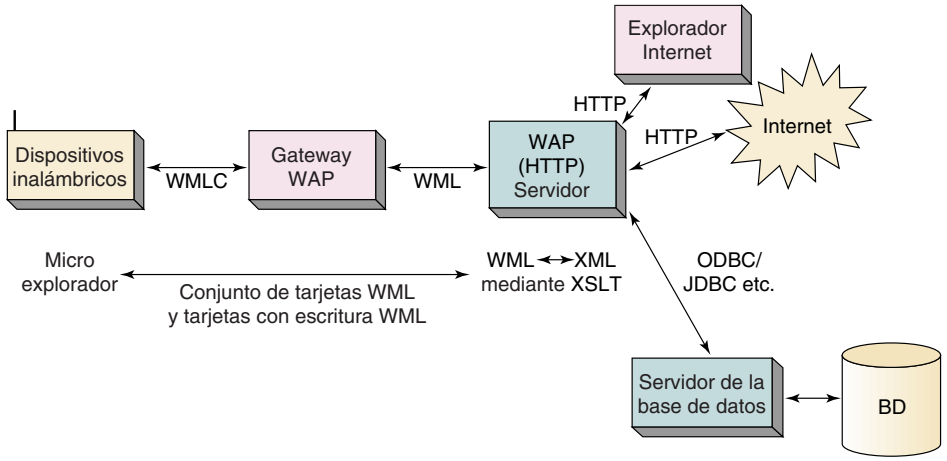
Protocolo de aplicaciones inalámbricas

El protocolo de aplicaciones inalámbricas tiene como propósito facilitar el uso de tecnología de Internet en dispositivos inalámbricos tales como teléfonos celulares y asistentes digitales personales. Al inicio del presente capítulo abordamos brevemente una aplicación WAP típica. En esta sección bosquejaremos la esencia del WAP; vea www.WapForum.org para mayor información.

En la figura 14-16 se muestra la mayoría de los componentes de un sistema WAP. Un servidor WAP es un servidor HTTP que transforma protocolos XML de la red en un **lenguaje de marcaje inalámbrico,** o WML. WML es un subconjunto de XML; así, los documentos WML se pueden validar en función de esquemas DTD o XML. Además,

➤ FIGURA 14-16

Uso del protocolo de aplicaciones inalámbricas (WAP)



XSLT se puede usar para transformar documentos XML en WML para el procesamiento en los dispositivos inalámbricos. XSLT se puede usar también para transformar la respuesta WML de los dispositivos inalámbricos dentro del estándar XML. El estándar WML incluye un lenguaje de script, **WML Script,** que es una variante de JavaScript.

Como se muestra en la figura 14-16, el WML lo transforma una computadora de Gateway a Web en una forma compacta que se denomina WMLC, la cual es similar a un programa compilado; es una versión binaria de un documento WML. El propósito de esta compresión es reducir el ancho de banda que se requiere para transmitir el documento.

Actualmente los dispositivos inalámbricos tienen un espacio insuficiente en pantalla y capacidad de teclado para soportar los exploradores completos. Los **microexploradores** son aplicaciones sumamente ligeras que procesan WML.

Cuando se usan documentos WML y XML se dividen en secciones que son lo bastante pequeñas como para que el microexplorador las despliegue en el dispositivo inalámbrico. Cada sección se llama **tarjeta** y la colección de tarjetas se denomina **conjunto de tarjetas.** Normalmente el conjunto de tarjetas completo se transmite al dispositivo inalámbrico una sola vez. A medida que se necesitan las tarjetas se despliegan en el dispositivo inalámbrico. Las referencias a una tarjeta particular se escriben en el formato *deck-identifier#card-identifier*. Se puede tener acceso a las tarjetas de un conjunto de tarjetas desde el dispositivo inalámbrico, mientras que las de otros conjuntos requieren regresar al servidor WAP.

WAP no requiere el uso de XML; también se puede usar HTML. Sin embargo, la ventaja de usar XML es que los documentos WML se pueden validar y se puede usar XSLT para convertir entre WML y XML.

La importancia de XML para las aplicaciones de bases de datos

Después del modelo relacional, XML es el desarrollo más importante de bases de datos. La figura 14-17 lista las razones del porqué. Primero, el esquema XML ofrece una manera estandarizada de representar dominios. Así mismo, XML proporciona un medio

➤ FIGURA 14-17

Características importantes para el procesamiento de bases de datos

> Medios estándar para representar dominios
> Medios estándar para expresar vistas de bases de datos
> Separación limpia de estructura, contenido y materialización
> Facilidad para verificar la validez del documento
> Estándares industriales para tipos de documentos

estándar para expresar la estructura de las vistas de bases de datos. Debido a la norma, cualquier aplicación que pueda procesar un documento en esquema DTD o XML puede interpretar correctamente cualquier vista de bases de datos.

Considere el esquema XML para la vista de Cliente en el caso de la galería View Ridge, en la figura 14-15. Antes del estándar XML, dos programas que intercambiaran esta vista necesitarían desarrollar un protocolo privado para especificar su estructura. Con XML sólo necesitan interpretar el Esquema XML de la figura 14-15, ahí no se requiere ningún acuerdo previo con respecto a protocolo.

En el pasado los programadores usaban SQL como reemplazo de una forma estándar para describir una vista de datos en una base de datos. Sin embargo, como vimos en el capítulo 10, SQL no se puede usar para cualquier vista que implique más de una trayectoria de valores múltiples a través del esquema. La vista en la figura 14-15 tiene dos patrones de valores múltiples; uno a través de Transaction y otro a través de Artist. Así, se requieren instrucciones SQL múltiples y no existe una manera estándar para expresar cómo deberían estar conectados si se usa SQL. XML supera esta deficiencia.

El tercer beneficio principal de XML es una distinción muy clara entre la estructura, el contenido y la materialización. La estructura de un documento puede estar almacenada en un lugar como un DTD o esquema, y todos los documentos basados en esa estructura pueden incluir referencias a éste. Por ejemplo, la compañía Boeing puede colocar un esquema de documento para los pedidos de partes de avión en uno de sus sitios de la red, y para todas las partes de avión que se han comprado puede usar este esquema cuando se creen los documentos de pedido.

Las compañías que hacen pedidos a Boeing pueden desarrollar sus propias aplicaciones de bases de datos para construir los documentos de pedidos XML que conformen el esquema. Los medios que usan para crear estos documentos están completamente ocultos para Boeing y cualquier otra compañía. Pueden usar una aplicación de bases de datos o usar un procesador de palabras; la forma como se crean los datos es desconocida y no tiene importancia.

Además, Boeing puede desarrollar diferentes hojas de estilo XSL para materializar sus documentos. Puede tener una para el ingreso de pedidos por departamento, una para su departamento de producción, otra para el de contabilidad, y otra más para su departamento de ventas. También pueden contar con una para el uso de sus clientes. Los clientes de Boeing también pueden crear sus propias hojas de estilo para documentos basados en este esquema. Así, cada uno puede materializar el contenido de un documento de pedido estándar en cualquier forma que les resulte conveniente.

La comprobación de la validez de un documento es la cuarta característica del XML. Usando el ejemplo de la Boeing, cuando un cliente crea un documento de pedido, puede verificar la validez de ese documento en función de un DTD o esquema que le asegure que está transmitiendo sólo pedidos de tipo válido, o esquema válido. De manera similar, cuando el sitio de Boeing recibe documentos de pedidos puede comprobarlos automáticamente en función del DTD, o del esquema, para asegurarse de que sólo están aceptando documentos de tipo válido o esquema válido.

Esta comprobación de validez también se puede realizar con XML, pero es mucho más difícil. Boeing necesitaría desarrollar un programa especializado para validar documentos de pedido y después mandar ese programa a los clientes o departamentos que lo necesitaran. Tendrían que crear un programa especializado diferente para un segundo tipo de documentos, otro más para un tercer tipo de documentos, y así sucesivamente. Con el estándar XML sólo necesitan colocar los DTD o esquemas en una ubicación pública y todas las partes interesadas pueden validar sus propios documentos de diferentes tipos.

La última ventaja importante es que con XML los grupos industriales pueden desarrollar DTD para grandes industrias y esquemas. **La Organización para el Avance de Estándares de Información Estructurada** (Organization for the Advancement of Structured Information Standards, **OASIS**) funciona como un centro distribuidor de publicaciones de esquemas estándares XML. Mantiene el registro de XML.ORG que lista esquemas publicados y un buscador para localizar esquemas sobre temas específicos. La figura 14-18 lista algunos de los estándares que existen actualmente.

➤ FIGURA 14-18

Ejemplos de los estándares XML industriales

Tipo de industria	Ejemplos de estándares
Contabilidad	• American Institute of Certified Public Accountants (AICPA): Extensible Financial Reporting Markup Language (XFRML)[OASIS Cover page] • Open Applications Group, Inc (OAG)
Arquitectura y construcción	• Architecture, Engineering, and Construction XML Working Group (aecXML Working Group) • ConSource.com: Construction Manufacturing and Distribution Extensible Markup Language (cmdXML)
Automotriz	• Automotive Industry Action Group (AIAG) • Global Automedia: • MSR: Standards for information exchange in the engineering process (MEDOC) • The Society of Automotive Engineers (SAE): XML for the Automotive Industry–SAE J2008[OASIS Cover page] • Open Applications Group, Inc (OAG)
Banca	• Banking Industry Technology Secretariat (BITS): [OASIS Cover page] • Financial Services Technology Consortium (FSTC): Bank Internet Payment System (BIPS)[OASIS Cover page] • Open Applications Group, Inc (OAG)
Intercambio electrónico de datos	• Data Interchange Standards Association (DISA): [OASIS Cover page] • EEMA EDI/EC Work Group[OASIS Cover page] • European Committee for Standardization/Information Society Standardization System (CEN/ISSS; The European XML/EDI Pilot Project[OASIS Cover page] • XML/EDI Group[OASIS Cover page]
Recursos humanos	• DataMain: Human Resources Markup Language (hrml) • HR-XML Consortium[OASIS Cover page]: JobPosting, CandidateProfile, Resume • Open Applications Group (OAG): Open Applications Group Interface Specification (OASIS)[OASIS Cover page] • Tapestry.Net: JOB markup language (JOB) • Open Applications Group, Inc (OAG)
Seguros	• ACORD: Property and Casualty[OASIS Cover page], Life (XMLife)[OASIS Cover page] • Lexica: iLingo

Ejemplo XML para procesamiento de negocio a negocio

La figura 14-19 muestra la función del XML para compartir documentos de bases de datos en una aplicación de negocio a negocio. La Compañía A de la izquierda (digamos, Boeing) usa una o más aplicaciones de bases de datos para crear documentos XML. Éstas se refieren a documentos de esquema XML para determinar la estructura de la vista que se requiere. Como se muestra, estos documentos residen en sitios locales, en bibliotecas de esquemas públicos y en bibliotecas de esquema que guarda la Compañía B (digamos, uno de los proveedores de Boeing).

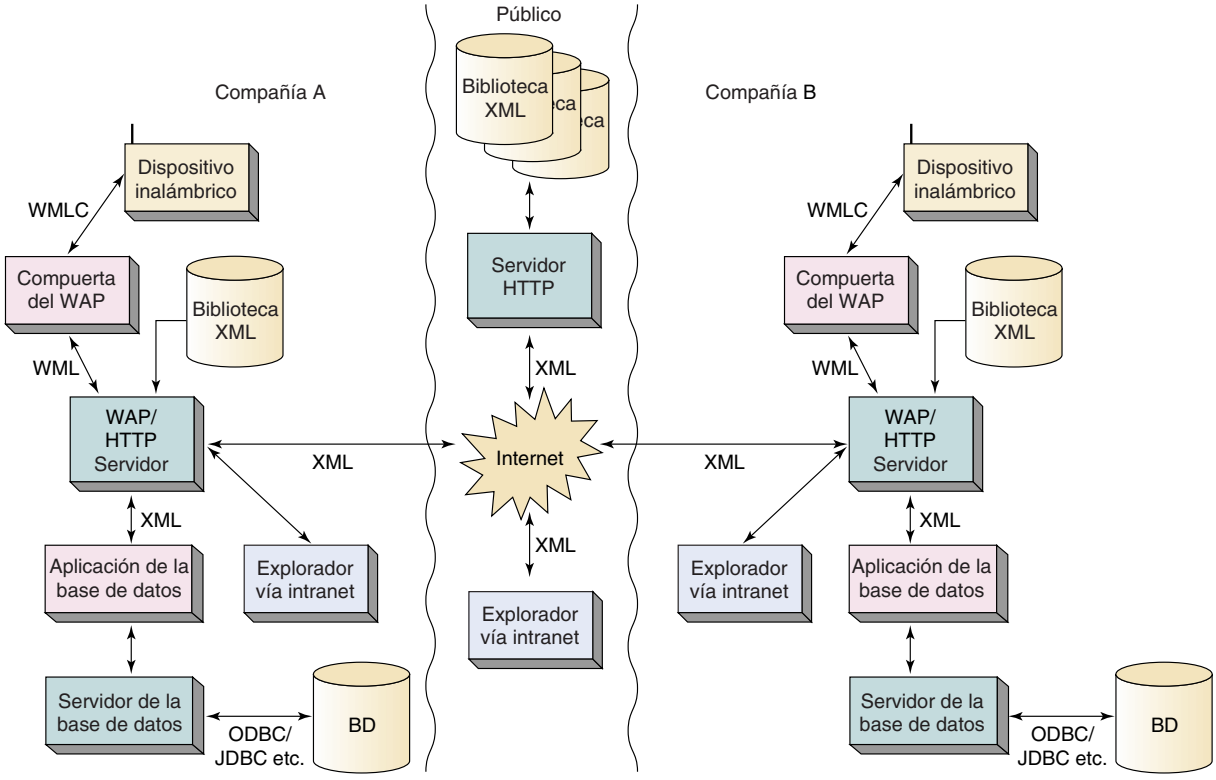
➤ FIGURA 14-18

(Continuación)

Tipo de industria	Ejemplo de estándares
Bienes raíces o inmuebles	• OpenMLS: Real Estate Listing Management System (OpenMLS)[OASIS Cover page] • Real Estate Transaction Standard working group (RETS): Real Estate Transaction Standard (RETS)[OASIS Cover page]
Software	• IBM: [OASIS Cover page] • Flashline.com: Software Component Documentation DTD • Flashline.com: • INRIA: Koala Bean Markup Language (KBML)[OASIS Cover page] • Marimba and Microsoft: Open Software Description Format (OSD)[OASIS Cover page] • Object Management Group (OMG): [OASIS Cover page]
Workflow	• Internet Engineering Task Force (IETF): Simple Workflow Access Protocol (SWAP)[OASIS Cover page] • Workflow Management Coalition (MfMC): Wf-XML[OASIS Cover page]

➤ FIGURA 14-19

XML en todo su esplendor



Una vez que han sido creados los documentos se pasan a un servidor de la red. De ahí son llevados a uno de los dos servidores WAP o HTTP. Por supuesto, no se requiere que los documentos sean accesibles mediante dispositivos inalámbricos; esta capacidad se muestra aquí con fines ilustrativos. Los documentos XML generados que contienen la vista de bases de datos se pueden dirigir a los servidores de la compañía A para uso interno, o publicar en un servidor de la red accesible al público a cargo de la compañía A o de alguna otra, o se pueden transmitir a un servidor que sea controlado por la Compañía B.

En cualquier etapa, el documento XML se puede validar en función del esquema XML para asegurarse de que no se haya perdido nada. Además, en cualquier etapa el documento XML se puede transformar de XSLT a un formato diferente XML, a WML, a voz, o al formato que requieran los dispositivos de manufactura tales como robots o cualquier otro formato. Si se transforma a un formato que sea por sí mismo XML (tal como WML), los esquemas XML se pueden utilizar para validar el documento transformado.

Las aplicaciones de bases de datos se pueden escribir para usar tecnologías XML y publicar aplicaciones en innumerables maneras, usando docenas de formatos diferentes, y todo lo anterior mediante el uso de estándares industriales.

Soporte de XML en servidores Oracle y SQL server

Los servidores Oracle y SQL Server han empezado a incorporar el manejo de XML. Ambos productos incluyen capacidad para almacenar documentos XML en las bases de datos, y proporcionan un analizador XML DOM para procesar documentos XML. Usando estos analizadores, los programadores pueden escribir un código para extraer campos de bases de datos desde documentos XML y almacenar dichos campos como columnas de tablas.

Además, ambos productos tienen soporte para XPath, el cual habilita a los documentos XML para que puedan ser buscados. También proveen medios para usar consultas SQL en función de datos de la base de datos, y tienen los resultados materializados como documentos XML. Ambos productos habilitan a los documentos XML generadores de bases de datos y a las formas para que sean procesadas usando tecnología Internet. Oracle utiliza páginas XSQL que se pueden incorporar a las páginas de servidores JAVA (vea el capítulo 16). Los servidores SQL utilizan páginas ADO y ASP, que estudiaremos en el siguiente capítulo.

El soporte para XML en productos de bases de datos comerciales como estos obviamente está en etapa formativa. La integración entre los constructores de las bases de datos y XML es un poco difícil. Sin duda estas capacidades serán mejoradas en futuras versiones de ambos productos. Este tema atañe a los procesamientos de bases de datos comerciales, así que será interesante ver cómo termina todo esto en el futuro. ¡Permanezca en contacto!

➤ RESUMEN

Una red es un conjunto de computadoras que se comunican usando un protocolo estandarizado. La red pública la puede usar cualquiera que pague la cuota establecida; los usuarios de las redes privadas deben estar preautorizados para conectar las redes. Internet es una red pública de computadoras basada en el protocolo TCP/IP; empezó como un proyecto de investigación conocido como ARPANET.

En 1989 el protocolo de transferencia de hipertexto, o http, fue desarrollado primero por Tim Berners-Lee en CERN. El HTTP está orientado a los requerimientos y no mantiene estados.

Una intranet es una LAN o WAN privada que usa TCP/IP, HTML y exploradores. Las intranets pueden estar o no estar conectadas a Internet, o estar conectadas mediante una firewall. Las intranets tienen un rendimiento de órdenes de magnitud de velocidad mayor a la de Internet. El acceso a redes inalámbricas es posible desde teléfonos celulares y otros dispositivos inalámbricos usando un protocolo de aplicaciones inalámbricas.

Las aplicaciones de bases de datos que emplean la tecnología Internet con frecuencia usan tanto la arquitectura de tres capas como la multicapa. La arquitectura de tres capas consta de un servidor de bases de datos, un servidor de la red y las computadoras de los clientes. Cada una de estas capas puede usar un sistema operativo diferente y los productos de los diferentes proveedores. El propósito de servidores de bases de datos es ejecutar los DBMS para procesos SQL y proporcionar servicios de bases de datos de usuarios múltiples. El servidor de la Web es un servidor HTTP que alberga lenguajes de script y habilita las aplicaciones de las bases de datos para procesar vistas de la base de datos (CRUD). La máquina del cliente opera como un cliente HTTP que alberga un segundo ambiente de escritura y materializa vistas de la base de datos.

Los servidores comunes de la red son IIS de Microsoft, Apache y Netscape Web Server. Las páginas de los servidores activos por lo general se usan con ISS sobre servidores Windows; los servlets Java y JSP comúnmente se usan con servidores Unix.

Los estándares son importantes para lenguajes de marcaje, de tal forma que las páginas de la red las puede procesar cualquier explorador. El W3C publica estándares y administra los procesos de desarrollo. Los productos comerciales extienden los estándares, pero cuando lo hacen crean problemas para los usuarios de productos que no soportan las extensiones. HTML 4.0 fue desarrollado como un estándar que superaba problemas de versiones anteriores de HTML. El HTML dinámico de Microsoft (DHTML) implementa y amplía el HTML a 4.0. Hay tres características de DHTML que son importantes para las aplicaciones de la base de datos: el modelo de objeto documento (DOM), las hojas de estilo en cascada (CSS) y los servicios de datos remotos (RDS), son un conjunto de controles ActiveX que permiten el ocultamiento y la materialización de los datos del cliente sin hacer viajes redondos al servidor.

El lenguaje de marcaje extensible (XML) fue diseñado para proporcionar una clara separación entre la estructura del documento, el contenido y la materialización. Es extensible, pero en una forma estandarizada. La estructura de los documentos está definida por ambos, por las descripciones del tipo de documento (DTDs), o por documentos de esquema XML. Los documentos XML que se adaptan a sus DTDs son documentos de tipo válido. Los documentos XML pueden ser materializados usando CSS o el Lenguaje de Estilo Extensible: transformaciones (XSLT). XSLT es declarativo y de transformación para el manejo de los documentos XML. Se puede usar para transformar documentos XML en HTML, o en otro documento XML que tenga diferente estructura.

Los estándares importantes XML se resumen en la figura 14-12. El esquema XML es un estándar para documentos XML restringidos; los documentos XML que se adaptan a sus esquemas se llaman de esquema válido. EL esquema XML mejora los DTD de diferentes maneras. Por ejemplo, los esquemas de los documentos XML son en sí mismos documentos XML que se pueden validar. Además, los esquemas XML proporcionan un medio extensible generalizado para imponer dominios a través del uso de namespaces XML.

WML es un estándar XML para procesar documentos XML usando microexploradores o dispositivos inalámbricos. Con WML las páginas XML se dividen en secciones llamadas tarjetas. Un conjunto completo de tarjetas WML se llama tablero de tarjetas. El lenguaje escrito que se usa es WML, el cual es una variante de ECMAScript.

XML es muy importante para las aplicaciones de bases de datos porque proporciona un medio estándar para expresar la estructura de vistas de bases de datos; divide claramente la estructura de documentos, el contenido y la materialización; permite comprobar la validez de documentos estandarizados y que los grupos industriales definan los estándares útiles para la estructura de vistas de bases de datos estándar para la industria. Ambos servidores, Oracle y SQL Server, incluyen soporte para XML.

➤ PREGUNTAS DEL GRUPO I

14.1 Defina los términos *red*, *red pública* y *red privada*.

14.2 Defina *Internet*.

14.3 ¿Qué es TCP/IP y cómo se usa?

14.4 Explique qué significa el hecho de que HTTP esté orientado a los requerimientos y a no mantener estados.

14.5 ¿Qué protocolo ha sido desarrollado para manejar el procesamiento de redes inalámbricas?

14.6 Nombre las tres capas de la arquitectura de tres capas y describa la función de cada una.

14.7 Explique cómo la arquitectura de tres capas permite la interoperabilidad del sistema operativo y de los productos Web.

14.8 Explique las funciones de cada uno de los componentes del servidor de la red en la figura 14-4.

14.9 Explique dos usos de las capas extra en una arquitectura de *multicapa*.

14.10 ¿Por qué son importantes los estándares de los lenguajes de marcaje?

14.11 ¿Qué es W3C y por qué es importante?

14.12 ¿Por qué los proveedores tienen una relación amor-odio con las normas o estándares?

14.13 Resuma las desventajas de las versiones antiguas de HTML.

14.14 ¿Cuál es la diferencia entre DHTML y HTML 4.0?

14.15 Explique la importancia del modelo de objeto documento(DOM).

14.16 Defina CSS y explique su importancia.

14.17 Defina RDS y explique su función.

14.18 Defina XML y explique por qué es superior a HTML y a DHTML.

14.19 Explique por qué no hay demasiada libertad con HTML.

14.20 ¿Qué es DTD y por qué es importante?

14.21 Defina documentos XML de *tipo válido* y de *tipo no válido*.

14.22 Explique cómo se usa CSS con documentos XML.

14.23 Defina XLST y explique su importancia.

14.24 ¿Por qué XSLT es declarativo? ¿Por qué es transformable?

14.25 Explique la importancia del contexto cuando se trabaja con XSLT.

14.26 Describa la diferencia entre DOM y SAX.

14.27 ¿Cuáles son los tipos de elementos en el Esquema XML? Dé un ejemplo sobre cada uno.

14.28 Explique la diferencia entre contenido de elementos y atributos de elemento.

14.29 ¿Qué es el namespace objetivo y cuál es su función?

14.30 ¿Cuál es el namespace predeterminado y cómo se usa?

14.31 ¿Cuál es el significado de dar al namespace objetivo y al namespace predeterminado el mismo nombre?

14.32 Proporcione un ejemplo de un namespace etiquetado.

14.33 Explique cómo usar el namespace para resolver la ambigüedad de nombre de elemento.

14.34 ¿Por qué el nombre de namespace parece una dirección de Internet? ¿El esquema XML estándar requiere que los documentos esquemas se localicen en esas direcciones?

14.35 Defina WAP, WML, WML Script, tablero de tarjetas y tarjetas.

14.36 ¿Por qué XML es importante para las aplicaciones de las bases de datos?

14.37 ¿Por qué SQL no es una forma eficaz para definir la estructura de las vistas de bases de datos?

➤ PREGUNTAS DEL GRUPO II

14.38 Visite www.w3.org y determine los estándares actuales que se recomiendan para HTML. ¿En qué difieren de lo que se describió en este capítulo? ¿Existe un nuevo estándar para HTML en formación? Si es así, ¿qué es y qué nuevas características deberá tener?

14.39 Visite www.w3.org y determine los estándares actuales recomendados para XSLT. ¿Cómo difieren de la implementación de XSLT descrita en este capítulo para Internet Explorer 5.0? ¿Qué estándares XSLT están en proceso?

14.40 Visite www.w3.org y determine el estándar actual recomendado para el esquema XML. ¿En qué difiere de lo descrito aquí?

14.41 Visite los dos sitios de la red Microsoft y Oracle e investigue productos con esquema XML, así como la tecnología que manejan. ¿En qué difieren estos productos del esquema XML descrito aquí?

14.42 Visite www.w3.org, vaya al registro y encuentre un estándar de tipo industrial que le interese. Describa el propósito del estándar. ¿Qué tan útil cree que pueda llegar a ser el estándar?

14.43 Visite www.w3.org y defina la recomendación actual del estándar WAP. ¿Cómo difiere de lo descrito aquí?

➤ PREGUNTAS DEL PROYECTO FIREDUP

Los Robard quieren que sus clientes registren sus estufas en línea. No están seguros de cómo se puede hacer y por eso han contratado a un consultor para que los ayude.

Suponga que deciden operar su propio sitio en la red en lugar de usar los servicios de un despacho.

A. Comente la decisión de operar su propio sitio. ¿Cuáles son las ventajas y desventajas?

B. Considere la arquitectura de tres capas a la luz de lo que el registro de FiredUp necesita. ¿Cuál será el propósito de cada capa?

C. ¿Recomendaría Windows 2000 o Linux para el servidor de la red? Explique las ventajas y desventajas de cada uno.

D. ¿Les recomendaría servidores Access, Oracle o SQL Server para sus bases de datos? Explique.

E. ¿Sería conveniente que FiredUp combinara el servidor Web y el servidor de la base de datos en una sola computadora? ¿Cuáles son las ventajas y desventajas?

F. Cree un XML DTD y muestre un documento de registro XML. Suponga que las tablas de la base de datos son:

CLIENTE (ClienteID, Nombre, Teléfono y CorreoElectrónico)
ESTUFA (NúmerodeSerie, Tipo, Versión, FechadeFabricación)
REGISTRO (*ClienteID, NúmerodeSerie*, Rfecha)
REPARACIÓNESTUFA (NúmerodeFacturadeReparación, *NúmerodeSerie*, FechadeReparación, Descripción, Costo, *ClienteID*)

G. Cree un Esquema XML para los documentos de registro.

CAPÍTULO 15

ODBC, OLE DB, ADO y ASP

Este capítulo analiza las interfaces estándar para el acceso a los servidores de bases de datos. ODBC (Open Database Connectivity standard), o el estándar de la conectividad abierta de base de datos, se desarrolló a principios de la década de 1990, con el fin de proporcionar medios independientes del DBMS para el procesamiento de los datos de una base de datos relacional. A mediados de la década de 1990 Microsoft anunció OLE DB, la cual es una interfaz orientada a objetos que encapsula la funcionalidad del servidor de datos. Como aprenderá, OLE DB fue diseñada no sólo para bases de datos relacionales, sino para muchos otros tipos de datos. Como una interfaz COM, OLE DB es fácilmente accesible para C++, C# y programadores Java, pero no para desarrolladores de Visual Basic o Script. Por lo tanto, Microsoft desarrolló Objetos de Datos Activos (ADO, Active Data Objects), que es un conjunto de objetos para el uso de OLE DB, diseñado para utilizar cualquier lenguaje, incluyendo VB, VBScript, y JScript.

Antes de considerar estos estándares, necesitamos tener una perspectiva del ambiente de datos que rodea al servidor de la red en aplicaciones de bases de datos de tecnología Internet.

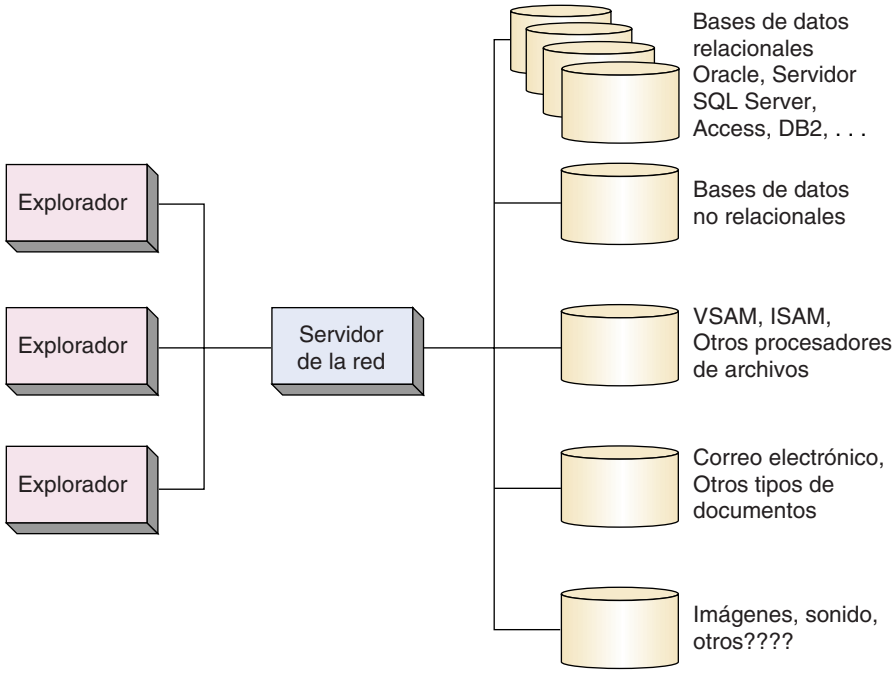
➤ EL AMBIENTE DE DATOS EN UN SERVIDOR DE LA RED

El ambiente en el que residen actualmente las aplicaciones de bases de datos de la tecnología Internet es rico y complejo. Como se muestra en la figura 15-1, un servidor común de la red necesita publicar las aplicaciones que involucren datos de docenas de diferentes tipos de datos. En este texto hemos tratado solamente las bases de datos relacionales, pero como podrá apreciar en esa figura, existen muchos tipos de datos.

Considere los problemas que tiene el analista de las aplicaciones del servidor Web cuando se integran estos datos. Probablemente necesitará conectar una base de datos Oracle, una DB2, una no relacional como IMS, datos del procesamiento de archivos como VSAM e ISAM, directorios de correos electrónicos, etc. Cada uno de estos productos tiene una interfaz de programación diferente que deberá aprender el analista. Ade-

FIGURA 15-1

Necesidades de los datos de aplicación de Internet

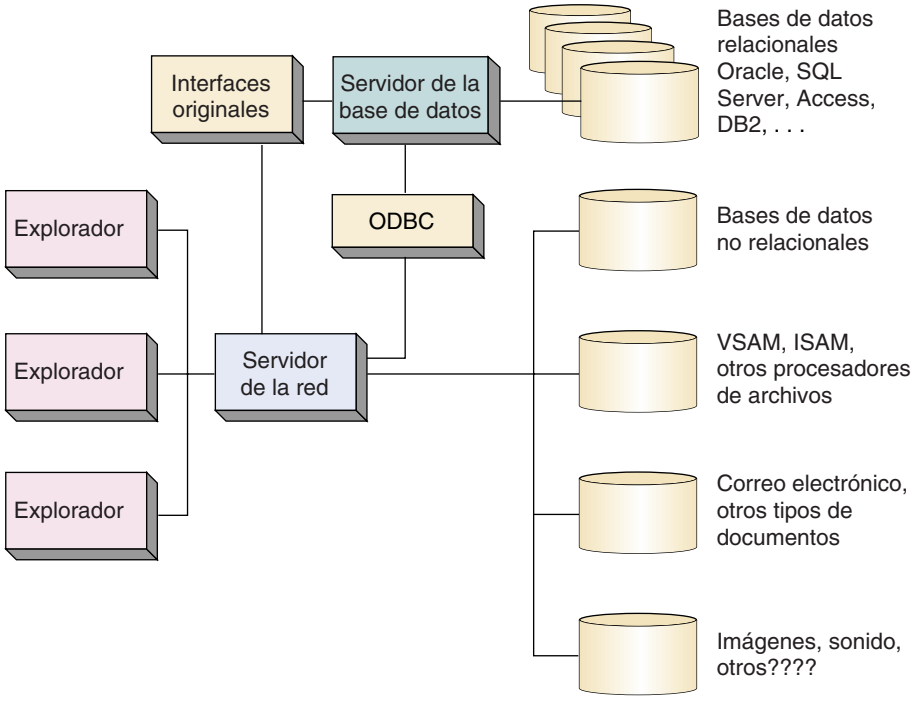


más, cada uno de estos productos evoluciona, así que con el tiempo habrá características y funciones nuevas que aumentarán el reto del analista.

ODBC fue creada para atender la parte de este problema que se refiere a las bases de datos relacionales y fuentes de datos parecidas a tablas, tales como las hojas de cálculo. Como se muestra en la figura 15-2, ODBC es una interfaz entre el servidor de la red (u otro usuario de la base de datos) y el de la base de datos. Consta de un conjunto de estándares mediante los cuales también se pueden emitir las instrucciones de

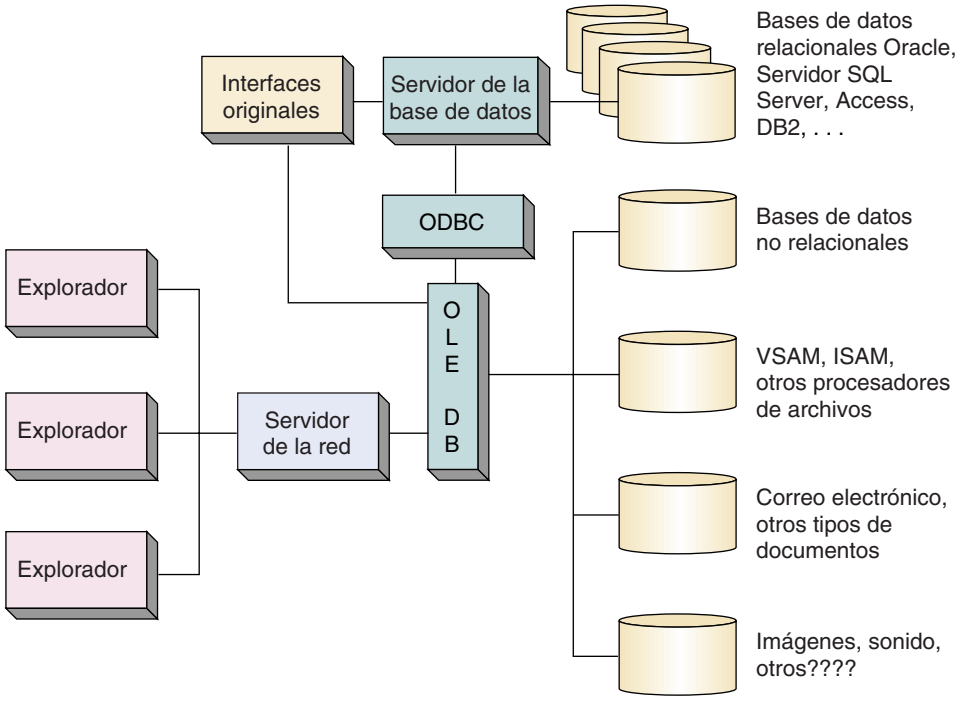
FIGURA 15-2

Función del estándar ODBC



➤ FIGURA 15-3

Función de OLE DB



SQL y transmitir los resultados y los mensajes de error. Como se muestra, los programadores, si lo desean, pueden llamar a los servidores de datos mediante las interfaces DBMS originales (algunas veces lo hacen para aumentar el desempeño), pero el analista agobiado que no tiene tiempo o que no desea aprender diferentes bibliotecas de DBMS, puede utilizar ODBC.

ODBC ha tenido un gran éxito y simplifica algunas tareas del desarrollo de las bases de datos. Como aprenderá, tiene una desventaja sustancial atribuible a Microsoft cuando desarrolló OLE DB. La figura 15-3 muestra la relación de OLE DB, ODBC, y otros tipos de datos. OLE DB proporciona una interfaz orientada a objetos para datos de casi cualquier tipo. Los proveedores de DBMS pueden integrar partes de sus bibliotecas originales en objetos OLE DB para mostrar la funcionalidad de sus productos mediante esta interfaz. OLE DB puede utilizarse como una interfaz para las fuentes de datos ODBC. Por último, OLE DB fue desarrollada para el procesamiento de datos no relacionales.

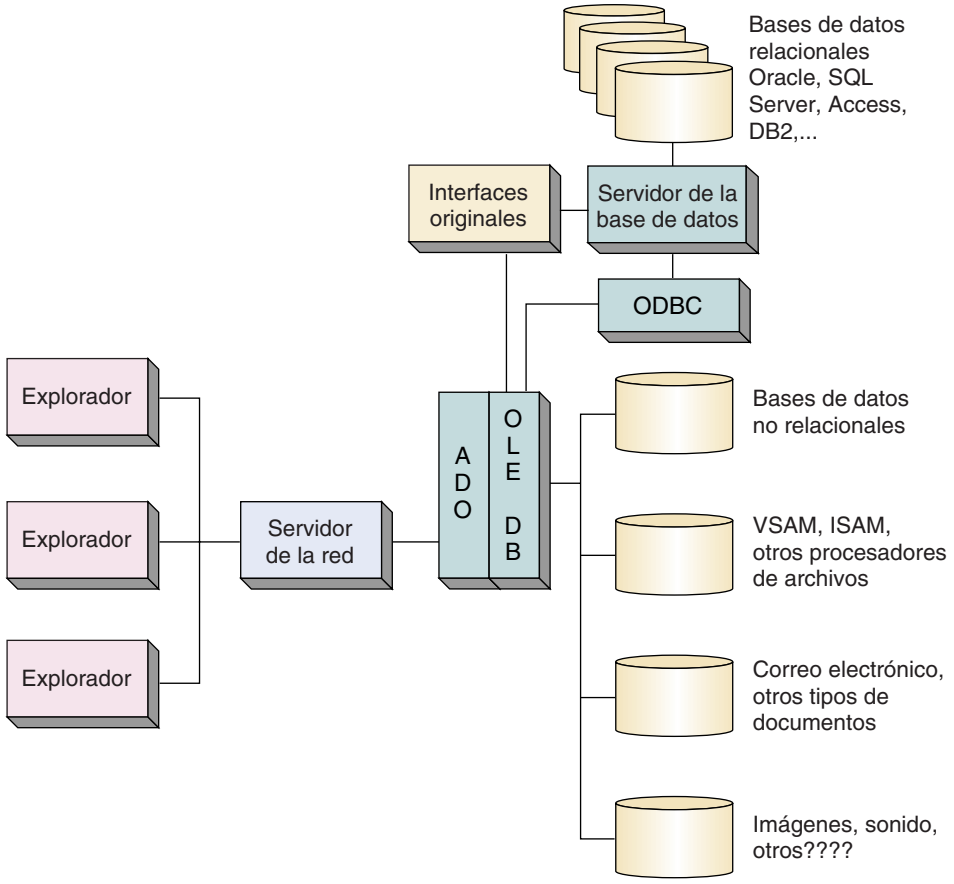
Debido a que OLE DB es una interfaz orientada a objetos, se ajusta particularmente a lenguajes con esa orientación, como C++. Sin embargo, muchos programadores de aplicación de bases de datos programan en Visual Basic, o en lenguajes script tales como VBScript y JScript. Consecuentemente, Microsoft definió a ADO como una protección sobre los objetos OLE DB (véase la figura 15-4). ADO permite que los programadores en casi cualquier lenguaje puedan ingresar a la funcionalidad OLE DB.

Quizás se sienta incómodo con la fuerte presencia de Microsoft en este análisis, pero esta compañía fue la que desarrolló y promocionó OLE DB y ADO, e incluso ODBC recibió una gran importancia, en buena medida debido al apoyo de Microsoft. De hecho otros proveedores y comités estándares propusieron alternativas para OLE DB y ADO, pero debido a que Microsoft Windows reside en aproximadamente 90% de las computadoras del mundo, es difícil para otros promulgar estándares opuestos. Además, en defensa de Microsoft, OLE DB y ADO son excelentes, simplifican el trabajo del analista de la base de datos y probablemente habrían tenido éxito en un campo de juego equitativo.

Sin embargo, nuestros objetivos son más básicos. Usted necesita aprender ADO para construir mejores aplicaciones de las bases de datos en tecnología Internet. Para tal fin, veremos con detalle cada uno de estos estándares.

➤ FIGURA 15-4

Función de ADO



➤ ESTÁNDAR DE LA CONECTIVIDAD ABIERTA DE UNA BASE DE DATOS (ODBC)

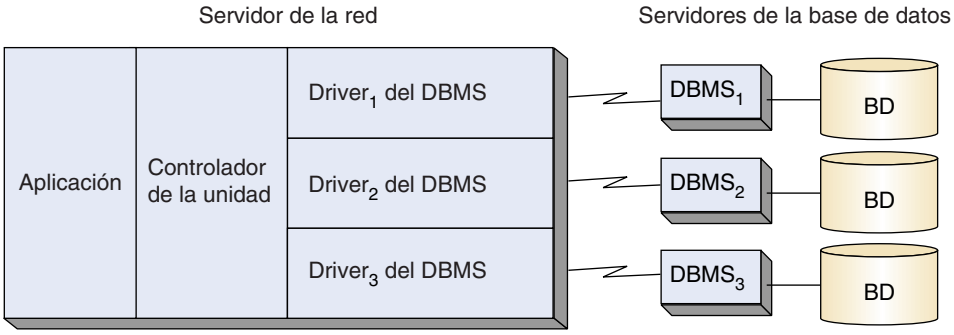
El estándar de la **conectividad abierta de una base de datos** (**ODBC,** por sus siglas en inglés) es una interfaz mediante la cual los programas de aplicación pueden ingresar y procesar bases de datos SQL independientemente del DBMS. Esto significa, por ejemplo, que una aplicación que utiliza la interfaz ODBC podría procesar una base de datos Oracle, una SYBASE, una INFORMIX y cualquier otra base de datos que sea compatible con ODBC sin ningún cambio de codificación en el programa. El propósito es permitir que un analista cree una aplicación sencilla que pueda tener acceso a bases de datos sustentadas por diferentes productos DBMS sin necesidad de cambiarla, o incluso compilarla de nuevo.

ODBC fue desarrollada por un comité de expertos industriales de los comités del Grupo X/Open y de SQL Access. Propusieron muchos de esos estándares, pero ODBC resultó ganadora, principalmente porque fue implementada por Microsoft y es una parte importante de Windows. El interés inicial de Microsoft al apoyar dicho estándar era permitir que productos como Microsoft Excel ingresaran a los datos de una base de datos desde una gran variedad de productos DBMS, sin necesidad de que tuvieran que ser compilados nuevamente. Por supuesto, los intereses de Microsoft han cambiado desde la presentación de OLE DB.

ODBC es importante para los sistemas de las bases de datos de la tecnología Internet debido a que, en teoría, es posible desarrollar una aplicación que pueda procesar bases de datos que funcionen con diferentes productos DBMS. Decimos “en teoría” porque los productos varían en la forma en la cual cumplen con la norma. Afortunadamente, como usted verá, la variación en los niveles de cumplimiento fue prevista por el Comité de la ODBC.

➤ FIGURA 15-5

Arquitectura de la ODBC



La aplicación puede procesar una base de datos mediante cualquiera de los tres productos DBMS.

ARQUITECTURA DE LA ODBC

La figura 15-5 muestra los componentes del estándar ODBC. El programa de aplicación, el controlador de drivers (unidades) y los drivers DBMS residen en la computadora del servidor de la red. Los drivers envían requerimientos a las fuentes de datos, que residen en el servidor de la base de datos. De acuerdo con el estándar, una **fuente de datos** es la base de datos, su DBMS asociado, el sistema operativo y la plataforma de la red. Una fuente de datos ODBC puede ser una base de datos relacional; también puede ser un servidor de archivos, tal como el BTrieve, e incluso puede ser una hoja de cálculo.

La aplicación emite peticiones para crear una conexión con una fuente de datos; para emitir las instrucciones de SQL y recibir resultados; para procesar los errores y empezar, confirmar y revertir las transacciones. ODBC proporciona un medio estándar para cada una de estas peticiones y define un conjunto estándar de códigos y mensajes de error.

El **controlador de drivers** sirve como un intermediario entre la aplicación y los drivers del DBMS. Cuando la aplicación pide una conexión, el driver determina el tipo de DBMS que procesará una fuente de datos ODBC determinada y carga dicho driver en la memoria (si es que aún no está cargado). Asimismo, el controlador de drivers procesa ciertas peticiones de inicialización y valida el formato y el orden de las peticiones al ODBC que recibe de la aplicación. El controlador de drivers lo proporciona Microsoft y está incluido en Windows.

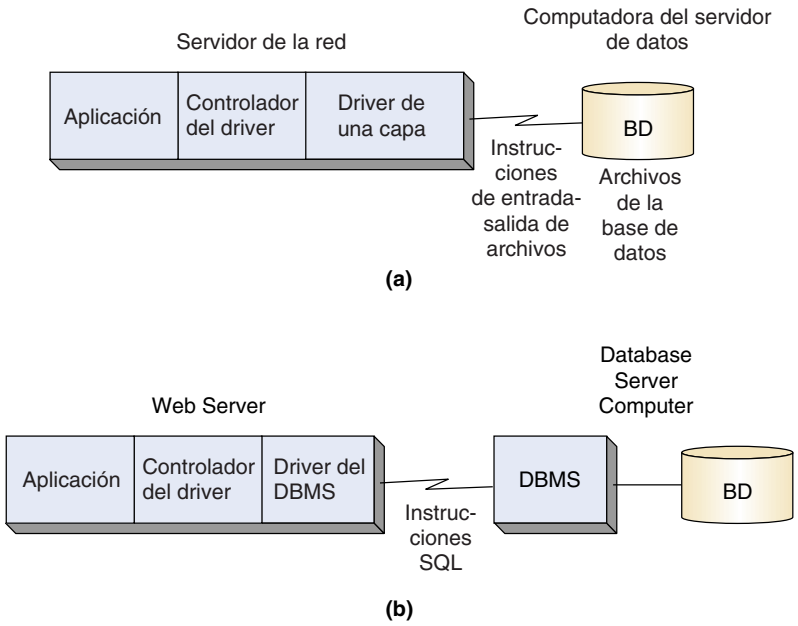
Un **driver** procesa las peticiones del ODBC y transmite instrucciones específicas del SQL a determinado tipo de fuente de datos. Existe un driver diferente para cada tipo de fuente de datos. Por ejemplo, hay drivers para DB2, Oracle, Access y todos los demás productos cuyos proveedores los han elegido para que participen en el estándar ODBC. Los drivers los proporcionan los proveedores de DBMS y las compañías de software independientes.

Es responsabilidad del driver asegurarse de que las órdenes del ODBC se ejecuten en forma correcta. En algunos casos, si la fuente de datos no es por sí misma compatible con SQL, el driver tal vez necesite ejecutar un procesamiento considerable para suplir la falta de capacidad en la fuente de datos. En otros casos, donde la fuente de datos opera completamente con SQL, el driver sólo necesita transferir la petición para su procesamiento mediante la fuente de datos. La unidad también convierte los códigos y mensajes de error de la fuente de datos en códigos y mensajes del estándar ODBC.

ODBC identifica dos tipos de drivers: de una capa y de capas múltiples, o multicapas. Un driver **de una capa** procesa las llamadas al ODBC y las instrucciones del SQL. Un ejemplo de un driver de una capa se muestra en la figura 15-6(a). En este ejemplo, los datos se almacenan en los archivos de base X (el formato utilizado por FoxPro, dBase y otros). Puesto que los controladores de los archivos de Xbase no procesan al SQL, es tarea del driver traducir la petición del SQL en órdenes de manipulación de archivos de Xbase, y transformar de nuevo los resultados en la forma del SQL.

➤ FIGURA 15-6

Tipos de unidades del ODBC: (a) driver de una capa (b) driver multicapas



Un driver **multicapas** procesa las llamadas del ODBC, pero transfiere directamente las peticiones del SQL al servidor de la base de datos. Aunque puede reformatear una petición del SQL para conformarla al lenguaje de una fuente de datos en particular, no procesa el SQL. En la figura 15-6(b) se muestra el uso de un driver multicapas.

Niveles de conformidad

Los creadores del estándar ODBC se enfrentaron a un dilema: si elegían definir un estándar con un nivel mínimo de capacidad, muchos proveedores habrían estado de acuerdo; pero si lo hubieran hecho, el estándar sólo representaría una pequeña parte del poder y la capacidad total del ODBC y del SQL. Por otra parte, si el estándar indicaba un nivel muy alto de capacidad, entonces muy pocos proveedores estarían de acuerdo con el estándar y éste se volvería poco importante. Para solucionar este dilema, el Comité eligió acertadamente definir los niveles de conformidad del estándar. Existen dos tipos: la conformidad del ODBC y la conformidad de SQL.

Nivel de conformidad del ODBC.

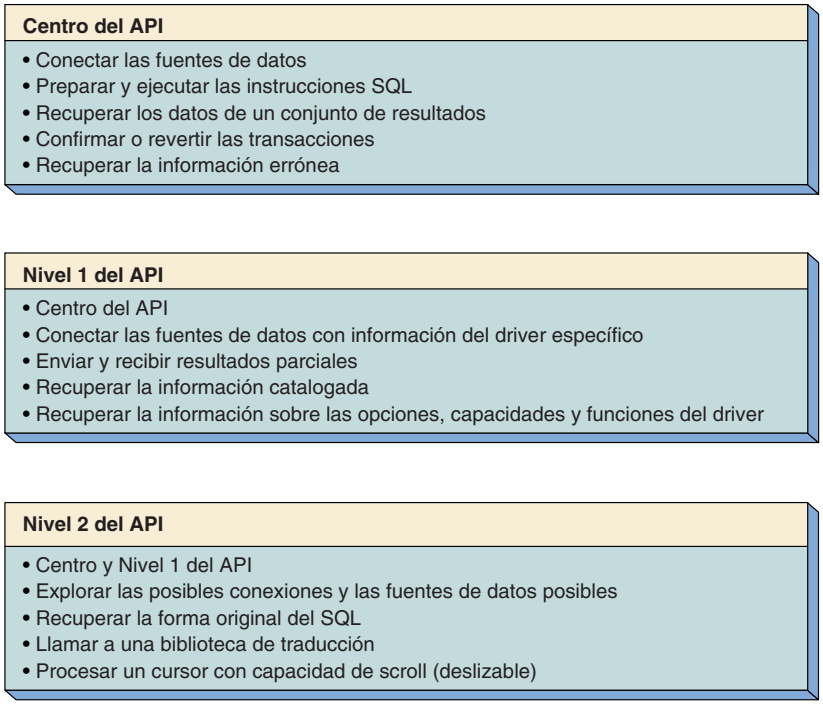
Los **niveles de conformidad del ODBC** se refieren a las características y funciones que se ofrecen mediante la interfaz del programa de aplicación del driver **(API).** Un driver API es un conjunto de funciones que la aplicación puede solicitar para recibir servicios. La figura 15-7 resume los tres niveles de conformidad del ODBC que se indican en el estándar. En la práctica, casi todos los drivers proporcionan por lo menos el Nivel 1 de la conformidad API; por lo tanto, el nivel base de API no es demasiado importante.

Una aplicación puede llamar a un driver para determinar el nivel de conformidad del ODBC que proporciona. Si la aplicación requiere un nivel de conformidad que no está presente, puede finalizar la sesión de manera ordenada y generar mensajes apropiados al usuario. También se puede escribir la aplicación para emplear características de un nivel alto de conformidad, si están disponibles, y trabajar en las funciones faltantes si no hay un nivel alto.

Por ejemplo, los drivers en el Nivel 2 de API deben proporcionar un cursor con capacidad de scroll (deslizable). Mediante el uso de los niveles de conformidad se puede escribir una aplicación para utilizar los cursores, si es que están disponibles; si no lo están, se puede trabajar en la característica faltante, seleccionando los datos que se requieren mediante cláusulas WHERE muy restrictivas. Lo anterior aseguraría que sólo unos pocos renglones serían regresados al mismo tiempo que la aplicación, y procesaría los renglones por medio de un cursor que se mantuviera por sí mismo. El desempe-

➤ FIGURA 15-7

Resumen de los niveles de conformidad del ODBC



ño puede parecer más lenta en el segundo caso, pero cuando menos la aplicación se podría ejecutar con éxito.

NIVEL DE CONFORMIDAD SQL. Los **niveles de conformidad SQL** especifican las instrucciones, expresiones y tipos de datos del SQL que puede procesar un driver. Se definen tres niveles, como se resume en la figura 15-8. La capacidad de la gramática SQL mínima es muy limitada y la mayoría de los drivers soportan por lo menos la gramática del centro del SQL.

Al igual que con los niveles de conformidad ODBC, una aplicación puede llamar al driver para determinar el nivel de conformidad del SQL que soporta. Con tal información, la aplicación puede determinar qué instrucciones del SQL se pueden emitir. Si es necesario, la aplicación puede finalizar la sesión o utilizar medios alternativos, menos potentes, para obtener los datos.

Establecimiento del nombre de una fuente de datos ODBC

Una **fuente de datos** es una estructura de datos ODBC que identifica una base de datos y el DBMS que lo procesa. Las fuentes de datos identifican otros tipos de datos tales como las hojas de cálculo y otros que no son bases de datos, pero que ahora no nos interesan.

Existen tres tipos de fuentes: del archivo, del sistema y del usuario. Una **fuente de datos del archivo** es aquel que pueden compartir los usuarios de una base de datos. El único requisito es que tengan el mismo driver de DBMS y el permiso para ingresar a la base de datos. El archivo de la fuente de datos se puede enviar por correo electrónico, o de lo contrario ser distribuido entre posibles usuarios. Una **fuente de datos del sistema** es local, para una computadora en particular. El sistema operativo y cualquier usuario en dicho sistema (con privilegios propios) puede utilizar una fuente de datos del sistema. La **fuente de datos del usuario** sólo está disponible para el usuario que la creó.

En general, la mejor opción para las aplicaciones de la red es crear una fuente de datos del sistema en el servidor de la red. Entonces, los usuarios del Explorador ingresan al servidor de la red, que, en cambio, utiliza una fuente de datos del sistema para establecer una conexión con el DBMS y la base de datos.

➤ FIGURA 15-8

Resumen de los niveles de conformidad del SQL

Gramática mínima del SQL

- CREATE TABLE, DROP TABLE
- SELECT simple (no incluye subconsultas)
- INSERT, UPDATE, DELETE
- Expresiones simples (A > B + C)
- Tipos de datos CHAR, VARCHAR, LONGVARCHAR

Gramática del centro del SQL

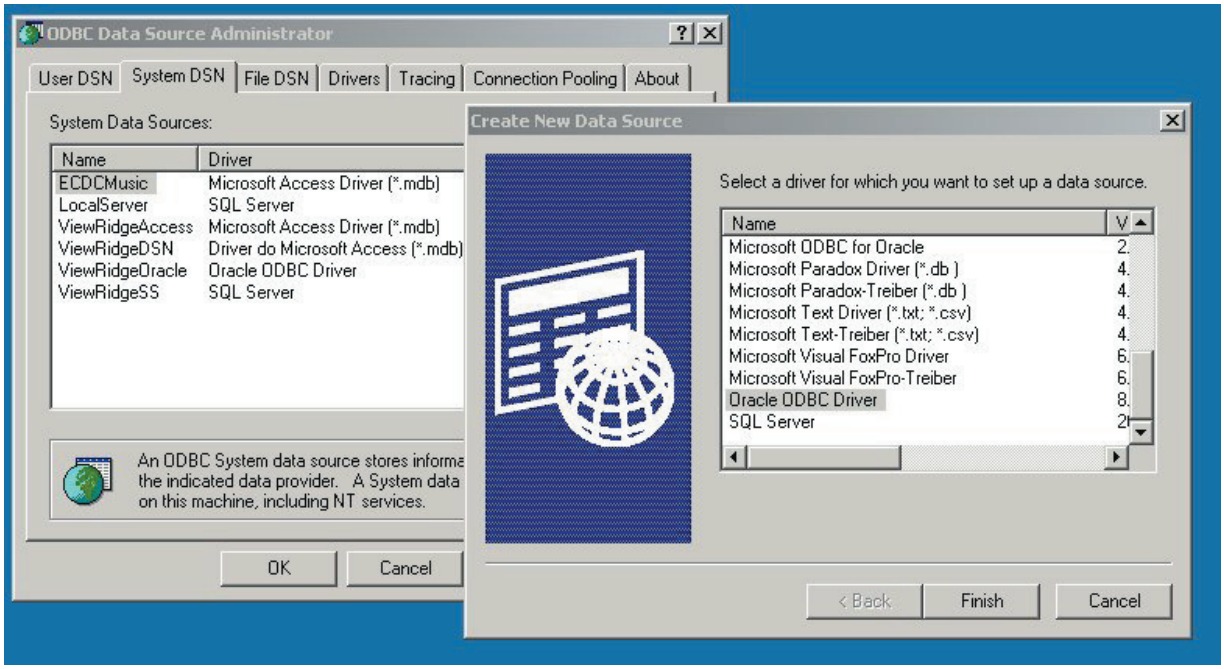
- Gramática mínima del SQL
- ALTER RABLE, CREATE INDEX, DROP INDEX
- CREATE VIEW, DROP VIEW
- GRANT, REVOQUE
- SELECT completo (incluye subconsultas)
- Funciones agregadas tales como SUM, COUNT, MAX, MIN, AVG
- Tipos de datos DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOTA, DOUBLE PRECISION

Gramática extendida del SQL

- Gramática del centro del SQL
- JOINS exteriores
- UPDATE y DELETE mediante las posiciones del cursor
- Funciones escalares tales como SUBSTRING, ABS
- Literales para la fecha, tiempo y timestamp
- Grupo de instrucciones SQL
- Procedimientos almacenados

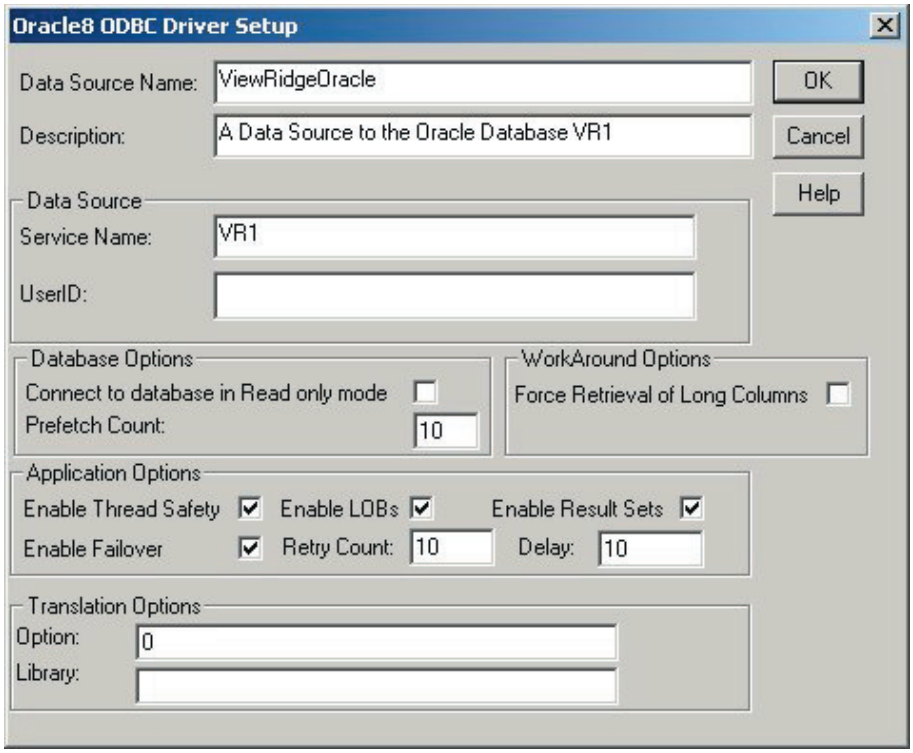
➤ FIGURA 15-9

Creación de una fuente de datos del sistema: (a) selección del driver, y (b) selección de la base de datos



(a)

➤ FIGURA 15-9



(b)

La figura 15-9 muestra el proceso de creación de una fuente de datos del sistema mediante el servicio administrador de la fuente de datos del ODBC, que se puede encontrar a través del panel de control de Windows. En la figura 15-9(a) el usuario está seleccionando el driver de la base de datos de Oracle. Observe que existen dos drivers: uno lo proporciona Microsoft y el otro Oracle. Los drivers pueden tener diferentes capacidades y el usuario debe revisar la documentación para cada uno con el fin de determinar cuál es el más apropiado para su aplicación. Otros drivers que se muestran en esta figura corresponden a Paradox, archivos de texto, y Visual FoxPro.

En la figura 15-9(b) el usuario está seleccionando la base de datos. Aquí, VR1 es el nombre asignado a la base de datos creada en el capítulo 12. Posteriormente, en el presente capítulo, utilizaremos esta DSN, denominada ViewRidgeOracle para procesar dicha base de datos. Asimismo, emplearemos un segundo archivo DSN llamado ViewRidgeSS para procesar la base de datos del SQL Server que creamos en el capítulo 13.

➤ OLE DB

OLE DB es el cimiento del acceso de datos en el mundo Microsoft. Como tal, es importante entender las ideas fundamentales del OLE DB, aun aunque usted sólo trabaje con las interfaces ADO que están en la parte superior. En esta sección presentamos los conceptos esenciales del OLE DB.

OLE DB es una implementación del estándar de los objetos del OLE Microsoft. Los objetos del OLE DB son COM y soportan todas las interfaces que se requieren para dichos objetos. Fundamentalmente, el OLE DB fragmenta las características y funciones de un DBMS en objetos COM. Pueden existir algunos que soporten operaciones de consulta, otros que ejecuten actualizaciones, otros más, la creación de construcciones de esquemas de la base de datos tales como tablas, índices y vistas, e incluso algunos que realicen el control de la transacción, como el bloqueo optimista.

Esta característica vence a una desventaja mayor del ODBC. Con ODBC, un vendedor debe crear un driver del ODBC para casi todas las características y funciones del DBMS, con el fin de participar de algún modo en el ODBC. Ésta es una gran tarea y requiere una inversión inicial cuantiosa. Sin embargo, con el OLE DB un vendedor puede implementar partes de su producto; por ejemplo, sólo el procesador de consultas, participar en el OLE DB y, por lo tanto, proveer ingreso a los usuarios mediante el ADO. Posteriormente, el proveedor puede agregar más objetos e interfaces para aumentar la funcionalidad de su OLE DB.

Este texto no supone que usted sea un programador orientado a objetos, por lo que es necesario desarrollar algunos conceptos. En particular, necesitará entender las abstracciones, métodos, propiedades y colecciones. Una **abstracción** es una generalización de algo. Las interfaces del ODBC son abstracciones de los métodos de acceso originales del DBMS. Cuando se abstrae algo, se pierde detalle, pero se gana habilidad para trabajar con un rango más amplio de géneros.

Por ejemplo, un **recordset** es una abstracción de una relación. En dicha abstracción se define un recordset para obtener ciertas características que serán comunes para todos los recordset. Por ejemplo, cada uno tiene un conjunto de columnas denominadas campos. Ahora, el objetivo de la abstracción es capturar todo lo importante, pero omitir los detalles que no necesitan los usuarios de la abstracción. Por consiguiente, las relaciones de Oracle pueden tener alguna característica que no se represente en un recordset; lo mismo puede aplicarse para las relaciones en el SQL Server, AS/400, DB2 y otros productos DBMS. Estas características únicas se perderán en la abstracción, pero si ésta es buena, eso no importará.

Adelantándose un nivel, un **rowset** es la abstracción del OLE DB de un recordset. Ahora bien, ¿por qué el OLE DB necesita definir otra abstracción? Porque el OLE DB indica las fuentes de datos que no son tablas, pero que poseen *algunas de* las características de las éstas. Considere todas las direcciones de correo electrónico en su archivo personal. ¿Dichas direcciones son lo mismo que una relación? No, pero comparten algunas de las características que poseen las relaciones. Cada dirección es un grupo semánticamente relacionado de elementos de datos. Como en los renglones de una tabla, es adecuado ir a la primera, trasladarse a la segunda, y así sucesivamente. Pero a diferencia de las relaciones, no todas son del mismo tipo. Algunas son de personas y otras para listas de correos. Por lo tanto, cualquier acción en un recordset que dependa de éste y que sea de un mismo tipo de cosas, no se puede utilizar en el rowset.

Trabajando de principio a fin, el OLE DB define un conjunto de propiedades y comportamientos de los datos para los rowsets. Cada conjunto posee dichas propiedades y comportamientos. Además, el OLE DB define un recordset como un subtipo de rowset. Los recordset poseen todas las propiedades y comportamientos que tienen los rowset; asimismo, poseen algunas características que son exclusivas de los recordset.

La abstracción es común y útil. Usted oirá hablar de las abstracciones del control de transacciones, o de las abstracciones de las consultas, o de las interfaces. Esto significa, simplemente, que ciertas características de un conjunto de cosas se definen formalmente como un tipo.

Un objeto de la programación orientada a objetos es una abstracción que se define por sus propiedades y métodos. Por ejemplo, el objeto de un recordset tiene la propiedad PermitirEdiciones, un TipodeConjuntosdeRegistros y una propiedad EOF. Estas **propiedades** representan las características de la abstracción del recordset. Un objeto también tiene acciones que se puede ejecutar, llamadas **métodos.** Un recordset tiene métodos, tales como Abrir, MoverPrimero, MoverSiguiente y Cerrar.

Estrictamente hablando, la definición de la abstracción de un objeto se denomina **clase de objetos,** o sólo clase. Una instancia de una clase de objetos, como por ejemplo un recordset particular, se llama un objeto. Todos los objetos de una clase tienen los mismos métodos y las mismas propiedades, pero los valores de dichas propiedades varían de objeto a objeto.

El último término al que necesitamos referirnos es la colección. Una **colección** es un objeto que contiene un grupo de otros objetos. Un recordset tiene una colección de otros objetos llamados campos. La colección posee propiedades y métodos. Una pro-

piedad de todas las colecciones es Cuenta, que es el número de los objetos en la colección. Por lo tanto, conjuntoderegistros.Campos.Cuenta es el número de los campos en la colección. En el ADO y el OLE DB, las colecciones se nombran conforme al plural de los objetos que coleccionan. Por consiguiente, existe una colección de campos de los objetos de campo; una colección de errores de los objetos de error; otra de parámetros de los parámetros, etc. Un método importante de una colección es un repetidor, o iterador, el cual es un método que puede emplearse para pasar a través de la colección, o de lo contrario identificar los elementos en ésta.

Si se siente frustrado con todas estas definiciones, no se rinda. ¡Apreciará el uso práctico de estos conceptos antes de que termine este capítulo!

Objetivos de OLE DB

Los propósitos principales del OLE DB se listan en la figura 15-10. Como mencionamos, el OLE DB primero fragmenta la funcionalidad y los servicios del DBMS en piezas de objetos. Esta partición representa una gran flexibilidad para los **consumidores de datos** (usuarios de la funcionalidad del OLE DB) y para los **proveedores de datos** (vendedores de los productos que distribuyen la funcionalidad del OLE DB). Los consumidores de datos sólo toman los objetos y la funcionalidad que necesitan; un dispositivo inalámbrico para leer una base de datos puede tener una huella muy pequeña. A diferencia del ODBC, los proveedores de datos sólo necesitan implementar una parte de la funcionalidad del DBMS. Asimismo, esta partición significa que los proveedores de datos pueden distribuir las capacidades en interfaces múltiples.

Conviene ampliar el concepto anterior. Una interfaz de objetos es un paquete de objetos. Una **interfaz** está especificada mediante un conjunto de objetos y las propiedades y métodos que expone. Un objeto no necesita mostrar todas sus propiedades y métodos en una interfaz determinada. Por lo tanto, un objeto de un recordset sólo mostraría los métodos de lectura en una interfaz de consulta; pero mostraría los métodos de crear, actualizar y eliminar en una interfaz de modificación.

La manera en que el objeto maneja la interfaz, o la **implementación,** queda completamente oculta para el usuario. De hecho, los programadores de un objeto tienen la libertad de cambiar la implementación cuando quieran. ¿Quién lo sabrá? ¡Sin embargo, jamás podrán cambiar la interfaz sin incurrir en el rechazo justificable de sus usuarios!

El OLE DB define interfaces estandarizadas. Sin embargo, los proveedores de datos tienen la libertad de agregar interfaces encima de los estándares básicos. Dicha extensi-

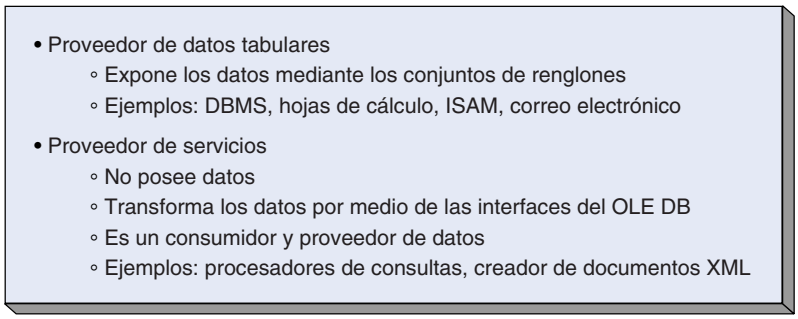
➤ FIGURA 15-10

Objetivos del OLE DB

- Crear interfaces de objetos para las piezas funcionales del DBMS
  - Consulta
  - Actualización
  - Control de transacciones
  - Etcétera
- Aumentar la flexibilidad
  - Permitir a los consumidores de datos utilizar solamente los objetos que necesiten
  - Permitir a los proveedores de datos exponer las piezas de la funcionalidad del DBMS
  - Los proveedores pueden distribuir la funcionalidad en interfaces múltiples
  - Las interfaces son estandarizadas y extensibles
- La interfaz de objetos sobre cualquier tipo de datos
  - Base de datos relacional
  - ODBC u original
  - Bases de datos no relacionales
  - VSAM y otros archivos
  - Correo electrónico
  - Otros
- No obligar a los datos a que sean convertidos, o trasladarlos desde donde se encuentran

➤ FIGURA 15-11

Dos tipos de proveedores de datos del OLE DB



bilidad es esencial para el siguiente propósito: proporcionar una interfaz de objetos a cualquier tipo de datos. Las bases de datos relacionales pueden ser procesadas mediante los objetos del OLE DB que utilizan el ODBC o los drivers propios del DBMS. Como se indica, el OLE DB incluye soporte para los otros tipos.

El resultado neto de estos objetivos de diseño es que los datos no necesitan convertirse de una forma a otra, ni moverse de una fuente de datos a otra. El servidor de la red en la figura 15-1 puede utilizar el OLE DB para procesar los datos en cualquiera de los formatos, justo donde éstos residen. Lo anterior quiere decir que las transacciones pueden extenderse sobre las fuentes de datos múltiples y distribuirse en diferentes computadoras. La provisión del OLE DB para esto es el **Controlador de Transacciones Microsoft** (MTS), pero su análisis está más allá del alcance de este capítulo.

Construcciones básicas del OLE DB

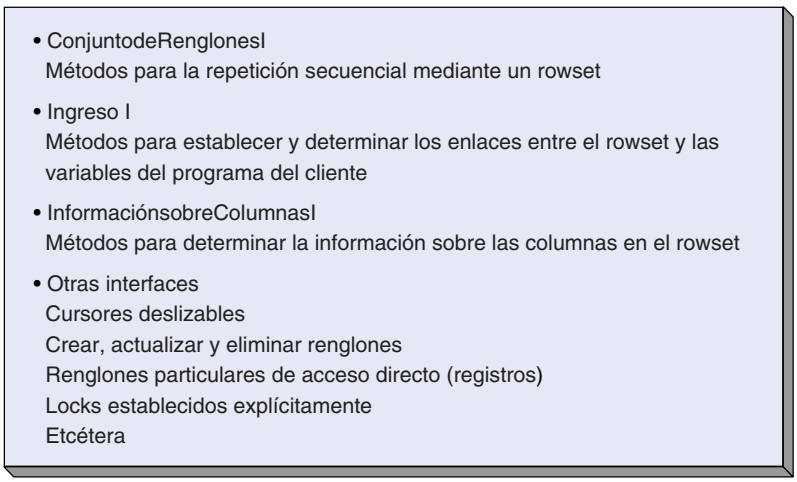
Como se muestra en la figura 15-11, el OLE DB tiene dos tipos de proveedores de datos. Los **proveedores de datos tabulares** presentan sus datos mediante los rowsets. Algunos ejemplos son los productos DBMS, hojas de cálculo y procesadores de archivos ISAM, como Dbase y FoxPro. Además, otros tipos de datos como el correo electrónico también pueden presentarse en los rowsets. Los proveedores de datos tabulares llevan a los datos de cierto tipo al mundo del OLE DB.

Por otra parte, un **proveedor de servicios** es un transformador de datos. Los proveedores de servicios aceptan los datos del OLE DB de un proveedor de datos tabulares del OLE DB y los transforman de alguna manera. Los proveedores de servicios son consumidores y proveedores de datos transformados. Un ejemplo de un proveedor de servicios es aquel que obtiene datos de un DBMS relacional y los transforma en documentos XML.

El objeto **rowset** es fundamental para el OLE DB; los rowsets son equivalentes a lo que denominamos **cursores** en el capítulo 11; de hecho los dos términos se utilizan como sinónimos. La figura 15-2 lista las interfaces básicas del rowset que son sustenta-

➤ FIGURA 15-12

Interfaces del rowset



das. El ConjuntodeRenglonesI proporciona métodos de objetos para seguir sólo el movimiento secuencial a través de un rowset. Cuando se declara sólo un cursor forward (hacia delante) en el OLE DB, se está invocando la interfaz ConjuntodeRenglonesI. La interfaz IngresoI se utiliza para enlazar las variables del programa con los campos del rowset. Cuando se utiliza el ADO, la mayor parte de esta interfaz se esconde debido a que se emplea tras bambalinas en el motor de scripting. No obstante, si trabaja bibliotecas de tipos en VB puede utilizar los métodos de esta interfaz.

La interfaz de InformaciónsobreColumnasI tiene métodos para obtener información sobre las columnas en un rowset. Emplearemos esta interfaz para aprovechar dos de los ejemplos del ADO al final de este capítulo. El ConjuntodeRenglonesI, IngresoI, e InformaciónsobreColumnasI son las interfaces básicas del rowset. Se definen otras interfaces para operaciones más avanzadas, tales como cursores deslizables, actualización, acceso directo a renglones específicos, locks explícitos, etcétera.

Considere estas interfaces en el contexto de dos rowsets —una que es una relación tradicional y otra que es una colección de direcciones de correo electrónico—. Las primeras tres interfaces pertenecen fácilmente a cualquiera de los tipos del rowset. El último conjunto de interfaces parecerá diferente en cuanto a las características y funciones entre los dos rowsets, si es que se pertenecen del todo. Una nota final: los rowsets pueden contener indicaciones de objetos, así que con ellos se pueden crear estructuras muy complicadas.

➤ ADO (OBJETOS DE DATOS ACTIVOS)

ADO es un modelo sencillo de objetos que pueden usar los usuarios de datos para procesar cualesquiera de los datos del OLE DB. Puede ser llamado desde los lenguajes de script como JScript y VBScript, y desde Visual Basic, Java, C#, y C++. Microsoft ha declarado que reemplaza todos los otros métodos de acceso de datos, por lo tanto, es importante aprenderlo, no sólo para las aplicaciones de bases de datos de la tecnología Internet, sino para cualquier aplicación de datos que utilice los productos Microsoft.

Debido a las abstracciones y a la estructura de los objetos del OLE DB, el modelo de objetos y las interfaces son lo mismo, a pesar del tipo de datos procesados. Por consiguiente, un analista que aprenda ADO para procesar una base de datos relacional puede emplear dicho conocimiento para procesar también un directorio de correo electrónico. Las características del ADO se listan en la figura 15-13.

Invocación del ADO desde páginas de servidor activas (ASP)

En este capítulo invocaremos al ADO en un servidor de la red mediante las páginas de servidor activas. Estas páginas contienen una mezcla del DHTML (o XML) y de las instrucciones del lenguaje de programa expresadas en VBScript o JavaScript. En este capítulo utilizaremos el VBScript. Las páginas ASP se pueden escribir con cualquier editor de textos, pero se escriben más fácilmente con FrontPage o con algún producto similar autorizado por la página Web.

El Servidor de Información de Internet (IIS) es un servidor de la red construido en Windows 2000 Profesional y en Windows para Trabajo en Grupo. Como se estableció en el capítulo 14, ASP es una extensión del ISAPI al IIS. Prácticamente, esto sólo significa que cada vez que el IIS recibe un archivo con la extensión .asp envía el archivo al programa del ASP para su procesamiento.

➤ FIGURA 15-13

Características del ADO

- Modelo simple de objetos para los consumidores de datos del OLE DB
- Puede utilizarse desde VBScript, JScript, Visual Basic, Java, C#, C++
- Estándar simple de acceso de datos de Microsoft
- Los objetos de acceso de datos son los mismos para todos los tipos de los datos del OLE DB

Cualquier instrucción del lenguaje del programa incluida dentro de los caracteres <%...%> será procesada en la computadora del servidor de la red. Otras instrucciones del lenguaje se enviarán al explorador del usuario para procesarlas. En el presente capítulo todo este código se procesará en el servidor de la red.

Para invocar sus páginas ASP colóquelas en algún directorio; por ejemplo, C:\MiDirectorio. Después, abra el IIS y establezca un directorio virtual que apunte al directorio en el cual ha colocado sus páginas ASP. Puede realizar lo anterior si abre los Servicios de Información de Internet, y hace clic derecho en el Sitio Web predeterminado. Elija el Directorio Virtual/Nuevo y aparecerá un asistente que le pedirá su directorio virtual y el directorio real (aquí, C:\MiDirectorio) en los cuales estarán sus páginas ASP. Seleccione Leer y Ejecutar scripts en el tercer panel del asistente. Para los ejemplos de este texto se empleará el nombre del directorio virtual Ejemplo1ViewRidge.

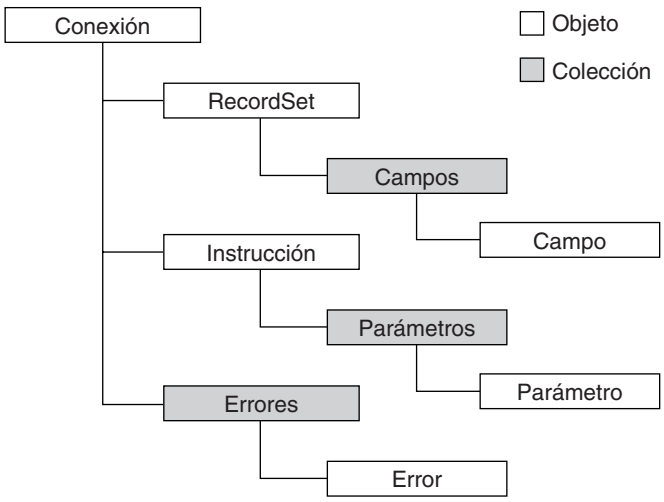
No es necesario que el DBMS y el servidor de la red se localicen en la misma máquina. Cuando cree el ODBC DSN puede señalar una base de datos en una computadora diferente que sea accesible desde la computadora del servidor de la red. Esto será más sencillo si es que otra ejecuta un sistema operativo de Windows, pero también se puede realizar con otros sistemas operativos. Por supuesto, el servidor de la red y el DBMS pueden estar en la misma máquina.

Modelo de objetos del ADO

El modelo de objetos del ADO que se muestra en la figura 15-14 está construido sobre el modelo de objetos del OLE DB. El objeto Conexión es el primero del ADO que se creará y es la base para los otros. A partir de una conexión, un programador puede crear uno o más objetos del RecordSet y uno o más de instrucciones. En el proceso de creación, o de trabajo con cualquiera de estos objetos, ADO colocará cualquier error generado en la colección Errores.

Cada objeto RecordSet tiene una colección Campos; cada elemento del Campo corresponde a una columna en el recordset. Además, cada objeto orden tiene una colección de Parámetros que contiene objetos para los parámetros de la instrucción.

OBJETO DE CONEXIÓN. El siguiente código VBScript puede depositarse en una página ASP para crear un objeto de conexión. Después de que se ejecute, el objeto de conexión variable señalará un objeto que esté conectado con la fuente de datos de ODBC ViewRidgeSS.



➤ FIGURA 15-14

Modelo de objetos del ADO

```
<%
Dim objetodeConexión
Set objetodeConexión = Server.CrearObjeto (“ADODB.conexión”)
objetodeconexión.NiveldeAislamiento = Lectura flexibleadXact ’usar ADOVBS
objetodeConexión.Open “ViewRidgeSS”, “sa” ’abrir la conexión
%>
```

En este código, la instrucción Server.CrearObjeto está invocando el método de creación de un objeto del objeto servidor ASP. El tipo del objeto, aquí ADODB.conexión, se admite como un parámetro. Después de que se ejecuta esta instrucción, la variable objeto de Conexión señala el nuevo objeto de conexión. Posteriormente, el nivel de aislamiento se establece mediante una constante del archivo ADOVBS. Este archivo puede estar a la disposición de esta escritura incluyendo la siguiente instrucción:

```
<!--#incluir virtual = “Ejemplo1ViewRidge/ADOVBS.inc-->>
```

Esta instrucción debe ser parte del archivo del ASP, pero está **fuera de** <%...%>. Para que esto funcione, debe copiar el archivo ADOVBS.inc en su directorio (encuéntrelo mediante Buscar). También, sustituya el nombre de su directorio virtual, si es diferente al Ejemplo1ViewRidge.

Los nombres y valores de las constantes importantes del ADOVBS se listan en la figura 15-15. La utilización de los nombres de las constantes en vez de sus valores hace que su código sea más legible. Asimismo, es más fácil que adapte su código para cuando Microsoft deba cambiar los significados de estos valores (sería muy raro, pero podría ocurrir).

En la última instrucción, el método abierto de la conexión se utiliza para abrir la fuente de datos del ODBC. La cuenta del usuario “sa” se admite como un parámetro. Si se requiriese de una contraseña, ésta se admitiría como el siguiente parámetro. Obviamente, ésta es una seguridad terrible, no obstante, no es tan mala como parece debido a que este código se ejecuta en el servidor de la red y nunca se envía a la red de trabajo. Sin embargo, sería mejor obtener el nombre y la contraseña del usuario al momento de la ejecución. Este tema no está directamente relacionado con el procesamiento de una base de datos y, por lo tanto, no lo abordaremos.

En este punto, se ha establecido una conexión con el DBMS mediante la fuente de datos del ODBC y la base de datos está abierta. El indicador del objeto de conexión se puede emplear para referirse a cualquier otro método para una conexión (véase la figura 15-14), incluyendo la creación y el uso de los objetos del recordset y de instrucción. La colección de errores también se puede procesar.

OBJETO DEL RECORDSET. Debido a que la conexión tiene una base de datos abierta, lo siguiente creará un objeto del recordset (de ahora en adelante omitiremos el <% y %>, pero todos estos ejemplos de códigos deberán insertarse entre ellos, o no los ejecutará el servidor de la red):

```
Dim objetodelconjuntoderegistros, varSql
varSQL = “SELECT * FROM ARTISTA
Set objetodelconjuntoderegistros = Server.CreateObject (“ADODB.Conjuntoderegistros”)
objetodelconjuntoderegistros.CursorType = adOpenStatic.
objetodelconjuntoderegistros.LockType = adLockReadOnly
objetodelconjuntoderegistros.Open varSQL.objetodeConexión
```

El Tipo de cursor (CursorType) y el Tipo de bloqueo (LockType) también se pueden admitir como parámetros en el método abierto del recordset como sigue:

```
Dim objetodelconjuntoderegistros, varSql
varSQL = “SELECT * FROM ARTISTA”
Set objetodelconjuntoderegistros = Server.CreateObject (“ADODB.Conjuntoderegistros”)
objetodelconjuntoderegistros.Open varSQL, objetodeConexión, adOpenStatic,
adLockReadOnly
```

➤ FIGURA 15-15

Valores constantes del procesamiento del ADO

Nivel de aislamiento	Nombre de la constante	Valor
Lecturas sucias	adXactReadUncommitted	256
Lectura confirmada	adXactReadCommitted	4096
Lectura repetible	adXactRepeatableRead	65536
Serializable	adXactSerializable	1048576

(a) Niveles de aislamiento

Tipo de cursor	Nombre de la constante	Valor
Forward only	adOpenForwardOnly	0
Key Set	adOpenKeyset	1
Dinámico	adOpenDynamic	2
Estático	adOpenStatic	3

(b) Tipos de cursores

Tipo de cursor	Nombre de la constante	Valor
Sólo lectura	adLockReadOnly	1
Bloqueo pesimista	adLockPessimistic	2
Bloqueo optimista	adLockOptimistic	3
Optimista con actualizaciones en grupo	adBatchOptimistic	4

(c) Tipos de bloqueos

De cualquier manera, estas instrucciones provocan que la instrucción SQL en varSQL se ejecute mediante un cursor estático de sólo lectura. Todas las columnas de la tabla ARTISTA se incluirán como campos en el recordset. Si la instrucción SELECT sólo nombra dos columnas, digamos, "SELECT IdentificacióndelArtista, Nacionalidad FROM ARTISTA", entonces sólo se incluirán esas dos columnas como campos en el recordset.

A propósito, cabe hacer un comentario sobre excepciones. Con SQL Server, si el nombre de una tabla tiene espacios o caracteres raros, o si es una palabra reservada de SQL Server, usted deberá incluir el nombre entre corchetes [ ]. Sin embargo, para Oracle el nombre de una tabla incluido entre corchetes es ilegal. En Oracle, cuando las tablas tienen un nombre extraño, éste se incluye entre comillas. Por consiguiente, si su base de datos tiene dichas tablas, tendrá que escribir un código diferente, dependiendo si está utilizando una base de datos de Oracle o del SQL Server.

COLECCIÓN DE CAMPOS. Una vez que se ha creado el recordset, se activa su colección de campos. Se puede procesar dicha colección con lo siguiente:

```
Dim variableI, variabledelNúmerodecolumnas, objetodeCampo
variabledelNúmerodecolumnas = Objetodelconjuntoderegistros.Campos.Cuenta
For var I = 0 to variabledelnúmerodecolumnas – 1
  Set Objetodecampo = Objetodelconjuntoderegistros.Campos(varI)
  'Objetodecampo.Nombre tiene ahora el nombre del campo
  'Objetodecampo.Valor tiene ahora el nombre del campo
  'puede hacer algo con ellos aquí
Next
```

En la segunda instrucción, variabledelNúmerodecolumnas se establece como el número de columnas en el recordset, mediante el acceso de la propiedad Cuenta de la Colección de campos. Entonces se ejecuta un circuito para repetir esta colección. La propiedad Campos(0) se refiere a la primera columna del recordset, por lo tanto el circuito necesita correr de 0 a la cuenta –1.

En este ejemplo no se lleva a cabo nada con los objetos de los campos, pero en una aplicación real el programador puede consultar objetoCampo.Nombre para obtener el nombre de una columna, y objetoCampo.Valor para obtener su valor. Verá usos como éste en los siguientes ejemplos.

CONJUNTO DE ERRORES. Cada vez que se presenta un error, el ADO activa una colección de errores. Debe ser una colección, puesto que una instrucción simple del ADO puede generar más de un error. Esta colección puede procesarse de un modo similar al que se utilizó para la colección de campos:

```
Dim varI, variabledelaCuentadeErrores, objetodeError
on Error Resume Next
VariabledelaCuentadeErrores = objetodeConexión.Errores.Cuenta
If variabledelaCuentadeErrores > 0 Then
    For varI = 0 to variabledelaCuentadeErrores –1
    Set objetodeError = objetodeConexión.Errores (varI)
    'objetodeError.Descripción contiene
    'una descripción del error
  Next
End If
```

En el ciclo, objetodeerror se establece como objetoconexión.errores(varI). Observe que esta colección pertenece al objeto de conexión, y no al del recordset. Asimismo, la propiedad Descripción Objetodeerror puede emplearse para desplegar el error al usuario.

Por desgracia, VBScript tiene un procesamiento de errores muy limitado. El código para verificarlos (empezando con On Error Resume Next) se debe colocar después de cada instrucción de objeto del ADO que pueda generar un error. Debido a que esto puede darle un tamaño no deseado al código, será mejor escribir un error que sostenga la función, y llamarlo después de cada invocación del objeto del ADO.

OBJETO ORDEN (COMANDO). Orden ADO se utiliza para ejecutar las consultas y procedimientos almacenados que se guardan con la base de datos. La colección de parámetros de Orden se emplea para admitirlos. Por ejemplo, suponga que la base de datos abierta mediante el objeto de conexión tiene un procedimiento almacenado que se llama EncontrarArtista el cual acepta un parámetro: la nacionalidad de los artistas que se recuperarán. El siguiente código invocará este procedimiento almacenado con el valor del parámetro “Español” y creará un recordset nombrados objRs que contiene los resultados del procedimiento almacenado:

```
Dim objCommand, objetodelParámetro, objetoRs

'Crear la instrucción objeto, conectarla con objetodeConexión y establecer su formato

Set objCommand = Server.CreateObject ("ADODB.instrucción")
Set objCommand.ConexiónActiva = objetodeConexión
objCommand.textodelaInstrucción = "{llamada EncontrarArtista(?)}"

'Establecer el parámetro con el valor necesario

Set objetodelParámetro = objetodeOrden.CrearParámetro ("Nacionalidad", adChar,
⇒ adParamInput, 25)
objCommand.Parameters.Append objetodelParámetro

objetodelParámetro.Valor = "Español"

'Activar el procedimiento almacenado
Set objetodelConjuntodeRegistros = objCommando.Executar
```

En este ejemplo se crea primero un objeto del tipo ADODB.instrucción y después se establece la conexión de dicho objeto con el objeto de conexión. Entonces declara el formato de la llamada al procedimiento almacenado mediante el establecimiento de la propiedad Texto de la orden. Esta propiedad se utiliza para aprobar el nombre del procedimiento almacenado y el número de parámetros que tiene. El signo de interrogación denota un parámetro. Si existiesen tres parámetros, el texto de la orden se establecería como {llamada EncontrarArtista (?,?,?)}"

Posteriormente se crea un objeto para el parámetro. Los valores adChar y adParamInput son de ADOVBS e indican que el parámetro es del tipo char y se utiliza como la entrada al procedimiento almacenado. La extensión máxima del valor del parámetro es 25. Una vez que se ha creado, se debe agregar a la instrucción con el método Append. Finalmente, el procedimiento almacenado se invoca mediante el método Execute. En este punto, se ha creado un recordset llamado objRs con los resultados del procedimiento almacenado.

➤ EJEMPLOS DE ADO

Los cinco ejemplos siguientes muestran cómo solicitar el ADO desde VBScript mediante las Páginas Activas del Servidor. Estos ejemplos se concentran en el uso del ADO y no en las gráficas, presentación o flujo de trabajo. Si desea una aplicación llamativa y más eficiente, será mejor que modifique estos ejemplos para obtener tal resultado. Aquí solamente aprenderá cómo se utiliza el ADO.

Todos estos ejemplos procesan la base de datos View Ridge. En algunos de ellos, se conecta la base del SQL Server a ViewRidgeSS. En otros, se conecta la base de datos Oracle a ViewRidgeOracle. De hecho, en este último ejemplo, ¡sólo se necesita modificar una instrucción para cambiarse de SQL Server a Oracle! Esto es sorprendente y exactamente lo que los creadores del ODBC deseaban cuando hicieron la especificación del ODBC.

Como veremos en el último capítulo, HTTP no mantiene estados. Sin embargo, el procesador ASP mantendrá el estado de la transacción. Para cada una, mantiene un conjunto de variables de la sesión. En estos ejemplos utilizaremos las variables de la sesión para preservar los objetos de conexión. La instrucción:

```
Set Sesión("abc") = "Wowzers"
```

crea una variable de sesión nombrada "abc" y le da el valor "Wowzers". A continuación presentamos más ejemplos útiles.

Para ejecutar cualquiera de estas páginas, abra su explorador y teclee:

http://localhost/ViewRidgeExample1/artist.asp

si está ejecutando en la misma computadora donde se encuentra IIS. De lo contrario, escriba el nombre o la dirección de su servidor en lugar de "localhost".

Ejemplo 1—Lectura de una tabla

La figura 15-16 muestra una página ASP que expone los contenidos de la tabla ARTISTA. En ésta, y en las diversas figuras siguientes, las instrucciones están incluidas dentro de los símbolos <% %>. Cualquier código que se transfiera al explorador, en este caso principalmente el HTML, se muestra en negritas.

La parte superior de la página es HTML estándar. La primera sección del código del servidor crea un objeto de conexión y después un objeto del recordset que contiene los resultados de la instrucción SQL.

```
SELECT    ArtistName.Nationality
FROM      ARTIST
```

Debido a que los objetos de conexión son costosos, en términos del tiempo para crearlos y de la memoria que se utiliza, este código preserva el objeto conexión en la sesión conexión-variable. La primera vez que el usuario invoque esta página no existirá el objeto de conexión. En este caso, la llamada de función IsObject(Session("_conn")) resultará falsa puesto que no se ha establecido _conn. El código después de Else se ejecutará para crear el objeto de conexión. Posteriormente, se creará el recordset para la instrucción seleccionada en variable varSql.

La siguiente sección de la página del ASP contiene el HTML para el explorador. Le siguen diversas porciones del código script del servidor entremezcladas con el HTML. La instrucción On Error Resume Next anula el procedimiento de errores del motor de script del ASP para continuarla. En lugar de eso se procesaría una página mejor.

La última parte de la página produce simplemente el HTML y lo llena con valores de lectura. El circuito objRecordSet.MoveFirst. . . . MoveNext es la lógica para el procesamiento secuencial estándar de un archivo.

El resultado de esta página ASP se muestra en la figura 15-17. No hay nada espectacular sobre esta página ni sobre este archivo ASP, excepto lo siguiente: si estuviera en Internet, ¡cualquiera de las más de 250 millones de personas en el mundo podría verla! No necesitaría ningún software diferente al que tiene su computadora.

Ejemplo 2—Lectura de una tabla en un modo generalizado

El primer ejemplo hizo un uso mínimo de los objetos del ADO en el modelo de objetos. Este ejemplo puede ampliarse al utilizar la colección de campos. Suponga que se quisiera tomar el nombre de una tabla como entrada y desplegar todas las columnas en ella, con excepción de la llave sustituta.

La página ASP en la figura 15-18 llevaría a cabo esta tarea, excepto que el nombre de la tabla está establecido en la variable varTableName. El siguiente ejemplo mostrará cómo obtener un valor para el nombre de la tabla deseada mediante el procesamiento de la forma HTML.

La primera parte del script del servidor tiene la misma función que en la figura 15-16. La única diferencia es que abre la fuente de datos Oracle.

La variable varSql se establece mediante la variable varTableName. El & es un operador que concatena dos cadenas de caracteres. El resultado de esta expresión es la condición:

```
SELECT * FROM CUSTOMER
```

➤ FIGURA 15-16

Artist.asp

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Artist</TITLE>
</HEAD>
<!--#include virtual="ViewRidgeExample1/adovbs.inc"-->
<BODY>
<%
Dim objConn, objRecordSet,varSql

      If IsObject(Session("_conn")) Then ' if already have a connection, use it
           Set objConn=Session("_conn")
      Else
           Set objConn = Server,CreateObject("ADODB.connection") ' get connection
           objConn.open "ViewRidgeSS", "sa" ' open SQL Server ODBC file as user "sa" with no password
           objConn.IsolationLevel = adXactReadCommitted ' avoid dirty reads
           Set Session("_conn") = objConn
      End If

   Set objRecordSet = Server.CreateObject("ADODB.Recordset") ' create the record set objetc

   varSql = "SELECT ArtistName, Nationality FROM ARTIST" ' set up SQL command
   objRecorset.Open varSql, objConn, adOpenStatic, adLockReadOnly ' static with no need to update %>

<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=5><FONT FACE="Arial"
COLOR=#000000><CAPTION><B>ARTIST</B></CAPTION></FONT>
<THEAD>
<TR>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000><FONT SIZE=2 FACE="Arial"
COLOR=#000000>Name</FONT></TH>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000><FONT SIZE=2 FACE="Arial"
COLOR=#000000>Nationality</FONT></TH>

</TR>
</THEAD>
<TBODY>
<%
On Error Resume Next
objRecordSet.MoveFirst
do while Not objRecordSet.eof
%>
<TR VALIGN=TOP>
<TD BORDERCOLOR=#c0c0c0><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=Server.HTMLEncode(objRecordSet("ArtistName"))%><BR></FONT></TD>
<TD BORDERCOLOR=#c0c0c0><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=Server.HTMLEncode(objRecordSet("Nationality"))%><BR></FONT></TD>

</TR>

<%
End If
Next%>
</TR>
```

➤ FIGURA 15-16

(Continuación)

```
<%
objRecorset.MoveNext
loop%
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

También observe que el nombre de la tabla se incluye en el título de la tabla del HTML con el código <CAPTION><B><%=varTableName%></B>></CAPTION>. El código dentro de los % generará el valor de varTableName para colocarlo en el HTML para el título.

El siguiente conjunto de instrucciones script del servidor procesa la colección de Fields. La variable varNumCols se establece como la propiedad de cuenta de la colección de campos, y entonces la colección se repite en el ciclo. Observe cómo se entremezcla el HTML en el código del servidor (o cómo el código del servidor se entremezcla en el HTML, dependiendo de su punto de vista). La variable del nombre de la llave se ha establecido previamente como el nombre de la llave sustituta, por lo tanto el ciclo verifica para determinar que el nombre del actual objeto del campo no tenga el nombre de la llave sustituta. En caso contrario, se genera el HTML para crear el encabezado de la tabla. En la siguiente página se emplea un ciclo similar para multiplicar la tabla con los valores del recordset.

➤ FIGURA 15-17

Resultado de Artist.asp de la figura 15-16

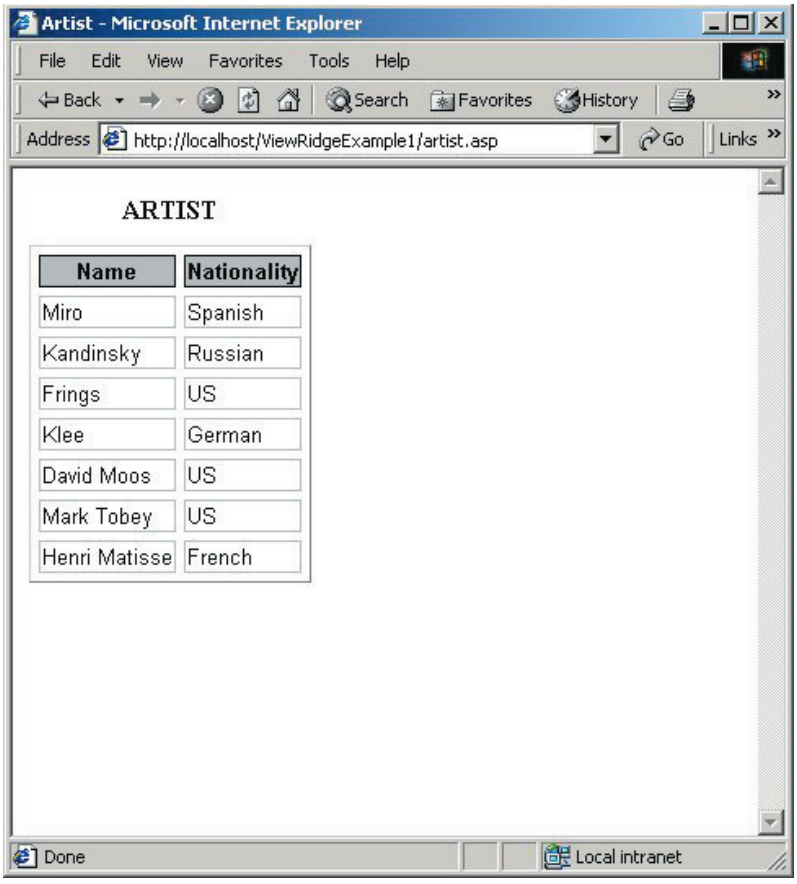


FIGURA 15-18

CustomerOracle.asp

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Table Display Page</TITLE>
</HEAD>
<BODY>
<!--#include virtual="ViewRidgeExample1/adovbs.inc"-->
<%
   Dim objConn, objRecorSet, objField
   Dim varNumCols, varI, varSql
   Dim varTableName, varKeyName

   varTablename = "CUSTOMER"
   varKeyName = "CUSTOMERID"

   If IsObject (Session("_conn")) Then ' if already have a connection, use it
      Set objConn = Session("_conn")
   Else
      Set objConn = Server.CreateObject("ADODB.connection") ' get connection

      ' open Oracle ODBC file using system account with manager password
      objConn.open "ViewRidgeOracle", "system", "manager"
      objConn.IsolationLevel = adXactReadCommtted ' avoid dirty reads
      Set Session("_conn") = objConn
   End if

   Set objRecordSet = Server.CreateObject("ADODB.RecordSet")

   varSQL = "SELECT * FROM " & varTableName
   objRecordSet.Open varSql, objConn    ' cursor type and lock type not
                                           supported by Oracle driver
%>

<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=5><FONT FACE="Arial" COLOR=#000000>
<CAPTION><B><%=varTableName%> (in Oracle database)</B></CAPTION></FONT>
<THEAD>
<TR>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecordSet.Fields(varI)
If objField.Name <> varKeyName Then    ' omit surrogate key %>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000 ><FONT SIZE=2 FACE="Arial"
COLOR=000000><%=objField.Name%></FONT></TH>
<%
End If
Next%>
</TR>
</THEAD>
<TBODY>
<%
On Error Resume Next
objRecordSet.MoveFirst
do while Not objRecordSet.eof
%>
<TR VALIGN=TOP>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecorSet.Fields(varI)
If objField.Name <> varKeyName Then    ' omit surrogate key%>
<TD BORDERCOLOR=#c0c0c0 ><FONT SIZE=2 FACE="ARIAL"
COLOR=#000000><%=Server. HTMLEncode(objField.Value)%><BR></FONT></TR>
```

➤ FIGURA 15-18

(Continuación)

```
<%
End If
Next%>
</TR>
<%
objRecordSet.MoveNext
loop%>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

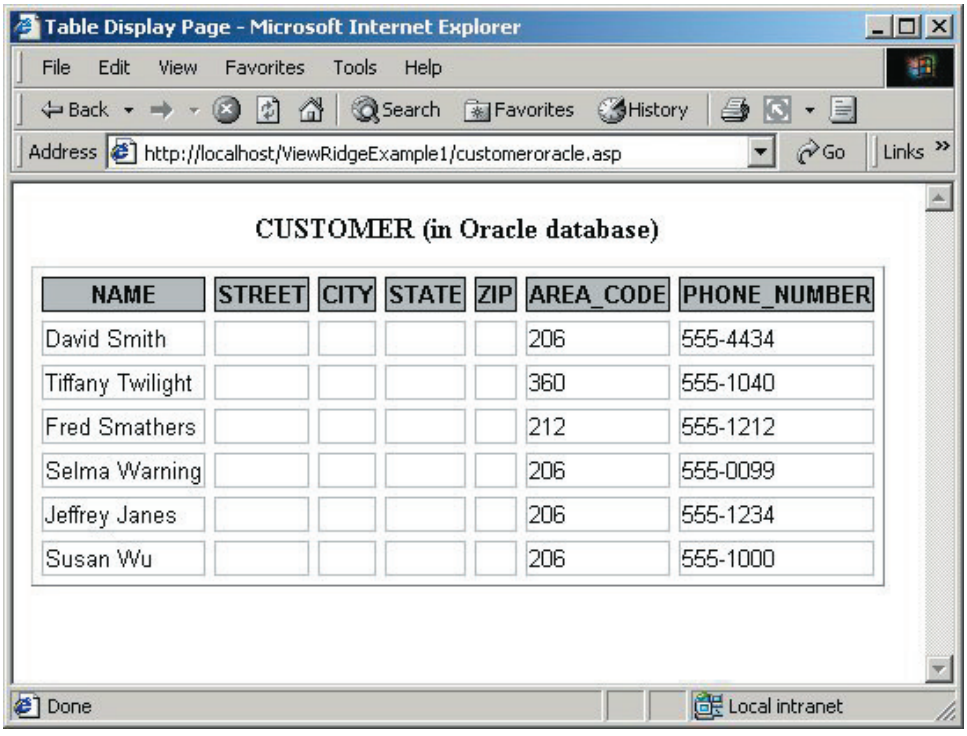
La ventaja de esta página es que puede procesar cualquier tabla, y no sólo una en particular. De hecho, mediante la tecnología anteriormente establecida, se puede decir que la página en la figura 15-18 es una abstracción de la de la figura 15-16. Los resultados de esta página se muestran en la figura 15-19. La columna CUSTOMERID no se despliega, como esperábamos.

EJEMPLO 3—LECTURA DE CUALQUIER TABLA

La figura 15-20(a) muestra una forma de entrada de datos en la cual un cliente puede escribir el nombre de la tabla que se desplegará (un diseño mejor utilizaría una lista desplegable para mostrar las opciones válidas, pero dicho análisis nos desviaría del análisis del ADO). El usuario de esta forma ha escrito *artist*. Ahora suponga que cuando él hace clic en el botón Show Table (Mostrar Tabla), la forma generará el diseño que se ejecutará en el servidor, el cual mostrará la tabla ARTIST en la misma sesión del explorador. Asimismo, suponga que la llave sustituta no se mostrará. Los resultados deseados se muestran en la figura 15-20(b).

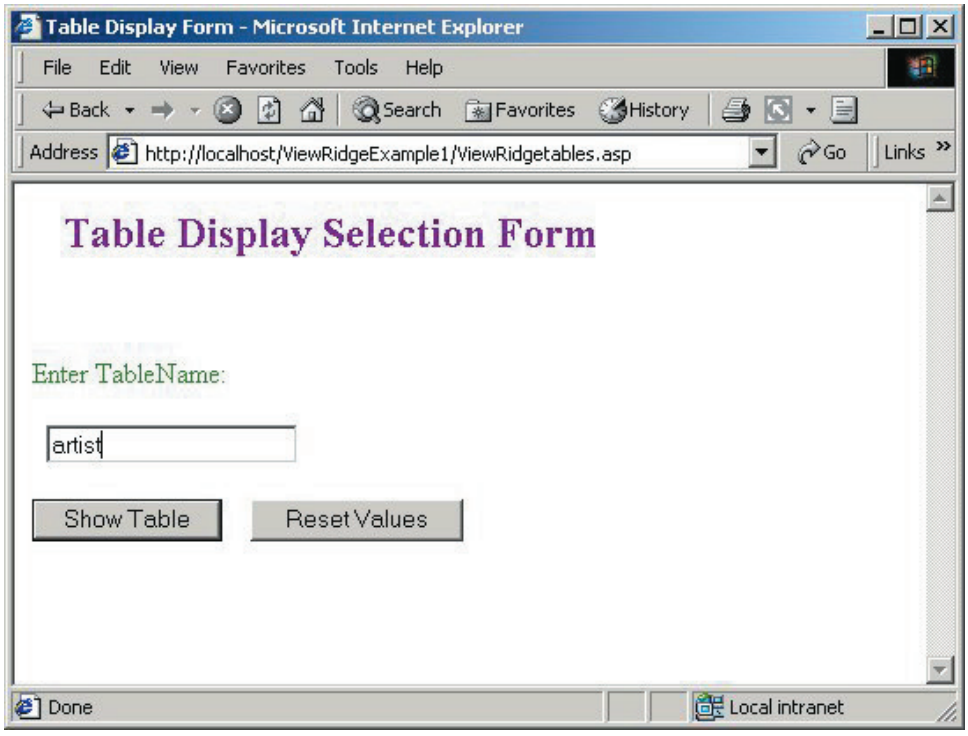
➤ FIGURA 15-19

Resultado de CustomerOracle.asp

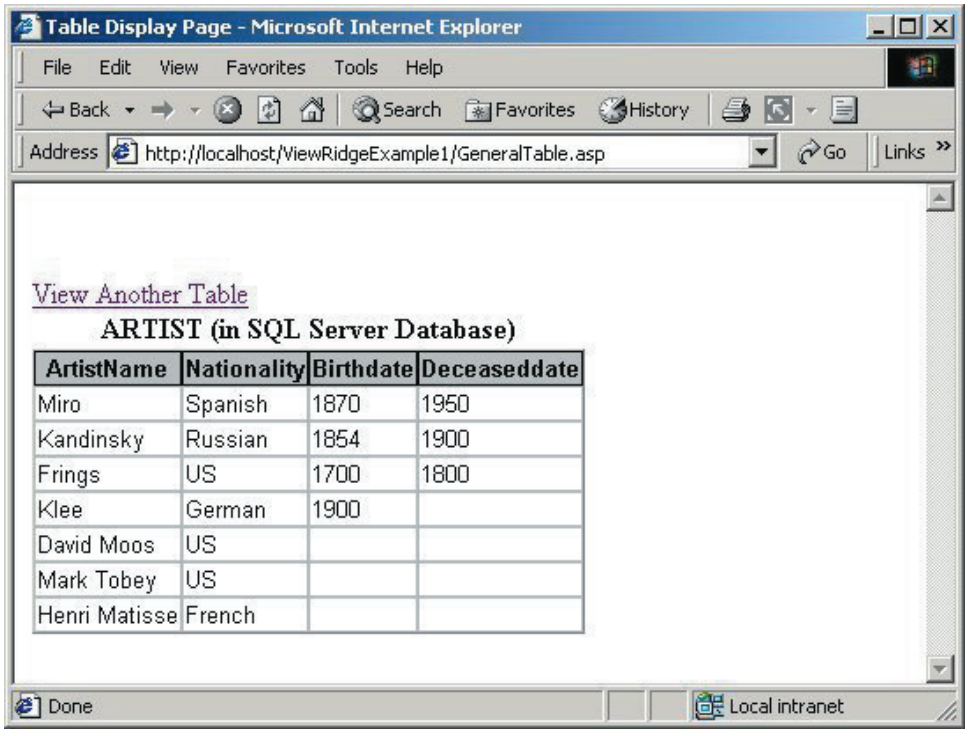


➤ FIGURA 15-20

Formas de despliegue de la tabla general: (a) forma de ingreso del nombre de la tabla, (b) despliegue de la tabla



(a)



(b)

Este proceso necesita dos páginas ASP. La primera, que se muestra en la figura 15-21(a), es una página HTML que contiene la etiqueta FORM

<FORM METHOD = "post"ACTION="GeneralTable.asp">

➤ FIGURA 15-21

Código de despliegue de la tabla general: (a) TablasViewRidge.asp, (b) TablasViewRidge.asp

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Table Display Form</TITLE>
</HEAD>
<BODY>

<FORM METHOD="post" ACTION="GeneralTable.asp">

 <P><STRONG><FONT color=purple face="" size=5>  Table Display
Selection Form</FONT></STRONG>
<P></P>
<P> </P>

<P><FONT style="BACKGROUND-COLOR: #ffffff"><FONT color=forestgreen face=""
style="BACKGROUND-COLOR: #ffffff">Enter
TableName:</FONT>     </FONT></P>

<P></P>

<P><FONT style="BACKGROUND-COLOR: #ffffff"></FONT> 
<INPUT id=text1 name=text1></P>

<P>FONT style="BACKGROUND-COLOR: #ffffff">
<INPUT id=submit1 name=submit1 type=submit value="Show Table"
>   
<INPUT id=reset1 name=reset1 type=reset value="Reset Values"></FONT></P>
</FORM>
</BODY>
</HTML>
```

(a) (*continúa*)

Esta etiqueta define parte de la sección de la página; la sección contendrá los valores de entrada de los datos. En esta forma sólo existe uno: el nombre de la tabla. El método posterior se refiere a un proceso del HTML que genera los datos en la forma (aquí, el nombre de la tabla "artist") que se darán al servidor del ASP en un objeto llamado Form. Un método alternativo es *obtener*, el cual generaría los valores de los datos que se transmitirán mediante parámetros. Esta distinción no es tan importante para nosotros. El segundo parámetro de la etiqueta FORM es ACTION, que es un conjunto de GeneralTable.asp. Este parámetro le informa a IIS cuando recibe la respuesta de esta forma y debe transferir el archivo del ASP GeneralTable.asp al procesador ASP. Los valores de la forma se admitirán en un objeto llamado Form.

El resto de la página es HTML estándar. Observe que el nombre del cuadro de entrada de texto es el text1.

La figura 15-21(b) muestra GeneralTable.asp, la página que se invocará cuando se reciba la respuesta de la página de la forma en la figura 15-21(a). La primera instrucción del script ejecutable es:

```
varTableName = Request.Form("text1")
```

Request.Forma es el nombre del objeto que contiene los valores enviados nuevamente desde el explorador. En este caso, text1 se establecerá como *artist*.

Esta versión de GeneralTable procesa la base de datos de View Ridge de SQL Server. Los nombres de la llave sustituta en SQL Server se encuentran en la forma ArtistID y no en ArtistID. Debido a que las comparaciones del VBSCript son sensibles a mayúsculas y

FIGURA 15-21 *(Continuación)*

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Table Display Page</TITLE>
</HEAD>
<BODY>
<!--#include virtual="ViewRidgeExample1/adovbs.inc"-->
<%

Dim objConn, objRecordSet, objField
Dim varNumCols, varI, varSql
Dim varTableName, varRecordSetName, varKeyName
Dim varTableNameFirst, varTableNameRest

varTablename = Request.Form("text1")

' Set key name to upper first initial and lower remainder with ID, e.g.,
CustomerID
varTableNameFirst = UCase(Left(varTableName, 1))
varTableNameRest = LCase(Right(varTableName, Len(varTableName) -1))
varKeyName = varTableNameFirst & varTableNameRest &"ID"

varRecordSetName = "_rs_" & varTableName ' use for saving recordset object
pointer

If IsObject(Session("_conn")) Then
    Set objConn = Session("_conn")
Else
    Set objConn = Server.CreateObject("ADODB.connection")
    objConn.IsolationLevel = adXactReadCommitted ' avoid dirty reads
    objConn.open "ViewRidgeSS", "sa"
    Set Session("_conn") = objConn
End If

If IsObject(Session(varRecordSetName)) Then
    Set objRecordSet = Session(varRecordSetName) ' used saved recordset object
if possible
    objRecordSet.Requery
Else
    varSql = "SELECT * FROM " & "[" & varTableName & "]" ' put brackets in case
table name has spaces, etc.
    Set objRecordSet = Server.CreateObject("ADODB.Recorset")
    ' in the next statement, note use of cursor and lock types
    objRecordSet.Open varSql, objConn, adOpenDynamic, adLockOptinistic ' allow
for updates
    Set Session(varRecordSetName) = objRecordSet

End If
%>

<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=0><FONT FACE="Arial" COLOR=#000000>
<CAPTION><B><%=UCase(varTableName)%> (in SQL Server
Database)</B></CAPTION></FONT>
<THEAD>
<TR>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecordSet.Fields(varI)
If objField.Name <> varKeyName Then %>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000 ><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=objField.Name%></FONT></TH>
```

(b)

➤ FIGURA 15-21

(Continuación)

```
<%
END IF
Next%>
</TR>
</THEAD>
<TBODY>
<%
On Error Resume Next
objRecordSet.MoveFirst
do while Not ObjRecorSet.eof
%>
<TR VALIGN=TOP>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecordSet.Fields(varI)
If objField.Name <> varKeyName Then %>
<TD BORDERCOLOR=#c0c0c0 ><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=Server.HTMLEncode(objField.Value)%><BR></FONT></TD
<%
End If
Next%>
</TR>

<%
objRecordSet.MoveNext
loop%>
<BR><BR><A HREF="ViewRidgeTables.asp">View Another Table</A>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

(b)

minúsculas, es necesario asegurarse de que la varKeyName tenga el nombre de la llave sustituta en el formato requerido. Para dicho fin, el usuario puede ingresar *ARTIST, artist, Artist, aRtIsT*, por lo que no se puede sólo agregar la ID al nombre de la tabla de entrada. Las tres instrucciones que comienzan con varKeyNameFirst emplean las funciones de letras mayúsculas y letras minúsculas para establecer correctamente la varKeyName.

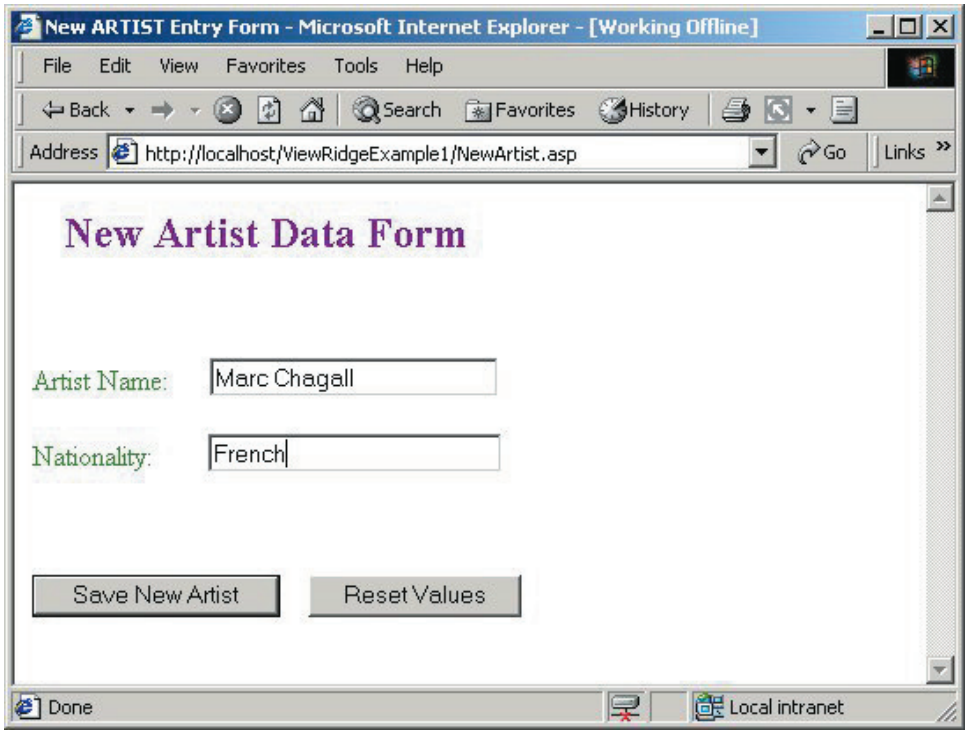
El resto de esta página es el mismo que el de la página Customer.asp, que se muestra en la figura 15-18(b). De nuevo, observe que la variable del nombre de la llave se establecerá como ArtistID, el cual es el nombre de la columna de la llave sustituta que no se desea desplegar.

Ejemplo 4—Actualización de una tabla

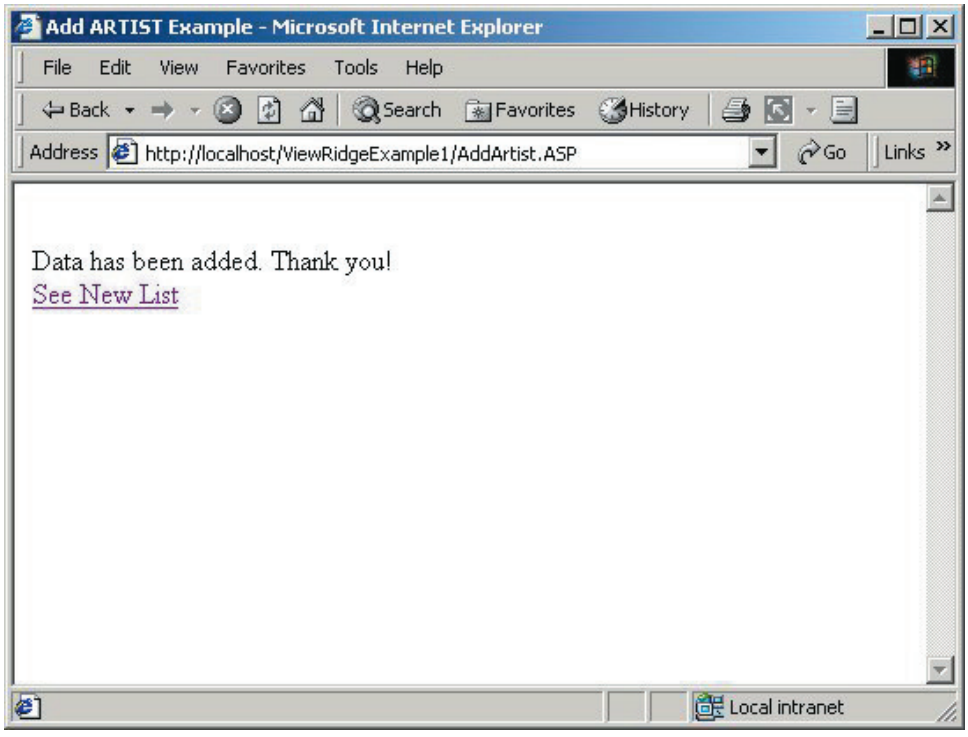
Los tres ejemplos anteriores se refieren a los datos de lectura. El siguiente ejemplo muestra la manera en que se actualizan los datos de la tabla mediante la adición de un renglón con el ADO. La figura 15-22(a) muestra una forma de entrada de datos que capturará el nombre y la nacionalidad del artista y creará un renglón nuevo. Esta forma es similar a la ViewRidgeTables.asp; en lugar de uno tiene dos campos de entrada de datos. Cuando el usuario haga clic en Save New Artist, el artista queda agregado a la base de datos, y si los resultados son exitosos se produce la forma de la figura 15-22(b). El URL See New List invocará la página Artist.asp, que desplegará la tabla ARTIST con el renglón nuevo, como se muestra en la figura 15-22(c).

➤ FIGURA 15-22

Adición de los datos de ARTIST: (a) forma de entrada de los datos de ARTIST, (b) forma de respuesta, y (c) despliegue de la tabla ARTIST



(a)

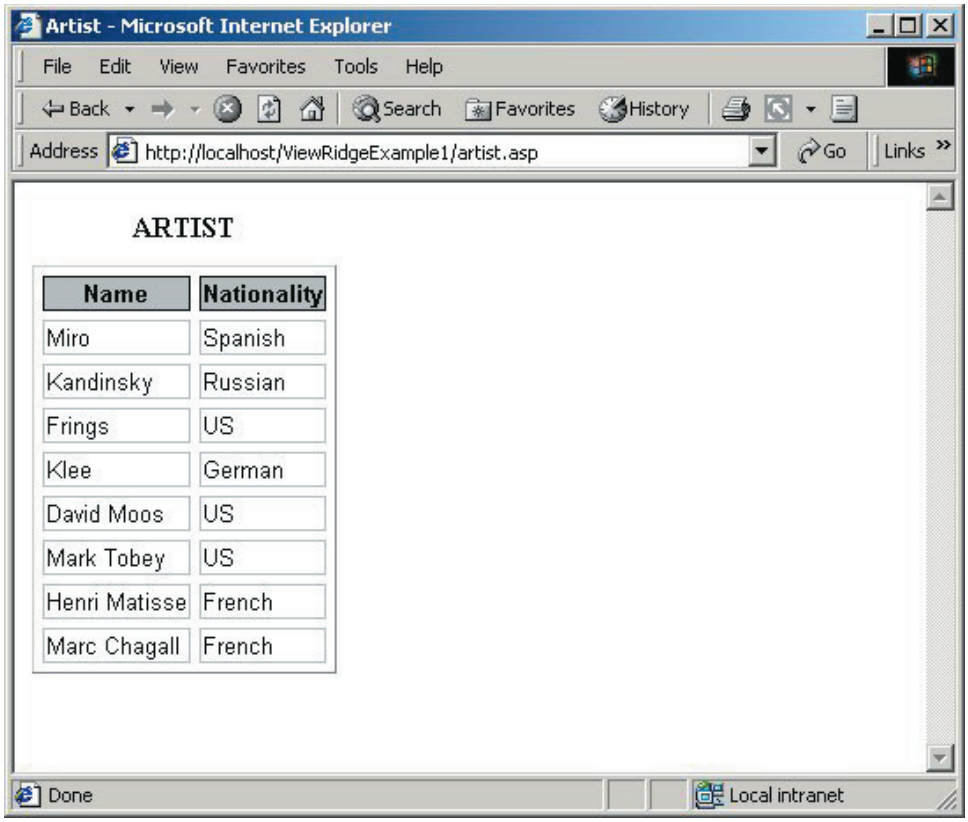


(b)

Las páginas ASP se muestran en la figura 15-23. La primera página es una forma de entrada de datos con dos campos: una para el nombre del artista (llamada *Name*), y la segunda para la nacionalidad de éste (llamada *Nation*). Cuando el usuario haga clic al botón transmitir, estos datos se devolverán al IIS, que a su vez los enviará junto con la página AddArtist.asp al procesador ASP.

➤ FIGURA 15-22

(Continuación)



(c)

Agregarartista.asp (AddArtist.asp), que se muestra en la figura 15-23(b), tiene objetos de conexión y del recordset. Aquí no se realiza ningún intento por guardar los apuntadores de éstos en las variables de la sesión. (Se supone que sólo se agregará un artista por sesión y que guardarlos sería innecesario.) Si se desea, se puede agregar un código, como se mostró en los ejemplos anteriores.

La diferencia clave de esta página se muestra en las instrucciones:

```
objRecordSet.AddNew
objRecordSet (“Name”) = Request.Form(“Name”)
objRecordSet (“Nationality”) = Request.Form(“Nation”)
objRecordSet.Update
```

La primera instrucción obtiene un renglón nuevo en el objeto objRecordSet y entonces se obtienen los valores para las columnas del nombre y la nacionalidad del objeto Request.Form. Observe que no se necesita que los nombres de las columnas y los nombres de Request.Form sean los mismos. Aquí, el nombre de la segunda columna es Nationality, pero el segundo valor de la forma es Nation. La llamada objRecordSet. Update ocasiona que la base de datos se actualice. Observe que el código de procesamiento de errores, que generará mensajes de error, se desplegará mediante la instrucción Responce.Write (es un método disponible en Response de objetos del ASP). La página termina con dos llamadas para enviar al usuario un mensaje de confirmación y crear un URL a la página Artist.asp, si es que el usuario desea ver la tabla con los nuevos valores.

➤ FIGURA 15-23

Inserción de datos mediante el ASP: (a) NewArtist.asp, y (b) AddArtist.asp

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>New ARTIST Entry Form</TITLE>
</HEAD>
<BODY>

<FORM METHOD="post" ACTION="AddArtist.ASP">

 <P><STRONG><FONT color=purple face="" size=5>   New Artist Data
Form</FONT></STRONG>
<P></P>
<P> </P>

<P><FONT style="BACKGROUND-COLOR: #ffffff"><FONT color=forestgreen face=""
style="BACKGROUND-COLOR: #ffffff">Artist
Name:</FONT>     
<INPUT id=text1 name=Name style="HEIGHT: 22px; WIDTH: 164px"></FONT></P>

<P><FONT color=forestgreen face="""
style="BACKGROUND-COLOR:
#ffffff">Nationality:              
<INPUT id=text2 name=Nation style="HEIGHT: 22px; WIDTH: 167"></FONT></P>

<P> </P>

<P><FONT style="BACKGROUND-COLOR: #ffffff">
<INPUT id=submit1 name=submit1 type=submit value="Save New
Artist">   
<INPUT id=reset1 name=reset1 type=reset value="Reset Values"></FONT></P>
</FORM>
</BODY>
</HTML>
```

(a)

EJEMPLO 5—INVOCACIÓN DE UN PROCEDIMIENTO ALMACENADO

Creamos un procedimiento almacenado llamado Cliente_Insertar (Customer_Insert) en las bases de datos de Oracle y de SQL Server (en los capítulos 12 y 13, respectivamente). En ambos casos, el procedimiento almacenado acepta un nombre nuevo del cliente, código de área, número local y la nacionalidad de todos los artistas en los que esté interesado el cliente. Entonces, se crea un nuevo renglón en CLIENTE (CUSTOMER) y se agregan los renglones apropiados a la tabla de intersección.

Para invocar el procedimiento almacenado mediante una página ASP, se crea una página de la red para reunir todos los datos necesarios, como vemos en la figura 15-24(a). Ahora, cuando el usuario haga clic en Add Customer (Agregar Cliente), deseamos invocar una página ASP que llame al procedimiento almacenado con los datos de la forma. De esta manera el usuario puede verificar que los datos nuevos se hayan ingresado correctamente; después la página ASP consultará una vista que una a los nombres de los clientes con los nombres y nacionalidades de los artistas. El resultado se muestra en la figura 15-24(b).

La figura 15-25(a) muestra el código para generar la forma de reunir los datos. Los campos del parámetro se llaman *text1* a *text4*. La forma invoca la página asp CustomerInsertOracle en la instrucción del FORM METHOD, por lo que cuando el usuario hace clic en AddCustomer, se enviarán los datos a CustomerInsertOracle; como la página que se muestra en la figura 15-25(b).

➤ FIGURA 15-23

(Continuación)

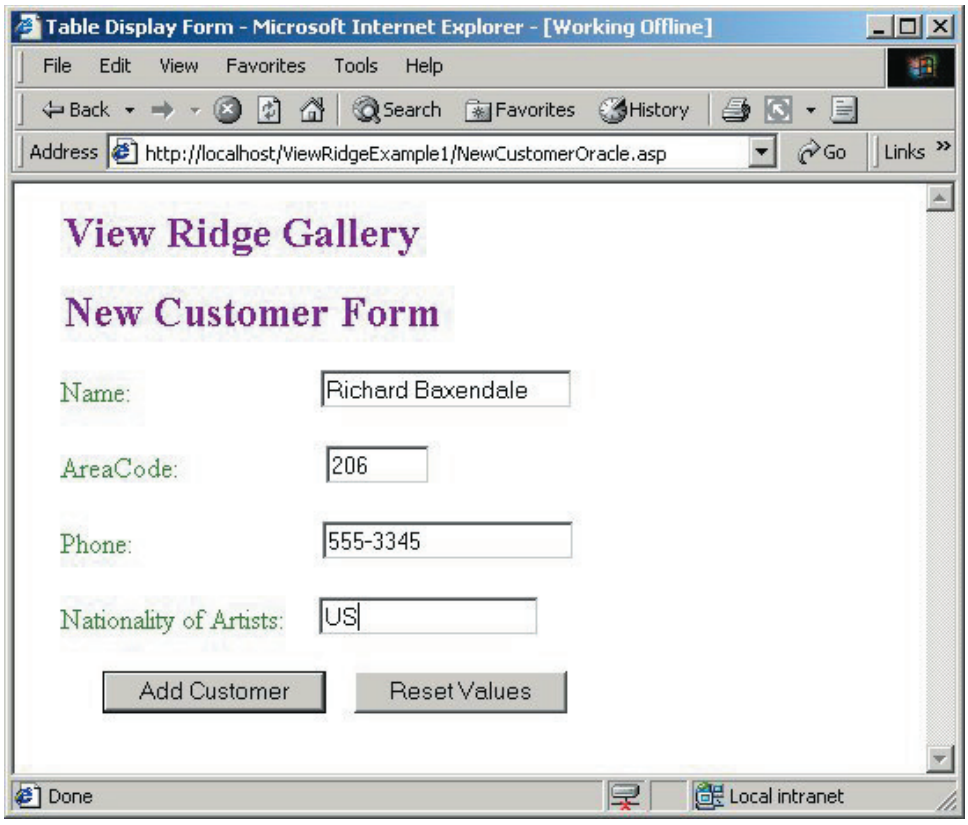
```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Add ARTIST Example</TITLE>
</HEAD>
<BODY>
<!--#include virtual="ViewRidgeExample1/adovbs.inc"-->
<%

Dim objConn, objRecordSet, objField
Dim varNumCols, varI, varSql

Set objConn = Server.CreateObject("ADODB.connection")
objConn.open "ViewRidgeSS", "sa"  ' open with sa
objConn.IsolationLevel = adXactReadCommitted ' avoid dirty reads

varSql = "SELECT * FROM [ARTIST]"
Set objRecordSet = Server.CreateObject("ADODB.Recordset")
' in the next statement, note use of cursor and lock types
objRecordSet.Open varSql, objConn, adOpenDynamic, adLockOptimistic

objRecordSet.AddNew
objRecordSet("ArtistName")= Request.Form("Name")
objRecordSet("Nationality")= Request.Form("Nation")
objRecordSet.Update

On Error Resume Next
varErrorCount = objConn.Errors.Count
If varErrorCount > 0 Then
   For varI = 0 to varErrorCount - 1
      Response.Write "<BR><I>" & objConn.Errors(varI).Description & "</I><BR>"
   Next
End If

objRecordSet.Close
objConn.Close

Response.Write "<BR>Data has been added.  Thank you!<BR>"
Response.Write "<A HREF="& """ &"artist.asp" & """& ">See New List</A>"

%>
<BR><BR>
</BODY>
</HTML>
```

(b)

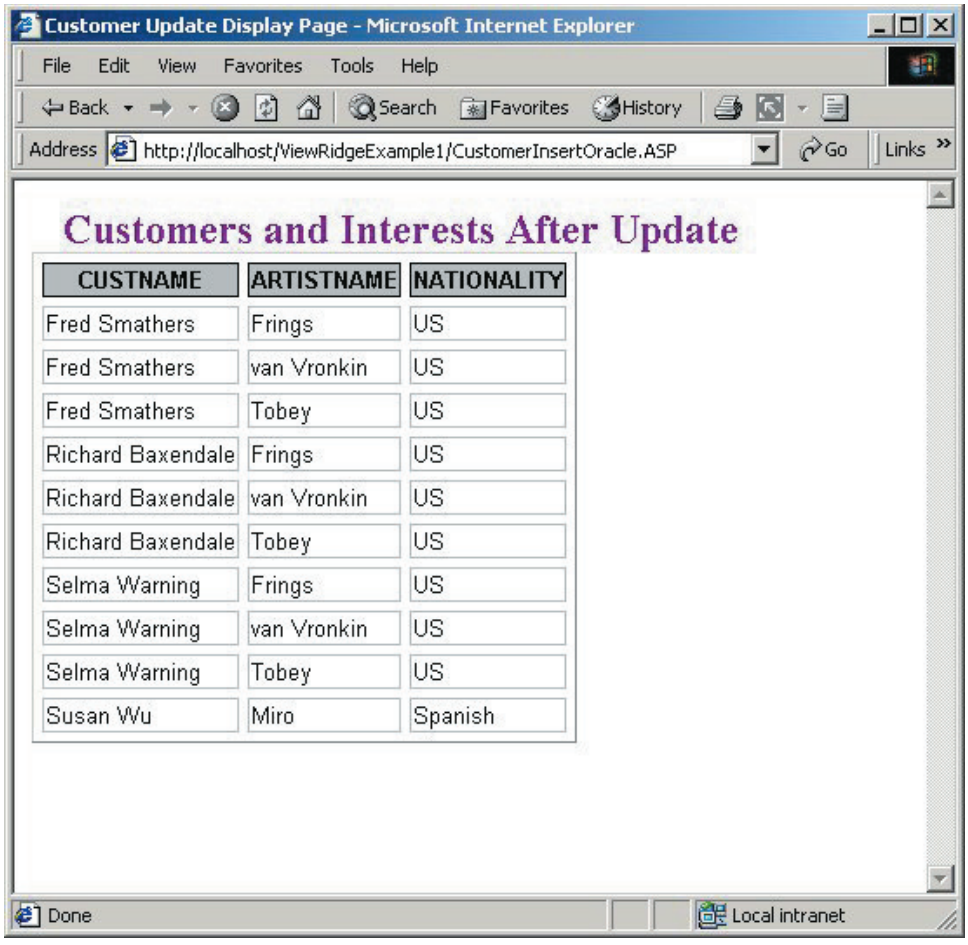
Esta página busca una conexión guardada y, si no la encuentra, obtiene una nueva, como mostramos anteriormente. Luego crea un objeto de orden, *objCommand* y lo asocia con el objeto de conexión. Después establece la llamada al procedimiento almacenado Customer_Insertar con el CommandText="{callCustomer_Insert(?,?,?,?)}." Este patrón indica que se admitirán los cuatro parámetros. Las siguientes secciones del código crean los parámetros y los agregan al objeto de comando. Por último, la instrucción se ejecuta, lo que provoca que se invoque el procedimiento almacenado. No se establece ningún nivel de aislamiento de la transacción, ni ninguna propiedad del cursor debido a que el procedimiento almacenado los establecerá por sí mismo.

➤ FIGURA 15-24

(a) Forma de entrada de los datos de CUSTOMER (CLIENTE), (b) forma que muestra el join de los intereses del cliente y del artista



(a)



(b)

Después de que se ejecuta la instrucción se crea un recordset en una selección de todas las columnas de la vista CUSTOMERINTERESTS y, por último, se muestran las columnas en el explorador mediante las técnicas que se incluyeron en los ejemplos anteriores.

En las bases de datos Oracle y de SQL Server se definió esta vista como el join de CUSTOMER y ARTIST sobre la tabla de intersección. La sintaxis empleada por Oracle fue

```
CREATE VIEW CUSTOMERINTERESTS AS
    SELECT CUSTOMER.NAME CUSTNAME, ARTIST.NAME
        ARTISTNAME, ARTIST.NATIONALITY
    FROM CUSTOMER.CUSTOMER_ARTIST_INT, ARTIST
    WHERE CUSTOMER.CUSTOMERID = CUSTOMER_ARTIST_
        INT.CUSTOMERID AND
        ARTIST.ARTISTID = CUSTOMER_ARTIST_INT.ARTISTID
    ORDER BY CUSTNAME
```

Algo muy interesante de esta página es que la única diferencia entre la versión de Oracle que se muestra aquí, y la versión de SQL Server es el nombre de la fuente de datos del ODBC, el nombre de la cuenta y la contraseña utilizados. Observe la línea de anotación debajo de la instrucción de objConn.open; todas las idiosincrasias de los productos DBMS están contenidas en los procedimientos almacenados, y el programador de la página ASP no necesita saber nada sobre ellas.

Estos ejemplos le dan una idea sobre el uso del ADO. La mejor manera de que aprenda más es que escriba algunas páginas por su propia cuenta. Este capítulo ha mostrado todas las técnicas básicas que necesitará. Se ha esforzado bastante para llegar a este punto y si puede entender lo suficiente como para crear algunas de sus propias páginas, ¡entonces verdaderamente habrá avanzado mucho desde el capítulo 1!

➤ FIGURA 15-25

Invocación de un procedimiento almacenado del ASP: (a) NewCustomerOracle.asp, (b) NewCustomerOracle.asp

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Table Display Form</TITLE>
</HEAD>
<BODY>

<FORM METHOD="post" ACTION="CustomerInsertOracle.ASP">

 <P><STRONG><FONT color=purple face="" size=5>   View Ridge
Gallery</FONT></STRONG>

 <P>   <strong><font color="purple" face size="5"> New Customer
Form</font></strong><P><font style="background-color: #ffffff"
color="forestgreen" face>   
 Name:   </font><FONT style="BACKGROUND-COLOR:
#ffffff">           &nbs
p;           
 </FONT><INPUT id=text1 name=text1></P>

 <P> <font style="background-color: #ffffff" color="forestgreen"
face>   AreaCode<font style="background-color:
#ffffff">:   </font>       &n
bsp;         
 </font><INPUT id=text2 name=text2 size="6"></P>

 <P><font style="background-color: #ffffff" color="forestgreen"
face>   

Phone:            &
nbsp;            &n
bsp;
 </font><INPUT id=text3 name=text3
size="20">        </P>

 <P>    <font style="background-color: #ffffff"
color="forestgreen" face>Nationality
 of Artists:  </font>   <INPUT id=text4 name=text4
size="17"></P>

 <P> <FONT style="BACKGROUND-COLOR: #ffffff">
   </FONT>     <FONT style="BACKGROUND-
COLOR: #ffffff">
<INPUT id=submit1 name=submit1 type=submit value="Add Customer"
>   
<INPUT id=reset1 name=reset1 type=reset value="Reset Values"></FONT></P>

</FORM>
</BODY>
</HTML>
```

(a)

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;charset=windows-1252">
<TITLE>Customer Update Display Page</TITLE>

</HEAD>
<BODY>

<P><STRONG><FONT color=purple face="" size5>   Customers and Interests After
Update</FONT></STRONG>
<!--#include virtual="ViewRidgeExample1/adovbs.inc"-->
<%

    Dim objConn, objCommand, objParam, oRs
    Dim objRecordSet, objField
    Dim varI, varSql, varNumCols, varValue

    If IsObject(Session("_conn")) Then
        Set objConn = Session("_conn") ' use current session if available
    Else
        Set objConn = Server.CreateObject("ADODB.connection")
        ' stored procedure will set its own isolation level
        objConn.open "ViewRidgeOracle", "system", "manager" ' use this to update via Oracle
        'objConn.open "ViewRidgeSS", "sa"  ' could use this to update via SQL Server
        Set Session("_conn") = objConn
    End If

    Set objCommand = Server.CreateObject("ADODB.Command") ' create a command object
    Set objCommand.ActiveConnection = objConn ' set the command objects connection

    objCommand.CommandText="{call Customer_Insert (?, ?, ?, ?)}" ' setup call to stored procedure

    ' Set up four parameters with necessary values
    Set objParam = objCommand.CreateParameter("NewName", adChar, adParamInput, 50)
    objCommand.Parameters.Append objParam
    objParam.Value = Request.Form("text1")

    Set objParam = objCommand.CreateParameter("AreaCode", adChar, adParamInput, 5)
    objCommand.Parameters.Append objParam
    objParam.Value = Request.Form("text2")

    Set objParam = objCommand.CreateParameter("PhoneNumber", adChar, adParamInput, 8)
    objCommand.Parameters.Append objParam
    objParam.Value = Request.Form("text3")

    Set objParam = objCommand.CreateParameter("Nationality", adChar, adParamInput, 25)
    objCommand.Parameters.Append objParam
    objParam.Value = Request.Form("text4")

' Fire the Stored Proc

    Set oRs = objCommand.Execute

    ' now read the data from a view having both CUSTOMER and ARTIST
    varSql = "SELECT * FROM CUSTOMERINTERESTS" ' use that joins via the intersection table
    Set objRecordSet = Server.CreateObject("ADODB.Recordset")
    objRecordSet.Open varSql, objConn
```

(b)

(*continúa*)

➤ FIGURA 15-25

(Continuación)

```
%>
<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=5><FONT FACE="Arial"
COLOR=#000000><CAPTION><B><CUSTOMERS AND INTERESTS</B></CAPTION></FONT>
<THEAD>
<TR>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecordSet.Fields(varI)
%>
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=objField.Name%></FONT></TH>

<%
Next%>
</TR>
</THEAD>
<TBODY>
<%
On Error Resume Next
objRecordSet.MoveFirst
do while Not objRecordSet.eof
%>
<TR VALIGN=TOP>
<%
varNumCols = objRecordSet.Fields.Count
For varI = 0 to varNumCols - 1
Set objField = objRecordSet.Fields(varI)
If objRecordSet.Fields(varI).Type = adNumeric then
      varValue=CDbl(objField.Value)
      varValue = convert(char, varValue)
else
      varValue=Server.HTMLEncode(objField.Value)
End If
%>
<TD BORDERCOLOR=#c0c0c0><FONT SIZE=2 FACE="Arial"
COLOR=#000000><%=(varValue)%><BR></FONT></TD>
<%
varValue=""
Next%>
</TR>

<%
objRecordSet.MoveNext
loop%>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

(b)

➤ RESUMEN

Las aplicaciones de la tecnología Internet residen en un ambiente rico y complejo. Además de las bases de datos relacionales, hay bases de datos no relacionales, VSAM y otros procesadores de datos, correo electrónico y otros tipos de datos como imágenes, sonido, etc. Para facilitar el trabajo del programador de aplicaciones se han desarrollado diversos estándares. El estándar ODBC es para las bases de datos relacionales; el OLE DB es para las bases de datos relacionales y otras. ADO se desarrolló para proporcionar un acceso más fácil a los datos de OLE DB para el programador orientado a no objetos.

ODBC, o el estándar de la conectividad abierta de una base de datos, proporciona una interfaz por medio de la cual los programas del servidor de la red pueden ingresar y procesar fuentes de datos relacionales de una manera independiente del DBMS. El ODBC fue desarrollado por un comité industrial y ha sido implementado por Microsoft y muchos otros proveedores. ODBC involucra programas de aplicación, al controlador de drivers, al driver del DBMS y a los componentes de la fuente de datos. Los drivers de una capa y multicapas, o de capas múltiples, están definidos. Existen tres tipos de nombres de fuentes de datos: archivo, sistema, y usuario. Se recomiendan las fuentes de datos de sistema para los servidores de la red. El proceso de definición de una fuente de datos de un sistema implica la especificación del tipo de driver y la identidad de la base de datos que se procesará.

OLE DB es el cimiento del mundo de acceso de datos de Microsoft. Implementa los estándares OLE y COM de Microsoft y es accesible para los programas orientados a objetos mediante dichas interfaces. El OLE DB fragmenta las características y funciones de un DBMS en objetos, por consiguiente hace más fácil que los vendedores implementen partes de la funcionalidad. Los términos clave del objeto son: abstracción, métodos, propiedades y colecciones. Un rowset es una abstracción de un recordset, el cual, en cambio, es una abstracción de una relación. Los objetos se definen por las propiedades que especifican sus características y métodos, los cuales son acciones que pueden realizar. Una colección es un objeto que contiene un grupo de otros objetos. Los propósitos del OLE DB se listan en la figura 15-10. Una interfaz es un conjunto de objetos y las propiedades y métodos que exponen en dicha interfaz. Los objetos pueden exponer diferentes propiedades y métodos en diferentes interfaces. Una implementación es la manera en que un objeto lleva a cabo sus tareas. Las implementaciones se esconden del mundo exterior y pueden cambiarse sin afectar a los usuarios de los objetos. Nunca debe cambiarse una interfaz.

Los proveedores de datos tabulares presentan los datos en forma de rowsets. Los proveedores de servicios transforman los datos en otra forma; dichos proveedores son consumidores y proveedores de datos. Un rowset es equivalente a un cursor. Las interfaces básicas del rowset son ConjuntoderenglonesI (IRowSet), IngresoI (IAccessor), e InformacióndecolumnasI (IColumnsInfo). Otras interfaces están definidas para una capacidad más avanzada.

ADO es un modelo de objetos sencillo que utilizan los consumidores de datos de OLE DB. Se puede utilizar desde cualquier lenguaje que maneje Microsoft. El modelo del objeto de ADO tiene objetos de Conexión, ConjuntodeRegistros, Instrucción y colección de Errores. Los recordsets tienen una colección de campos y órdenes, y éstas tienen una colección de parámetros.

Un objeto de conexión establece una conexión al proveedor de datos y a la fuente de datos. Las conexiones tienen un modo de aislamiento. Una vez que se crea una, se pueden utilizar los objetos del recordset y de órdenes. Los objetos del recordset representan cursores, tienen las propiedades Tipo de cursor y Tipo de bloqueo. Los RecordSet pueden crearse con las instrucciones del SQL. La colección de Campos de un RecordSet se puede procesar para manipular individualmente los campos.

La colección Errores contiene uno o más mensajes de error que resultan de una operación de ADO. El objeto comando se emplea para ejecutar consultas almacenadas de parámetros o procedimientos almacenados. Los datos de entrada se pueden enviar a la página ASP correcta mediante la etiqueta FORMA HTML. Las actualizaciones de la tabla se realizan a través del método de Actualización del RecordSet.

➤ PREGUNTAS DEL GRUPO I

15.1 Describa por qué el ambiente de datos es complicado para los servidores de la red.

15.2 Explique la relación de ODBC, OLE DB y ADO.

15.3 Explique la justificación del autor para describir los estándares de Microsoft. ¿Está usted de acuerdo?

15.4 Nombre los componentes del estándar ODBC.

15.5 ¿Qué función desempeña el controlador de drivers? ¿Quién lo proporciona?

15.6 ¿Qué papel desempeña el controlador del DBMS? ¿Quién lo proporciona?

15.7 ¿Qué es un driver de una capa?

15.8 ¿Qué es un driver multicapas?

15.9 ¿Los usos del término *capa* en la arquitectura de tres capas y su uso en el ODBC se relacionan uno con el otro?

15.10 ¿Por qué son importantes los niveles de conformidad?

15.11 Resuma los tres niveles de conformidad del API del ODBC.

15.12 Resuma los tres niveles de conformidad de la gramática del SQL.

15.13 Explique la diferencia entre los tres tipos de fuentes de datos.

15.14 ¿Qué tipo de fuente de datos se recomienda para los servidores de la red?

15.15 ¿Cuáles son las dos tareas que se llevan a cabo cuando se establece una fuente de datos ODBC?

15.16 ¿Por qué es importante el OLE DB?

15.17 ¿Qué desventaja del ODBC puede superar el OLE DB?

15.18 Defina *abstracción* y explique cómo se relaciona con el OLE DB.

15.19 Dé un ejemplo sobre una abstracción que implique el rowset.

15.20 Defina *propiedades* y *métodos* del objeto.

15.21 ¿Cuál es la diferencia entre la clase de un objeto y un objeto?

15.22 Explique el papel de los consumidores de datos y de los proveedores de datos.

15.23 ¿Qué es una interfaz?

15.24 ¿Cuál es la diferencia entre una interfaz y una implementación?

15.25 Explique por qué se puede cambiar una implementación, pero no se debe cambiar una interfaz.

15.26 Resuma los objetivos del OLE DB.

15.27 ¿Qué es el MTS y qué hace?

15.28 Explique la diferencia entre un proveedor de datos tabulares y un proveedor de servicios.

15.29 En el contexto del OLE DB, ¿cuál es la diferencia entre un rowset y un cursor?

15.30 ¿Qué lenguajes puede utilizar ADO?

15.31 Liste los objetos en el modelo del objeto de ADO y explique sus relaciones.

15.32 ¿Cuál es la función del objeto de conexión?

15.33 Muestre una pequeña parte de VBScript para la creación de un objeto de conexión.

15.34 ¿Cuál es la función del objeto del RecordSet?

15.35 Muestre una pequeña parte del VBScript para la creación de un objeto del RecordSet.

15.36 ¿Qué contiene la colección de Campos? Explique una situación en la cual la utilizaría.

15.37 Muestre una pequeña parte de VBScript para el procesamiento de una colección de Campos.

15.38 ¿Qué contiene la colección Errores? Explique una situación en la cual la utilizaría.

15.39 Muestre una pequeña parte de VBScript para el procesamiento de una colección de Errores.

15.40 ¿Cuál es el propósito del objeto Comando?

15.41 Muestre una pequeña parte de VBScript para la ejecución de una consulta de parámetros almacenada que tenga dos parámetros: A y B.

15.42 Explique el propósito de las etiquetas <% y %> en las páginas ASP.

15.43 Explique el propósito de la variable de conexión en la figura 15-16.

15.44 ¿Cuál es la razón del código que crea varKeyName (variable del nombre clave) en la figura 15-21(b)?

15.45 Explique el propósito del parámetro ACTION en la etiqueta FORM en la figura 15-21(a).

15.46 Explique qué pasa cuando se ejecuta la siguiente instrucción en la página ASP en la figura 15-21(b).

```
varTableName = Request.Form (“text1”)
```

15.47 Muestre una pequeña parte del VBScript para agregar un registro nuevo a un nombre de un recordset objetodeMiConjuntoderegistros. Suponga que los campos son A y B y sus valores serán “valorA” y “valorB”, respectivamente.

15.48 ¿Qué propósito cumple la instrucción Respuesta.Escribir (Response.Write)?

➤ PREGUNTAS DEL GRUPO II

15.49 Microsoft hizo muchos esfuerzos para promocionar los estándares de OLE DB y ADO. Directamente no recibe los ingresos de estos estándares. El IIS es gratis con Windows NT y Windows 2000. Su sitio en Internet tiene numerosos ejemplos de los elementos para ayudar a los programadores a que aprendan más, sin costo alguno. ¿Por qué cree que Microsoft hace esto? ¿Qué objetivo se cumple?

15.50 En el código en la figura 15-23(b) el tipo de cursor se establece como dinámico. ¿Qué efecto tendrá esto en el procesamiento y en las páginas Cliente.asp y Artista.asp? Explique cómo cree que deberían establecerse los parámetros de los niveles de aislamiento, tipo de cursor y tipo de bloqueo para una aplicación que implique estas tres páginas.

15.51 Explique cómo cambiar el ejemplo de la página ASP en la figura 15-16 para ejecutarlo con el DSN ViewRidgeOracle. Explique cómo cambiar la página ASP que se muestra en la figura 15-18 para que corra con ViewRidgeSS. La facilidad para realizar estos cambios resulta interesante desde un punto de vista tecnológico, ¿esta facilidad tiene alguna importancia en el mundo del comercio?

15.52 Si ya ha instalado Oracle, utilice su explorador para ejecutar la página en la figura 15-18. Ahora abra el SQL Plus y elimine dos renglones de los datos de CLIENTE mediante la orden SQL DELETE. Regrese a su explorador y ejecute de nuevo la página en la figura 15-18. Explique los resultados.

15.53 Si ya ha instalado SQL Server, utilice su explorador para ejecutar la página en la figura 15-16. Ahora abra el Analizador de consultas del SQL y elimine dos renglones de los datos de CLIENTE mediante la instrucción SQL DELETE. Regrese a su explorador y ejecute de nuevo la página en la figura 15-16. Explique los resultados. Si contestó la pregunta 15.52, mencione la diferencia de los resultados que obtuvo, si es que hubo alguna.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Si aún no lo ha hecho, cree su base de datos FiredUp con Oracle o con el SQL Server. Siga las instrucciones que están al final de los capítulos 12 o 13.

A. Codifique una página ASP para desplegar la tabla ESTUFA.

B. Codifique una página ASP para desplegar cualquier tabla de la base de datos de FiredUp. Utilice la figura 15-21 como ejemplo.

C. Codifique una página ASP para ingresar nuevos datos de la tabla ESTUFA. Justifique su elección de aislamiento de cursor.

D. Codifique una página ASP para permitir que los clientes registren sus propias estufas. Justifique su elección de aislamiento de cursor.

E. Cree un procedimiento almacenado para ingresar nuevos datos de la reparación de estufas.

F. Codifique una página ASP para invocar el procedimiento almacenado creado en la actividad E. Utilice la figura 15-25 como ejemplo.

CAPÍTULO 16

JDBC, Páginas del Servidor Java y MySQL

Este capítulo analiza las alternativas de Microsoft OLE DB, ADO, y la tecnología IIS y productos relacionados. En particular, analizaremos JDBC, Páginas del Servidor Java (JSP) utilizando Apache/Tomcat y el producto DBMS MySQL. El movimiento open source (fuente abierta) ha desempeñado una larga función en el desarrollo de estas tecnologías, y todos son productos open source. En realidad, sólo el software de la open source se utilizó para desarrollar todos los ejemplos de este capítulo.

Sin embargo, open source no es un requisito para usar JDBC. Puede emplear JDBC en Windows 2000 y otros sistemas operativos para tener acceso a SQL Server, Oracle o a otros sobresalientes productos DBMS. También puede ejecutar páginas JSP y Apache/Tomcat en Windows 2000. Sin embargo, en este capítulo todos los ejemplos se desarrollaron y ejecutaron en Linux.

Como podrá adivinar, el único requisito para utilizar JDBC es que los programas estén escritos en Java. Debido a que este texto no supone que usted sea un programador en Java, explicaremos los ejemplos a un alto nivel. No es importante que comprenda cada línea de código. Su meta será entender la naturaleza y la capacidad de las tecnologías que aquí se presentan. Si ya programa en Java, estos ejemplos estimularán su pensamiento para comprender otros más complejos y reales. De cualquier manera, después de leer este capítulo será capaz de comparar las capacidades de ODBC, ADO, y ASP para JDBC y JSP.

➤ JDBC

Para comenzar, contrario a muchas opiniones, JDBC *no* representa la Conectividad de la Base de Datos Java. De acuerdo con Sun, el Inventor de Java y fuente de muchos productos orientados a Java, JDBC no es un acrónimo, sólo representa a JDBC. Uno no puede más que imaginarse los pleitos legales o de ego que están detrás de la afirmación, pero JDBC es eso.

Hay drivers JDBC para casi cualquier producto DBMS concebible. Sun mantiene un directorio de ellos en http://java.sun.com/products/jdbc. Algunos drivers son gratis

y casi todos tienen una edición de evaluación que se puede utilizar gratis durante cierto tiempo. Los drivers JDBC utilizados para la preparación de este capítulo son drivers abiertos de MySQL desarrolladas por Mark Mathews. Se pueden bajar de http://world-server.com/mm.mysql.

Tipos de drivers

Sun define cuatro tipos de drivers. El tipo uno son drivers puente JDBC-ODBC. Éstos proporcionan una interfaz entre Java y drivers regulares de ODBC. La mayoría de los drivers ODBC están escritos en C o C++. Por razones que no son importantes para nosotros, hay incompatibilidades entre Java y C/C++. Los drivers puente resuelven estas incompatibilidades y permiten acceder a las fuentes de datos ODBC de Java. Debido a que describimos el uso de ODBC en el último capítulo, no consideraremos a los drivers puente sino hasta más adelante.

Los drivers del tipo dos al cuatro están escritos totalmente en Java; difieren únicamente en cómo se conectan al DBMS. Los drivers del tipo dos se conectan a la API-original del DBMS; llaman a Oracle, por ejemplo, utilizando el estándar (no-ODBC) de interfaces de programación a Oracle. Los drivers de los tipos tres y cuatro se propusieron para el uso en las redes de comunicaciones. Un driver del tipo tres traslada llamadas JDBC a un protocolo de red independiente del DBMS. Entonces, dicho protocolo se traslada a uno de red que usa un DBMS particular. Por último, los drivers del tipo cuatro trasladan las llamadas JDBC a protocolos DBMS de red específica.

Para comprender las diferencias entre los drivers del tipo dos al cuatro, primero debe comprender la diferencia entre un **servlet** y un **applet.** Como probablemente ya lo sabe, Java fue diseñado para ser portátil. Con el fin de lograr ese objetivo, los programas de Java no están compilados en un lenguaje de máquina en particular, sino en un código de bytes de una máquina independiente. Sun, Microsoft y otros han escrito **los interpretadores de bytecode** para el medio ambiente de cada máquina (Intel 386, Alpha, etc.). Estos interpretadores se llaman **máquinas virtuales Java.**

Para ejecutar un programa compilado de Java, el bytecode independiente de la computadora se interpreta mediante la máquina virtual en el tiempo de ejecución. El costo de esto, por supuesto, es que la interpretación del bytecode constituye un paso extra y, en consecuencia, estos programas nunca pueden ser tan rápidos como los que están directamente compilados en código de la máquina. Esto podría ser un problema, dependiendo de la carga de trabajo de la aplicación.

Un applet es un programa de bytecode Java que se ejecuta en la computadora de la aplicación del usuario. El bytecode applet se transmite al usuario mediante HTTP y se invoca utilizando el protocolo HTTP en la computadora de los usuarios. El bytecode lo interpreta una máquina virtual, que por lo general parte del explorador. Debido a su portabilidad, el mismo bytecode puede ser enviado a una computadora Windows, Unix o Apple.

Un servlet es un programa en Java que se invoca mediante HTTP en la computadora del servidor de la red. Responde a las peticiones de los usuarios del explorador. Los servlets se interpretan y ejecutan mediante una máquina virtual Java corriendo en el servidor.

Debido a que tienen una conexión para un protocolo de comunicaciones, los drivers del tipo 3 y 4 se pueden utilizar en código servlet o applet. Las drivers del tipo 2 se pueden utilizar únicamente en situaciones en que el programa Java y el DBMS residen en la misma máquina, o cuando el driver del tipo 2 se conecta a un programa DBMS que maneja las comunicaciones entre la computadora que ejecuta el programa Java y la que ejecuta el DBMS.

Si escribe el código que conecta a una base de datos de un applet (dos-capas), entonces sólo podrá usar un driver tipo 3 o 4. En estas situaciones, si su producto DBMS tiene un driver del tipo 4 úselo, porque será más veloz que un driver del tipo 3.

En la arquitectura de capas o de *n*-capas, si el servidor de la red y el DBMS se ejecutan en la misma máquina, pueden utilizar un driver de cualquiera de los cuatro tipos. Si el servidor de la red y el DBMS se ejecutan en diferentes máquinas, entonces los drivers

➤ FIGURA 16-1

Resumen de los tipos de drivers JDBC

Tipo de driver	Características
1	Puente JDBC-ODBC. Proporciona un Java API que hace Interfaz con un driver ODBC. Habilita el procesamiento de las fuentes de datos ODBC desde Java.
2	Un Java API que conecta a la biblioteca propia de un producto DBMS. El programa en Java y el DBMS deben residir en la misma máquina, o el DBMS debe manejar la comunicación entre las máquinas, si no es así.
3	Un JAVA API que conecta un protocolo de red independiente de DBMS. Se puede utilizar para servlets y applets.
4	Un JAVA API que conecta un protocolo de red dependiente de DBMS. Puede ser utilizado para servlets y applets.

del tipo 3 y 4 se pueden utilizar sin problema. Los drivers del tipo 2 también se pueden utilizar si el vendedor del DBMS maneja las comunicaciones entre el servidor de la red y el DBMS. Las características de los tipos de driver JDBC están resumidas en la figura 16-1.

Uso del JDBC

A diferencia del ODBC, con JDBC no hay programa de utilería por separado para crear de una fuente de datos JDBC. En su lugar, todo el trabajo para definir una conexión se hace en código Java mediante un driver JDBC. El patrón de código para la utilización de un driver JDBC es el siguiente:

1. Cargar el driver.
2. Establecer una conexión con la base de datos.
3. Crear una instrucción.
4. Hacer algo con la instrucción.

Como podrá ver, el nombre del producto DBMS que será utilizado y el nombre de la base de datos se proporciona en el paso número dos.

CARGA DEL DRIVER. Para cargar el driver primero debe obtener la biblioteca del driver e instalarlo en un directorio. Necesita asegurarse que el directorio se llame en la TRAYECTORIAdeCLASE (CLASSPATH), tanto para el compilador Java como para la máquina virtual Java. Hay varias formas de cargar el driver en un programa; la más segura es:

```
Class.forName(string).newInstance();
```

El valor del parámetro string depende del driver que utilice. Para los drivers MM MySQL se utiliza:

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

Este método no será la excepción, así que convendría escribir este código en un bloque try:catch (si no es programador de Java, no se desespere, sólo comprenda que estas herramientas hacen disponibles las clases JDBC para el programador).

ESTABLECIMIENTO DE UNA CONEXIÓN A LA BASE DE DATOS. Una vez que haya cargado el driver, el siguiente paso será crear un objeto que tenga una conexión a su base de datos. El formato es:

```
Connection conn = DriverManager.getConnection(string);
```

La clase DriverManager es parte de la biblioteca JDBC que usted cargó en el paso uno. Realiza la misma función que el driver administrador ODBC. Los drivers JDBC se

registran a sí mismos con esta clase. En una máquina determinada puede haber varios drivers registrados. Cuando usted llama a DriverManager.getConnection, busca en su lista de drivers JDBC uno adecuado y lo usa. Seleccionará el primer driver adecuado que encuentre, así que si su conexión puede procesar más de un driver, es probable que usted no consiga el que esperaba.

El parámetro de string que pasó a getConnection tiene tres partes separadas por dos puntos. La primera siempre es "jdbc", la segunda es una palabra clave que identifica al DBMS que está usando, y la tercera es un URL a la base de datos que quiere procesar, junto con los parámetros opcionales tales como usuario y contraseña.

La siguiente instrucción conectará a una base de datos MySQL nombrada vr1 con el usuario *dk1* y la contraseña *sesame*:

```
Connection conn = DriverManager.getConnection
    ("jdbc:mysql://localhost/vr1?user = dk1&password=sesame")
```

El contenido de la segunda y la tercera partes de esta cadena (string) depende de su driver JDBC. En realidad, con algunos drivers determinados usted especifica el nombre del usuario y la contraseña como parámetros por separado. Consulte la documentación de su driver para encontrar el código.

A propósito, la mayoría de esta tecnología surgió en el mundo de Unix. Unix es sensible a mayúsculas y minúsculas y casi todo lo que introduzca aquí también es sensible a mayúsculas y minúsculas. De esta manera, *jdbc* y *JDBC* no son lo mismo. Alimente todo tal como se muestra aquí. Hay pocas excepciones de casos no sensibles a mayúsculas y minúsculas, pero no vale la pena mencionarlos; sólo escriba el texto como se muestra.

El método getConnection (obtenerConexión) será una excepción, así que también deberá aparecer en un bloque try:catch.

CREACIÓN DE UNA INSTRUCCIÓN. El paso siguiente es crear un nuevo objeto Statement. Esto es similar a lo que hicimos en el capítulo anterior para crear un objeto de comando con ADO. La sintaxis es:

```
Statement stmt = conn.createStatement();
```

No hay parámetros para pasar a este método.

En este punto puede procesar la instrucción de las diferentes formas que analizaremos más adelante.

PROCESAMIENTO DE LA INSTRUCCIÓN. Los métodos de Statement están estandarizados en la especificación JDBC. Su driver procesará cualesquiera de las instrucciones que se muestran aquí (y muchas más también). Consulte la documentación API del driver para conocer los detalles. En nuestros ejemplos utilizaremos los métodos de executeQuery y executeUpdate de la siguiente manera:

```
ResultSet rs = stmt.executeQuery (querystring);
```

y

```
Int result = stmt.executeUpdate(updatestring);
```

La primera instrucción regresa un conjunto de resultados que se pueden utilizar de la misma forma en que hemos usado los cursores en los capítulos anteriores. La segunda instrucción regresa un entero que indica el número de renglones actualizados. Algunos ejemplos específicos son:

```
ResultSet rs = stmt.executeQuerry("SELECT * FROM CUSTOMER");
```

y

```
int result = stmt.executeUpdate("UPDATE ARTIST SET Nationality='English'
    WHERE Name='Foster'" );
```

Observe el uso de la comilla sencilla para evitar problemas con los enunciados que están entrecomillados.

Después de que se ha ejecutado el método executeQuery (ejecute Consulta), el objeto resultset se puede repetir para obtener todos los renglones. El número de columnas y los nombres de éstas en el resultset se pueden obtener del método getMetaData. Su sintaxis es:

```
ResultSetMetaData rsMeta = rs.getMetaData();
```

En este punto, los métodos getColumnCount (conseguirCuentaColumna) y getColumnName (conseguirNombreColumna) se pueden invocar desde rsMeta, como podrá verlo en los siguientes ejemplos.

INSTRUCCIONES PREPARADAS E INTRUCCIONES INVOCABLES. Los objetos Prepared Statement y Callable Statement se pueden utilizar para invocar consultas compiladas y procedimientos almacenados en la base de datos. Su uso es similar al del objeto Command que analizamos en el capítulo 15. Debido a que ni las consultas compiladas ni los procedimientos almacenados los maneja MySQL, no los utilizaremos en los ejemplos de este capítulo.

Para ilustrar una instrucción invocable suponga que estamos procesando la base de datos de View Ridge creada con Oracle en el capítulo 12, y que queremos invocar el procedimiento almacenado CustomerInsert. En lo que sigue suponga que *conn* ha sido establecido en una conexión a la base de datos View Ridge de Oracle:

```
CallableStatement cs = conn.prepareCall("{call CustomerInsert(?,?,?,?)}");
cs.setString (1, "Mary Johnson");
cs.setString (2, "212");
cs.setString (3, "555-1234");
cs.setString (4, "US");
cs.execute();
```

Esta secuencia, que llamará al procedimiento almacenado CustomerInsert con los datos que se muestran, es similar a la que se incluye para ODBC en el capítulo anterior. También es posible recibir valores de regreso de los procedimientos, pero eso está más allá del ámbito de nuestro análisis. Véase http://java.sun.com/productos/jdk/1.1/docs/guia/jdbc para mayor información.

La figura 16-2 resume los componentes JDBC. La aplicación crea objetos Conexión, Instrucción, Conjunto de Resultados y ConjuntodeResultadosMetaDatos (Connection, Statement, ResultSet y ResultSetMetaData). Las llamadas de estos objetos se enrutan mediante el DriveManager al driver apropiado. Entonces, los drivers procesan su base de datos. Observe que la base de datos Oracle en esta figura se podría procesar mediante JDBC-ODBC o mediante un driver JDBC puro.

Ejemplos de JDBC

Las figuras 16-3 y 16-4 presentan dos ejemplos en los que se utilizan los drivers mm.mysqlJDBC y MySQL. Observe que ambos programas importan java.sql*. También, observe que los drivers JDBC no son importados, sino que se cargan. Si intenta importarlos el resultado será un desorden.

La base de datos utilizada en todos estos ejemplos es la de View Ridge, que se muestra en la figura 10-3(d). Las tablas en la base de datos son:

CLIENTE (ClienteID, Nombre, CódigodeÁrea, NúmerodeTeléfono, Calle, Ciudad, Estado, CódigoPostal)

ARTISTA (ArtistaID, Nombre, Nacionalidad, FechadeNacimiento, FechadeFallecimiento)

CLIENTE_ARTISTA_INT (*ClienteID, ArtistaID*)

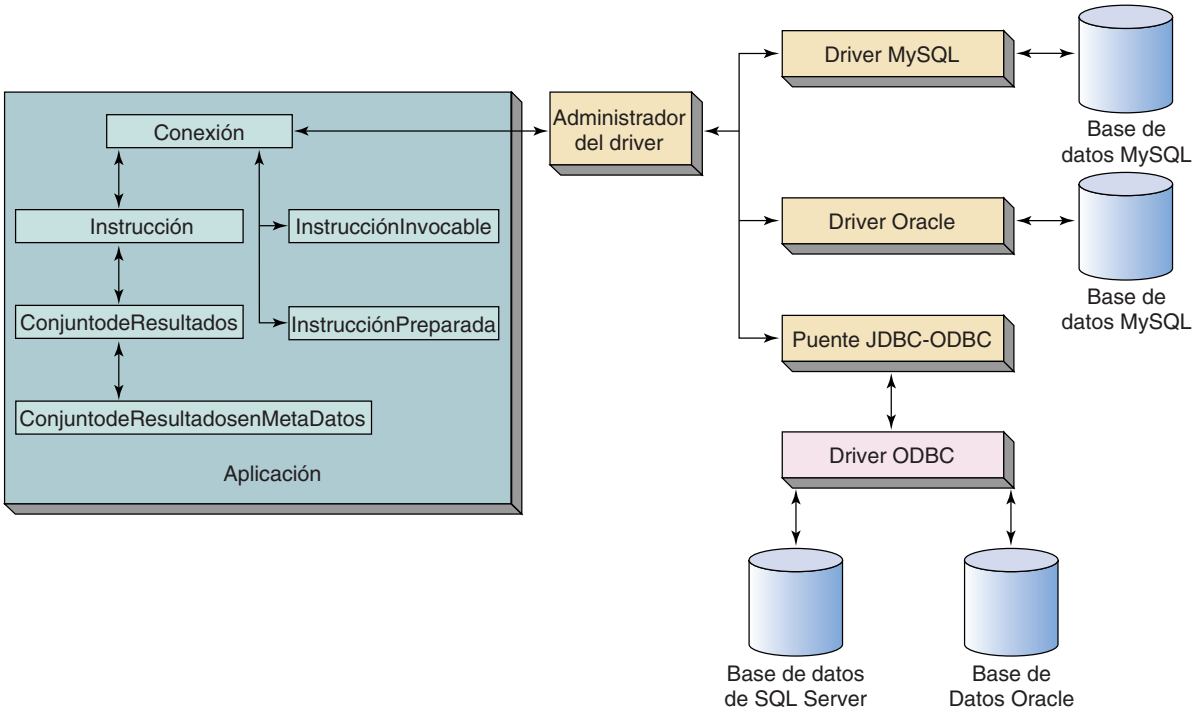
TRABAJO (TrabajoID, Descripción, Título, Copia *ArtistaID*)

TRANSACCIÓN (TransacciónID, FechadeAdquisición, PreciodeCompra, PreciodeVenta, *ClienteID, TrabajoID*)

Las relaciones y la integridad referencial son como las describimos en el capítulo 10.

➤ FIGURA 16-2

Componentes JDBC



LA CLASE GeneralTable. La figura 16-3 muestra la clase Java de GeneralTable. Acepta un solo parámetro que es el nombre de una tabla en la base de datos MySQL vr1. MySQL es sensible a mayúsculas y minúsculas y todos los nombres de las tablas en la base de datos fueron creados con letras mayúsculas. De esta manera, el código debe convertir el nombre introducido de la tabla a letras mayúsculas.

Este ejemplo es una aplicación directa de los conceptos que apenas describimos. La clase de GeneralTable tiene un método accesible al público que no regresa parámetros. (Ése es el significado de "estática pública no válida".) El programa verifica cuando menos un parámetro, coloca la variable verTableName en la entrada de nombre tabla y convierte ese nombre a letras mayúsculas. Entonces procesa la base de datos utilizando los drivers JDBC en un bloque try. Se utiliza un bloque de prueba (try) porque muchos de los métodos arrojan excepciones; éstas se tomarán del bloque catch.

Todo el manejo de excepciones se asume de manera genérica en estos ejemplos. Si programa en Java verá muchas formas diferentes de mejorar el manejo de excepciones que se muestra aquí. Sin embargo, ahora nos enfocaremos en los conceptos de la base de datos.

Si no programa en Java, sólo suponga que todas las instrucciones que aparecen en el bloque try, denotadas mediante "try {. . . . .}," son lo que ocurre bajo circunstancias normales. Todas las instrucciones que aparecen en el bloque catch, denotadas "catch{. . .}," son lo que pasa cuando ocurre un error. También, como en SQL, en Java un comentario multilínea empieza con "/*" y termina con "*/". Una sola línea de comentario comienza con "//".

Los drivers mm.mysql se cargan como se describió antes y entonces se crean un objeto de Conexión conn y un objeto Statement stmt. La base de datos es vr1 y el usuario es dk1. No hay contraseña. (Para este trabajo, se debe haber definido el usuario dk1

en MySQL y otorgado el permiso para usar la base de datos vr1 sin contraseña. Analizaremos estas acciones en la última sección.)

Se crea un ResultSetMetaData llamado rsMeta para el conjunto de resultados (result set rs). Una vez que se ha hecho esto se obtienen los nombres de las columnas y se imprimen en un string largo (varColumnNames). Entonces, rs se vuelve a usar y se despliega en cada columna de cada renglón. La salida no es muy bonita que digamos, pero funciona.

Un despliegue típico es:

```
Showing Table ARTIST
Trying connection with jdbc:mysql://localhost/vr1?user=dk1
Name Nationality Birthdate DeceasedDate ArtistID
Miro Spanish null null 1
```

➤ FIGURA 16-3

Clase GeneralTable

```
import java.io.*;
import java.sql.*;

public class GeneralTable {

      /** A Java program to present the contents of any table
       *  Call with one parameter which is table name --
       *  table name parameter will be converted to uppercase
       */

      public static void main(String[] args) {
            if (args.length <1) {
                  System.out.println ("Insufficient data provided.");
                  return;
            }

            String varTableName = args[0];
            varTableName = varTableName.toUpperCase();
            System.out.println ("Showing Table " + varTableName);
            try {

                  //  Load the MySQL JDBC classes from Mark Mathews
                  //      mm.mysql.jdbc-1.2c

                  Class.forName("org.gjt.mm.mysql.Driver").newInstance();

                  // Set connect string to local MySQL database, user is dk1
                  String connString = "jdbc:mysql://localhost/vr1?user=dk1";

                  System.out.println (" Trying connection with " + connString);
                  Connection conn = DriverManager.getConnection(connString);

                  // Get result set
                  Statement stmt = conn.createStatement();
                  String varSQL = "SELECT * FROM " + varTableName;
                  ResultSet rs = stmt.executeQuery(varSQL);

                  // Get meta data on just opened result set
                  ResultSetMetaData rsMeta = rs.getMetaData();
```

(*continúa*)

➤ FIGURA 16-3

(Continuación)

```
        // Display column names as string
        String varColNames ="";
        int varColCount = rsMeta.getColumnCount();
        for (int col =1; col <= varColCount; col++) {
              varColNames = varColNames + rsMeta.getColumnName(col) +" ";
        }
        System.out.println(varColNames);

        // Display column values
        while (rs.next()) {
              for (int col = 1; col <= varColCount; col++) {
                    System.out.print(rs.getString(col) + "   ");
              }
              System.out.println();
        }

        // Clean up
        rs.close();
        stmt.close();
        conn.close();

}

catch (Exception e) {
        e.printStackTrace();
}
```

```
Tobey US null null 2
Van Vronken US null null 3
Matise French null null 4
```

Like I said, it's not pretty!

LA CLASE CustomerInsert. La figura 16-4 muestra un segundo programa en Java que actualiza la base de datos vr1. Este programa implementa la lógica para el procedimiento View Ridge CustomerInsert, como se describió en los capítulos 10, 12, 13 y 15. (¿Se siente cansado? ¡Cuando menos la lógica es familiar!)

Como recordará, este procedimiento acepta sus parámetros: un nuevo nombre del cliente, CódigodeÁrea, NúmeroLocal y la Nacionalidad (Name, AreaCode, LocalNumber y Nationality) de todos los artistas que le interesan al cliente. Estos parámetros se reciben mediante el procedimiento principal y pasan al método InsertData (InsertarDatos). Éste no es necesario aquí; podríamos tener un método de una sola clase como en la figura 16-3. La lógica se aísla en un método por separado, debido a que en la siguiente sección transformaremos este método en un Java bean. Esa transformación será más fácil si aislamos el código aquí.

El método InsertData carga primero los drivers y después establece un renglón de conexión a vr1 para el usuario dk1. Después verifica si hay datos duplicados consultando vr1 para los datos de entrada Name, AreaCode y LocalNumber (Nombre, Código-

deÁrea y NúmeroLocal). Si encuentra alguno, imprime un mensaje, y el resultset, statement y connection quedan cerrados. De lo contrario, se inserta un nuevo renglón en CUSTOMER.

CustomerID, que es la columna llave sustituta para CUSTOMER, ha sido definida como una columna AUTO_INCREMENT en la base de datos. No es necesario proporcionarle un valor, MySQL lo establecerá.

Si se logra la inserción, la variable *result* no deberá ser igual a cero; si sucede, es que hubo un error durante la actualización. En este caso, se imprime un mensaje y se limpian los objetos. Suponiendo que no ocurra un error, el valor de CustomerID se lee del respaldo de la base de datos y entonces se insertan los renglones en la tabla de intersección CUSTOMER_ARTIST_INT. Esto es muy similar a la lógica que se mostró para los procedimientos almacenados CustomerInsert en Oracle y en SQL Server.

En esta sección de código, la variable *result* no se revisa por el valor cero; una versión mejor de este programa sí lo haría. En realidad, si es programador en Java, sabrá que todos estos mensajes de error y actividades de limpieza deben hacerse utilizando

➤ FIGURA 16-4

Clase CustomerInsert

```
import java.io.*;
import java.sql.*;

public class CustomerInsert {

      /** A Java implementation of the View Ridge Galleries CustomerInsert procedure.
       *  Receives values Customer Name, AreaCode, LocalNumber and Nationality
       *  Inserts the new customer if not already in the database and then
       *  connects that customer to Artists of the given nationality by
       *  adding appropriate rows to the intersection table.
       */

      public static void main(String[] args) {

            if (args.length < 4) {
                  System.out.println ("Insufficient data provided");
                  return;
            }

            String varName = args[0];
            String varAreaCode = args[1];
            String varLocalNumber = args[2];
            String varNationality = args[3];

            insertData(varName, varAreaCode, varLocalNumber, varNationality);

      }

      public static void insertData(String varName,
                                    String varAreaCode,
                                    String varLocalNumber,
                                    String varNationality) {

            System.out.println ("Adding row for " + varName);
            try {

                  // Load JDBC driver class from Mark Mathews
                  // mm.mysql.jdbc-1.2c

                  Class.forName("org.gjt.mm.mysql.Driver").newInstance();
```

(*continúa*)

➤ FIGURA 16-4

(Continuación)

```
// Set up connection to db vr1 with user dk1, no password
String connString = "jdbc:mysql://localhost/" + "vr1" + "?user=dk1";
System.out.println (" Trying connection with " + connString);
Connection conn = DriverManager.getConnection(connString);


// If we get here, we have a connection.  Now check for duplicated data
Statement stmt = conn.createStatement();
String varSQL = "SELECT Name ";
String varWhere = "FROM CUSTOMER WHERE Name= '";
varWhere = varWhere + varName + "' AND AreaCode = '";
varWhere = varWhere + varAreaCode + "' AND PhoneNumber = '";
varWhere = varWhere + varLocalNumber + "'";
varSQL = varSQL + varWhere;

ResultSet rs = stmt.executeQuery(varSQL);
while (rs.next()) {
      // if get here, there is duplicate data
      System.out.println
      ("Data duplicates an existing customer.  No changes made.");
      rs.close();
      stmt.close();
      conn.close();
      return;
}

// OK to insert new data
varSQL = "INSERT INTO CUSTOMER (Name, AreaCode, PhoneNumber)";
varSQL = varSQL + " VALUES ('" + varName + "', '";
varSQL = varSQL + varAreaCode + "', '";
varSQL = varSQL + varLocalNumber + "')";

int result = stmt.executeUpdate(varSQL);
if (result == 0) {
      System.out.println ("Problem with insert");
      rs.close();
      stmt.close();
      conn.close();
      return;
}
```

las excepciones. Estamos caminando en torno a estos asuntos para enfocarnos sólo en aspectos de la base de datos.

Considerando esta rápida introducción a JDBC, por ahora no analizaremos su uso en las Páginas del Servidor Java (JSP).

➤ FIGURA 16-4

(Continuación)

```
        // Update OK, add intersection rows - first get new ID
        varSQL = "SELECT CustomerID " + varWhere;
        rs = stmt.executeQuery(varSQL);
        String varCid ="";
        while (rs.next()) {
              varCid = rs.getString(1);
              if (varCid == "0" ) {
                    System.out.println("Can't find new CustomerID");
                    rs.close();
                    stmt.close();
                    conn.close();
                    return;
              }
        }

        // Now add to intersection table
        varSQL = "SELECT ArtistID FROM ARTIST WHERE Nationality = '" + varNationality +"'";
        String varInsertStart = "INSERT INTO CUSTOMER_ARTIST_INT (CustomerID, ArtistID) VALUES ("
                              + varCid +", ";
        String varInsertEnd = ");";
        rs = stmt.executeQuery(varSQL);
        System.out.println("Adding intersection values for customer " + varCid);
        while (rs.next()) {
              result = stmt.executeUpdate(varInsertStart + rs.getString(1) + varInsertEnd);
        }

        // Clean up
        rs.close();
        stmt.close();
        conn.close();

        }

catch (Exception e){
        e.printStackTrace();
```

➤ PÁGINAS DEL SERVIDOR JAVA

Las Páginas del Servidor Java (JSP) proporcionan un medio para crear páginas Web dinámicas utilizando HTML (y XML) y el lenguaje de programación de Java. Las Páginas del Servidor Java se parecen mucho a las Páginas del Servidor Activo (ASP), pero esto es falso porque la tecnología implícita es totalmente diferente. JSP y ASP parecen similares porque armonizan HTML con el código de programación. La diferencia es que los ASP están restringidos para usar lenguajes de script, como VBScript o JScript. Sin embargo, con JSP el código se hace sólo en Java y no se permiten ni el VBScript ni el JScript. Con Java, las capacidades de un lenguaje por completo orientado a objeto están disponibles para el programador de la página Web.

Debido a que Java es una máquina independiente, las páginas JSP también lo son. Con JSP usted no se limita al uso de Windows 2000 y de IIS. Puede ejecutar la misma página JSP en un servidor Linux, en un servidor Windows, y también en otros.

La especificación oficial para JSP se puede encontrar en: http://java.sun.com/products/jsp.

PÁGINAS JSP Y SERVLETS

Las páginas JSP se transformaron a lenguaje estándar Java y, por lo tanto, se compilan como un programa regular. En particular, se transformaron en servlets Java, lo que significa que, tras bambalinas, las páginas JSP se transformaron en subclases de la clase HTTPServlet. Así, el código JSP tiene acceso a la petición HTTP y a los objetos respuesta, a sus métodos y también a otra funcionalidad HTTP.

Debido a que las páginas JSP se convirtieron en subclases servlet, no se necesita codificar las clases Java o los métodos en una página JSP. Puede insertar fragmentos del código Java donde quiera y se colocarán correctamente en una subclase servlet cuando la página se amplíe. Así, se podrían colocar las siguientes instrucciones en una página JSP sin cualquier otro código Java y se ejecutarían bien:

```
<% String partyName = “fiesta”;
partyName = partyName.toUpperCase();
out.println (“Come to our”  + partyName); %>
```

En este caso, la cadena “Come to our FIESTA (“Ven a nuestra FIESTA”) se desplegará en el explorador cuando se procese esta sección de la página JSP. A propósito, observe que el código Java está aislado entre <% y %> igual que VBScript y JScript en las páginas ASP.

Con el fin de usar las páginas JSP, su servidor de la red debe implementar el servlet Java 2.1+ y las especificaciones de las Páginas del Servidor Java 1.0+. En http://java.sun.com/products/servlet/industry.html puede verificar una lista de los servidores que dan sustento a estas especificaciones. Por lo menos hay media docena de posibilidades. En lo que resta de este capítulo usaremos Tomcat para este propósito.

Apache Tomcat

A partir de diciembre de 2000 el servidor de red Apache no da soporte a los servlets. Sin embargo, la fundación Apache y Sun se incorporaron al Proyecto Jakarta que desarrolló un procesador para servlet llamado Apache Tomcat. Puede obtener la fuente y el binario de Tomcat en el sitio de la red, Proyecto Jakarta, en http://jakarta.apache.org.

Tomcat es un procesador para servlet que puede trabajar en conjunto con Apache como servidor de la red autónomo. Tomcat tiene facilidades limitadas al servidor de la red; sin embargo, se utiliza normalmente en el modo autónomo sólo para examinar los servlets y las páginas JSP. Debe utilizarse Tomcat en conjunto con Apache para aplicaciones comerciales y de producción.

Si está ejecutando Tomcat y Apache por separado en el mismo servidor de la red, necesitará utilizar diferentes puertos. El puerto predeterminado para un servidor de la red es de 80 y Apache por lo general lo utiliza. Cuando se usa de modo autónomo, Tomcat está configurado para un puerto de 8080, aunque por supuesto se puede cambiar.

En los ejemplos que siguen Tomcat está utilizando el puerto 8080. Estos ejemplos se ejecutaron en una Intranet privada en la que la máquina servidor Tomcat se asignó a la dirección IP 10.0.0.3. Así, para invocar la página *somepage.jsp* usaremos la cadena http://10.0.0.3:8080/somepage.jsp en el campo de dirección del explorador.

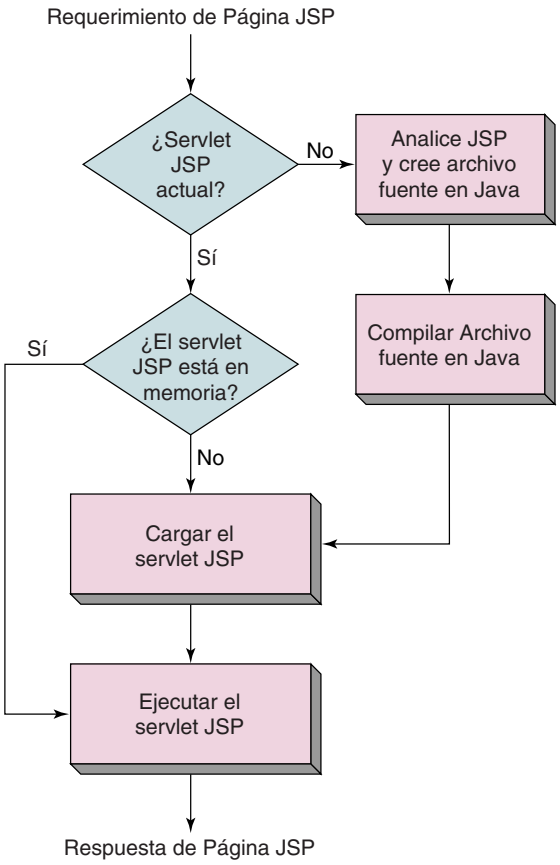
Instalación de Tomcat para procesamiento JSP

Cuando instale Tomcat se creará una estructura de directorio en la que debe colocar las bibliotecas de clases y las páginas Web. Al igual que con la versión Tomcat 3.1, coloque las bibliotecas de clases en el directorio *install-dir*/lib y las páginas JSP en el directorio de clases *install-dir*/webapps/ROOT/WEB-INF/. En Linux, la utilidad RPM instala Tomcat en el directorio predeterminado /usr/local/jakarta-tomcat/. Por lo tanto, en este caso coloque las bibliotecas en /usr/local/jakarta-tomcat/lib y las páginas JSP dentro de /urs/local/jakarta-tomcat/webapps/ROOT/WEB-INF/classes. Si está instalando con otro sistema operativo, o con un procesador servlet diferente, deberá consultar las instrucciones.

Cuando se instala una nueva clase de archivos en el subdirectorio lib, hay una pequeña consideración: Tomcat crea su CLASSPATH (ruta de clases) cuando inicia. De esta manera, después de que haya instalado una nueva clase de archivos dentro del directorio lib, debe detener y reiniciar Tomcat antes de ver su nuevo archivo. Si sólo copió el nuevo archivo dentro del subdirectorio lib sin reiniciar Tomcat, recibirá una lección inolvidable. (Créame, lo sé. . .)

➤ FIGURA 16-5

Proceso de Compilación JSP



Las páginas JSP que siguen usan mm.mysql de los drivers MySQL. Para trabajar, la biblioteca de clases de drivers apropiada debe colocarse en el subdirectorio lib. En estos ejemplos, se instaló el archivo "mm_uncomp.jar" en ese directorio.

La figura 16-5 muestra el proceso por el cual se compilan las páginas JSP. Cuando se recibe un requerimiento para una página JSP, un procesador servlet Tomcat (u otro) encuentra la versión compilada de la página y la verifica para determinar si es actual. Lo hace para buscar una versión no compilada de la página, que tenga fecha y hora de creación después de la hora y fecha de la página compilada. Si la página no es actual, analiza la nueva y la transforma en un archivo fuente en Java y después compila ese archivo fuente. El servlet entonces se carga y ejecuta. Si la página JSP compilada es actual, la carga en memoria, y si aún no está ahí, entonces la ejecuta. Si está en memoria, simplemente la ejecuta.

(A propósito, la desventaja de esta compilación automática es que si usted comete errores de sintaxis y olvida examinar sus páginas, ¡el primer usuario en ingresar a su página recibirá los errores de compilación!)

A diferencia de los archivos CGI, y de algunos otros programas del servidor de la red, hay más de una copia de la página JSP en la memoria a la vez. Más adelante se ejecutan las páginas mediante una de las conexiones de Tomcat, y no a través de un proceso independiente. Esto significa que se requerirá mucho menos memoria y tiempo del procesador para ejecutar una página JSP, que ejecutar un script CGI comparable.

Ejemplos JSP

Esta sección analiza dos páginas JSP simples. La primera es una versión de la clase GeneralTable que se mostró en la figura 16-3. La segunda encapsula la lógica en la figura 16-4 en un Java bean, y después invoca ese bean desde una página JSP.

GeneralTable.JSP. La figura 16-6 muestra una página JSP que despliega los contenidos de cualquier tabla en la base de datos MySQL nombrada vr1. El formato y la lógica de esta página es similar a la de GeneralTable.asp en la figura 15-21. Aquí suponemos que el usuario pasa el nombre de la tabla para desplegarla como un parámetro para la página.

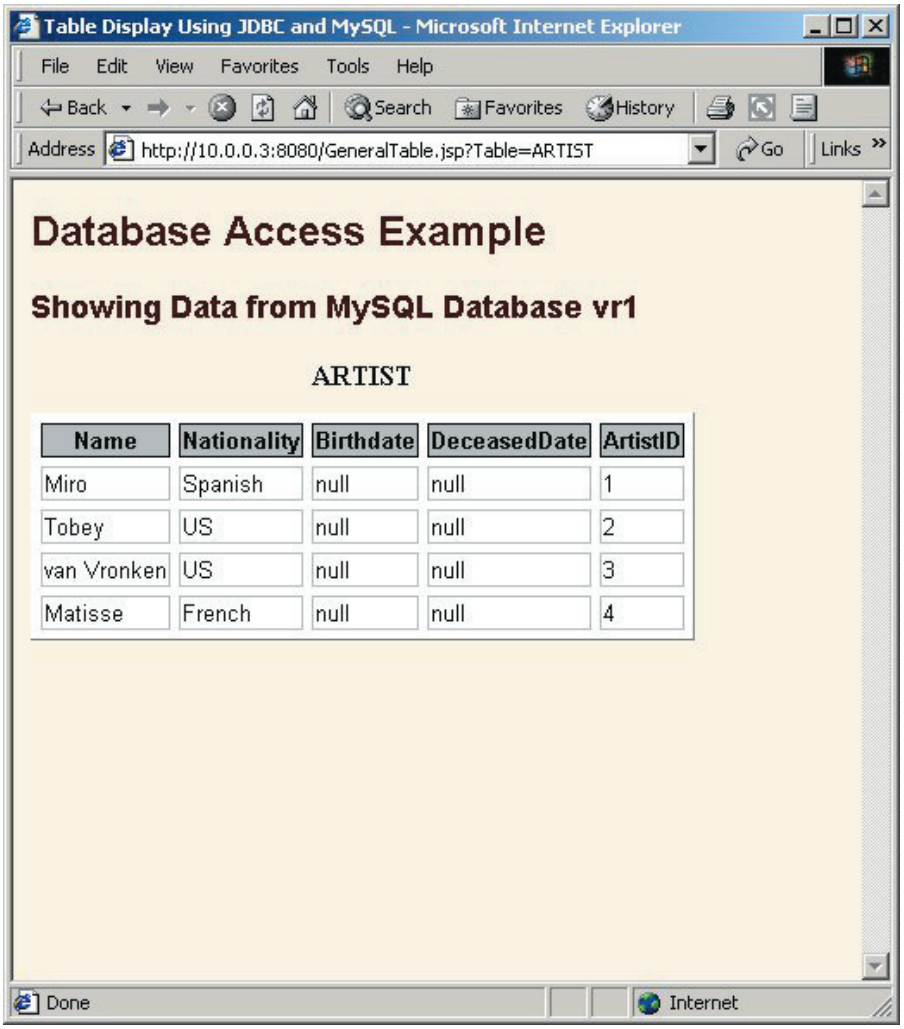
➤ FIGURA 16-6

GeneralTable.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- Example of Database Access from a JSP Page -->
<%@ page import="java.sql.*" %>
<HTML>
<HEAD>
<TITLE>Table Display Using JDBC and MySQL</TITLE>
<META NAME="author" CONTENT="David Kroenke">
<META NAME="keywords"
      CONTENT="JSP,JDBC,Database Access">
<META NAME="description"
      CONTENT="An example of displaying a table using JSP.">
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<H2>Database Access Example</H2>
<% String varTableName= request.getParameter("Table");
   varTableName = varTableName.toUpperCase(); %>
<H3>Showing Data from MySQL Database vr1</H3>
<%
try     {
    // Load the Mark Mathew MySQL JDBC Drivers
    Class.forName("org.gjt.mm.mysql.Driver").newInstance();

    // Connect to vr1 with user dk1
    String connString = "jdbc:mysql://localhost/" + "vr1" + "?user=dk1";
        Connection conn = DriverManager.getConnection(connString);

    // Get rs and rsMeta for the SELECT statement
    Statement stmt = conn.createStatement();
    String varSQL = "SELECT * FROM " + varTableName;
    ResultSet rs = stmt.executeQuery(varSQL);
    ResultSetMetaData rsMeta = rs.getMetaData();
%>
<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=5><FONT FACE="Arial" COLOR=#000000
><CAPTION><B><%=varTableName%></B></CAPTION></FONT>
<THEAD>
<TR><%
    String varColNames ="";
    int varColCount = rsMeta.getColumnCount();
    for (int col =1; col <= varColCount; col++) {
        %><TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000 ><FONT SIZE=2 FACE ="Arial" COLOR=#000000
        ><%=rsMeta.getColumnName(col)%></FONT> </TH>
<%  }%>
</TR>
</THEAD>
<TBODY><%
    while (rs.next()) {
         %><TR VALIGN=TOP><%
              for (int col = 1; col <= varColCount; col++) {
           %><TD BORDERCOLOR=#C0C0C0 ><FONT SIZE=2 FACE="Arial" COLOR=#000000
           ><%=rs.getString(col)%><BR></FONT></TD>
<%        }
    }
        // Clean up
        rs.close();
        stmt.close();
        conn.close();
}
catch (ClassNotFoundException e) {
    out.println("Driver Exception " + e);
}%>
</TR>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

➤ FIGURA 16-7

Despliegue de la GeneralTable ARTIST



La figura 16-7 muestra los resultados de invocar esta página usando Internet Explorer en una computadora Windows 2000. La página se procesó con Tomcat en una computadora Linux. Observe la llamada al puerto 8080; en producción, Tomcat se ejecutaría con Apache en el puerto 80 y no sería necesaria esta especificación del puerto.

En la figura 16-6 se muestra el código Java en negritas. La primera línea invoca una página de directorio, la cual importa la biblioteca java.sql. Entonces, el parámetro que tiene el nombre de la tabla se obtiene utilizando el requerimiento del objeto HTTP del método getParameter (obtener parámetro). El valor se coloca con letras mayúsculas. A continuación se cargan las clases JDBC, como se hizo en la figura 16-3, y se crea una conexión a la base de datos vtr1 para el usuario vr1. El resto del código es el mismo que el de la figura 16-4, sólo se expande entre las instrucciones HTML utilizadas para el despliegue de los resultados.

De nuevo, esta página parece engañosamente similar a la versión ASP de esta página, la GeneralTable.asp que se muestra en el capítulo 15. La diferencia no es sólo que se utiliza JDBC en lugar de ADO y ODBC. Una diferencia aún mayor es que esta página será compilada en un programa Java y, por lo tanto, es portátil y más rápida.

CustomerInsertUsingBean.JSP (InsertarClienteUsandoBean.JSP). Debido a que Java se utiliza con páginas JSP, están disponibles las capacidades totales de un programa orientado a objetos. Esto significa que las páginas JSP pueden solicitar objetos precompilados. Hacer esto es importante y útil por varias razones. En primer lugar, separa las tareas del programa de escritura lógica de la generación HTML. Las empresas pueden tener al

mismo tiempo grupos diferentes y gente trabajando en tareas muy diferentes. También permite encapsular la lógica en módulos independientes para reuso y otros beneficios del encapsulado. Finalmente, reduce la complejidad de administrar un sitio de la red.

Si usted no programa en Java, ignore el párrafo siguiente. Piense en un bean como en una clase de Java de modales muy finos (como una de esas chicas que podría llevar a casa tranquilamente y presentársela a su mamá.)

En términos simples, un bean Java es una clase de Java que tiene tres propiedades: primera, no hay variables de ejemplos. Segunda, accede a todos los valores constantes utilizando métodos denominados getxxx y setxxx. Por ejemplo, un valor constante llamado miValor se obtiene mediante un método llamado getmyValue () (obtenerMiValor) y se establece mediante un método llamado setMyValue () (establecerMiValor). Finalmente, la clase bean no tiene constructores o debe tener un constructor de argumento cero definido explícitamente.

➤ FIGURA 16-8

Clase CustomerInsertBean

```
import java.io.*;
import java.sql.*;

public class CustomerInsertBean {

      /** A Java bean for the View Ridge Galleries CustomerInsert procedure.
       *  Persistent values obtained by accessors getxxx and setxxx
       *
       *  Inserts the new customer if not already in the database and then
       *  connects that customer to Artists of the given nationality by
       *  adding appropriate rows to the intersection table.
       */

      private String newName = "unknown";
      private String newAreaCode = "";
      private String newLocalNumber = "";
      private String newNationality = "";


      public String getnewName() {
            return(newName);
      }

      public void setnewName(String newName) {
            if (newName != null) {
                  this.newName = newName;
            } else {

                  this.newName = "unknown";
            }
      }

      public String getnewAreaCode() {
            return(newAreaCode);
      }

      public void setnewAreaCode(String newAreaCode) {
            if (newAreaCode != null) {
                  this.newAreaCode = newAreaCode;
            } else {
                  this.newName = "";
```

➤ FIGURA 16-8

(Continuación)

```
public String getnewLocalNumber() {
      return(newLocalNumber);
}

public void setnewLocalNumber (String newLocalNumber) {
      if (newLocalNumber != null) {
            this.newLocalNumber = newLocalNumber;
      } else {
            this.newName = "";
      }
}

public String getnewNationality() {
      return(newNationality);
}

public void setnewNationality(String newNationality) {
      if (newNationality != null) {
            this.newNationality = newNationality;
      } else {
            this.newName = "";
      }
}


public String InsertData() {

      try {

            // Load JDBC driver class from Mark Mathews
            // mm.mysql.jdbc-1.2c

            Class.forName("org.gjt.mm.mysql.Driver").newInstance();

            // Set up connection to db vr1 with user dk1, no password
            String connString = "jdbc:mysql://localhost/" + "vr1" + "?user=dk1";
            Connection conn = DriverManager.getConnection(connString);

            // If we get here, we have a connection.  Now check for duplicated data
```

(*continúa*)

La figura 16-8 muestra un Java Bean llamado CustomerInsertUsingBean. Esta clase tiene un método que implementa el procedimiento CustomerInsert de la Galería View Ridge (sí, ¡REGRESÓ!, pero por última vez). Esta clase tiene cuatro valores constantes: newName, newAreaCode, newLocalNumber y newNationality (nuevoNombre, nuevoCódigodeÁrea, nuevoNúmeroLocal y nuevaNacionalidad). Para cada uno de estos valores constantes se definen los métodos de acceso getXXX y setXXX. No hay valores públicos constantes ni método constructor. Por lo tanto, CustomerInsertUsingBean reúne los requisitos para un bean.

El método actual de actualización está codificado en uno llamado InsertData (InsertarDatos). Éste es idéntico al método InsertData de la figura 16-4.

La figura 16-9(a) muestra una forma de entrada de datos para reunir los datos del cliente nuevo. La página HTML para esta forma se muestra en la figura 16-9(b). Observe que el valor de FORM ACTION es CustomerInsertUsingBean.jsp. También observe que las cajas de texto se llaman newName, NewAreaCode, NewLocalNumber y newNationality. Estos nombres tienen importancia porque cuando se hace clic en Add

➤ FIGURA 16-8

(Continuación)

```
Statement stmt = conn.createStatement();
String varSQL = "SELECT Name ";
String varWhere = "FROM CUSTOMER WHERE Name= '";
varWhere = varWhere + newName + "' AND AreaCode = '";
varWhere = varWhere + newAreaCode + "' AND PhoneNumber = '";
varWhere = varWhere + newLocalNumber + "'";
varSQL = varSQL + varWhere;
ResultSet rs = stmt.executeQuery(varSQL);
while (rs.next()) {
      // if get here, there is duplicate data
      rs.close();
      stmt.close();
      conn.close();
      return ("Duplicate data - no action taken");
}

// OK to insert new data
varSQL = "INSERT INTO CUSTOMER (Name, AreaCode, PhoneNumber)";
varSQL = varSQL + " VALUES ('" + newName + "', '";
varSQL = varSQL + newAreaCode + "', '";
varSQL = varSQL + newLocalNumber + "')";
int result = stmt.executeUpdate(varSQL);
if (result == 0) {
      // if get here, there is a problem with insert
      rs.close();
      stmt.close();
      conn.close();
      return ("Problem with insert");
}

// Update OK, add intersection rows - first get new ID
varSQL = "SELECT CustomerID " + varWhere;
rs = stmt.executeQuery(varSQL);
String varCid ="";
while (rs.next()) {
      varCid = rs.getString(1);
      if (varCid == "0" ) {
            // if get here, can't find new CustomerID
            rs.close();
            stmt.close();
            conn.close();
            return ("Can't find new customer after insert");
```

Customer, CustomerInsertUsingBean.jsp pasará los parámetros con estos nombres. Como podrá ver, JSP puede empatar los parámetros de entrada con las mismas propiedades del objeto del mismo nombre cuando se le ordene.

La página JSP CustomerInsertUsingBean.jsp está limitada en la figura 16-10. Las instrucciones importantes de esta página son las dos instrucciones jsp:

```
<jsp:useBean id= “insert” class =“CustomerInsertBean” />
```

➤ FIGURA 16-8

(Continuación)

```
        }
}

// Now add to intersection table
varSQL = "SELECT ArtistID FROM ARTIST WHERE Nationality = '"
                                        + newNationality +"'";
String varInsertStart = "INSERT INTO CUSTOMER_ARTIST_INT
                            (CustomerID, ArtistID) VALUES

            → (" + varCid +", ";
String varInsertEnd = ")";
rs = stmt.executeQuery(varSQL);
while (rs.next()) {
      result = stmt.executeUpdate
          (varInsertStart + rs.getString(1) + varInsertEnd);
}

// Clean up
rs.close();
stmt.close();
conn.close();
return("Success");
}

catch (Exception e){
      return ("Exception: " + e );
}
```

y

```
<jsp:setPropertyName = “insert” property =“*” />
```

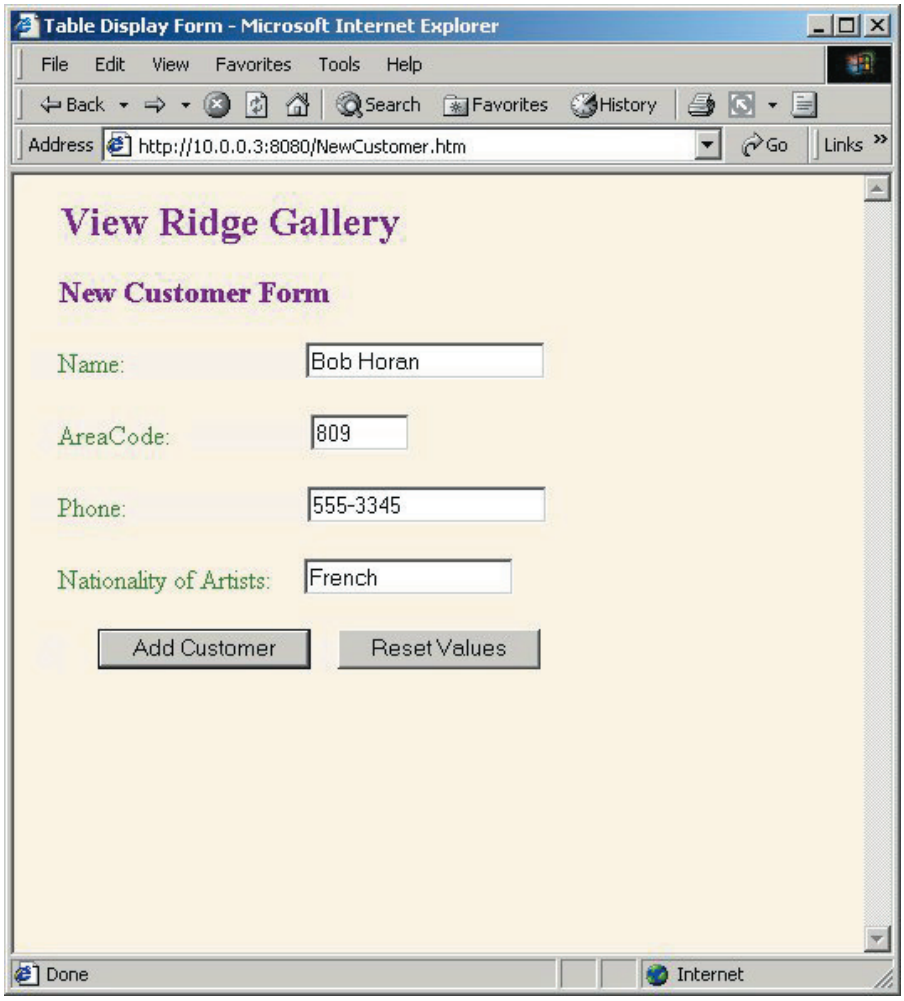
La primera instrucción le dice al compilador JSP que cargue la clase CustomerInsertBean y lo afilia con el nombre *insert*. Para este trabajo con Tomcat 3.1 debe haber una versión compilada del bean llamado CustomerInsertBean.class en el directorio *install-dir*/webapps/ROOT/WEB-INF/classes. Para el estándar install RPM, esto debería ser /urs/local/jakarta-tomcat/webapps/ROOT/WEB-INF/classes.

La segunda instrucción jsp: establece que el JSP analiza los conjuntos de las propiedades de clase usando la forma de los parámetros de entrada. Los signos “*” señalan todas las propiedades con los parámetros del mismo nombre. Esta instrucción es un sustituto corto para lo siguiente:

```
<jsp:setProperty name=“insert”
      property=“newName”
      value=‘<%= request.getParameter(“newName”) %>’ />
<jsp:setProperty name=“insert”
      property=“newAreaCode”
      value=‘<%= request.getParameter(“newAreaCode”) %>’ />
<jsp:setProperty name=“insert”
      property=“newLocalNumber”
      value=‘<%= request.getParameter(“newLocalNumber”) %>’ />
<jsp:setProperty name=“insert”
      property=“newName”
      value=‘<%= request.getParameter(“newNationality”) %>’ />
```

➤ FIGURA 16-9

Forma de entrada de datos del nuevo cliente: (a) forma NewCustomer, (b) forma HTML para NewCustomer



(a)

Por supuesto, se puede utilizar cualquier versión. De hecho, se requiere la versión más larga si los nombres de los parámetros de la forma son diferentes a los nombres de las propiedades del objeto.

En la figura 16-10 se invoca el método InsertData mediante la instrucción:

```
String result=insert.InsertData();
```

Si el resultado no es "Success" ("Éxito") se imprime el mensaje resultante. De lo contrario, el join de las tablas CUSTOMER, CUSTOMER_ARTIST_INT, ARTIST se despliega utilizando un código muy similar al de la figura 16-3. El resultado aparece como en la figura 16-11.

Ésta es una introducción muy breve al desarrollo con JSP. Hay mucho más por utilizar y comprender, pero un análisis más largo rebasaría el ámbito de este libro. Una referencia excelente sobre este tema es *Core Servlets and Java Server Pages*, por Marty Hall.[1]

[1] Marty Hall, *Core Servlets and Java Server Pages*. Upper Saddle River, NJ: Prentice Hall, 2000.

➤ FIGURA 16-9

(Continuación)

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html">
<TITLE>Table Display Form</TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>

<FORM METHOD="post" ACTION="CustomerInsertUsingBean.jsp">

 <P><STRONG><FONT color=purple face="" size=5>   View Ridge
Gallery</FONT></STRONG>

 <P>   <strong><font color="purple" face size="4"> New Customer
Form</font></strong><P><font style="background-color: #FDF5EC"
color="forestgreen" face>   
 Name:   </font><FONT style="BACKGROUND-COLOR:
#FDF5EC">           &nbs
p;            
 </FONT><INPUT id=newName name=newName></P>

 <P> <font style="background-color: #FDF5EC" color="forestgreen"
face>   AreaCode<font style="background-color:
#FDF5E6">:   </font>       &n
bsp;         
 </font><INPUT id=newAreaCode name=newAreaCode size="6"></P>

 <P><font style="background-color: #FDF5EC" color="forestgreen"
face>   

Phone:            &
nbsp;           &n
bsp;
 </font><INPUT id=newLocalNumber name=newLocalNumber
size="20">        </P>

 <P>    <font style="background-color: #FDF5EC"
color="forestgreen" face>Nationality
 of Artists:  </font>   <INPUT id=newNationality
name=newNationality size="17"></P>

 <P> <FONT style="BACKGROUND-COLOR: #FDF5EC">
   </FONT>     <FONT style="BACKGROUND-
COLOR: #FDF5EC">
<INPUT id=submit1 name=submit1 type=submit value="Add Customer"
>   
<INPUT id=reset1 name=reset1 type=reset value="Reset Values"></FONT></P>

</FORM>
</BODY>
</HTML>
```

(b)

➤ FIGURA 16-10

Insertar Cliente utilizando Bean.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!--
Example of Database Access from a JSP Page
-->
<%@ page import="java.sql.*" %>
<HTML>
<HEAD>
<TITLE>Updating Using a Java Bean</TITLE>
<META NAME="author" CONTENT="David Kroenke">
<META NAME="keywords"
      CONTENT="JSP,JDBC, Database Access">
<META NAME="description"
      CONTENT="An example of invoking a bean and displaying results.">
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>
<BODY>
<H2>Database Update Using JDBC from a Java Bean</H2>
<H3>Processing the View Ridge Customer Insert for MySQL Database vr1</H3>
<jsp:useBean id="insert" class="CustomerInsertBean" />
<jsp:setProperty name="insert" property="*" />
<%
//  Bean properites were set in statement above, now call the
//  bean for insert
String result=insert.InsertData();

if (result != "Success") {
    // print problem and return
    out.println("Problem" + result);
    return;
}
//  Data was inserted successfully; now display the intersection table
try     {
    // Load the Mark Mathew MySQL JDBC drivers
    //

    Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    String connString = "jdbc:mysql://localhost/" + "vr1" + "?user=dk1";

    Connection conn = DriverManager.getConnection(connString);

    // Join Customer to Artist via intersection table
    // Note synonyms for CUSTOMER.Name and ARTIST.Name
    Statement stmt = conn.createStatement();
    String varSQL = "SELECT CUSTOMER.Name Customer, ARTIST.Name  Artist, Nationality ";
    varSQL = varSQL + "FROM CUSTOMER, CUSTOMER_ARTIST_INT, ARTIST ";
    varSQL = varSQL + "WHERE CUSTOMER.CustomerID = CUSTOMER_ARTIST_INT.CustomerID AND ";
    varSQL = varSQL + "ARTIST.ArtistID = CUSTOMER_ARTIST_INT.ArtistID";
    ResultSet rs = stmt.executeQuery(varSQL);
    ResultSetMetaData rsMeta = rs.getMetaData();
%>
<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=5><FONT FACE="Arial" COLOR=#000000>
<CAPTION><B>Customers and Interests</B></CAPTION></FONT>
<THEAD>
<TR><%
    String varColNames ="";
    int varColCount = rsMeta.getColumnCount();
    for (int col =1; col <= varColCount; col++) {
        %><TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000 ><FONT SIZE=2 FACE ="Arial" COLOR=#000000
        ><%=rsMeta.getColumnName(col)%></FONT> </TH>
<%  }%>
</TR>
</THEAD>
<TBODY><%
    while (rs.next()) {
         %><TR VALIGN=TOP><%
       for (int col = 1; col <= varColCount; col++) { %>
              <TD BORDERCOLOR=#C0C0C0 ><FONT SIZE=2 FACE="Arial" COLOR=#000000
              ><%=rs.getString(col)%><BR></FONT></TD><%
```

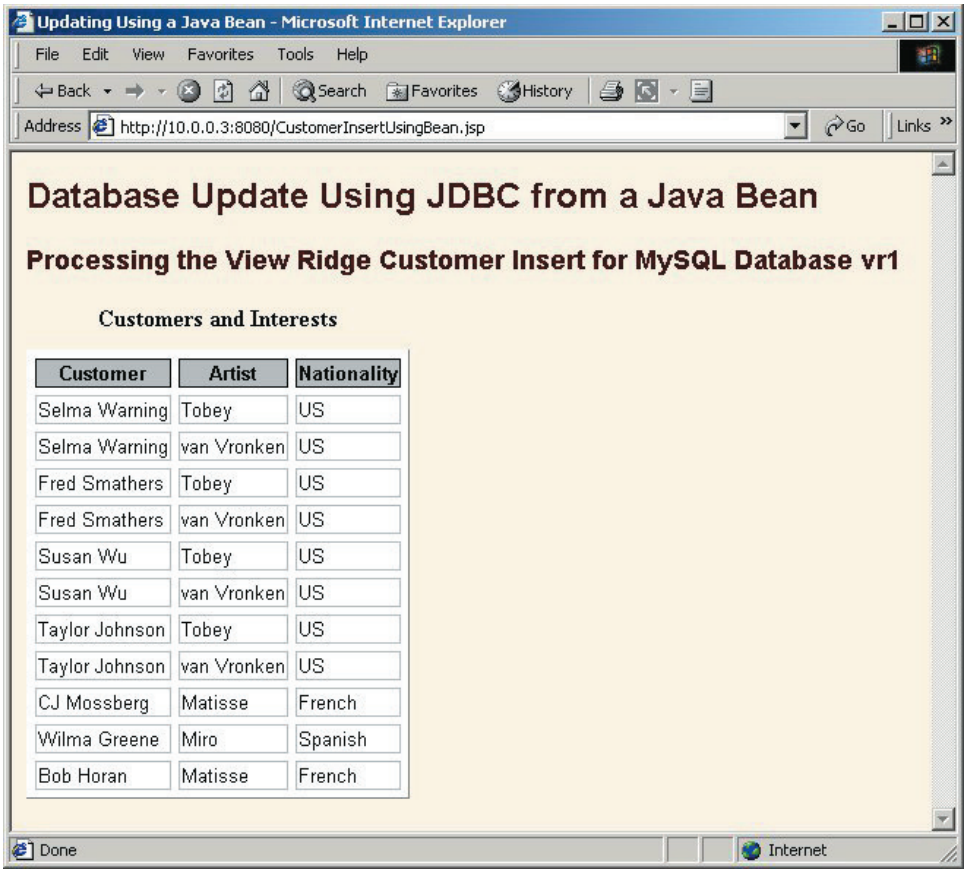
➤ FIGURA 16-10

(Continuación)

```
=rs.getString(col)%><BR></FONT></TD><%
        ]
    }
    // Clean up
    rs.close();
    stmt.close();
    conn.close();
}
catch (ClassNotFoundException e) {
    out.println("Driver Exception  " + e);
}%>
</TR>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

➤ FIGURA 16-11

Resultado de CustomerInsertUsingBean.jsp



➤ MYSQL

MySQL es un producto DBMS open source que corre en Unix, Linux y Windows. Puede cargar el código fuente y binario MySQL del sitio de la red MySQL: http://www.mysql-.com. Los ejemplos en este texto se ejecutaron en MySQL en Linux, pero los ejemplos que se presentan aquí funcionarían en MySQL y en otros sistemas operativos. Desde

diciembre del 2000 no hay costo de licencia, a menos que construya MySQL en una aplicación comercial. Vea el acuerdo de licencia en el sitio MySQL para mayor información.

MySQL carece de muchas de las capacidades de productos comerciales DBMS, como Oracle y SQL Server; si tiene acceso a uno de estos productos probablemente debería usarlo. Sin embargo, si está trabajando con poco presupuesto, o si quiere participar en el movimiento de open source, entonces MySQL puede ser una buena elección. En el medio ambiente de Linux/Unix, MySQL no sólo es más barato que Oracle y otros productos comerciales, sino más fácil de instalar. Una buena referencia para MySQL es *MySQL* de Paul DuBois.[2]

Irónicamente, debido a su administración limitada de transacciones y a sus capacidades logísticas, MySQL es muy rápido para aplicaciones de consultas puras. Existen compañías de la red orientadas a la publicidad de datos que mantienen sus bases de datos utilizando Oracle, pero las cargan a MySQL para publicación de consultas en sus servidores Web.

Limitaciones de MySQL

A partir de la versión 3.x, MySQL no maneja vistas, procedimientos almacenados, o disparadores. Todos están en la lista de cosas por hacer de MySQL; sin embargo, revise la documentación más reciente para verificar si en versiones más recientes se han agregado algunas de estas características al producto.

Además, mientras que MySQL analice correctamente las restricciones de llaves externas (integridad referencial), no hace nada con ellas. Esto significa que, por ejemplo, analizará una expresión ON DELETE CASCADE en una restricción de llave sustituta, pero no realizará las eliminaciones en cascada que usted espera. Por ejemplo, en la aplicación de View Ridge, MySQL analizará la siguiente restricción sin ningún problema:

```
ALTER TABLE CUSTOMER_ARTIST_INT ADD CONSTRAINT CustomerIntFK
FOREIGN KEY(CustomerID) REFERENCES CUSTOMER ON DELETE
CASCADE;
```

Sin embargo, cuando elimine un renglón en CUSTOMER no se harán las eliminaciones en cascada. Necesitará hacerlas usted mismo.

Uso de MySQL

Para iniciar MySQL desde el símbolo de arranque, escriba:

```
MySQL -u username -p
```

Ponga un nombre de usuario válido; se le pedirá que introduzca una contraseña. Si está utilizando una cuenta que no tiene contraseña, puede teclear:

```
MySQL -u username
```

Para ver qué bases de datos han sido creadas, escriba:

```
Show databases;
```

Observe que las instrucciones MySQL terminan con punto y coma. También, las instrucciones MySQL son insensibles a mayúsculas y minúsculas; pero los nombres que ha definido el programador para las construcciones, tales como nombres de tablas y nombres de columnas, sí son sensibles a mayúsculas y minúsculas.

---

[2] Paul DuBois, *MySQL*. Indianapolis, IN: New Riders, 2000.

USO DE UNA BASE DE DATOS EXISTENTE. Para usar una de las bases de datos introduzca:

Use *databasename*;

Por ejemplo,

Use vr1;

Para determinar las tablas en esta base de datos, introduzca:

Show tables;

Puede desplegar la tabla de metadatos con la instrucción describir:

Describe CUSTOMER;

En este punto, se puede utilizar cualquier instrucción estándar SQL. SELECT, UPDATE, INSERT y DELETE funcionan como usted esperaría.

CREACIÓN DE UNA NUEVA BASE DE DATOS. Para crear una nueva base de datos dé de alta en MySQL la cuenta que quiere que tenga la nueva base de datos. Entonces introduzca:

Create Database *newdatabasename*;

Por ejemplo,

Create Database vr2;

La base de datos será creada en ese momento y puede alimentar instrucciones create SQL, como lo hemos hecho antes. La figura 16-12 muestra las instrucciones que puede usar para crear la base de datos View Ridge. Las llaves sustitutas se dan en la propiedad AUTO_INCREMENT. Este tipo de datos es una secuencia que mantiene MySQL, la cual comienza en uno y se incrementa de uno en uno. Observe que MySQL da soporte a los tipos de datos de Year. Este tipo de datos, que es un entero de cuatro dígitos, se usa para las columnas Birthdate y DeceasedDate en ARTIST. Observe también que no están definidas las restricciones de llave externa. Como se indicó antes, MySQL las analizará, pero no hará nada con ellas.

El esquema en la figura 16-12 crea un índice único en (Title, Copy, ArtistID) en WORK. Este índice único evita que se inserten renglones duplicados en la base de datos. Esto significa que la lógica en los programas en Java los muestra primero para evitar renglones duplicados que son innecesarios. No cambia nada, excepto tal vez el desempeño de un solo bit. Como se indicó en el capítulo 10, siempre es mejor imponer las reglas de integridad en la base de datos, si es posible.

Puede escribir todas estas instrucciones en MySQL. Sin embargo, si las tiene en un archivo las puede importar a MySQL. Suponga que las instrucciones en la figura 16-12 están en un archivo llamado VRSQL.txt. Para procesarlas, introduzca la siguiente orden en el prompt de shell (¡no en MySQL!):

command prompt$ mysql--user=dk1 - -password=sesame<VRSQL.txt

Esto iniciará MySQL para el usuario *dk1* y la contraseña *sesame* y procesará las instrucciones en el archivo de texto.

Establecimiento de permisos de acceso para el uso de JDBC

Una vez que comprenda las limitaciones de MySQL, lo usará con facilidad. Las instrucciones SQL se procesan exactamente como se esperaría. Sin embargo, hay una pequeña deformación que debería conocer. Las conexiones de JDBC son tratadas de manera di-

➤ FIGURA 16-12

Instrucciones SQL para la creación de la base de datos de View Ridge en MySQL

```
CREATE TABLE CUSTOMER(
CustomerID          int          AUTO_INCREMENT PRIMARY KEY,
Name                varchar(25)  NULL,
Street              varchar(30)  NULL,
City                varchar(35)  NULL,
State               varchar(2)   NULL,
Zip                 varchar(9)   NULL,
AreaCode            varchar(3)   NULL,
PhoneNumber         varchar(8)   NULL);

CREATE INDEX CUSTOMER_Name_IDX ON   CUSTOMER(Name);

CREATE TABLE ARTIST(
ArtistID            int          AUTO_INCREMENT PRIMARY KEY,
Name                varchar(25)  NOT NULL,
Nationality         varchar(30)  NULL,
Birthdate           year         NULL,
DeceasedDate        year         NULL);

CREATE UNIQUE INDEX ARTIST_Name_IDX ON ARTIST(Name);

CREATE TABLE WORK(
WorkID              int          AUTO_INCREMENT PRIMARY KEY,
Description         text         NULL,
Title               varchar(25)  NOT NULL,
Copy                varchar(8)   NOT NULL,
ArtistID            int          NOT NULL);

CREATE UNIQUE INDEX WORK_ID_IDX ON WORK (Title, Copy, ArtistID);

CREATE TABLE CUSTOMER_ARTIST_INT(
ArtistID            int          NOT NULL,
CustomerID          int          NOT NULL);

ALTER TABLE CUSTOMER_ARTIST_INT
 ADD
 CONSTRAINT CUST_ARTIST_PK    PRIMARY KEY ( ArtistID, CustomerID );

CREATE TABLE TRANSACTION (
TransactionID       int          AUTO_INCREMENT    PRIMARY KEY,
DateAcquired        date         NOT NULL,
AcquisitionPrice    decimal(7,2)NULL,
PurchaseDate        date         NULL,
SalesPrice          decimal(7,2)NULL,
CustomerID          int          NULL,
WorkID              int          NOT NULL);
```

ferente desde otras conexiones de usuarios. Para comprender cómo solucionar con esto, primero considere el diccionario de datos MySQL.

MySQL conserva los metadatos en la base de datos *mysql*. Dos tablas de interés especial son *user* (*usuario)* y *db*. Para ver sus metadatos, abra la base de datos *mysql* y use la orden de descripción en *user* y *db*. Para ver una lista de usuarios y sus anfitriones, en el arranque de mysql introduzca:

```
Use mysql;
SELECT Host, User FROM user;
```

Para ver usuarios, anfitriones y sus bases de datos permitidas, introduzca:

```
SELECT Host, User, Db FROM db;
```

Los valores de Host (Anfitrión) son típicamente "localhost" o nombres del MySQL y otras computadoras. También pueden ser direcciones IP. Un valor del anfitrión de "%" significa que otro usuario se puede conectar desde cualquier lugar.

Ahora, suponga que los programas JDBC que se ejecutan en la misma máquina que MySQL serán considerados como derivados del anfitrión local. Esto no es verdad. El problema es que para MySQL, anfitrión local significa que se conecta mediante un socket, y las conexiones JDBC mediante TCP/IP. De esta manera, si se quiere conectar utilizando JDBC, el método más simple (pero menos seguro) de hacer esto es dar acceso a la base de datos a la cuenta que quiera de cualquier lugar. El siguiente GRANT hará esto:

```
GRANT ALL ON vr1.* TO dk1@"%" IDENTIFIED BY "sesame";
```

Esta instrucción transferirá todos los privilegios a todas las tablas en la base de datos vr1 para la cuenta del usuario nombrada dk1 con la contraseña sésamo. El comodín % indica que dk1 se puede conectar desde cualquier ubicación. Para mayor seguridad, podría reemplazar el comodín con una dirección específica IP.

Después de que haya ejecutado esta instrucción, consulte la tabla db de User, Host, Db, para permitir que haya una entrada para su usuario con el valor del anfitrión de %. Si es así, debería ser capaz de hacer contacto mediante JDBC para esa base de datos.

Control de concurrencia

MySQL tiene un apoyo limitado para el control de concurrencia. Como en la versión 3.0, no hay soporte para transacciones y, por lo tanto, no hay nivel de aislamiento de transacción. Tampoco es posible revertir transacciones. Las aplicaciones deben realizar su propia reversión, cuando sea necesario.

MySQL utiliza locks de lectura y escritura por tabla. Cuando se ejecuta una instrucción SELECT, MySQL obtiene locks de lectura y escritura en todas las tablas en la instrucción select. Este lock bloqueará a otras sesiones de escribir en cualquiera de esas tablas, pero no bloqueará a otras de leer. Cuando se ejecuta una instrucción INSERT, UPDATE, o DELETE, MySQL aplica locks de escritura en todas las tablas involucradas. Estos locks bloquearán a otras secciones de escribir o leer. El resultado de esta estrategia de bloqueo es que los datos consistentes se leen o se actualizan instrucción por instrucción de MySQL. Todos los datos de escritura o de lectura son necesarios.

Un programador puede trabajar en la falta de transacciones colocando las órdenes LOCK TABLES/UNLOCK TABLES (BLOQUEAR TABLAS/DESBLOQUEAR TABLAS) en las fronteras de la transacción. Así,

```
LOCK TABLES T1, T2, T3 WRITE;
UPDATE TABLE T1 SET Col1="xzy" WHERE Col1="abc";
```

```
UPDATE TABLE T2 . . .
UPDATE TABLE T3 . . .
 . . . hacer que otras transacciones funcionen en las tablas T1, T2, T3
UNLOCK TABLES;
```

conservará a otros usuarios de las tablas de leer y escribir T1, T2, T3 mientras se procesa la transacción. Todas las actualizaciones entre las instrucciones LOCK y UNLOCK son atómicas porque se procesan antes de que se libere cualquier lock.

Por desgracia, cuando se utilizan locks de esta manera, el acceso es cero para otros usuarios de las tablas con locks. Mientras que la transacción esté en proceso nadie puede leer ni escribir en las tablas T1, T2, T3. Durante el periodo del bloqueo MySQL será sólo un sistema usuario en las tablas. Esto parece ser un problema serio si las transacciones son de larga duración y si la carga de trabajo de la aplicación involucra una actualización sustancial.

Una sesión debe bloquear todas las tablas a la vez. Si quiere bloquear más, debe liberar el bloqueo que tiene y adquirir uno nuevo que incluya las tablas adicionales. De esta manera, ninguna sesión puede tener más de una instrucción LOCK TABLES abierta a la vez. Recuerde que en el capítulo 11 señalamos que esta estrategia elimina la posibilidad de un deadlock.

Las lecturas sucias pueden ser posibles o no, dependiendo de cómo estén escritas las aplicaciones. MySQL no revierte el trabajo, así que si ninguna aplicación realiza su propia reversión, entonces no son posibles las lecturas sucias. Si una aplicación realiza su propia lectura sucia, y si la aplicación y sus acciones de reversión se colocan entre las instrucciones LOCK TABLES y UNLOCK TABLES, entonces no son posibles las lecturas sucias. Sin embargo, como se mencionó, esta estrategia puede dar como resultado un rendimiento eficaz inaceptable. Finalmente, si una aplicación realiza reversiones, pero no utiliza las instrucciones LOCK y UNLOCK, o no aparece en el mismo conjunto de instrucciones LOCK y UNLOCK como las instrucciones que se revierten, entonces son posibles las lecturas sucias.

Respaldo y recuperación

MySQL proporciona facilidades limitadas de recuperación y respaldo, una utilidad para guardar las bases de datos, así como tablas individuales dentro de la base de datos. Sin embargo, en algunos casos es más rápida y fácil de usar que las órdenes de copiado del sistema operativo para guardar los archivos MySQL de la base de datos en medios de respaldo.

MySQL mantendrá un archivo de log de las acciones que ha procesado. Sin embargo, este log es de órdenes y trabajo, y no de imágenes anteriores y posteriores. Para recuperar una base de datos se hace una copia de respaldo de la versión más vieja y los comandos en el log se aplican nuevamente. Los cambios en bloque se registran como órdenes; sólo el nombre del archivo que se usó como fuente de los cambios de datos aparece en el log. Los cambios individuales no aparecen.

A propósito, si está recuperando una base de datos debido a un error, como por ejemplo una instrucción errónea tal como:

```
DROP TABLE CUSTOMER;
```

asegúrese de eliminar esta instrucción DROP del log antes de procesarlo nuevamente. De lo contrario, el administrador de registro procesará DROP TABLE y recuperará exactamente hasta donde estaba cuando empezó: sin la tabla CUSTOMER.

Resumen MySQL

Usted probablemente dirá que a MySQL le faltan muchas características y funciones de un producto moderno DBMS. Quizá le sorprenda por qué debe usarlo completo. Como

mencionamos, es gratis y también es open source; si quiere participar en un proyecto de open source en el dominio de DBMS, ésta es una buena oportunidad. También, MySQL es fácil de usar e incluso divertido. Las características y funciones que posee permiten hacer un buen trabajo. Según parece, la comunidad que está desarrollando MySQL sólo hace pocas cosas, pero las hace bien. Es un placer trabajar con este producto.

➤ RESUMEN

JDBC es una alternativa para ODBC y ADO que proporciona acceso a la base de datos para programas escritos en Java. Hay drivers JDBC para casi cada producto DBMS. Sun define cuatro tipos de drivers. Los drivers del tipo uno proporcionan un puente entre Java y ODBC. Los drivers de los tipos del dos al cuatro están escritos totalmente en Java. Los drivers del tipo dos dependen del producto DBMS para la comunicación entre las máquinas. Los drivers del tipo tres trasladan las llamadas JDBC en protocolo de red independiente DBMS. Los drivers del tipo cuatro trasladan las llamadas JDBC en protocolos de red dependientes del DBMS.

Un applet es un programa compilado bytecode en Java que se transmite al explorador mediante HTTP y se invoca utilizando el protocolo HTTP. Un servlet es un programa de Java que se invoca en el servidor para responder a las peticiones HTTP. Los drivers tipo tres y cuatro se pueden usar en servlets y applets. Los drivers del tipo dos se pueden utilizar sólo en servlets y únicamente si el DBMS y el servidor de la red no están en la misma máquina, o si el proveedor DBMS maneja la comunicación de las máquinas entre el servidor de la red y el de la base de datos.

Hay cuatro pasos cuando se utiliza JDBC: (1) cargar el driver; (2) establecer una conexión para la base de datos; (3) crear una instrucción, y (4) ejecutarla. Las bibliotecas de clases de drivers necesitan estar en la CLASSPATH para el compilador Java y para la máquina virtual Java. Están cargadas en un programa Java con el método de Class ForName. Se establece una conexión usando el método getConexión del DriverManager. Un string de conexión incluye la literal jdbc: seguida del nombre del driver y de un URL a la base de datos.

Los objetos Statement se crean utilizando el método CreateStatement de un objeto de conexión. Las instrucciones se pueden procesar con los métodos de un objeto de instrucción, executeQuery (ejecutarConsulta) y executeUpdate (ejecutarActualización). Los objetos ResultSetMetaData (Conjunto de resultados de metadatos) se crean utilizando el método getMetaData de un objeto ResultSet. Tanto las consultas compiladas como los procedimientos almacenados se pueden procesar mediante JDBC, utilizando PreparedStatement y objetos CallableStatement.

Las páginas del servidor Java (JSP) proporcionan un medio para crear páginas dinámicas de la red utilizando HTML (y XML) y Java. Las páginas JSP proporcionan capacidades de un lenguaje completamente orientado a objetos al programador de la página. VBScript o JavaScript no se pueden utilizar en una JSP. Las páginas JSP se compilan en una máquina de bytecode independiente.

Las páginas JSP se compilan como subclases de la clase Servlet HTTP. Consecuentemente, los fragmentos pequeños del código se pueden colocar en una página JSP, también en los programas completos de Java. Para usar las páginas JSP, el servidor de la red debe implementar el Servlet Java 2.1+ y las especificaciones de las Páginas del Servidor Java 1.0+. Apache Tomcat, un producto open source del Proyecto Jakarta, implementa estas aplicaciones. Tomcat puede trabajar en conjunto con Apache, u operar como un Servidor de la red autónomo para propósitos de prueba.

Cuando se utiliza Tomcat (o cualquier otro procesador JSP), los drivers JDBC y las páginas JSP deben ubicarse en directorios especificados. Cualquier Java bean que se utilice mediante la página JSP también se debe almacenar en directorios específicos. Cuando se requiere una página JSP, Tomcat permite que se utilice la más reciente. Si hay disponible una versión más nueva no compilada, Tomcat automáticamente la analiza y la compila. Hay un máximo de una página JSP en memoria a la vez, y la página JSP requiere que se ejecuten como una parte del procesador del servlet, y no como

un proceso por separado. El código Java en una JSP se puede cargar e invocar un Java bean compilado, si se desea.

MySQL es un DBMS open source que opera en Unix, Linux, y Windows. No hay pago de licencia. MySQL puede proporcionar un procesamiento rápido de consultas, pero no da soporte a vistas, procedimientos almacenados, o disparadores. Se puede definir la integridad referencial, pero no se impone mediante MySQL. Mantiene un diccionario de datos en una base nombrada *mysql*. El usuario y las tablas db pueden ser consultados para determinar los permisos del usuario. Para tener acceso a MySQL desde JDBC, se le debe otorgar acceso a la cuenta del cliente a la base de datos desde cualquier lugar o de direcciones TCP/IP que represente la computadora local.

MySQL proporciona soporte limitado para el procesamiento concurrente. No hay soporte para las transacciones y, por lo tanto, no hay instrucciones COMMIT (ejecutar) o ROLL BACK (revertir), ni aislamiento de transacción. MySQL bloquea por tabla. Los locks de lectura compartida se obtienen cuando se procesan las instrucciones SELECT y los locks exclusivos, cuando se escriben. El rendimiento eficaz puede ser un problema cuando se bloquea al nivel de la tabla. Los usuarios pueden delimitar la lógica de la transacción con las instrucciones LOCK TABLES (BLOQUEAR TABLAS) y UNLOCK TABLES (DESBLOQUEAR TABLAS). El deadlock se evita permitiendo que cuando menos una instrucción de bloqueo se abra a la vez. Las lecturas sucias son posibles si algunas aplicaciones revierten su propio trabajo y no rodean su actividad con bloqueos de tablas.

MySQL proporciona facilidades limitadas de recuperación y respaldo. Hay una facilidad de respaldo que aumenta las de copia del sistema operativo. MySQL mantiene un log de instrucciones procesadas. El log no incluye las imágenes antes y después, ni valores de datos de actualizaciones o eliminaciones en volumen. Aunque aún tiene muchas limitaciones, MySQL es fácil de usar y sus características y funciones están bien implementadas.

➤ PREGUNTAS DEL GRUPO I

16.1 ¿Cuál es el requisito principal para utilizar JDBC?

16.2 ¿Qué significa JDBC?

16.3 ¿Cuáles son los cuatro tipos de drivers JDBC?

16.4 Explique la finalidad de los drivers del tipo uno.

16.5 Explique la finalidad de los drivers JDBC del tipo dos al cuatro.

16.6 Defina *applet* y *servlet*.

16.7 Explique cómo realiza Java la portabilidad.

16.8 Enumere los cuatro pasos en la utilización de un driver JDBC.

16.9 Muestre la función Java para cargar los drivers mm.mysql que se utilizaron en este capítulo.

16.10 Muestre la función Java para la conexión de una base de datos utilizando los drivers mm.myqsl. Suponga que la base de datos se llama *DatosCliente,* el usuario es *Lew* y la contraseña es *Secreto*.

16.11 Muestre la herramienta de Java para crear una función objeto.

16.12 Muestre la función Java para la creación de un objeto ResultSet que despliegue Name y Nationality de la tabla ARTIST, utilizando un objeto instrucción ya creado llamado s.

16.13 Muestre una función Java para repetir el Result Set creado en la pregunta 16.12.

16.14 Muestre la función Java para ejecutar una actualización y cambiar la Nacionalidad de un artista llamado “Jones” a “French”. Utilice un objeto Statement ya creado llamado s.

16.15 En la pregunta 16.14, ¿cómo puede determinar si se logró la actualización?

16.16 Muestre una función Java para la creación de un objeto referenciando metadatos para el ResultSet creado en la pregunta 16.12.

16.17 Muestre la función Java necesaria para invocar un procedimiento almacenado llamado Customer_Delete. Suponga que el procedimiento tiene tres parámetros de texto con valores de nombre cliente, código de área y número telefónico. Pase los valores a 'Mary Orange', '206' y '555-1234' a este procedimiento.

16.18 ¿Cuál es la finalidad de las páginas del Servidor Java?

16.19 Describa las diferencias entre las páginas ASP y JSP.

16.20 Explique cómo es que las páginas ASP son portátiles.

16.21 ¿Cómo es posible que los segmentos pequeños de Java se puedan codificar en páginas JSP? ¿Por qué no se requieren completos los programas Java?

16.22 ¿Cuál es la finalidad de Tomcat?

16.23 Con la instalación estándar de Tomcat, ¿qué acciones se deben realizar antes de utilizar las páginas JSP que cargan las clases JDBC?

16.24 Cuándo se agregan nuevas bibliotecas de clases para el uso de Tomcat, ¿qué se debe hacer para colocar la biblioteca en las CLASSPATH de Tomcat?

16.25 Describa el proceso mediante el cual se compilan y ejecutan las páginas JSP. ¿Un usuario siempre puede ingresar a una página obsoleta? ¿Por qué? Explique.

16.26 ¿Por qué son preferibles los programas JSP a los programas CGI?

16.27 ¿Qué condiciones se requieren para que una clase Java sea un bean?

16.28 Muestre la directiva jsp para acceder a un bean llamado CustomerDeleteBean. Dé a este bean la identidad de *custdel*.

16.29 Muestre las directivas JSP para colocar una propiedad bean llamada *Prop1* al valor de un parámetro de forma llamado *Param1*.

16.30 ¿Por qué es conveniente darles los mismos nombres a propiedades de objeto y forma de parámetros? Muestre una directiva JSP para asociar parámetros y propiedades cuando sea el caso.

16.31 ¿Cuál es la diferencia entre invocar un bean de un programa puro de Java e invocar un bean del código de Java en una JSP?

16.32 ¿Bajo qué condiciones elegiría utilizar MySQL?

16.33 ¿Para qué tipo de carga intensa de trabajo es MySQL?

16.34 Liste las principales limitaciones de MySQL.

16.35 ¿Cómo procesa MySQL 3.0 las restricciones de la integridad referencial?

16.36 ¿Qué instrucción emplearía para crear una nueva tabla utilizando MySQL?

16.37 ¿Qué problemas se enfrentan cuando se hace una conexión a MySQL utilizando JDBC?

16.38 Muestre la instrucción MySQL para dar al usuario *Lew* permiso para ingresar a cualquier tabla de CustomerData en la base de datos. Suponga que la contraseña es *Secret*.

16.39 Describa los dispositivos de la administración de transacción en MySQL 3.0.

16.40 ¿Cómo usa MySQL los locks de lectura?

16.41 ¿Cómo usa MySQL los locks de escritura?

16.42 ¿En qué nivel invoca MySQL los locks? ¿Cuáles son las ventajas y desventajas de esto?

16.43 Muestre cómo una aplicación podría proporcionar una atomicidad de transacción utilizando LOCK TABLES y UNLOCK TABLES.

16.44 ¿Cuál es la desventaja de la estrategia utilizada en su respuesta a la pregunta 16.43?

16.45 ¿Por qué no es posible el deadlock con MySQL?

16.46 ¿Bajo qué condiciones son posibles las lecturas sucias con MySQL?

16.47 Describa las facilidades para respaldar de MySQL.

16.48 ¿Cuáles son las limitaciones de log en MySQL?

16.49 De acuerdo con el autor, ¿por qué alguien elegiría utilizar MySQL?

➤ PREGUNTAS DEL GRUPO II

16.50 Compare y contraste ASP con JSP. Describa las fuerzas y debilidades de cada uno. ¿Bajo qué circunstancias recomendaría a uno en lugar del otro? ¿Qué importancia tiene la portabilidad para los servidores de la red? ¿Cuál es la desventaja de depender de Microsoft? Algunas personas dicen que prefieren uno en lugar de otro, es más un asunto de preferencia personal y de valores que otra cosa. ¿Está usted de acuerdo?

16.51 Reescriba el Java bean mostrado en la figura 16-8 para usar excepciones antes de que el *result (resultado)* regrese el parámetro. Modifique la página JSP para procesar correctamente este bean. ¿De qué manera es mejor su bean que el de la figura 16-8?

➤ PROYECTOS

A. Escriba un programa Java para utilizar MySQL y los drivers mm.mysql que se emplearon en este capítulo. Su programa deberá implementar la lógica del procedimiento almacenado CustomerInsertWithTransaction descrito para Oracle en el capítulo 12 y el SQL Server en el capítulo 13. Agregue lógica a su programa para desplegar los mismos resultados que están expuestos en la CustomerPurchasesView. Ejecute su programa como un programa autónomo.

B. Convierta el programa que escribió en el Proyecto A a un Java bean. Escriba una página JSP para invocar su bean.

C. Obtenga un drive JDBC Oracle y escriba un programa Java para conectarlo a la versión Oracle de la base de datos de View Ridge y despliegue los contenidos de cualquier tabla en la base de datos. Escriba un programa en Java para solicitar el procedimiento almacenado CustomerInsert. Utilice un objeto CallableStatement JDBC para invocar ese procedimiento.

D. Convierta el programa que escribió en el Proyecto C a un Java bean. Escriba una página JSP para invocar su bean.

E. Obtenga un driver del servidor JDBC SQL y escriba un programa en Java para conectarlo a la versión SQL de la base de datos de View Ridge y despliegue los contenidos de cualquier tabla en la base de datos. Escriba un programa en Java para solicitar el procedimiento almacenado CustomerInsert. Utilice un objeto CallableStatement para invocar ese procedimiento.

F. Convierta el programa que escribió en el Proyecto E a un Java bean. Escriba una página JSP para invocarlo.

➤ PREGUNTAS DEL PROYECTO FIREDUP

Cree la base de datos FiredUp utilizando MySQL u Oracle. Si está en MySQL no podrá crear las restricciones de integridad referencial; de lo contrario, siga las instrucciones al final de los capítulos 12 o 13.

A. Codifique una página JSP para desplegar la tabla STOVE (ESTUFA).

B. Codifique una página JSP para desplegar cualquier tabla en la base de datos de Fired-Up. Utilice la figura 16-3 como ejemplo.

C. Codifique una página JSP para introducir los datos nuevos de STOVE. Utilice un Java bean para la inserción; siga las figuras 16-8 y 16-9 como ejemplo.

D. Codifique una página JSP para permitir que los clientes registren sus estufas.

E. Cree un Java bean para introducir los nuevos datos de reparación de estufa.

F. Codifique una página JSP para invocar al Java creado en la pregunta E del proyecto. Utilice las figuras 16-8 y 16-9 como ejemplo.

CAPÍTULO 17

Distribución de datos de la Empresa

Hasta ahora hemos descrito el procesamiento de la base de datos en el contexto de la computadora personal y de la tecnología de Internet, utilizando las arquitecturas de tres capas y multicapas (o *n* capas). Las empresas también utilizan otros tipos de sistemas diferentes, más antiguos. Debido a que usted se puede encontrar con ellos, en la primera parte del presente capítulo estudiaremos las características de esos tres tipos. Un cuarto tipo, el procesamiento de la base de datos distribuida, se está empezando a utilizar para el procesamiento de la base de datos comercial. Tanto Oracle como el SQL Server proporcionan el soporte para ello, así que también los analizaremos.

Los datos son un activo importante de las empresas, un activo que se puede utilizar no sólo para facilitar las operaciones de una compañía, sino también para la organización, planeación, previsión, análisis de estrategias y tareas similares. Por desgracia, aun cuando muchas organizaciones saben que efectivamente sus bases de datos dan soporte a operaciones organizacionales, están conscientes de que sus bases de datos no son utilizadas eficazmente para el análisis, la planeación, y otros aspectos administrativos. En este capítulo abordamos temas que son importantes para incrementar la recuperación de la inversión que ha hecho la empresa en bases de datos: datos centralizados cargados, OLAP, almacenamiento de datos y administración de datos.

➤ ARQUITECTURAS DEL PROCESAMIENTO DE LA BASE DE DATOS EMPRESARIAL

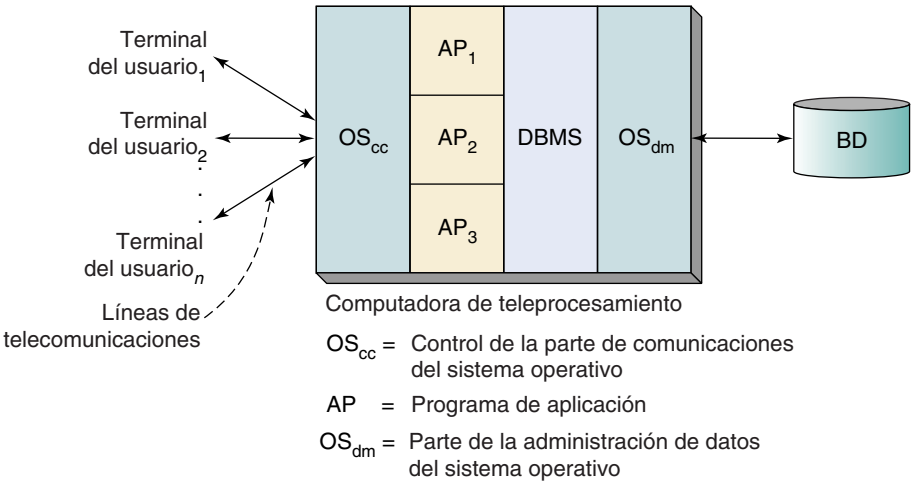
Se usan varios sistemas diferentes de arquitecturas para el procesamiento de la base de datos de la empresa. En el pasado, los sistemas de teleprocesamiento eran los más usados; pero a medida que las microcomputadoras se hicieron más comunes en los escritorios y más poderosas que los servidores de datos, surgieron nuevas arquitecturas de bases de datos multiusuario. En esta sección introducimos el teleprocesamiento, cliente-servidor, archivos compartidos y alternativas distribuidas.

Sistemas de teleprocesamiento

El método clásico para dar apoyo a un sistema de bases de datos multiusuario es el teleprocesamiento, que utiliza una computadora y un CPU. Todo el procesamiento se realiza con una sola computadora.

FIGURA 17-1

Relaciones de programas en un sistema de teleprocesamiento



La figura 17-1 muestra un sistema típico de teleprocesamiento. Los usuarios operan terminales tontas (o microcomputadoras que emulan a las terminales tontas) que transmiten los mensajes y los datos de una transacción a la computadora central. La parte del control de las comunicaciones del sistema operativo recibe los mensajes y los datos y los envía al programa de aplicación apropiado. Después, el programa llama al DBMS para sus servicios y el DBMS utiliza parte de la administración de datos del sistema operativo para procesar la base de datos. Cuando se termina una transacción, los resultados se regresan a los usuarios en las terminales tontas mediante la parte de control de las comunicaciones del sistema operativo.

La figura 17-1 muestra *n* usuarios presentando transacciones procesadas por los tres programas de aplicación diferentes. Debido a que hay un poco de inteligencia en las terminales de los usuarios (esto es, las *terminales* son tontas), todas las instrucciones para formatear la pantalla las debe generar la CPU y se transmiten sobre las líneas de comunicaciones. Esto significa que la interfaz de los usuarios por lo general es orientada a caracteres y primitiva. A los sistemas como éste se les llama sistemas de teleprocesamiento, debido a que todas las entradas y salidas se comunican a distancia (*tele* significa distancia) con la computadora central para el procesamiento.

Históricamente, los sistemas de teleprocesamiento fueron la alternativa más común para las bases de datos de sistemas de multiusuarios. Pero conforme ha disminuido la razón precio-desempeño de las computadoras, y en particular con el advenimiento de la computadora personal, otras alternativas que usan computadoras múltiples los han suplantado.

Sistemas cliente-servidor

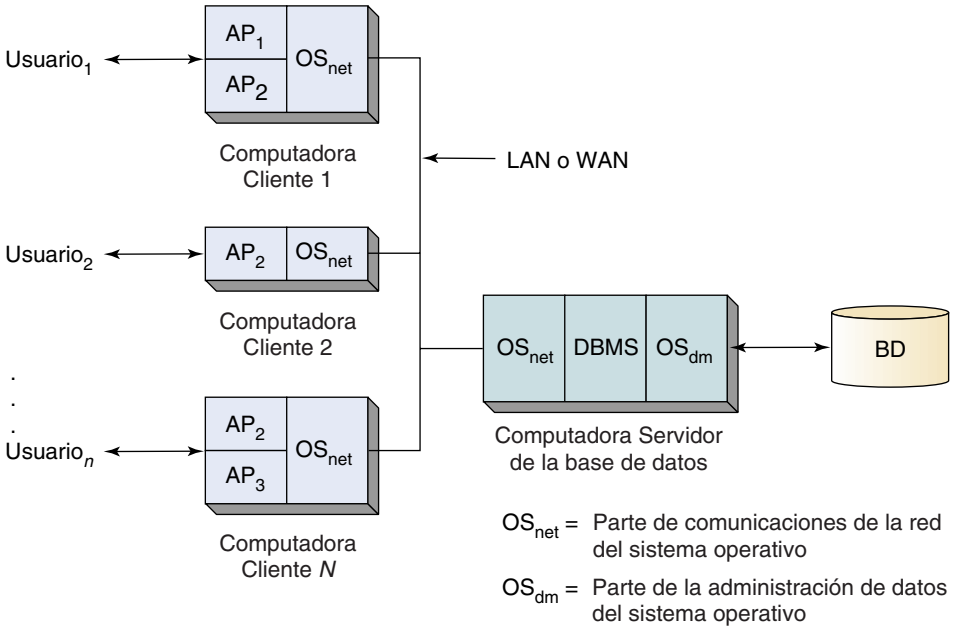
La figura 17-2 es un esquema de una de estas alternativas, llamado un **sistema cliente-servidor.** A diferencia del teleprocesamiento, que implica a una sola computadora, la computación cliente-servidor implica múltiples computadoras conectadas a una red. Algunas procesan los programas de aplicación y están designadas como *clientes*. Otra computadora procesa la base de datos y se denomina *servidor.*

La figura 17-2 muestra un ejemplo en el que cada uno de los *n* usuarios tiene su propia computadora cliente (procesamiento de aplicación): el Usuario_1 procesa AP_1 y AP_2 en la Computadora 1. El Usuario_2 procesa AP_2 en la Computadora 2, y el Usuario_n procesa AP_2 y AP_3 en la Computadora *N*. La otra computadora es el servidor de la base de datos.

Hay muchas opciones, considerando el tipo de computadora. Teóricamente, las computadoras cliente pueden ser macrocomputadoras o microcomputadoras. Sin embargo, debido al costo, en casi todos los casos las computadoras del cliente son microcomputadoras. De igual manera, cualquier tipo de computadora puede ser el servidor pero, nuevamente, debido al costo, el servidor es con más frecuencia una microcomputadora. Los clientes y el servidor están conectados utilizando una red de área local (LAN) o una red de área amplia (WAN).

FIGURA 17-2

Arquitectura cliente-servidor



Aunque es poco común que las computadoras cliente sean algo más que micros, algunas veces el servidor es una macrocomputadora, especialmente cuando se requiere que el servidor tenga una gran potencia. Por razones de seguridad y control no conviene ubicar la base de datos en una microcomputadora.

El sistema de la figura 17-2 tiene un solo servidor, aunque no siempre es el caso. Los servidores múltiples pueden procesar bases de datos diferentes o proporcionar otros servicios en favor de los clientes. Por ejemplo, en una empresa consultora de ingeniería, un servidor puede procesar la base de datos mientras un segundo servidor da apoyo a las aplicaciones de diseño asistidas por computadora.

Si hay múltiples servidores de procesamiento de bases de datos, cada uno debe procesar una base de datos diferente con el fin de que el sistema sea considerado un sistema cliente-servidor. Cuando dos servidores procesan la misma base de datos, el sistema ya no se llama cliente-servidor, sino sistema de base de datos distribuida.

Sistemas de archivos compartidos

En la figura 17-3 se muestra una segunda arquitectura de usuarios múltiples. Esta arquitectura, llamada de **archivos-compartidos**, no solamente distribuye los programas de aplicaciones a las computadoras de los usuarios, sino también al DBMS. En este caso, el *servidor* es un servidor de archivos y no uno de base de datos. Casi todos los sistemas de archivos compartidos emplean LANS de microcomputadoras.

La arquitectura de archivos compartidos fue desarrollada antes de la arquitectura cliente-servidor, y es en muchas formas más primitiva. Con los archivos compartidos, el DBMS en cada computadora de un usuario envía peticiones a la parte administradora de datos del sistema operativo en el servidor de archivos para el procesamiento de nivel de archivos. Esto significa que demasiado tráfico cruza la LAN, en comparación con la arquitectura cliente-servidor.

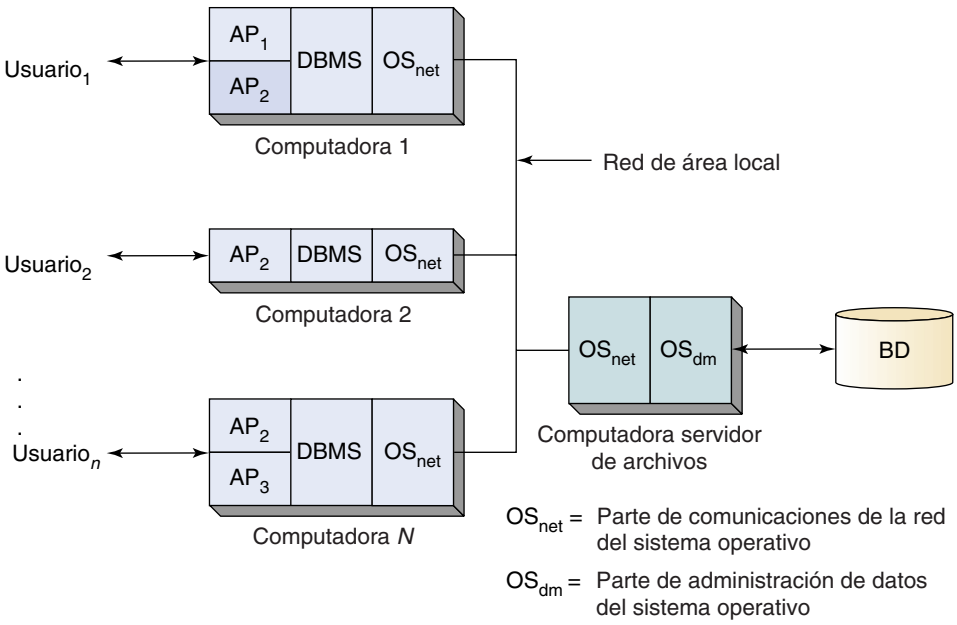
Por ejemplo, piense en el procesamiento de una consulta para obtener el Nombre y Dirección de todos los renglones en la tabla CLIENTE, donde CódigoPostal es igual a 98033. En un sistema cliente-servidor el programa de aplicación enviaría la siguiente instrucción SQL:

```
SELECT    NOMBRE, DIRECCIÓN
FROM      CLIENTE
WWHERE    CódigoPostal= 98033
```

El servidor respondería con todos los Nombres y Direcciones que califican.

FIGURA 17-3

Arquitectura de archivos compartidos



En un sistema de archivos compartidos el DBMS está en la computadora local y así el programa en el servidor puede procesar la instrucción SQL. Todos estos procesamientos se deben hacer en la computadora del usuario, por lo que el DBMS debe consultar al servidor de archivos para transmitir la tabla completa CLIENTE. Si esa tabla tiene índices u otros datos generales asociados, las estructuras de datos generales también deben transmitirse. Obviamente, con los archivos compartidos se necesita transmitir mucho más datos por toda la LAN.

Debido a estos problemas, los sistemas de archivos compartidos se utilizan pocas veces para el procesamiento orientado a la transacción de grandes volúmenes. Cuando hay demasiados datos se necesitan bloquear y transmitir únicamente para cada transacción, y esta arquitectura dará como resultado un desempeño muy lento. Sin embargo, hay una aplicación de bases de datos para la cual tiene sentido esta arquitectura: el procesamiento de consulta de datos extraídos y cargados en la computadora. Si uno o más usuarios necesitan tener acceso a grandes partes de la base de datos con el fin de producir reportes o contestar consultas, puede tener sentido tener un servidor que descargue grandes secciones de datos. En este caso los datos que se descargaron no están actualizados y no regresaron a la base de datos. Posteriormente en este capítulo mostraremos ejemplos del procesamiento de extracción de datos.

Los sistemas de archivos compartidos también se utilizan para aplicaciones que no usan bases de datos, tales como las que requieren discos grandes, rápidos y de gran almacenamiento para archivos voluminosos, tales como gráficas, audio y animaciones. También se utilizan para compartir impresoras costosas, plotters (graficadoras) y otro equipo periférico.

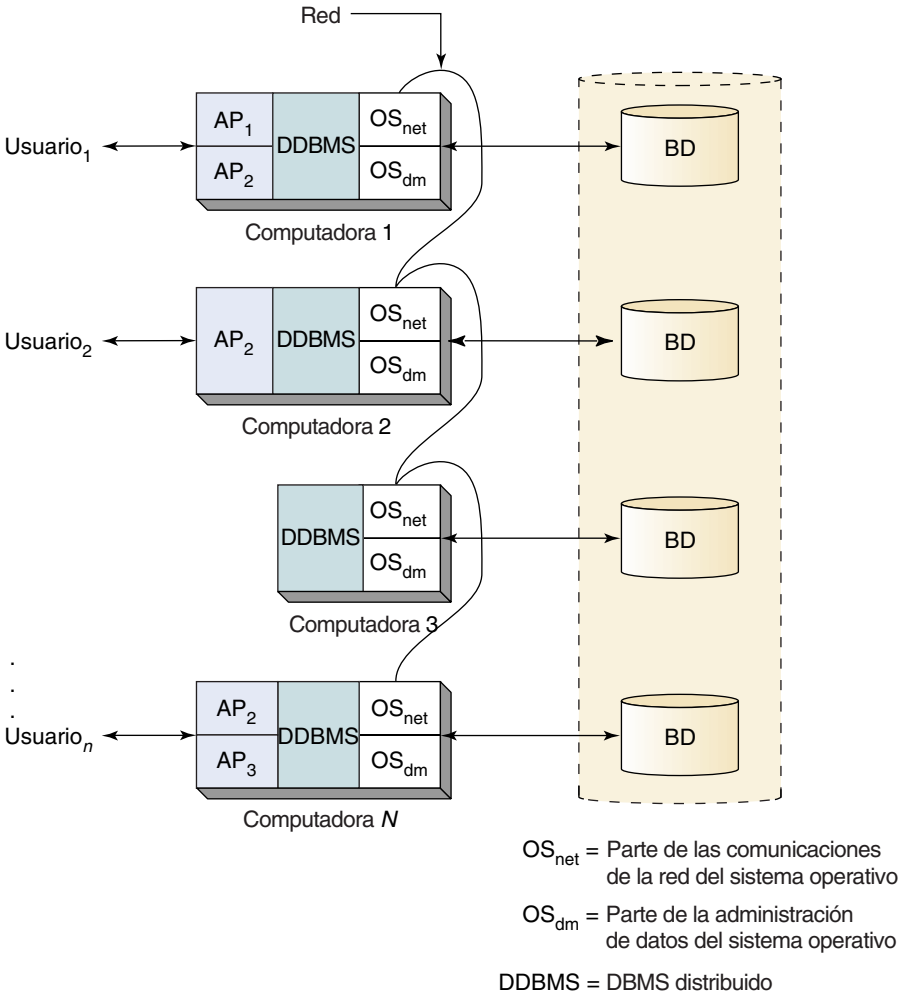
Sistemas de bases de datos distribuidas

En la figura 17-4 se muestra una cuarta alternativa, la de un sistema de bases de datos distribuidas, en el que la base de datos está distribuida por sí misma. En la figura 17-4 la base de datos (o parte de ésta) se almacena en todas las *N* computadoras. Como se muestra, las Computadoras 1, 2 y *N* procesan las aplicaciones y la base de datos, y la Computadora 3 procesa solamente la base de datos.

En la figura 17-4 la línea punteada alrededor de los archivos indica que la base de datos está compuesta de todos los segmentos de la base de datos en todas las *N* computadoras. Éstas se pueden ubicar físicamente en el mismo lugar, en diferentes partes del mundo, o en algún lugar intermedio.

➤ FIGURA 17-4

Arquitectura de la base de datos distribuida



PROCESAMIENTO DISTRIBUIDO CONTRA PROCESAMIENTO DE BASE DE DATOS DISTRIBUIDAS. Observe nuevamente las figuras 17-1, 17-2, 17-3 y 17-4. Las alternativas de compartir archivos, cliente-servidor y las bases de datos distribuidas difieren considerablemente del teleprocesamiento: todas utilizan múltiples computadoras para el proceso de aplicaciones o de DBMS. Por lo tanto, la mayoría de las personas dirían que tres de estas arquitecturas son ejemplos de **sistemas distribuidos,** porque el procesamiento de aplicaciones ha sido repartido entre varias computadoras.

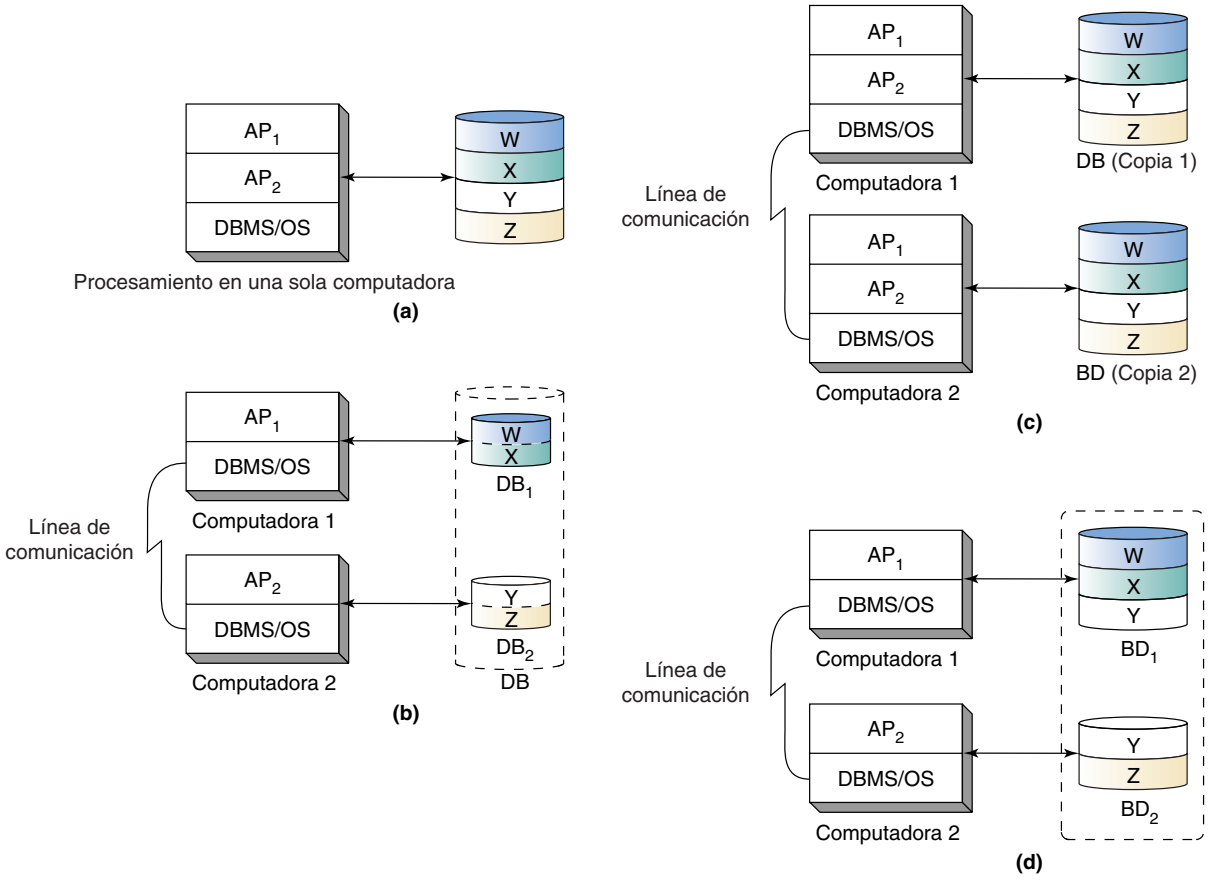
Sin embargo, observe, que la base de datos se distribuye por sí misma sólo en la arquitectura que se muestra en la figura 17-4. Ni el cliente-servidor ni las arquitecturas cliente–servidor o de archivos compartidos distribuyen la base de datos a computadoras múltiples. Por ende, la mayoría de las personas *no* se referirían a las arquitecturas de archivos-compartidos o de cliente-servidor como **sistemas de base de datos distribuidos.**

TIPOS DE BASES DE DATOS DISTRIBUIDAS. Hay varios tipos de sistemas de bases de datos distribuidas. Primero, observe la figura 17-5(a), la cual muestra una base de datos no distribuida con cuatro partes, W, X, Y y Z. Las cuatro piezas de estos segmentos se localizan en una sola base de datos y no hay duplicación de datos.

Ahora considere las alternativas distribuidas de la figura 17-5(b) a la (d). La figura 17-5(b) muestra la primera alternativa de distribución, en la que la base de datos se ha distribuido a dos computadoras; las piezas W y X se almacenaron en la Computadora 1

➤ FIGURA 17-5

Tipos de bases de datos distribuidas: (a) no particionada, alternativa sin réplica; (b) particionada, alternativa sin réplica; (c) no particionada, alternativa replicada; (d) particionada, alternativa replicada



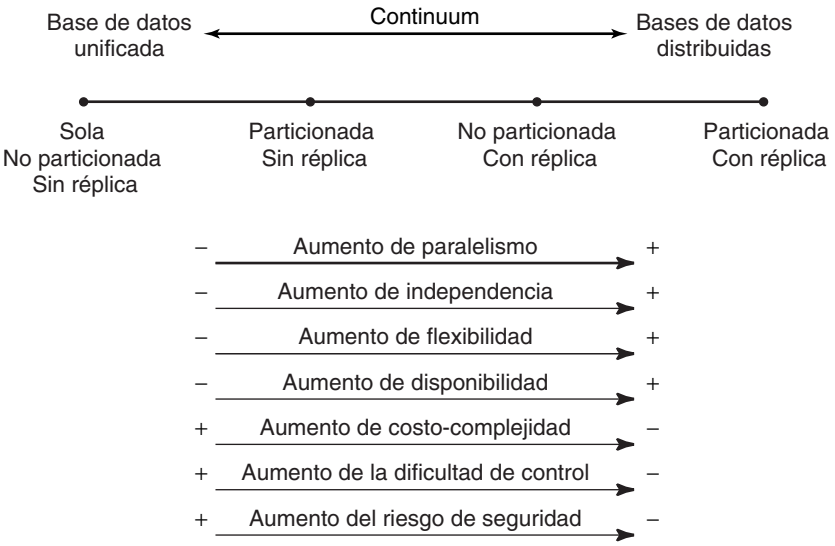
y las piezas Y y Z están almacenadas en la Computadora 2. En la figura 17-5(C), la base de datos completa ha sido duplicada en dos computadoras. Finalmente, en la figura 17-5(d), la base de datos ha sido particionada y una parte (Y) tiene réplica.

Algunas veces se utilizan dos términos con respecto a la partición de las bases de datos. Una partición vertical o **fragmento vertical** se refiere a una tabla que se parte en dos o más conjuntos de columnas. Así, una tabla R(C1, C2, C3, C4) podría dividirse en dos partes verticales de P1(C1, C2) y P2 (C3, C4). Dependiendo de la aplicación y de la razón para crear particiones, la llave de R comúnmente sería colocada también en P2 para formar P2(C1, C3, C4). Una partición horizontal, o **fragmento horizontal,** se refiere a los renglones de una tabla cuando están divididas en piezas. De esta manera, en la relación R, si los primeros 1000 renglones se colocan en R1(C1, C2, C3, C4) y los renglones restantes en R2(C1, C2, C3, C4), dará como resultado dos particiones horizontales, y ese resultado a veces se llama partición mixta.

COMPARACIÓN DE ALTERNATIVAS DE BASE DE DATOS DISTRIBUIDA. Estas alternativas se resumen en un *continuum* en la figura 17-6, arreglado en orden ascendente de acuerdo a la distribución, de izquierda a derecha. La base de datos no distribuida está en el extremo izquierdo del *continuum* y la base de datos particionada y replicada está en el extremo derecho. Entre estos extremos está una base de datos particionada. Las particiones están localizadas en dos o más computadoras y la base de da-

➤ FIGURA 17-6

Continua de alternativas de distribución de bases de datos



tos que no está particionada, pero cada entrada de base de datos se duplica en dos o más computadoras.

Las características de las alternativas en este *continuum* se enumeran en la figura 17-6. Las alternativas hacia la derecha aumentan el paralelismo, independencia, flexibilidad y disponibilidad, pero también significan más costos, complejidad, dificultad de control y riesgo para la seguridad.

Una de estas ventajas es particularmente importante para las empresas. Las alternativas a la derecha de la figura 17-6 proporcionan mayor flexibilidad y, por lo tanto, se pueden adaptar mejor a la estructura empresarial y al proceso organizacional. Por ejemplo, en una compañía de fabricación altamente descentralizada, en la que los administradores de la planta tienen un amplio rango de aplicabilidad en su planeación, nunca estarán satisfechos con un sistema de información organizacional con la estructura de la figura 17-5(a), ya que la estructura de la arquitectura del sistema de información y la estructura de la compañía se contraponen entre sí. Las alternativas del lado derecho proporcionan un mejor y más apropiado ajuste a esa organización que las de la izquierda.

La gran desventaja es la dificultad de control y el resultado de pérdida de potencial de la integridad de los datos. Considere la arquitectura de la base de datos en la figura 17-5(d). Un usuario conectado a la Computadora 1 puede leer y actualizar un elemento de datos en la partición Y en la Computadora 1, al mismo tiempo que un usuario diferente conectado a la Computadora 2 puede leer y actualizar ese elemento de datos en la partición Y de la Computadora 2.

TÉCNICAS DE PROCESAMIENTO DISTRIBUIDO. Los vendedores de DBMS proporcionan varias técnicas para dar soporte al procesamiento distribuido. Oracle y el SQL Server proporcionan tipos de apoyo similares, pero usan diferentes nombres para las mismas cosas, y un mismo nombre para cosas diferentes. Otros vendedores de DBMS incluso usan otra terminología. Aquí nos concentraremos en las ideas básicas.

El tipo más simple de procesamiento de base de datos distribuida en la descarga de datos de sólo lectura. En este caso, únicamente una computadora actualiza cualquiera de los datos de la base, pero múltiples computadoras (aunque pueden ser miles) están enviando copias para procesar en el modo de sólo lectura. Oracle llama a estas copias de sólo lectura **vistas materializadas.** El SQL Server llama a estas copias de sólo-lectura **snapshots.** Abordaremos este tipo de procesamiento de distribución en la parte VII.

Una técnica más compleja para el procesamiento distribuido es permitir que peticiones de actualización de datos se originen en múltiples computadoras, pero transmitir dichos requerimientos a una computadora designada para el procesamiento. Por ejemplo, la Computadora A podría ser designada como el único sitio en que se pueda actualizar la tabla EMPLEADO (y vistas basadas en EMPLEADO), mientras que la Computadora B se puede designar como el único sitio que pueda actualizar la tabla CLIENTE (y vistas). De vez en cuando, las actualizaciones se deben transmitir de regreso a todas las computadoras distribuidas en la red y a las bases de datos sincronizadas.

La alternativa más complicada es permitir actualizaciones múltiples de los mismos datos, en varios sitios. En este caso, pueden ocurrir tres tipos de problemas de **conflicto de actualización distribuida.** En uno, la unicidad se puede violar. En la vista de la Galería View Ridge dos sitios diferentes pueden crear un renglón TRABAJO con igual Copia, Título y ArtistaID. Otro posible problema es similar al de actualización perdida. Dos sitios pueden actualizar el mismo renglón. Un tercer problema ocurre cuando un sitio actualiza un renglón que ha sido eliminado en un segundo sitio.

Para resolver las actualizaciones en conflicto se designa un solo sitio para la resolución de conflictos. Todas las actualizaciones se examinan mediante éste y los conflictos de actualizaciones se resuelven utilizando la lógica incluida en el DBMS, o con un código de aplicación que es similar a lo que se escribió para los triggers. En el caso más extremo, los conflictos se escriben en un registro y se resuelven manualmente. Esta última alternativa no se recomienda, porque muchos renglones en las bases de datos operacionales se pueden dejar en suspenso hasta que se resuelvan las resoluciones. Lo anterior puede reducir de manera inaceptable el rendimiento efectivo.

Ninguna de estas técnicas se enfoca al problema de proporcionar transacciones atómicas cuando se distribuyen las bases de datos. Esto es especialmente problemático cuando pueden ocurrir los conflictos de actualización. ¿En qué punto se confirma una acción en una base de datos? Si las actualizaciones pueden ser revertidas durante la resolución de la actualización distribuida, entonces no se puede confirmar la transacción distribuida durante horas. Este retraso obviamente no es aceptable.

Haciendo a un lado el asunto del conflicto de la actualización distribuida, la coordinación de las transacciones distribuidas es difícil. Para ser atómica, una acción no actualizada en una transacción distribuida se puede confirmar hasta que todas lo estén. Esto significa que un sitio determinado debe confirmar provisionalmente sus actualizaciones, dependiendo de la notificación del administrador de la transacción distribuida de que todos los sitios han confirmado. Para este propósito, se utiliza un algoritmo llamado **commit de dos fases.** Microsoft y el servicio OLE la denominan **Servicio de distribución de transacción (DTS, por sus siglas en inglés)** al que implementa el commit de dos fases. En el mundo Java, los **Java beans empresariales (EJB, por sus siglas en inglés)** se utilizan para este fin. Ambos están más allá del enfoque de este texto, pero es bueno que usted conozca su existencia. Busque la palabra llave *replication* en la documentación del SQL Server y en Oracle.

➤ DESCARGA DE DATOS

Con el advenimiento de las poderosas computadoras personales es factible cargar grandes cantidades de datos empresariales para las computadoras departamentales, así como datos de usuarios para el procesamiento local. Los usuarios pueden consultar estos datos utilizando productos del DBMS local y también pueden importar datos de hojas de cálculo electrónicas, programas de análisis financiero, programas gráficos, y otras herramientas que le resultan familiares.

En general, los datos descargados se pueden utilizar únicamente para consultar y con fines de reporte. No se pueden actualizar porque una vez que se eliminan los datos de la base de datos operacional dejan de ser temas del control de concurrencia. Para comprender más acerca del procesamiento de cargar datos piense en un escenario típico.

Equipo Universal

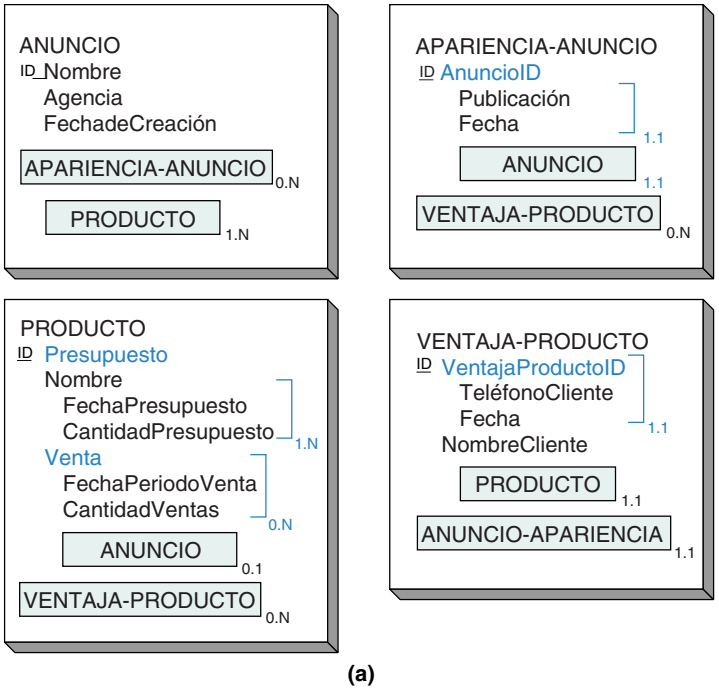
La Compañía Equipo Universal fabrica y vende equipo pesado para la industria de la construcción. Sus productos incluyen excavadoras, graduadores, cargadoras y torres de perforación. Cada producto se asigna a un administrador de productos en el departamento de mercadotecnia, el cual es responsable de la planeación del producto, publicidad, apoyo de la mercadotecnia, desarrollo de material de soporte de ventas, etc. Cada representante de producto se asigna a un grupo de dos o tres productos relacionados.

La publicidad es el punto de mayor presupuesto de la administración del producto, así que los administradores desean medir la efectividad de los anuncios, los cuales siempre tienen una tarjeta de correo para que el destinatario solicite información. Dichas tarjetas tienen un número único preimpreso por cada forma de anuncio, así que dicho número se puede utilizar para identificar el anuncio que generó una ventaja en particular. Para facilitar el registro, el departamento de mercadotecnia ha desarrollado una aplicación de base de datos en una microcomputadora para que la usen los administradores del producto.

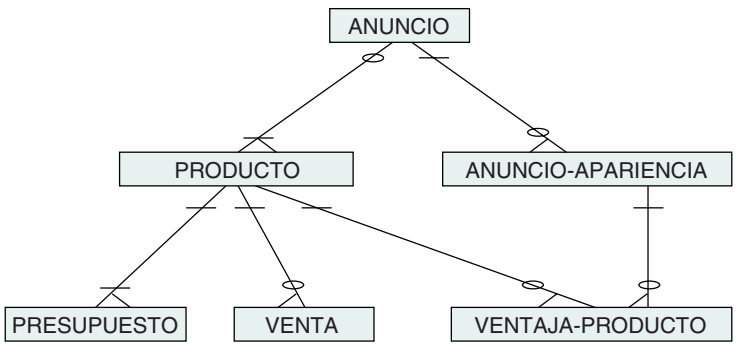
La figura 17-7(a) muestra los objetos semánticos procesados mediante esta aplicación. ANUNCIO es un anuncio. La APARIENCIA-ANUNCIO es la ocurrencia de un anuncio en particular; PRODUCTO representa uno en particular, por ejemplo una excavadora, y PRODUCTO contiene dos grupos repetidos, uno en presupuesto y uno en ventas. Los grupos son de valores múltiples (o multivaluados) porque los presupuestos

➤ FIGURA 17-7

Objetos y relaciones que sustentan la base de datos de mercadotecnia de los productos Universal: (a) objetos procesados por los administradores de los productos Universal, y (b) Estructura relacional que soporta esos objetos



(a)



(b)

de ventas se asignan para cada trimestre, y las ventas del producto se registran semanalmente.

Esta vista de PRODUCTO es totalmente sencilla. El objeto completo PRODUCTO en realidad contiene más atributos; pero debido a que no se necesitan los otros datos para la aplicación del administrador del producto, los hemos omitido. En la figura 17-7(b) se muestra la estructura de la base de datos que maneja estos objetos.

Proceso de descarga

A los administradores de producto se les asigna una computadora personal conectada a otras PC a través de una LAN del departamento de mercadotecnia de Universal. Para obtener ventas y datos sobre las ventajas del producto, las computadoras llaman a un archivo del servidor que sirve como acceso a la macrocomputadora de Universal (computadora de procesamiento-transacción). La arquitectura es similar a la que se muestra en la figura 17-8.

Cada lunes un usuario clave en el departamento de mercadotecnia ejecuta un programa que desarrolló el departamento MIS de Universal, el cual actualiza las tablas VENTAS, PRESUPUESTO y VENTAJA-PRODUCTO en las tablas de la base de datos del archivo del servidor, con datos de la base de datos de la macrocomputadora. Este programa agrega a la base los datos de la semana anterior y también hace correcciones. Los datos del producto y de las ventas se importan de todos los productos relacionados para apoyar la elaboración de estudios comparativos de los administradores. Una vez que los datos se han cargado al archivo del servidor, cada administrador del producto puede obtener los

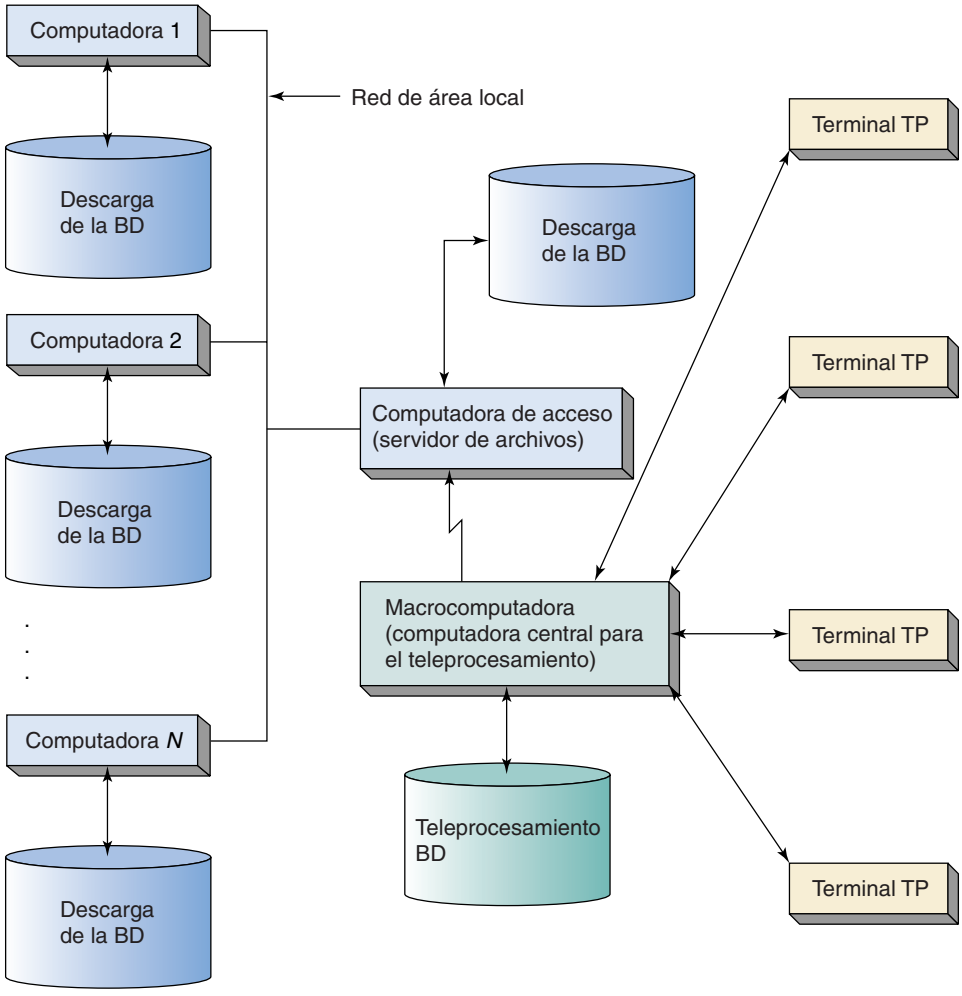


FIGURA 17-8

Cómo se comparten los datos descargados con una micro de acceso como el servidor de archivos

datos que le interesen a partir de ese servidor. Los controles aseguran que los administradores de producto no obtengan datos para los que no tengan autorizado el acceso.

Problemas potenciales en el procesamiento de las bases de datos descargadas

Los datos descargados se acercan más al usuario y aumentan su utilidad. Por desgracia, esto puede ocasionar problemas, incluyendo la coordinación, la consistencia, el control de acceso, y la violación computacional.

COORDINACIÓN. Primero considere la coordinación utilizando las tablas VENTAJA-PRODUCTO y ANUNCIO-APARIENCIA con fines ilustrativos. La tabla VENTAJA-PRODUCTO se actualiza a partir de los datos de la macrocomputadora (las ventajas las maneja el personal de ventas y se registran en la macrocomputadora). Pero la tabla ANUNCIO-APARIENCIA la actualiza "localmente" el usuario clave del departamento de mercadotecnia, el cual obtiene los datos de los reportes que preparan el gerente de mercadotecnia y la agencia de publicidad.

Esta situación podría causar problemas cuando un anuncio se publica por primera vez. Por ejemplo, podría generar ventajas que están registradas en la base de datos de la computadora central antes de que los datos de ANUNCIO-APARIENCIA se almacenen en el servidor de archivos. Después, cuando se descargan estas ventajas, el programa importa los datos que tendrán que rechazar los datos ventaja, porque dichos datos violan la restricción de que una VENTAJA-PRODUCTO debe tener un padre ANUNCIO-APARIENCIA. Así, las actividades de actualización local y descarga se deben coordinar cuidadosamente: el usuario clave necesita insertar los datos ANUNCIO-APARIENCIA antes de importar los datos de la computadora central. Pueden ocurrir problemas similares en la coordinación cuando se actualizan los datos de VENTAS y PRESUPUESTO.

CONSISTENCIA. El segundo problema con los datos descargados se refiere a la **consistencia.** Cada uno de los administradores del producto recibe datos descargados de VENTAS y PRESUPUESTO que se supone no cambian. Pero, ¿qué pasaría si un administrador del producto cambió los datos? En este caso, los datos de esa base del administrador del producto no coinciden con los de la base de datos de la compañía, con los del servidor de archivos y posiblemente con los de otra base de datos de otro administrador del producto. Los reportes que produzca ese administrador de producto podrían estar en desacuerdo con otros reportes. Si varios administradores del producto actualizan los datos, podrían generarse muchos datos inconsistentes.

Obviamente, esta situación requiere un control estricto. La base de datos debe designar qué datos no se pueden actualizar. Si esto no es posible, por ejemplo si el producto de la base de datos de la computadora personal no impone una restricción como ésta y los costos de los programas de escritura para la imposición son prohibitivamente altos, la solución al problema sería la educación. Los administradores del producto deben estar conscientes de los problemas que surgirán si cambian los datos y no deben hacerlo directamente.

CONTROL DE ACCESO. Un tercer problema es el control de acceso. Cuando los datos se transfieren a varios sistemas computacionales, el control de acceso se vuelve muy difícil. Por ejemplo, en Universal los datos de VENTAS y PRESUPUESTO son una cuestión muy delicada y probablemente el vicepresidente de ventas no quiere que el personal de ventas se entere de los próximos presupuestos hasta que se conozcan las ventas anuales. Pero si 15 administradores de producto tienen copias de esto en sus bases de datos, por el momento puede ser difícil asegurar la confidencialidad de los datos.

Además, el servidor de archivos recibe todos los datos de VENTAS y PRESUPUESTO, los cuales se supone que se descargaron de tal manera que un administrador del producto recibe solamente esos datos para los productos que administra. Sin embargo, los administradores de producto pueden ser muy competitivos y quizá quieran encontrar los datos de los productos de otros. Permitir que dichos datos estén disponibles en el servidor de archivos en el departamento de mercadotecnia puede crear problemas administrativos.

➤ FIGURA 17-9

Asuntos y problemas potenciales respecto a las aplicaciones de datos descargados

Coordinación
- Los datos descargados deben ser acordes a las restricciones de la base de datos
- Las actualizaciones locales deben estar coordinadas con las descargas de datos

Consistencia
- En general, los datos descargados no deberán actualizarse
- Las aplicaciones necesitan características adicionales para evitar la actualización
- Los usuarios deben estar conscientes de los posibles problemas

Control de acceso
- Los datos se pueden reproducir en muchas computadoras
- Los procedimientos para el control de acceso de datos son más complicados

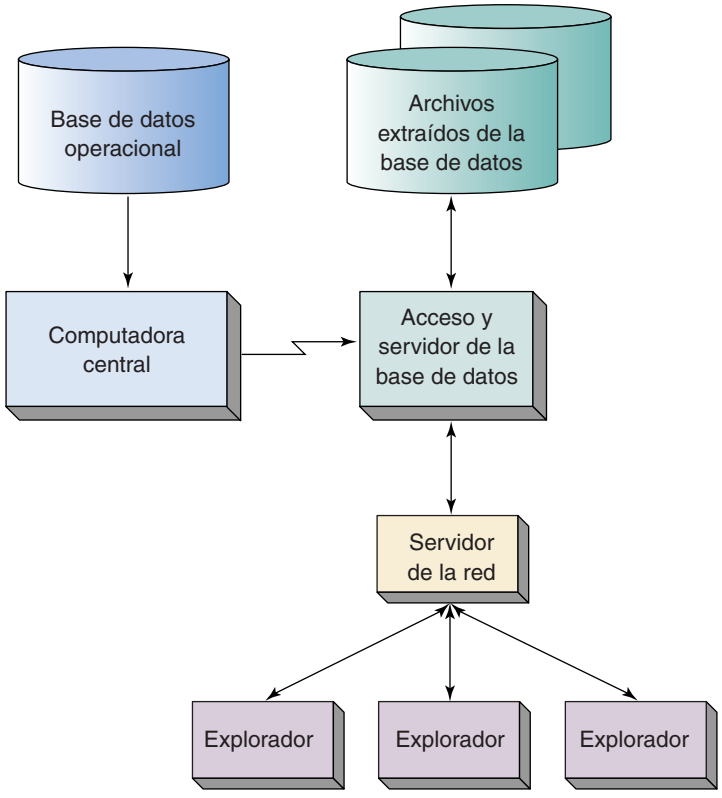
Potencialidad del delito computacional
- El copiado ilegal es difícil de evitar
- Los disquetes y el acceso ilegal en línea son fáciles de ocultar
- El riesgo puede evitar el desarrollo de aplicaciones para descarga de datos

VIOLACIÓN COMPUTACIONAL. El cuarto problema, una gran posibilidad de delito computacional, está estrechamente vinculado con el control de acceso. Mientras que dicho control se refiere a actividades inapropiadas pero legales, el delito se refiere a las acciones ilegales. Los datos de la computadora central de la compañía pueden ser muy valiosos; por ejemplo, los de ventas y presupuestos de Equipo Universal son de gran interés para sus competidores.

Cuando los datos se descargan en volumen al servidor de archivos y después a una o muchas computadoras personales, el copiado ilegal es difícil de evitar. Un disquete o un CD se pueden ocultar fácilmente y los empleados a veces tienen conexiones en línea con las que ingresan a las computadoras de trabajo en lugares remotos. En estas si-

➤ FIGURA 17-10

Procesamiento de datos descargados con un servidor Web



tuaciones es casi imposible detectar o evitar el copiado de datos. El riesgo del delito computacional sólo puede prohibir que un sistema se desarrolle, aun cuando de otra manera sería una excelente solución. Los problemas potenciales de las bases de datos descargadas se resumen en la figura 17-9.

Utilización de un servidor de la red para publicar datos descargados

La figura 17-10 muestra una manera de usar un servidor de la red para publicar los datos que se descargaron. Los servidores de salida y de la base de datos se muestran aquí en una computadora, pero podrían residir en dos —una computadora para el acceso y otra para el servidor—. El servidor de la red se comunica con el de la base de datos para obtener los datos que se descargaron. Entonces, se publican dichos datos para los usuarios del explorador.

➤ PROCESAMIENTO ANALÍTICO EN LÍNEA (OLAP)

En años recientes ha surgido una nueva forma de procesar información denominada **Procesamiento analítico en línea,** o más comúnmente, **OLAP.** Con OLAP los datos se ven en el marco de una tabla, o con tres ejes, en un **cubo.** OLAP no tiene límite en el número de ejes; así, algunas veces escuchará el término **hipercubo OLAP.** Este término significa un despliegue con un número ilimitado de ejes. El término *cubo OLAP* se utiliza con más frecuencia.

➤ FIGURA 17-11

Fuente de datos relacionales para el cubo OLAP

Categoría	Tipo	Ciudad	Estado	Fecha	Precio de venta	Precio inicial
Nuevo	Una sola familia	San Francisco	California	1/1/2000	679,000	685,000
Existente	Condominio	Los Ángeles	California	3/5/2001	327,989	350,000
Existente	Una sola familia	Elko	Nevada	7/17/2001	105,675	125,000
Nuevo	Condominio	San Diego	California	12/22/2000	375,000	375,000
Existente	Una sola familia	Paradise	California	11/19/2001	425,000	449,000
Existente	Una sola familia	Las Vegas	Nevada	1/19/2001	317,000	325,000
Nuevo	Una sola familia	San Francisco	California	1/1/2000	679,000	685,000
Existente	Condominio	Los Ángeles	California	3/5/2001	327,989	350,000
Existente	Condominio	Las Vegas	Nevada	6/19/2001	297,000	305,000
Existente	Una sola familia	Los Ángeles	California	4/1/2000	579,000	625,000
Nuevo	Condominio	Los Ángeles	California	8/5/2001	321,000	320,000
Etc.						

Considere la relación de ejemplo que se muestra en la figura 17-11. Estos datos se refieren a las ventas de bienes inmuebles para una sola familia y en condominio en California y Nevada. Como puede ver en los datos, se incluyen el Precio de venta y el Precio inicial, tanto para construcciones nuevas como para propiedades anteriormente edificadas.

TERMINOLOGÍA OLAP. En la figura 17-12 se muestra el cubo OLAP para estos datos, los cuales tienen dos **ejes:** columnas y renglones. Los ejes del renglón despliegan la **dimensión** de Fecha y los ejes de las columnas despliegan las dimensiones Categoría y Ubicación. Cuando dos o más dimensiones se muestran sobre un eje, cada combinación de una se muestra para cada combinación de la otra. Ejemplo: las celdas del cubo representan las **medidas** del cubo, o de los datos que se despliegan. En este cubo la medida es el promedio de Precio de Venta. Otras medidas se refieren al Precio Inicial o incluso la diferencia entre Precio de Venta y Precio Inicial.

Observe que todos los datos en la figura 17-12 se refieren a viviendas unifamiliares. En este cubo no hay datos referentes a condominio. De hecho, hay dos de estos cubos: uno para familias y otro para condominios. Podría pensar que ambos están uno detrás del otro, como se bosqueja en la figura 17-13. Cuando se ve de esta manera, es-

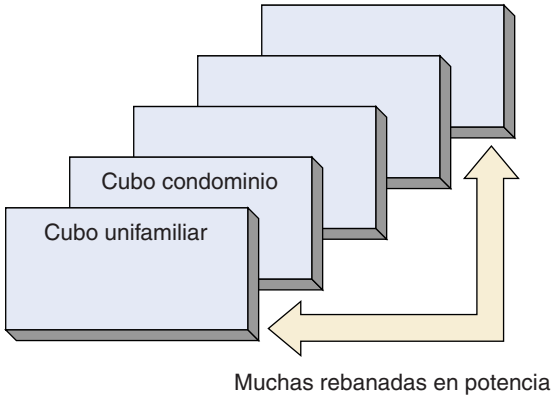
➤ FIGURA 17-12

Ejemplo de cubo OLAP

Promedio de precio de venta de viviendas unifamiliares ($miles)										
			Estructuras ya edificadas				Construcción nueva			
			California			Nevada	California			Nevada
			San Francisco	Los Ángeles	San Diego		San Francisco	Los Ángeles	San Diego	
2000	Q1	Enero	408	465	375	179	418	468	371	190
		Febrero	419	438	382	180	429	437	382	185
		Marzo	427	477	380	195	426	471	387	198
	Q2		433	431	382	188	437	437	380	193
	Q3		437	437	380	190	438	439	382	190
	Q4		435	439	377	193	432	434	370	198
2001	Q1	Enero	452	454	368	198	450	457	367	197
		Febrero	450	467	381	187	457	464	388	191
		Marzo	432	444	373	188	436	446	371	201
	Q2		437	437	368	190	444	432	363	196
	Q3		436	452	388	196	447	455	385	199
	Q4		441	455	355	198	449	455	355	202

➤ FIGURA 17-13

Dimensiones de las rebanadas del cubo OLAP



tos dos cubos aparentan ser dos rebanadas de los datos y, de hecho, la o las dimensiones que se mantienen constantes en un cubo se llaman **rebanadas (slices).** Así, en este ejemplo el cubo está rebanado en Tipo.

Los valores de una dimensión se llaman **miembros (members).** Los miembros de la dimensión Tipo son {familia, condominio}, y los de la dimensión Categoría son {Nuevo, Ya existente}. Para este cubo, los miembros de la dimensión Estado son {California, Nevada}, pero en general podría haber 50 miembros para las propiedades de Estados Unidos de América. A veces hay un número muy grande de miembros en una dimensión; considerando todos los miembros para la combinación {Estado, Ciudad}. Por último, en algunos casos se calculan los miembros. La fecha y la hora son buenos ejemplos. Con una fecha determinada podemos calcular el mes, trimestre, año, o miembros del siglo para esa fecha.

Un último término importante de OLAP es el **nivel (level).** El nivel de una dimensión es su posición en una jerarquía. Por ejemplo, considere la dimensión Fecha. Sus niveles son Año, Trimestre, Mes. Los niveles en la dimensión de ubicación son Estado, Ciudad. La terminología OLAP se resume en la figura 17-14.

➤ FIGURA 17-14

Terminología OLAP

Término	Descripción	Ejemplo en la figura 17-12
Eje	Una coordenada del hipercubo	Renglones, columnas
Dimensión	Una característica de los datos para colocarlos sobre un eje	Tiempo, tipo de vivienda, Ubicación
Nivel	Un subconjunto (jerárquico) de una dimensión	{California, Nevada} {San Francisco, Los Ángeles, otros} {Q1, Q2, Q3, Q4}
Miembro	Un dato valuado en una dimensión	{Nuevo, Ya existente}, {enero, febrero, marzo}
Medida	La fuente de datos para el hipercubo	Precio de venta, Precio inicial
Rebanada	Una dimensión o medida que se mantiene constante para el despliegue	Tipo de vivienda –los que se muestran son unifamiliares– existe otro cubo para condominio

CUBO Y DEFINICIONES DE VISTAS. La terminología OLAP está evolucionando y actualmente es bastante ambigua. El término *cubo* se utiliza para describir una estructura semántica y también para materializaciones de la estructura implícita. El cubo que se muestra en la figura 17-12 es una posible vista o materialización de una estructura semántica que tiene ciertas dimensiones, niveles y medidas. Podríamos crear un segundo cubo en estos datos mediante el intercambio de los renglones y las columnas; podríamos crear un tercer cubo en estos datos mostrando la Ubicación en la parte superior y después colocando una columna Nueva y Existente por cada miembro de la Ubicación. A medida que lea los documentos de OLAP, tenga cuidado de comprender el significado del *cubo* que está utilizando.

Para ilustrar más este punto, considere la definición del cubo en la figura 17-15. La sintaxis utilizada aquí está basada en OLE DB de Microsoft por la documentación OLAP, pero también es similar a lo que usaron otros proveedores. La instrucción Create Cube (Crear Cubo) define cuatro dimensiones y dos niveles en la estructura lógica. Las de Tiempo y Ubicación tienen niveles y las de CategoríaVivienda y TipoVivienda no tienen dimensiones. Aunque no lo mostramos aquí, es posible que una dimensión tenga más de un conjunto de Niveles. En ese caso, se definen dos o más jerarquías para esa dimensión.

La estructura que se muestra en la figura 17-15 es una definición de una forma de interpretar o entender los datos de las viviendas. No es una presentación de datos. Para definir una presentación de datos, o materialización, el mundo OLAP ha extendido

➤ FIGURA 17-15

Uso ampliado de SQL para OLAP: (a) ejemplo de la instrucción Definición de datos para crear un cubo

```
CREATE CUBE CuboVentaVivienda (
        DIMENSION Tiempo TYPE TIME,
                LEVEL Año TYPE YEAR,
                LEVEL Trimestre TYPE QUARTER,
                LEVEL Mes TYPO MONTH,
        DIMENSION Ubicación,
                LEVEL Estados Unidos TYPE ALL,
                LEVEL Estado,
                LEVEL Ciudad,
        DIMENSION CategoríadeVivienda,
        DIMENSION TipodeVivienda,
        MEASURE PreciodeVenta,
                FUNCTION AVG
MEASURE PrecioInicial,
        FUNCTION AVG
)
```

(a)

➤ FIGURA 17-15

SQL ampliado usado por OLAP: (b) ejemplo multidimensional de la instrucción SELECT

```
SELECT      CROSSJOIN ({Estructura existente, Nueva construcción}, {California.Niños,
            Nevada})
            ON COLUMNS,
            {1998.Q1.Niños, 1998.Q2, 1998.Q3, 1998.Q4,
            1999.Q1.Niños, 1999.Q2, 1999.Q3, 1999.Q4}
            ON ROWS
FROM        CuboVentasViviendas
WHERE       (PrecioVentas, TipoVivienda= 'Unifamiliar')
```

(b)

la sintaxis de SQL. La figura 17-15(a) muestra el OLAP SQL para crear la materialización del cubo que se muestra en la figura 17-11. Lo único confuso de esta instrucción es el término CROSSJOIN ({A, B}, {1, 2}) que da como resultado el siguiente despliegue:

A		B	
1	2	1	2

Un CROSSJOIN ({1, 2}, {A, B}) da como resultado el siguiente despliegue:

1		2	
A	B	A	B

Extendiendo un poco esta idea, el CROSSJOIN ({Estructura existente, Nueva construcción}, {California.Niños, Nevada}) da como resultado:

Estructuras existentes				**Nueva construcción**			
California			**Nevada**	**California**			**Nevada**
San Francisco	Los Ángeles	San Diego		San Francisco	Los Ángeles	San Diego	

La única adición a esta última instrucción fue la expresión California.Niños. Este término simplemente significa omitir todos los niños de California para todos los niveles definidos en el cubo.

El SQL en la figura 17-15(b) incluye la expresión ON COLUMNS y ON ROWS; declara los ejes en los que se colocan las dimensiones. Observe también que la cláusula WHERE se utiliza para especificar las rebanadas para la presentación. Sólo se muestran

Precio de Venta y TipodeVivienda unifamiliar. Observe que una medida y una dimensión pueden servir como rebanador.

Una de las ideas clave de OLAP es que los usuarios son capaces de reformatear dinámicamente un cubo con facilidad, desde sus escritorios (de ahí las palabras **en línea**). Para hacer esto, los programas que procesan las materializaciones del cubo necesitan ser dinámicamente capaces de construir OLAP SQL, como la de la figura 17-16(b).

ESTRUCTURAS DE LOS ESQUEMAS OLAP. Todos los datos para el ejemplo OLAP de las figuras 17-11 y 17-12 surgen de una sola tabla. Esto no es común; por lo general los datos para cuando menos alguna de las dimensiones se almacenan en la figura 17-16 que muestra el ejemplo de las estructuras de la tabla de OLAP simple para la Galería View Ridge. La medida de los datos, es decir, SalesPrice (PreciodeVenta), se almacena en la tabla TRANS, pero los datos de las dimensiones se guardan en las tablas padre que están conectadas mediante llaves externas a TRANS.

Un cubo basado en los datos de la figura 17-16(a) podría tener dimensiones CUSTOMER, ART_DEALER, SALESPERSON y WORK (CLIENTE, ART_VENDEDOR, VENDEDOR y TRABAJO). Los datos miembro para estas dimensiones serán obtenidos de las tablas relacionadas. La estructura que se muestra en la figura 17-16(a) ocurre con tanta frecuencia en un procesamiento OLAP, que se le ha dado el nombre **esquema estrella,** para referirse al patrón de la dimensión de las tablas que rodean a la que tiene los datos de medida.

Observe que esta figura no incluye ARTIST Name (Nombre del ARTISTA). Para incluirlo el diseñador OLAP tiene una opción para juntar Name del ARTIST a la tabla WORK, mediante la llave ArtistID. Si se hace esto, WORK no estaría en la forma normal dominio-llave y consecuentemente ahí se duplicarían los datos.

En la figura 17-16(b) se muestra una alternativa para el almacenamiento de estos joins. Aquí, la tabla ARTIST no hace join con WORK, sino que se mantuvo en una forma normalizada. Se ha agregado también otra tabla, GENRE. Esta estructura de tabla ocurre también, con frecuencia, y se le ha dado el nombre de **esquema copo de nieve**. La diferencia entre la estructura de estrella y la de copo de nieve es que con la estrella cada dimensión de la tabla está adyacente a la que almacena los valores de las medidas. Estas tablas pueden estar o no normalizadas. Con la estructura copo de nieve puede haber múltiples niveles de tablas y cada una estará normalizada.

La elección entre estas dos estructuras depende del tamaño y la naturaleza de los datos, y también del trabajo de carga OLAP. En general, el esquema estrella requiere mayor almacenamiento, pero es más rápido de procesar. El del copo de nieve es más lento, pero usa menos almacenamiento.

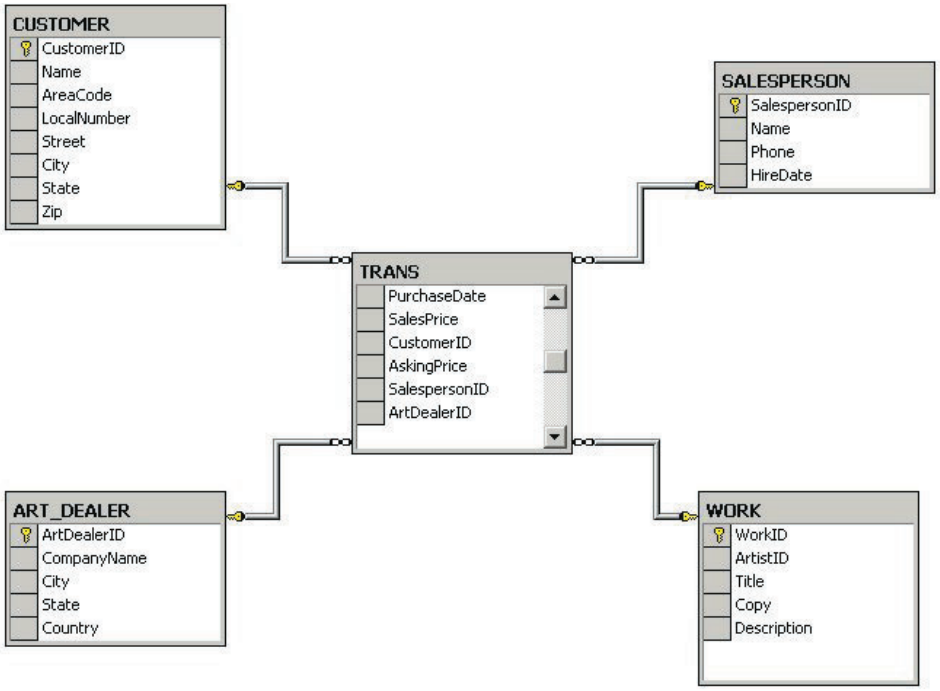
ALTERNATIVAS DE ALMACENAMIENTO OLAP (ROLAP, MOLAP Y HOLAP). No, no estamos hablando de una versión de alta tecnología de los siete enanos. ROLAP, MOLAP y HOLAP se refieren a diferentes significados para el almacenamiento de datos OLAP. Básicamente la pregunta es, con el fin de lograr un mejor desempeño, ¿los productos relacionales DBMS se deben ampliar para incluir facilidades especiales para OLAP, o se debe usar un mecanismo de procesamiento de propósito especial, o sería mejor usar ambos?

Los partidarios del almacenamiento ROLAP (OLAP relacional) declaran que con el preprocesamiento de ciertas consultas, y con otras extensiones, los productos relacionales DBMS son los más adecuados. Los que apoyan el almacenamiento de MOLAP (OLAP multidimensional) creen que, aun cuando el DBMS relacional está bien para el procesamiento de transacción de consultas y reportes, los requisitos de procesamiento de OLAP son tan especializados que ningún DBMS puede producir un rendimiento OLAP aceptable. El tercer grupo, HOLAP (OLAP híbrido), afirma que los productos DBMS y las máquinas especializadas OLAP tienen una función y se pueden utilizar con provecho.

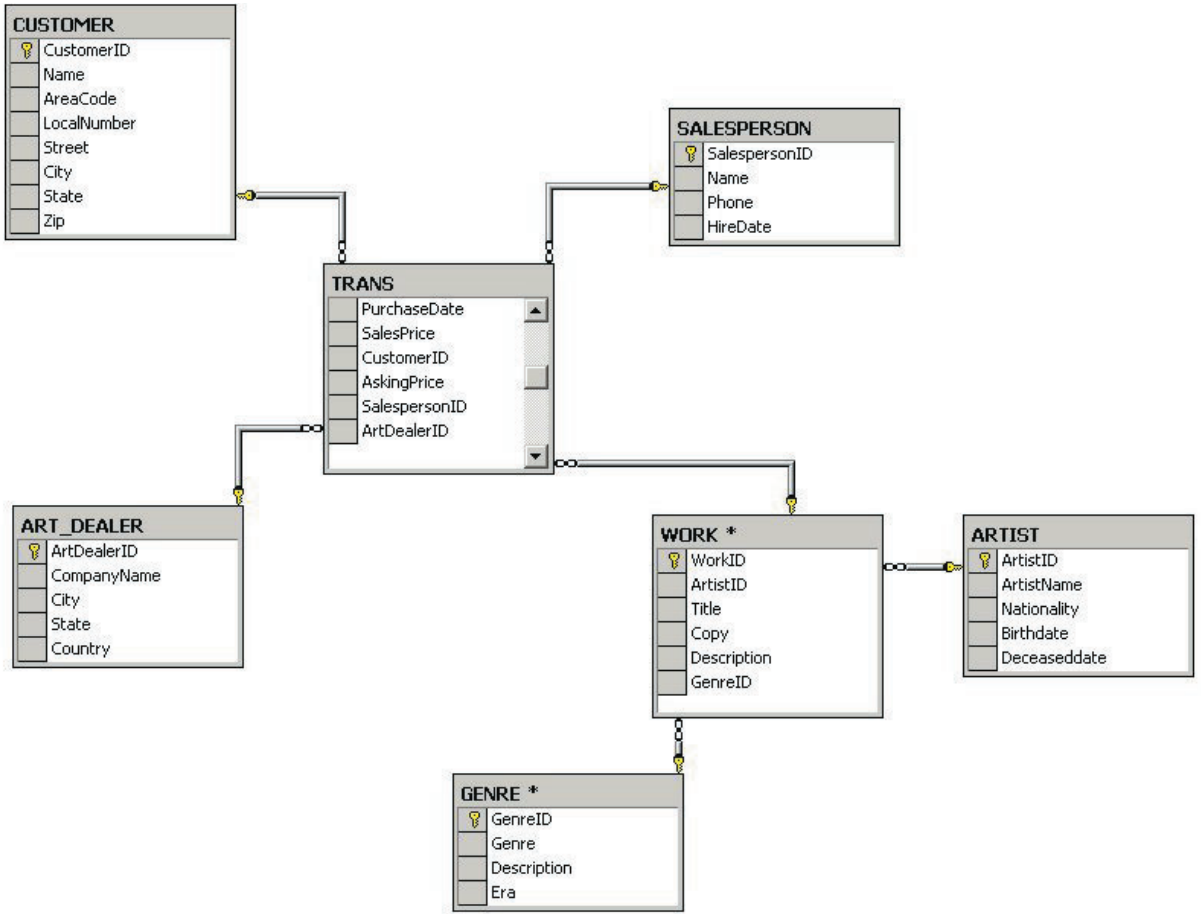
Microsoft usa estos términos más cerca del contexto del SQL Server OLAP. Para Microsoft, ROLAP significa que la fuente de datos y las agregaciones precalculadas de los datos serán almacenadas en una base de datos del SQL Server. Por otra parte, con MOLAP los datos, la estructura del cubo y las agregaciones precalculadas de datos estarán almacenadas en una estructura de datos multidimensional de propósito especial. Con HOLAP los datos se quedan en la base de datos relacional, pero las agregaciones de datos se almacenan en una estructura de datos multidimensional.

➤ FIGURA 17-16

Tipos de esquema OLAP: (a) ejemplo del esquema estrella, y (b) ejemplo del esquema copo de nieve



(a)



(b)

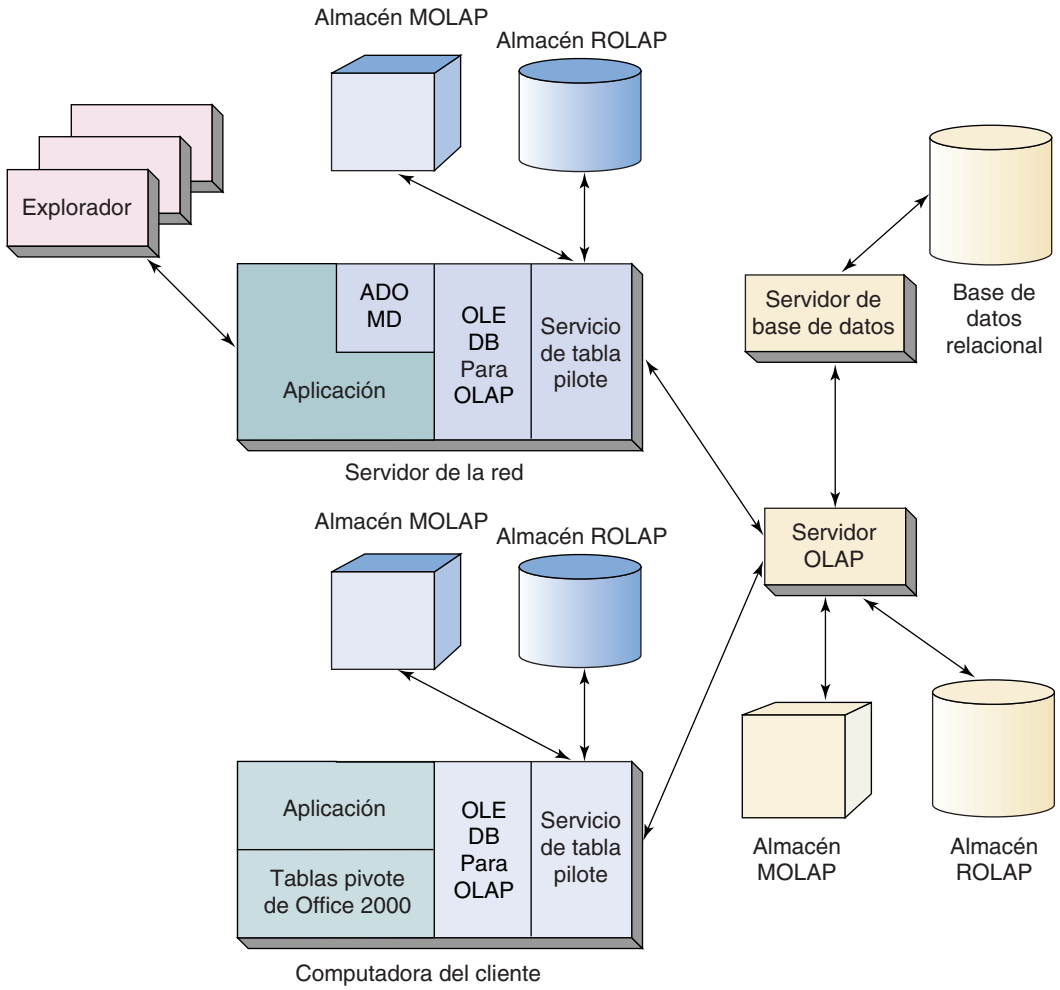
MOLAP da como resultado el mejor desempeño, pero requiere más almacenamiento. ROLAP usa menos almacenamiento, pero es lento. Conviene para grandes bases de datos que rara vez tienen consultas. HOLAP es un compromiso con el desempeño rápido para actividad OLAP de alto nivel, pero será muy lento para explorar niveles finos de detalle.

La figura 17-17 muestra la arquitectura OLAP que anunció Microsoft con el SQL Server 7.0 y Office 2000. Esta arquitectura OLAP implica el procesamiento OLAP en los servidores centrales de datos, en el Servidor de la red y en las computadoras de los clientes. Las bases de datos empresariales se procesan mediante los servidores centrales de datos que se muestran en el lado derecho de este diagrama. Los resultados de este procesamiento están disponibles para el servicio de la tabla pivote en un Servidor de la red o en la computadora de un cliente. Además, el Servidor de la red o la computadora de un cliente pueden tener versiones locales de datos OLAP.

Hay varios elementos clave de esta arquitectura. Primero, el servicio de la tabla pivote reside en un procesador OLAP, el cual está disponible como servicio NT y Windows 2000. También se encuentra disponible en otras versiones de Windows que se ejecutan en Office 2000. En efecto, el servicio de la tabla pivote se invoca cada vez que se crean datos para acceder a las páginas en Access. Este servicio se utiliza con más frecuencia en Excel.

➤ FIGURA 17-17

Arquitectura OLAP Microsoft



El servicio de la tabla pivote se despliega a través de una extensión a OLE DB llamada OLE DB para OLAP. Esta extensión se basa en el OLE DB que usted estudió en el capítulo 15; básicamente, se extiende a la abstracción del rowset no sólo para incluir los conjuntos de registros, sino también los conjuntos de datos, que son abstracciones de los cubos. La extensión ADO para el procesamiento OLE DB para OLAP se llama ADO MD (multidimensional). Con ADO MD, los objetos Conexión y Comando pueden abrir conjuntos de datos y procesarlos dinámicamente en forma parecida a la que muestra para los conjuntos de registros en el capítulo 15. Los datos pueden ser de lectura y escritura.

Esta arquitectura mueve tanto procesamiento OLAP como es posible para el cliente, debido a que los requisitos de procesamiento de OLAP pueden ser mayores. No hay desventaja cuando el procesamiento de datos se almacena localmente, pero cuando se crean cubos que requieren datos de un servidor de la empresa central puede ocurrir una transmisión de datos considerable. Desde luego, esto puede no ser aceptable; estos sistemas necesitarán afinarse conforme se obtenga experiencia.

Como se indica en la figura 17-17, el procesamiento OLAP se puede hacer sobre cargas centralizadas, o sobre datos locales. A medida que las organizaciones proporcionan más datos a los usuarios, aumentan los problemas de administración de datos. Una posible solución a este problema es el data warehouse que consideraremos a continuación.

➤ DATA WAREHOUSES

La descarga de datos hace que se muevan los datos más cerca del usuario y de ese modo aumenta su utilidad potencial. Por desgracia, mientras que uno o dos sitios para descargar datos se pueden administrar sin ningún problema, si cada departamento quiere tener su propia fuente de descarga de datos los problemas de administración se vuelven inmensos. Por lo tanto, las organizaciones comenzaron a buscar algunos medios para proporcionar un servicio estandarizado para mover los datos al usuario y hacerlos más útiles. Ese servicio se llama data warehousing.

Un **data warehouse** es un almacén de datos empresariales que fue diseñado para facilitar la toma de decisiones en administración. No sólo incluye datos, sino herramientas, procedimientos, capacitación, apoyo de personal y otras facilidades que facilitan el acceso a los datos para aquellos que toman decisiones. El objetivo del data warehouse es aumentar el valor de los activos de datos de la organización.

Como se muestra en la figura 17-18, el papel del data warehouse es almacenar extractos de los datos operacionales y hacer que estén disponibles para los usuarios en un formato útil. Los datos se pueden extraer de las bases de datos y archivos, pero también pueden ser documentos, imágenes, registros, fotos, y otros datos no escalables. La fuente de datos también se podría adquirir de otras organizaciones. El data warehouse almacena los datos extraídos y también los combina, conjunta, transforma y pone a disposición de los usuarios mediante herramientas que están diseñadas para el análisis y la toma de decisiones, como por ejemplo OLAP.

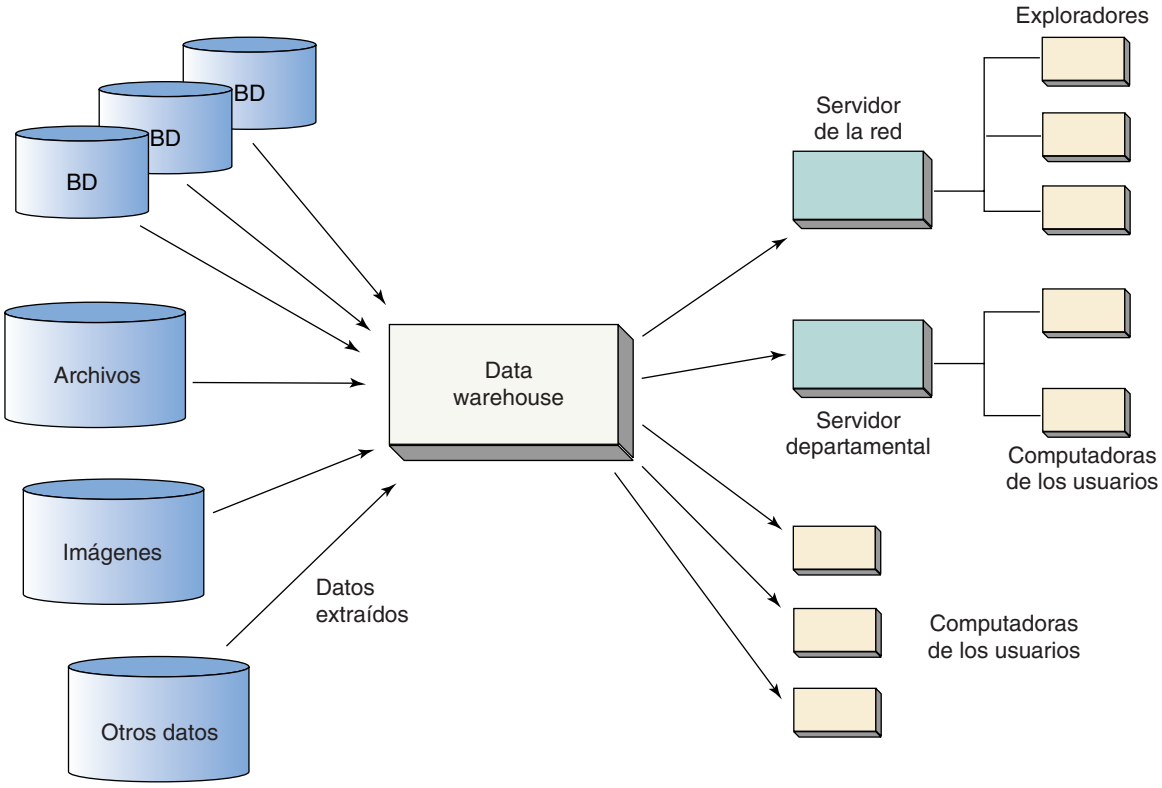
Componentes de un Data Warehouse

Los componentes de un data warehouse se enumeran en la figura 17-19. Como se estableció, la fuente del depósito son datos operacionales. Por lo tanto, el data warehouse necesita herramientas para la extracción y el almacenamiento de los datos. Sin embargo, éstos no son útiles sin los metadatos que describen la naturaleza de los datos, sus orígenes, formato, límites de uso y otras características que influyen en la forma en que se pueden y deben utilizar.

Potencialmente, el data warehouse contiene billones de bytes de datos en muchas formas diferentes. Por lo tanto, necesita sus propios servidores DBMS y OLAP para almacenar y procesar datos. De hecho, se pueden utilizar varios productos DBMS y OLAP, y sus características y funciones pueden aumentar mediante un software desa-

➤ FIGURA 17-18

Data warehouse



rrollado que reformatea, agrega, integra y transfiere datos de un procesador a otro dentro del Data warehouse. También se pueden necesitar programas para almacenar y procesar datos sin escala, tales como gráficas y animaciones.

Debido a que la finalidad del data warehouse es hacer que los datos organizacionales estén más disponibles, el depósito debe incluir herramientas no sólo para repartir datos a los usuarios, sino también para transformarlos o realizar análisis, consultas y reportes, y OLAP para agregaciones específicas de los usuarios y desagregaciones.

El data warehouse proporciona un importante, pero complicado, conjunto de fuentes y servicios. Por lo tanto, el depósito necesita incluir cursos de capacitación, materiales didácticos, utilerías de ayuda en línea y otros productos similares de capaci-

➤ FIGURA 17-19

Componentes de un data warehouse

- Herramientas de extracción de datos
- Datos extraídos
- Metadatos de los contenidos del depósito
- Depósito DBMS y servidores OLAP
- Herramientas para la administración del data warehouse
- Programas de entrega de datos
- Herramientas de análisis del usuario final
- Cursos y materiales de capacitación para el usuario
- Asesores del data warehouse

tación para facilitar que los usuarios saquen provecho de los recursos del depósito. Por último, éste incluye personal que realiza tareas de asesoría.

Requerimientos para un data warehouse

Los requisitos para un data warehouse son diferentes a los de una aplicación de una base de datos tradicional. Para unos, en una aplicación de una base de datos tradicional la estructura de reportes y consultas está estandarizada. Mientras que los datos en un reporte o consulta varían mes con mes, por ejemplo, la estructura del reporte o consulta permanece igual. Por otra parte, los usuarios del data warehouse con frecuencia necesitan cambiar la estructura de las consultas y reportes.

Considere el siguiente ejemplo. Suponga que una compañía define los territorios de ventas geográficamente —por simplicidad, digamos que se asigna un vendedor a cada estado o provincia en EUA—. Ahora, digamos que un usuario de un data warehouse quiere investigar el impacto en las comisiones de ventas, si en lugar de asignar a los vendedores geográficamente se les asigna por cuentas específicas. Para comparar estas alternativas, en primer lugar las ventas deben estar agrupadas por compañías, y en segundo, por estado. Para tal propósito será necesario crear consultas y reportes con diferentes estructuras.

Otra diferencia es que los usuarios quieren hacer sus propias agregaciones de datos. Por ejemplo, un usuario quiere investigar el impacto de diferentes campañas de mercadotecnia; tal vez quiere agregar las ventas de los productos de acuerdo al color del empaque en primera instancia, de acuerdo al programa de mercadotecnia, al color del empaque dentro del programa de mercadotecnia en un tercer caso, y al programa de mercadotecnia con el color del empaque. El analista quiere los mismos datos en cada reporte; simplemente quiere *rebanarlas y acomodarlas (slice and dice) de manera diferente*.

No sólo los usuarios de los data warehouses quieren hacer agregaciones de datos en sus propios términos, quizá también quieran desagregarlos en sus propios términos, o, como se llama comúnmente, estos usuarios quieren hacer **drill down** a sus datos. Por ejemplo, un usuario podría querer presentar una pantalla que muestre las ventas totales de los productos en un año determinado, hacer clic en los datos y tener el desglose de ventas por mes; hacer clic nuevamente y tener los datos desglosados por ventas de producto por mes, o ventas por región, por producto y por mes. Mientras que las aplicaciones de la base de datos pueden estar escritas para satisfacer esta necesidad con respecto a un conjunto específico de requerimientos de datos trabajados, con más frecuencia, los requisitos varían por usuario y por tarea. De hecho, algunas veces el usuario no sabe cómo quiere trabajar con sus datos hasta que los ve y comienza a manejarlos; por lo tanto, las herramientas para trabajar con los datos necesitan ser flexibles.

La salida gráfica es otro requisito común. Los usuarios quieren ver los resultados de los datos geográficos en una forma geográfica. Las ventas por estado y provincia se deberán mostrar en un mapa de América del Norte. Una reorganización de empleados y oficinas se deben mostrar en un diagrama del espacio de la oficina. De nuevo, estos requisitos son más difíciles porque varían de un usuario a otro y de una tarea a otra.

Por último, muchos usuarios de dispositivos de depósitos de datos quieren importar depósitos de datos a programas de dominio específico. Por ejemplo, los analistas financieros quieren importar datos en sus modelos de hojas de cálculo y en programas de análisis financiero más sofisticados. Los administradores de la cartera de clientes quieren importar datos a los programas de administración de dicha cartera, y los ingenieros de aceite para perforadoras quieren importar datos de programas de análisis sísmicos. Todas estas importaciones normalmente significan que el data warehouse necesita formatearse en formas específicas. Estos requisitos se resumen en la figura 17-20.

Retos para los data warehouses

Hasta ahora hemos descrito los data warehouses en una forma idealizada que los hace parecer como una panacea para la toma de decisiones administrativas. En realidad, es muy difícil repartir las capacidades que hemos descrito. Hay varios retos y problemas importantes que hay que conocer y resolver.

➤ FIGURA 17-20

Categorías de requerimientos para un data warehouse

- Consultas y reportes con estructura variable
- Agregación de datos de un usuario específico
- El usuario especifica la forma de trabajar con los datos
- Salidas gráficas
- Integración con programas de dominio específico

DATOS INCONSISTENTES. Obviamente, el data warehouse es inservible, si no es que realmente perjudicial, si los datos que proporciona no son exactos. El asunto va más allá de la calidad de los datos que el depósito extrae de sus fuentes. La fuente de datos puede ser exacta cuando se extrajeron, pero se pueden introducir imprecisiones involuntarias al integrar datos que son inconsistentes en cuanto a coordinación o dominio.

Considere el ejemplo de los datos extraídos en la figura 17-21. La tabla uno es un extracto de datos de pedido, y la segunda es un extracto de cheques que fueron expedidos como bonos para el personal de ventas. Suponga que un usuario del data warehouse quiere investigar la relación entre el desarrollo de las ventas con los bonos por ese concepto. A primera vista parecería que todo lo que se necesita hacer es sumar las cantidades ordenadas por cada vendedor y comparar el total con los bonos. El siguiente código SQL realizará esa tarea:

```
SELECT      SPNombre, Suma(CantidaddelPedido), CantidaddelBono
FROM        PEDIDO, BONO
WHERE       PEDIDO.SPNum = BONO.SP#
GROUP BY    SpNum
```

(Por otra parte, el típico usuario del data warehouse probablemente no sepa lo suficiente de SQL como para escribir este código, así que necesitará escribirle a él o a ella, para que se lo proporcione indirectamente a través de algún tipo de interfaz de consulta gráfica.)

Ahora suponga que estos datos eran correctos cuando se extrajeron, y que se obtuvieron de dos sistemas de información diferentes —procesamiento del pedido y cantidades pagadas—. Puesto que se obtuvieron de dos sistemas diferentes no se sabe si la coordinación de los datos es consistente o no. Puede ser que los datos de los pedidos sean correctos hasta el último viernes del mes mientras que los datos de la revisión de los bonos estuvieran correctos hasta el último día del mes. No hay nada en los datos que indique la diferencia y, en realidad, nadie puede notar si ésta existe. Sin embargo, puede tener un impacto sustancial en los resultados del análisis.

➤ FIGURA 17-21

Ejemplo de los datos extraídos de pedido y de bono

Tabla PEDIDO

SPNúm	NúmerodePedido	CantidaddelPedido
100	1000	$12,000
200	1200	$17,000
100	1400	$13,500
300	1600	$11,335

Tabla BONOS

SP#	SPNombre	CantidaddelBono
100	Mary Smith	$3,000
200	Fred Johnson	$2,500
300	Laura Jackson	$3,250

➤ FIGURA 17-22

Datos por vendedor y región para dos años

Tabla PERSONALDEVENTAS

SPNombre	Región	Año	VentasTotales
Johnson	SO	2000	$175,998
Wu	NW	2000	$223,445
O'Connor	NE	2000	$337,665
Abernathy	SE	2000	$276,889
Lopez	SW	2000	$334,557
Johnson	SO	2001	$225,998
Wu	NW	2001	$276,445
O'Connor	NE	2001	$389,737
Abernathy	SO	2001	$362,768
Lopez	SO	2001	$419,334

Tabla REGIÓN

RegióndeVentas	Administrador
NW	Allen
NE	Brendlmann
SO	Currid

Además de las diferencias de coordinación, también puede haber diferencias en los dominios implícitos. Considere las tablas PERSONALDEVENTAS y REGIÓN de la figura 17-22, y suponga que alguien quiere producir un reporte de las ventas totales para cada región. Para hacerlo se necesita ejecutar el siguiente SQL:

```
SELECT      RegiónVentas, Sum(VentasTotales)
FROM        REGIÓN, PERSONALDEVENTAS
WHERE       REGIÓN.RegiónVentas = PESONALDEVENTAS.Región
GROUP BY    Región
```

Para los datos que se muestran, el resultado de esta consulta será una tabla que tenga tres renglones, uno por cada región de ventas (NE, SW y SO). Debido a que ninguna de las regiones SE o SW tenían un par en REGIÓN, se omitieron del join y las ventas de estas regiones no aparecerán en el resultado. Esto con frecuencia no es lo que el usuario desea.

Entre los años 2000 y 2001 estos negocios cambiaron sus territorios de ventas al combinar las regiones de ventas SE y SW que se deberían haber agregado al renglón SO en el resultado de la consulta. Ponga los términos en una base de datos, el dominio implícito de RegiónVentas y Región son diferentes. El dominio de RegiónVentas es el conjunto de regiones actuales; el dominio de Región es el nombre de la región y el momento en que ocurrió la venta.

Para las cantidades pequeñas de datos en las figuras 17-21 y 17-22 estos problemas son obvios. Sin embargo, si hubiera miles de renglones de datos estos problemas podrían escapársele al analista y proporcionaría información incorrecta para el proceso de toma de decisiones.

Con el fin de resolver este problema deben crearse metadatos que describan ambas cosas: la coordinación y los dominios de la fuente de datos. Estos metadatos se deben hacer fácilmente accesibles para los usuarios del data warehouse, y ellos necesitan ser capacitados sobre la importancia de ponderar estos asuntos.

INTEGRACIÓN DE HERRAMIENTAS. Otro serio problema para los depósitos de datos es la integración de varias herramientas que el usuario necesita. Los paradigmas de productos diferentes, en productos de diferentes categorías, usualmente son distintos. Los productos DBMS están orientadas a las tablas; los productos OLAP están orientados a los cubos; las hojas de cálculo, los paquetes de planeación financiera están

➤ FIGURA 17-23

Ejemplo de la diferencia conceptual entre hoja de cálculo y productos de la base de datos

Númerode-Empleado	Nombredel-Empleado	Númerode-Departamento	Administrador	Teléfonodel-Administrador	Códigodel-Departamento
1000	Wu	10	Murphy	232-1000	A47
2000	O'Connor	20	Joplin	244-7788	D87
3000	Abernathy	10	Murphy	232-1000	A47
4000	Lopez	20	Joplin	244-7788	D87

(a)

EMPLEADO (NúmerodeEmpleado, NombredelEmpleado, *NúmerodeDepartamento*)
DEPARTAMENTO (NúmerodeDepartamento, CódigodelDepartamento, *Administrador*)
ADMINISTRADOR (Administrador, TeléfonodelAdministrador)

(b)

orientados a un plan, y así sucesivamente. Como resultado, las interfaces del usuario en los productos son diferentes. Los usuarios pueden necesitar capacitación sustancial para aprender cómo usar varios productos de diversas categorías, y con frecuencia no tienen tiempo o disposición para aprender.

El proceso para importar y exportar datos para todos los productos de diferentes categorías puede ser difícil. Considere la hoja de cálculo de la figura 17-23(a) que contiene datos acerca de tres temas: Departamentos, Administradores y Empleados. Para importar esta hoja de cálculo a una base de datos normalizada, se necesitará ubicar cada uno de los temas en tablas separadas como las de la figura 17-23(b). Si no se hace la normalización, el resultado será una duplicación de datos, como se describió en el capítulo 15. Sin embargo, el usuario típico del data warehouse no entenderá la necesidad de la normalización, ni tendrá idea de cómo se hace.

Por último, cuando se adquieren productos de diferentes vendedores con frecuencia es difícil llegar a la fuente de los problemas cuando ocurren. Por ejemplo, el vendedor del producto que está exportando datos puede creer que un problema en el proceso de importar-exportar se debe al producto que está importando datos, y el proveedor del producto que está importando los datos puede afirmar lo contrario. Puesto que los proveedores por lo general no tienen experiencia en el uso de los productos de otros, tampoco tienen razones para fomentar las ofertas de otras compañías; el soporte técnico puede ser una pesadilla.

FALTA DE HERRAMIENTAS PARA LA ADMINISTRACIÓN DEL DATA WAREHOUSE. Aun cuando hay muchos productos y herramientas para la extracción de datos de las fuentes, y muchas herramientas para la consulta del usuario final, o para el análisis y el reporte de datos, actualmente se carece de herramientas para la administración del data warehouse. Si éste consta sólo de extracciones de las bases de datos relacionales, y si los problemas de coordinación y las diferencias de dominio se pueden resolver con capacitación y procedimientos, entonces se puede usar un DBMS cualquiera para administrar los recursos del data warehouse. Sin embargo, en la mayoría de los ejemplos éste no es el caso.

La mayoría de los data warehouses contienen extracciones de las bases de datos, archivos, hojas de cálculo, imágenes y fuentes de datos externas. Puesto que éste es el caso, estos recursos no los puede administrar un DBMS comercial, así que la organización que crea los datos del data warehouse debe escribir su propio software. Usualmen-

te, éste tiene un DBMS comercial como núcleo, y el personal del data warehouse desarrolla las funciones necesarias para administrar los recursos del mismo.

La administración de metadatos presenta otro problema similar. Pocos diccionarios de datos del DBMS tienen suficiente capacidad para recibir los metadatos que se necesitan del data warehouse. Como ya se mencionó, los usuarios no sólo necesitan saber qué está en el data warehouse, sino también de dónde viene, cuál fue su coordinación, cuáles fueron los dominios implícitos de los datos, qué suposiciones se hicieron cuando se extrajeron los datos, etc. El personal del data warehouse necesitará escribir su propio software de administración de metadatos para aumentar las capacidades del DBMS y otros productos del diccionario de datos con los que cuenta.

La redacción de un software de administración de datos es difícil y costosa. Una vez que está escrita, se le debe dar soporte. Los vendedores de programas de extracción y de análisis de datos cambiarán sus productos, y cualquier software desarrollado en casa que los usuarios necesiten que sea modificado para ajustarse a las nuevas interfaces. Además, los requerimientos de los usuarios cambiarán y será necesario agregar los nuevos programas que habrá que integrar al software de la organización del data warehouse.

REQUERIMIENTOS DE NATURALEZA AD HOC. Existen data warehouses para apoyar la toma de decisiones administrativas. Mientras que una buena parte de las decisiones administrativas son regulares y recurrentes, muchas otras son de una naturaleza especial para tal propósito. Preguntas como las siguientes no son comunes o recurrentes: ¿Deberíamos combinar territorios de ventas? ¿Vender una línea de producto? ¿Consolidar depósitos? ¿Adoptar nuevas ventas basadas en Internet y en estrategias de mercadotecnia?

Los sistemas computacionales, al igual que la burocracia, son lentos y costosos, relativamente inflexibles y trabajan mejor con necesidades que siguen una norma. Por esta razón existen sistemas tales como Excel para tareas tales como entradas de pedidos y el procesamiento de reservaciones. Sin embargo, es más difícil diseñar sistemas que respondan rápidamente a necesidades y requerimientos cambiantes. De esta manera, los data warehouses tienen más éxito en aplicaciones en las que la variación de requerimientos sigue un mismo patrón. Si un requerimiento nuevo es de estructura similar a uno anterior, es decir, “la consolidación de las ventas en la región norte será como el proceso que seguimos cuando se consolidó la región sur”, el data warehouse probablemente podrá responder de manera oportuna. De lo contrario, habrá que invertir tiempo, dinero y mucha angustia para satisfacer los requerimientos.

Data Mart

Debido a los retos que acabamos de describir, algunas empresas han decidido limitar el ámbito del warehouse en partes más manejables. Un **data mart** es una facilidad parecida a un data warehouse, pero con un dominio mucho más pequeño. El data mart se puede restringir a un tipo particular de datos, a determinada función de negocios, a una unidad de negocios específica, o a un área geográfica.

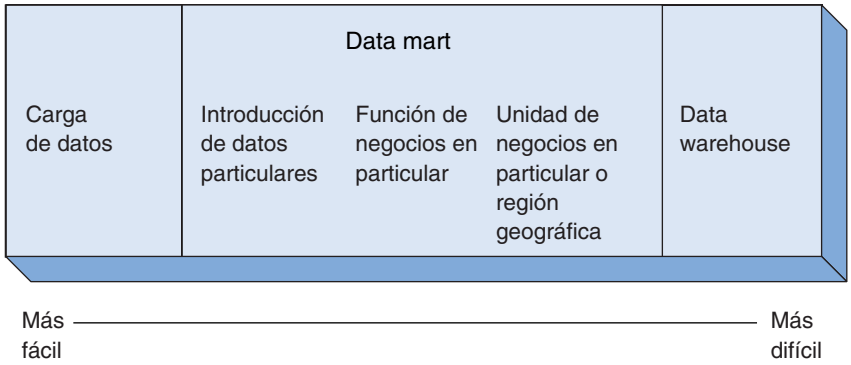
Restringir el data mart a un tipo específico de datos (por ejemplo, base de datos y hojas de cálculo) hace la administración del data warehouse más sencilla, y probablemente significa que se puede utilizar un producto DBMS desencapsulado para administrar el data warehouse. Los metadatos son más sencillos y también más fáciles de mantener.

El data mart que está restringido a una función particular de negocios, tal como el análisis de mercadotecnia, puede tener muchos tipos de datos y metadatos que mantener, pero todos éstos sirven para el mismo tipo de usuario. Las herramientas para la administración del data warehouse y para proporcionar datos a los usuarios, se pueden escribir con base en lo que a los analistas de mercadotecnia les gustaría tener.

Por último, el data mart que está restringido a unidades de negocios particulares o áreas geográficas, puede tener muchos tipos de entradas y de usuarios, pero la cantidad

➤ FIGURA 17-24

Constante de datos empresariales compartidos



de datos que administrar es menor que para toda la compañía. También serán muy pocos los requerimientos de servicio, así que los recursos del data warehouse se pueden asignar a muy pocos usuarios.

La figura 17-24 resume el ámbito de las alternativas para los datos compartidos sobre los cuales hemos hablado en este capítulo. La descarga de datos es la alternativa más simple y sencilla. Los datos se extraen de los sistemas operacionales y se envían a determinados usuarios, para propósitos específicos. La descarga de datos se realiza de manera regular y recurrente, así que la estructura de la aplicación es fija, los usuarios están bien capacitados y problemas tales como la coordinación y las inconsistencias de dominio rara vez ocurren porque los usuarios adquieren experiencia trabajando con los mismos datos. En el otro extremo, un data warehouse proporciona tipos de datos extensos y servicios para los requerimientos recurrentes y a la medida. El data mart se ubica en medio. Conforme nos movemos de izquierda a derecha en esta figura, las alternativas se vuelven más poderosas, pero también más costosas y difíciles de crear.

➤ ADMINISTRACIÓN DE DATOS

Los datos de una empresa son un recurso, como lo son las instalaciones, el equipo y los activos financieros. Se invierten tiempo y dinero para adquirir los datos y tienen utilidad más allá de las operaciones. La información derivada de los datos se puede utilizar para calcular la eficacia del personal, productos y programas de mercadotecnia, y para determinar las tendencias en las preferencias de los clientes, los patrones de compra, etc. También se puede usar para simular el efecto de los cambios en los productos, estrategias de ventas y territorios. Las listas de aplicaciones potenciales son tan largas que, de hecho, los datos a menudo sirven para establecer y mantener las ventajas competitivas de la empresa. Sin embargo, a medida que los datos se guardan en bases de datos operacionales, su utilidad es limitada.

Debido al valor potencial de las fuentes de datos organizacionales muchas empresas han establecido oficinas de administración de datos, cuya finalidad no sólo es guardar y proteger los datos, sino asegurarse de que sean utilizados eficazmente.

De cierta manera, la administración de datos es para éstos lo que el tesorero para el dinero. La responsabilidad de un tesorero es permitir que no sólo los bienes financieros sean protegidos y contabilizados, sino que se utilicen de manera eficaz. Guardar el dinero de una empresa en un baúl es una forma de protegerlo, pero no será utilizado de manera conveniente. En lugar de eso, se deben investigar formas que favorezcan los objetivos de la administración. De manera similar, para la administración de datos no basta simplemente con protegerlos, también implica tratar de aumentar la utilidad de los datos de la organización.

Necesidad de la administración de datos

Para entender la necesidad de la administración de datos reflexione en la analogía de una biblioteca universitaria típica, que contiene cientos de miles de libros, periódicos, revistas, informes gubernamentales, etc., los cuales no son útiles guardados en un librero. Para que lo sean deben estar a la disposición de las personas que se interesan en ellos o los necesitan.

Obviamente, la biblioteca debe tener algunos medios para describir sus colecciones, así los usuarios potenciales podrán saber con qué cuentan. A primera vista, esto puede parecer un problema trivial. Quizás usted diga: "Bueno, pues que hagan un catálogo de tarjetas"; pero se requiere mucho trabajo para hacerlo. ¿Cómo se deben identificar las obras de la biblioteca? ¿Cómo se deberían describir? Algo aun más básico, ¿qué constituye una obra? ¿Cómo podemos adaptar diferentes formas para la identificación de las obras (ISBN, sistema decimal Dewey, número de informe gubernamental)? ¿Cómo ayudamos a la gente a encontrar cosas que no se sabe que existen?

Surgen otras complicaciones. Suponga que la universidad es tan grande que tiene varias bibliotecas. En este caso, ¿se deben administrar las colecciones como un recurso? Además, algunos departamentos pueden tener sus propias bibliotecas. ¿Éstas son parte del sistema de la universidad? Muchos profesores tienen extensas bibliotecas personales. ¿Éstas deberían formar parte del sistema?

Retos de la administración de datos

Sin embargo, la analogía de la biblioteca no es lo suficientemente buena, ya que la administración de datos organizacionales es mucho más difícil que la de una biblioteca. Primero, no está claro qué es una "obra". Las bibliotecas contiene libros, revistas, periódicos, etc.; pero los datos organizacionales vienen en una gran variedad de formatos. Las organizaciones tienen registros tradicionales de datos, pero también documentos, hojas de cálculo, gráficas e ilustraciones, dibujos técnicos y archivos de audio y video. ¿Cómo se deberían describir todos éstos? ¿Cuáles son las categorías básicas de los datos organizacionales? Estas preguntas son importantes porque sus respuestas determinan cómo serán organizados, catalogados, protegidos, y cómo se podrá tener acceso a los datos.

La mayoría de las organizaciones tienen muchos nombres para una misma cosa. Por ejemplo, un número telefónico se puede describir como un NúmeroTelefónico, Teléfono, NúmerodeTeléfono, TeléfonoEmpleado, o TeléfonoDepto. ¿Cuál de estos nombres es preferible? Cuando un diseñador gráfico coloca un número telefónico en una forma nueva, ¿qué etiqueta debe usar?, o cuando un programador escribe un nuevo programa, ¿qué nombre debe usar para disponer del programa que tiene el número telefónico? Cuando un usuario quiere consultar el código de área de un cliente mientras desarrolla un análisis de tendencias de compra, ¿qué nombre debería utilizar en su consulta?

Hay varias formas para representar los elementos de los datos. Un número telefónico se puede representar como un entero de 10 dígitos, campo de texto de 10 dígitos, un campo de texto 13 dígitos en la forma *(nnn)nnn-nnnn*, un campo de texto de 12 dígitos en la forma *(nnn)-nnn-nnnn*, o incluso en otros formatos. ¿Cuál se debería permitir? ¿Cuál debería ser el estándar?

Sin embargo, las diferencias entre datos organizacionales y materiales de biblioteca son pequeñas si se comparan con la siguiente diferencia: las personas deben poder cambiar los datos organizacionales.

Considere qué pasaría en la biblioteca si la gente investiga en los libros, escribe en ellos, quita o agrega páginas, y después los regresa al estante. O peor aún, suponga que alguien investigó en tres libros, hizo cambios en ellos, los regresó y le dice al bibliotecario: "Cambie cualquiera de éstos o ninguno."

Puesto que los datos son bienes, o activos compartidos, se deben establecer límites a los derechos y responsabilidades del procesamiento. Por ejemplo, cuando un empleado se va de la compañía, sus registros no se pueden borrar inmediatamente; es necesa-

➤ FIGURA 17-25

Retos de la administración de datos

- Muchos tipos de datos
- Categorías básicas de datos que no son obvios
- Los mismos datos pueden tener muchos nombres
- Los mismos datos pueden tener muchas descripciones y formatos
- Los datos cambian con mucha frecuencia
- Los asuntos de política organizacional complican los asuntos operacionales

rio conservarlos durante varios años con fines de reportes organizacionales e impuestos. Por lo tanto, un departamento no puede eliminar datos sólo porque el departamento ya los desocupó. La oficina de administración de datos necesita ayudar a definir los derechos de procesamiento y las responsabilidades de los usuarios. Este papel es similar al que se describió para la administración de la base de datos en el capítulo 11; sin embargo, ahí el ámbito era una base de datos en particular, aquí es toda la organización.

Además de todos estos cambios orientados a la operactividad, hay asuntos organizacionales. Por ejemplo, los datos y los derechos de procesamiento pueden significar poder empresarial; por lo tanto, las variaciones en el control de datos pueden representar cambios en el poder. Así, detrás de las tareas de la administración de datos están asuntos políticos de todos tipos. El análisis de éstos está más allá del ámbito de este texto, pero realmente no son importantes. Los retos para la administración de datos se resumen en la figura 17-25.

Funciones de la administración de datos

Debido a los retos apenas descritos, la administración de datos es compleja. Para proteger datos, o activos, y al mismo tiempo aumentar su utilidad para la organización, se deben realizar varias funciones o tareas diferentes, las cuales, como se muestra en la figura 17-26, se pueden agrupar en diferentes categorías.

MERCADOTECNIA. Es la primera y la más destacada. La administración de datos es responsable de declarar su existencia y vender sus servicios al resto de la organización. Los empleados necesitan saber que existe una administración de datos y que hay políticas, estándares y normas para los datos organizacionales, así como las razones de éstos. Necesitan conocerlos para respetar las reglas de administración de datos, las directrices y las restricciones.

La administración de datos tiene que ser una función de servicio y los usuarios deben percibirla de esa manera. Así, las actividades de la administración de datos se deben comunicar a la organización de una forma positiva, y con una intención de servicio. Los empleados deben creer que tienen algo que ganar con la administración de datos. De otra manera, la función se traducirá en costos, no beneficiará a los usuarios y será ignorada.

ESTÁNDARES DE DATOS. Para que la organización de datos sea administrada de manera eficaz, se debe organizar coherentemente. Si cada departamento, función, o empleado eligiera una definición diferente para un elemento de datos, o los medios mediante los cuales se nombran o describen los elementos de los datos, el resultado sería un caos. Incluso sería imposible compilar un inventario de datos y dejarlo que se administre por sí sólo. Consecuentemente, muchas empresas deciden qué elementos de datos son importantes para que sean descritos de una manera estándar. Por ejemplo, la administración de datos puede decidir la importancia de cada elemento de datos en la empresa y la describirá por un nombre estándar, definición, descripción, conjunto de restricciones de procesamiento, etc. Una vez que se determina esta estructura, la siguiente pregunta es: ¿quién colocará los valores de estas descripciones estándar? Por

➤ FIGURA 17-26

Funciones de la administración de datos

Mercadotecnia
- Comunicar la existencia de la administración de datos a la empresa
- Explicar las razones de la existencia de normas, políticas y directrices
- Describir de una manera positiva los servicios que se proporcionan

Estándares de datos
- Establecer los medios estándares para la descripción de elementos de datos. Los estándares incluyen nombre, definición, descripción, restricciones de procesamiento, etcétera
- Establecer propuestas de datos

Políticas de datos
- Establecer una amplia política de organización de los datos. Los ejemplos son seguridad, propuesta de datos y distribución

Foro para la resolución de conflictos de datos
- Establecer procedimientos para dar un informe sobre los conflictos
- Proporcionar medios para escuchar todas las perspectivas y puntos de vista
- Tener autoridad para tomar decisiones y resolver conflictos

Recuperar la inversión de la empresa en datos
- Enfocar la atención en el valor de la inversión en datos
- Investigar nuevas metodologías y tecnologías
- Asumir una actitud positiva hacia la administración de la información

ejemplo, ¿quién decidirá el nombre estándar o la norma de las restricciones de procesamiento?

En muchas empresas el grupo de administración de datos no determina las descripciones estándar. Más bien, un departamento u otra unidad organizacional a cargo de la organización de estos registros de datos le asigna a cada elemento un **dato propuesta.** El proponente tiene la responsabilidad de establecer y mantener las definiciones organizacionales oficiales para los registros de datos que se le asignaron. Aunque la intención del grupo de administración de datos puede ser proponer algunos elementos de datos, la mayoría de las propuestas vienen de otros departamentos.

Usted puede encontrar el término *propietario de datos*, el cual se utiliza generalmente de la misma forma en que el término *datos propuestos* se emplea en este texto. Evitaremos el término porque implica un grado de propiedad que no existe. Tanto de forma legal como práctica, la empresa es la única propietaria de los datos. Aunque el o los grupos tienen una exigencia legítima de mayor grado de autoridad sobre los datos particulares de otros, estos grupos no son dueños de esos datos. En consecuencia, mejor utilizaremos el término *datos propuestos*.

Para resumir, los fundamentos de la administración de datos es un sistema de datos estandarizados. El grupo de administración de datos es responsable del trabajo con los usuarios y con la empresa para desarrollar un sistema laborable de estándares, que se debe documentar y comunicar a la organización por medios eficaces. También se deben establecer los procedimientos para evaluar el cumplimiento de las normas por parte de los empleados.

POLÍTICAS DE DATOS. Otro grupo de funciones de administración de datos se refiere a las políticas de datos. Para ilustrar la necesidad de estas políticas primero considere la seguridad de datos. Cada empresa tiene datos que son privados o sensibles, y la administración de datos es responsable de desarrollar un sistema de seguridad para protegerlos. Se necesitan hacer preguntas como las siguientes: ¿Qué esquemas de seguridad se deben colocar? ¿La organización necesita un sistema de seguridad de niveles

múltiples similar a uno militar? O, ¿será suficiente un sistema más simple? La política de seguridad también debe decidir qué se requiere de la persona que tiene acceso a datos delicados o confidenciales, y a qué acuerdos se debe llegar. ¿Qué hay en cuanto a los empleados de otras organizaciones? ¿Se deberán copiar esos datos confidenciales? ¿Cómo debería ser la capacitación de los empleados referente a seguridad? ¿Qué se debe hacer cuando se violan los procedimientos de seguridad?

Un segundo tipo de política de datos se refiere a los datos propuestos y a los derechos de procesamiento. ¿Qué significa un dato propuesto? ¿Qué derechos tiene un proponente que otro grupo no tiene? ¿Quién decide al que se convertirá en un proponente de datos y cómo se puede cambiar?

Un tercer ejemplo sobre la necesidad de política de datos es la distribución de éstos, como por ejemplo los datos oficiales que se distribuyen en más de una computadora y, si es así, ¿debería haber una copia oficial? ¿Qué procesamiento debería permitirse en la distribución de datos? ¿Los datos ya distribuidos deberían regresarse al almacén de datos oficiales? Sí es así, ¿qué revisión se tendría que hacer para validarlos antes de aceptarlos?

FORO PARA LA RESOLUCIÓN DE CONFLICTOS POR DATOS. Para que sean eficaces, los datos organizacionales debe ser compartidos, pero los seres humanos tenemos dificultades para compartir. Por lo tanto, la empresa necesita estar preparada para dirimir disputas sobre los datos propuestos, las restricciones de procesamiento y otros asuntos.

La primera responsabilidad de la administración de datos es establecer procedimientos para informar sobre los conflictos. Cuando las necesidades de un grupo de usuarios entran en conflicto con las de otro, ambos necesitan alguna forma en la cual dirimir sus conflictos de manera ordenada. Cuando el conflicto ha sido reconocido, los procedimientos establecidos deberán permitir a todas las partes involucradas plantear su caso. El personal de la administración de datos, quizás en conjunto con los proponentes de datos involucrados, debe resolver dicho conflicto. Este escenario supone que la organización ha otorgado a la administración de datos autoridad para tomar una decisión y hacer que se cumpla.

La administración de datos proporciona un foro para la resolución de conflictos que se aplican a toda la organización. La administración de base de datos también proporciona un foro para la resolución de conflictos, pero sólo para aquellos que pertenecen a una base de datos en particular.

AUMENTO AL RETORNO EN LA INVERSIÓN DE LOS DATOS DE LA ORGANIZACIÓN. Una función final de la administración de datos es la necesidad de aumentar las ganancias de la empresa por su inversión en datos. La administración de datos es el departamento que plantea preguntas tales como: ¿Estamos obteniendo lo que deberíamos de nuestra fuente de datos? Si es así, ¿qué más podemos lograr? Si no es así, ¿a qué se debe? ¿Vale la pena? Esta función implica a todos los demás: incluye mercadotecnia, establecimiento de estándares o políticas, resolución de conflictos, etc. Algunas veces esta función también significa investigar nuevas técnicas de almacenamiento, procesamiento, o presentación de datos; nuevas metodologías y tecnología, etcétera.

El cumplimiento bien realizado de esta función requiere una actitud *proactiva* con respecto al manejo de información. Algunas preguntas son importantes, si podemos utilizar la información para aumentar nuestra posición en el mercado, nuestra competitividad económica y el valor de nuestro capital. La administración de datos debe trabajar estrechamente con la planeación empresarial y el desarrollo de departamentos para anticiparse a las necesidades de nueva información, en lugar de reaccionar hasta que exista el requerimiento.

Por último, los datos deben estar disponibles para sus usuarios potenciales. Disponibilidad significa no sólo hacer que el acceso a los datos sea técnicamente posible para una persona muy motivada y especializada, sino que los datos se deben proporcionar a los usuarios de una manera que les resulte fácil de usar y en formatos aplicables en forma directa al trabajo que se debe realizar.

➤ RESUMEN

El teleproceso es la arquitectura clásica para el procesamiento de bases de datos para usuarios múltiples. Con éste, los usuarios operan terminales tontas, o computadoras personales que emulan a las terminales tontas. El programa de control de comunicaciones, los programas de aplicación, los DBMS y los sistemas operativos son procesados mediante una sola computadora personal centralizada. Debido a que todo el procesamiento se lleva a cabo a través de una sola computadora, la interfaz del usuario de un sistema de teleprocesamiento generalmente es sencillo y primitivo.

Un sistema cliente-servidor consta de una red de computadoras que con frecuencia se conectan mediante una LAN. En casi todos los casos las computadoras de los usuarios, llamadas clientes, son computadoras personales, y en la mayoría de los casos la computadora servidor también es una computadora personal, aunque se pueden utilizar macrocomputadoras. Los programas de aplicación se procesan en la computadora del cliente; el DBMS y la parte administradora de datos del sistema operativo residen en un servidor.

Los sistemas de archivos compartidos también implican redes de computadoras y, como las arquitecturas cliente servidor, por lo general constan de micros conectadas mediante LAN. La diferencia principal entre los sistemas de archivos compartidos es que la computadora servidor proporciona muy pocos servicios para las de los usuarios. El servidor, el cual se llama servidor de *archivos*, y no servidor de *base de datos*, proporciona acceso a archivos y a otros recursos. Por lo tanto, el DBMS y los programas de aplicación se deben distribuir a las computadoras de los usuarios.

Con un sistema de bases de datos distribuidas varias computadoras procesan la misma base de datos. Hay diferentes tipos de distribución de base de datos: particionada, no replicada; no particionada, replicada; y particionada, replicada. En general, mientras más grande sea el grado de partición y replicación, serán mayor la flexibilidad, la independencia y la confiabilidad. Al mismo tiempo, se incrementan los problemas de costo, control, y seguridad.

Los tres tipos de procesamiento de bases de datos distribuidos son: la carga de datos de sólo lectura, la actualización de la base de datos mediante una computadora designada, y la actualización de los datos de la base por medio de computadoras múltiples. Pueden ocurrir tres tipos de conflictos de actualización: pérdida de unicidad, pérdida de actualizaciones debido a transacciones concurrentes, y actualizaciones de datos eliminados. Si la actualización se permite en más de una computadora, estos problemas se deben resolver.

La coordinación de las transacciones atómicas distribuidas es difícil y requiere un commit de dos fases. El Servidor de Transacción distribuida OLE y los Enterprise Java Beans son dos tecnologías para resolver esos problemas.

Con el advenimiento de las poderosas computadoras personales, es posible descargar cantidades sustanciales de datos de la empresa a los usuarios para el procesamiento local. Éstos pueden solicitar y reportar los datos descargados utilizando productos DBMS en sus propias máquinas. En la mayoría de los casos no se les permite a los usuarios actualizar y regresar los datos, porque de lo contrario se crearían problemas de integridad. Aun cuando los datos cargados no se actualizan y se regresan, pueden ocurrir problemas de coordinación, consistencia, control de acceso, y posiblemente delito computacional. Se puede usar un servidor de la red para publicar los datos descargados.

El Procesamiento Analítico en Línea (OLAP, por sus siglas en inglés) es una nueva forma de presentar la información. Con éste, los datos se ven en cubos con ejes, dimensiones, medidas, rebanadas y niveles. Los ejes se refieren a la estructura física de la presentación, como renglones y columnas. Las dimensiones son características de los datos que se colocan en los ejes. Las medidas son los valores de los datos a ser desplegados. Las rebanadas son los atributos del cubo (dimensiones o medidas) que se mantendrán constantes en la presentación. El nivel es un atributo de una dimensión que describe su posición en una jerarquía.

El término *cubo* se utiliza tanto para referirse a la estructura semántica implícita que se usa para interpretar los datos, como para una materialización particular de da-

tos en esa estructura semántica. La figura 17-15(a) muestra una forma para definir la estructura implícita, y la figura 17-15(b) una forma para definir una materialización de una estructura cúbica.

ROLAP, MOLAP y HOLAP son tres de los siete enanos de OLAPlandia. Los defensores de ROLAP dicen que un DBMS relacional con extensiones es suficiente para satisfacer los requerimientos OLAP; los que apoyan a MOLAP dicen que se necesita un procesador especializado multidimensional; y los que proponen el uso de HOLAP quieren usar ambos.

Microsoft ha expandido OLE DB y ADO para OLAP. OLE DB para OLAP incluye un objeto data set; ADO MD tiene nuevos objetos para el procesamiento de objetos data set en una manera similar a los objetos rowset. El nuevo Servicio de Tabla Pivote ha sido agregado a Office 2000 y a Windows 2000. La arquitectura de Microsoft mueve mucho procesamiento OLAP a las computadoras de los clientes; aún no se sabe si esto será aceptable o no para el procesamiento de datos en los servicios de la empresa.

Un data warehouse es un almacén de datos de la empresa, diseñado para facilitar la toma de decisiones administrativas. Un data warehouse almacena extractos de bases de datos operacionales, archivos, imágenes, registros, fotos, datos externos y otros, y hace que dichos datos estén disponibles para los usuarios en un formato que les sea útil.

Los componentes de un data warehouse son las herramientas de extracción de datos, extractos de datos, metadatos, uno o más productos DBMS, herramientas de organización de los data warehouses desarrollados en casa, programas de reparto de datos, herramientas de análisis para el usuario, capacitación del usuario y asesoría de personal altamente capacitado. Los requerimientos típicos para un data warehouse incluyen consultas de estructura variable y reportes, agregación de datos para un usuario específico, trabajo de los datos, salidas gráficas e integración con programas de dominio específico.

Los data warehouses deben superar varios retos importantes; uno es que cuando se integran los datos se pueden desarrollar inconsistencias debido a las diferencias en el tiempo y el dominio. También, debido a las herramientas que se requieren en un data warehouse, éstas tendrán diferentes interfaces de usuarios y medios inconsistentes de importación y exportación de datos, y puede ser difícil obtener soporte técnico.

Otro reto es que hay una carencia de herramientas para la organización del propio data warehouse. La organización puede desarrollar sus propias herramientas para la organización de datos no relacionales y para el mantenimiento apropiado de los metadatos. Ese desarrollo es difícil y costoso. Por último, la naturaleza de muchos requerimientos en el data warehouse es a la medida; tales requerimientos son difíciles de satisfacer. Como resultado, algunas empresas han desarrollado depósitos data warehouse de un ámbito limitado llamado data mart.

Los datos son una ventaja organizacional importante, que puede dar soporte tanto a las operaciones como a la toma de decisiones administrativas. La finalidad de la oficina de administración de datos no sólo es proteger y guardar los activos de datos, sino asegurarse de que sean utilizados eficazmente. Una de las funciones más importantes de la administración de datos es documentar los contenidos de los activos de datos de la empresa. Ésta es una tarea complicada porque los datos pueden surgir en formatos diferentes en muchos lugares de la empresa. La administración de datos necesita ayudar a establecer normas organizacionales para nombres y formatos de registros de los datos, y también para definir los derechos del procesamiento organizacional y las responsabilidades. Por último, los datos son un activo y su uso puede significar poder; en consecuencia, la administración de datos debe encargarse de los asuntos organizacionales y políticos.

Las funciones específicas de la administración de datos incluyen sus servicios de mercadotecnia, establecimiento de normas de datos e identificación de proponentes de datos, asegurando que se establezcan políticas de datos adecuados y proporcionando un foro para la resolución de conflictos. Todas estas funciones están dirigidas al objetivo de aumentar las ganancias de la inversión de datos de la empresa.

➤ PREGUNTAS DEL GRUPO I

17.1 Enumere las arquitecturas que se utilizan para dar soporte a las bases de datos multiusuarios.

17.2 Esquematice la arquitectura de un sistema de teleprocesamiento. Nombre e identifique la(s) computadora(s) y los programas involucrados y explique qué computadoras procesan cuáles programas.

17.3 ¿Por qué la interfaz del usuario en aplicaciones de teleproceso por lo general es de carácter orientado y primitivo?

17.4 Haga un esquema de la arquitectura de un sistema cliente-servidor. Nombre e identifique la(s) computadora(s) y los programas implicados, y explique cuáles son los procesos de la computadora y cuáles los programas.

17.5 ¿Qué tipos de hardware de procesamiento se utilizan con los sistemas cliente-servidor?

17.6 ¿Cuántos servidores puede tener un sistema cliente servidor? ¿Qué restricciones se aplican a los servidores?

17.7 Esquematice la arquitectura de un sistema de archivos compartidos. Nombre e identifique la(s) computadora(s) y los programas involucrados y explique los procesos de la computadora y de los programas.

17.8 Explique cómo debería diferir el procesamiento de la siguiente consulta SQL en un sistema cliente-servidor en un sistema de archivos compartidos:

```
SELECT   NombreEstudiante, NombreClase
FROM     ESTUDIANTE, GRADO
WHERE    ESTUDIANTE.NúmeroEstudiante = GRADO.NúmeroEstudiante
AND      GRADO.Grado = 'A'
```

Suponga que la base de datos contiene dos tablas:

ESTUDIANTE (NúmerodeEstudiante, NombredelEstudiante, TeléfonodelEstudiante)
GRADO (NúmerodeClase, NúmerodeEstudiante, Grado)

También suponga que las llaves primarias y externas tienen índices.

17.9 Explique por qué los sistemas de archivos compartidos rara vez se utilizan para el procesamiento de las aplicaciones de transacciones de alto volumen.

17.10 Defina los términos *particionado* y *duplicado* con respecto a las aplicaciones de la distribución de base de datos.

17.11 Explique la diferencia entre un fragmento vertical y uno horizontal.

17.12 Explique las diferencias en los cuatro tipos de distribución de bases de datos en la figura 17-5.

17.13 Nombre y describa tres técnicas para el soporte del procesamiento de la base de datos distribuida.

17.14 Describa tres tipos de conflictos de actualización distribuida.

17.15 ¿ Cuál es la finalidad del commit de dos fases?

17.16 Resuma el problema de coordinación en el procesamiento de bases de datos descargadas.

17.17 Resuma el problema de consistencia en el procesamiento de las descargas de bases de datos.

17.18 Resuma el problema de control de acceso al procesamiento de las bases de datos cargadas.

17.19 ¿Por qué es un riesgo de delito computacional procesar las bases de datos descargadas?

17.20 Bosqueje los componentes de un sistema que use un servidor Web para publicar los datos descargados.

17.21 ¿Qué es un cubo OLAP? Dé otro ejemplo diferente al que muestra la figura 17-12.

17.22 Explique la diferencia entre un eje OLAP y una dimensión OLAP.

17.23 ¿Qué es la medida de un cubo OLAP?

17.24 ¿Qué significa el término *rebanada* con respecto a los cubos OLAP?

17.25 ¿Qué es un miembro de una dimensión? Mencione ejemplos sobre dimensiones de tiempo y ubicación.

17.26 Explique el uso de los niveles de la figura 17-12.

17.27 Explique la ambigüedad del término *cubo*.

17.28 ¿Cuál es el resultado de la expresión CROSSJOIN{Mary, Linda}, {navegación esquí}? De CROSSJOIIN {navegación, esquí}, {Mary, Linda}?

17.29 Proporcione una instrucción SELECT SQL para producir un cubo similar al de la figura 17-12, excepto que los renglones y columnas están al revés y la ubicación se presenta antes de Categoría (cuando se lee de izquierda a derecha).

17.30 Explique la diferencia entre los esquemas estrella y copo de nieve.

17.31 Defina ROLAP, MOLAP, y HOLAP.

17.32 Considerando sólo el análisis de este texto, ¿cómo se amplió OLE para OLAP?

17.33 ¿Qué se hace para establecer ADO MD y cuál es su función?

17.34 Defina *data warehouse*.

17.35 ¿Cuál es la similitud entre el data warehouse y el procesamiento de descarga de datos?

17.36 Enumere y describa los componentes de un data warehouse.

17.37 Explique qué significa cambiar la estructura de una consulta o reporte antes de que se cambien los datos en una consulta o reporte.

17.38 Mencione un ejemplo, diferente al de este libro, sobre la necesidad de un usuario de agregar datos.

17.39 Proporcione un ejemplo, diferente del de este libro, sobre la necesidad de los usuarios para trabajar sus datos.

17.40 Explique dos fuentes de inconsistencia de datos y dé un ejemplo, diferente al de este libro.

17.41 Resuma los problemas de tener herramientas que usan paradigmas diferentes y que son licenciadas por proveedores diferentes.

17.42 Explique qué herramientas de data warehouse se deben desarrollar en la propia empresa.

17.43 ¿Por qué la naturaleza *ad hoc* del data warehouse plantea un problema?

17.44 ¿Qué es un data mart y por qué una compañía lo desarrollaría?

17.45 Enumere y explique brevemente tres tipos de data marts.

17.46 Explique por qué los datos son un activo organizacional importante.

17.47 Describa varios ejemplos sobre los usos de datos, además de los sistemas operacionales.

17.48 ¿En qué se parece la administración de datos al trabajo de un tesorero?

17.49 Resuma brevemente la necesidad de la administración de datos.

17.50 Enumere y explique brevemente los retos de la administración de datos.

17.51 Describa la función de la mercadotecnia en la administración de datos.

17.52 ¿Qué papel desempeña la administración de datos con respecto a los estándares de datos?

17.53 Defina *proponentes de datos*.

17.54 ¿Cuál es la diferencia entre el proponente de datos y el propietario de los datos?

17.55 Resuma el papel de la administración de datos de acuerdo a la política de datos.

17.56 Explique qué está involucrado en el establecimiento de un foro para la resolución de conflictos.

17.57 ¿Cómo puede ayudar la administración a aumentar las ganancias de los bienes de datos de una organización?

➤ PREGUNTAS DEL GRUPO II

17.58 Considere que una compañía tiene un gerente de ventas nacionales y 15 vendedores regionales. Cada semana, el personal de ventas baja de la macrocomputadora los datos de las ventas y los usa para actualizar sus proyectos de ventas para el mes siguiente. Después, se conectan vía módem con un servidor de base de datos y almacenan sus proyectos de ventas en esa base de datos. Entonces, el gerente acumula los datos de las ventas en un pronosticador de toda la compañía. ¿Qué problemas, asuntos y dificultades pueden existir en esta situación en términos de coordinación, consistencia, control de acceso y delito computacional?

17.59 Considere los datos empresariales que existen en la escuela o universidad a la que asiste. ¿Desde su punto de vista se hace un buen uso de los activos de datos? ¿De qué manera puede usted identificar que los activos de datos se utilizan para algo más que el proceso operacional? Describa formas en las cuales piense que la escuela o universidad podría sacar ventaja de sus activos de datos en las áreas de:

- ➤ Reclutamiento de estudiantes
- ➤ Aumento de recursos
- ➤ Programas de planeación
- ➤ Asuntos estudiantiles
- ➤ Asuntos de los alumnos
- ➤ Otras áreas

PARTE VII

PROCESAMIENTO DE BASES DE DATOS ORIENTADA A OBJETOS

Esta parte consta de un capítulo que aborda la programación orientada a objetos y el almacenamiento con el ODBMS. Incluye un tutorial breve de la programación orientada a objetos y discusiones sobre Oracle objeto relacional, referente a las extensiones de objetos del SQL, denominadas SQL3, y la administración de los datos de objetos, denominada ODMG-93.

Esta parte complementa los análisis de los capítulos 15 y 16, con respecto a OLE DB, ADO, y JDBC. Aunque dichos capítulos presentaron los aspectos prácticos del uso de interfaces de objetos, con el fin de obtener los servicios de la base de datos, este capítulo presenta una vista más conceptual sobre el motivo y el propósito del procesamiento de bases de datos orientadas a objetos.

CAPÍTULO 18

Procesamiento de Bases de Datos orientadas a objetos

Este capítulo señala el almacenamiento persistente de objetos creados en lenguajes de programación como Java, C#, y C++. Como usted sabe, las bases de datos relacionales almacenan los datos en forma de tablas, renglones y columnas. De tal manera, las bases de datos relacionales no son adecuadas para almacenar objetos, ya que pueden contener estructuras complejas de elementos de datos y también apuntadores a otros objetos. Además, los objetos incluyen instrucciones ejecutables, o métodos, y para hacer persistentes a los objetos también deben proporcionar ciertos medios para almacenarlos.

Los productos DBMS de propósito especial, llamados **DBMS orientados a objetos (ODBMS,** o a veces **OODBMS**), se desarrollaron a principios de la década de 1990 para proporcionar un almacenamiento de objetos persistentes. Sin embargo, estos productos no se han comercializado con éxito debido a que necesitan convertir los datos al formato ODBMS. Las organizaciones casi no realizan dicha conversión porque es muy costosa y las ganancias no compensan la inversión.

Apenas se está desarrollando la programación orientada a objetos, sin embargo, aún no ha desaparecido la necesidad del almacenamiento de objetos persistentes. En respuesta, los proveedores de los DBMS tradicionales están aumentando las capacidades de sus productos para permitir el almacenamiento de objetos, así como el almacenamiento tradicional de datos relacionales. A tales productos se les llama **DBMS de objeto relacional,** y probablemente su uso aumentará en los próximos años. En particular, Oracle ha desarrollado diversos recursos para la modelación y almacenamiento de objetos.

Debido a que no asumimos que usted sea un programador orientado a objetos, este capítulo inicia con un esbozo de los términos y conceptos de la orientación a objetos. Después describiremos las alternativas para proporcionar el almacenamiento de objetos persistentes e ilustrar el soporte de Oracle para la persistencia de objetos. Finalmente, examinaremos dos estándares importantes de objetos: el SQL3 y el ODMG-93.

➤ ESBOZO DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

La **programación orientada a objetos** (Object-oriented programming, OOP) es una manera de diseñar y codificar programas. La OOP es substancialmente diferente a la programación tradicional, ya que es una nueva forma de pensar las estructuras de programación. En lugar de considerar a los programas como secuencias de instrucciones que se procesan, la OOP considera a los programas como conjuntos de estructuras de datos que tienen elementos de datos e instrucciones del programa.

Otra manera de entender la diferencia entre la programación tradicional y la OOP es que la programación tradicional se organiza primero de acuerdo a la lógica y después de conformidad con los datos, mientras que la OOP se organiza primero conforme a los datos y después a la lógica. Por ejemplo, para diseñar un programa tradicional que cree una orden, primero se desarrolla un diagrama de flujos, o un pseudocódigo de la lógica del proceso de la orden. Los datos que se procesarán quedarán documentados como parte de la lógica.

Cuando se desarrolla un programa orientado a objetos para crear una orden, primero se identifican los objetos involucrados, por ejemplo, ORDER (ORDEN), SALESPERSON (VENDEDOR), ITEM (ARTÍCULO), y CUSTOMER (CLIENTE). Posteriormente, dichos objetos se diseñan como elementos de datos y programas que se comparten o exponen entre sí. Finalmente, se crea un diagrama de flujos, o un pseudocódigo de los comportamientos de los objetos.

Terminología OOP

Un objeto OOP[1] es una **estructura encapsulada** que tiene **atributos** y **métodos.** El término *encapsulada* significa que es completo en sí mismo; los programas externos de un objeto no conocen nada de su estructura ni necesitan saberlo. A la apariencia externa de un objeto se le denomina **interfaz.** La interfaz está compuesta por los atributos y métodos que son visible desde el mundo exterior. A las estructuras interiores encapsuladas de un objeto se les denomina **implementación.** Los *atributos*[1] de los objetos OOP se encuentran ordenados en una estructura específica.

Los objetos OOP contienen *métodos*, o secuencias de instrucciones que el objeto ejecuta. Por ejemplo, un objeto OOP puede tener un método para desplegarse, otro para crearse, y otro más para modificar parte de su estructura. Considere un método que modifica un objeto CLIENTE. Este método, que es parte del objeto OOP, es un programa; para modificar el objeto OOP este programa contiene instrucciones para obtener datos del usuario o de otra fuente.

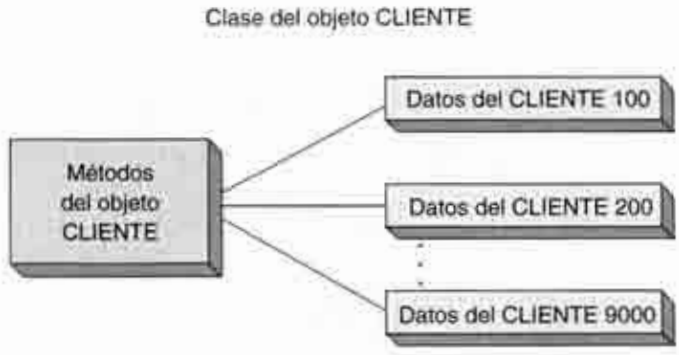
Los objetos OOP interactúan mediante la llamada de cada uno de sus métodos. Por ejemplo, el método de modificación CLIENTE invoca los métodos de otros objetos para obtener datos, ejecutar las modificaciones en sí mismo, y solicitar los servicios. Se llaman los métodos de otros objetos y luego se pueden invocar otros métodos, y así sucesivamente. Debido a que todos los objetos están encapsulados, ninguno puede o necesita saber la estructura de cualquier otro objeto. Esto reduce la complejidad y promueve la cohesión eficaz.

Muchos objetos tienen métodos en común. Para reducir la duplicación en la programación, se subclasifican los objetos de clases más generales. Un objeto, por ejemplo O_1, el cual es una subclase de otro objeto, digamos O_2, **hereda** todos los atributos y métodos del O_2. Por ejemplo, una aplicación puede tener una clase general EMPLEADO con dos subtipos, VENDEDOR e INGENIERO. Los métodos comunes para las tres clases de objetos tales como ObtenerNúmeroTelefónico se consideran como una parte de la clase EMPLEADO. Las subclases VENDEDOR e INGENIERO heredan dichos méto-

[1] Algunas veces se utiliza el término *propiedad* en lugar de atributo. En el estándar ODMG-93 se utiliza el término *propiedad* en vez de *atributo*, y este último se usa con un sentido más restrictivo, como veremos después. Cuando lea los términos *clase, tipo, propiedad* y *atributo* ponga atención en el contexto, ya que diversos autores utilizan estos términos de una manera un poco diferente. Aquí se utilizarán los términos de manera consistente con la fuente del tema.

➤ FIGURA 18-1

Ejemplo del objeto CLIENTE



dos. Por lo tanto, cuando un programa realiza una llamada a ObtenerNúmeroTelefónico en una subclase VENDEDOR o en una INGENIERO, está invocando el método ObtenerNúmeroTelefónico en EMPLEADO. Si los requerimientos de la aplicación son que los objetos INGENIEROs proporcionen los números telefónicos de una manera diferente a las de otros empleados; entonces INGENIERO puede tener una versión especial del método ObtenerNúmeroTelefónico como parte de su clase. Dicha versión especial se llamará cuando un programa invoque el método ObtenerNúmeroTelefónico en la clase INGENIERO. A esta característica se le llama **polimorfismo.**

En los análisis del OOP comúnmente se utilizan diversos términos. La estructura lógica de un objeto, su nombre, atributos, y métodos se llama **clase de objetos.** A un grupo de clase de objetos se le llama **bibilioteca de clase de objetos.** Y a las instancias de los objetos se les denomina **instancias de objetos,** o simplemente **objetos.**

Los objetos se crean al invocar a los constructores de objetos, los cuales son programas que obtienen la memoria principal y crean las estructuras necesarias para representar un objeto mediante un ejemplo concreto. Los destructores de objetos son programas que desligan los objetos y liberan la memoria. Los objetos pueden ser **transitorios** o **persistentes.** Un objeto transitorio sólo existe en la memoria volátil durante la ejecución de un programa. Cuando éste finaliza se pierde el objeto. Un objeto persistente es aquel que se guarda en un almacenamiento permanente, como por ejemplo un disco. Un objeto persistente sobrevive a la ejecución de un programa y se puede volver a leer en la memoria desde el almacenamiento.

El objetivo de un ODBMS es proporcionar un almacenamiento de objetos persistentes. Un objeto está conformado por datos y métodos; esto significa que un ODBMS, distinto a un DBMS tradicional, debe almacenar los programas de objetos y también los datos. Puesto que cada objeto de una clase específica tiene el mismo conjunto de métodos, éstos necesitan almacenarse sólo una vez por clase; en contraste, los elementos de los datos deben almacenarse una vez por cada caso de objetos. La figura 18-1 ilustra este punto. De hecho, hoy sólo algunos ODBMS proporcionan la persistencia del método, pero es probable que esto cambie en el futuro.

➤ EJEMPLO DE OOP

Las figuras 18-2 y 18-3 muestran una parte de una interfaz orientada a objetos y un ejemplo de un método. Con el fin de omitir los detalles poco significativos para este análisis, se escribe el código en una forma genérica que sea consistente con la programación del objeto, pero que no se encuentre en ningún lenguaje específico orientado a objetos. Considere que este código es un pseudocódigo de un programa de objetos.

La figura 18-2 muestra una parte de la interfaz de varios objetos utilizados para el procesamiento de órdenes. Cada objeto tiene un conjunto de métodos y atributos que expone. Cada objeto tiene un método constructor (Create) y uno destructor (Destroy).

➤ FIGURA 18-2

Ejemplos de objetos, métodos y atributos

Objeto	Métodos	Atributos
EMPLEADO	Crear Guardar Destruir ...	Número(R) Nombre(R) ...
VENDEDOR (subclase de EMPLEADO)	Crear Guardar Destruir Asignar(ORDEN, CódigoPostal) ...	ComisiónTotal(R) TotalPedido(RW) ...
CLIENTE	Crear Guardar Destruir Asignar(ORDEN) Buscar ...	Nombre(R) Teléfono(R) CódigoPostal(R) BalanceActual(RW) ...
ARTÍCULO	Crear Guardar Destruir Buscar(Número) Tomar(ORDEN, Cantidad) Poner(ORDEN, Cantidad) Find ...	Número(R) Nombre(R) Descripción(R) Precio ...
ORDEN	Crear Guardar Destruir Imprimir ...	Número(R) Fecha(R) Total(R) NombredelCliente(R) NombredelVendedor(R) ...

Algunos métodos tienen parámetros; por ejemplo, el método de asignación de VENDEDOR toma como sus parámetros un apuntador del objeto PEDIDO y un valor de código Postal. Los atributos que están marcados con (R) son de sólo lectura; los atributos marcados con (RW) pueden leerse o escribirse (cambiarse).

La notación de la figura 18-3 en la página 559 necesita explicación. Primero, las llaves {} representan observaciones. Aquí se utilizan para describir la función del código del programa que necesita escribirse, sin embargo no se consideró en este ejemplo debido a razones de brevedad y a que no es importante para este análisis. La instrucción Dim se utiliza para declarar las variables y sus tipos, como se hace en Basic. Se declara a ArtículodeLínea como una estructura que tiene los elementos de datos enlistados en los corchetes [ ]. El signo de admiración (!) se utiliza como separador entre un objeto y uno de sus métodos. De esta manera, CUSTOMER!Find se refiere al método Buscar del objeto CUSTOMER. Este carácter se pronuncia "bang". Un punto se utiliza como separador entre un objeto y uno de sus atributos. Por lo tanto, CustObj.ZipCode se refiere al atributo ZipCode del objeto señalado por CustObj.

Asimismo, en la figura 18-3 se utilizan dos palabras claves. **Nothing** es un valor especial de un apuntador a un objeto que se emplea para representar un valor nulo. En esta figura, la expresión:

```
if CustObj=Nothing
```

expresa la comparación del valor de la variable objeto CustObj con el apuntador del objeto nulo. **Me** es un apuntador objeto, el cual expresa que el objeto ejecuta al códi-

➤ FIGURA 18-3

Segmento de un programa orientado a objetos

```
ORDER!Create method
Dim CustObj as object, SPObj as object, ItemObj as object
Dim OrderTotal as Currency, OrderDate as Date, OrderNumber as Number
Dim LineItem as Structure
        [
        ItemNumber as Number,
        ItemName as Text(25),
        ItemQuantity as Count,
        QuantityBackOrdered as Count,
        ExtendedPrice as Currency
        ]

        (Get CustomerName from some source)
        Set CustObj = CUSTOMER!Find (CustomerName)
        If CustObj = Nothing then
                Set CustObj = CUSTOMER!Create(CustomerName)
        End If

        CustObj!Assign(Me)
        Set SPObj = SALESPERSON!Assign(Me, CustObj.ZipCode)

        (Get ItemNumber, Quantity of first ITEM from some source)
        Me.OrderTotal = 0
        While Not ItemNumber.EOF

                Me.LineItem!Create
                Me.LineItem.ItemNumber = ItemNumber

                Set ItemObj = ITEM!Find (ItemNumber)
                (process problem if ITEM not exist)

                Me.LineItem.ItemName = ItemObj.Name
                Me.LineItem.Quantity = ITEM!Take (Quantity)
                If Me.LineItem.Quantity <> Quantity Then
                        Me.LineItem.QuantityBackOrdered = Quantity - Me.LineItem.Quantity
                End If

                Me.LineItem.ExtendedPrice = Me.LineItem.Quantity* ITEM.Price
                Me.OrderTotal = Me.OrderTotal + Me.LineItem.ExtendedPrice

                ItemObj!Save
                Me.LineItem!Save
                (Get ItemNumber, Quantity of next ITEM from some source
                assume the source sets EOF to true when there are no more)
        While End

        SPObj.TotalOrders = SPObj.TotalOrders + Me.OrderTotal
        CustObj.CurrentBalance = CustObj.CurrentBalance + Me.OrderTotal

        SPObj!Save
        CustObj!Save
        ME!Save
End ORDER!Create
```

go. Cuando se ejecuta el código de la figura 18-3, lo ejecuta una instancia del objeto ORDER, debido a que es un método ORDER. Me se refiere al objeto particular ORDER que ejecuta el código.

El método ORDER!Crear comienza a obtener los datos sobre el nombre del cliente mediante la colocación del pedido; pero no importa cómo se obtiene este valor, debido a que no es significativo para nuestros propósitos —probablemente se obtiene de un cuadro de texto de una forma—. Después se invoca el método Buscar del objeto CUSTOMER para localizar un cliente que tenga un nombre específico y establecer un apuntador al objeto encontrado. Las particularidades de cómo se encuentra una ins-

tancia están encapsuladas en CUSTOMER!Encontrar, sin embargo, no se sabe cómo se realiza la selección, ni qué sucede si existe más de un cliente que tenga ese nombre, así como tampoco otros detalles similares.

El resultado de esta operación es el establecimiento de CustObj en el valor de un apuntador válido a un objeto CUSTOMER, o el valor especial Nada, el cual es el apuntador nulo. Si CustObj es nulo, entonces éste se establece en un apuntador a un objeto nuevo CUSTOMER creado por CUSTOMER!Crear. Como se muestra, el código supone que regresa a este punto un apuntador a un objeto válido del cliente. De hecho, debe volver a verificarse CustObj para ver si es nulo, pero, por cuestiones de brevedad, en el resto de este segmento del programa se omitirá toda esa verificación.

CUSTOMER muestra un método Asignar, el cual será llamado para asignarle un CUSTOMER a un ORDER. Debido al encapsulamiento no se sabe qué acciones realiza el método Asignar, pero se le llamará y se transmitirá el Me, que es el indicador del objeto en ejecución. De hecho, en esta aplicación el método Asignar es un ejemplo de lo que se denomina **repetición de llamada (callback).** ORDER!Crear se proporciona a sí mismo un apuntador al objeto CUSTOMER para que éste pueda rastrear los ORDERs que tenga. Una razón de hacer esto es para que un objeto CUSTOMER, que está a punto de destruirse, pueda llamar, antes de su anulación, a todos los ORDERs que se relacionen con él. De esta manera, el objeto ORDER puede destruir su apuntador al objeto CUSTOMER cuando dicho apuntador se vuelva inválido. Asimismo, existen muchos otros usos de los callback.

Posteriormente, ORDER!Crear asigna a SPObj a un vendedor objeto. El valor de ZipCode del cliente se está transmitiendo, por lo que puede parecer que el ZipCode está involucrado en el modo en que se asigna a un vendedor. Sin embargo, nuevamente, y debido al encapsulamiento, no se sabe cómo se realizó esto. Ocultando la metodología de asignación, el objeto SALESPEARSON puede cambiar libremente su método de asignación sin dañar la lógica de éste o de cualquier otro programa. De hecho, en la figura 18-3 ¡no se necesita cambiar ningún código, si SALESPEARSON.Asignación inició utilizando las fases de la luna para asignar a los vendedores!

La siguiente sección del código suple los valores de ArticulodeLínea. Observe que la palabra clave Me se utiliza para referirse a los elementos de datos locales. (De hecho, en la mayoría de los lenguajes OO, Me será asumida y no será necesaria; aquí la ponemos para ser más explícitos.) Al inicio de cada ciclo While (Mientras), se le asigna un almacenamiento a otro articulodeLínea en el método LineItem!Create.

El método ITEM!Tomar se utiliza para retirar artículos del inventario. Observe que la lógica supone que si se asignó un número de artículos menor a la cantidad requerida, entonces se reordenará el balance de los artículos. También, observe que el objeto cambiado ITEM se guarda después del procesamiento de cada línea. Asimismo, a diferencia de los objetos CUSTOMER o SALESPEARSON, ITEM no recibe ninguna repetición de llamada (callback). Esto significa que los objetos ITEM no saben qué ORDERs se conectan con ellos. Aparentemente, para esta aplicación, no es importante que los objetos ITEM sepan en qué ORDERs están utilizando sus datos.

El ciclo continúa hasta que ya no haya más artículos que colocar en el pedido. En este punto, se ajustan los totales en los objetos CUSTOMER y SALESPEARSON, se guardan éstos y la palabra Me.

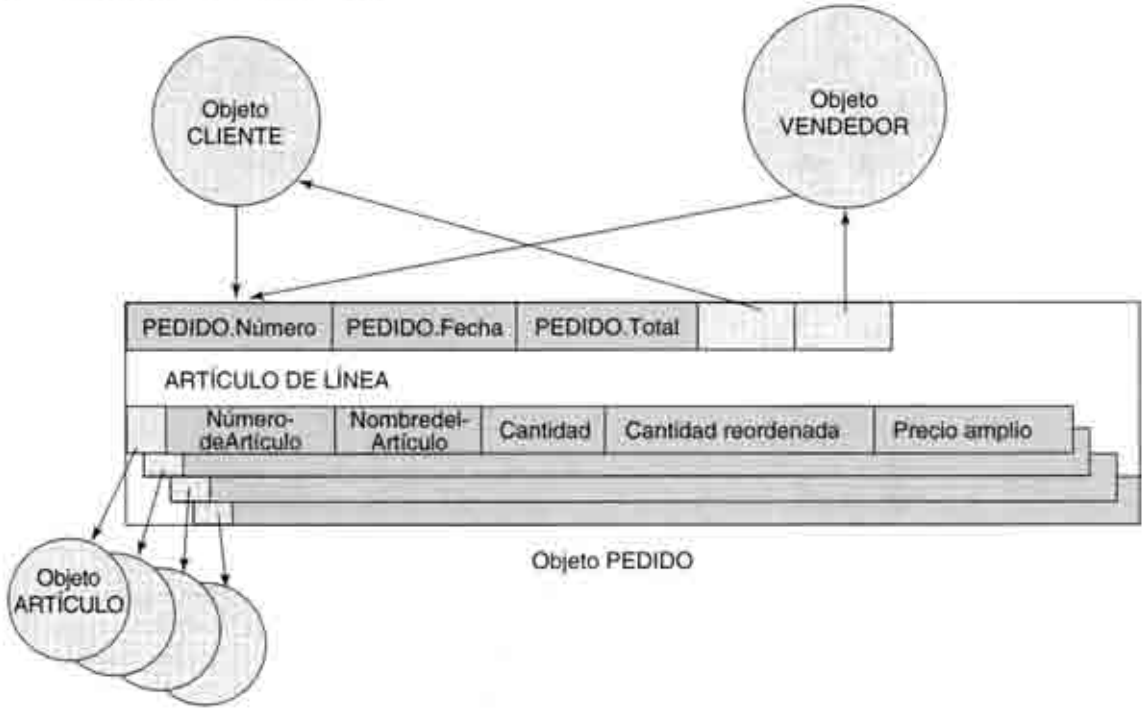
El segmento del código en la figura 18-3 es un código común orientado a objetos, y trae a colación diversos puntos importantes para los sistemas de bases de datos orientados a objetos. En particular, ¿cómo son los objetos para que se hagan persistentes?

➤ PERSISTENCIA DE OBJETOS

La figura 18-4 resume las estructuras de datos que existen después de que se ha creado un PEDIDO (ORDER). En el objeto pedido existen datos básicos, incluyendo PEDIDO .Número, PEDIDO.Fecha, y PEDIDO.Total (ORDER.Number, ORDER.Date, ORDER.Total), así como un grupo repetitivo para los artículos de línea, el cual contiene ArticuloNúmero, ArticuloNombre, Cantidad, CantidadOrdenadaNuevamente, y PrecioAmpliado (Item-Number, ItemName, Quantity, a QuantityBackordered y Extended Price). Además,

➤ FIGURA 18-4

Ejemplo de las estructuras de datos objetos



los datos base del pedido tienen un apuntador al objeto del CLIENTE asignado, un apuntador al objeto VENDEDOR asignado, y un apuntador a cada ARTÍCULO (ITEM) para cada artículo de línea. Estos apuntadores son parte de los datos del objeto PEDIDO. Para hacer a este objeto persistente, se deben almacenar todos estos datos. Además, aunque no se conoce su estructura, se debe almacenar cada objeto CLIENTE, VENDEDOR y ARTÍCULO. Los objetos CLIENTE y VENDEDOR también almacenan un apuntador de regreso a PEDIDO, como resultado de las repeticiones de las llamadas (los métodos !Asign) que se emitieron.

Los apuntadores poseen un problema en particular. En la mayoría de los lenguajes orientados a objetos, los apuntadores se encuentran en alguna forma de dirección almacenada en memoria. Dichas direcciones sólo son válidas durante la ejecución del programa; si éste termina y después se reinicia, las direcciones de los objetos serán diferentes. Por consiguiente, cuando se almacena un objeto, los apuntadores almacenados en la memoria necesitan transformarse en un identificador único permanente que sea válido durante el tiempo de vida del objeto, esté o no almacenado en la memoria. El proceso de la transformación de los identificadores permanentes en direcciones almacenadas en la memoria se llama **intercambio (swizzling).**

Finalmente, recuerde que un objeto se define como los valores de los datos más los métodos. Por lo tanto, para hacer persistente un objeto se deben guardar los métodos y los valores del objeto. Sin embargo, a diferencia de los valores de datos, cada objeto de una clase específica tiene los mismos métodos, por lo que sólo se necesitan almacenar los métodos una vez para todas las instancias del objeto en la clase del objeto. En la figura 18-5 se listan los requisitos para la persistencia del objeto.

➤ FIGURA 18-5

Tareas necesarias para la persistencia del objeto

- Guardar los valores de los datos de los casos del objeto
- Convertir los apuntadores de objetos almacenados en la memoria en identificadores únicos permanentes (intercambio)
- Guardar los métodos de la clase del objeto

➤ FIGURA 18-6

Empleo de un archivo de longitud determinada que soporte los datos del objeto

Número de registro	Código del registro	Contenido	Enlace
1	PEDIDO	PEDIDO 100 datos	4
2	VENDEDOR	VENDEDOR Jones datos	nulo
3	CLIENTE	CLIENTE 10000 datos	nulo
4	ARTÍCULO DE LÍNEA	Artículo de línea del PEDIDO 100 datos	5
5	ARTÍCULO DE LÍNEA	Artículo de línea del PEDIDO 100 datos	nulo
...	...	...	...

Los objetos pueden hacerse persistentes mediante el almacenamiento tradicional de archivos, un DBMS relacional, o un ODBMS. A continuación se considerará cada uno de éstos.

Persistencia de objetos mediante el almacenamiento tradicional de archivos

Los objetos pueden guardarse usando el almacenamiento tradicional de archivos; sin embargo, hacerlo le otorga una gran carga al programador. Considere los datos en la figura 18-4. El programador debe decidir crear un archivo que contenga métodos para todos los objetos, y un segundo archivo que contenga los datos para todos los objetos. Para llevar a cabo esto se necesita desarrollar una estructura generalizada para introducir los métodos y los datos en los archivos, y restaurarlos cada vez que sea necesario. La figura 18-6 muestra un ejemplo de dicho archivo sólo para el almacenamiento de los elementos de los datos. (Se necesitará desarrollar otro para almacenar los métodos.)

Para utilizar dicho archivo, el programador escribirá el código en los métodos de Guardar para empacar o desempacar datos objetos en estos registros, para encontrar los objetos solicitados, controlar el espacio disponible del archivo, etc. Asimismo, el programador necesitará idear e implementar algoritmos de intercambio (swizzling) y de no intercambio (de-swizzling). Además, existe un problema de secuencia. Todos los métodos se almacenan en archivos, incluyendo los que almacenan y leen métodos. ¿Cómo es el método que lee el primer método que se obtiene?

Todos estos problemas pueden superarse; durante muchos años se han resuelto en los subsistemas de procesamiento de archivos del sistema operativo. Pero, justamente ése es el punto. La programación es lenta, tediosa, arriesgada y difícil, y ya realizó el procesamiento tradicional de archivos. ¿Por qué se necesita realizar de nuevo dicha programación?

Debido a estos problemas, el almacenamiento tradicional de archivos es viable para la persistencia de objetos solamente cuando la aplicación tenga algunos objetos simples cuyas estructuras no cambien. Algunas aplicaciones comerciales entran en esta categoría.

Persistencia de objetos mediante el DBMS relacional

Otro enfoque sobre la persistencia de objetos es utilizar los productos comerciales del DBMS relacional. A diferencia del procesamiento tradicional de archivos, este enfoque le da una carga pequeña al programador, debido a que el DBMS controla las tareas de la administración básica de archivos, tales como la asignación de registros, indización, administración del espacio, etcétera. Las tareas de la administración de datos que el programador debe realizar son la definición de las estructuras relacionales para representar los objetos y escribir el código que interactué con el DBMS, con el fin de obtener y establecer objetos e intercambiar los apuntadores.

La figura 18-7 muestra las tablas necesarias para almacenar los objetos PEDIDO, CLIENTE, ARTÍCULODELÍNEA, VENDEDOR, y ARTÍCULO. Este diseño ya se ha visto anteriormente. El único elemento nuevo es una tabla que almacena los métodos de los objetos, la cual contiene un campo de notas que almacena el código del método.

➤ FIGURA 18-7

Relaciones necesarias para almacenar los objetos EMPLEADO, VENDEDOR, CLIENTE, ARTÍCULO y PEDIDO

EMPLEADO (Número, Nombre...)
VENDEDOR (*Número*, ComisiónTotal, PedidosTotales...)
CLIENTE (Nombre, teléfono, CódigoPostal, BalanceActual...)
ARTÍCULO (Número, Nombre, Descripción, Precio...)
ORDEN (*Número*, Fecha, Total, *VENDEDOR.Número, CLIENTE.Nombre*...)
ARTÍCULODELÍNEA (*PEDIDO.Número, ARTICULO.Número*, NombredelArtículo, CantidaddeArtículos, CantidadReordenada, PrecioAumentado...)
MÉTODOS (NombredelObjeto, NombredelMétodo, CódigodelMétodo)

Las bases de datos relacionales representan relaciones mediante llaves externas. Esto significa que el programador de la aplicación debe idear algunos medios para utilizar las llaves externas y hacer persistentes las relaciones. La manera más común de realizar esto es codificar la creación de una ID (Identificación) única en el método constructor del objeto. Esta ID se puede almacenar en la tabla básica del objeto y mostrarse como una propiedad de sólo lectura. Los objetos que necesitan enlazarse al objeto pueden guardar el valor de la ID. Con esta estrategia el problema se genera cuando se destruye un objeto, el objeto debe notificarlo a todos los que están conectados a él para que puedan eliminar el apuntador al objeto que está a punto de destruirse y considerar otra acción como apropiada. Ésta es una de las razones por las que se tienen repeticiones de llamadas como las que se mostraron en los métodos Assign (asignar) en la figura 18-3.

La ideología y el diseño de la orientación a objetos se encuentran en las relaciones dentro del contexto. Por lo tanto, cuando un objeto PEDIDO se asigna a sí mismo a un VENDEDOR, sólo se interesa en la parte de dicha relación. Si PEDIDO desea conectarse a muchos VENDEDORes, simplemente lo hace. El objeto PEDIDO desconoce si un VENDEDOR tiene una relación con uno o con muchos PEDIDOs. Dicho conocimiento está encapsulado en el objeto VENDEDOR y no es parte de la lógica PEDIDO.

Esta característica representa una ventaja y una desventaja, dependiendo de su punto de vista. Suponga que PEDIDO puede tener varios VENDEDORes y que un VENDEDOR puede tener muchos PEDIDOs. En lenguaje de bases de datos, PEDIDO y VENDEDOR tienen una relación M:N. Consecuentemente, en el mundo de las bases de datos relacionales, se define una tabla de intersección para mantener a los identificadores de PEDIDOs y VENDEDORes relacionados entre sí.

En el mundo de los objetos, el objeto VENDEDOR sabe que tiene muchos PEDIDOs, y el objeto PEDIDO sabe que tiene muchos VENDEDORes, pero ninguno de ellos sabe nada del otro. Por consiguiente, las estructuras de datos que sostienen la relación están separadas. PEDIDO contendrá el almacenamiento de muchos enlaces a VENDEDOR, y VENDEDOR contendrá el almacenamiento de muchos enlaces a PEDIDO. El conjunto de los enlaces estará separado uno del otro.

¿Esto importa? No, mientras no existan errores en el procesamiento de objetos. Pero existe un riesgo debido a que los enlaces de objetos están separados, pero no son independientes. Si el PEDIDO 1000 se enlaza a VENDEDOR A, entonces, por definición, VENDEDOR A se enlaza al PEDIDO 1000. En el mundo del DBMS relacional, puesto que la relación se sostiene en un renglón de una tabla de intersección, la eliminación de un lado del renglón borra automáticamente el otro lado. Pero en el mundo de los objetos se puede borrar un lado de la relación, pero el otro no. Por lo tanto, el PEDIDO 1000 puede enlazarse al VENDEDOR A, pero éste no puede enlazarse al PEDIDO 1000. Claramente, éste es un error y no se debería permitir que se presentara, pero es posible si las relaciones se definen desde una perspectiva puramente orientada a objetos.

El uso de un DBMS relacional para la persistencia de objetos le facilita el trabajo al programador, a diferencia del uso de las estructuras de archivos tradicionales. Sin embargo, aún existe la necesidad de que el programador convierta los objetos en un diseño relacional, escriba el SQL (u otro código), obtenga y coloque los objetos usando el DBMS, e intercambie (swizzle). El ODBMS está diseñado para llevar a cabo estas tareas.

➤ FIGURA 18-8

Aplicación del trabajo de desarrollo de la persistencia de objetos para tres alternativas

ODBMS	DBMS relacional	Procesamiento tradicional de archivos
• Invocar los métodos ODBMS para salvar	• Convertir las direcciones de la memoria en una ID permanente e invertir el intercambio (swizzling) • Definir las estructuras de datos relacionales • Crear el SQL (u otro código) • Implantar el SQL en el programa	• Convertir las direcciones de la memoria en una ID permanente e invertir el intercambio (swizzling) • Definir las estructuras de datos relacionales • Crear el código de persistencia de objetos • Invocar el código de persistencia de objetos • Introducir y extraer los objetos en las estructuras de archivos • Encontrar los objetos solicitados • Controlar la capacidad de archivos • Otras tareas de administración de archivos

Persistencia de objetos usando el ODBMS

La tercera alternativa para la persistencia de objetos es utilizar un ODBMS. Tales productos están diseñados especialmente para la persistencia de objetos y, por lo tanto, ahorran la mayor parte del trabajo al programador de la aplicación.

Un ODBMS está diseñado para integrarse con un lenguaje orientado a objetos. Debido a que no existe ninguna estructura especial, como el SQL, se necesita implantar una en el código de aplicación. Para el ejemplo de la figura 18-4 es posible que los métodos de salvado (save) sean, de hecho, métodos proporcionados por el ODBMS. Por consiguiente, el programador, al invocar el método de salvado, llama al ODBMS.

Además, los productos ODBMS incluyen un compilador (o están incluidos con el compilador, dependiendo de su punto de vista) que procesa el código de la fuente y automáticamente crea estructuras de datos en la base de datos de objetos para almacenar objetos. Por lo tanto, a diferencia de lo que sucede con la base de datos relacional, o el procesamiento de archivos, el programador orientado a objetos no necesita transformarlos en estructuras de relaciones o archivos; el ODBMS lo hace automáticamente.

Finalmente, debido a que los ODBMS están diseñados para la persistencia del objeto, se construye alguna forma de intercambio (swizzling). De esta manera, un código parecido al de la figura 18-3 no estaría enterado del problema. Un objeto obtiene un enlace con otro objeto, y ese enlace es válido todo el tiempo. Si el enlace toma diferentes formas, el programa no se enterará.

Esto nos lleva a una característica del ODBMS denominada memoria de un nivel. Con un ODBMS determinado, el programa (y, por lo tanto, el programador) no necesita saber si un objeto se encuentra o no en la memoria. Si el PEDIDO 1000 tiene un enlace con el VENDEDOR A, entonces el PEDIDO 1000 puede utilizar las propiedades expuestas del VENDEDOR A, sin haber verificado si dichos datos existen en la memoria, o sin haber realizado una lectura o una instrucción SQL. Si el VENDEDOR A se encuentra en la memoria, el ODBMS hace el enlace; de lo contrario, el ODBMS lee en la memoria al VENDEDOR A y luego hace el enlace.

La figura 18-8 compara el trabajo que se requiere con cada una de las tres alternativas para la persistencia de objetos. Obviamente, un ODBMS proporciona un beneficio sustancial al programador orientado a objetos, entonces ¿por qué no se utilizan en todos lados dichos productos? Consideraremos esta pregunta en la siguiente sección.

➤ PERSISTENCIA DE OBJETOS MEDIANTE ORACLE

Oracle ha extendido los dispositivos de sus productos de bases de datos para incluir el soporte para la modelación de objetos y el almacenamiento de objetos persistentes. Como mencionamos antes, dichas bases de datos se llaman bases de datos objeto-relacional.

Conforme lea este análisis, reflexione sobre las maneras en que Oracle ha implementado el pensamiento orientado a objetos en el modelo relacional. Aunque esta implementación en algunas ocasiones es delicada, permite que las organizaciones pasen gradualmente del almacenamiento de datos relacionales al de datos de objetos. Como

se mencionó, se rechazó el ODBMS orientado en forma única a objetos, el cual requiere el cambio abrupto de un paradigma a otro. De todas maneras, Oracle ha hecho un trabajo maestro al soportar su base de clientes actuales mientras amplía el producto al mundo de los objetos.

Este análisis se basa en la versión 8 de Oracle. Estas características y funciones ciertamente se ampliarán y mejorarán, y usted necesitará repasar la documentación de Oracle en las publicaciones más recientes que hablen de las últimas características del almacenamiento del objeto.

Tipos de objetos y colecciones

Para desarrollar un almacenamiento persistente para los objetos en Oracle, primero cree un TYPE (TIPO) que represente al objeto. Dicho tipo puede utilizarse posteriormente en una relación en cualquiera de las cuatro maneras diferentes. La más simple, llamada **objeto columna,** es la utilización del tipo de objeto para definir una columna de una tabla. Las otras maneras crean una colección de objetos de uno de los siguientes tres tipos: **arreglos de longitud variable, tablas anidadas, y objetos renglón.** Abordaremos cada uno en su momento.

OBJETOS COLUMNA. La siguiente instrucción define un tipo de objetos llamado *obj_Apartment*:

```
CREATE TYPE obj_Apartment AS OBJECT (
    BuildingName          VARCHAR2(25),
    ApartamentNumber      CHAR(4),
    NumberBedrooms        NUMBER)
/
```

Este tipo tiene tres columnas que utilizan los tipos de datos construidos con Oracle. (Recuerde que la coma manda al SQL Plus ejecutar la instrucción que acaba de introducirse. De ahora en adelante, se omitirá mostrarla al final de cada uno de estos ejemplos.)

La siguiente instrucción CREATE (CREAR) define un objeto de columna llamado Location que utiliza el tipo de objetos obj_Apartment:

```
CREATE TABLE PERSON (
    Name                  VARCHAR(50),
    Location              obj_Apartment)
```

Esta tabla puede ser consultada y procesada como cualquier tabla relacional. Sin embargo, la sintaxis para insertar y actualizar las instrucciones es ligeramente diferente. El siguiente SQL insertará un renglón en PERSON:

```
INSERT INTO PERSON (Name, Location) VALUES
('Selma Whitebread', obj_Apartment('Eastlake','206',2));
```

Observe el uso del nombre del tipo de datos en el enunciado de valores. La siguiente instrucción SQL actualizará un renglón:

```
UPDATE PERSON
SET Location = obj_Apartment('Eastlake', '412',3)
WHERE Name= 'Selma Whitebread';
```

De nuevo, observe el uso del nombre del tipo de datos. La figura 18-9(a) muestra el resultado de una consulta en esta tabla.

ARREGLOS DE LONGITUD VARIABLE. Los arreglos de longitud variable son una de las tres maneras de crear colecciones de tipos de objetos. Para entender el uso de dichos arreglos, suponga que deseamos crear una tabla que contenga los datos para un

➤ FIGURA 18-9

Ejemplo de instrucciones SELECT (SELECCIONAR) para estructuras de objetos: (a) selección de un objeto de columna, (b) selección de un arreglo variable, (c) selección de una tabla anidada, (d) selección de un objeto renglón

```
SQL> SELECT * FROM PERSON;

NAME
--------------------------------------------------
LOCATION(BUILDINGNAME, APARTMENTNUMBER, NUMBERBEDROOMS)
--------------------------------------------------------------------------------
Lynda James
OBJ_APARTMENT('Eastlake', '206 ', 2)

Selma Whitbread
OBJ_APARTMENT('Eastlake', '444 ', 3)
```

(a)

```
SQL> SELECT * FROM BUILDING1;

BUILDINGID NAME
---------- --------------------------------------------------
UNITS(APARTMENTNUMBER, NUMBERBEDROOMS)
--------------------------------------------------------------------------------
         1 Eastlake
APARTMENT_LIST1(APT_UNIT('100  ', 1), APT_UNIT('200  ', 2), APT_UNIT('300  ', 1))

         2 Westview
APARTMENT_LIST1(APT_UNIT('101  ', 1), APT_UNIT('201  ', 2), APT_UNIT('301  ', 1))
```

(b)

```
SQL> SELECT * FROM BUILDING2;

BUILDINGID NAME
---------- --------------------------------------------------
UNITS(APARTMENTNUMBER, NUMBERBEDROOMS)
--------------------------------------------------------------------------------
         1 Eastlake
APARTMENT_LIST2(APT_UNIT('100  ', 1), APT_UNIT('200  ', 2), APT_UNIT('300  ', 1))

         2 Westview
APARTMENT_LIST2(APT_UNIT('101  ', 1), APT_UNIT('201  ', 2), APT_UNIT('301  ', 1))
```

(c)

```
SQL> SELECT * FROM APARTMENTS;

BUILDINGNAME                   APAR NUMBERBEDROOMS
------------------------------ ---- --------------
Westview                       333               2
Westview                       235               2
```

(d)

edificio de departamentos. Queremos almacenar un valor de la llave sustituta, el nombre del edificio, y una lista de los departamentos del edificio.

Para hacerlo, primero creamos un objeto para departamento y después lo asignamos a un arreglo de longitud variable, como se muestra a continuación:

```
CREATE TYPE Apt_Unit AS OBJECT (
    ApartmentNumber     char(5),
    NumberBedrooms      int);
CREATE    TYPE          APARTMENT_LIST1 AS VARRAY(50) OF APT_Unit
```

En este caso, el tipo APARTMENT_LIST1 puede tener hasta 50 elementos del tipo de objetos Apt_Unit.

La siguiente instrucción creará una tabla que utiliza este arreglo de longitud variable:

```
CREATE    TABLE         BUILDING1 (
    BuildingID          NUMBER,
    Name                VARCHAR2(50),
    Units               APARTMENT_LIST1);
```

Ahora, para insertar datos en la tabla, debemos utilizar el nombre del arreglo y de los elementos en dicho arreglo, como se muestra a continuación:

```
INSERT INTO BUILDING1 (BuildingID, Name, Units) VALUES
    (1, 'Eastlake',
    APARTMENT_LIST1  (Apt_Unit ('100', 1),
                      Apt_Unit ('200', 2),
                      Apt_Unit ('300', 1)));
```

Una instrucción normal SELECT trabajará para obtener los valores de todas las columnas, siempre y cuando la cláusula WHERE no se refiera a elementos en Apt_Unit. La figura 18-9(b) muestra los resultados de una instrucción SELECT* para todos los renglones.

Sin embargo, si usted sólo desea obtener los valores de Apt_Unit, o desea utilizar los elementos de Apt_Unit en una cláusula WHERE, deberá revertir la consulta como se muestra a continuación:

```
SELECT ApartmentNumber
FROM TABLE (
    SELECT UNITS
    FROM BUILDING1
    WHERE Name='Eastlake')
WHERE ApartmentNumber>100;
```

Esta consulta selecciona un ApartmentNumber de UNIDADES, el cual es el arreglo de longitud variable. La tabla EDIFICIO se procesa como una subconsulta. El resultado será una tabla con una columna de ApartmentNumber con dos renglones: 200 y 300.

Con la versión 8 de Oracle, usted no puede actualizar o eliminar los renglones individuales dentro de un arreglo variable usando las instrucciones UPDATE o DELETE. Debe escribir un procedimiento PL/SQL para hacer un ciclo a través del arreglo. Si quiere utilizar las instrucciones UPDATE y DELETE para llevar a cabo este propósito, y crear una tabla anidada, como se describe a continuación.

TABLAS ANIDADAS. Las tablas anidadas se definen casi de la misma manera que los arreglos de longitud variable. La diferencia entre ellos es que mientras los datos del arreglo de longitud variable se almacenan junto con la tabla en la que se definen, los datos de la tabla anidada se almacenan en una tabla por separado. Para crear la tabla EDIFICIO mediante arreglos anidados se utiliza:

```
CREATE TYPE APARTMENT_LIST2 AS TABLE OF Apt_Unit;
/
CREATE    TABLE          BUILDING2 (
    BuildingID           NUMBER,
    Name                 VARCHAR2(50),
    Units                APARTMENT_LIST2)
    NESTED TABLE Units STORE AS UNITS_TABLE;
```

La única diferencia de la sintaxis de vararray es que debe nombrarse la tabla anidada. Aquí se llama UNITS_TABLE.

La figura 18-9(c) muestra el resultado de una instrucción SELECT* en todas las filas de BUILDING2; observe que este resultado es idéntico al del ejemplo del arreglo variable. Las instrucciones de insertar y consultar utilizadas con tablas anidadas también son idénticas a las que se usan para los arreglos variables:

```
INSERT INTO BUILDING2 (BuildingID, Name, Units) VALUES
    (1, 'Eastlake',
    APARTMENT_LIST2 (Apt_Unit ('100', 1),
                     Apt_Unit ('200', 2),
                     Apt_Unit ('300', 1)));
```

y

```
SELECT ApartmentNumber
FROM TABLE (
    SELECT UNITS
    FROM BUILDING2
    WHERE Name='Eastlake')
WHERE ApartmentNumber>100;
```

Sin embargo, como afirmamos anteriormente, usted puede actualizar y eliminar los elementos en una tabla anidada:

```
UPDATE TABLE (
    SELECT          Units
    FROM            BUILDING2
    WHERE           Name='Eastlake')
SET NumberBedrooms=5
WHERE ApartmentNumber=100;
```

y

```
DELETE FROM TABLE (
    SELECT          Units
    FROM            BUILDING2
    WHERE           Name='Eastlake')
WHERE ApartmentNumber=100;
```

Como puede apreciar, los arreglos de longitud variable y las tablas anidadas son muy similares, pero también tienen diferencias. Como se estableció anteriormente, una diferencia es que UPDATE y DELETE sólo trabajan junto con tablas anidadas. Adicionalmente, los arreglos de extensión variable tienen un tamaño máximo, pero las tablas anidadas, no. Asimismo, Oracle almacena los datos del arreglo de longitud variable alineados con la tabla, pero almacena por separado los datos de la tabla anidada. Finalmente, se mantiene el orden de los renglones en un arreglo de longitud variable; con las tablas anidadas puede cambiar el orden de los renglones conforme se agreguen nuevos renglones a la tabla anidada.

OBJETOS RENGLÓN. Los objetos renglón son una cuarta manera de utilizar tipos de objetos en tablas. Una tabla de objetos renglón simplemente es una tabla que úni-

camente contiene objetos. Para fines de este ejemplo, defina obj_Apartment, como se hizo anteriormente:

```
CREATE TYPE obj_Apartment AS OBJECT (
    BuildingName        VARCHAR2(25),
    ApartmentNumber     CHAR(4),
    NumberBedrooms      NUMBER);
```

Entonces lo siguiente creará una tabla de obj_Apartments:

```
CREATE TABLE APARTMENTS OF obj_Apartment;
```

Las selecciones, inserciones, actualizaciones, y eliminaciones de filas en esta tabla pueden llevarse a cabo mediante la sintaxis SQL regular. La figura 18-9(d) muestra una instrucción común SELECT. Por ejemplo, para insertar un nuevo renglón, utilice:

```
INSERT INTO APARTMENTS (BuildingName, ApartmentNumber,
    NumberBedrooms)
VALUES ('Westview', '333',2);
```

Lo siguiente actualizará un renglón en departamentos (apartments):

```
UPDATE apartments
    Set numberbedrooms=5
    Where ApartmentNumber='100';
```

Finalmente, para eliminar un renglón, utilice:

```
DELETE FROM APARTMENTS
WHERE ApartmentNumber='100';
```

Objetos Oracle

Hasta este punto, hemos mostrado cómo definir los tipos de objetos y utilizarlos como elementos con las tablas. Estas técnicas agregan estructuras del objeto en las relaciones; las relaciones resultantes pueden procesarse mediante las variaciones del SQL. Sin embargo, Oracle proporciona otra perspectiva, una que agrega estructuras relacionales en los objetos. El SQL no se puede usar con estas estructuras. En lugar de eso, son objetos que se almacenarán en una base de datos, pero deben manipularse mediante programas orientados a objetos.

DEFINICIÓN DEL TIPO OBJETO. La figura 18-10 muestra las definiciones del objeto de Oracle para el procesamiento de Pedido que se ilustra en la figura 18-4. Como se mostró anteriormente, la instrucción CREATE TYPE se utiliza para definir las estructuras de objetos y los tipos que define el usuario. En esta figura, las dos primeras definiciones de tipo se utilizan para declarar un tipo de direcciones, llamado obj_ADDRESS, que ha definido el usuario y un tipo de arreglos variables (varray) llamado obj_PHONE_LIST de una extensión máxima de cinco. Ahora, estos dos tipos pueden utilizarse en las instrucciones CREATE TYPE AS OBJECT de la misma manera en que se utilizaron los tipos con las instrucciones de la creación de la tabla.

Las siguientes definiciones en la figura 18-10 son de obj_SALESPEARSON y obj_CUSTOMER (obj_VENDEDOR y obj_CLIENTE). Ambos utilizan los tipos definidos por el usuario obj_ADDRESS y obj_PHONE_LIST (obj_DIRECCIÓN y obj_TELÉFONO_LISTA). Este uso significa que los objetos obj_SALESPEARSON y obj_CUSTOMER tienen atributos llamados Street, City, State, Zip y Country (Calle, Ciudad, Estado, Código postal y País). Asimismo, cada uno tiene un arreglo de longitud variable de números telefónicos.

La siguiente instrucción CREATE TYPE está vacía; se utiliza para informar al analizador de tipos Oracle que habrá un objeto obj_ITEM (obj_ARTÍCULO) definido subsecuentemente. Esta instrucción permite una definición de tipos como la siguiente de obj_LINEITEM (obj_ARTÍCULODELÍNEA) para utilizar el símbolo obj_ITEM (obj_ARTÍCULO), aunque aún no se haya definido el tipo obj_ITEM (obj_ARTÍCULO).

Como se muestra en la figura 18-4, la definición para obj_LINEITEM incluye ItemNumber, Quantity, QuantityBackordered y ExtendedPrice. Sin embargo, también

➤ FIGURA 18-10

Definiciones de objetos Oracle

```
CREATE TYPE obj_ADDRESS AS OBJECT (
        Street          VARCHAR2(50),
        City            VARCHAR2(50),
        State           VARCHAR2(2),
        Zip             VARCHAR2(10),
        Country         VARCHAR2(15)
)
/
CREATE TYPE obj_PHONE_LIST AS VARRAY(5) OF VARCHAR2(12)
/
CREATE TYPE obj_SALESPERSON AS OBJECT (
        SalespersonID NUMBER,
        Name            VARCHAR2(50),
        Address         obj_ADDRESS,
        PhoneNums       obj_PHONE_LIST
)
/
CREATE TYPE obj_CUSTOMER AS OBJECT (
        CustomerID      NUMBER,
        Name            VARCHAR2(50),
        Address         obj_ADDRESS,
        PhoneNums       obj_PHONE_LIST
)
/
CREATE TYPE obj_ITEM
/
CREATE TYPE obj_LINEITEM AS OBJECT (
        ItemNumber              NUMBER,
        ItemRef                 REF obj_ITEM,
        Quantity                NUMBER,
        QuantityBackOrdered NUMBER,
        ExtendedPrice           NUMBER

)
/
CREATE TYPE list_LINEITEM AS TABLE OF obj_LINEITEM
/
CREATE TYPE obj_ITEM AS OBJECT (
        ItemNumber              NUMBER,
        ItemName                VARCHAR2(25),
        Price                   NUMBER
)
/
CREATE TYPE obj_ORDER AS OBJECT (
        OrderNumber             NUMBER,
        OrderDate               DATE,
```

➤ FIGURA 18-10

(Continuación)

```
        LineItems          list_LINEITEM,
        ShipToAddress      obj_ADDRESS,
        Customer           REF obj_CUSTOMER,
        Salesperson        REF obj_SALESPERSON,

MEMBER FUNCTION totalItems RETURN NUMBER
)
```

incluye una definición de un atributo apuntador de referencia. ItemRef se define como REF obj_ITEM. Esto significa que este atributo contendrá un valor que proporciona el sistema, el cual se refiere a un artículo en particular. Esta referencia será válida sin importar si el artículo referido se encuentra en la memoria principal o si está almacenado en disco. Si se requiere hacer algún intercambio (swizzling), Oracle lo hará tras bambalinas.

Por supuesto, el programa de aplicación debe asignar un valor a ItemRef. Una manera de hacerlo es mediante una instrucción SQL como la siguiente:

```
INSERT INTO ItemRef
SELECT REF(itemPtr) FROM obj_ITEM itemPtr
WHERE itemPtr.ItemNumber= 10000
```

Lo anterior supone que sólo un artículo tiene un ItemNumber de 10000.

Quizá se pregunte cuál es la diferencia entre los tipos de datos REF y las llaves externas. La primera diferencia es que los valores REF se ocultan de los usuarios y no tienen ninguna importancia para éstos. Por lo tanto, dichas referencias, al igual que las llaves sustitutas, no tienen necesidad de modificaciones en cascada. Sin embargo, si se elimina el obj_ITEM referido, aquí el valor será inválido. Depende del programa probarlo para validarlo antes de su uso.

Segundo, dichas referencias indican objetos, no renglones en tablas. El objeto referido puede tener por sí mismo una estructura de datos compleja y, como usted verá, tendrá métodos. Asimismo, Oracle proporciona una biblioteca de clases que facilita la manipulación de tales referencias. Una diferencia final es que tales referencias son unidireccionales. El obj_ITEM al cual se hace referencia puede o no tener un apuntador de regreso al obj_LINEITEM y, de hecho, en este ejemplo no lo tiene. Esto significa que podemos navegar del obj_LINEITEM al obj_ITEM, pero no en sentido contrario.

En la siguiente instrucción, list_LINEITEM se define como una tabla de obj_LINEITEM. Esto es similar a la definición anterior de APARTMENT_LIST2. La definición de tipos final describe el obj_ORDER. A diferencia de la definición de BUILDING2, la cual era una tabla, obj_ORDER se define como un objeto. Debido a que es un objeto, el atributo de la tabla anidada LineItem, la cual se refiere al tipo list_ LINEITEM, no necesita tener una tabla anidada definida para éste.

A medida que examine la figura 18-10 tenga presente que se están definiendo los miembros de los datos de un objeto, y no las tablas relacionales o algo parecido a éstas. Las estructuras de datos en este objeto sólo serán procesadas por los métodos del objeto.

La última sección de la definición del tipo de objetos es definir la interfaz de los métodos de este objeto. En este caso, el objeto sólo tiene un método llamado ArticulosTotales, el cual regresa un parámetro simple del tipo NUMBER. En un ejemplo más realista existirán muchos métodos definidos.

DEFINICIÓN DEL MÉTODO DEL OBJETO. La figura 18-11 muestra un ejemplo del método del objeto de Oracle. El propósito de este método es repetirse a lo largo de los LineItems y totalizar el precio ampliado de cada uno. La función comienza decla-

➤ FIGURA 18-11

Ejemplo del método de Oracle

```
CREATE OR REPLACE TYPE BODY obj_ORDER AS

  MEMBER FUNCTION totalItems RETURN NUMBER IS
      itemPtr              obj_ITEM;
      orderTotal           NUMBER :=0;
      i                    INTEGER;

      BEGIN

        FOR i in 1..SELF.LineItems.COUNT LOOP
            UTL_REF.SELECT_OBJECT(LineItems(i).ItemRef, itemPtr);
            orderTotal := orderTotal + SELF.LineItems(i).Quantity * itemPtr.Price;
        END LOOP;
      RETURN orderTotal;

      END;
END;
```

rando tres variables, y después empieza un ciclo FOR (PARA) sobre el conjunto de LineItems. La instrucción:

```
FOR i in 1..SELF.LineItems.COUNT LOOP
```

significa iniciar la variable i en 1, para procesar las instrucciones a lo largo del bloque que termina en END LOOP, agregar 1 a i, y repetir estas instrucciones hasta que i sea mayor que la cuenta de renglones en el atributo ArticulosdeLínea (LineItems).

La instrucción:

```
UTL_REF.SELECT_OBJECT(LineItems(i).ItemRef, itemPtr)
```

invoca una clase que se proporciona en una biblioteca de clases de Oracle. El objetivo de esta función es establecer el valor de artículoPtr para un apuntador válido de objetos del objeto obj_ITEM, al cual hace referencia ItemRef en el renglón actual.

Después de que se ha ejecutado esta instrucción, ItemPtr se puede emplear para referirse a cualesquiera de las propiedades de obj_ITEM. Esto se hace en la siguiente instrucción donde artículoPtr.Price tiene el valor de Price en el obj_ITEM al cual hace referencia al renglón actual.

No se preocupe si todo esto no tiene sentido. Haga un esfuerzo por entender la función y el propósito, en lugar de enfocarse en los detalles de las instrucciones de la figura 18-11. A partir de este análisis, usted debe entender las características generales y la naturaleza de los objetos de Oracle, e imaginarse cómo trasladan la administración de bases de datos tradicionales relacionales uno o dos pasos más en la dirección de la programación orientada a objetos.

➤ ESTÁNDARES ODBMS

Diversos grupos han estado trabajando en la definición de un estándar de base de datos orientada a objetos que pueda utilizarse como base para la construcción de los productos ODBMS. Aquí, examinaremos el trabajo de dos de dichos grupos. El primero es

una combinación de los comités ANSI e ISO (Organización Internacional de Estándares) que se han enfocado a extender el estándar SQL92 para el procesamiento de objetos. El segundo grupo es un consorcio de proveedores de objetos de bases de datos y otras partes interesadas que se basan en otro estándar importante en la industria, el modelo del objeto común y el lenguaje de diseño de interfaces del grupo de administración de objetos y el lenguaje de diseño de interfaces. Como usted quizás espera, el primer estándar comienza con una perspectiva de la base de datos y se dirige al pensamiento de objetos. El segundo estándar comienza con una perspectiva del objeto y se dirige al pensamiento de la administración de los datos. Ambos estándares son importantes.

SQL3

El SQL3 es una extensión del estándar de bases de datos SQL92 que incluye soporte para la administración de bases de datos orientada a objetos. Los comités de estandarización ANSI X3H2 e ISO/IEC JTC1/SC21/WG3 han trabajado para desarrollar el anteproyecto del estándar SQL3, el cual se analizará en esta sección. Este estándar es por mucho un trabajo en progreso, y los cambios a este anteproyecto son posibles. Además, el SQL3 es un estándar para productos y no un producto en sí. Actualmente existen productos DBMS no comerciales que implementan este estándar. Usted debe ver esta sección más como una descripción sobre la posible evolución de los productos DBMS relacionales, que como una descripción de las características específicas del producto.

El SQL3 está fuera de la tradición de la administración de bases de datos, pero está dentro de la del pensamiento de objetos. La meta de los comités que trabajan en el SQL3 ha sido describir un estándar que logre compatibilidad con el SQL92. Esto significa que todas las características y funciones del SQL92 también funcionarían con el SQL3. Consecuentemente, el SQL3 funciona así y es una facilidad de base de datos relacional con características de objetos agregadas, en comparación con una nueva facilidad de base de datos orientada a objetos.

En el SQL3 se incorporan tres grupos de nuevas ideas: soporte para tipos de datos abstractos; mejoras a las definiciones de las tablas, y extensiones a las construcciones de lenguaje, para que el SQL3 sea completo desde el punto de vista computacional.

TIPOS DE DATOS ABSTRACTOS. En el SQL3 un **tipo de datos abstractos (ADT)** es una estructura que define el usuario, la cual es equivalente a un objeto OOP. Los ADT tienen métodos, elementos de datos e identificadores. Los ADT pueden ser subtipos de otros ADT; se apoya la herencia. Pueden utilizarse tanto el SQL (con las nuevas extensiones de lenguaje) como un lenguaje externo, tal como Java, C#, y C++ para expresar la lógica de los métodos del ADT.

Un ADT se puede utilizar en una expresión SQL, almacenarse en una tabla, o ambas cosas. Si el ADT aparece en una o más expresiones SQL, pero no está almacenado en ninguna tabla, entonces el ADT es transitorio; de lo contrario, se hace persistente mediante su almacenamiento en una tabla.

La figura 18-12 muestra la definición de un ejemplo de ADT para un tipo de objeto empleado. La sintaxis actual del SQL3 se muestra en mayúsculas, y el código proporcionado por el programador, en minúsculas. Esta sintaxis específica no es importante, puesto que puede cambiar. En vez de esto, observe que este ADT, al igual que un objeto OOP, tiene elementos de datos y funciones (métodos). Los elementos de datos de empleado son nombre, número, fechadecontratación, salarioactual, y salario, el cual es un elemento de datos virtual (aquél que sólo existe como resultado del cálculo en una función). Las funciones son obtener_salario, cambiar_salario, y eliminar_empleado).

El SQL define dos tipos de ADT: los de OBJETO y los ADTs de VALORES. Un ADT DE OBJETO es una estructura de datos identificable e independiente a la que se le asigna un identificador denominado **OID.** Este identificador es un valor único que persiste durante la vida del objeto. Si el programador desea utilizar el valor del OID para pasar a otras funciones, o almacenar en otras tablas, debe agregarse la expresión WITH OID VISIBLE (CON OID VISIBLE) a la primera línea de la definición del objeto. Esto se ha realizado en la figura 18-12.

Los valores del OID son apuntadores a objetos; al guardar un valor del OID en una tabla guarda un apuntador a objeto. Esto puede ser conveniente, pero también genera

➤ FIGURA 18-12

Ejemplo de la definición del ADT en el SQL3

```
CREATE OBJECT TYPE employee WITH OID VISIBLE
        (name VARCHAR NOT NULL,
        number CHAR(7)
        salary UPDATABLE VIRTUAL GET with get_salary SET WITH change_salary,
        PRIVATE
        hiredate DATE
        currentsalary CURRENCY
        PUBLIC
        ACTOR FUNCTION get_salary (:E employee) RETURNS CURRENCY
        {code to perform security processing
         and return value of currentsalary if appropriate}
        RETURN salary
        END FUNCTION,

        ACTOR FUNCTION change_salary (:E employee) RETURNS employee
        {code to perform security processing
         and compute and set new currentsalary, if appropriate}
        RETURN :E
        END FUNCTION,

        DESTRUCTOR FUNCTION remove_employee (:E employee)
                                                RETURNS NULL
        {code to get ready to delete employee data}
        DESTROY :E
        RETURN :E
        END FUNCTION,
```

un problema. Cuando se destruye un ADT, su OID es inválido, pero dicho valor particular del OID pudo haberse almacenado en los renglones de las tablas que ni siquiera están en la memoria cuando se destruye el ADT. El estándar SQL3 no indica qué es lo que pasará en este caso. Aparentemente, los programas se escriben para verificar si un OID es válido antes de intentar utilizarlo.

El segundo tipo de ADT es un ADT DE VALORES. A los ADT DE VALORES no se les asignan OIDs y no pueden existir, excepto en el contexto en el cual se crean. Si un ADT DE VALORES se crea como una columna en una tabla, se guardará junto con ella. No es posible referirse a dicho ADT, excepto mediante el nombre de la tabla. Si se crea un ADT DE VALORES en una función, entonces será transitorio y se destruirá cuando se libere la memoria de la función.

El código de la figura 18-12 define el ADT DE OBJETOS *employee* como un tipo. Como tal, el nombre del tipo se puede utilizar en las definiciones de la tabla, de la misma manera en que pueden utilizarse los tipos de datos incorporados al SQL. En la figura 18-13 se define una tabla Dept; tiene un Deptname del tipo Char(10), un Manager del tipo empleado, y un Admin también del tipo empleado. Por consiguiente, el tipo ADT se utiliza como cualquier otro tipo de datos en la definición de una tabla.

Cuando se define que una columna tiene un tipo ADT, se utiliza la palabra clave INSTANCE para indicar si se almacenará el objeto o un apuntador a éste. Si se especifica la palabra clave INSTANCE, entonces se almacenan los datos del objeto en la columna. Si se omite INSTANCE, entonces se almacena un apuntador al objeto en la columna. Si el ADT es un ADT DE VALORES, entonces se asume la palabra clave INSTANCE.

En la figura 18-13 la columna manager no especifica la palabra clave INSTANCE, pero la columna admin sí. Esto significa que cada renglón de una tabla departamento

➤ FIGURA 18-13

Definición de la tabla mediante el ADT Empleado

```
CREATE TABLE Dept
        ( DeptName          char(10),
          Manager           employee,
          Admin             employee INSTANCE
        )
```

contendrá un apuntador a un empleado en la columna Manager (Gerente) y los datos actuales y los métodos para un Empleado en la columna Admin (Administrador).

Los elementos de los datos públicos de un objeto pueden utilizarse en las instrucciones SQL de la misma manera que en las columnas de las tablas. Por ejemplo, considere la tabla en la figura 18-13 y el siguiente código SQL:

```
SELECT   DeptName, Manager.OID, Manager.Name, Admin.OID, Admin.Name,
SELECT   NombreDepartamento, Gerente.OID, NombredelGerente, Administrador.OID
FROM     Dept
FROM     Departamento
```

Cuando se ejecute este código, se extraerán de la tabla NombredelDepartamento, Gerente-OID, NombredelGerente, Administrador.OID. Tras bambalinas, el DBMS utilizará el valor de Gerente.OID para encontrar el caso del empleado que señala. Después el DBMS extraerá Gerente.Nombre de dicho objeto y lo regresará como parte de la respuesta a esta instrucción SQL. El resultado será el mismo que si se elimina todo el objeto Gerente. Obviamente, si se ha vuelto inválido el OID que está almacenado en la tabla debido a que se ha eliminado su objeto, entonces el DBMS necesitará procesar este error de alguna manera.

Considere la instrucción SQL:

```
SELECT   NombredelDepartamento, NombredelGerente, SalariodelGerente
FROM     Departamento
```

Para procesar esta instrucción el DBMS necesitará ingresar la tabla Departamento, obtener el OID del gerente, obtener la instancia del empleado que es gerente, y luego invocar la función conseguir_salario en el empleado que materialice la columna virtual de salario. La función conseguir salario puede ejecutar una verificación de seguridad cuando se ejecute y de esta manera se le pedirá al usuario que proporcione un nombre o una contraseña, o que realice otras tareas antes de que el DBMS reciba una respuesta de la función obtener_salario. Una vez que la función conseguir_salario haya regresado un valor o un código de errores que indique que no resultará ningún valor, el DBMS podrá formatear los datos para la tabla Departamento. Se necesitará hacer un procesamiento similar para cada renglón de esa tabla.

Los elementos de datos privados son privados para las funciones en el objeto. Por lo tanto, la siguiente instrucción SQL es inválida:

```
SELECT   NombredelDepartamento, Salarioactual.Gerente
FROM     Departamento
```

La única forma en que pueden extraerse los datos de salarioactual de un objeto empleado es mediante la función obtenersalario.

Los valores pueden asignarse a las columnas igual que otras instrucciones SQL. Por consiguiente, la expresión SQL:

```
UPDATE   Departamento
SET      Administrador.Nombre = "Fred P. Johnson"
WHERE    Departamento.Nombre = "Contabilidad"
```

establecerá el nombre del objeto administración que se instaura en el departamento de Contabilidad.

Debido a que algunos objetos se representan mediante apuntadores, y no mediante valores de datos, pueden presentarse algunos resultados sorprendentes. Considere la siguiente instrucción SQL:

```
UPDATE   Departamento
SET      Gerente.Nombre = "Fred P. Johnson"
WHERE    Departamento.Nombre = "Contabilidad"
```

Esta instrucción no cambia la asignación de empleado, por lo que se asigna un empleado diferente cuyo nombre sea "Fred P. Johnson" al departamento de Contabilidad. En

lugar de eso, cambia el nombre del empleado, el cual actualmente es el gerente. El nombre del *employee* cambia, lo cual significa que cualquier otra tabla que haga referencia a este objeto empleado también tendrá cambiado su nombre. Si aún no se ha asignado ningún empleado a Gerente en el renglón Contabilidad, esta instrucción generará un error.

Con el fin de reemplazar al gerente del departamento de Contabilidad con un empleado diferente cuyo nombre sea "Fred P. Johnson", se necesita establecer el objeto *Gerente* en la instancia correcta del objeto. La siguiente instrucción SQL hará esto:

```
UPDATE    Departamento
SET       GERENTE
          SELECT    empleado.OID
          FROM      empleado
          WHERE     nombre = "Fred P. Johnson"
WHERE     Departamento.nombre = "Contabilidad"
```

Conceptualmente, esta instrucción es correcta. Independientemente de que funcione o no, en realidad con un DBMS que implemente el SQL3 dependería de los diseñadores del DBMS. Como se estableció antes, puesto que el SQL3 es un trabajo en progreso, y que actualmente ningún producto lo implementa, considere que el análisis presentado aquí indica la dirección de la industria, y no una sintaxis fija industrialmente aceptada.

La definición de los ADTs le da al SQL3 la habilidad de definir, almacenar y manipular objetos. Asimismo, se muestran otros dos cambios del SQL en el SQL3, los cuales se considerarán posteriormente.

EXTENSIONES DE LAS TABLAS SQL3. El SQL3 amplía la definición de las tablas de muchas maneras. Primero, las tablas SQL3 tienen un **identificador de renglones,** el cual es un identificador único de cada renglón de una tabla. Éste es igual a una *llave sustituta*, el término que se ha empleado en los análisis anteriores. Las aplicaciones pueden utilizar este identificador si se hace explícito al incluir la expresión WITH IDENTITY (CON IDENTIDAD) en la definición de la tabla. A cualquier tabla definida de esta manera se le proporciona una columna implícita llamada IDENTITY (IDENTIDAD). La aplicación puede utilizar los valores en la columna, pero no se incluyen en los resultados de una expresión SELECT *.

Considere la tabla en la figura 18-14 y las dos expresiones SQL siguientes:

```
SELECT    NombredelProfesor, Identidad
FROM      PROFESOR

SELECT    *
FROM      PROFESOR
```

El resultado de la primera instrucción SQL es una tabla con dos columnas; la primera tiene el nombre del profesor, y la segunda, el valor del identificador del renglón. El resultado de la segunda expresión SQL es una tabla de tres columnas, que son NombredelProfesor, Teléfono, y Oficina.

La segunda extensión del concepto de tabla en el SQL3 es la definición de tres tipos de tablas: CONJUNTO, MULTICONJUNTO y LISTA. Una tabla CONJUNTO no tiene renglones duplicados; una tabla MULTICONJUNTO puede tener renglones duplica-

➤ FIGURA 18-14

Definición de la tabla mediante la expresión WITH IDENTITY (CON IDENTIDAD)

```
CREATE TABLE PROFESSOR WITH IDENTITY
          (ProfessorName    char(10),
           Phone            char(7),
           Office           char(5)
          )
```

➤ FIGURA 18-15

Definiciones de subtablas

```
CREATE TABLE PROFESSOR WITH IDENTITY
                (ProfessorName      char(10),
                 Phone              char(7),
                 Office             char(5)
                )

CREATE TABLE TENURED-PROFESSOR UNDER PROFESSOR
                (DateTenureGranted  Date)

CREATE TABLE NON-TENURED-PROFESSOR UNDER PROFESSOR
                (NextReviewDate     Date)
```

dos y es equivalente al concepto de tabla en el SQL92. (Por supuesto, esta definición ignora la columna IDENTIDAD, puesto que con la columna IDENTIDAD ninguna tabla tiene renglones duplicados.) Finalmente, una tabla LISTA tiene un orden definido por una o más columnas.

Una tercera extensión del concepto de tabla en el SQL3 es la **subtabla.** Una subtabla es un subconjunto de otra tabla, llamada **supertabla.** Una subtabla hereda todas las columnas de su supertabla y también puede tener columnas propias. Una tabla que tiene una subtabla o una supertabla tiene un identificador de renglón definido implícitamente. La figura 18-15 define dos tipos de profesor: TENURED-PROFESSOR (PROFESOR ACTIVO) y NONTENURED-PROFESSOR (PROFESOR NO ACTIVO). Las columnas de TENURED-PROFESSOR (PROFESOR ACTIVO) son NombredelProfesor, Teléfono, Oficina y FechadeOcupacióndelPuesto. Las columnas de NONTENURED-PROFESSOR (PROFESOR NO ACTIVO) son NombredelProfesor, Teléfono, Oficina y FechadePróximaRevisión. Aunque WITH IDENTITY (CON IDENTIDAD) no está especificada para PROFESOR ACTIVO o PROFESOR NO ACTIVO, ya que ambos son subtipos que tienen una columna IDENTIDAD.

Reflexione por un momento sobre las consecuencias lógicas de agregar los ADTs y los subtipos a la construcción de la tabla. Los ADTs y las tablas pueden tener subtipos, sin embargo, son distintos. Los subtipos ADTs definen una jerarquía de generalización, y los subtipos de las tablas definen otra. Una jerarquía puede anidarse en la otra o viceversa, o pueden no ser un joint, o ejecutarse parcialmente al mismo tiempo. Aquí, el SQL3 está abierto a la crítica por complejidad excesiva, y será muy interesante ver qué tanto de esta complejidad se implementa realmente en los productos DBMS.

EXTENSIONES DEL LENGUAJE SQL. De acuerdo con el SQL3, los métodos ADT pueden codificarse en el propio lenguaje de SQL. Para hacer más robusta esta habilidad, se proponen los elementos del lenguaje que harán al SQL computacionalmente completo. Las adiciones propuestas se resumen en la figura 18-16.

A la fecha, el SQL ha sido un lenguaje orientado a conjuntos. Las instrucciones SELECT identifican un conjunto de renglones y operan sobre ellos. En la figura 18-16 la adición de las instrucciones de lenguaje cambiará esta característica. Será posible desarrollar una lógica de un renglón a la vez dentro del mismo SQL. Este cambio hará al SQL cada vez más parecido a un lenguaje de programación tradicional. Esto es necesario si se utiliza al SQL como el lenguaje para la lógica en los métodos ADT, pero también representa un cambio en el carácter fundamental del SQL.

ODMG-93

El Grupo de Administración de Datos de Objetos es un consorcio de proveedores de bases de datos de objetos y de otros expertos de la industria que han aplicado las ideas de otro grupo, el Grupo de administración de objetos, al problema de las bases de datos de objetos. El primer reporte del ODMG se produjo en 1993 y por consiguiente se le denomina ODMG-93. La herencia de este estándar es la programación de objetos, no la

➤ FIGURA 18-16

Extensiones propuestas del lenguaje SQL3

Instrucción	Propósito
DESTROY	Destruye un objeto ADT; válido solamente en las funciones DESTRUCT
ASSIGNMENT	Permite que el resultado de una expresión de valores SQL se asigne a una variable local, columna, o atributo ADT
CALL	Invoca un procedimiento SQL
RETURN	Regresa un valor de un cálculo de valores en un procedimiento o función
CASE	Selecciona la trayectoria de ejecución con base en los valores alternativos
IF THEN ELSE (SI ES ASÍ ENTONCES)	Permite la lógica condicional
WHILE LOOP (MIENTRAS EL RIZO)	Permite la lógica iterativa

administración de bases de datos relacionales tradicionales. Por consiguiente, se basa en el objeto como la construcción fundamental, en lugar de basarse en la tabla, como vimos antes con el SQL3.

El ODMG-93 es una definición de las interfaces para los productos de administración de datos de objetos. Las implementaciones de las ideas en el ODMG-93 pueden ser un poco diferentes. Un producto ODMG-93 que está diseñado para el almacenamiento y el manejo de datos de objetos de C++ puede tener una implementación completamente diferente a la de un producto que está diseñado para el almacenamiento y manejo de objetos Smalltalk. Los dos productos pueden ser muy diferentes, sin embargo, ambos implementan las interfaces del ODMG-93.

Debido a que el ODMG-93 está fuera del contexto de la programación de objetos, una descripción detallada de él requiere un conocimiento sustancial del OOP. Consecuentemente, dicha descripción está fuera del alcance de este texto. En su lugar, el presente análisis se restringe a las ideas fundamentales del reporte del ODMG-93. La figura 18-17 enumera cinco conceptos principales en la forma en que los describe Loomis.[2]

LOS OBJETOS SON FUNDAMENTALES. De acuerdo con el modelo de objetos del ODMG, el objeto es la entidad fundamental que se almacena y maneja. A diferencia del SQL3, en el cual la entidad fundamental es una tabla y los objetos se almacenan en las columnas de las tablas, en el ODMG-93 el objeto es la entidad básica. El concepto del ODMG-93 se asemeja más al que se describió para el programa de objetos de la figura 18-3. Es decir, el programa de aplicación define los objetos en y por sí mismos, y depende del ODBMS hacer a dichos objetos persistentes. No se requiere ninguna otra estructura, como una tabla.

De acuerdo con el modelo ODMG, los objetos pueden ser **mutables** o **inmutables**. Los objetos mutables pueden cambiarse; los inmutables están fijos, y no se permite que ninguna aplicación altere su estado. El ODBMS se requiere para la imposición de la inmutabilidad.

➤ FIGURA 18-17

Elementos clave del modelo de objetos del ODMG

- Los objetos son fundamentales
- Cada objeto tiene un identificador único, durante la vida del objeto
- Los objetos pueden organizarse en tipos y subtipos
- El estado se define mediante los valores de datos y las relaciones
- El comportamiento se define mediante las operaciones de los objetos

[2]Mary E. S. Loomis, *Object Databases. The Essentials*. Reading, Addison-Wesley, Ma., 1995, pp. 88-110.

CADA OBJETO TIENE UN IDENTIFICADOR A LO LARGO DE SU VIDA. El segundo concepto fundamental en el modelo de objetos del ODMG es que a cada objeto se le da un identificador único, que es válido durante el tiempo de vida del objeto. Además, el identificador debe ser válido si el objeto está almacenado externamente, o si se encuentra en la memoria. El ODBMS realizará el intercambio (swizzling) transparentemente; el programa de aplicación puede utilizar indicadores de objetos como si éstos fueran siempre válidos.

El estándar deja abierta la forma particular de un identificador de objetos. De esta manera, los diferentes proveedores del ODBMS pueden utilizar distintos medios para especificar los ID (identificadores) del objeto. Esto significa que los identificadores de objetos de distintas bases de datos de proveedores diferentes no son necesariamente compatibles. Para bases de datos no distribuidas esto no parece ser un problema, ya que todos los objetos en una base de datos de objetos determinada los habrá creado y almacenado el mismo ODBMS.

En un ambiente distribuido, el problema de identificación de objetos es más difícil por dos razones: primero, debido a que los IDs de los objetos en ODBMS diferentes pueden tener formatos distintos, y segundo, porque los IDs de los objetos no son necesariamente únicos en diferentes bases de datos. Este tema no lo aborda el estándar ODMG-93.

LOS OBJETOS PUEDEN ARREGLARSE EN TIPOS Y SUBTIPOS. El modelo de objetos del estándar ODMG especifica que los objetos se organizan en grupos por tipo. Los objetos están creados para ser de un tipo específico. Todos los objetos de un tipo en particular tienen las mismas características de datos y comportamiento. Los objetos pueden definirse como subtipos de otros objetos. En este caso, heredan todas las características de los datos y del comportamiento de su tipo padre. Según el estándar, un objeto está creado como un caso de un tipo específico y no puede cambiarlo.

A menudo, los términos *tipo* y *clase* se utilizan como sinónimos. De acuerdo con Loomis, esto es incorrecto. Una clase de objetos es un grupo lógico de objetos, como se definió en el ODMG-93; tales clases tienen subclases que heredan de las primeras. Un tipo es la implementación de una clase en un lenguaje particular. Por lo tanto, la clase Empleado es una definición lógica de los datos y los métodos; puede tener las subclases Vendedor y Contador que heredan de Empleado. Por ejemplo, una implementación de Empleado en C++ se llama tipo; las implementaciones en C++ de Vendedor y Contador son subtipos.[3] Probablemente exista otra implementación diferente de Empleado en, por ejemplo, Smalltalk. Dicha implementación sería un tipo diferente de Empleado. La distinción entre *clase* y *tipo* ayuda a delimitar las definiciones lógicas de las implementaciones particulares de dichas estructuras lógicas.

Las clases de objetos (y, por tanto, los tipos) pueden tener propiedades. El estándar ODMG especifica que cada clase tiene un nombre y limitaciones de unicidad como sus propiedades. Todas las ocurrencias de una clase de objetos se denominan **extensión** del objeto. Cualquier atributo o combinación de atributos puede declararse como único en la extensión. Por consiguiente, en Empleado, Número de empleado puede definirse como único, como puede ser {Nombre, Apellido}, etc. Ya que los requerimientos de unicidad se aplican para toda la extensión, y no para cualquier ocurrencia determinada del objeto, tales requerimientos son propiedades de la clase, y no de la ocurrencia de la clase. De esta manera, el nombre Empleado y el requerimiento de que Número de empleado sea único son propiedades de la clase. Sin embargo, Número de empleado es por sí mismo una propiedad de una ocurrencia de Empleado.

Puesto que el ODMG es un estándar para una interfaz y no para una implementación, no se hace ningún intento para describir la forma en que deban almacenarse o manejarse los tipos y los subtipos. Más bien, la interfaz simplemente indica que los objetos debe almacenarlos y recuperarlos la clase y debe proporcionarse dicha herencia.

EL ESTADO SE DEFINE POR LOS VALORES DE DATOS Y LAS RELACIONES. De acuerdo con el estándar ODMG, el estado de cualquier objeto está representado por sus propiedades, las cuales pueden ser atributos o relaciones. Un atributo es un valor literal o

[3] *Ibid.*, p. 96.

➤ FIGURA 18-18

Operaciones de relación del ODMG

Operación	Función
Set (Establecer)	Crear una relación 1:1
Clear (Borrar)	Destruir una relación 1:1
Insert_element (Insertar_elemento)	Agregar un elemento al lado muchos de una relación 1:N o N:M
Remove_element (Eliminar_elemento)	Eliminar un elemento del lado muchos de una relación 1:N o N:M
Get (Obtener)	Regresar una referencia a un objeto en una relación 1:1
Traverse (Repasar ejecución)	Regresar una referencia a un conjunto de objetos en el lado muchos de una relación 1:N o N:M
Create_iterator (Crear_repetidor)	Crear una estructura para procesar los elementos de un conjunto de objetos obtenido mediante una operación Traverse (Repasar ejecución)

un conjunto de valores literales. FechadeContratación y SalarioActual son valores literales. SalarioAnterior es un conjunto de valores literales. Una relación es una propiedad que indica una conexión entre una ocurrencia del objeto y una o más ocurrencias de otros objetos. Departamento es un ejemplo de una propiedad de la relación.

El ODMG especifica un conjunto de operaciones que pueden ejecutarse en las relaciones, las cuales se enumeran en la figura 18-18. Las operaciones se distinguen por medio de la cardinalidad máxima de la relación. Esto se hace puesto que en el caso de una relación 1:1 las propiedades sólo tienen un valor, y no se necesita considerar ningún conjunto de propiedades. En el caso de una relación 1:N o N:M, el número de elementos en una propiedad es plural; se crea un conjunto y el programa debe ser capaz de repetirse en los elementos del conjunto.

Cuando los objetos tienen una relación, ésta debe hacerse persistente cuando el objeto se convierte en persistente. El estándar no especifica cómo se representará la relación y qué medios se utilizarán para intercambiar (swizzle) los apuntadores entre las relaciones. Sin embargo, estas cuestiones deben resolverse cuando se implementa un ODBMS.

EL COMPORTAMIENTO LO DEFINEN LAS OPERACIONES DE LOS OBJETOS. El comportamiento de un tipo de objetos se determina a través de sus métodos. Todos los objetos de un tipo determinado tienen los mismos métodos y los objetos de los subtipos los heredan. Si un objeto del subtipo vuelve a definir un método, entonces la redefinición anulará el método heredado. Por ejemplo, si el objeto Empleado tiene el método Obtener_Salario, y si el objeto Vendedor, el cual es un subtipo de Empleado, también tiene un método llamado Obtener_Salario, el método local se utilizará para las operaciones Vendedor!Obtener_Salario.

Los objetos interactúan entre sí al invocar los otros métodos de cada uno. Al respecto, a veces se dice que los objetos se pasan mensajes entre sí, donde un mensaje es una cadena como Vendedor!Obtener_Salario que incluye el nombre del tipo de objetos y el nombre del método de dicho tipo. Por supuesto, los mensajes pueden incluir parámetros.

El propósito de un ODBMS es hacer persistentes a los objetos. El estándar ODMG indica que los objetos incluyen métodos, por lo que parecería que se incluyen el almacenamiento y la administración de los métodos en las funciones de un ODBMS compatible con el ODMG. En realidad, los ODBMS actuales varían mucho en su soporte para el almacenamiento de métodos. De hecho, algunos ODBMS no proporcionan ningún soporte de ninguna clase para la persistencia de métodos. Otros proporcionan algún soporte, pero no soportan versiones de objetos.

La persistencia del método es importante, y posiblemente las capacidades y habilidades del ODBMS en esta área mejorarán en el futuro. Ninguna aplicación es estática; los requerimientos cambian y deben adaptarse los comportamientos de los objetos. Además, los métodos pueden cambiar sin modificar las propiedades fundamentales de los datos de un objeto. Sin la administración de métodos, dos ocurrencias de un objeto se pueden basar en dos versiones de métodos diferentes.

Considere un ejemplo: suponga que se crea y se almacena una ocurrencia de una clase de Vendedor, por ejemplo Vendedor A, mediante una versión del método Establecer_Salario. Ahora suponga que los medios de calcular los salarios de los vendedores cambian, y que en efecto se altera el método Obtener_Salario. En este punto, se crea y se almacena el Vendedor B. Ahora, parecería que el Vendedor A y el Vendedor B son equivalentes en sus propiedades de datos, pero en realidad no lo son. Sin la administración de métodos en la parte del ODBMS no existe ninguna manera de determinar que los Vendedores A y B representan versiones diferentes del objeto Vendedor.

Esta situación es similar a lo que ocurre actualmente con los programas de aplicación en ambientes distintos al ODBMS, por lo que los proponentes del ODBMS desearían que el ODBMS no empeorara la situación. Sin embargo, esto parece revocarse. Si los objetos están definidos para incluir las propiedades de datos y las del comportamiento, entonces no se puede reclamar que la persistencia del objeto pertenezca a uno y no al otro. Gran parte de la promesa del pensamiento de objetos se deja en la tabla, si es que el ODBMS no soporta la administración de métodos ni la administración de datos.

➤ RESUMEN

Las bases de datos relacionales no se adaptan bien para almacenar los objetos de la programación orientada a objetos, debido a que éstos pueden contener estructuras complejas que no se adapten fácilmente a la tabla. Asimismo, los objetos incluyen métodos de objetos que también necesitan almacenarse. Los productos DBMS orientados a objetos con un propósito especial se desarrollaron para proporcionar almacenamiento del objeto persistente, pero comercialmente no han tenido éxito debido a que los datos relacionales deben convertirse al formato del ODBMS. La ganancia no compensa el costo. En lugar de eso, los productos DBMS están empezando a dar soporte a los híbridos del objeto y el almacenamiento relacional denominado bases de datos objeto-relacional.

Con la programación orientada a objetos (OOP), los programas están compuestos de objetos que son estructuras lógicas encapsuladas, las cuales poseen elementos de datos y comportamientos. Una interfaz es la apariencia externa de un objeto; una implementación es el interior encapsulado de un objeto. Los objetos pueden dividirse en subclases; una subclase hereda los atributos y los métodos de su superclase. El polimorfismo permite que existan varias versiones del mismo método; el compilador invoca la versión apropiada en el tiempo de ejecución, dependiendo de la clase de objeto.

Una clase de objeto es la estructura lógica de un objeto; un grupo de clases de objetos se llama biblioteca de clases de objetos. Las ocurrencias de una clase de objetos se denominan ocurrencias del objeto, o simplemente objetos. Los constructores de objetos son métodos que obtienen memoria y crean estructuras de objetos; los destructores de objetos desenlazan los objetos y liberan la memoria. Los objetos transitorios sólo existen durante la ejecución de un programa; los objetos persistentes se guardan para almacenar y sobrevivir a la ejecución de un programa.

Los objetos pueden hacerse persistentes mediante el almacenamiento tradicional de archivos, mediante el DBMS relacional, o a través de los productos ODBMS. El uso del almacenamiento tradicional le da al programador de la aplicación una cantidad considerable de trabajo y sólo es viable para las aplicaciones que tienen algunos objetos simples, cuya estructura no cambie frecuentemente. El DBMS relacional puede emplearse para la persistencia de objetos, pero el programador de la aplicación debe convertir las estructuras de objetos en relaciones, escribir el SQL y desarrollar algoritmos de intercambio (swizzling). El uso de un ODBMS es el medio más directo y fácil de la persistencia de objetos.

Oracle proporciona soporte para la persistencia de objetos junto con los dispositivos de bases de datos relacionales de objetos. Los tipos de objetos pueden definirse y luego utilizarse con las estructuras de las tablas como objetos de columnas, arreglos de longitud variable, tablas anidadas, y como objetos renglón. Asimismo, los objetos puros pueden definirse; pueden incluir arreglos de longitud variable y tablas anidadas. También pueden incluir apuntadores a objetos como atributos REF. Finalmente, dichos objetos incluyen métodos que se pueden utilizar para el procesamiento de las bibliotecas de clases que proporciona Oracle.

El SQL3 es una extensión del SQL-92 que ofrece, a los tipos de datos abstractos (ADT), mejoras en las tablas y nuevas características en el lenguaje SQL. Los ADTs, que se hacen persistentes al implantarlos en las tablas, pueden ser de objetos o de valores; los ADTs de objetos tienen identificadores llamados OIDs. En el SQL3 las tablas tienen un identificador de renglón y pueden tener subtipos. Se definen tres tipos de tablas: SET, MULTISET y LIST. La figura 18-13 muestra las extensiones al lenguaje SQL propuestas en el SQL3.

Los cinco elementos básicos del estándar ODMG-93 radican en que los objetos son la estructura fundamental de los datos, a los objetos se les da un identificador persistente, durante la vida del objeto los objetos pueden organizarse en tipos y subtipos, el estado del objeto es soportado por los valores de los datos y las relaciones, y el comportamiento del objeto está definido por las operaciones del objeto. Como se definió en el capítulo 4, los objetos semánticos implementan el estándar ODMG para los atributos de datos, pero no para el comportamiento de los objetos.

➤ PREGUNTAS DEL GRUPO I

18.1 Explique en qué difiere la programación orientada a objetos de la programación tradicional.

18.2 ¿Por qué las bases de datos relacionales son más populares actualmente que las bases de datos de objetos?

18.3 Defina un objeto OOP.

18.4 Defina los términos *encapsulado, atributo* y *método*.

18.5 Explique la diferencia entre una interfaz y una implementación.

18.6 ¿Qué es la herencia?

18.7 ¿Qué es el polimorfismo?

18.8 Defina los términos *clase del objeto, biblioteca de clases de objetos,* y *ocurrencia del objeto*.

18.9 Explique la función de los constructores de objetos y de los destructores de objetos.

18.10 Explique la diferencia entre un objeto transitorio y un objeto persistente.

18.11 Explique la diferencia en la notación CUSTOMER!Find y CUSTOMER.ZipCode.

18.12 ¿Cuál es la función de la palabra clave *NOTHING (NADA)* en la figura 18-3?

18.13 ¿Cuál es la función de la palabra clave *ME* en la figura 18-3?

18.14 ¿Qué es una repetición de llamada, y por qué se utiliza?

18.15 ¿A qué se refiere el término *swizzling (intercambio)*?

18.16 Explique brevemente las tareas que se requieren para utilizar el almacenamiento tradicional de archivos para la persistencia de objetos.

18.17 Explique brevemente las tareas que se requieren para utilizar un DBMS relacional para la persistencia de objetos.

18.18 Resuma las ventajas y desventajas de la utilización de un ODBMS para la persistencia de objetos.

18.19 Muestre las instrucciones Oracle para definir un tipo de objetos llamado Pnombre que tenga tres atributos: Primernombre, Segundonombre, y Apellido. Utilice este tipo como un objeto de la columna en la tabla llamada PERSONA.

18.20 Muestre las instrucciones Oracle para crear un arreglo variable de hasta 100 Pnombres. Muestre las instrucciones para crear una tabla llamada CLUB1 con la llave sustituta, un Nombredelclub y el atributo del arreglo variable de los nombres de las personas.

18.21 Muestre las instrucciones Oracle para crear una tabla CLUB2 de la misma forma que en la pregunta 18.20, pero utilice una tabla anidada de los nombres de las personas en lugar de un arreglo variable. Nombre al almacenamiento Pnombre_Tabla.

18.22 Explique las diferencias entre la tabla CLUB1 y la tabla CLUB2.

18.23 Muestre las instrucciones Oracle para crear una tabla que tenga a Pnombre como objeto renglón.

18.24 Explique el objetivo de los atributos REF en la figura 18-19. ¿Cómo difieren éstos de las llaves externas?

18.25 Explique el propósito de las dos instrucciones siguientes:

a. FOR i in 1..SELF.Lineitems.COUNT LOOP

b. UTL_REF.SELECT_OBJECT(Lineitems(i).ItemRef, itemPtr)

18.26 ¿Qué es el SQL3?

18.27 ¿Qué es un tipo de datos abstractos (ADT)?

18.28 Explique la diferencia entre un objeto ADT y un valor ADT.

18.29 ¿Qué es un OID? ¿Cómo puede utilizarse uno?

18.30 Explique qué debe hacer el DBMS cuando se ejecuta la siguiente instrucción SQL en el ADT de la figura 18-13:

```
SELECT   DeptName, Manager.Phone, Admin.Phone
FROM     Dept
```

18.31 ¿Qué sucede cuando se ejecuta la siguiente instrucción SQL en el ADT de la figura 18-13?:

```
UPDATE   Dept
SET      Manager.Name = "John Jacob Astor"
```

18.32 Codifique el SQL que se necesitaría ejecutar para cambiar la ocurrencia del gerente de un departamento para el ADT en la figura 18-13.

18.33 ¿Qué es un identificador de renglón en el SQL3?

18.34 Explique las diferencias entre un SET, un MULTISET, y un LIST en el SQL3.

18.35 Explique las diferencias entre una subtabla, una supertabla y una tabla.

18.36 ¿Qué es el ODMG-93?

18.37 Liste los cinco conceptos principales en el ODMG-93.

18.38 ¿Cuál es la diferencia entre un tipo y una clase en el ODMG-93?

18.39 ¿Cuál es la diferencia entre una propiedad y un atributo en el ODMG-93?

18.40 ¿Qué es una extensión?

18.41 ¿Cuáles son las propiedades de una clase en el ODMG-93?

18.42 En el ODMG estándar, ¿qué valores pueden tener las propiedades?

18.43 En el ODMG estándar, ¿qué valores pueden tener los atributos?

18.44 En el ODMG estándar, ¿qué valores pueden tener las propiedades de las relaciones?

18.45 ¿Por qué es importante la persistencia del método? Dé un ejemplo sobre un problema que podría presentarse cuando no se proporciona dicha persistencia.

18.46 Explique de qué manera los objetos semánticos CUMPLEN el estándar ODMG y de qué manera no lo cumplen.

➤ PREGUNTAS DEL GRUPO II

18.47 En el capítulo 10 repase los requerimientos y el diseño relacional de la Galería View Ridge. Considere la utilización de los tipos de Oracle y las tablas de arre-

glos variables, anidadas, y los objetos renglón en el contexto de las necesidades de View Ridge. ¿Qué cambios haría al diseño relacional? ¿Recomendaría reemplazar las tablas TRANSACTION o WORK con los arreglos variables o tablas anidadas? Explique su respuesta. Muestre cómo puede utilizar los tipos de datos REF de Oracle para eliminar la necesidad de la tabla de intersección. ¿Usted recomendaría esta acción?

18.48 Considere los tipos de Oracle, los arreglos variables y las tablas anidadas en el contexto del modelo del objeto semántico. ¿Cuáles elementos de dicho modelo se prestan a sí mismos a los tipos? ¿Cuáles, a los arreglos variables? ¿ Y cuáles, a las tablas anidadas? Muestre cómo utilizaría los dispositivos relacionales de los objetos de Oracle para modelar cada uno de los tipos de los objetos semánticos descritos en el capítulo 7.

APÉNDICE A

Estructuras de datos para el procesamiento de bases de datos

Todos los sistemas operativos proporcionan servicios de administración de datos. Sin embargo, estos servicios no son, por lo general, suficientes para las necesidades especializadas de un DBMS. Por lo tanto, para mejorar el rendimiento, los productos DBMS construyen y mantienen estructuras especializadas de datos, que son el tema del presente apéndice.

Iniciaremos con el análisis de archivos planos y algunos de los problemas que pueden presentarse cuando se necesita procesar dichos archivos en diferentes órdenes. Posteriormente se indicarán las tres estructuras especializadas de datos: listas secuenciales, listas encadenadas, e índices (o listas invertidas). A continuación ilustraremos cómo se representan cada una de estas estructuras especiales, las cuales analizamos en el capítulo 6 —árboles, redes simples y redes complejas— mediante diversas estructuras de datos. Por último, exploraremos cómo representar y procesar las llaves múltiples.

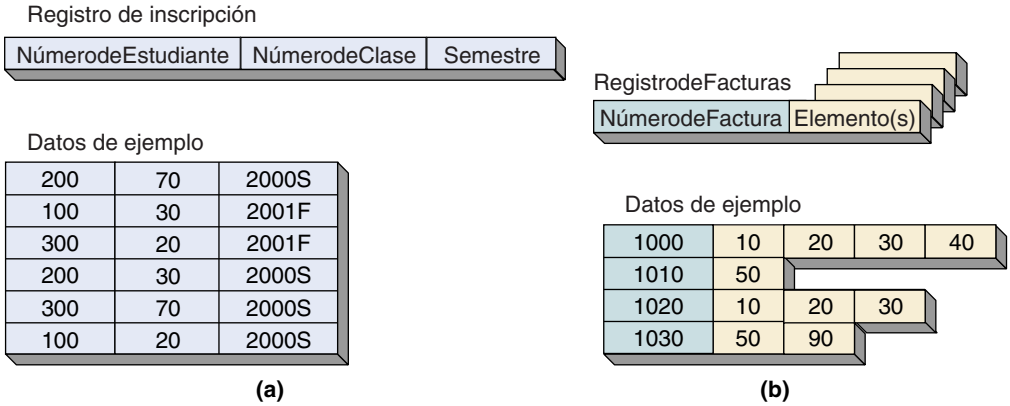
Aunque no se requiere un conocimiento completo de las estructuras de datos para utilizar la mayoría de los productos DBMS, estos conocimientos son esenciales para los administradores de bases de datos y programadores de sistemas que trabajan con un DBMS. Familiarizarse con estas estructuras de datos también le ayudará a evaluar y comparar los productos de las bases de datos.

➤ ARCHIVOS PLANOS

Un *archivo plano* es aquel que no tiene grupos repetitivos. La figura A-1(a) muestra un archivo plano, y la (b) un archivo que no es plano debido al campo repetitivo Elemento. Un archivo plano puede almacenarse en cualquier organización de archivos comunes tales como la secuencial, secuencial indexada, o directa. Los archivos planos se han utilizado en el procesamiento comercial durante muchos años. A menudo se procesan en un orden predeterminado, por ejemplo, en una secuencia ascendente en un campo llave.

➤ FIGURA A-1

Ejemplos del registro de inscripción de un archivo (a) plano y (b) no plano



Procesamiento de archivos planos en ordenamientos múltiples

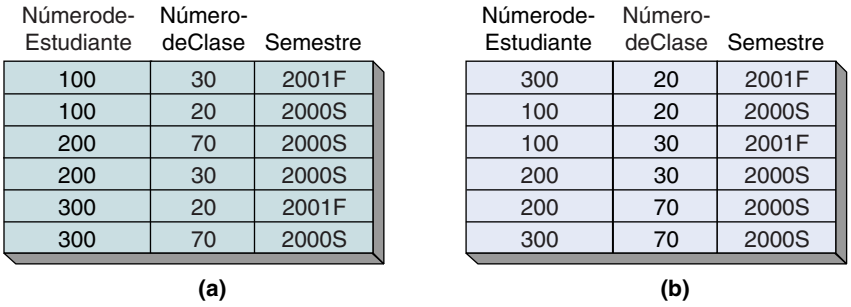
A veces los usuarios desean procesar los archivos planos de formas que no soporta fácilmente la organización del archivo. Por ejemplo, considere los registros de INSCRIPCIÓN en la figura A-1(a). Para producir las agendas de los estudiantes, éstas deben procesarse en la secuencia NúmerodeEstudiante. Pero para producir los horarios de clases, necesitan procesarse los registros en la secuencia de NúmerodeClase. Por supuesto, los registros sólo pueden almacenarse en una secuencia física. Por ejemplo, pueden estar en el orden de NúmerodeEstudiante o en el de NúmerodeClase, pero no en ambos al mismo tiempo. La solución tradicional al problema del procesamiento de registros en diferentes ordenamientos es clasificarlos en el orden de estudiante, procesar las agendas de los estudiantes y después clasificar los registros en orden de clase y producir los horarios de clases.

Para ciertas aplicaciones, tales como el sistema de modo de procesamiento por lotes, esta solución, aunque voluminosa, es efectiva. Pero suponga que ambos ordenamientos necesitan existir simultáneamente debido a que dos usuarios concurrentes tienen vistas diferentes de los registros de INSCRIPCIÓN. ¿Qué se hace entonces?

Una solución es crear dos copias del archivo INSCRIPCIÓN y clasificarlas como se muestra en la figura A-2. Puesto que los datos se listan en orden secuencial, algunas veces esta estructura de datos se denomina *lista secuencial*. Las listas secuenciales pueden almacenarse fácilmente como archivos secuenciales. Sin embargo, por lo general esto no lo realizan los productos DBMS debido a que la lectura secuencial de un archivo es un proceso lento. Además, los archivos secuenciales no pueden actualizarse a la mitad del proceso sin volver a escribir el archivo entero. Asimismo, el mantenimiento de diversos ordenamientos mediante la conservación de copias múltiples de la misma lista secuencial, por lo general, no es eficaz debido a que la lista secuencial duplicada puede generar problemas referentes a la integridad de los datos. Afortunadamente, otras estructuras de datos permiten procesar los registros en ordenamientos diferentes y no requieren la duplicación de datos. Éstos incluyen las listas *indexadas* y los *índices*.

➤ FIGURA A-2

Datos de INSCRIPCIÓN almacenados como listas secuenciales: (a) almacenados por NúmerodeEstudiante, y (b) almacenados por NúmerodeClase



➤ FIGURA A-3

Datos de INSCRIPCIÓN en el orden NúmerodeEstudiante mediante una lista encadenada

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Enlace
1	200	70	2000S	4
2	100	30	2001F	6
3	300	20	2001F	5
4	200	30	2000S	3
5	300	70	2000S	0
6	100	20	2000S	1

Inicio de la lista = 2

Una nota sobre el direccionamiento de registros

Usualmente el DBMS crea registros físicos grandes, o bloques, en sus archivos de acceso directo. Éstos se emplean como contenedores para registros lógicos. Por lo general existen muchos registros lógicos por registro físico. Aquí supondremos que cada registro físico se localiza mediante su número de registro relativo (RRN). Por lo tanto, se debe asignar un registro lógico al número de registro físico 7 o 77 o 10000. Por consiguiente, el número del registro relativo es la dirección física de un registro lógico. Si existe más de un registro lógico por registro físico se debe especificar también la dirección en la que se encuentra el registro lógico, dentro del registro físico. Por lo tanto, la dirección completa de un registro lógico debe ser el número de registro relativo 77, localización en byte 100. Esto significa que el registro comienza en el byte 100 del registro físico 77.

Para simplificar las ilustraciones en este texto, supondremos que sólo existe un registro lógico por registro físico, por lo que no es necesario interesarse en la ubicación en los bytes dentro de los registros físicos. Aunque esto no es realista, sí simplifica este análisis en los puntos esenciales.

Mantenimiento del orden con listas encadenadas

Las listas encadenadas pueden emplearse para conservar en un orden lógico los registros que no necesariamente se encuentren en un orden físico. Para crear una lista encadenada se le agrega un campo a cada registro de datos. El campo *link* mantiene la dirección (en las ilustraciones, el número de registro relativo) del registro *siguiente* en una secuencia lógica. Por ejemplo, la figura A-3 muestra ampliados los registros de INSCRIPCIÓN para incluir una lista encadenada; ésta mantiene los registros en el orden NúmerodeEstudiante. Advierta que el enlace para el último estudiante numérico en la lista es cero.

La figura A-4 muestra los registros de INSCRIPCIÓN con dos listas encadenadas: una mantiene el orden NúmerodeEstudiante y la otra, el orden NúmerodeClase. Se han agregado dos campos de link a los registros, uno para cada lista.

➤ FIGURA A-4

Datos de INSCRIPCIÓN en dos órdenes mediante listas encadenadas

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Link de Estudiante	Link de la Clase
1	200	70	2000S	4	5
2	100	30	2001F	6	1
3	300	20	2001F	5	4
4	200	30	2000S	3	2
5	300	70	2000S	0	0
6	100	20	2000S	1	3

Inicio de la lista de estudiantes = 2
Inicio de la lista de clases = 6

➤ FIGURA A-5

Datos de INSCRIPCIÓN después de la inserción del registro nuevo (en dos órdenes mediante listas encadenadas)

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Link de Estudiante	Link de la Clase
1	200	70	2000S	4	5
2	100	30	2001F	6	7
3	300	20	2001F	5	4
4	200	30	2000S	7	2
5	300	70	2000S	0	0
6	100	20	2000S	1	3
7	200	45	2000S	3	1

Inicio de la lista de estudiantes = 2
Inicio de la lista de clases = 6

Cuando se realizan las inserciones y eliminaciones, las listas encadenadas tienen una gran ventaja sobre las listas secuenciales. Por ejemplo, para insertar el registro de INSCRIPCIÓN para el Estudiante 200 y la Clase 45, se necesitaría volver a escribir las listas en la figura A-2. Sin embargo, para las listas encadenadas en la figura A-4, se le puede agregar el nuevo registro al final físico de la lista, y sólo se necesitaría cambiar los valores de los dos campos de enlace para colocar al registro nuevo en las secuencias correctas. Dichos cambios se muestran en la figura A-5.

Cuando se elimina un registro de una lista secuencial se crea una separación (gap). Pero en una lista encadenada simplemente puede eliminarse un registro mediante el cambio de los valores del link, o del campo *apuntador*. En la figura A-6 el registro de INSCRIPCIÓN para el Estudiante 200, Clase 30, se ha eliminado en forma lógica. Ningún otro registro apunta a su dirección, por lo que se ha removido con éxito de la cadena, aunque aún exista físicamente.

Hay muchas variaciones de las listas encadenadas. La lista puede transformarse en lista *circular*, o *anillo*, mediante el cambio del link del último registro de cero a la dirección del primer registro en la lista. Ahora, se puede llegar a cada artículo en la lista comenzando desde cualquier artículo. La figura A-7(a) muestra una lista circular para el orden Número de estudiante. Una *lista doblemente encadenada* contiene cadenas en ambas direcciones. En la figura A-7(b) se ha creado una lista doblemente encadenada para el acceso ascendente y descendente del estudiante.

Los registros ordenados mediante las listas encadenadas no pueden almacenar un archivo secuencial, debido a que se necesita cierto tipo de organización de los archivos de acceso directo para emplear los valores de enlace (link). Por consiguiente, se requiere de la organización secuencial indexada o de archivos directos para el procesamiento de la lista encadenada.

Manteniendo el orden con índices

También se puede mantener un orden de registros lógicos mediante los *índices* o, como se denominan algunas veces, las *listas invertidas*. Un índice es simplemente una tabla

➤ FIGURA A-6

Datos de INSCRIPCIÓN después de la eliminación de Estudiante 200, Clase 30 (en dos órdenes mediante las listas encadenadas)

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Link de Estudiante	Link de la Clase
1	200	70	2000S	7	5
2	100	30	2001F	6	7
3	300	20	2001F	5	2
4	200	30	2000S	7	2
5	300	70	2000S	0	0
6	100	20	2000S	1	3
7	200	45	2000S	3	1

Inicio de la lista de estudiantes = 2
Inicio de la lista de clases = 6

➤ FIGURA A-7

Datos de INSCRIPCIÓN ordenados por NúmerodeEstudiante mediante una lista: (a) circular, y (b) doblemente encadenada

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Enlace (link)
1	200	70	2000S	4
2	100	30	2001F	6
3	300	20	2001F	5
4	200	30	2000S	3
5	300	70	2000S	2
6	100	20	2000S	1

Inicio de la lista = 2

(a)

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre	Link ascendente	Link descendente
1	200	70	2000S	4	6
2	100	30	2001F	6	0
3	300	20	2001F	5	4
4	200	30	2000S	3	1
5	300	70	2000S	0	3
6	100	20	2000S	1	2

Inicio de la lista ascendente = 2
Inicio de la lista descendente = 5

(b)

que hace una referencia cruzada de las direcciones de los registros con algún valor de campo. Por ejemplo, la figura A-8(a) muestra los registros de INSCRIPCIÓN almacenados en un orden no particular, y la figura A-8(b) muestra un índice en NúmerodeEstudiante. En éste los NúmerosdeEstudiante se ordenan en secuencia, con cada entrada en la lista, indicando un registro correspondiente en los datos originales.

➤ FIGURA A-8

Datos de INSCRIPCIÓN clasificados por sus índices correspondientes: (a) datos de INSCRIPCIÓN, (b) índice del NúmerodeEstudiante, y (c) índice del NúmerodeClase

Número de registro relativo	Númerode-Estudiante	Número-deClase	Semestre
1	200	70	2000S
2	100	30	2001F
3	300	20	2001F
4	200	30	2000S
5	300	70	2000S
6	100	20	2000S

(a)

Númerode-Estudiante	Número de registro relativo
100	2
100	6
200	1
200	4
300	3
300	5

(b)

Número-deClase	Número de registro relativo
20	3
20	6
30	2
30	4
70	1
70	5

(c)

Como usted puede ver, el índice es simplemente una lista ordenada de los NúmerosdeEstudiantes. Para procesar la INSCRIPCIÓN secuencialmente en el NúmerodeEstudiante, simplemente se procesa de manera secuencial el índice, obteniendo los datos de INSCRIPCIÓN mediante la lectura de los registros señalados por los apuntadores. La figura A-8(c) muestra otro índice para la INSCRIPCIÓN, el cual mantiene el orden de NúmerodeClase.

Para utilizar un índice, los datos que se ordenarán (aquí, de INSCRIPCIÓN) deben residir en un archivo secuencial indexado o directo, aunque los índices pueden residir en cualquier tipo de archivo. En la práctica, casi todos los productos DBMS conservan los datos y los índices en archivos directos.

Si usted compara la lista encadenada con el índice notará una diferencia esencial entre ellos. En una lista encadenada los apuntadores se almacenan junto con los datos. Cada registro contiene un campo de enlace (link) con un apuntador de la dirección del siguiente registro relacionado. En cambio, en un índice, los apuntadores se almacenan en los índices, por separado de los datos. Así, los registros de los datos no contienen en sí mismos ningún apuntador. Los productos comerciales DBMS utilizan ambas técnicas.

Árboles B

Una aplicación especial del concepto de índices, o listas invertidas, es un *árbol B*; un índice de niveles múltiples que permite el procesamiento secuencial o directo de los registros de datos. Asimismo, asegura un cierto nivel de eficiencia en el procesamiento, debido a la manera en que están estructurados los índices.

Un árbol B es un índice que consta de dos partes: el conjunto secuencia y el conjunto índice (estos términos los utiliza la documentación de la organización de archivos VSAM de IBM. Usted podrá encontrar otros términos sinónimos). El *conjunto secuencia* es un índice que contiene una entrada para cada registro en el archivo. El índice se encuentra en una secuencia física, usualmente mediante el valor de la llave primaria. Este orden permite el acceso directo y rápido a los registros de datos como sigue: procesa el conjunto secuencia en orden, lee la dirección de cada registro, y después lee el registro.

El *conjunto índice* es el que señala los grupos de las entradas en el índice del conjunto de la secuencia. Este orden permite el acceso directo y rápido a los registros en el archivo, y el conjunto índice es el que hace a los árboles B únicos.

Un ejemplo de un árbol B aparece en la figura A-9, y un caso de su estructura puede apreciarse en la figura A-10. Advierta que en el renglón inferior de la figura A-9, el conjunto secuencia es simplemente un índice. Contiene una entrada para cada registro en el archivo (aunque por brevedad, se han omitido los registros de los datos y sus direcciones). También note que las entradas del conjunto secuencia se encuentran en grupos de tres. Las entradas en cada grupo se encuentran físicamente en secuencia, y cada grupo se encadena al siguiente por medio de una lista encadenada, como puede apreciarlo en la figura A-10.

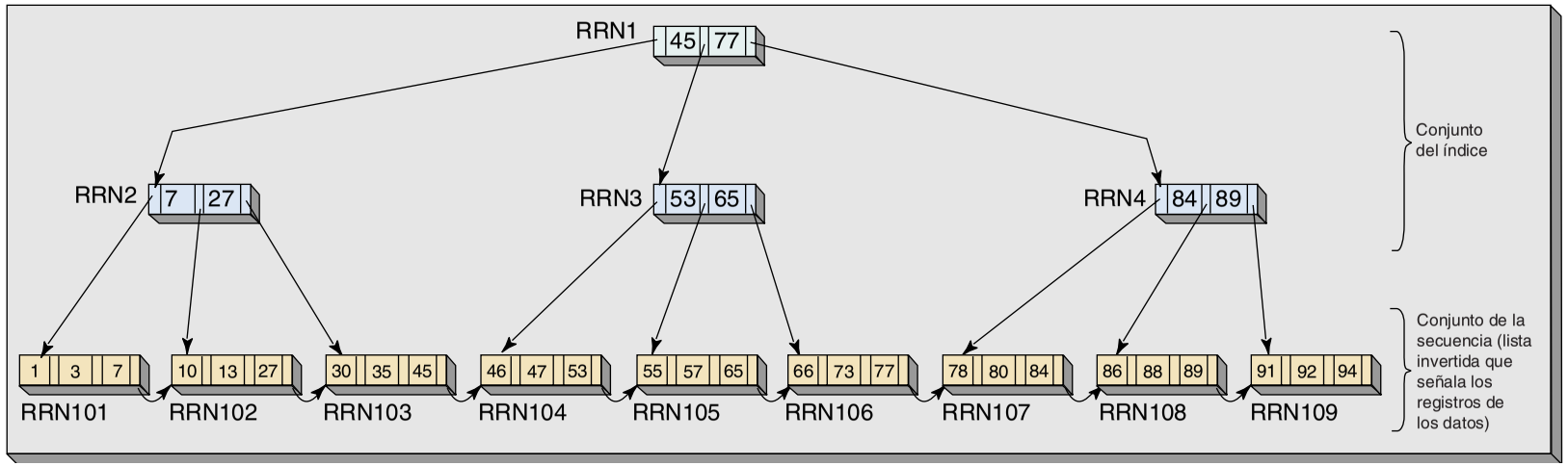
Examine el conjunto del índice en la figura A-9. La entrada superior contiene dos valores: 45 y 77. Siguiendo el enlace de la extrema izquierda (el RRN2), se pueden accesar todos los registros cuyos valores del campo llave sean menores que, o iguales a 45; mediante el seguimiento del apuntador medio (el RRN3) se pueden accesar todos los registros cuyos valores del campo llave sean mayores que 45 y menores que, o iguales a 77; y mediante el seguimiento del apuntador de la derecha (el RRN4) se pueden ingresar todos los registros cuyos valores del campo llave sean mayores que 77.

De manera similar, en el siguiente nivel existen dos valores y tres apuntadores en cada entrada del índice. Cada vez que se va a otro nivel se limita la búsqueda a un registro en particular. Por ejemplo, si se continúa el seguimiento del apuntador de la izquierda desde la entrada superior, y después se sigue el apuntador de la derecha desde ese punto, se pueden accesar todos los registros cuyo valor del campo llave sea mayor que 27 y menores que, o iguales a 45. En el primer nivel se ha eliminado todo lo que es mayor que 45.

Por definición, los árboles B están balanceados. Esto es, todos los registros de datos se encuentran exactamente a la misma distancia de la entrada superior en el conjunto del índice. Este aspecto de los árboles B asegura la eficiencia de la ejecución, aunque los algoritmos para la inserción y eliminación de los registros sean más complejos que

➤ FIGURA A-9

Estructura general de un árbol B sencillo



➤ FIGURA A-10

Caso del árbol B en la figura A-9

RRN	Link 1	Valor 1	Link 2	Valor 2	Link 3
1	2	45	3	77	4
2	101	7	102	27	103
3	104	53	105	65	106
4	107	84	108	89	109

Conjunto del índice

.
.
.

	R1	Dirección 1	R2	Dirección 2	R3	Dirección 3	Link (enlace)
101	1	Apuntador a 6	3	Apuntador a 8	7	Apuntador a 12	102
102	10	· · ·	13	· · ·	27	· · ·	103
103	30	· · ·	35	· · ·	45	· · ·	104
104	46	· · ·	47	· · ·	53	· · ·	105
105	55	· · ·	57	· · ·	65	· · ·	106
106	66	· · ·	73	· · ·	77	· · ·	107
107	78	· · ·	80	· · ·	84	· · ·	108
108	86	· · ·	88	· · ·	89	· · ·	109
109	91	· · ·	92	· · ·	94	· · ·	0

Conjunto de la secuencia (Se omiten las direcciones de los registros de los datos)

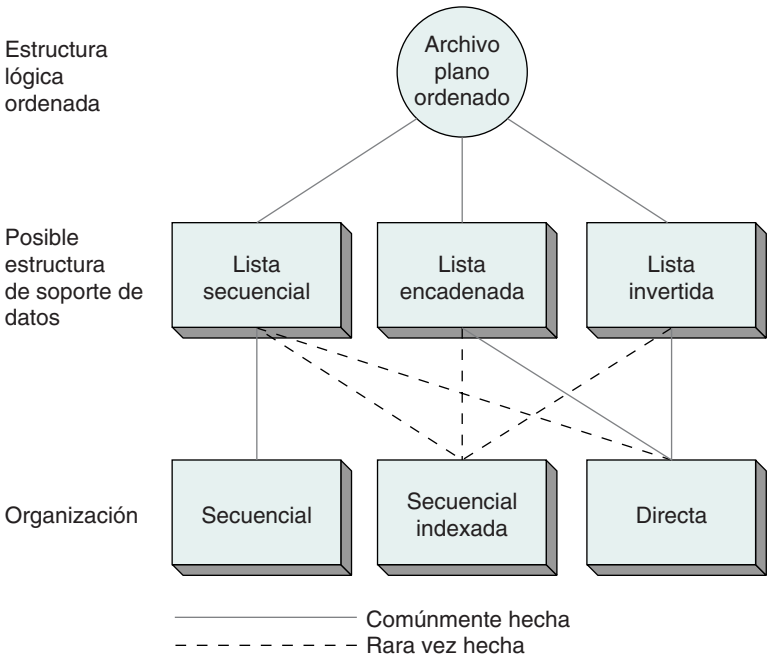
los que se utilizan para los árboles comunes (los cuales pueden estar desbalanceados), debido a que probablemente diversas entradas del índice pudieran ser modificadas cuando se agreguen o eliminen los registros, para mantener todos los datos a la misma distancia de la entrada superior del índice.

Resumen de las estructuras de datos

La figura A-11 resume las técnicas para mantener los archivos planos ordenados. Son posibles tres estructuras de soporte de datos. Se pueden utilizar las listas secuenciales,

➤ FIGURA A-11

Resumen de las estructuras de datos y las organizaciones de datos empleadas para los archivos planos ordenados



pero los datos deben duplicarse para mantener diversos ordenamientos. Debido a que no se utilizan las listas secuenciales en el procesamiento de bases de datos, no las consideraremos más adelante. Las listas encadenadas y los índices pueden emplearse sin la duplicación de datos. Los árboles B son aplicaciones especiales de los índices.

Como se muestra en la figura A-11, las listas secuenciales se pueden almacenar mediante cualquiera de estas tres organizaciones de archivos. Sin embargo, en la práctica por lo general se conservan en archivos secuenciales. Además, aunque las listas encadenadas y los índices pueden almacenarse mediante archivos secuenciales indexados o directos, casi siempre los productos DBMS las almacenan en archivos directos.

➤ REPRESENTACIÓN DE LAS RELACIONES BINARIAS

En esta sección examinaremos cómo se pueden representar cada una de las relaciones especializadas de los registros que analizamos en el capítulo 6 —árboles, redes simples, y redes complejas— mediante las listas encadenadas y los índices.

REPASO DE LAS RELACIONES DE LOS REGISTROS

Los registros pueden relacionarse de tres maneras. Una relación de *árbol* tiene una o más relaciones uno a muchos, pero cada registro de un hijo tiene cuando mucho un padre. El caso de los datos de la facultad mostrados en la figura A-12 ilustra un árbol. Existen diversas relaciones 1:N, pero cualquier registro de un hijo tiene sólo un padre, como se muestra en la figura A-13.

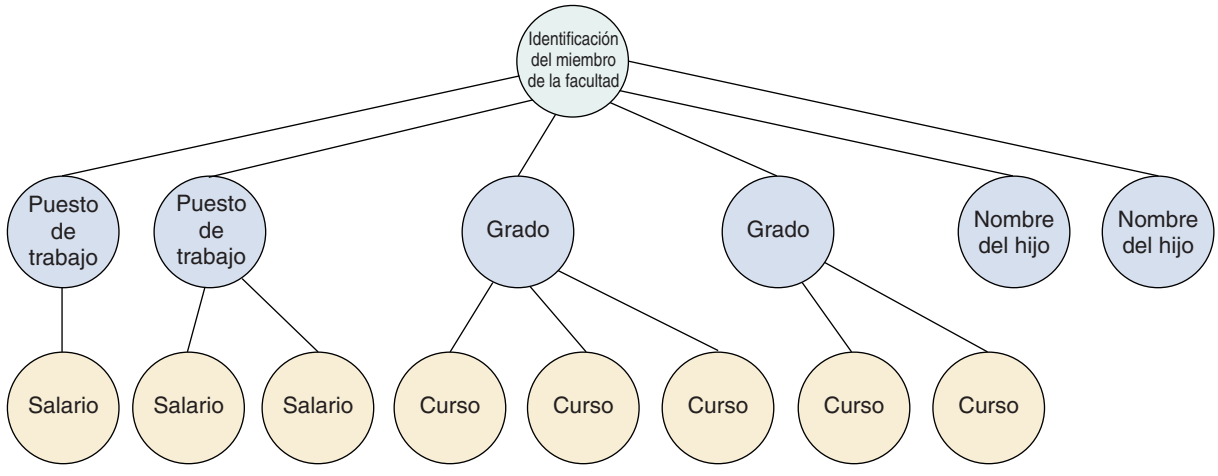
Una *red simple* es una colección de los registros y de las relaciones 1:N entre ellos. Lo que distingue a una red simple de un árbol, es el hecho de que en una red simple un hijo puede tener más de un padre, siempre y cuando los padres sean diferentes tipos de registros. En la figura A-15 se presenta esquemáticamente el caso de una red simple de estudiantes, consejeros y campos especializados de estudio en la figura A-14.

Una *red compleja* es también un conjunto de registros y relaciones, pero éstas son muchos a muchos, en lugar de uno a muchos. La relación entre los estudiantes y las clases es una red compleja. Un caso de esta relación puede apreciarse en la figura A-16, y el esquema general se encuentra en la figura A-17.

Anteriormente vimos que podemos emplear las listas encadenadas y los índices para procesar los registros en ordenamientos diferentes a aquel en el que se almacenan físicamente. Asimismo, podemos utilizar esas mismas estructuras de datos para almacenar y procesar las relaciones entre los registros.

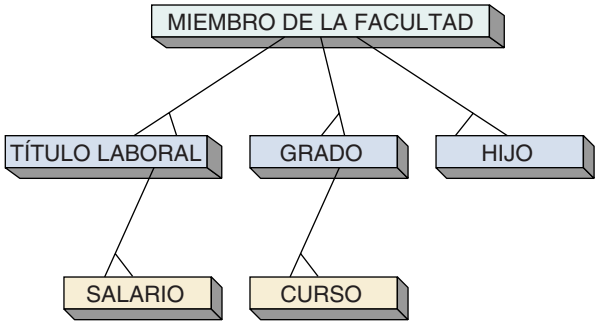
➤ FIGURA A-12

Caso del registro de un miembro de la facultad



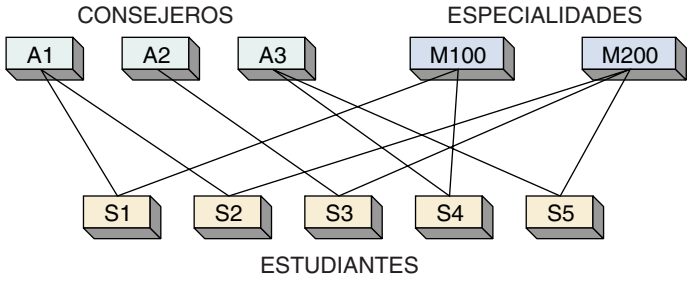
➤ FIGURA A-13

Esquema de la estructura de árbol de un miembro de la facultad



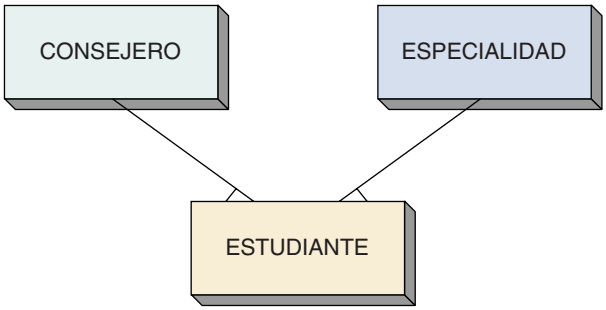
➤ FIGURA A-14

Caso de una red simple



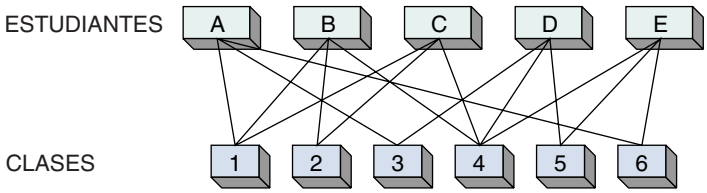
➤ FIGURA A-15

Estructura general de una red simple



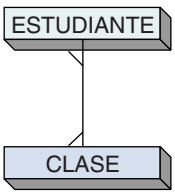
➤ FIGURA A-16

Caso de una red compleja



➤ FIGURA A-17

Esquema de una red compleja

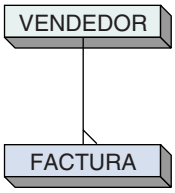


Representación de árboles

Podemos emplear listas secuenciales, listas encadenadas, e índices para representar árboles. Cuando usamos las listas secuenciales duplicamos muchos datos, y además, los productos DBMS no utilizan las listas secuenciales para representar árboles. Por lo tanto, sólo describimos listas encadenadas e índices.

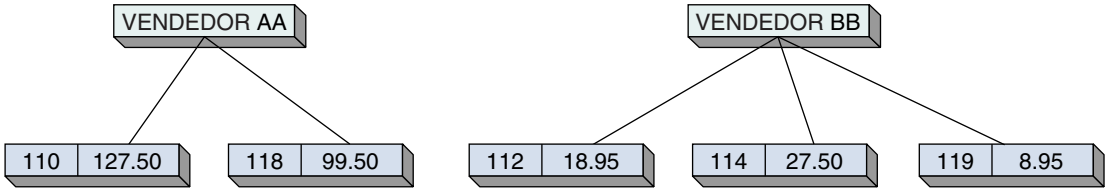
➤ FIGURA A-18

Ejemplo de un árbol que relaciona los registros de VENDEDOR y FACTURA



➤ FIGURA A-19

Dos casos del árbol VENDEDOR-FACTURA



REPRESENTACIÓN DE ÁRBOLES MEDIANTE LISTAS ENCADENADAS. La figura A-18 muestra una estructura de árbol en la cual los registros de VENDEDOR son los padres y los registros de FACTURA son los hijos. La figura A-19 muestra dos casos de esta estructura: en la figura A-20 el VENDEDOR AA se encuentra en el número de registro relativo 1 (RRN1), y el VENDEDOR BB se encuentra en el número de registro relativo 2. Como se muestra, los registros de FACTURA se han almacenado en los registros subsecuentes. Observe que estos registros no están almacenados en ningún orden particular, ni necesitan estarlo.

El problema es que a partir de este archivo no podemos afirmar qué facturas pertenecen a cuáles vendedores. Para resolver este problema con una lista encadenada, le agregamos un campo apuntador a cada registro. En este campo se almacena la dirección de algún otro registro relacionado. Por ejemplo, en el campo de enlace (link) del VENDEDOR AA se coloca la dirección de la primera factura correspondiente a éste. Éste es el RRN7, el cual es la Factura 110. Después se hace que la Factura 110 se dirija a la siguiente factura correspondiente al VENDEDOR AA, en este caso RRN3. Esta muesca sostiene a Factura 118. Para indicar que ya no hay más hijos en la cadena, se inserta un 0 en el campo de enlace (link) para el RRN3.

Esta técnica se muestra en la figura A-21. Si usted examina la figura verá que se utilizan un conjunto de enlaces (links) similares para representar la relación entre el VENDEDOR BB y sus facturas.

Es más fácil modificar la estructura en la figura A-21, que modificar una lista secuencial de los registros. Por ejemplo, suponga que se agrega una nueva factura, por decir algo la número 111, al VENDEDOR AA. Para realizar esto, sólo se agrega el regis-

➤ FIGURA A-20

Representación en archivo de los árboles en la figura A-19

Número de registro	Contenidos del registro	
1	VENDEDOR AA	
2	VENDEDOR BB	
3	118	99.50
4	119	8.95
5	112	18.95
6	114	27.50
7	110	127.50

➤ FIGURA A-21

Árbol de ocurrencias representadas por listas encadenadas

Número de registro relativo	Contenidos del registro		Campo de enlace
1	VENDEDOR AA		7
2	VENDEDOR BB		5
3	118	99.50	0
4	119	8.95	0
5	112	18.95	6
6	114	27.50	4
7	110	127.50	3

tro al archivo y se inserta en la lista encadenada. Físicamente, el registro puede colocarse donde sea. ¿Pero dónde debe colocarse lógicamente? A menudo la aplicación tendrá un requerimiento, como por ejemplo que se conserven los hijos en orden ascendente del número de factura. Si es así, se necesita hacer que la Factura 110 apunte a Factura 111 (en el RRN8), y se necesita hacer que Factura 111, la factura nueva, apunte a Factura 118 (en el RRN3). Esta modificación se muestra en la figura A-22.

De igual manera, es fácil eliminar una factura. Si se eliminara Factura 114, simplemente modificaríamos el apuntador en la factura que ahora está apuntando a Factura 114. En este caso, Factura 112 se encuentra en el RRN5. Se le otorga a Factura 112 el apuntador que tenía Factura 114 antes de la eliminación. De este modo, Factura 112 apunta a Factura 119 (véase figura A-23). En realidad hemos quitado un enlace (link) de la cadena y unido a los que antes conectaba.

REPRESENTACIÓN DE ÁRBOLES MEDIANTE ÍNDICES. Una estructura de árbol se puede representar fácilmente mediante los índices. La técnica es almacenar cada relación uno a muchos como un índice. Posteriormente se utilizan estas listas para asociar a los padres con los hijos.

Mediante los registros VENDEDOR y FACTURA en la figura A-21, se aprecia que VENDEDOR AA (en el RRN1) posee las FACTURAs 110 (RRN7) y 118 (RRN3). Por consiguiente, el RRN1 es el padre del RRN7 y del RRN3. Este hecho puede representarse con el índice en la figura A-24. La lista simplemente asocia la dirección de un padre con las de cada uno de sus hijos.

Si el árbol tiene varias relaciones 1:N, entonces se requerirán de varios índices, uno para cada una. En la estructura en la figura A-13 se necesitan cinco índices.

Representación de redes simples

Como los árboles, las redes simples también se pueden representar mediante listas encadenadas e índices.

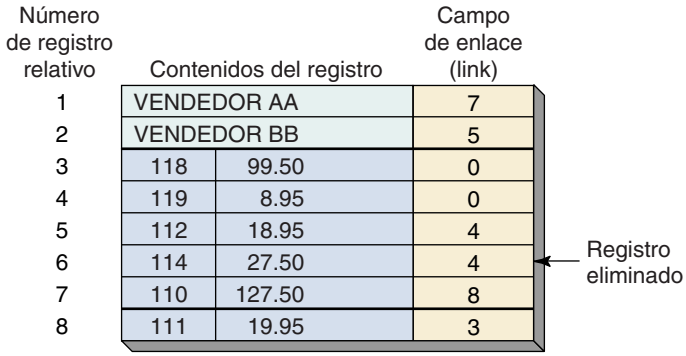
➤ FIGURA A-22

Se inserta la factura 111 en archivo en la figura A-21

Número de registro relativo	Contenidos del registro		Campo de enlace (link)	
1	VENDEDOR AA		7	
2	VENDEDOR BB		5	
3	118	99.50	0	
4	119	8.95	0	
5	112	18.95	6	
6	114	27.50	4	
7	110	127.50	8	
8	111	19.95	3	← Registro insertado

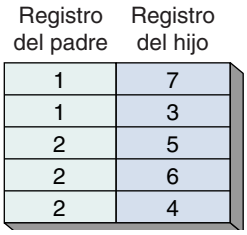
➤ FIGURA A-23

Se elimina la factura del archivo en la figura A-22



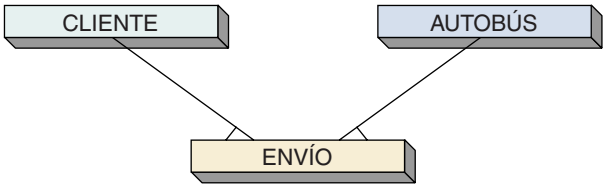
➤ FIGURA A-24

Representación de índices de la relación VENDEDOR-FACTURA



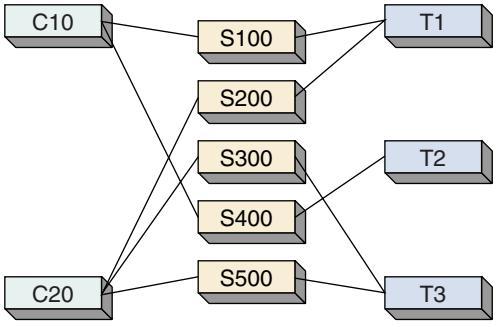
➤ FIGURA A-25

Estructura de red simple



➤ FIGURA A-26

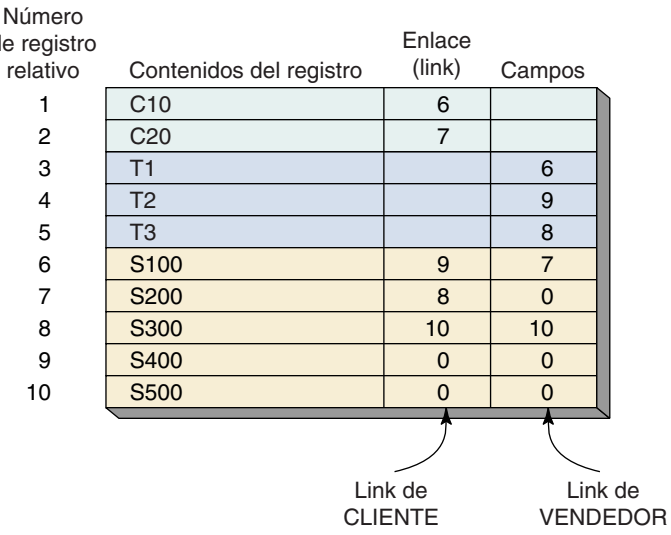
Ocurrencia de la red simple en la figura A-25



REPRESENTACIÓN DE REDES SIMPLES MEDIANTE LISTAS ENCADENADAS. Considere la red simple en la figura A-25. Es una red simple puesto que todas las relaciones son 1:N, y los registros de ENVÍO tienen dos padres de diferente tipo. Cada ENVÍO tiene un padre CLIENTE y un padre AUTOBÚS. La relación entre CLIENTE y ENVÍO es 1:N debido a que un cliente puede tener varios envíos, y la relación de AUTOBÚS a ENVÍO es 1:N, puesto que un autobús puede contener diversos envíos (suponiendo que éstos son lo suficientemente pequeños como para caber en uno, o en menos de un autobús). Un caso de esta red se muestra en la figura A-26.

➤ FIGURA A-27

Representación de una red simple con listas encadenadas



Para representar esta red simple con listas encadenadas se necesita establecer un conjunto de apuntadores para cada relación 1:N. En este ejemplo, eso significa que un conjunto de apuntadores conectará los CLIENTEs con sus ENVÍOs, y que otro conjunto conectará los CAMIONes con sus ENVÍOs. Por consiguiente, un registro CLIENTE contendrá un apuntador (al primer ENVÍO que posea); un registro AUTOBÚS tendrá dos apuntadores, uno para el siguiente ENVÍO que posea el mismo CLIENTE y otro para el siguiente ENVÍO que posea el mismo AUTOBÚS. Este esquema se ilustra en la figura A-27.

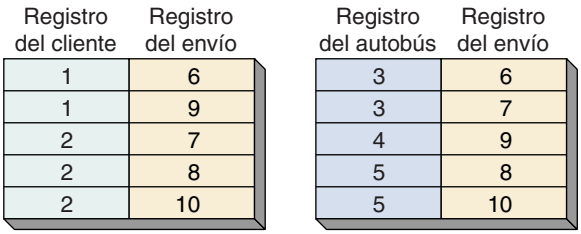
Una red simple tiene cuando menos dos relaciones 1:N, cada una de las cuales se puede representar mediante un índice, como se explicó en el análisis de los árboles. Por ejemplo, considere la red simple mostrada en la figura A-25. Ésta tiene dos relaciones 1:N: una entre AUTOBÚS y ENVÍO y una entre CLIENTE y ENVÍO. Cada una de estas relaciones se puede almacenar en un índice. La figura A-28 muestra los dos índices necesarios para representar el ejemplo en la figura A-26. Suponga que los registros se localizan en las mismas posiciones que en la figura A-27.

Representación de redes complejas

Las redes complejas se pueden representar físicamente en una gran variedad de maneras. Pueden descomponerse en árboles o redes simples, y así estas estructuras más simples pueden representarse mediante una de las técnicas ya descritas. Alternativamente, éstas se pueden representar de manera directa mediante índices. Ningún producto

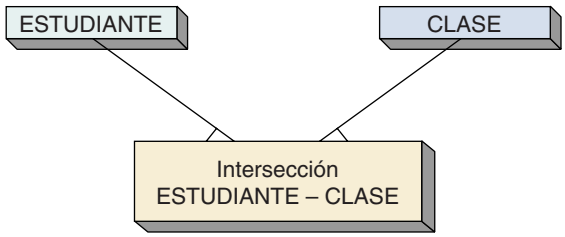
➤ FIGURA A-28

Representación de una red simple con índices



➤ FIGURA A-29

Descomposición de una red compleja en una red simple



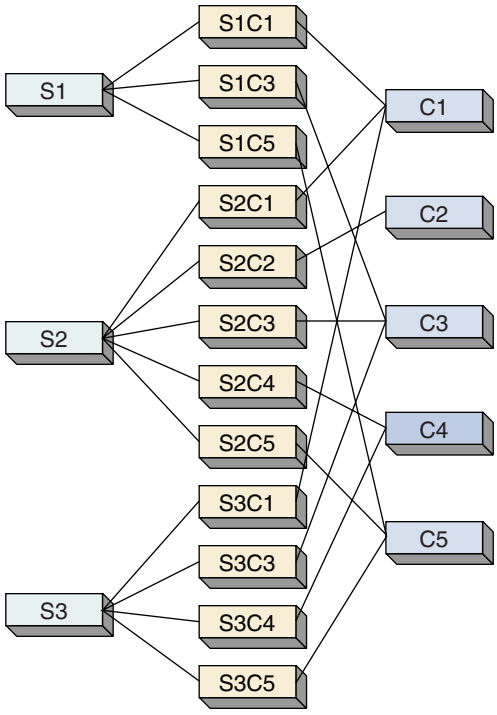
DBMS emplea las listas encadenadas para representar directamente redes complejas. En la práctica, casi siempre las redes complejas se descomponen en estructuras más simples, por lo que sólo se consideran aquellas representaciones que emplean la descomposición.

Un enfoque común para representar las redes complejas es reducirlas a redes simples y después representar a las redes simples con listas encadenadas o índices. Sin embargo, advierta que una red compleja implica una relación entre dos registros, mientras que una red simple involucra a las relaciones entre tres registros. Por consiguiente, para descomponer una red compleja en una simple se necesita crear un tercer tipo de registro.

El registro que se crea cuando se descompone una red compleja en una simple, se denomina *registro de intersección*. Considere la red compleja ClasedelEstudiante. Un registro de intersección contendrá una llave única del registro de ESTUDIANTE y una llave única de un registro de CLASE correspondiente. No contendrá los datos de aplicación, a pesar de que deba contener los campos de enlace (link). La estructura general de esta relación se muestra en la figura A-29. Supongamos que los nombres de los registros son únicos (tales como S1, S2, y C1), se ilustra una ocurrencia de la relación ESTUDIANTE-CLASE en la figura A-30.

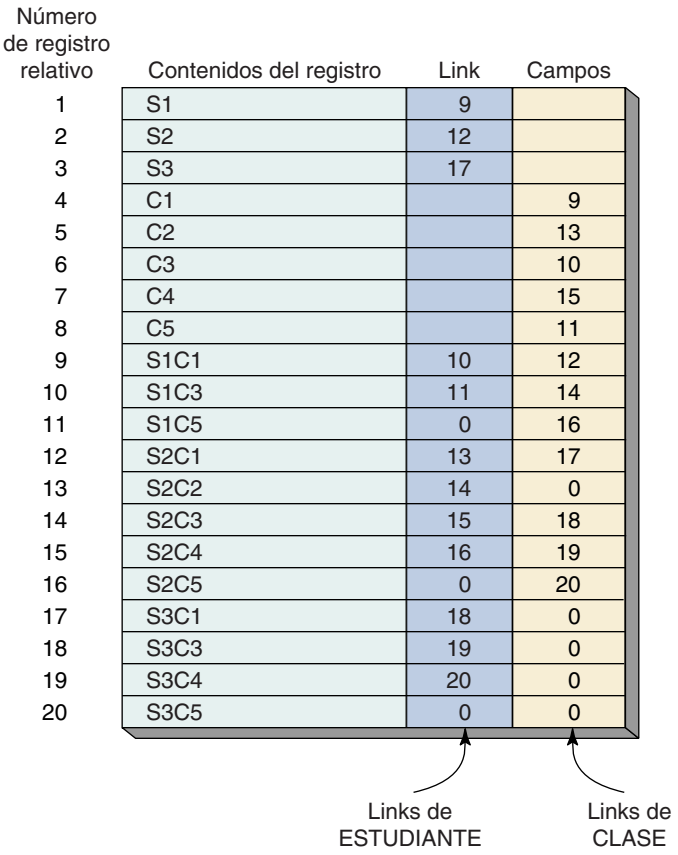
➤ FIGURA A-30

Ocurrencia de la relación ESTUDIANTE-CLASE que muestra los registros de intersección



➤ FIGURA A-31

Ocurrencia de la red en la figura A-30



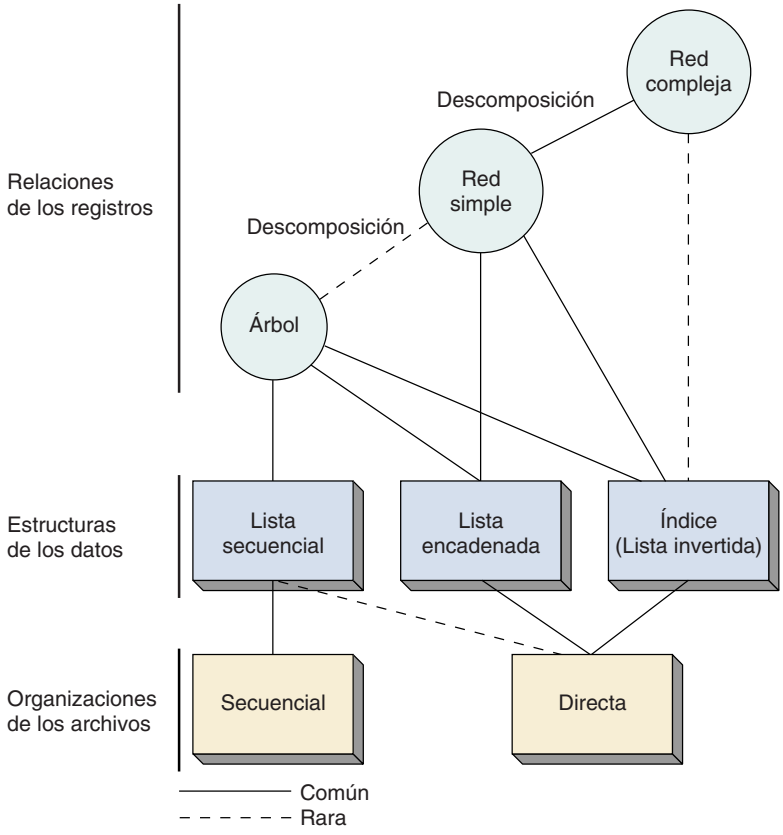
Advierta que la relación entre el ESTUDIANTE y el registro de intersección, y aquella entre la CLASE y el registro de intersección son 1:N. Por lo tanto, se ha creado una red simple que ahora puede representarse con las técnicas de listas encadenadas, o índices, descritas anteriormente. En la figura A-31 se muestra un archivo de esta ocurrencia mediante la técnica de listas encadenadas.

Resumen de las representaciones de las relaciones

La figura A-32 resume las representaciones de las relaciones de los registros. Los árboles pueden representarse mediante listas secuenciales (aunque no se analizó este enfoque), listas encadenadas, o índices. Las listas secuenciales no se utilizan en los productos DBMS. Una red simple puede descomponerse en árboles y después representarse, o pueden representarse directamente mediante las listas encadenadas o índices. Por último, una red compleja puede descomponerse en un árbol o en una red simple (mediante los registros de intersección), o puede representarse directamente mediante índices.

➤ FIGURA A-32

Relaciones de los registros, estructuras de los datos y organizaciones de los archivos



➤ REPRESENTACIONES DE LAS LLAVES SECUNDARIAS

En muchos casos la palabra *llave* indica un campo (o campos) cuyo valor únicamente identifica un registro. A menudo se le llama la *llave primaria*. Sin embargo, algunas veces las aplicaciones necesitan accesar y procesar los registros mediante una *llave secundaria*, la cual es diferente a la llave primaria. Las llaves secundarias pueden ser únicas (tales como el nombre de un profesor), o no únicas (tales como el código postal de un cliente). En esta sección utilizaremos el término *conjunto* para referirnos a todos los registros que tengan el mismo valor de una llave secundaria no única; por ejemplo, un conjunto de registros que tengan el código postal 98040.

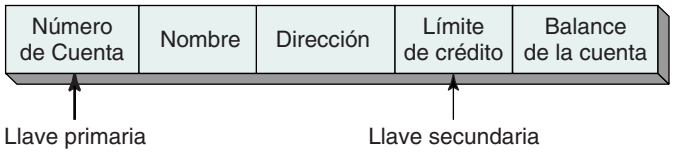
Las listas encadenadas y los índices se utilizan para representar las llaves secundarias, pero las listas encadenadas sólo son prácticas para las llaves no únicas. Sin embargo, los índices se pueden utilizar para las representaciones de llaves únicas y no únicas.

Representación de las llaves secundarias mediante listas encadenadas

Considere el ejemplo de los registros CLIENTE que se muestran en la figura A-33. La llave primaria es NúmerodeCuenta, y existe una llave secundaria en LímitedeCrédito. Los valores posibles del LímitedeCrédito son 500, 700, y 1000. Por consiguiente, existirá un conjunto de registros para el límite de 500, otro para 700, y uno más para 1000.

➤ FIGURA A-33

Registro CLIENTE



Para representar esta llave mediante listas encadenadas se le agrega un campo de enlace (link) a los registros de CLIENTE. Dentro de este link se crea una lista encadenada para cada conjunto de registros. La figura A-34 muestra una base de datos de once clientes, pero, por brevedad, sólo se muestran el NúmerodeCuenta y el LímitedeCrédito. Se ha adjuntado un campo de link a los registros. Suponga que un registro de una base de datos ocupa un registro físico en un archivo directo, mediante la dirección del registro relativo.

Se necesita establecer tres apuntadores para saber dónde inicia cada lista encadenada. Éstos se llaman *cabezas* y se almacenan por separado de los datos. La cabeza de la lista encadenada $500 es RRN1. El registro 1 se encadena con el registro 2, el cual en cambio se encadena con el registro 7. El registro 7 tiene un cero en la posición de enlace (link), lo cual indica que ése es el final de la lista. Consecuentemente, el conjunto del límite de crédito $500 consta de los registros 1, 2, y 7. De manera similar, el conjunto $700 contiene los registros 3, 5, y 10, y el conjunto $1000 los registros relativos 4, 6, 8, 9 y 11.

Para responder a la pregunta: ¿cuántas cuentas superiores a 900 tiene un balance en el conjunto 1000?, se puede utilizar la lista encadenada del conjunto 1000. De esta manera, sólo se necesitarían leer del archivo y examinar los registros en el conjunto 1000. Aunque la ventaja de este enfoque no es fácilmente distinguible en este pequeño ejemplo, suponga que existen 100000 registros de CLIENTE, y sólo 100 de ellos se encuentran en el conjunto 1000. Si existiera una lista encadenada se deberían examinar todos los registros de 100000, pero con la lista encadenada sólo es necesario examinar 100 registros; es decir, los del conjunto 1000. Por lo tanto, mediante la lista encadenada se ahorra 99900 lecturas.

Las listas encadenadas no son una técnica eficaz para cada aplicación de la llave secundaria. En particular, si los registros no se procesan secuencialmente en un conjunto, las listas encadenadas son ineficaces. Por ejemplo, si a menudo es necesario encontrar el décimo, el duodécimo o el *n* registro en el conjunto de Límite de crédito 500, el procesamiento será lento. Las listas encadenadas son ineficaces para el acceso directo.

Además, si la aplicación requiere que se creen o se destruyan automáticamente las llaves secundarias, el enfoque de la lista encadenada no será conveniente. Cada vez que se crea un diseño nuevo, se le debe agregar un campo de enlace (link) a cada registro, lo cual frecuentemente requiere de la reorganización de la base de datos, un proceso costoso y que lleva mucho tiempo.

Por último, si las llaves secundarias son únicas, cada lista tendrá una longitud de uno y existirá una lista encadenada por separado para cada registro en la base de datos. Debido a que no se puede trabajar con esta situación, no se deben utilizar las listas encadenadas para las llaves únicas. Por ejemplo, suponga que los registros de CLIENTE contienen otro campo único; por ejemplo, Número del Seguro Social. Si se tratara de representar esta llave secundaria única mediante una lista encadenada, cada lista tendría sólo un artículo en ella; esto es, el registro simple que tuviera el Número del Seguro Social indicado.

➤ FIGURA A-34

Representación de la llave secundaria del límite de crédito mediante listas encadenadas

Número del registro relativo	Link	Número de cuenta	Límite de crédito	Otros datos
1	2	101	500	
2	7	301	500	
3	5	203	700	
4	6	004	1000	
5	10	204	700	
6	8	905	1000	
7	0	705	500	
8	9	207	1000	
9	11	309	1000	
10	0	409	700	
11	0	210	1000	

CABEZA-500 = 1
CABEZA-700 = 3
CABEZA-1000 = 4

➤ FIGURA A-35

Representación de una llave secundaria única mediante índices: (a) ejemplo de los datos de CLIENTE (con el SSN,) y (b) índice para la llave secundaria SSN

Número del registro relativo	Número de cuenta	Límite de crédito	Número de Seguridad Social (SSN)
1	101	500	000-01-0001
2	301	500	000-01-0005
3	203	700	000-01-0009
4	004	1000	000-01-0003

(a)

SSN	Número del registro relativo
000-01-0001	1
000-01-0003	4
000-01-0005	2
000-01-0009	3

(b)

Representación de las llaves secundarias mediante índices

Una segunda técnica para la representación de las llaves secundarias emplea un índice: uno por cada llave secundaria. El enfoque varía dependiendo de que los valores de la llave sean únicos o no.

LLAVES SECUNDARIAS ÚNICAS. Suponga que los registros de CLIENTE en la figura A-33 contienen el Número del Seguro Social (SSN), así como los campos mostrados. Para proporcionar el acceso de la llave de los registros CLIENTE mediante el SSN, simplemente se construye un índice en el campo SSN. El ejemplo de los datos CLIENTE se muestra en la figura A-35(a), y en la figura A-35(b) se ilustra un índice correspondiente. Este índice utiliza los números de registro relativos, como por ejemplo las direcciones. En su lugar se podrían utilizar NúmerodeCuenta, en tal caso el DBMS localizaría en el índice el SSN deseado, obtendría el NúmerodeCuenta asociado, y entonces convertiría el NúmerodeCuenta en la dirección de un registro relativo.

LLAVES SECUNDARIAS NO ÚNICAS. Los índices también se pueden utilizar para representar las llaves secundarias no únicas, pero debido a que cada conjunto de los registros relacionados puede contener un número desconocido de miembros; las entradas en el índice son de longitud variable. Por ejemplo, la figura A-36 muestra el índice para los conjuntos de LímitedeCrédito para los datos CLIENTE. Los conjuntos $500 y $700 tienen tres miembros, de esta manera existen tres números de cuenta en cada entrada. El conjunto $1000 tiene cinco miembros, por lo que existen cinco números de cuenta en dicha entrada.

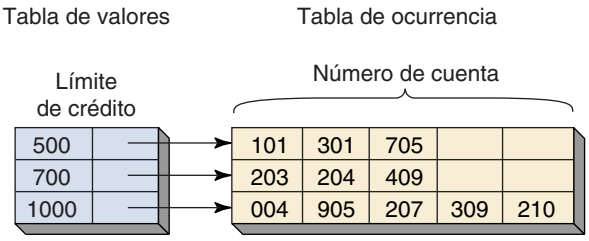
En realidad, la representación y el procesamiento de llaves secundarias no únicas son tareas complejas. Los productos comerciales DBMS utilizan diversos esquemas. Un método común utiliza una tabla de valores y una tabla de ocurrencia. Cada entrada de la tabla de valores consta de dos campos, el primero tiene un valor de la llave. Para la llave LímitedeCrédito de CLIENTE, los valores son 500, 700, y 1000. El segundo campo de la entrada de la tabla de valores es un apuntador dentro de la tabla de ocurrencia. La

➤ FIGURA A-36

Índice para la llave LímitedeCrédito en la figura A-33

Límite de crédito	Número de cuenta				
500	101	301	705		
700	203	204	409		
1000	004	905	207	309	210

➤ FIGURA A-37



Tablas de valores y de ocurrencia para la llave del Límitede-Crédito en la figura A-33

tabla de ocurrencia contiene las direcciones de los registros, y en la tabla aparecen juntas aquéllas que tienen un valor común en el campo de la llave secundaria. La figura A-37 muestra las tablas de valores y de ocurrencia para la llave LímitedeCrédito.

Para localizar los registros que tienen un valor determinado de la llave secundaria, se busca en la tabla de valores el valor deseado. Una vez que se localiza en la tabla de valores el valor de la llave determinada, el apuntador señala la tabla de ocurrencia para obtener las direcciones de aquellos registros que tengan dicho valor de la llave. Posteriormente se utilizan estas direcciones para obtener los registros deseados.

Cuando se inserta un registro nuevo en el archivo, el DBMS debe modificar los índices para cada campo de la llave secundaria. Para las llaves no únicas el DBMS se debe asegurar que el valor de la llave del nuevo registro se encuentre en la tabla de valores. Si es así, agregará la dirección del nuevo registro a la entrada apropiada en la tabla de ocurrencia. De lo contrario, insertará nuevas entradas en las tablas de valores y de ocurrencia.

Cuando se elimine un archivo se debe remover su dirección de la tabla de ocurrencia. Si no permanece ninguna dirección en la entrada de la tabla de ocurrencia, deberán eliminarse también los valores correspondientes.

Cuando se modifica el campo de la llave secundaria de un registro se debe eliminar la dirección del registro de una entrada de la tabla de ocurrencia, e insertarse en otra. Si la modificación es un valor nuevo para la llave hay que agregar una entrada a la tabla de valores.

El enfoque del índice para la representación de llaves secundarias origina objeciones al enfoque de la lista encadenada. Es posible el procesamiento directo de los conjuntos; por ejemplo, el tercer registro en un conjunto puede restaurarse sin procesar el primero o el que sigue. Asimismo, es posible crear y eliminar las llaves secundarias de manera dinámica. No se realiza ningún cambio en los registros por sí mismos; el DBMS crea tablas de valores y de ocurrencia adicionales. Por último, las llaves únicas se pueden procesar de forma eficaz.

Las desventajas del enfoque del índice son que se requiere más espacio de archivos (las tablas ocupan más que los apuntadores), y que la tarea de programación del DBMS es más complicada. Observe que la tarea de la programación de la aplicación no es más o menos difícil —pero es más complicado escribir el software DBMS que procese los índices, que escribir el software que procese las listas encadenadas—. Por último, en general, las modificaciones se procesan más lentamente debido a las acciones de lectura y escritura que se requieren para accesar y mantener los valores en las tablas de ocurrencia.

➤ RESUMEN

En este apéndice analizamos las estructuras de datos que se utilizan en el procesamiento de bases de datos. Un archivo plano es un archivo que consta de grupos no repetitivos. Los archivos planos se pueden ordenar mediante las listas secuenciales (colocación física de los registros en la secuencia en la que se procesarán), las listas encadenadas (agregación a cada registro de datos de un apuntador de otro registro lógicamente relacionado), y los índices (construcción de una tabla, separada de los registros de datos, que contenga los apuntadores de los registros relacionados). Los árboles B son aplicaciones especiales de los índices.

Las listas secuenciales, las listas encadenadas y los índices (o listas invertidas) son estructuras de datos fundamentales. Sin embargo, las listas secuenciales rara vez se utilizan en el procesamiento de bases de datos. Estas estructuras de datos se pueden emplear para representar las relaciones de los registros y también las llaves secundarias.

Las tres estructuras básicas de los registros —árboles, redes simples, y redes complejas— se pueden representar mediante las listas encadenadas y los índices. Las redes simples pueden descomponerse en árboles y entonces ser representadas; las redes complejas se pueden descomponer en redes simples que contengan un registro de intersección y entonces ser representadas.

Las llaves secundarias se utilizan para ingresar los datos en algún campo cerca de la llave primaria. Las llaves secundarias pueden ser únicas o no únicas. Las llaves secundarias no únicas pueden representarse con las listas encadenadas y los índices. Las llaves secundarias únicas sólo pueden representarse con índices.

➤ PREGUNTAS DEL GRUPO I

A.1 Defina un archivo plano. Dé un ejemplo (diferente al de este texto) sobre un archivo plano y otro ejemplo de un archivo que no sea plano.

A.2 Muestre cómo se pueden utilizar las listas secuenciales para mantener al archivo en la Pregunta A.1 en dos ordenamientos distintos simultáneamente.

A.3 Muestre cómo pueden utilizarse las listas encadenadas para mantener al archivo en la Pregunta A.1 en dos ordenamientos distintos simultáneamente.

A.4 Muestre cómo se pueden utilizar los índices para mantener al archivo en la Pregunta A.1 en dos ordenamientos distintos simultáneamente.

A.5 Defina un árbol y dé un ejemplo sobre su estructura.

A.6 Mencione una ocurrencia del árbol en la Pregunta A.5.

A.7 Represente la ocurrencia en la Pregunta A.6 mediante listas encadenadas.

A.8 Represente la ocurrencia en la Pregunta A.6 mediante índices.

A.9 Defina una red simple y dé un ejemplo de la estructura.

A.10 Mencione una ocurrencia de la red simple en la Pregunta A.9.

A.11 Represente la ocurrencia en la Pregunta A.10 mediante listas encadenadas.

A.12 Represente la ocurrencia en la Pregunta A.10 mediante índices.

A.13 Defina una *red compleja* y proporcione un ejemplo de la estructura.

A.14 Dé una ocurrencia de la red compleja en la Pregunta A.13.

A.15 Descomponga la red compleja en la Pregunta A.14 en una red simple, y represente una ocurrencia de ésta mediante índices.

A.16 Explique la diferencia entre las llaves primarias y las llaves secundarias.

A.17 Explique la diferencia entre llaves únicas y no únicas.

A.18 Defina un archivo que contenga una llave secundaria. Represente una ocurrencia de dicho archivo mediante un índice en la llave secundaria.

A.19 Defina una llave secundaria no única para el archivo en la Pregunta A.18. Represente una ocurrencia de dicho archivo mediante una lista encadenada en la llave secundaria.

A.20 Realice la misma tarea que en la Pregunta A.19, pero para representar la llave secundaria mediante un índice.

➤ PREGUNTAS DEL GRUPO II

A.21 Desarrolle un algoritmo para producir un reporte que liste las ID (Identificaciones) de los estudiantes inscritos en cada clase, mediante la estructura de la lista encadenada en la figura A-4.

A.22 Desarrolle un algoritmo para insertar los registros en la estructura en la figura A-4. La estructura resultante debe ser similar a la de la figura A-5.

A.23 Desarrolle un algoritmo para producir un reporte que liste las ID de los estudiantes inscritos en cada clase, mediante la estructura del índice que se muestra en las Figuras A-8(a), (b), y (c).

A.24 Desarrolle un algoritmo para insertar un registro en la estructura en la figura A-8(a), que asegure la modificación de los índices asociados en la figura A-8(b) y (c).

A.25 Desarrolle un algoritmo para eliminar un registro de la estructura en la figura A-34, el cual muestre una llave secundaria representada con una lista encadenada. Si se eliminan todos los registros para una de las categorías del límite de crédito (por ejemplo, $1000), ¿debe eliminarse también el apuntador del encabezado asociado? ¿Por qué?

A.26 Desarrolle un algoritmo para insertar un registro en la estructura que se muestra en la figura A-34. Suponga que el registro nuevo tiene un valor del límite de crédito diferente a los ya establecidos. ¿Debe insertarse el registro y establecerse una nueva lista encadenada? ¿O debe rechazarse el registro? ¿Quién debe tomar esa decisión?

APÉNDICE B

Modelos de objeto semántico con Tabledesigner

El diseñador de tablas (Tabledesigner) es una aplicación Windows 95, 98, NT y Windows 2000 que usted puede utilizar para crear modelos de objeto semántico y transformarlos dentro de Microsoft Access u otras bases de datos. Además, puede hacer ingeniería de reversa de una base de datos para crear modelos de objeto semántico a partir de ella. Una vez que haya hecho esto, puede modificar el modelo y el Tabledesigner cambiará el esquema de la base de datos asociada (y los datos) de la manera adecuada. Por último, puede utilizar el Tabledesigner para generar páginas ASP con el fin de crear, leer, actualizar y eliminar las vistas de objeto semántico desde cualquier explorador, utilizando solamente HTML. Para esta última función, necesitará colocar las páginas ASP generadas en un servidor de la red que esté ejecutando IIS, o en un servidor de la red personal con extensiones ASP. En la figura B-1 se muestra un resumen de estas funciones.

Para obtener una copia del Tabledesigner ingrese a www.prenhall.com/kroenke/ donde encontrará instrucciones de cómo obtenerlo. Una vez que haya cargado el software, instálelo según las instrucciones. No hay cargo por el uso de este producto en conjunto con su clase de base de datos. El Tabledesigner incluye una documentación entendible. Haga clic en ayuda y verá el despliegue que se muestra en la figura B-2. Tal vez quiera terminar este análisis con esa documentación —especialmente el análisis en el libro "Aprendiendo a Usar el Desarrollador del Diseñador de Tabla" cuyo contenido se muestra en la figura B-2.

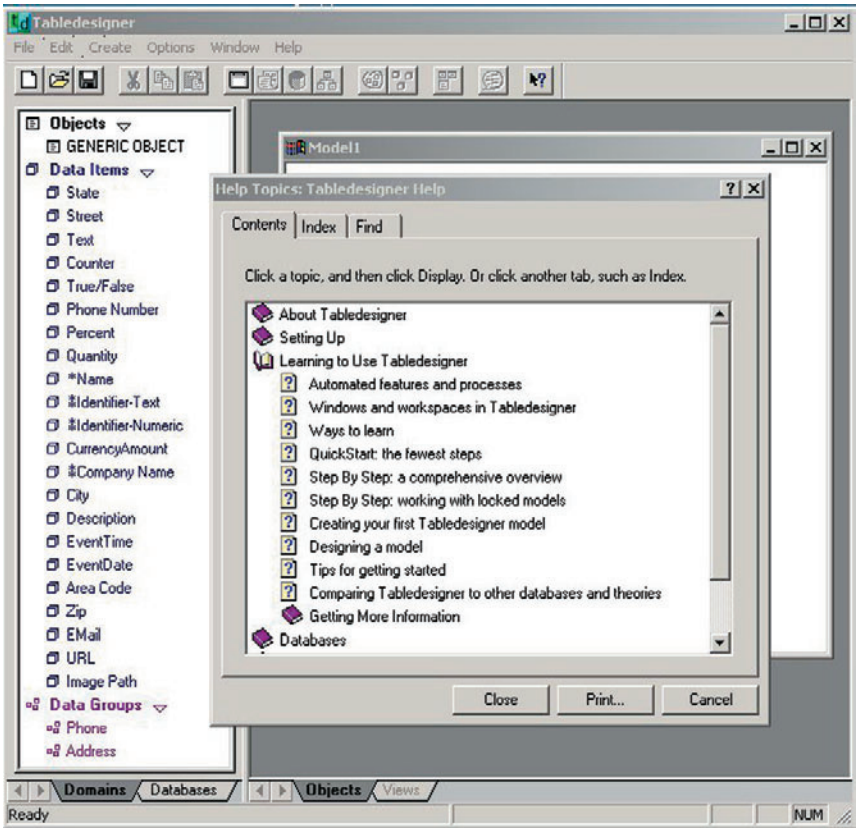
➤ FIGURA B-1

Funciones del Tabledesigner

• Para crear un modelo SOM:
Genere una base de datos Access a partir del modelo
Genere una base de datos en el SQL Server
Genere una instrucción SQL para crear o modificar una base de datos
• Para hacer ingeniería de reversa a partir de una base de datos existente:
Cree una copia de la base de datos y modifique el esquema
Cree una copia de la base de datos, con los datos y use la modelación para modificar el esquema y los datos
Cree una copia del esquema de la base datos, sin datos y use el modelado para modificar el esquema
Conéctese a la base de datos y use el modelado para modificar el esquema y los datos
• Para generar una aplicación de la red:
Cree un objeto semántico de las vistas
Genere páginas ASP para crear, leer, actualizar y eliminar acciones desde estas vistas
Genere páginas que se puedan colocar en Interdev Microsoft Visual para los clientes

➤ FIGURA B-2

Ayuda del Tabledesigner



➤ CREACIÓN DE UN MODELO DE OBJETO SEMÁNTICO

Para crear un nuevo modelo de objeto semántico (SOM), elija File New (Archivo Nuevo) del menú, o haga clic en el botón nuevo documento (el primer botón en la barra de herramientas). El Tabledesigner desplegará la ventana que se muestra en la figura B-3. Puede elegir una herramienta de inicio, que es una lista de modelos preconstruidos y dominios. En este ejemplo elegiremos Generic (Genéricos), pero tal vez quiera utilizar también los otros.

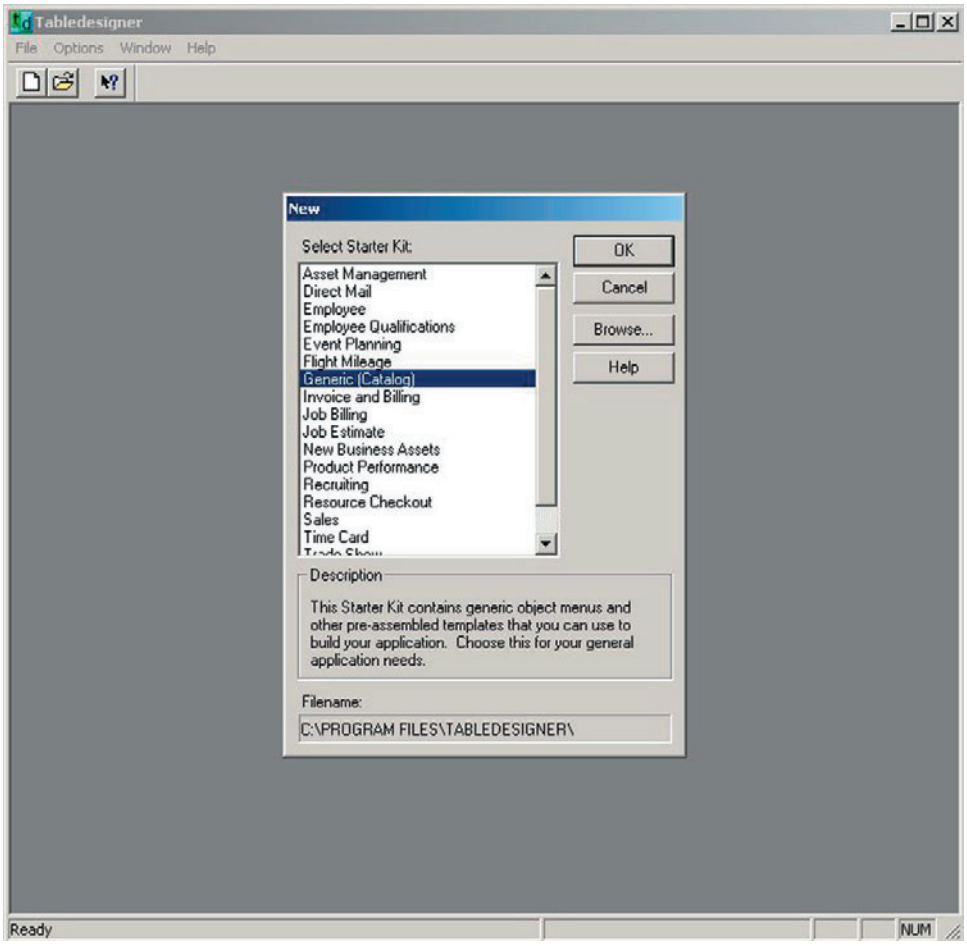
El siguiente despliegue mostrará una lista de dominios en el lado izquierdo con un espacio de diseño vacío de entidad a la derecha. Para crear un objeto semántico mantenga el botón derecho del ratón en el espacio de diseño y suéltelo. Aparecerá un rectángulo. Suelte el botón del ratón y verá el despliegue que se muestra en la figura B-4. Desde este punto, puede nombrar el objeto como usted quiera escribiendo en la caja de texto abierta. Nombre a este objeto STUDENT (ESTUDIANTE).

Para agregar atributos al objeto STUDENT, haga clic en el campo Name (Nombre) en la lista de la izquierda, arrástrelo y suéltelo en STUDENT. El Tabledesigner creará un atributo en STUDENT que hereda todas las propiedades del dominio Name. Haga clic en Name en STUDENT y presione Enter. Reescriba el nombre como StudentName en la caja de texto abierta en el objeto semántico.

Todos los dominios y atributos (observe que se refiere a los atributos como *items* en el Tabledesigner) tienen propiedades. Para verlas, haga clic a la derecha en Name en la lista de dominios. El Tabledesigner desplegará las propiedades del campo Name. Ahora, haga clic en el lado derecho en StudentName en el objeto STUDENT. El diseñador de tabla mostrará las Item Properties (propiedades del atributo) StudentName como se

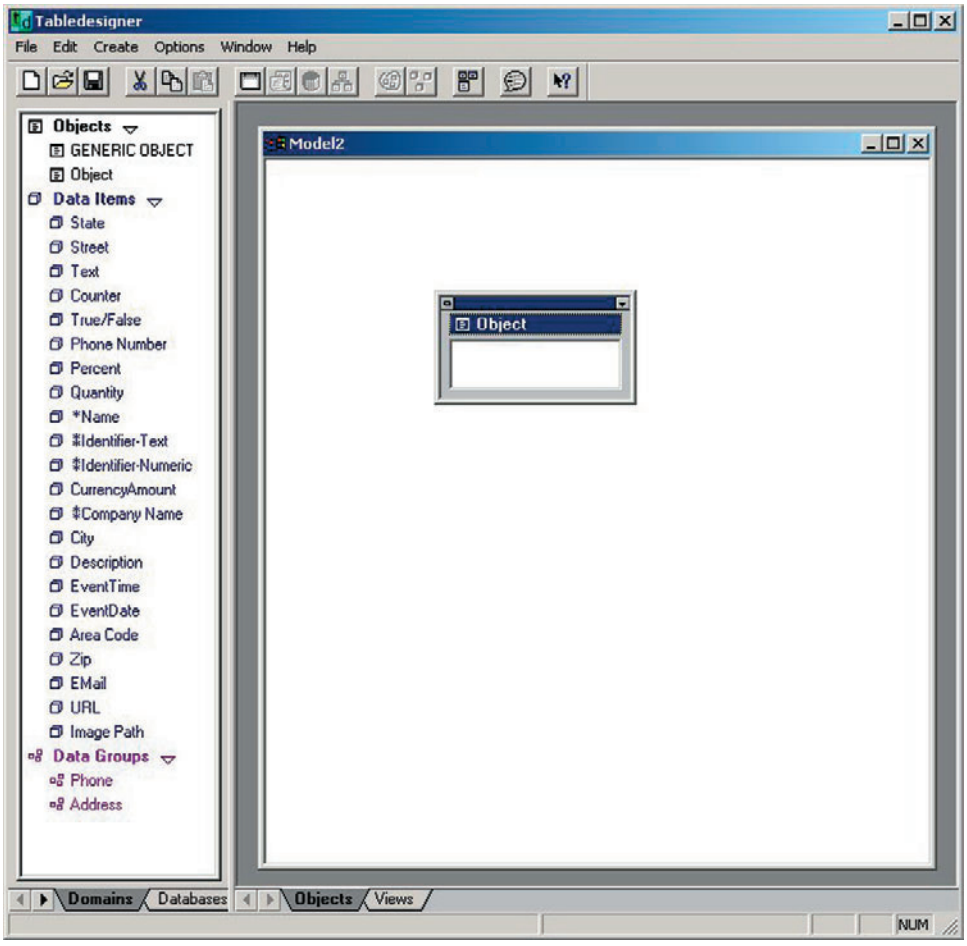
➤ FIGURA B-3

Elección de un kit de inicio



➤ FIGURA B-4

Creación y nombramiento de un objeto semántico



muestra en la figura B-5. Estas propiedades serán las mismas del dominio Name, excepto que el nombre del atributo es StudentName. La información de esta propiedad se muestra por medios de lectura que prevalecen en la propiedad en el dominio implícito. Cierre la ventana de propiedad. Presione simultáneamente las teclas <*Ctrl*> y *Z* para deshacer el cambio del nombre del atributo. (Por otra parte, puede deshacer hasta 30 etapas de trabajo con el Tabledesigner.)

Para continuar, arrastre Phone de la sección de Data Groups (grupo de datos) en la lista de campos y suéltelo en STUDENT, justo abajo de Name. Haga clic en la flecha cercana a Phone y se abrirá para mostrar su contenido, que comprende Area Code y Phone Number (Código de Área y Número Telefónico), como se muestra en la figura B-6. Arrástrelo y suéltelo en el grupo Address group (Dirección) en la misma forma.

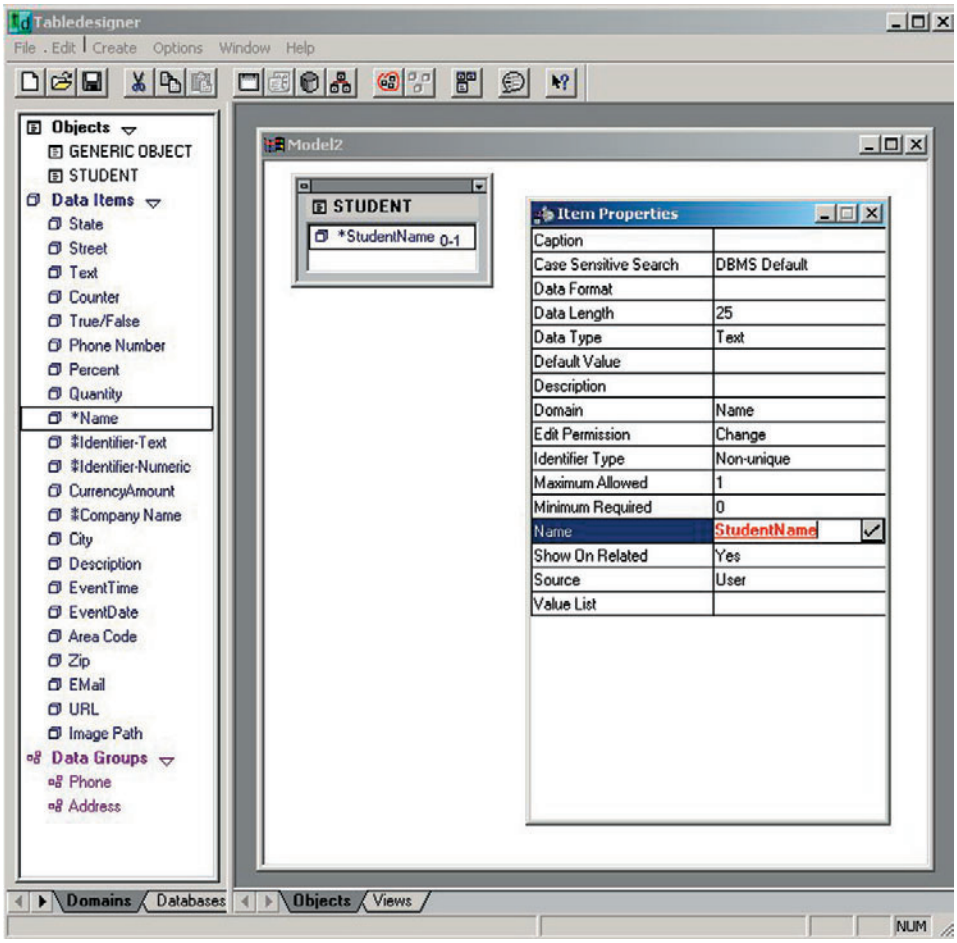
Para continuar con el modelo, cree un segundo objeto en el espacio de diseño arrastrando otro rectángulo y nombre a este segundo objeto CLASS. Arrastre Name y suéltelo en CLASS. También, arrastre Quantity de la lista de dominios y suéltelo en CLASS. Renombre Quantity como CreditHours (HorasCrédito). Si examina las propiedades de Cantidad, verá que son datos de tipo entero. Si su escuela permite fracciones de CreditHours (HorasCrédito), puede cambiar el tipo de datos de CreditHours haciendo clic en el lado derecho para mostrar la hoja de propiedades y cambiar el tipo de datos a Número Decimal de 7 dígitos.

Para crear una relación entre dos objetos semánticos haga clic en el icono pequeño justo a la derecha de STUDENT, en la parte de arriba del objeto semántico STUDENT. Arrastre este icono y suéltelo en CLASS, justo debajo de CreditHours. Se Creará una relación entre STUDENT y CLASS. Su modelo ahora deberá aparecer como se muestra en la figura B-7.

Las cardinalidades de la relación se pueden modificar haciendo clic en el subíndice 0-1 de los atributos de enlace de objetos CLASS o STUDENT. Cuando haga esto, apa-

➤ FIGURA B-5

Las propiedades de Student Name (Nombre del Estudiante)



recerá el despliegue que se muestra en la figura B-8. Abra la lista de la caja etiquetada como "Maximum allowed" (máximo permitido) para encontrar N. También, si lo desea puede introducir un número específico como 7. Hacerlo no afectará su esquema, pero ocasionará que las páginas ASP generadas cumplan ese límite.

Continúe con estas operaciones para crear un tercer objeto, ASESOR (ADVISOR), con los atributos que se muestran en la figura B-9. Para observar las consecuencias de la herencia de dominios, cambie el nombre de Phone en la lista de dominios. Para hacerlo, haga clic en Phone y después presione la tecla Enter. Escriba el campo Campus Phone como el nuevo nombre del dominio. Observe que tanto Phone en STUDENT, como Phone en ADVISOR han cambiado al Campus Phone. Estos atributos heredan cualquier cambio a las propiedades de sus dominios.

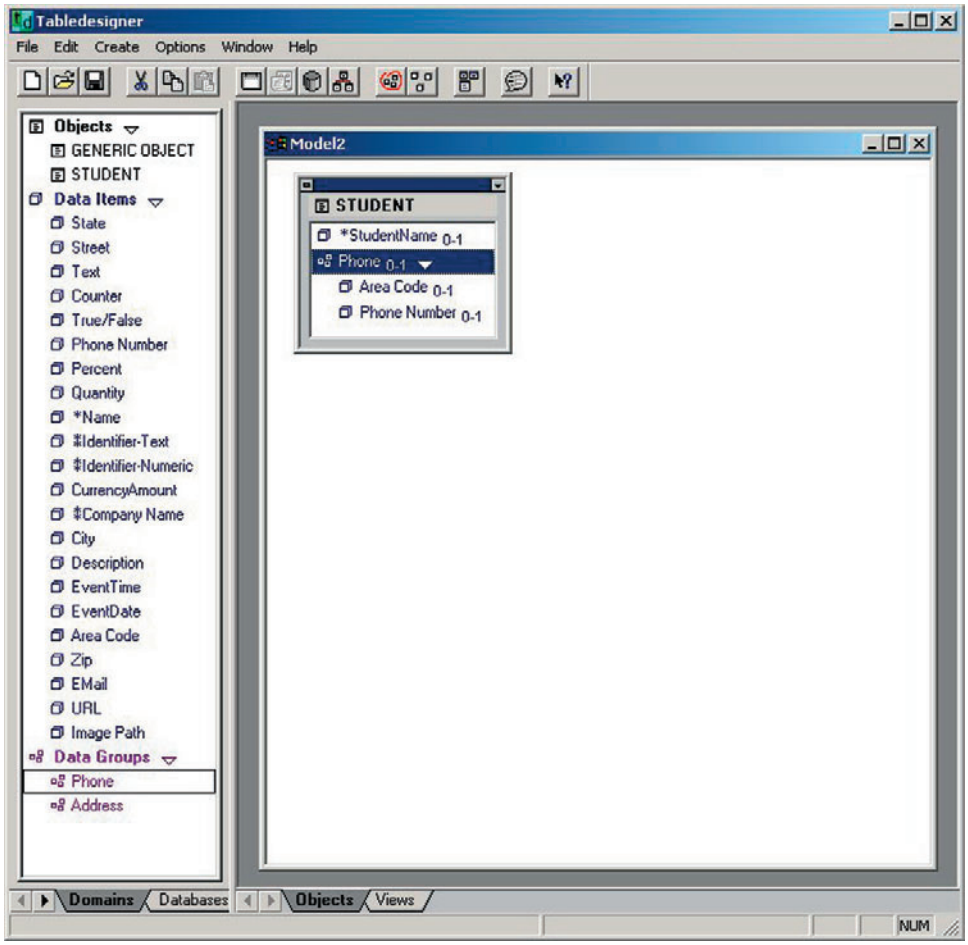
Ahora estamos listos para generar una base de datos. Puesto que el Tabledesigner puede modificar su esquema una vez que lo ha creado, no necesita terminarlo cuando genere su primera base de datos. Puede generarla e irla incrementando conforme avance, si usted quiere.

Haga clic en el disco pequeño de la barra de herramientas (el noveno icono en la barra de herramientas) o seleccione Create/Database (Crear/Basededatos) del menú. Guarde su modelo con el nombre que usted desee; aquí utilizamos Example1 (Ejemplo 1). Después que el modelo haya sido guardado, el Tabledesigner desplegará la ventana que se muestra en la figura B-10. Puede elegir el DBMS utilizado para la base de datos de esta lista.

Debido a que hemos instalado el SQL Server antes de este ejemplo, aparece en la lista. Para generar tablas en la base de datos del SQL Server haga clic en SQL Server y entonces registre en el SQL Server como se analizó en el capítulo 13. Use el Servidor (local), introduzca *sa* para el login *id* y cambie la base de datos predeterminada para que sea la única en la cual quiera colocar las tablas. Aquí utilizamos la base de datos ejemplo del SQL Server, la cual fue creada previamente. Hasta ahora, las tablas necesa-

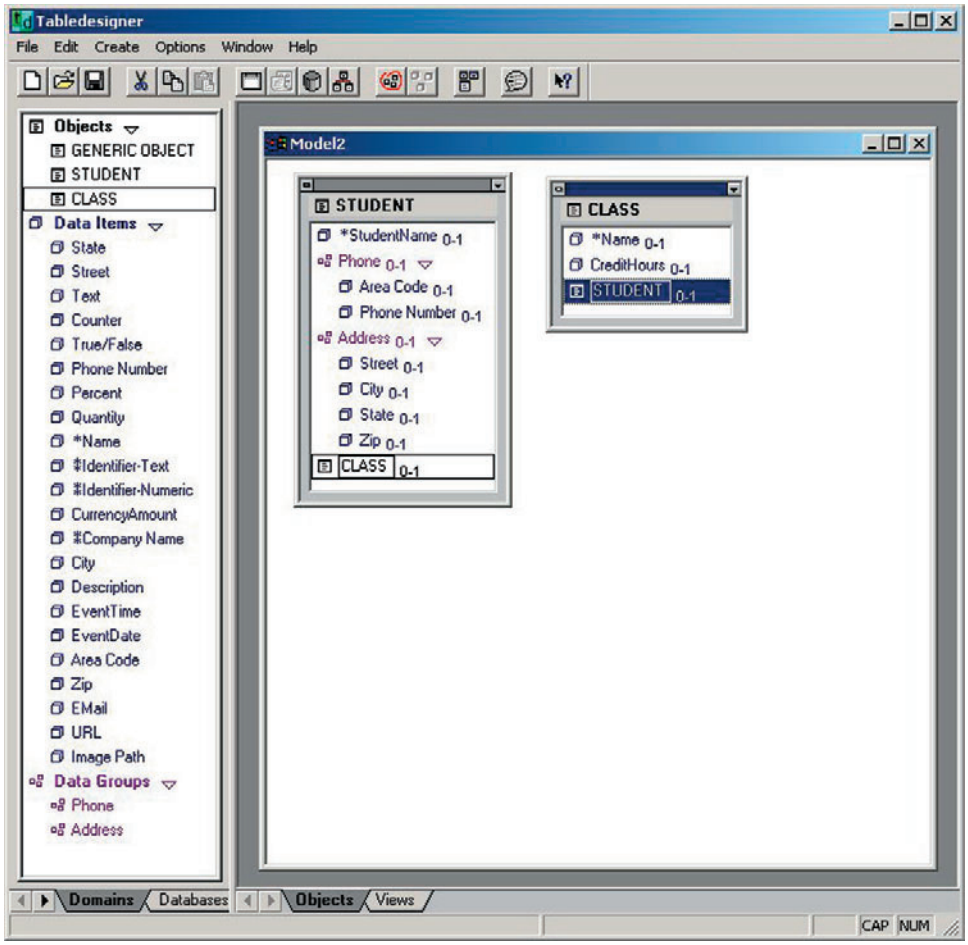
➤ FIGURA B-6

Ejemplo de un grupo de atributos



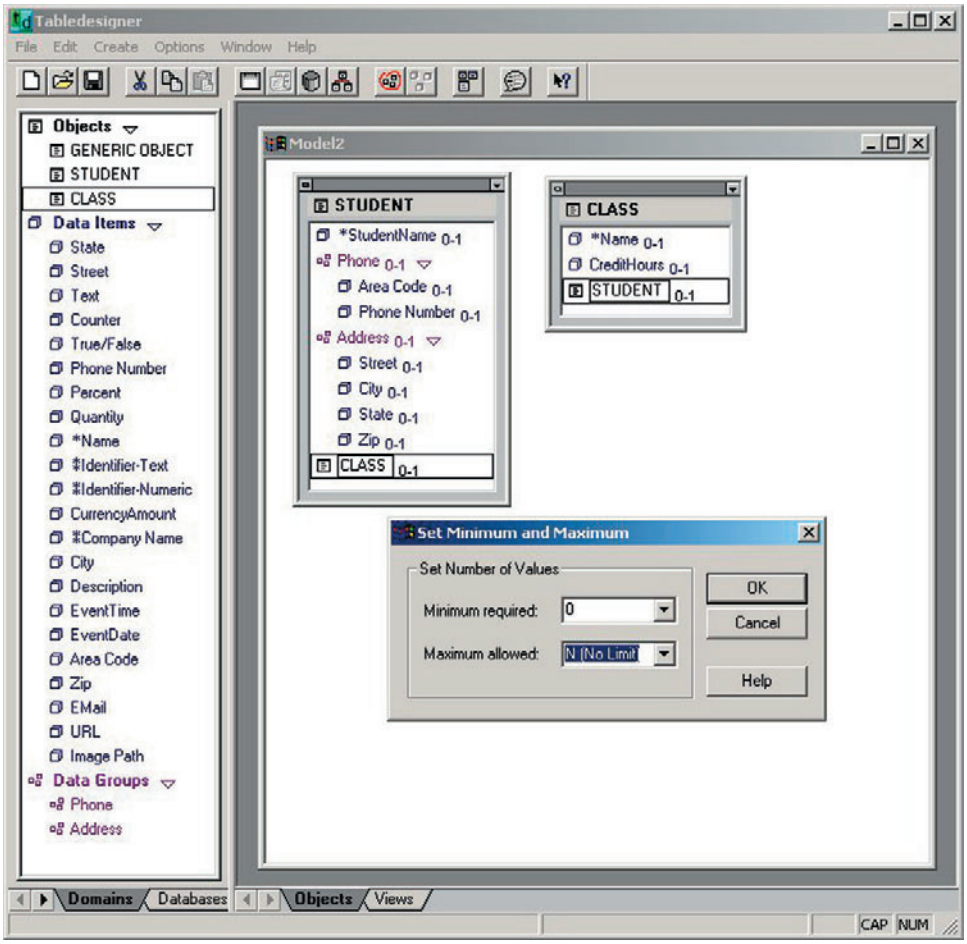
➤ FIGURA B-7

Creación de una relación entre STUDENT y CLASS (ESTUDIANTE y CLASE)



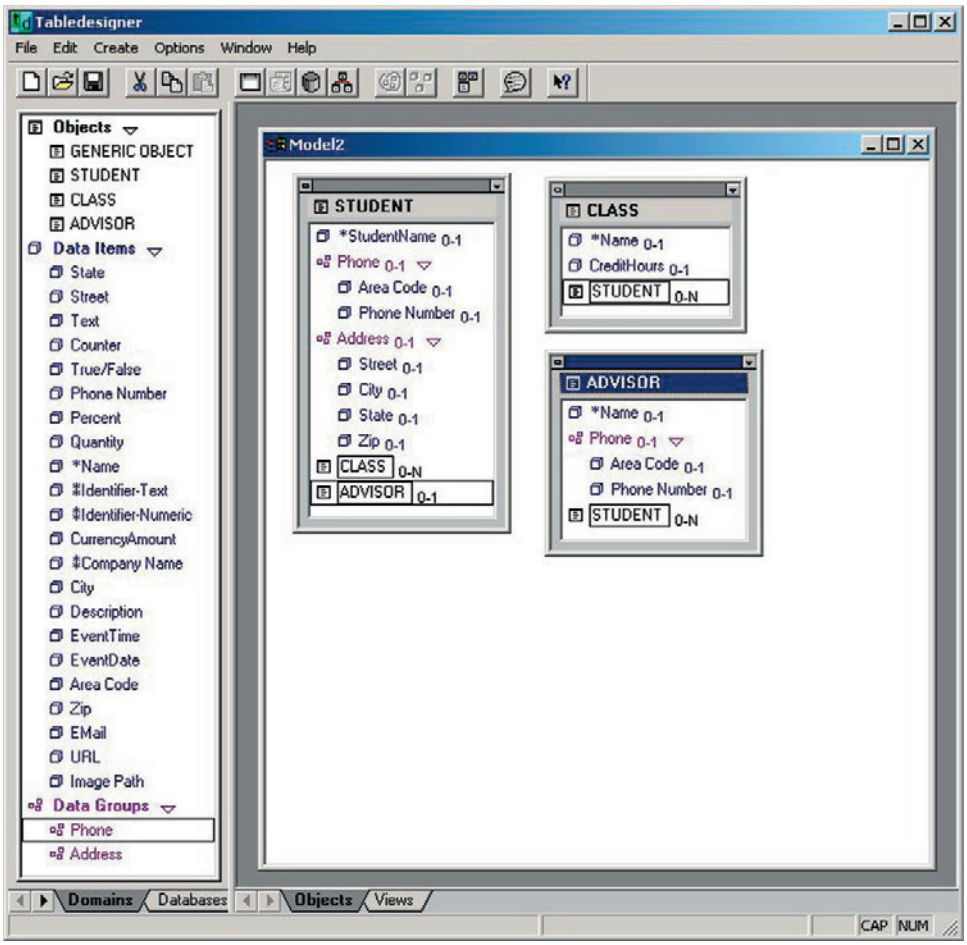
➤ FIGURA B-8

Cambio de la cardinalidad de la relación



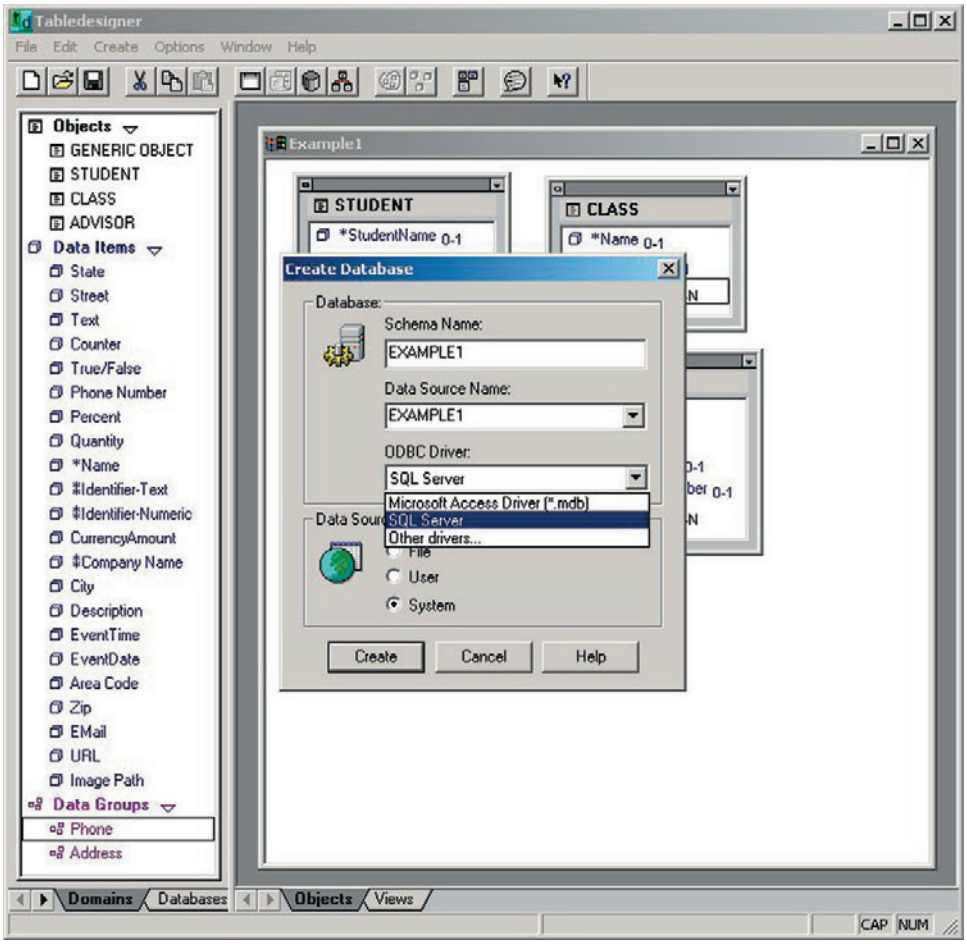
➤ FIGURA B-9

Árbol de modelo de objetos



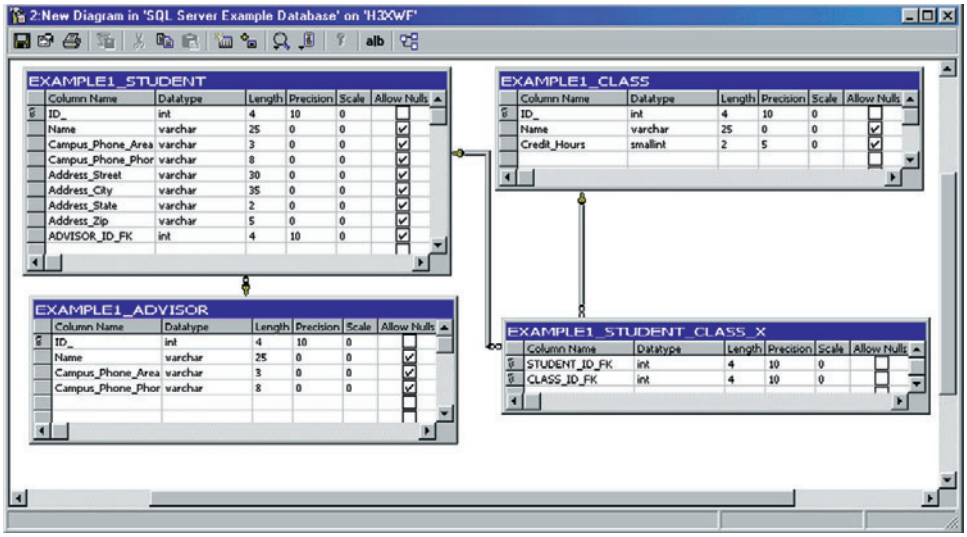
➤ FIGURA B-10

Selección del DBMS



➤ FIGURA B-11

Tablas del SQL Server para el modelo en la figura B-9



rias para el modelo fueron creadas por el Tabledesigner y colocadas en la base de datos de ejemplo del SQL Server. La figura B-11 muestra las tablas que fueron generadas para el modelo en la figura B-9.

Si no ha instalado el SQL Server, pero ya ha instalado Access, puede seleccionarlo de la lista en la figura B-10 y el Tabledesigner generará una base de datos Access.

También puede causar que el Tabledesigner despliegue las instrucciones SQL que se ejecutan antes de generar la base de datos. Para hacer esto, seleccione Create/SQL Text Only y las instrucciones SQL se colocarán en un archivo de texto en un directorio llamado SQL en el directorio del Tabledesigner. Debe hacer esto antes de generar su base de datos, de otra forma no hay SQL en fila de espera para la ejecución.

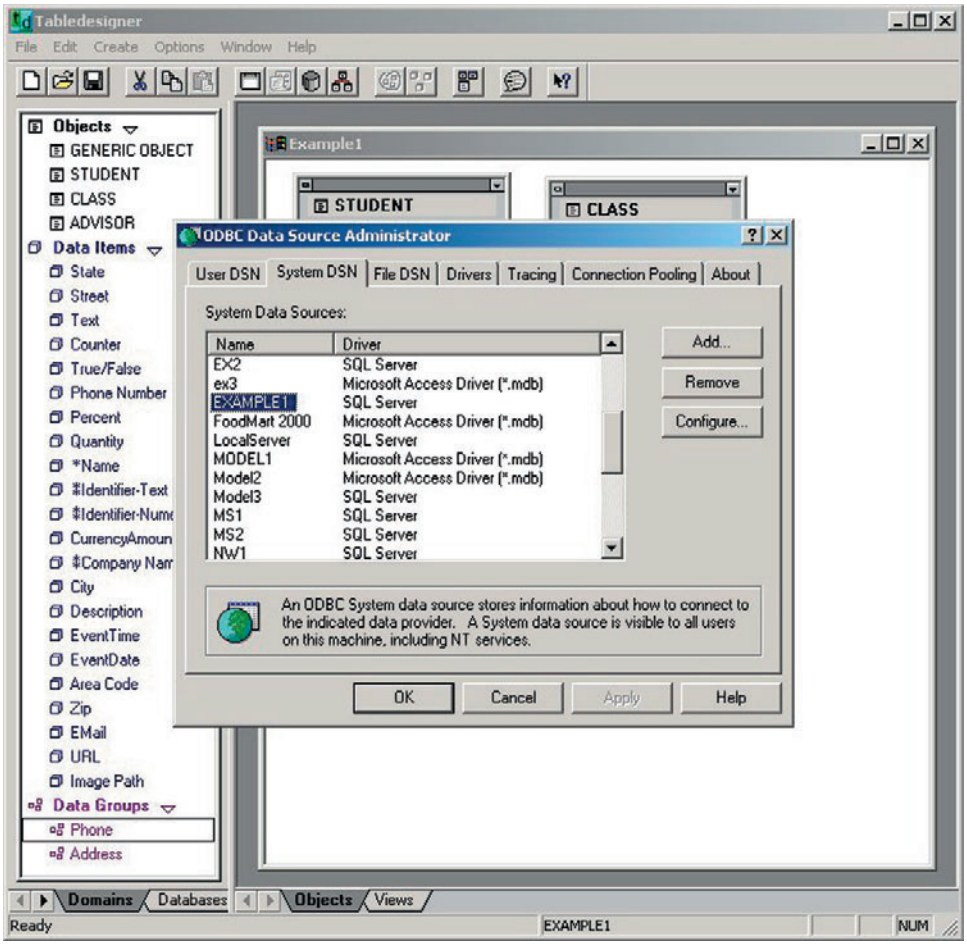
➤ UN MODELO SOM DE INGENIERÍA DE REVERSA

Además de la creación de nuevos modelos, puede utilizar el Tabledesigner para generar un modelo en toda la base de datos o sólo en parte de ésta. Para hacerlo, primero necesita crear un ODBC Data Source Name (Nombre de Fuente de Datos ODBC) en la base de datos en la que desee revertir la maquinaria. En este ejemplo utilizaremos Northwind.mdb, una base de datos que trabaja con Microsoft Access.

Para colocar una Fuente de Datos en Northwind, primero abra el ODBC Administrator (Administrador ODBC) seleccionando File/ODBC Administrator del Tabledesigner. Seleccione el System tab (Sistema Tabulador) como se muestra en la figura B-12 y haga clic en Add. Seleccione Microsoft Access de la lista de drivers y después haga clic en Finish. Introduzca un nombre para la fuente de datos (utilizamos Northwind en es-

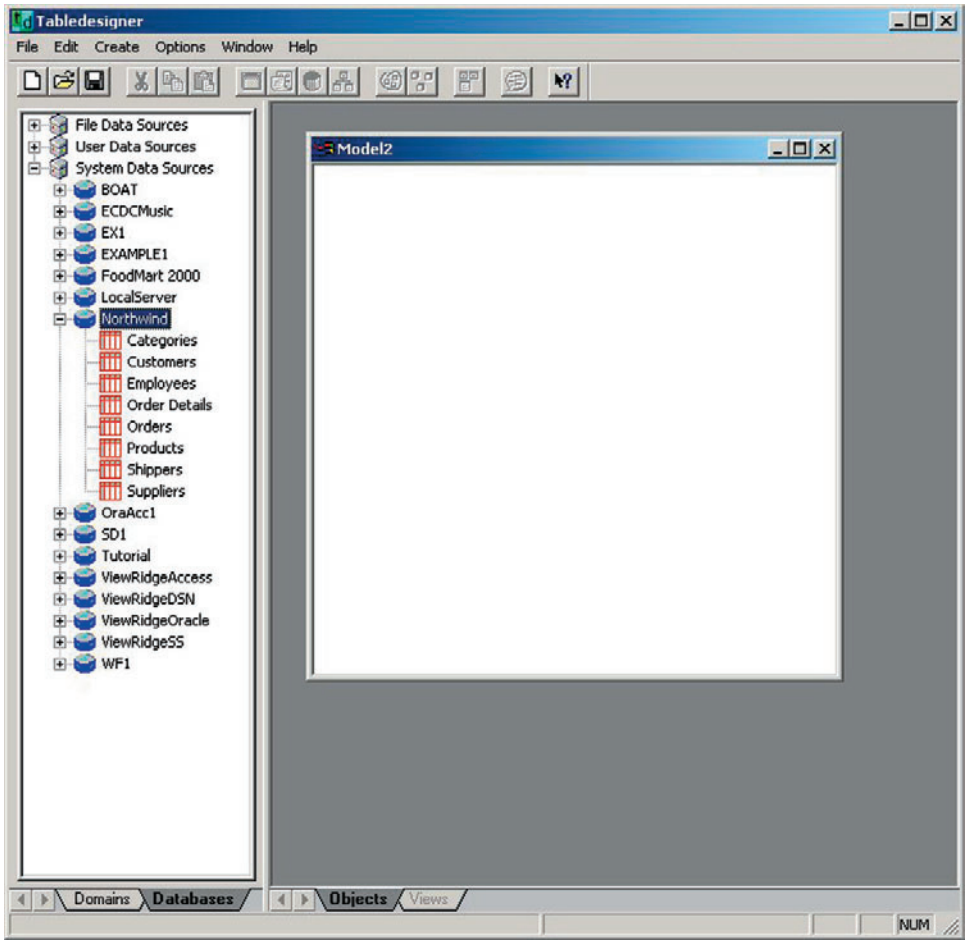
➤ FIGURA B-12

Uso del Administrador ODBC para crear una fuente de datos ODBC



➤ FIGURA B-13

Ingeniería de reversa de Northwind



te ejemplo) y haga clic en el botón Select. Explore el directorio donde se localiza Northwind.mdb. Esta ubicación varía dependiendo de cómo haya instalado Access. Si no sabe dónde está, use Windows Find para localizar el archivo llamado Northwind.mdb. Vaya a ese directorio en el ODBC administrator (administrador ODBC) y haga clic en O.K.

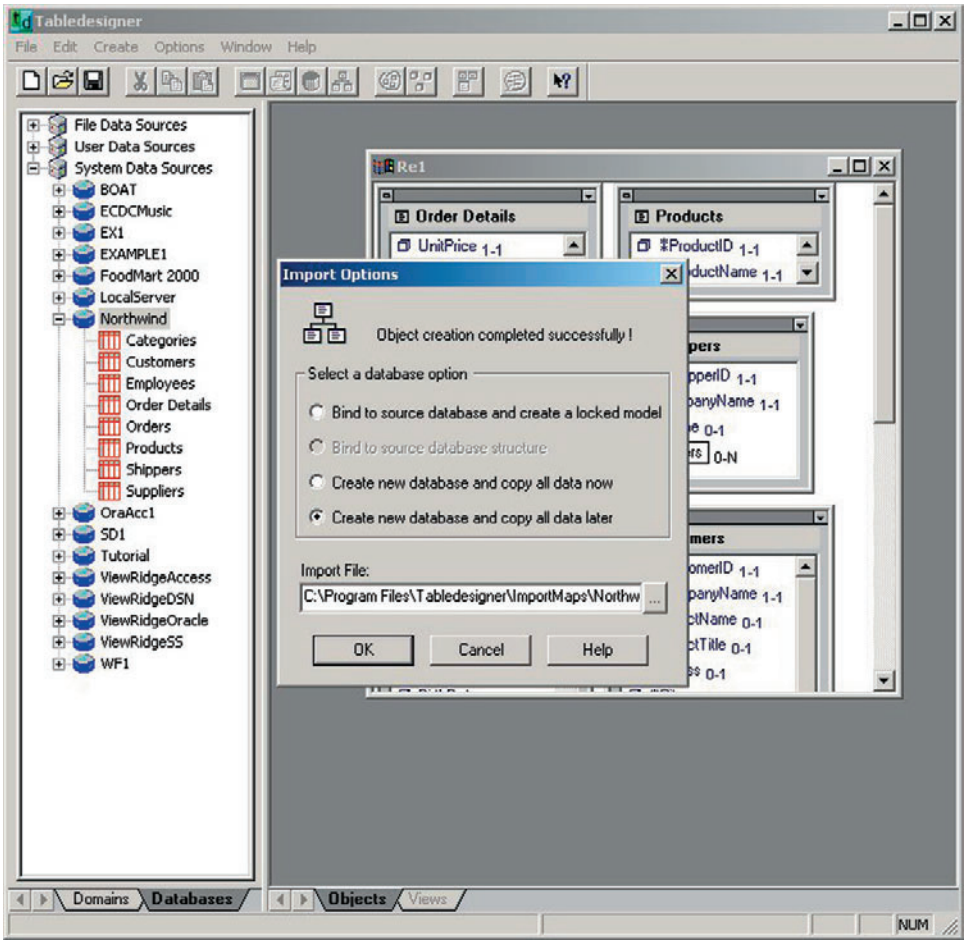
Abra un nuevo modelo dentro del Tabledesigner. Mire el botón de la lista de dominios y haga clic en el tabulador etiquetado Databases. El Tabledesigner desplegará una lista de su File, User y System data sources (Archivo, Usuario y Sistema de Fuente de Datos). Puesto que creamos un System data source (Sistema de Fuente de datos), haga clic ahí y busque el nombre de su fuente de datos. Haga clic en el icono de la izquierda y le mostrará todas las tablas de la base de datos (como en la ventana de la izquierda de la figura B-13). Arrastre el nombre de su fuente de datos (aquí Northwind) en el espacio de diseño de entidad en la ventana de la derecha. Guarde el modelo; utilizamos el nombre de RE1 para este ejemplo.

Ahora aparecerá la caja de diálogo que se muestra en la figura B-14. Haga clic en Help (Ayuda) para encontrar varias opciones. Por ahora, seleccione la última opción, la cual creará una nueva base de datos, pero no copiará ninguno de los datos; colocará los archivos importados del Tabledesigner para un copiado más tarde. Normalmente usaría esa opción si quisiera aplicar filtros para copiar sólo algunos datos, pero aquí omitiremos este ejercicio.

Ahora, el Tabledesigner genera una base de datos que conforma el modelo. Seleccione un DBMS de la lista y después haga clic en Create. El resultado de esta acción será un modelo como el que se muestra junto de la figura B-15 con una base de datos que tiene una copia de las tablas de Northwind. Ahora puede hacer cambios a este modelo y use el Tabledesigner para cambiar la estructura de la base de datos.

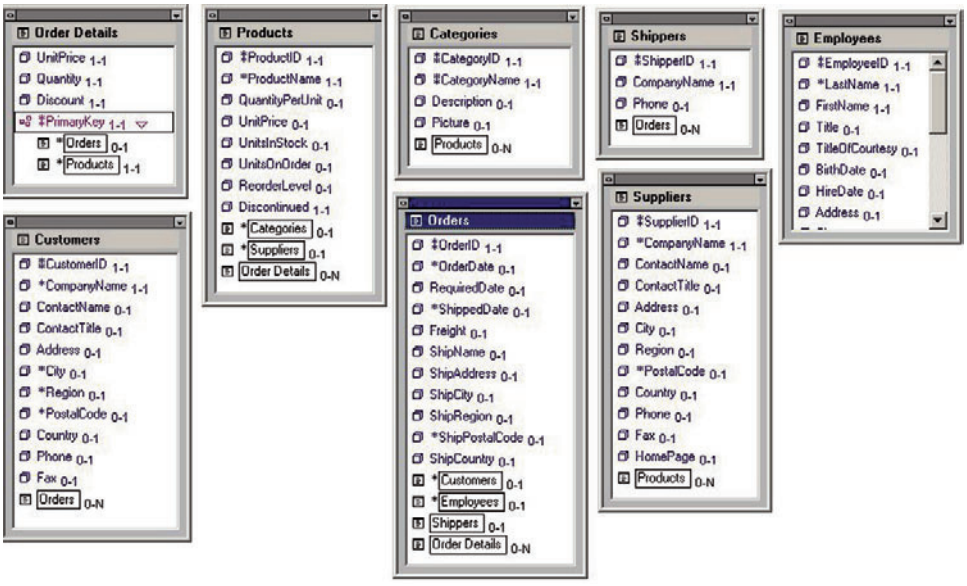
➤ FIGURA B-14

Creación de un esquema de una base de datos sin datos



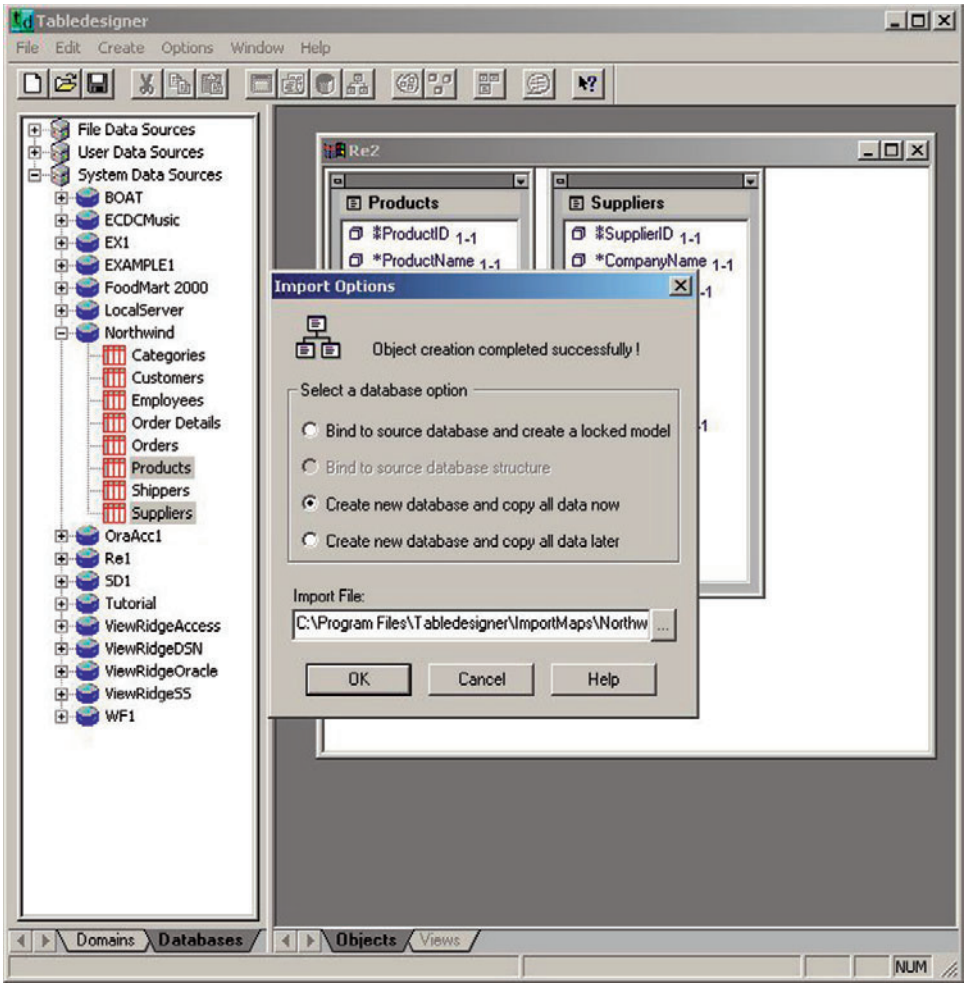
➤ FIGURA B-15

Modelo de objeto semántico de Northwind utilizando la reversión



➤ FIGURA B-16

Construcción de un extracto de dos tablas de Northwind



Utilizaremos un ejemplo diferente para ilustrar el copiado de datos y el cambio de esquemas. Cierre su modelo y abra uno nuevo. Haga clic en los tabuladores de las bases de datos y seleccione System Data Sources (Sistema de Fuente de Datos). Haga clic en Northwind para abrir la lista de las tablas. Mantenga oprimida la tecla <Ctrl> y haga clic en Products y Suppliers (Productos y Proveedores). Arrastre esas dos tablas al espacio de diseño de entidad y suéltelas ahí. Guarde su modelo con un nombre (nosotros utilizamos RE2) y seleccione la tercera opción, Create new database and copy all data now (*Crear Nueva base de datos y copiar todos los datos ahora),* como se muestra en la figura B-16. Haga clic en OK y después seleccione su DBMS. De nuevo utilizaremos el SQL Server en este ejemplo. También puede utilizar Access.

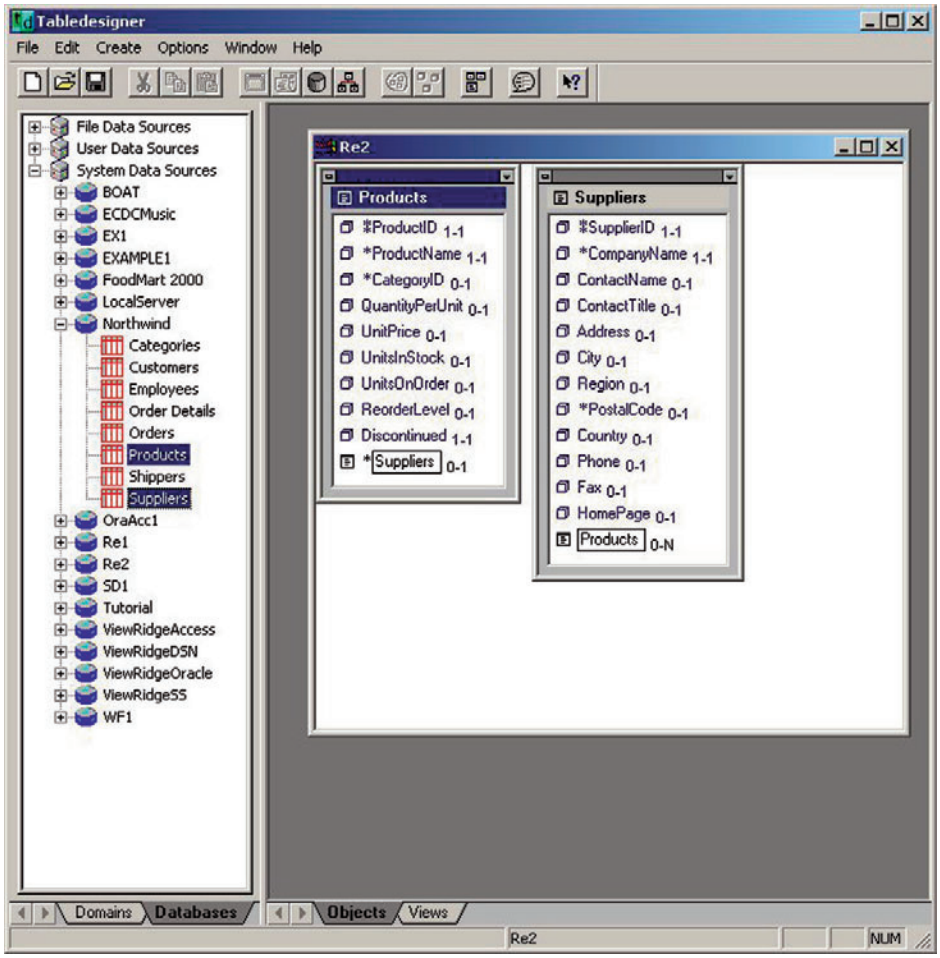
Si selecciona el SQL Server necesitará registrar en tres tiempos: el primero es mediante un manejador ODBC para colocar la conexión, el segundo tiempo es justo antes de la creación de las tablas y el tercero es cuando se agregan datos a las tablas. Use la notación *id sa* como se analizó en el capítulo 13. Si usó Access, el proceso será mucho más simple.

Hasta ahora tiene una nueva base de datos con datos en dos tablas: Products y Suppliers. Las dos tablas tienen una relación 1:N, como se muestra en el modelo en la figura B-17. Suponga que queremos hacer dos cambios. Primero, queremos tener la posibilidad de múltiples Suppliers de un producto, y segundo, queremos poder tener muchos contacts (contactos) por un Supplier determinado. Ahora haremos esos dos cambios al modelo y observe sus consecuencias en la base de datos implícita.

Primero, cambie la cardinalidad máxima de la relación (link) Suppliers en el objeto Products de 1 a N. Esto se hace haciendo clic en el subíndice 0-1 en Suppliers en Pro-

➤ FIGURA B-17

Productos y objeto semántico de proveedores utilizando la ingeniería de reversa



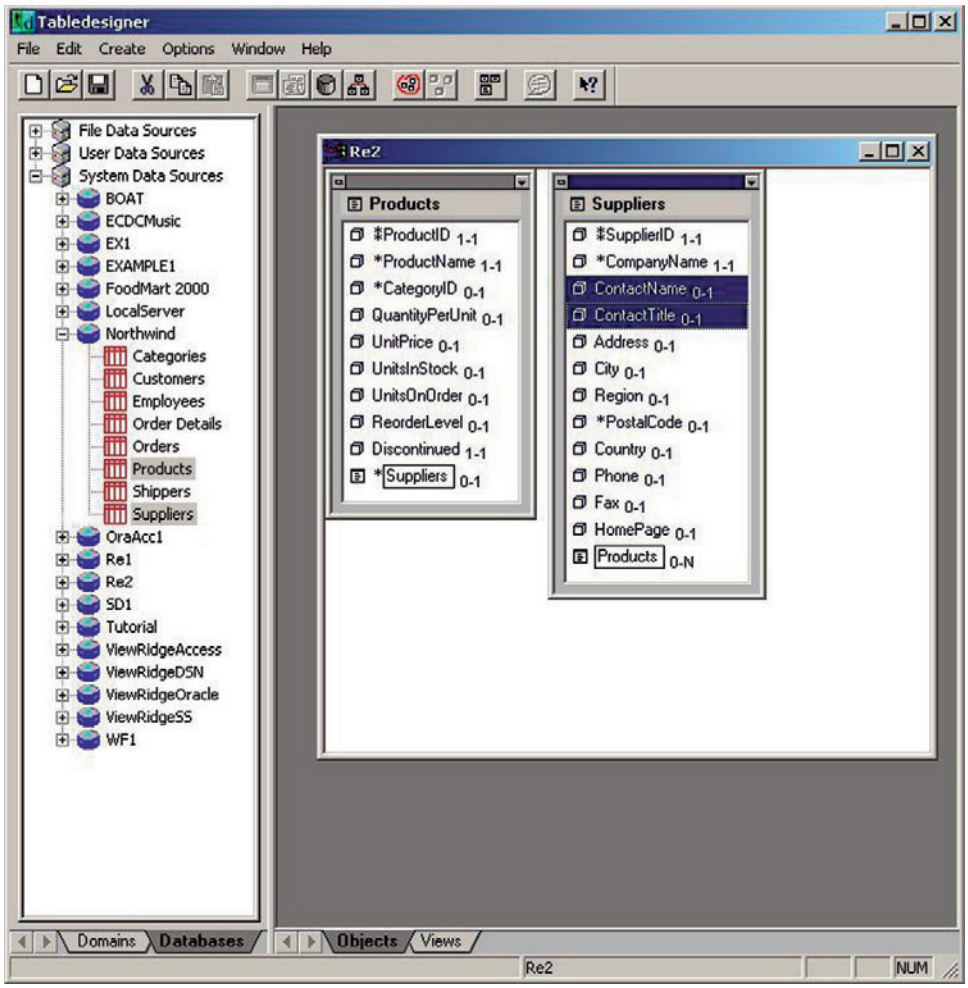
ducts y seleccionando N de la lista de cardinalidad máxima en la caja de diálogo que aparece. Es el mismo proceso que describimos antes.

Ahora, para permitir múltiples contactos para un Supplier, primero crearemos un grupo para datos de Contact y después se coloca la cardinalidad máxima para ese grupo a N. Para crear el grupo, mantenga apretada la tecla <Shift>, haga clic en ContactName y después en ContactTitle. Ambos deberán estar resaltados como se muestra en la figura B-18a. Haga clic en el botón group, el cual parece una pizza en la barra de herramientas. El apuntador del ratón está sobre ese botón en la figura B-18a. Introduzca el name Contact en la caja de diálogo que aparece y haga clic en OK. Ahora tendrá un grupo que contiene los atributos ContactName y ContactTitle. Con el fin de permitir múltiples contactos, establezca la cardinalidad máxima del grupo contacto de tal forma que sea mayor que uno. Nuevamente haga clic en el subgrupo 0-1 del grupo Contact y establezca la cardinalidad máxima. En la figura B-18b la cardinalidad máxima se estableció en N. Haga clic en OK.

Antes del procedimiento, considere que el Tabledesigner necesita hacer esto. Debido a que la relación entre Products y Suppliers se cambió de 1:N a N:M, se debe de crear una nueva intersección. Sin embargo, la base de datos tiene datos, así que el Tabledesigner necesitará mover los datos dentro de la tabla de intersección para preservar la relación. Segundo, se necesitará crear una nueva tabla para los datos de Contact y todos los datos que se movieron a esa tabla nueva. Todo esto se hará cuando haga clic en el icono de la base de datos, o en seleccionar.

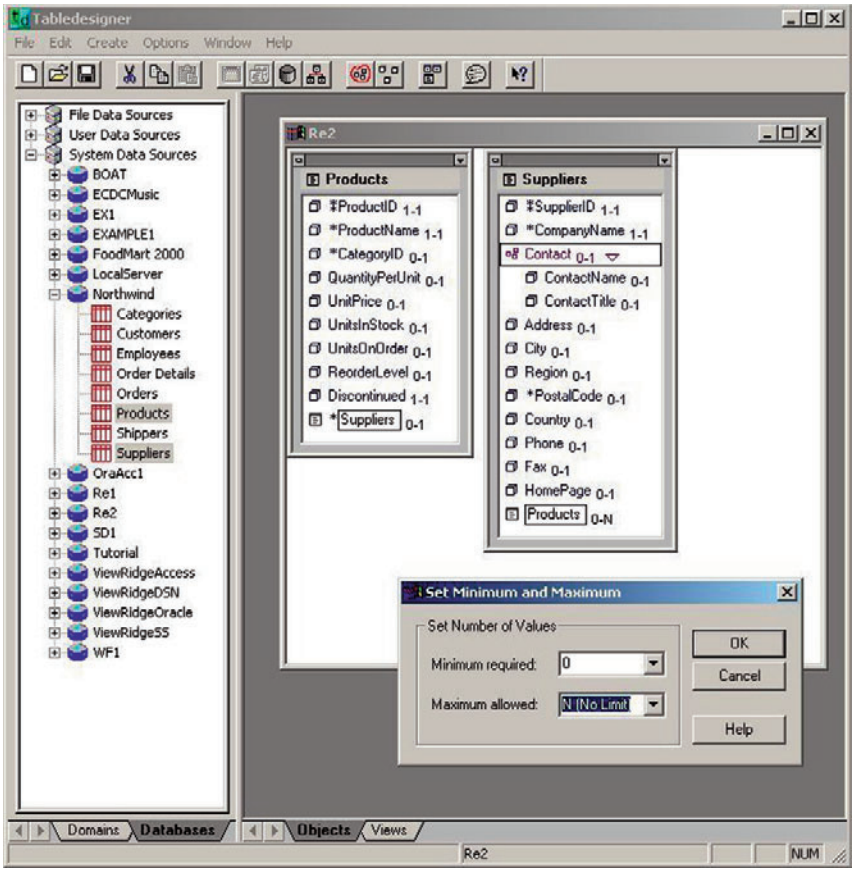
➤ FIGURA B-18A

Creación de un grupo Contact



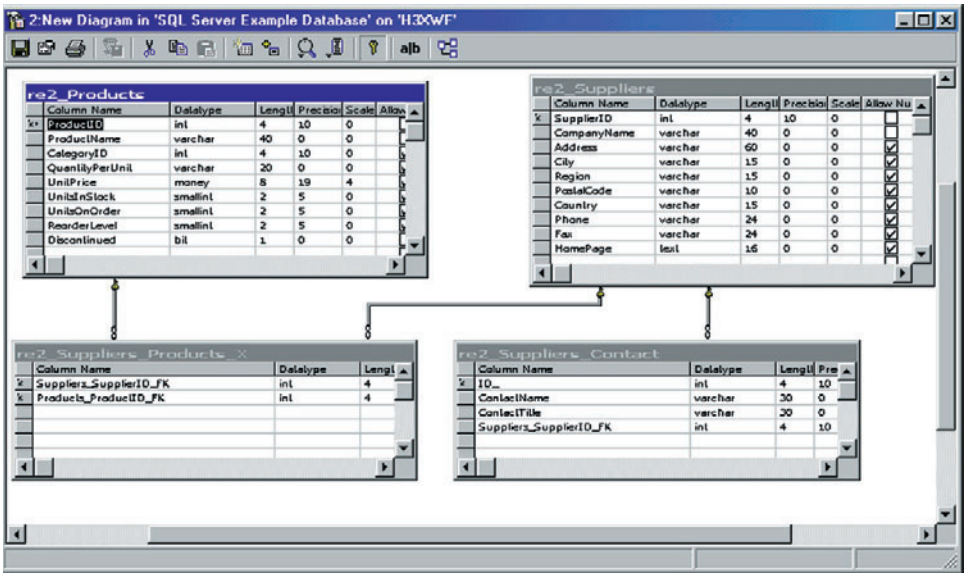
➤ FIGURA B-18B

Establecimiento de la cardinalidad máxima del grupo Contact



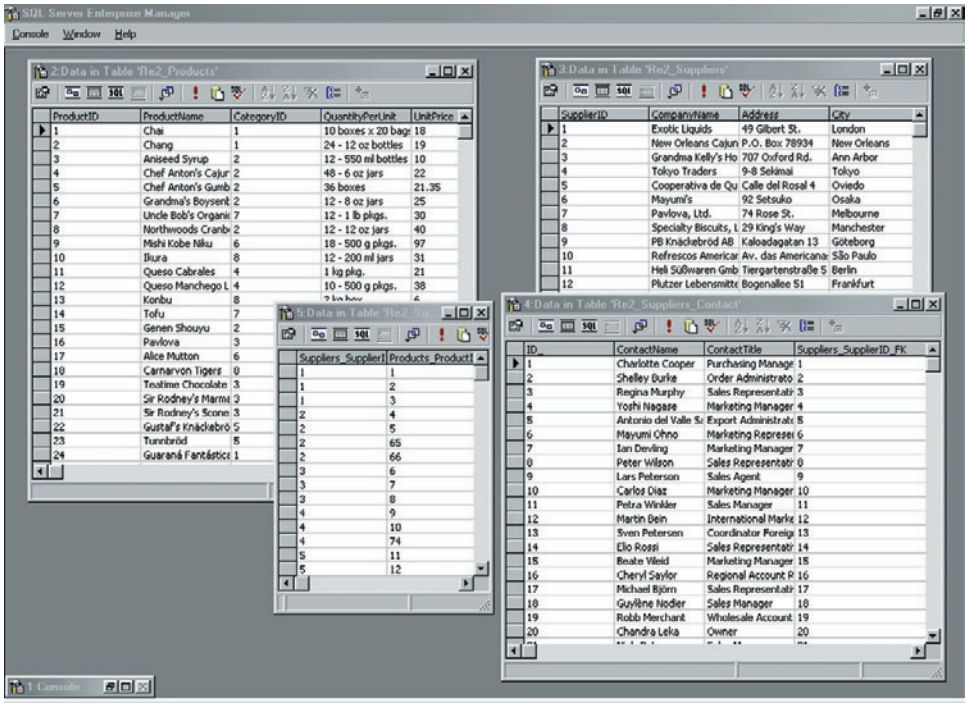
➤ FIGURA B-19

Estructura de la base de datos del SQL Server después de los cambios de modelo



➤ FIGURA B-20

Base de datos simple después de los cambios de modelos



La figura B-19 muestra el esquema del SQL Server que fue creado por el Tabledesigner después de estos cambios. Observe que la tabla de intersección se ha creado correctamente y que se generó la nueva tabla Contact. La figura B-20 muestra una parte de los datos. De nuevo, observe que los datos se han movido correctamente a las nuevas tablas.

Este ejemplo muestra el beneficio de trabajar por modelo en lugar de por base de datos. Si estos cambios se realizaran manualmente, se requerirían cuando menos varias horas de trabajo.

➤ EDICIÓN DE LAS VISTAS DE BASES DE DATOS EN LA RED

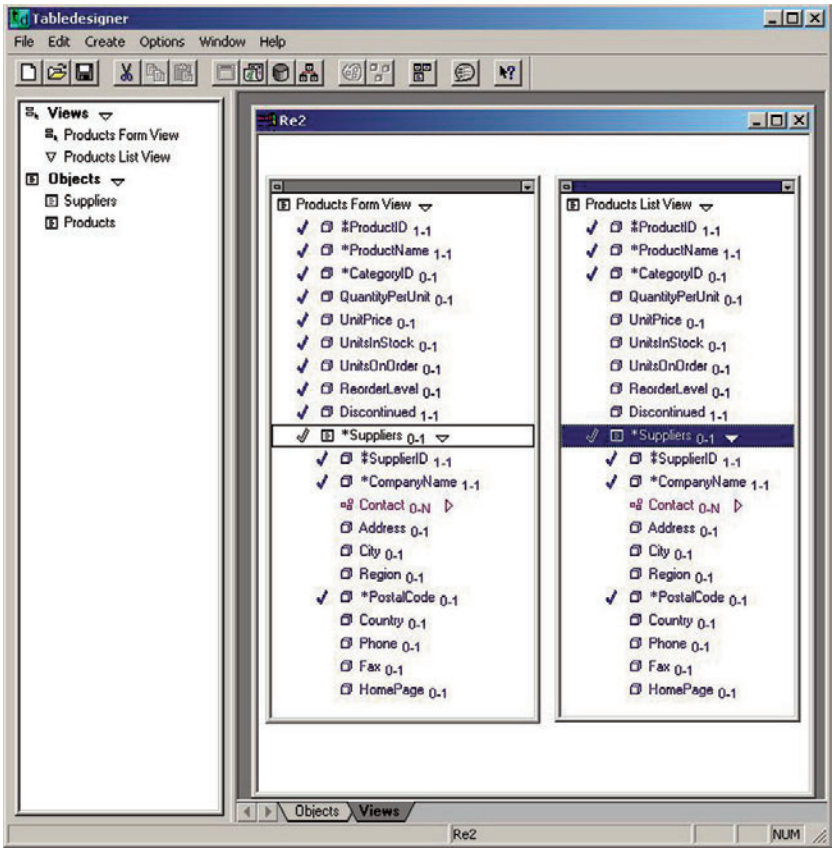
Además de la creación y modificación del esquema, el Tabledesigner puede generar unas series de páginas ASP que capacitarán al usuario para crear, leer, actualizar y eliminar datos en las vistas de bases de datos en la red. (Vea el capítulo 11 si estos términos no le son familiares.) Estas páginas se pueden procesar mediante IIS ejecutable en Windows 2000 y también con el Servidor de la red personal que se ejecute en Windows 98 o ME. En el último caso, necesitarán instalarse los componentes necesarios para el procesamiento de la página ASP. Para mayor información vea la documentación para el Servidor de la red personal.

El proceso de generar estas páginas es simple. Primero, cree las vistas que quiera publicar y entonces ejecute el asistente del Tabledesigner de Publicación de la Red para crear las páginas. Después, necesita colocar esas páginas en un directorio, en el cual IIS o el Servidor Personal de la red pueda encontrarlas. Puesto que las páginas usan Jscript, el directorio necesitará ser marcado para la ejecución de la escritura. Puede hacer esto modificando las propiedades del directorio de aplicación. Consulte la documentación de Microsoft o la del Tabledesigner para mayor información, si es necesario.

Ilustraremos el proceso de generación de páginas utilizando el modelo RE2 creado en la sección anterior. Por supuesto, puede usar cualquier modelo que quiera para este proceso. Para generar una vista de objeto semántico, abra el modelo con el que desee

➤ FIGURA B-21

Forma inicial y listas de vistas para Products



trabajar (aquí RE2.apm) y haga clic en el tabulador View (Vista) en el botón de su ventana de diseño. La ventana del lado izquierdo ahora se cambia para desplegar una lista de vistas y objetos.

En la sección de objetos, arrastre Products y suéltelo en el espacio de diseño vacío. Se abrirán dos ventanas en el espacio de diseño como se muestra en la figura B-21. La lista de Vistas de productos se utiliza para mostrar una serie de instancias de objetos. A menudo se utiliza para desplegar resultados de una consulta. La segunda vista, la de Products Form, se utiliza para introducir y cambiar valores de datos. Se muestra sólo una instancia de objeto en una vista de Form.

Considere la primera Lista de Vista (List View). Una marca de verificación que está junto a artículo indica que el artículo aparecerá en la forma Web. En este caso, ProductID, ProductName y CategoryID) (ProductoID, NombreProducto y CategoríaID) han sido verificados de manera predeterminada. Suponga que no queremos ProductID o CategoryID en nuestra lista de vistas. Para eliminarlas de la vista, haga doble clic en sus marcas de verificación; las marcas de verificación desaparecerán indicando que han sido eliminadas de la vista. Ahora, haga clic en la flecha que está junto a Suppliers para abrirlo. Este grupo representa la relación del objeto Suppliers en la vista de objeto semántico. Modifique las marcas de revisión, así CompanyName y Contact aparecerán en la lista de vistas. Expanda el grupo Contact y su pantalla deberá lucir como en la figura B-22.

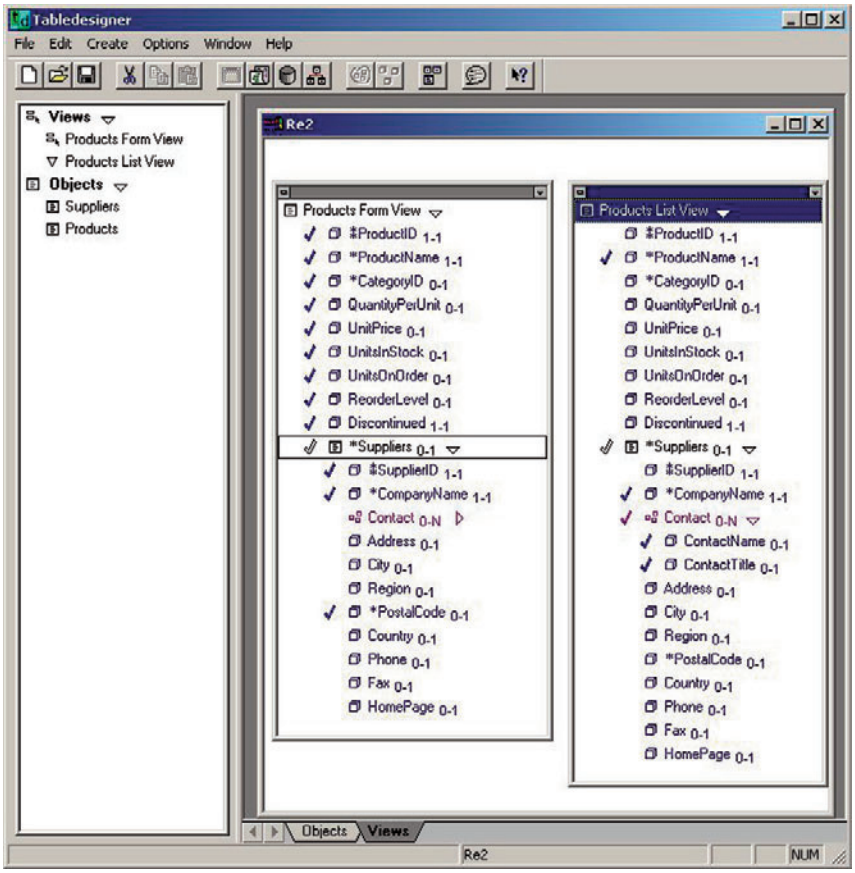
Ahora, considere la vista de Products Form. Suprima las marcas de verificación, así sólo ProductID, ProductName, UnitPrice, UnitsInStock y Company Name (en Suppliers) estarán en la vista. Su pantalla deberá verse como la de la figura B-23.

Siga un proceso similar para la vista de Suppliers. Agregue o elimine cualesquiera de los atributos que considere apropiados. En la figura B-24 se muestra un posible conjunto de vistas de Suppliers. Ahora estamos listos para publicar la aplicación en la Web.

Para comenzar el wizard de publicación Web, seleccione File/Publish Web Application (Archivo/Aplicación Publicación de la Web). Guarde el modelo cuando se le solicite y haga clic en Next. Las vistas para publicar se muestran en la siguiente pantalla.

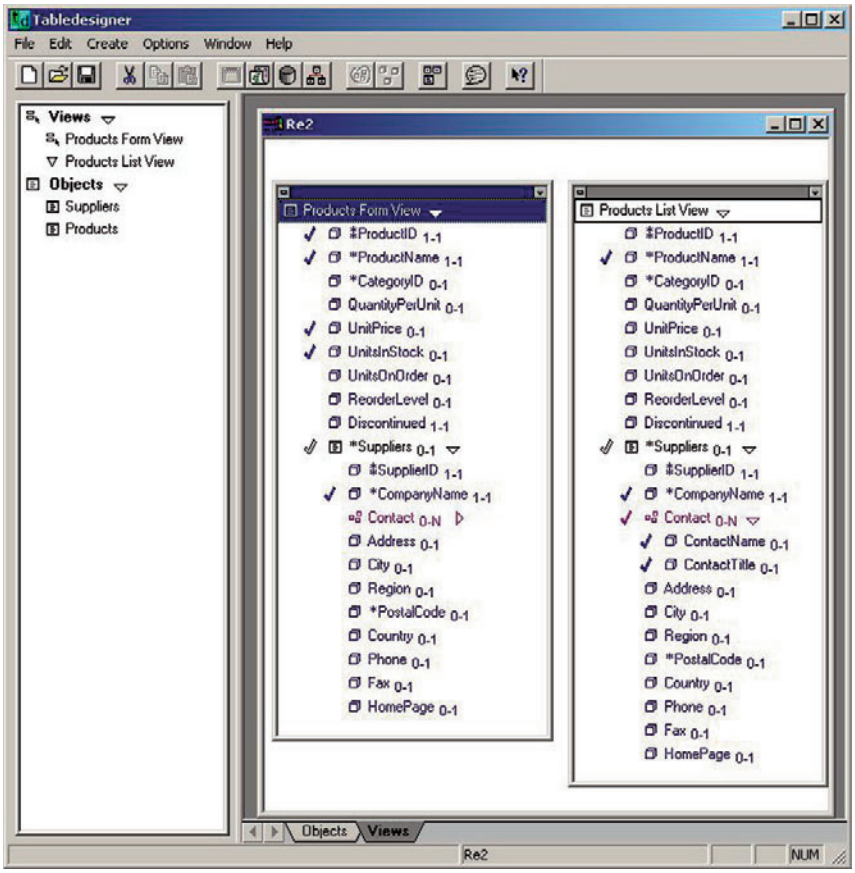
➤ FIGURA B-22

Modificaciones en la vista Products List



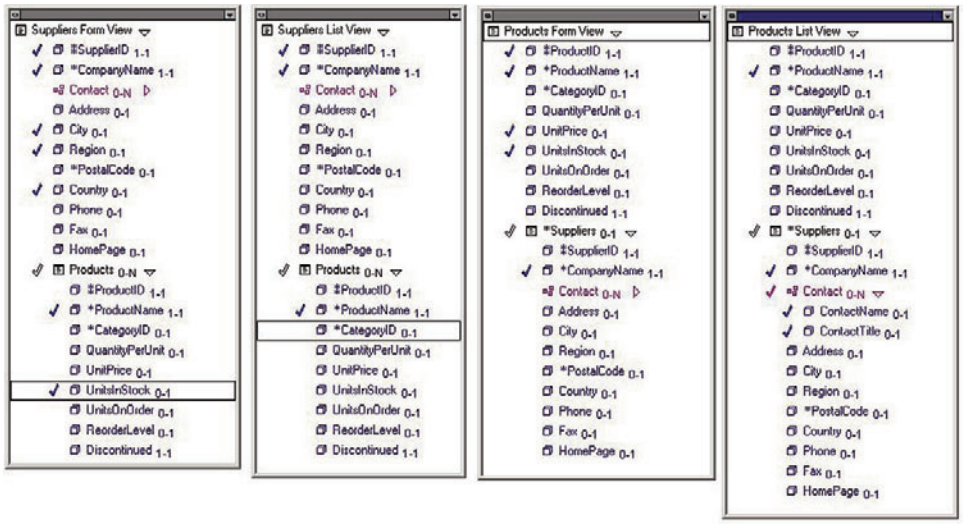
➤ FIGURA B-23

Modificaciones en la vista Products Form



➤ FIGURA B-24

Vistas de Products and Suppliers (Productos y Proveedores)



Éstas son las que queremos publicar, así que haga clic otra vez en Next. En la siguiente pantalla, que se muestra en la figura B-25, el Tabledesigner está preguntando qué conjunto de plantillas quiere utilizar. Seleccione las acciones predeterminadas haciendo clic en Next.

El siguiente wizard (asistente) necesita saber en cuál directorio almacena las páginas ASP. Debería explorar en la dirección que usted estableció con su Web publisher. Alternativamente, puede usar el directorio que seleccionó el Tabledesigner y crear privilegios de publicación en la red para ese directorio. De cualquier manera, introduzca un directorio válido aquí y haga clic otra vez en Next. En la siguiente pantalla introduzca Sample Application, o algún otro título, y haga clic otra vez en Next.

En el siguiente panel se acepta la selección predeterminada para no registrar la aplicación ahora. En la siguiente pantalla, que se muestra en la figura B-26, puede introducir una cuenta ID predeterminada y una contraseña. Esto lo haría si estuviera creando una aplicación a la cual usted deseara que los extraños pudieran tener acceso; pero como no estamos haciendo eso, no introduzca nada y haga clic en Finish. En este punto, el Tabledesigner generará las páginas ASP. Cuando haya terminado, preguntará si quiere abrir el asistente de Microsoft de Publicación en la red, haga clic en No.

Para ejecutar su aplicación generada, abra un explorador y entre al directorio en el cual ha colocado sus páginas ASP, seguidas por la palabra clave *default.asp*. Si ha colocado sus páginas en el directorio "mywebs/apps/" entonces teclee los caracteres "/mywebs/apps/default.asp" en del campo de dirección de su explorador y presione <Enter>. Quizá necesite modificar algo de este texto dependiendo de cómo ha colocado su servidor de red.

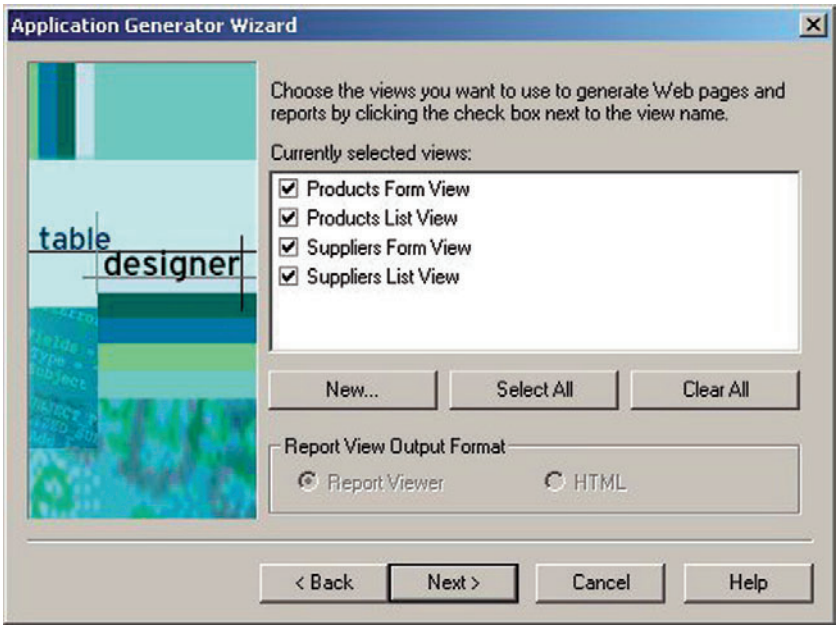
Ahora su explorador deberá conectarse con el servidor de la red y presentar una página de cuenta. Si está utilizando el SQL Server para su base de datos, introduzca *sa* sin contraseña, o alguna otra cuenta ID válida, y haga clic en Login. Si está utilizando Access no introduzca nada y haga clic en Login.

En este momento puede utilizar sus páginas para procesar su base de datos. Seleccione la página Products de la caja de lista desplegable y haga clic en Query. Verá un despliegue como el de la figura B-27. Puesto que el Tabledesigner da soporte a la consulta por forma, puede ingresar caracteres de búsqueda en esta forma. En la figura B-27, el usuario ha ingresado con >0 para UnitsInStock y New* en CompanyName. Este usuario está buscando productos de los que hay más de cero en existencia, y en los cuales el nombre del proveedor empieza con las letras *New*.

Cuando el usuario hace clic en Search (Buscar), se despliega la vista que se muestra en la figura B-28. Observe que todos los productos son abastecidos por compañías cuyos nombres empiezan con New. Ahora, si el usuario hace clic en alguno de los productos, el primer producto será desplegado en la vista Products Form. Se ha hecho esto en

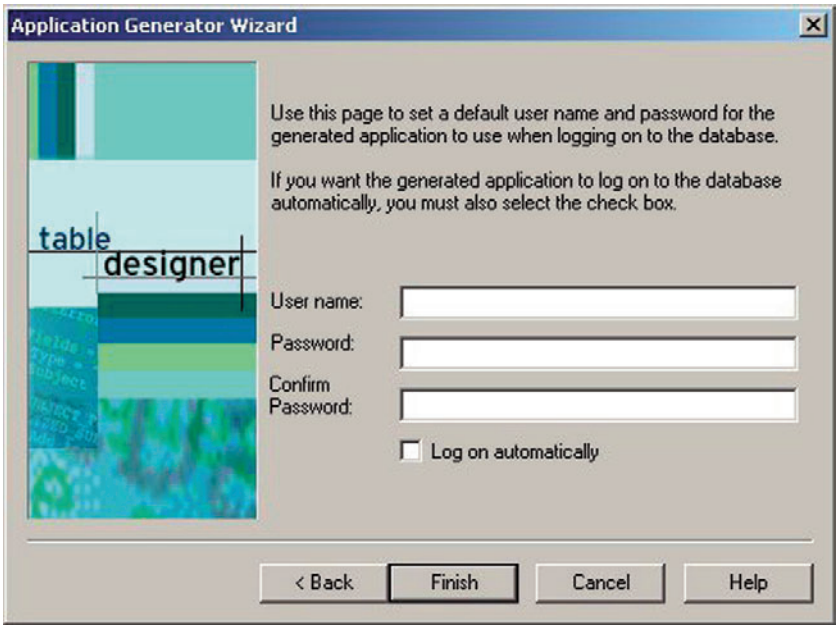
➤ FIGURA B-25

Especificación de las vistas para publicación



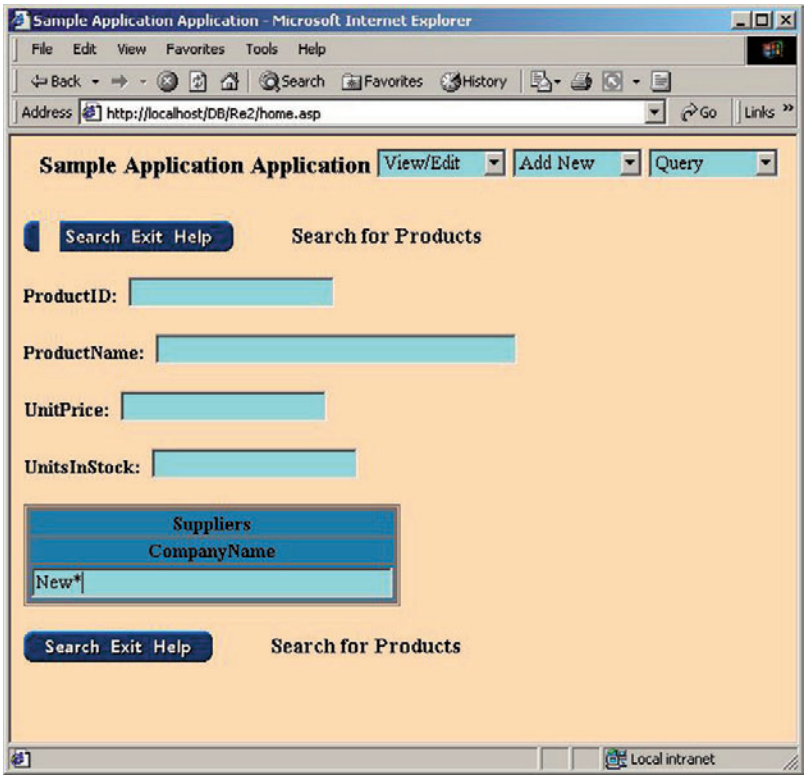
➤ FIGURA B-26

Pantalla para la colocación del nombre del usuario y de la contraseña predeterminada



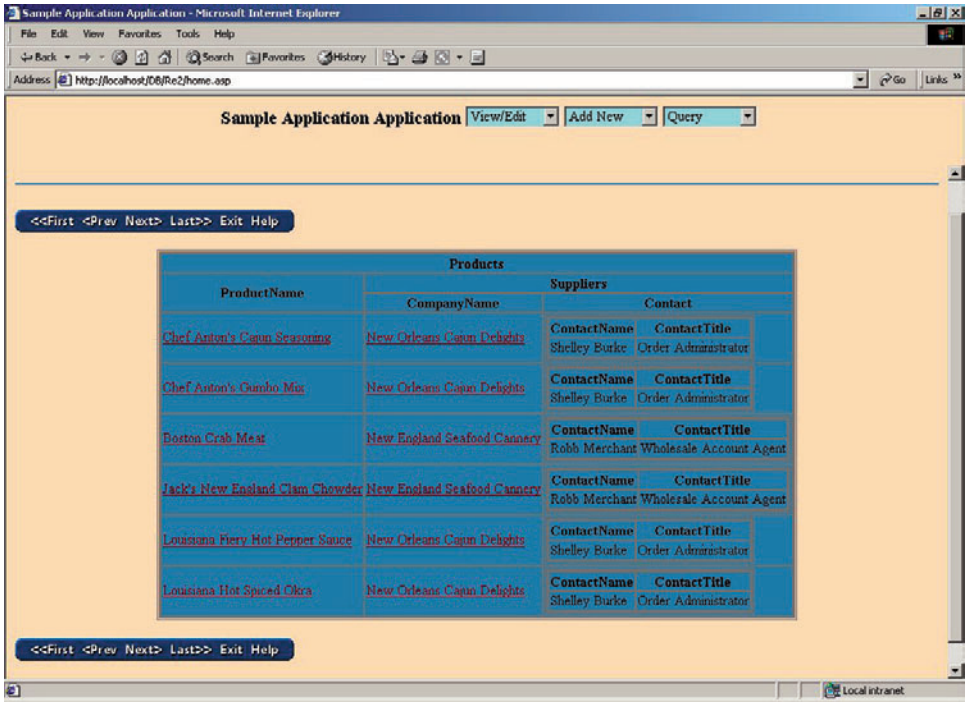
➤ FIGURA B-27

Uso de la consulta por forma



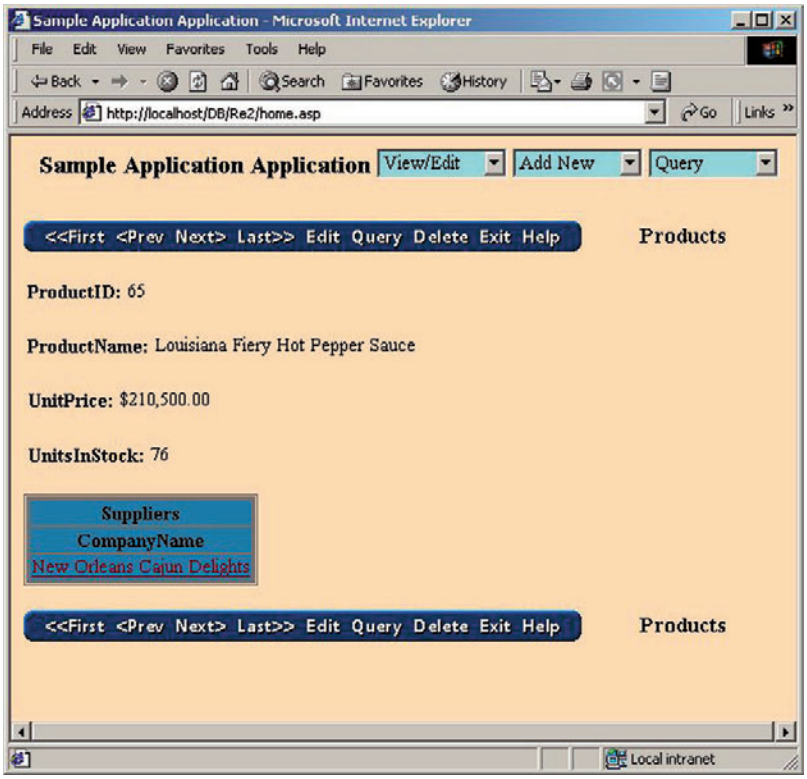
➤ FIGURA B-28

Resultados de la búsqueda en la figura B-27



➤ FIGURA B-29

Productos que se muestran en la vista Formas de Productos



la figura B-29, las páginas generadas aquí son realmente interactivas, así que usted necesita crear esta aplicación y utilizarla para obtener una apreciación cabal de ellas.

Por un lado, muestra en qué se basó Tabledesigner para responder a esta consulta. El filtro en CompanyName se aplica a la tabla Suppliers, pero la vista de lista se basa en la tabla Products. Además, hay una relación N:M entre ambas tablas. Esto significa que el Tabledesigner tuvo que crear un join externo izquierdo y un join interior a través de la tabla de intersección para responder a esta consulta.

➤ PASOS SIGUIENTES

Hay muchas otras características y funciones del Tabledesigner además de las descritas aquí. Sería conveniente que consultara la documentación de Ayuda del Tabledesigner para obtener mayor información. Un buen lugar para comenzar es con un resumen de Applications/Development Process Overview (Aplicaciones/Repaso del Proceso de Desarrollo). Puede utilizar el Tabledesigner para bases de datos normalizadas y no normalizadas y hoja de cálculo de datos. Vea el repaso de normalización para documentarse acerca de ese proceso.

➤ EJERCICIOS

B.1 Utilice el Tabledesigner para crear el modelo como el de la Universidad Highline que describimos en el capítulo 4. Cree el modelo que se muestra en la figura 14-13. Genere una base de datos en Access o en el SQL Server. Examine el esquema resultante y explique cómo se relaciona cada tabla con el modelo. Cambie el modelo de tal forma que PROFESSORs (PROFESOR) pueda trabajar en más de un DEPARTMENT (DEPARTAMENTO). Regenere su base de datos y explique los cambios que fueron hechos.

B.2 Utilice el Tabledesigner para crear un modelo con un ejemplo de cada uno de los siguientes tipos de objeto semántico (vea el capítulo 4):

a. Simple
b. Compuesto
c. Combinado
d. Híbrido
e. Arquetipo/versión
f. Asociación

Genere una base de datos de su modelo y examine las tablas resultantes. Explique de qué manera el Tabledesigner usa, o no usa, los mismos principios de diseño que se describieron en el capítulo 7.

B.3 Abra un nuevo modelo en el Tabledesigner y seleccione Recruiting starter kit (herramienta de inicio de reclutamiento). Examine el modelo y genere una base de datos. Proponga formas para que se pueda mejorar este modelo. Haga cambios en el modelo y regenere una base de datos. Explique los cambios que se hicieron en la estructura de la base de datos.

B.4 Termine el ejercicio B.1. Cree una Forma y una vista de Lista para cada objeto y use el Tabledesigner para generar páginas ASP para procesarlas. Coloque las páginas generadas en el directorio para que las procese el IIS o el Servidor de la red personal. Abra un explorador y utilice las páginas para crear objetos simples DEPARTMENT, STUDENT y PROFESSOR (DEPARTAMENTO, ESTUDIANTE y PROFESOR). Utilice las páginas de la red para enlazar varios objetos. Abra la base de datos implícita y examine los campos de las llaves externas. Justifique los valores de la llave externa a la luz de los enlaces que hizo entre los objetos.

B.5 Instale el SQL Server si aún no lo ha hecho. Siguiendo el ejemplo en este apéndice, cree un Sistema ODBC de fuente de datos para la publicación de la base de datos. Necesitará elegir el SQL Server como el driver y seguir el proceso de la cuenta del SQL Server. Introduzca el servidor (local) *sa* para la cuenta id sin contraseña, y cambie la base de datos predeterminada para las publicaciones. Use la ingeniería de reversa en las tablas publishers, authors, titles y titleauthors (publicitarias, autores, títulos y tituloautor) y cree una nueva base de datos Access (use el SQL Server si no tiene Access) y copie los datos ahí. Nombre a su nueva base de datos pubs2. Cambie el modelo (de pubs2) para que un título pueda tener más de un publicista. Regenere la base de datos y examine los cambios. Abra la tabla título en la base de datos de publicidad e imprima los renglones. Abra la tabla título en pubs2 e imprima los renglones. También abra la nueva tabla de intersección en pub2 e imprima los renglones. Explique las diferencias entre estas tablas.

B.6 Cree pubs2 como se describió en el ejercicio B.5. Genere una vista para publisher, author y title. Asegúrese de que la vista author incluya todos los títulos para ese autor y que su vista title incluya los nombres de todos los autores para ese título. Genere páginas de la red para estas vistas. Use las páginas Web para asignar segundos y terceros autores a alguno de los títulos. Cierre su aplicación Web y examine los valores de la llave externa en las tablas de la base de datos. Explique esos valores a la luz de las adiciones title/author que se hicieron.

GLOSARIO

Abstracción. Generalización de algo que oculta ciertos detalles sin importancia, pero que permite trabajar con una clase más amplia de tipos. Un conjunto de registros es una abstracción de una relación. Un conjunto de renglones es una abstracción de un conjunto de registros.

Administración de base de datos. La función que concierne al uso y control eficaces de una base de datos en particular y sus aplicaciones relacionadas.

Administración de datos. La función de extensión industrial que se refiere al uso y control de los activos de datos de la empresa. Puede ser una persona, pero normalmente es un grupo. Las funciones específicas incluyen el establecimiento de estándares y políticas de datos y el suministro de un foro para la resolución de conflictos. Véase también Administrador de base de datos.

Administrador de base de datos. La persona o grupo responsable del establecimiento de políticas y procedimientos para controlar y proteger una base de datos. Él (o ella) trabaja con base en un conjunto de pautas establecidas para la administración de datos, con el fin de controlar la estructura de la base de datos, administrar los cambios y conservar los programas DBMS.

ADO. Objetos de datos activos; una implementación de OLE DB que es accesible mediante lenguajes de objetos y lenguajes no orientados a objetos; se utiliza principalmente como lenguaje de escritura (JScript, VBScript) que se interconecta con el OLE BD.

ADT. Véase Tipo de datos abstractos.

Anomalía. Consecuencia indeseable de una modificación de datos utilizada principalmente en los análisis de normalización. Con una anomalía de inserción se deben agregar dos o más temas diferentes a un renglón de una relación. Con una anomalía de eliminación se pierden dos o más temas cuando se elimina un solo renglón.

Anomalía de eliminación. En una relación, se refiere a la situación en la cual la eliminación de un renglón de una tabla suprime datos sobre dos o más temas.

Anomalía de inserción. En una relación, la condición que existe cuando, para agregar un renglón completo a una tabla, se debe agregar información sobre dos o más temas lógicamente diferentes.

Anomalía de modificación. La situación que existe cuando el almacenamiento de un renglón en una tabla registra dos datos separados, o cuando la eliminación de un renglón de una tabla elimina dos datos separados.

API. Véase Interfaz del programa de aplicación.

Aplicación. Sistema computacional de negocios que procesa parte de una base de datos para satisfacer las necesidades de información del usuario. Contiene menúes, formas, reportes, consultas, páginas Web y programas de aplicación.

Aplicación de base de datos distribuida. Sistema computacional de negocios en el que la recuperación y la actualización de datos se presentan a través de dos o más computadoras independientes, y por lo general geográficamente distribuidas.

Applet. Programa compilado, codificado en bytes e independiente de la máquina Java, el cual se ejecuta mediante la máquina virtual de Java implementada en un explorador.

Árbol. Una colección de registros, entidades u otras estructuras de datos en la cual cada elemento tiene cuando mucho un padre, excepto el elemento superior, el cual no tiene padre.

Archivo plano. Un archivo que tiene solamente un valor en cada campo. El significado de las columnas es el mismo en cada fila.

Archivos ReDo. En Oracle, los respaldos de los segmentos de la recuperación regresiva utilizados para el respaldo y la recuperación. Existen archivos ReDo en línea y fuera de línea.

Arquitectura de base de datos cliente-servidor. Estructura de un sistema computacional en red, en la cual una computadora (usualmente una computadora personal) ejecuta los servicios en nombre de otras computadoras (por lo general una computadora personal). Para un sistema de base de datos, la computadora servidor, la cual se llama servidor de base de datos, procesa el DBMS, y las computadoras cliente procesan los programas de aplicación. Todas las actividades de la base de datos se llevan a cabo mediante el servidor de la base de datos.

Arquitectura de recursos compartidos. La estructura de una red de área local en la cual una microcomputadora ejecuta los servicios de procesamiento de archivos para otras computadoras. En una aplicación de base de datos, cada computadora de usuario contiene una copia del DBMS que solicita los requisitos entrada-salida del servidor de archivos. El servidor de archivos solamente procesa el archivo E/S; todas las actividades de base de datos los procesa el DBMS en la computadora del usuario.

Arquitectura de tres capas. Sistema de computadoras que tiene un servidor de base de datos, un servidor Web y una o más computadoras clientes. El servidor de base de datos hospeda un DBM; el servidor Web, un servidor http, y la computadora cliente hospeda un explorador. Cada renglón puede ejecutar un sistema operativo diferente.

ASP. Página del servidor activo. Un archivo que contiene un lenguaje de generación, una escritura del servidor, una escritura del cliente que se procesa mediante el Procesador del servidor activo en Microsoft IIS.

Atómico. Conjunto de acciones que se llevan a cabo como unidad. Se realizan todas las acciones, o no se realiza ninguna.

Atributo. (1) Una columna de una relación, también llamada columna, campo, o elemento de datos; (2) una propiedad en una entidad u objeto semántico; (3) una propiedad de datos o de relaciones en un objeto OOP; (4) en el modelo ODMG-3, una implementación de una propiedad de objetos en una implementación OOP, tal como C++ o una Smalltalk.

Atributo de no objeto. Atributo de un objeto semántico que no es un objeto.

Atributo de objeto. Atributo de un objeto semántico que representa un enlace con un objeto.

Atributo de un solo valor. En un objeto semántico, un atributo que tiene una cardinalidad máxima de uno.

Atributo de valores múltiples. El atributo de un objeto semántico que tiene una cardinalidad máxima mayor que uno.

Atributo pareado. En un objeto semántico, los atributos del objeto se parean. Si el objeto A tiene un atributo de objeto del objeto B, entonces el objeto B tendrá un atributo de objeto del objeto A, es decir, los atributos de objeto se parean uno con otro.

Atributos de clase. En el lenguaje del modelado unificado (UML), los atributos que pertenecen a la clase de todas las entidades de un tipo determinado.

Banda. Sección de una definición de reporte que contiene el formato de una sección de reporte. Por lo general, existen bandas para el encabezado y pie del reporte, el encabezado y pie de página y la línea de detalle del reporte. También existen bandas para puntos de agrupamiento o de interrupción dentro del reporte.

Banda de reporte. Véase Banda.

Base de datos. Conjunto autodescriptivo de registros integrados.

Base de datos distribuida. Una base de datos almacenada en dos o más computadoras. Los datos distribuidos pueden ser particionados o no particionados, duplicados o no duplicados.

Base de datos relacional. Base de datos que consta de relaciones. En la práctica, las bases de datos relacionales contienen relaciones con filas duplicadas. La mayoría de los productos DBMS incluyen una característica que elimina las filas duplicadas cuando es necesario. Por lo general, esa eliminación no se lleva a cabo debido a que requiere mucho tiempo y es costosa.

Bloqueo. El proceso de asignar un recurso de una base de datos a una transacción particular en un sistema de procesamiento concurrente. El tamaño del recurso bloqueado se conoce como granularidad de bloqueo. Con un bloqueo exclusivo, ninguna otra transacción puede leer o escribir el recurso. Con un bloqueo compartido, otras transacciones pueden leer el recurso, pero ninguna otra puede escribir en éste.

Bloqueo compartido. Bloqueo de un recurso de datos en el cual sólo una transacción puede actualizar los datos, pero muchas transacciones pueden leer concurrentemente dichos datos.

Bloqueo de dos fases. Procedimiento mediante el cual se obtienen los bloqueos y se liberan en dos fases. Durante la fase de crecimiento se obtienen los bloqueos, y durante la fase de decrecimiento se liberan los cierres. Una vez que se libera un cierre a ningún otro bloqueo se le concederá dicha transacción. Este procedimiento asegura la consistencia en las actualizaciones de base de datos en un ambiente de procesamiento concurrente de base de datos.

Bloqueo de dos fases distribuido. Bloqueo de dos fases en un ambiente distribuido, en el cual se obtienen y recuperan los candados en todos los nodos de la red. Véase Bloqueo de dos fases.

Bloqueo de recurso. Véase Bloqueo.

Bloqueo exclusivo. Bloqueo en un recurso de datos que ninguna otra transacción puede leer o actualizar.

Bloqueo explícito. Un bloqueo requerido por una instrucción de un programa de aplicación.

Bloqueo implícito. Bloqueo que el DBMS coloca automáticamente.

Bloqueo mortal. Condición que puede presentarse durante el proceso concurrente en el cual cada una de dos (o más) transacciones está esperando ingresar los datos que ha bloqueado otra transacción. También se llama abrazo mortal.

Bloqueo optimista. Estrategia de bloqueo en la cual se asume que no se presentarán conflictos; procesa una transacción y después verifica para determinar si se presentó un conflicto. Si se presentó alguno, se aborta la transacción. Véase también bloqueo pesimista.

Bloqueo pesimista. Estrategia de bloqueo que previene algún conflicto mediante la colocación de bloqueos antes del procesamiento de los requisitos de lectura y escritura de la base de datos. Véanse también Bloqueo optimista y Bloqueo mortal.

Botón de opción. En un ambiente GUI, un elemento de la interfaz del usuario en el cual éste puede seleccionar un elemento de una lista. Al hacer clic en un botón se deshace la selección del botón que se está presionando, si es el caso. Opera como los botones del radio de un automóvil, y es el mismo que un botón de radio (véase Botón de radio), pero se introdujo bajo un nombre diferente para evitar discusiones entre los proveedores.

Botón de radio. En un ambiente GUI, un elemento de la interfaz del usuario con el cual puede seleccionar un elemento de una lista. Al hacer clic en un botón suprime la selección del botón que se está presionando, si es el caso. Opera como los botones de radio en un estéreo de automóvil.

Buscar. El proceso de obtener datos relacionados mediante el valor de una llave externa, por ejemplo, cuando se procesa un renglón PEDIDO (*NúmerodePedido*, FechadePedido, *NúmerodeCliente*...).

Calendario de ejecución consistente. Lista ordenada de las operaciones de transacción que se realizan en una base de datos en la cual el resultado del procesamiento es consistente.

Campo. (1) Un grupo lógico de bytes en un registro utilizado para procesamiento de archivos. (2) En el contexto de un modelo relacional, es un sinónimo de atributo.

Cardinalidad. En una relación binaria, el número máximo o mínimo de elementos permitidos en cada lado de la relación. La cardinalidad máxima puede ser 1:1, 1:N, N:1 o N:M. La mínima puede ser opcional-opcional, opcional-obligatoria, obligatoria-opcional u obligatoria-obligatoria.

Cardinalidad máxima. (1) El número máximo de valores que puede tener un atributo dentro de un objeto semántico; (2) en una relación entre tablas, el número máximo de filas con las que puede relacionarse un renglón de una tabla en la otra tabla.

Cardinalidad mínima. (1) El número mínimo de valores que puede tener un atributo dentro de un objeto semántico; (2) en una relación entre tablas, el número de filas con las que se puede relacionar un renglón de una tabla en la otra tabla.

CGI. Véase Interfaz de compuerta común.

Clase de entidad. Conjunto de entidades del mismo tipo; por ejemplo, EMPLEADO y DEPARTAMENTO.

Clase de objeto. En la programación orientada a objetos, un conjunto de objetos con una estructura común.

Colección. Un objeto que contiene un grupo de otros objetos. Los ejemplos son las colecciones ADO de nombres, errores y parámetros.

Columna. Un grupo lógico de bytes en un renglón de una relación o una tabla. El significado de una columna es el mismo para cada renglón de la relación.

COM. Modelo de objetos componentes; una especificación Microsoft para el desarrollo de los programas orientados a objetos, los cuales permiten que tales programas trabajen conjuntamente de manera fácil.

Commit (comprometer). Una instrucción emitida al DBMS para convertir en permanentes las modificaciones de la base de datos. Una vez que se ha procesado la instrucción, los cambios se escriben en la base de datos y en un registro, de tal manera que sobrevivan a las suspensiones de energía y a otras fallas del sistema. Por lo general, una instrucción commit se utiliza al final de una transacción atómica. Compárese esto con recuperación mediante reprocesamiento.

Compatibilidad de unión. Condición en la que cualquiera de las dos tablas tienen el mismo número de atributos y los atributos en las columnas correspondientes que derivan del mismo dominio.

Computadora cliente. (1) Computadora personal en una red de área local con arquitectura de cliente servidor. En una aplicación de base de datos, la computadora cliente procesa los programas de aplicación de base de datos. Las solicitudes de acciones en la base de datos se envían a la computadora de la base de datos. (2) En la arquitectura de tres capas y de capas múltiples, es una computadora que hospeda a una memoria intermedia para ingresar a un servidor Web.

Concurrencia. Una condición en la cual se procesan al mismo tiempo dos o más transacciones de la base de datos. En un sistema de un solo CPU, los cambios se intercalan; en un sistema de CPU múltiples, se pueden procesar las transacciones simultáneamente y los cambios se intercalan en el servidor de la base de datos.

Confirmación de dos fases. En un sistema de base de datos distribuida, un proceso de confirmación entre nodos en el cual primero votan los nodos que puedan confirmar una transacción. Si todos los nodos votan por sí, se confirma la transacción. Si algún nodo vota por no, se aborta la transacción. Se requiere de una confirmación de dos fases para prevenir el procesamiento inconsistente de las bases de datos distribuidas.

Conflicto. Dos operaciones que tienen conflicto operan en el mismo elemento de datos, y cuando menos una de las operaciones es una escritura.

Conjunto de filas. En OLE DB, una abstracción de las colecciones de datos tales como conjuntos de registros, direcciones de correo electrónico y otros datos no relacionales.

Conjunto de registros. Un objeto ADO que representa una relación; se crea como resultado de la ejecución de una instrucción SQL, o un procedimiento almacenado.

Conjunto de tarjetas. En el lenguaje de generación inalámbrico, el conjunto de todas las tarjetas que comprenden un documento XML.

Consistencia. Dos o más transacciones simultáneas son consistentes si el resultado de su procesamiento es el mismo que se habría obtenido si se hubiesen procesado en algún orden serial.

Consistencia de nivel de instrucción. Todas las filas afectadas por una instrucción SQL están protegidas contra los cambios que realizan otros usuarios durante la ejecución de la instrucción. Compárese con consistencia de nivel de transacción.

Consistencia de nivel de transacción. Todas las filas impactadas por cualquiera de las instrucciones SQL en una transacción están protegidas de los cambios realizados durante la transacción entera. Este nivel de consistencia es costoso para llevarlo a cabo y puede (probablemente lo hará) reducir el rendimiento. También puede significar que una transacción no pueda ver sus propios cambios. Compárese con Consistencia de nivel de intrucción.

Constructor de objetos. En la programación orientada a objetos, una función que crea un objeto.

Consumidor de datos. Usuario de la funcionalidad OLE DB.

Control ActiveX. Objeto ActiveX el cual soporta las interfases que permiten el ingreso de las propiedades y los métodos del control en diversos ambientes de programación.

Controlador del driver (manejador). En ODBC, un programa que sirve como interfaz entre un programa de aplicación y un driver (manejador) ODBC. El driver determina lo que se requiere, lo carga en la memoria y coordina la actividad entre la aplicación y el driver. En los sistemas Windows lo proporciona Microsoft.

Copia de base de datos. Copia de los archivos de bases de datos que se puede utilizar para restaurar la base de datos en algún estado consistente previo.

CPU. Unidad Central de Procesamiento; la parte del hardware de la computadora que procesa instrucciones lógicas y aritméticas. Por lo general, el término CPU incluye a la memoria principal.

CRUD. Un acrónimo que representa las instrucciones Create (Crear), Read (Leer), Update (Actualizar), Delete (Eliminar), las cuales son las cuatro acciones que se pueden realizar en una vista de base de datos.

Cuadro de lista. En un ambiente GUI, un elemento de la interfaz del usuario en la cual se presenta una lista de opciones en un rectángulo. El usuario mueve el cursor para sombrear el elemento que quiere seleccionar de la lista.

Cuadro de verificación. En un ambiente GUI, un elemento de la interfaz del usuario, en el cual un usuario puede seleccionar uno o más registros de una lista. Los elementos se seleccionan al hacer clic en ellos.

Cuarta forma normal. Una relación en la tercera forma normal Boyce-Codd en la que cada dependencia de valor múltiple es una dependencia funcional.

Cubo. En OLAP, una estructura de presentación que tiene ejes sobre los cuales se colocan las dimensiones de los datos. Las medidas de los datos se muestran en las celdas del cubo. También se llama hipercubo.

Cuenta de materiales. Una estructura recursiva de datos en la cual una parte (o, por lo general, cualquier elemento) pueden ser un ensamble de otras partes y un componente de otros ensambles.

Cursor. Un apuntador de la posición o foco actual.(1) En una pantalla de computadora, un cuadro intermitente o un guión bajo que indica la posición en la cual se realizará la siguiente entrada. (2) En un archivo o en una instrucción SQL SELECT incorporada, la identidad del siguiente registro o renglón que se procesará.

Cursor deslizable. Un tipo de cursor que permite el movimiento de retroceso y de avance a lo largo de un conjunto de registros. En este texto se analizan tres tipos de cursores deslizables: instantáneo, de conjunto de teclas y dinámico.

Datos de base de datos. La parte de una base de datos que contiene datos de interés y utilidad para los usuarios finales de la aplicación.

Datos duplicados. En una base de datos distribuida, los datos que se almacenan en dos o más computadoras.

Datos encapsulados. Propiedades o atributos contenidos en un programa u objeto no visible o accesible para otros programas u objetos.

Datos significativos. Los metadatos que crea el DBMS para mejorar la ejecución; por ejemplo, índices y listas enlazados.

DBA. Véase Administrador de bases de datos.

DBM. Controlador de bases de datos. En un DBMS, el software que procesa alguna parte de una base de datos distribuida conforme a los requerimientos de acción que envían los controladores de transacciones distribuidas (DTM).

DBMS. Sistema de administración de base de datos; conjunto de programas utilizados para definir, administrar, y procesar la base de datos y sus aplicaciones.

DBMS de objeto relacional. Los productos DBMS que dan soporte a las estructuras de datos relacionales y orientadas a objetos, tal como Oracle.

DDBMS. Sistema de administración de bases de datos distribuidas. (1) Un producto comercial DBMS que soporta el procesamiento de una base de datos distribuida. Oracle y el servidor SQL proporcionan dispositivos limitados para el procesamiento de base de datos distribuidas.

DDL. Véase Lenguaje de definición de datos.

Declaración de tipo documental (DOM). Conjunto de elementos de generación que define la estructura de un documento XML.

Dependencia de valores múltiples. Una condición en una relación con tres o más atributos, en la cual parece que los atributos independientes tienen relaciones que en realidad no tienen. Formalmente, en una relación R (A, B, C) que tenga la clave (A, B, C), donde A se asocia con valores múltiples de B (o de C o de ambos), B no determina a C, y C no determina a B. Un

ejemplo es la relación EMPLEADO (NúmerodeEmpleado, HabilidadesdelEmpleado, NombredelDependiente), en donde un empleado puede tener valores múltiples de Habilidadesdelempleado y NombredelDependiente Habilidades del empleado, NombredelDependiente no tiene ninguna relación, pero en la relación parece tenerla.

Dependencia funcional. Relación entre atributos en la cual un atributo o grupo de éstos determina el valor de otro. Las expresiones X→Y, "X determina Y" y "Y es funcionalmente dependiente en X" significan que a partir de un valor específico de X se puede determinar el valor de Y.

Dependencia transitiva. En una relación que tiene cuando menos tres atributos, R (A,B,C), la situación en la cual A determina a B, B determina a C, pero B no determina a A.

Depósito de datos. Almacén de datos empresariales que está diseñado para facilitar la toma de decisiones administrativas. Un depósito de datos no sólo incluye datos, sino también metadatos, herramientas, procedimientos, capacitación, personal, y otros recursos que hacen más fácil e importante el acceso a los datos para aquellos que toman las decisiones.

Derechos y responsabilidades de procesamiento. Políticas organizacionales referentes a los grupos que pueden realizar acciones en elementos de datos específicos u otras colecciones de datos.

Derogar una instrucción. El desacuerdo directo del usuario con los datos que se usaron para fragmentar los totales de nivel superior en componentes.

Descargar. Copiar los datos de la base de datos de una computadora a otra, usualmente de una macrocomputadora o microcomputadora a una computadora personal o LAN.

Destructor de objetos. En la programación orientada a objetos, una función que destruye un objeto.

Detección del abrazo mortal. El proceso de determinar si dos o más transacciones están o no en un estado de abrazo o bloqueo mortal.

Determinante. Uno o más atributos que determinan funcionalmente a otro atributo o atributos. En la dependencia funcional (A, B)→ C, los atributos (A, B) son los determinantes.

DHTML. Una implementación Microsoft del estándar HTML 4.0. Las características clave las soporta el modelo documental de objetos, al organizar en cascada las hojas de estilo y los servicios de datos remotos.

Diagrama de estructura de datos. Una exposición gráfica de tablas (archivos) y sus relaciones. Las tablas se muestran en rectángulos y las líneas muestran las relaciones. Una relación a muchos se muestra con una muesca al final de la línea; una relación opcional se representa mediante un óvalo; y una relación obligatoria se representa con líneas punteadas.

Diagrama de objeto. Rectángulo orientado verticalmente que representa la estructura de un objeto semántico.

Diagrama de objeto semántico. Lo mismo que Diagrama de objeto.

Diagrama entidad-relación. Gráfica utilizada para representar las entidades y sus relaciones. Por lo general, las entidades se muestran en cuadrados o rectángulos, y las relaciones, en diamantes. La cardinalidad de relación se muestra dentro del diamante.

Diagrama E-R. Véase Diagrama entidad-relación.

Diccionario de datos. Catálogo de bases de datos y de metadatos de aplicación al que un usuario puede ingresar. Un diccionario de datos activo es aquel cuyos contenidos los actualiza automáticamente el DBMS cada vez que se realiza algún cambio en la estructura de base de datos o la aplicación. Un diccionario de datos pasivo es aquél cuyos contenidos deben actualizarse manualmente cuando se realizan los cambios.

Diferencia. Una operación algebraica relacional realizada en dos relaciones compatibles a la unión, A y B, de la que procede una tercera relación, C. Cada renglón en C se presenta en A, pero no en B.

Dimensión. En OLAP, una característica de datos que se coloca en un eje.

Diseño de aplicación. El proceso de creación de la estructura de programas y datos para satisfacer los requerimientos de la aplicación; también es la estructura de la interfaz de los usuarios.

Diseño de base de datos de lo general a lo particular. El diseño de una base de datos que trabaja de lo general a lo particular. La base de datos resultante puede satisfacer todas las necesi-

dades de una organización; el peligro es que nunca puede completarse. Véase Diseño de base de datos de lo particular a lo general.

Diseño de base de datos de lo particular a lo general. El diseño de una base de datos que trabaja de lo particular y específico a lo general. Aunque esta clase de diseño toma poco tiempo, puede resultar en una base de datos que tenga un alcance muy limitado.

Disparador. Un tipo especial de procedimiento almacenado que invoca el DBMS cuando se presenta una condición específica. Los disparadores (triggers) ANTES se ejecutan antes de una acción específica de base de datos, los disparadores DESPUÉS se ejecutan después de una acción específica de base de datos, y los disparadores EN VEZ DE normalmente se utilizan en lugar de una acción específica de base de datos. Los disparadores EN VEZ DE por lo general se utilizan para actualizar los datos en las vistas SQL.

DML. Véase Lenguaje de manejo de datos.

Documento de tipo no válido. Documento XML que no conforma a su DTD ni tiene un DTD; compárese esto con un documento de tipo válido.

Documento de tipo válido. Un documento XML que conforma su DTD; compárese con el documento de tipo no válido.

Documento válido de esquema. Un documento XML de conformidad con su definición de esquema XML.

DOM. Véase Modelo documental de objetos.

Dominio. (1) El conjunto de todos los valores posibles que puede tener un atributo. (2) Descripción del formato (tipo de datos, longitud) y la semántica (significado) de un atributo.

Dominio de fórmula. Un dominio cuyos valores están calculados por una expresión la cual contiene argumentos que son dominios.

Driver (manejador). En ODBC, un programa que sirve como interfaz entre el controlador del driver ODBC y un producto DBMS en particular. Funciona con las computadoras cliente en una arquitectura cliente-servidor.

Driver de filas múltiples. En ODBC, un driver de dos partes, usualmente para un sistema de base de datos cliente-servidor. Una parte del driver reside en el cliente e interactúa con la aplicación; la segunda parte reside en el servidor e interactúa con el DBMS.

Driver de una sola fila. En ODBC, un driver de base de datos que acepta las instrucciones SQL del controlador del driver y los procesa sin tener que invocar a otro programa o al DBMS. Un driver de un solo renglón es un driver ODBC y un DBMS; se utiliza en los sistemas de procesamiento de archivos.

DSD. Véase Diagrama de estructura de datos.

DSS. Sistema de soporte de decisiones; un dispositivo interactivo, basado en computadoras para el apoyo a la toma de decisiones, especialmente en el caso de problemas semiestructurados y no estructurados. A menudo dicho sistema incluye una base de datos y un dispositivo para consultar-actualizar el procesamiento particular de requisitos.

DTD. Véase Declaración de tipo documental.

DTS. Véase Servicio de transacción distribuida.

Duplicación. Para Oracle y el servidor SQL, es un término que se refiere a las bases de datos que se distribuyen en más de una computadora.

ECMAScript-262. La versión estándar de un lenguaje de fácil aprendizaje e interpretación que se utiliza para el servidor Web y las aplicaciones de procesamiento del cliente Web. La versión Microsoft se llama Jscript, y la versión Netscape, JavaScript.

Eje. En el OLAP, una coordenada de un cubo o hipercubo.

Elemento de datos. (1) Un grupo lógico de bytes en un registro, generalmente utilizado con el procesamiento de archivos. (2) En el contexto del modelo relacional, un sinónimo de atributo.

Eliminación en cascada. Una propiedad de una relación que indica que cuando se elimina una fila, las filas relacionadas también deben eliminarse.

Empresa Java Beans. Dispositivo para la administración de objetos distribuidos y de procesamiento distribuido en el mundo del desarrollo de Java.

Enlazar. Conectar una variable de programa o un control GUI a una columna de una tabla o consulta.

Entidad. (1) Algo de importancia para el usuario, lo cual se necesita representar en una base de datos; (2) en un modelo entidad-relación, las entidades se restringen de cosas que pueden representarse mediante una tabla. Véanse también entidad dependiente de existencia, entidad fuerte, y entidad débil.

Entidad débil. En un modelo entidad-relación, una entidad cuya existencia lógica en la base de datos depende de la existencia de otra entidad. Véase también Entidad dependiente de ID y Entidad fuerte.

Entidad dependiente de existencia. Sinónimo de entidad débil. Una entidad que no puede aparecer en la base de datos, a menos que también aparezca una ocurrencia de una u otras entidades en la base de datos. Una subclase de las entidades dependientes de existencia son las entidades dependientes de ID.

Entidad dependiente de ID. Entidad que no puede existir lógicamente sin la existencia de otra entidad. Por ejemplo, una CITA no puede existir sin un CLIENTE que concerte la cita. La entidad dependiente de ID siempre contiene la clave de la entidad de la cual depende. Dichas entidades son un subconjunto de una entidad débil. Véase también Entidad fuerte y Entidad débil.

Entidad fuerte. En un modelo entidad-relación, cualquier entidad cuya existencia en la base de datos no dependa de la existencia de alguna otra entidad. Véase también Entidad dependiente de ID y Entidad débil.

Escrito forzado. Escrito de una base de datos en el cual el DBMS espera el reconocimiento del sistema operativo de que la imagen después se ha escrito exitosamente en el registro.

Escritor de reportes con bandas. Un escritor de reportes en el cual las secciones de éstos están definidas por bandas. Véase también Banda.

Espacio de nombre de objetivo. En un documento de esquema de XML, el espacio de nombre que el esquema crea.

Espacio de nombre etiquetado. En un documento de esquema XML, un espacio de nombre al que se le da un nombre (etiqueta) dentro del documento. Se supone que todos los elementos precedidos del espacio de nombre etiquetado se definirán en ese espacio de nombre etiquetado.

Espacio de nombre predeterminado. En un programa de esquema XML, el espacio del nombre que se utiliza en todos los elementos no etiquetados.

Espacios de nombre XML. Un estándar para la asignación de nombres para colecciones definidas. X.Nombre se interpreta como el elemento Nombre, como lo define el espacio de nombre X. Y.Nombre se interpreta como el elemento Nombre, como lo define el espacio de nombre Y. Es útil para evitar la ambigüedad de términos.

Esquema. Vista lógica completa de la base de datos.

Esquema copo de nieve. En una base de datos OLAP, la estructura de las tablas para dicha tablas de dimensión puede estar a diversos niveles de la tabla que almacena los valores de medida. Generalmente, se normalizan dichas tablas de dimensiones. Compárese con esquema estrella.

Esquema estrella. En una base de datos OLAP, la estructura de las tablas en la que cada dimensión de la tabla es adyacente a la tabla que almacena los valores de medida. Compárese con el esquema copo de nieve. En el esquema estrella a menudo las tablas de dimensión no están normalizadas.

Esquema relacional. Conjunto de relaciones con restricciones interrelacionales.

Esquema XML. Un lenguaje flexible del XML para la restricción de la estructura de un documento XML. Extiende y reemplaza los DTD. Se encuentra en desarrollo y es muy importante para el procesamiento de datos.

Estándar de conectividad abierta de una base de datos (ODBC). Una interfaz estándar por medio de la cual los programas de aplicación pueden accesar y procesar las bases de datos SQL (o estructuras de tablas como hojas de cálculo y tablas de texto) de manera independiente al DBMS. La porción del controlador del driver del ODBC se incorpora en Windows. Los drivers ODBC los proporcionan los proveedores de DBMS, Microsoft y otro tercer grupo de desarrolladores de software.

Estructura encapsulada. Parte de un objeto que no es visible para otros objetos.

Exportar. Función del DBMS para escribir un archivo de datos en volumen. Se intenta leer el archivo mediante otro DBMS o programa.

Extensión (de un objeto). En el modelo ODMG, la unión de todas las ocurrencias del objeto. Los atributos pueden declararse para que sean únicos en toda la extensión de un objeto.

Extracción. Parte de una base de datos operacional descargada a una red de área local, o a una computadora personal para procesamiento local. Las extracciones se crean para reducir el costo y el tiempo de comunicación, cuando se consultan y se crean reportes de datos mediante el procesamiento de transacción.

Falla de aplicación. Una falla en el procesamiento de una instrucción DBMS, o en una transacción, la cual se debe a los errores de lógica de la aplicación.

Falla de medios. Falla que ocurre cuando el DBMS no puede escribir en un disco. Por lo general es provocada por una falla en la cabeza del disco, u otro problema del disco.

Falla de ocurrencia. Falla en el sistema operativo o hardware que provoca que el DBMS funcione mal.

Fase de crecimiento. La primera etapa en el bloqueo de dos fases en la cual se adquieren pero no se liberan los bloqueos.

Fase de decrecimiento. En el bloqueo de dos fases, la etapa en la cual se liberan los bloqueos, pero no se adquiere ninguno.

Forma. (1) Un despliegue en un monitor de una computadora utilizada para presentar, introducir y modificar datos. También se denomina forma o panel de entrada de datos. (2) Documento en papel empleado en un sistema de negocios para registrar datos, usualmente referentes a una transacción. Las formas se analizan en el proceso de la construcción de un modelo de datos.

Forma normal. Regla o conjunto de reglas que rigen la estructura permitida de las relaciones. Las reglas se aplican a los atributos, dependencias funcionales, dependencias de valores múltiples, dominios y restricciones. Las formas normales más importantes son 1NF, 2NF, 3NF, Boyce-Codd, NF, 4NF, 5NF, y la forma normal dominio-clave.

Forma normal de Boyce-Codd. Una relación en la tercera forma normal en la cual cada determinante es una clave candidata.

Forma normal Dominio-Clave (DK/NF). Relación en la que todas las restricciones son consecuencias lógicas de los dominios y claves.

Fragmento. Un renglón en una tabla (o registro en un archivo) en el que no se presenta un padre o hija requeridos. Por ejemplo, un renglón en una tabla ARTÍCULO DE LÍNEA en la que no existe un renglón ORDEN.

FTP. Protocolo de transferencia de archivos. Un servicio estándar de Internet para el copiado de archivos o de un servidor remoto o de éste.

Fuente de datos. En el estándar ODBC, una base de datos junto con su DBMS asociado, sistema operativo y plataforma de red.

Fuente de datos de archivos. Una fuente de datos ODBC almacenada en un archivo que puede enviarse por correo electrónico, o distribuirse entre los usuarios.

Fuente de datos del sistema. Fuente de datos ODBC que es local para una computadora, a la que pueden ingresar el sistema operativo de dicha computadora y los usuarios seleccionados de tal sistema operativo.

Fuente de datos del usuario. Fuente de datos ODBC que sólo está a la disposición del usuario que la creó.

Función incorporada. En el SQL, cualquiera de las funciones COUNT (CONTAR), SUM (SUMAR), AVG (PROMEDIO), MAX (MAXIMIZAR) o MIN (MINIMIZAR).

Generador de formas. Porción del subsistema de desarrollo de aplicaciones utilizado para crear una forma de entrada de datos sin tener que escribir cualquier código de programa de aplicación.

Grado. Para las relaciones en el modelo entidad-relación, el número de entidades que participan en la relación. En casi todos los casos, dichas relaciones son de grado 2.

Granularidad. El tamaño de un recurso de base de datos que puede bloquearse. El bloqueo de una base datos entera es una granularidad grande; el bloqueo de una columna de un renglón particular es una granularidad pequeña.

Granularidad de bloqueo. El tamaño de un elemento de datos bloqueado. El bloqueo del valor de una columna de un renglón particular es una granularidad de bloqueo pequeña, y el bloqueo de una tabla entera es una granularidad de bloqueo grande.

Grupo compuesto. Grupo de atributos en un objeto semántico que tiene valores múltiples y no contiene otros atributos de objetos.

Herencia. Característica de los sistemas orientados a objetos, en la cual los que son subtipos de otros objetos obtienen atributos (datos o métodos) de sus subtipos.

Hermano. Registro o nodo que tiene el mismo padre de otros registros y nodos.

Hija. Una fila, registro, o nodo en muchos lados de una relación uno a varios.

Hipercubo. En OLAP, una estructura de presentación que tiene ejes sobre los cuales se colocan las dimensiones de los datos. Las medidas de los datos se muestran en las celdas del hipercubo. También se llama cubo.

HOLAP. OLAP híbrido que utiliza una combinación de ROLAP y MOLAP para el soporte del procesamiento OLAP.

HTML. Véase Lenguaje de marcaje de hipertexto.

HTML 4.0. El estándar del consorcio mundial de la Web para una versión avanzada de HTML. Véase DHTML.

Huérfano. Cualquier renglón (registro) al que le falta su padre en una relación obligatoria uno a muchos.

Identificador de filas. En SQL3, un identificador único proporciona el sistema, una clave sustituta. El identificador de filas se puede hacer visible mediante la instrucción CON IDENTIDAD (WITH IDENTITY) en la definición de la tabla.

Identificador de grupo. Un atributo que identifica una ocurrencia única de un grupo dentro de un objeto semántico u otro grupo.

Identificador de objeto. Atributo que se utiliza para especificar una ocurrencia de un objeto. Los identificadores de objetos pueden ser únicos, es decir, identifican a una (y sólo a una) ocurrencia, o no únicos, esto es, que identifican exactamente una ocurrencia del objeto.

I desconocido (IUnknown). Interfaz ActiveX en la cual un programa ActiveX puede llamar a otro ActiveX desconocido. Una vez que se ha establecido una conexión, el primer programa puede utilizar la interfaz Consultar para decidir a cuáles objetos, métodos y propiedades les da soporte el programa.

IIS. Servidor de información de Internet; un producto Microsoft que opera como un servidor HTTP. IIS requiere de Windows 2000.

Imagen antes. Registro de una entidad de base de datos (por lo general un renglón o una página) antes de un cambio. Se utiliza para la recuperación mediante reprocesamiento.

Imagen después. Registro de una entidad de base de datos (generalmente un renglón o una página) después de un cambio. Se utiliza para las recuperaciones progresivas.

Implementación. En OOP, un conjunto de objetos que crea una interfaz particular OOP.

Importar. Función del DBMS, para leer un archivo de datos en volumen.

Incompatibilidad de unión. La condición en la cual cualquiera de las dos tablas tiene un número diferente de atributos, o los atributos en las columnas correspondientes que se originan en dominios diferentes.

Independencia programa-datos. La condición que existe cuando la estructura de los datos no está definida en programas de aplicación. En su lugar, se define en la base de datos, y después los programas de aplicación lo obtienen del DBMS. De esta forma, se pueden hacer los cambios en las estructuras de datos que no necesariamente se pueden realizar en los programas de aplicación.

Indicador. Dirección de una ocurrencia en un elemento de datos de una estructura.

Indicador X. Un estándar para enlazar un documento con otro. La trayectoria X tiene muchos elementos del indicador X.

Índice. Datos significativos utilizados para mejorar y clasificar la ejecución. Los índices pueden construirse para una columna o grupos de columnas. Son especialmente útiles para las columnas que se utilizan para saltos de control en reportes, y para especificar las condiciones en las juntas.

Instrucción. Ingreso de una instrucción a una aplicación de base de datos, mediante la cual los usuarios especifican la actividad que se realizará. Compárese esto con menú.

Integridad de datos. El estado de una base de datos en la cual se satisfacen todas las restricciones; usualmente se refiere a las restricciones de interrelación en las cuales se requiere que se presente el valor de una clave ajena en la tabla que tiene a ésta como su clave primaria.

Intercambio (swizzling). En OOP, el proceso de la conversión de un identificador de objeto permanente en una dirección en la memoria y viceversa.

Interfaz. (1) Medios por el cual dos o más programas se llaman uno al otro; la definición de las llamadas de procedimiento entre dos o más programas. (2) En OOP, el diseño de un conjunto de objetos que incluye los nombres, métodos y atributos del objeto.

Interfaz de consulta. Una interfaz en Microsoft COM que puede utilizarse para determinar los objetos, los métodos y las propiedades soportados por un programa ActiveX.

Interfaz de lenguaje natural. Una interfaz para un programa de aplicación o DBMS mediante el cual los usuarios pueden introducir requisitos en la forma del inglés estándar, u otro lenguaje humano.

Interfaz de pasarela común. Estándar para definir formas enviar y recibir datos de formas en documentos HTML.

Interfaz del programa de aplicación (API). Conjunto de procedimientos o funciones del programa que pueden llamarse para solicitar un grupo de servicios. El API incluye los nombres de los procedimientos y funciones y una descripción del nombre, propósito y tipo de datos de los parámetros que se determinan. Por ejemplo, un producto DBMS puede proporcionar una librería de funciones para llamar los servicios de la base de datos. Los nombres de los procedimientos y sus parámetros constituyen el API de dicha librería.

Internet. Red mundial pública de computadoras que se comunican mediante el TCP/IP.

Intersección. Operación algebraica relacional ejecutada en dos relaciones de unión compatibles, A y B, que forma una tercera relación, C, para que C contenga solamente las filas que aparecen en A y B.

Intranet. (1) Red local privada o de área amplia que usa TCP/IP, HTML y una tecnología de explorador relacionada en las computadoras cliente y la tecnología del servidor Web en donde funciona. (2) Menos común, una LAN o WAN privadas que involucran a clientes y servidores.

Java. Lenguaje de programación de objetos, la cual tiene una mejor administración de memoria y una verificación de límites más eficiente que el C++. Se utiliza principalmente para las aplicaciones de Internet, pero también se puede emplear como lenguaje de programación con un propósito general. Los compiladores Java generan el código de bytes Java que se interpreta en las computadoras clientes.

Java bean. Una clase de métodos exclusivos de Java, que puede uno llevar tranquilamente a casa de mamá. Los beans no tienen variables de ocurrencias públicas, todos sus valores persistentes se ingresan mediante métodos de acceso y no tienen ningún constructor, o sólo tienen un constructor, con ningún argumento explícitamente definido.

JavaScript. Lenguaje propiamente de escritura de Netscape. La versión de Microsoft se llama Jscript; la versión estándar, ECMAScript-262. Éstos son lenguajes de fácil interpretación y aprendizaje que utiliza el servidor Web y el procesamiento de aplicaciones del cliente Web. A veces se escribe "Java Script".

JDBC. Interfaz estándar mediante la cual los programas de aplicación escritos en Java pueden accesar y procesar las bases de datos SQL (o las estructuras de tablas tales como hojas de cálculo y tablas de texto) en forma independiente del DBMS. No se aplica a la Conectividad de la base de datos Java.

Jerarquía de generalización. Conjunto de objetos o entidades del mismo tipo lógico que se organizan en una jerarquía de subtipos lógicos. Por ejemplo, EMPLEADO tiene los subtipos INGENIERO y CONTADOR, e INGENIERO tiene los subtipos INGENIERÍA ELÉCTRICA e INGENIERÍA MECÁNICA. Los subtipos heredan las características de sus subtipos.

Join (Junta). Operación algebraica relacional en dos relaciones, A y B, que produce una tercera relación, C. Un renglón de A se conecta con una de B para formar una nueva en C, si es que las filas en A y B satisfacen las restricciones concernientes a sus valores. Por ejemplo, A1 es un atributo en A, y B1 es un atributo en B. La junta de A con B en la cual A1 < B1 resultará en una relación, C, que tiene la concatenación de las filas en A y B, en las cuales el valor de A1 es menor que el valor de B1. Véase Join igual y Join natural.

Join exterior. Junta en la cual todas las filas de una tabla aparecen en el resultado, sin importar si tienen una asociación en la condición del join. En el join exterior izquierdo aparecen to-

das las filas del lado izquierdo en la relación; en un join exterior derecho, aparecen todas las filas de la relación del lado derecho.

Join igual. El proceso de la unión de la relación A que contenga el atributo A1, con la relación B que contenga el atributo B1 para formar la relación C, para que cada renglón en C, A1= B1. A1 y B1 se representan en C.

Join interior. Sinónimo de join.

Join natural. Un join de una relación A que tiene un atributo A1 con una relación B que tiene un atributo B1, donde A1 es igual a B1. La relación de los join, C, contiene cualquiera de las columnas A1 o B1, pero no ambas. Compárese esto con Join igual.

JScript. Lenguaje de escritura del propietario desarrollado por Microsoft. La versión Netscape se llama JavaScript; la versión estándar, ECMAScript-262. Estos lenguajes son de fácil aprendizaje e interpretación que utilizan el servidor Web y las aplicaciones de procesamiento del cliente Web.

JSP. Véase Página del servidor Java.

LAN. Red de área local; grupo de microcomputadoras conectadas entre sí por medio de líneas de comunicación con una proximidad relativamente pequeña, usualmente menos de una milla. Véanse Arquitectura de base de datos cliente-servidor y Arquitectura de recursos compartidos.

Lectura confirmada. Un nivel de aislamiento de transacción que prohíbe las lecturas sucias, pero permite las lecturas no repetibles y las lecturas fantasmas.

Lectura no confirmada. Un nivel de aislamiento de transacción que permite las lecturas sucias, las lecturas no repetibles y las lecturas fantasmas.

Lectura repetible. Nivel de aislamiento de transacción que no permite las lecturas sucias y no repetibles. Pueden presentarse lecturas fantasmas.

Lectura sucia. La lectura de datos que ha sido cambiada, pero que aún no se ha enviado en la base de datos. Dichos cambios pueden revertirse y removerse de la base de datos.

Lecturas fantasmas. Situación que se presenta cuando una transacción lee datos que ya ha leído anteriormente y encuentra filas nuevas que fueron insertadas por una transacción realizada.

Lecturas no repetibles. La situación que se presenta cuando una transacción lee datos que ya ha leído anteriormente y encuentra modificaciones o eliminaciones generadas por una transacción ocurrida.

Lenguaje de consulta-actualización. Lenguaje que pueden utilizar los usuarios finales para consultar la base de datos y hacer cambios en los datos.

Lenguaje de definición de datos (DDL). Lenguaje utilizado para describir la estructura de una base de datos.

Lenguaje de marcaje de hipertexto. Sistema estandarizado de identificación de texto para el formato, localización de imágenes y otros archivos que no son de texto y colocación de enlaces o referencias a otros documentos.

Lenguaje de manipulación de datos (DML). Un lenguaje utilizado para describir el procesamiento de una base de datos.

Lenguaje de marcaje inalámbrico (WML). Una variante del XML desarrollada para permitir que las páginas Web se desplieguen en dispositivos inalámbricos.

Lenguaje de modelado unificado. Véase UML.

Lenguaje estándar de marcaje extensible (SGML). Un medio estándar para etiquetar y marcar formato, estructura y contenido de documentos. El HTML es un subconjunto del SGML.

Lenguaje orientado a la transformación. Un sublenguaje de datos, tal como el SQL, que proporciona instrucciones y habilidades para transformar un conjunto de relaciones en una nueva relación.

Librería de clases de objetos. En la programación orientada a objetos, una colección de clases de objetos; usualmente una colección que sirve a una finalidad particular.

Límite de transacción. El grupo de instrucciones de base de datos que deben realizarse o abortarse como unidad.

Lista enumerada. Lista de valores permitidos para un dominio, atributo o columna.

Llave. (1) Un grupo de uno o más atributos que identifican un renglón único en una relación. Debido a que las relaciones no pueden tener filas duplicadas, cada relación debe tener cuando

menos una llave, que es la combinación de todos los atributos en la relación. Algunas veces una llave se denomina llave lógica. (2) Con algunos productos relacionales DBMS, se utiliza un índice en una columna para mejorar el acceso y la rapidez de clasificación. Algunas veces se denomina llave física.

Llave ajena. Un atributo que es una llave de una o más relaciones diferentes de aquélla en que aparece.

Llave candidata. Atributo o grupo de atributos que identifican un renglón único en una relación. Se elige una de las llaves candidatas para que sea la llave primaria.

Llave compuesta. Clave con más de un atributo.

Llave física. Columna que tiene un índice u otra estructura de datos creadas para ella; es un sinónimo de índice. Dichas estructuras se crean para mejorar la búsqueda y la clasificación en los valores de las columnas.

Llave lógica. Una o más columnas que únicamente determinan el renglón de una tabla o el registro de un archivo; un sinónimo de llave. Compárese esto con una llave física, la cual es sinónimo de índice.

Llave primaria. Llave candidata seleccionada como la llave de una relación.

Llave sustituta. Un identificador único proporcionado por el utilizado como la llave primaria de una relación. Los valores de una llave sustituta no tienen ningún significado para los usuarios, y generalmente se ocultan en formas y reportes.

Máquina DBMS. Subsistema DBMS que procesa requisitos lógicos E/S de otros subsistemas DBMS, y envía requisitos físicos E/S al sistema operativo.

Máquina virtual Java. Un interpretador de código de bytes de Java que se ejecuta en el ambiente de una máquina en particular; por ejemplo, Intel 386, Alfa. Por lo general, se implementan dichos interpretadores en exploradores, incluido con el sistema operativo, o forman parte de un ambiente de desarrollo de Java.

Materialización. Una vista de base de datos, ya sea que aparezca en una forma, reporte, o página Web.

Me. En OOP, un indicador especial de la ocurrencia del objeto actual. Por ejemplo, Me.Nombre se refiere al nombre del atributo del objeto actual.

Medida. En OLAP, los datos de fuente para el cubo; los datos que se despliegan en las celdas. Pueden ser datos no procesados, o funciones de datos no procesados como SUM (SUMA), AVG (PROMEDIO) u otros cálculos.

Memoria intermedia. Área de memoria utilizada para conservar datos. Para una lectura, los datos se leen en una memoria intermedia desde un mecanismo de almacenamiento; para una escritura, los datos se escriben en el almacenamiento desde la memoria intermedia.

Menú. Lista de opciones que se presentan al usuario de una aplicación de base de datos u otra aplicación. El usuario selecciona de entre lista la siguiente acción o actividad. Las acciones se restringen a las opciones de la lista. Compárese esto con Instrucción.

Menú acción/objeto. Sistema de menúes en el cual el menú principal se refiere a las acciones, y los menúes subsecuentes se refieren a los objetos en los cuales se realizan dichas acciones.

Mercado de datos. Dispositivo similar a un depósito de datos, pero para un dominio restringido. A menudo los datos se restringen a tipos, funciones de negocios, o unidades de negocios en particular.

Metadatos. Datos referentes a la estructura de datos en una base almacenada en el diccionario de datos. Los metadatos se utilizan para describir tablas, columnas, restricciones, índices, etcétera. Compare esto con los metadatos de aplicación.

Metadatos de aplicación. Diccionario de datos; datos referentes a la estructura y contenido de los menúes de aplicación, formas y reportes.

Método. (1) Un programa sujeto a un objeto de una programación orientada a objetos (OOP). Un método puede heredarse de los objetos OOP de nivel inferior. (2) En OOP, un atributo de programa de un objeto.

Miembros. En OLAP, los valores de una dimensión.

Modelo de datos. (1) Modelo de los requerimientos de datos de los usuarios, expresado en términos del modelo entidad-relación, o del modelo de objeto semántico. Algunas veces se deno-

mina modelo de datos de los usuarios. (2) Lenguaje para la descripción de la estructura y procesamiento de una base de datos.

Modelo de datos de red. Modelo de datos que soporta cuando menos las relaciones simples de una red. El CODASYL DBTG, el cual soporta las relaciones simples de una red, pero no relaciones complejas, es el modelo de datos de red más importante.

Modelo de datos jerárquicos. Un modelo de datos que representa todas las relaciones mediante jerarquías o árboles. Las estructuras de red se deben descomponer en árboles antes de que se representen mediante un modelo de datos jerárquico. DL/I es el único modelo de datos jerárquico que existe.

Modelo de datos relacional. Modelo de datos en el cual se almacenan los datos en relaciones y éstas se representan entre las filas mediante valores de datos.

Modelo de objeto semántico. Las construcciones y convenciones utilizadas para crear un modelo de los datos del usuario. Las cosas en el mundo de los usuarios se representan mediante objetos semánticos (algunas veces llamados objetos). Las relaciones se modelan en los objetos, y por lo general los resultados se documentan en los diagramas de objetos.

Modelo documental de datos (DOM). Un API que representa un documento XML como un árbol. Cada nodo del árbol representa una pieza del documento XML. Un programa puede accesar directamente y manejar un nodo de la representación DOM.

Modelo entidad-relación. Las construcciones y convenciones que se utilizan para crear un modelo de datos del usuario (véase Modelo de datos). Las cosas en el mundo del usuario están representadas por entidades, y las asociaciones entre éstas están representadas mediante relaciones. Usualmente, los resultados se documentan en un diagrama entidad-relación.

MOLAP. OLAP multidimensional que utiliza un procesador diseñado, con el fin de dar soporte al procesamiento OLAP.

Nada (Nothing). En OOP, una referencia de objetos nulos utilizada para establecer un indicador de objetos que anule o verifique un indicador de objetos para determinar si es nulo.

Nivel. En OLAP, un subconjunto (posiblemente jerárquico) de una dimensión.

Nivel de aislamiento. Véase nivel de aislamiento de transacción.

Nivel de aislamiento de transacción. El grado en el cual se protege una base de datos de las acciones realizadas por otras transacciones. El estándar SQL 1992 especificó cuatro niveles de aislamiento: lectura no confirmada, lectura confirmada, lectura repetible y serializable.

N.M. Una abreviación de una relación muchos a muchos entre las filas de dos tablas.

Nodo. (1) Una entidad en un árbol; (2) una computadora en un sistema de procesamiento distribuido.

Nodo de transacción. En un sistema de base de datos distribuida, una computadora que procesa un administrador de base de datos distribuida.

Normalización. Proceso por el que se evalúa una relación para determinar si está o no en una forma normal especificada y, si es necesario, convertirla en relaciones que estén en dicha forma normal especificada.

Número de cambio de sistema (SCN). En Oracle, un valor amplio de una base de datos que se utiliza para ordenar los cambios realizados a los datos de la base de datos. El SCN se incrementa cada vez que se realiza algún cambio en la base de datos.

Objeto. (1) Objeto semántico, (2) estructura en un programa orientado a objetos que contiene una estructura de datos encapsulados y métodos de datos. Tales objetos se organizan en una jerarquía para que los objetos puedan heredar los métodos de sus padres. (3) En sistemas de seguridad, una unidad de datos protegida por una contraseña u otros medios.

Objeto ActiveX. Objeto COM que da soporte a la versión limitada de la especificación del objeto OLE.

Objeto arquetipo/versión. Estructura de dos objetos que representa versiones múltiples de un elemento estandarizado; por ejemplo, un PRODUCTO DE SOFTWARE SUAVE (el arquetipo) y VERSIÓN DEL PRODUCTO (la versión del arquetipo). El identificador de la versión siempre incluye el objeto arquetipo.

Objeto COM. Objeto acorde al estándar COM.

Objeto combinado. Objeto que contiene cuando menos algún otro objeto.

Objeto compuesto. Objeto con por lo menos un atributo de valores múltiples o grupo de atributos. Se llama objeto compuesto debido a que la llave de la relación que representa el atributo de valores múltiples es una llave compuesta.

Objeto de asociación. Un objeto que representa la combinación de cuando menos otros dos objetos, y que contiene datos sobre dicha combinación. A menudo se utiliza en las aplicaciones de contratación y asignación.

Objeto de columna. En Oracle, una estructura de objeto que se define y se soporta en una columna de tabla.

Objeto de filas. En Oracle, una tabla que contiene objetos y sus filas.

Objeto de generalización. Objeto que contiene los objetos subtipos. El objeto de generalización y sus subtipos tienen todos la misma clave. Los objetos subtipos heredan atributos del objeto de generalización. Un objeto de generalización se llama también objeto subtipo.

Objeto híbrido. Objeto que contiene un grupo de valores múltiples, el cual posee cuando menos un atributo de objeto.

Objeto implícito. Objeto que existe en la mente del usuario cuando él o ella solicitan un reporte "clasificado por x" o "agrupado por x". Por ejemplo, cuando el usuario solicita todos los PEDIDOs clasificados por Fecha de orden, el objeto implícito es el conjunto de todos los objetos PEDIDO.

Objeto inmutable. En el estándar ODMG, un objeto cuyos atributos no se pueden cambiar.

Objeto mutable. En el estándar ODMG, un objeto cuyos atributos se pueden cambiar.

Objeto OLE. Objeto de enlace e implementación de objetos. Los objetos COM que soportan las interfaces para implementarse dentro de otros objetos.

Objeto persistente. Objeto OOP que se ha escrito para el almacenamiento persistente.

Objeto simple. Un objeto que contiene atributos no repetidos y atributos de no objetos.

Objeto transitorio. En OOP, un objeto que no se ha escrito en un almacenamiento permanente. El objeto se perderá cuando termine el programa.

Ocurrencia de entidad. Una ocurrencia particular de una entidad, por ejemplo, Empleado 100 y el Departamento de contaduría.

Ocurrencia de objeto. La ocurrencia de un objeto semántico en particular; por ejemplo, el objeto semántico VENDEDOR en el cual el Apellido es igual a Jones.

ODBC. Véase Estándar de conectividad abierta de una base de datos.

ODMG-93. Reporte que emite el Grupo de administración de datos de objetos, el cual es un consorcio integrado por vendedores de bases de datos de objetos y otros expertos industriales interesados. El reporte aplica las ideas de otro grupo, el Grupo de administración de objetos, al problema de las bases de datos de objetos. El primer reporte ODMG se produjo en 1993 y por consiguiente se llama ODMG-93.

OLAP. Procesamiento analítico en línea; una forma de presentación de datos en la cual los datos se resumen agregados, desagregados. y vistos en el marco de una tabla o cubo.

OLE DB. La fundación basada en COM de acceso de datos en el mundo Microsoft. Los objetos OLE DB dan soporte el estándar del objeto OLE. ADO se basa en OLE DB.

OOP. Véase Programación orientada a objetos.

Padre. Una fila, registro o nodo en el lado uno de la relación uno a muchos.

Página del servidor activo. Véase ASP.

Página del servidor Java (JSP). Combinación de HTML y Java que se compila en un servelet de Java, que es una subclase de la clase HttpServelet. El código Java implementado en un JSP tiene acceso a los objetos y métodos HTTP. Las páginas JSP se utilizan de igual manera que las ASP, pero están compiladas en vez de interpretadas, como lo están las páginas ASP.

Pared de fuego (firewall). Una computadora que sirve como salida de seguridad entre una intranet e Internet. Las paredes de fuego monitorean la fuente y el destino del tráfico de la red y lo filtran.

Partición. (1) Parte de una base de datos distribuida; (2) la parte de una red que está separada del resto de la red durante una falla de ésta.

Partición horizontal. Subconjunto de una tabla que consta de filas completas de la tabla. Por ejemplo, en una tabla con 10 filas, las primeras cinco filas.

Partición mezclada. Una combinación de una partición vertical y horizontal; por ejemplo, en una tabla con cinco columnas y cinco filas, las primeras tres columnas de las primeras tres filas.

Partición vertical. Un subconjunto de las columnas de una tabla. Por ejemplo, en una tabla con 10 columnas, las primeras cinco columnas.

PERL. Lenguaje de extracción y reporte práctico. Un lenguaje de programación interpretado, el cual desarrolló Larry Wall y originalmente fue desarrollado por UNIX. Actualmente, los interpretadores del PERL están disponibles para Windows y Macintosh. A menudo se utiliza para programas de servidor en Internet e intranets.

Persistencia de objeto. En la programación orientada a objetos, la característica que puede guardar un objeto en la memoria no volátil, como por ejemplo un disco. Los objetos persistentes existen entre las ejecuciones de un programa.

PL/SQL. Lenguaje de programación/SQL. Un lenguaje que proporciona Oracle, el cual aumenta el SQL con las estructuras de lenguajes de programación, tales como rizos, bloques, si entonces (if then), y otros constructores parecidos. El PL/SQL se utiliza para crear procedimientos almacenados y disparadores.

Polimorfismo. En OOP, la situación en la cual se puede utilizar un nombre para invocar diferentes funciones. El polimorfismo en el cual las funciones se distinguen por tener secuencias de parámetros distintas se llama poliformismo paramétrico. En este último, los nombres se resuelven mediante el compilador en el tiempo de compilación. El polimorfismo en el cual las funciones se distinguen por la herencia de objetos se llama polimorfismo de herencia. Tales nombres se resuelven en el tiempo de ejecución al determinar el tipo de objeto que se invoca.

Prevención del bloqueo (abrazo) mortal. Forma de manejo de las transacciones para que no pueda presentarse el bloqueo mortal.

Primera forma normal. Cualquier tabla que encaje en la definición de una relación.

Problema de actualización concurrente. Una condición de error en la cual los cambios de datos de un usuario los sobreescriben los cambios de datos de otros usuarios. Es lo mismo que el problema de actualización perdida.

Problema de lectura inconsistente. Anomalía que se presenta en el procesamiento concurrente, en el cual las transacciones ejecutan una serie de lecturas inconsistentes con alguna otra. Se puede prevenir con un bloqueo de dos fases y otras estrategias.

Problema de pérdida de actualización. Es lo mismo que el problema de actualización concurrente.

Procedimiento almacenado. Colección de instrucciones SQL almacenada como un archivo que puede solicitarse mediante una instrucción. Usualmente, los productos DBMS proporcionan un lenguaje de creación de procedimientos almacenados que le agrega al SQL constructores de lenguajes de programación. Para dicho fin, Oracle proporciona el PL/SQL; el servidor SQL proporciona TRASNACT/SQL. Con algunos productos, los procedimientos almacenados que se pueden escribir en un lenguaje estándar como Java. Por lo general, los procedimientos almacenados se almacenan dentro de la misma base de datos.

Procesamiento analítico en línea. Véase OLAP.

Procesamiento concurrente. En las aplicaciones del teleprocesamiento, es el hecho de que varias transacciones compartan la CPU. A cada transacción se le asigna la CPU de una manera circular o en alguna otra manera durante cierto periodo de tiempo. Las operaciones se llevan a cabo tan rápidamente que a los usuarios les parecen simultáneas. En las redes de área local y otras aplicaciones distribuidas, se utiliza el procesamiento concurrente para referirse al procesamiento de aplicaciones (posiblemente simultáneo) en computadoras múltiples.

Procesamiento de base de datos distribuida. Procesamiento de base de datos en el cual se recuperan y actualizan los datos de las transacciones a través de dos o más computadoras independientes, y por lo general geográficamente distribuidas.

Producto. Una operación relacional en dos relaciones, A y B, que produce una tercera C, con C que contiene la concatenación de cada renglón en A con cada renglón en B. Es lo mismo que un producto cartesiano.

Producto cartesiano. Operación relacional en dos relaciones, A y B, que produce una tercera relación, C; con C que contiene la concatenación de cada renglón en A con cada renglón en B.

Programa de aplicación. Un programa desarrollado de acuerdo con las necesidades del cliente para el procesamiento de una base de datos. Puede escribirse en un lenguaje de procedimientos estándar como COBOL, C++, C o Visual Basic, o en un lenguaje único para el DBMS.

Programación orientada a objetos. Un estilo de la programación computacional en el cual los programas se desarrollan como conjuntos de objetos que tienen miembros de datos y métodos. Los objetos interactúan entre sí al llamar los métodos de cada uno.

Propiedad. Sinónimo de atributo.

Propietario. En la administración de datos, el departamento u otra unidad organizacional que se encarga de la administración de un elemento de datos en particular. Un propietario también puede llamarse proponente de datos.

Propietario de datos. Sinónimo de proponente de datos.

Proponente. Véase Proponente de datos.

Proponente de datos. En la administración de datos, un departamento u otra unidad organizacional a cargo del mantenimiento de un elemento particular de datos.

Protección inconsistente. Archivo de respaldo que contiene cambios no realizados.

Protocolo de Access de Objeto Simple (SOAP). Protocolo para las llamadas del procedimiento de transmisión, así como para los documentos pequeños XML que utilicen HTTP.

Protocolo de aplicaciones inalámbricas (WAP). Un protocolo de comunicaciones desarrollado para crear un estándar mundial para el acceso de redes desde dispositivos inalámbricos. Véase también Lenguaje de marcaje inalámbrico.

Protocolo de transferencia de hipertexto. HTTP; un medio estandarizado para la utilización de TCP/IP para la comunicación de los documentos http en redes.

Prototipo. Una demostración rápidamente desarrollada de una aplicación, o una parte de una aplicación.

Proveedor de datos. Proveedor de la funcionalidad OLE DB. Ejemplos de éste son los proveedores de datos tabulares y los proveedores de datos de servicios.

Proveedor de datos tabulares. Un proveedor de datos tabulares OLE DB que presenta datos en la forma de conjuntos de filas.

Proveedor de servicios. Un proveedor de datos del OLE DB que transforma los datos. Un proveedor de servicios es un consumidor de datos y un proveedor de datos.

Proyección. Una operación algebraica relacional realizada en una relación, A, que resulta en una relación, B, donde B tiene un subconjunto (posiblemente inadecuado) de los atributos de A. La proyección se utiliza para formar una nueva relación que reordena los atributos en la relación original, o que tiene solamente algunos de los atributos de la relación original.

Punto de verificación. Punto de sincronización entre una base de datos y un registro de transacciones. Todas las memorias intermedias se escriben forzosamente en el almacenamiento externo. Ésta es la definición estándar del punto de comprobación, pero algunas veces los proveedores de DBMS utilizan este término de otras maneras.

QBE. Consulta por ejemplo. Un estilo de interfaces de consultas, desarrollado primero por IBM aunque ahora lo utilizan otros proveedores, el cual permite a los usuarios expresar sus consultas al proporcionar ejemplos de los resultados que buscan.

Raíz. El registro, renglón o nodo de la parte superior de un árbol. Una raíz no tiene padre.

Rama. Subelemento de un árbol que puede constar de uno o varios nodos.

RDS. Véase Servicios de datos remotos.

Rebanada. En OLAP, una dimensión o medida que se mantiene constante durante un despliegue.

Recuperación progresiva. Proceso de recuperación de una base de datos mediante la aplicación de las imágenes después a una copia guardada de la base de datos, para llevarla al punto de verificación en la cual la base de datos sea lógicamente consistente.

Recuperación regresiva. El proceso de recuperar una base de datos en el cual se aplican imágenes antes a la base de datos para regresar a un punto de verificación u otro punto anterior en el que la base de datos sea lógicamente consistente.

Red. (1) Grupo de computadoras interconectadas. (2) Una intranet. (3) Internet.

Red compleja. Colección de entidades, objetos o relaciones y sus relaciones, de la cual cuando menos una de las relaciones es compleja (muchos a muchos).

Red simple. (1) Conjunto de tres relaciones y dos relaciones binarias en el cual una de las relaciones R, tiene una relación muchos a uno con las otras dos relaciones. El renglón en R tiene dos padres y los padres son de diferentes tipos. (2) Cualquier conjunto de tablas y relaciones que contenga la estructura definida en (1).

Registro. (1) Un grupo de campos pertenecientes a la misma entidad; se utiliza en los sistemas de procesamiento de archivos. (2) En un modelo relacional, un sinónimo de renglón y tuple.

Registro de incidencias. Archivo que contiene un registro de los cambios de la base de datos. El registro de incidencias contiene imágenes antes e imágenes después.

Reglas de autorización. Conjunto de permisos de procesamiento que describe qué usuarios o grupos de usuarios pueden realizar acciones particulares en determinadas partes de la base de datos.

Relación. Un arreglo bidimensional que contiene las entradas de un solo valor y de renglones no duplicados. El significado de las columnas es el mismo en cada renglón. El orden de los renglones y las columnas es inmaterial.

Relación 1.N. Abreviatura de una relación uno a muchos entre los renglones de una tabla.

Relación binaria. Una asociación entre dos entidades, objetos o renglones de relaciones.

Relación de intersección. Relación que se utiliza para representar una relación muchos a muchos. Contiene las claves de las relaciones en la relación. Cuando se utiliza para representar objetos compuestos muchos a muchos, no tiene datos sin clave. Cuando se utiliza para representar las entidades que tienen una relación muchos a muchos, puede tener datos sin clave si la relación contiene datos.

Relación ES UN. Una relación entre dos entidades u objetos del mismo tipo lógico. Con respecto a INGENIERÍA ES UN(n) EMPLEADO, ambas entidades son empleados y son del mimo tipo lógico. Compárese esto con una relación TIENE UN.

Relación punto de entrada. Se utiliza en las relaciones que representan un objeto. La relación punto de entrada es aquella cuya llave es la misma que la del objeto que representa. El punto de entrada por lo general es la primera relación que se procesa. Asimismo, el nombre de la relación punto de entrada generalmente es la misma que el nombre del objeto.

Relación recursiva. Relación entre entidades, objetos o filas del mismo tipo. Por ejemplo, si CLIENTEs se refiere a otros CLIENTEs, la relación a la que *hace referencia* es recursiva.

Relación TIENE-UN. Una relación entre dos entidades u objetos que son de diferentes tipos lógicos, por ejemplo, EMPLEADO TIENE UN(n) AUTO. Compárese esto con una relación ES-UN.

Renglón. Un grupo de columnas en una tabla. Todas las columnas en un renglón pertenecen a la misma entidad. Un renglón es lo mismo que un tuple y un registro.

Repetición de llamada. Una práctica del diseño OOP mediante la cual un objeto pasa su identidad a otro, con el fin de que el objeto llamado notifique al objeto que llama cuando se presenten ciertos sucesos. A menudo, el suceso es la destrucción del objeto llamado, pero también se utiliza con otras finalidades.

Reporte. Extracción de datos de una base de datos. Los reportes se pueden imprimir, desplegar en un monitor de computadora o almacenar en un archivo. Un reporte es parte de una aplicación de base de datos. Compárese esto con una Forma.

Repositorio. Colección de metadatos sobre la estructura de base de datos, aplicaciones, páginas Web, usuarios y otros componentes de aplicación. Los repositorios activos se mantienen automáticamente mediante las herramientas en el ambiente de desarrollo de la aplicación. Los repositorios pasivos se deben mantener manualmente.

Respaldo consistente. Archivo de respaldo desde el cual se han eliminado todos los cambios que no se realizaron.

Respaldo diferencial. Un archivo de respaldo que sólo contiene los cambios realizados después de un respaldo anterior.

Restricción. Regla que concierne a los valores de atributos permitidos cuya veracidad se pueden evaluar. Por lo general, una restricción no incluye reglas dinámicas, como “el pago al vendedor nunca puede disminuir”, o “el salario de ahora debe ser mayor al del trimestre pasado”.

Restricción de cardinalidad de una relación. Una restricción en el número de filas que pueden participar en una relación. Las restricciones de cardinalidad mínima determinan el número de filas que deben participar; las restricciones de cardinalidad máxima especifican el mayor número de filas que pueden participar.

Restricción de integridad referencial. Una restricción de relación en los valores de la clave ajena. Una restricción de integridad referencial especifica que los valores de una llave externa deben ser un subconjunto apropiado de los valores de la llave primaria a la que hace referencia.

Restricción de interrelación. Restricción que requiere el valor de un atributo en un renglón de una relación para asociar el valor de un atributo encontrado en otra relación. Por ejemplo, Número de cliente en PEDIDO debe ser igual a Número de cliente en CLIENTE.

Restricción de la relación. Restricción de integridad referencial, o restricción de cardinalidad de la relación.

ROLAP. OLAP relacional que utiliza un DBMS relacional para dar soporte al procesamiento OLAP.

Salida real. La salida transmitida al cliente de un sistema de información, tal como una confirmación de un pedido. Cuando se produce un error no se pueden cambiar dichas salidas mediante la recuperación de la base de datos, sino que se deben ejecutar las de compensación.

SAX. API simple (interfaz del programa de aplicación) para XML. Un analizador basado en eventos que le informa a un programa cuando se han encontrado los elementos de un documento XML durante el análisis del documento.

SCN. Véase Número de cambio de sistema.

Segmento de recuperación regresiva. En Oracle, se utiliza una memoria intermedia para almacenar las imágenes antes con los fines del control de concurrencia y el registro de transacción. Los segmentos de recuperación regresiva se pueden archivar y utilizar subsecuentemente para la recuperación.

Segunda forma normal. Relación en la primera forma normal en la que todos los atributos sin clave dependen de todas las claves.

Seguridad vertical. Acceso limitado a ciertas columnas de una tabla o junta.

Selección. Operación algebraica relacional ejecutada en una relación, A, que produce una relación, B, con B que contiene solamente las filas en A que satisfacen las restricciones especificadas en la selección.

Serializable. Un nivel de aislamiento de transacción que no permite la lectura sucia, las lecturas no repetibles y las lecturas fantasmas.

Servicio de transacción distribuida (DTS). Servicio OLE desarrollado por Microsoft que soporta el procesamiento distribuido y, en particular, implementa dos fases realizadas.

Servicios de datos remotos. Conjunto de controles y características ActiveX que permite que los datos se puedan almacenar en una máquina cliente y puedan formatearse, clasificarse y filtrarse sin ayuda del servidor Web.

Servidor de archivos. En una red de área local, es una computadora personal la cual contiene un archivo que procesa lo de otras computadoras personales en la red. El término servidor de archivos por lo general se utiliza para la arquitectura de recursos compartidos. Véase Computadora cliente, arquitectura de base de datos cliente-servidor, servidor de base de datos y arquitectura de recursos compartidos.

Servidor de base de datos. (1) En una red de área local con arquitectura de base de datos cliente servidor, la computadora que ejecuta el DBMS y procesa la ejecución de acciones en la base de datos a nombre de sus computadoras clientes. (2) En la arquitectura de tres capas y de capas múltiples, una computadora que hospeda un DBMS y responde a los requisitos de la base de datos desde el servidor Web.

Servidor de información de Internet. Véase IIS.

Servidor Web. En la arquitectura de tres capas, una computadora que se sitúa entre las computadoras cliente y el servidor de base de datos y hospeda a un servidor HTTP.

Servlet. Programa codificado en bytes compilado e independiente de una máquina Java que se ejecuta mediante una máquina virtual de Java localizada en un servidor Web.

Servlet de Java. Véase Servlet.

Sistema cliente servidor. Sistema de dos o más computadoras en el cual por lo menos una proporciona los servicios para una o más computadoras. Los servicios pueden ser servicios de base de datos, de comunicación, de impresión o alguna otra función.

Sistema de administración de base de datos distribuida (DDBMS). En una base de datos distribuida, es la colección de la transacción distribuida y los administradores de base de datos en todas las computadoras.

Sistema de base de datos distribuida. Sistema distribuido en el cual una base de datos, o porciones de ésta, se distribuyen en dos o más computadoras.

Sistema de procesamiento de archivos. Sistema de información en el cual se almacenan los datos en archivos por separado. No hay ningún diccionario de datos integrado. Generalmente, el formato de los archivos se almacena en programas de aplicación.

Sistema distribuido. Sistema en el que los programas de aplicación de una base de datos se procesan en dos o más computadoras.

SOAP. Véase Protocolo de Access de objeto simple.

SQL. Lenguaje de consulta estructurado; un lenguaje para la definición de la estructura y el procesamiento de una base de datos relacional. Se utiliza como un lenguaje de consulta autónomo, o puede implementarse en programas de aplicación. El SQL es aceptado como un estándar por el American National Standards Institute (Instituto Nacional Americano de Estándares). Fue desarrollado por IBM.

SQL3. El SQL3 es una extensión del estándar de la base de datos SQL92 que incluye soporte para la administración de base de datos orientada a objetos. Desarrollado por los comités de estandarización ANSI X3H2 e ISO/IEC JTC1/SC2/WG3.

Sublenguaje de datos. Lenguaje para la definición y procesamiento de una base de datos diseñada para implementarse en programas escritos en otro lenguaje en la mayoría de los casos, un lenguaje de procedimientos como COBOL, C++, o Visual Basic. Un sublenguaje de datos es un lenguaje de programación incompleto, puesto que sólo contiene construcciones para el acceso de datos.

Subsistema de definición de herramientas. Parte del programa DBMS utilizado para definir y cambiar la estructura de la base de datos.

Subsistema de diccionario de datos y de administración de bases de datos. Colección de programas en el DBMS utilizados para accesar al diccionario de datos y ejecutar las funciones de la administración de la base tales como el mantenimiento de contraseñas y la ejecución del respaldo y recuperación.

Subsistema de interfaz de procesamiento. La porción de las rutinas DBMS que ejecuta instrucciones para el procesamiento de la base de datos. Acepta el ingreso de los programas de consulta interactivos y de programas de aplicación escritos en lenguajes estándar, o en lenguajes específicos del DBMS.

Subtabla. En SQL3, una tabla que es un subtipo de una segunda tabla llamada supertabla.

Subtipo. En jerarquías de generalización, una entidad u objeto que es una subespecie o subcategoría de un tipo de nivel superior. Por ejemplo, INGENIERO es un subtipo de EMPLEADO.

Supertabla. En SQL3, una tabla que tiene una o más subtablas definidas en ella.

Supertipo. En jerarquías de generalización, una entidad u objeto que lógicamente contiene subtipos. Por ejemplo, EMPLEADO es un supertipo de INGENIERO, CONTADOR, y REPRESENTANTE.

Tabla base. En las implementaciones relacionales, la tabla desde donde se definen las vistas relacionales.

Tarjeta. En el lenguaje de generación inalámbrico, una parte de un documento XML que es lo suficientemente pequeño como para desplegarlo en un dispositivo inalámbrico. Véase también conjunto de tarjetas.

TCP/IP. Programa de control terminal/Protocolo de Internet.

Tercera forma normal. Una relación en la segunda forma normal que no tiene dependencias transitivas.

Tipo de cursor. Una declaración en el cursor que determina la manera en la cual el DBMS coloca los bloqueos implícitos. Los cuatro tipos de cursor que se analizan en este texto son: deslizable, instantáneo, de teclado y dinámico.

Tipo de datos abstractos. En el SQL3 es una estructura que define el usuario, la cual tiene métodos, elementos de datos e identificadores; es una versión de un objeto OOP. La persistencia se proporciona mediante la unión del ADT con una columna de una relación.

Transacción. (1) Una transacción atómica. (2) El registro de un suceso en el mundo de los negocios.

Transacción ACID. Acrónimo que representa lo atómico, consistente, aislado y durable. Una transacción atómica es aquella en la cual se realizan todos los cambios de la base de datos como unidad; se hacen todos los cambios o ninguno. Una transacción consistente es aquella en la cual se realizan todas las acciones en las filas en el mismo estado lógico. Una transacción aislada es la que está protegida de cambios que realicen otros usuarios. Una transacción durable es aquella que, una vez que se confirma en la base de datos, es permanente, a pesar de las fallas subsecuentes. Existen diferentes niveles de consistencia y aislamiento. Véase la consistencia del nivel de la transacción y la consistencia del nivel de la instrucción. Véase también el nivel de aislamiento de la transacción.

Transacción atómica. Grupo de operaciones de base de datos lógicamente relacionadas que se llevan a cabo como unidad. Se realizan todas las operaciones, o no se realiza ninguna.

TRANSACT/SQL. Un lenguaje proporcionado por Microsoft que es parte del servidor SQL. Le agrega al SQL estructuras de lenguaje de programación, tales como rizos while, bloques Si entonces otro, y otros constructores parecidos. TRANSCT/SQL se utiliza para crear procedimientos almacenados y disparadores.

Transparencia de concurrencia. En un sistema de distribución de base de datos, es la condición en la cual los programas de aplicación no saben, ni necesitan saber, si los datos se están procesando simultáneamente. El DDBMS organiza las actividades de actualización para que los resultados producidos, cuando se está realizando el procesamiento concurrente, sean consistentes con los resultados que se presentarían si no hubiera un procesamiento concurrente.

Transparencia de duplicación. En un sistema de base de datos distribuida, se refiere a la condición en la cual los programas de aplicación no saben ni necesitan saber si los datos se duplican. Si se duplican, el DDBMS asegurará que todas las copias se actualicen consistentemente, sin involucrar al programa de aplicación.

Transparencia de falla. En un sistema de base de datos distribuida, la condición en la cual los programas de aplicación se aíslan de la falla.

Transparencia de localización. En un sistema de base de datos distribuida, la condición en la cual los programas de aplicación no saben ni necesitan saber dónde se localizan los datos. El DBMS encuentra los datos, donde quiera que estén ubicados, sin involucrar al programa de aplicación.

Trayectoria X. Un sublenguaje dentro de XSLT que se utiliza para identificar qué partes de un documento XML se transformarán. También se puede utilizar para cálculos y manipulación de registros. Se mezcla con SXLT.

Tuple. Lo mismo que Renglón.

UML. Lenguaje de modelado unificado; conjunto de estructuras y técnicas para el modelado y diseño de programas y aplicaciones orientados a objetos. Es tanto una metodología como un conjunto de herramientas para dicho desarrollo. UML incorpora el modelo entidad-relación para el modelado de datos.

Unión. Una operación algebraica relacional realizada en dos relaciones compatibles con la unión, por ejemplo A y B, que forma una tercera relación, C, con C que contiene cada renglón en A y B, menos algún renglón duplicado.

Valor calculado. Una columna de una tabla que se calcula a partir de los valores de otras columnas. Los valores no se almacenan, pero se calculan cuando se despliegan.

Valor nulo. Valor de un atributo que nunca se ha proporcionado. Dichos valores son ambiguos y pueden significar que: (a) el valor es desconocido, (b) el valor no es apropiado, o (c) el valor está en blanco.

Variable anfitriona. Una variable en un programa de aplicación en el cual un DBMS coloca un valor de la base de datos.

VBScript. Un lenguaje de fácil interpretación y aprendizaje que utiliza servidor Web y el procesamiento de aplicaciones del cliente Web; un subconjunto de Microsoft Visual Basic.

Vista. Lista estructurada de elementos de datos de entidades u objetos semánticos definidos en el modelo de datos.

Vista actualizable. Vista SQL que se puede actualizar. Por lo general, tales vistas son muy simples y las reglas que permiten la actualización son muy restrictivas. Las vistas no actualizables normalmente se pueden actualizar mediante la escritura de disparadores EN VEZ DE en aplicaciones específicas.

Vista de objeto. La parte de un objeto semántico que es visible para una aplicación en particular. Una vista consta del nombre del objeto semántico, más una lista de los atributos visibles en dicha vista.

Vista de objeto semántico. La parte de un objeto semántico que es visible en una forma o reporte.

Vista del usuario. Vista de una base de datos de un usuario particular.

Vista SQL. Relación que se construye a partir una sola instrucción SQL SELECT. Las vistas SQL tienen cuando mucho una trayectoria de valores múltiples. En la mayoría de los productos DBMS, incluidos Access, Oracle y el servidor SQL, el término *vista* significa vista SQL.

WAP. Véase Protocolo de aplicaciones inalámbricas.

WML. Véase Lenguaje de marcaje inalámbrico.

WML Script. Un lenguaje de escritura que es parte del lenguaje de generación inalámbrico. Es una variante del ECMAScript, mejor conocido como Java Script.

XML. Lenguaje de marcaje extensible; un lenguaje de marcaje estándar que proporciona una separación clara entre la estructura, el contenido y la materialización; puede representar jerarquías arbitrarias y, por lo tanto, se puede utilizar para transmitir cualquier vista de base de datos.

XQL. Un estándar para la expresión de consultas de base de datos como documentos XML. La estructura de la consulta utiliza dispositivos de trayectoria, X y el resultado de la consulta se representa en un formato XML. Se encuentra en desarrollo y probablemente será importante en el futuro.

XSL. (1) Primer significado: Lenguaje de estilo extensible. Originalmente un dispositivo para la transformación de XML en estilos potencialmente complicados. Dividido en dos partes. XSLT para la transformación de estructura de documentos y producción de formatos simples, y XSL-FO para el formateo de documentos en estilos complicados y sofisticados. (2) Significado actual: hoja de estilo XSLT. El documento que proporciona los pares {asociación, acción} y otros datos para que los utilice el XSLT cuando transforma un documento XML.

XSL-FO. Lenguaje de estilo extensible-Formateo de objetos. La parte del XSL original que se abandona después de que se eliminó el XSLT. Un estándar muy grande y complicado para el formateo sofisticado, el cual aún está en desarrollo.

XSLT. Lenguaje de estilo extensible: transformaciones. Un programa (o proceso) que aplica las hojas de estilo XSLT a un documento XML para producir un documento transformado XML.

BIBLIOGRAFÍA

Artículos de bases de datos

Visite los siguientes sitios para encontrar artículos recientes sobre los distintos temas de las bases de datos:
CNet—www.news.com
Database Programming and Design—www.dbpd.com
Databased Advisor—www.advisor.com
Intelligent Enterprise—www.intelligententerprise.com
PCMagazine—www.zdnet.com/pcmag
ZDNet—www.zdnet.com

DBMS y otros fabricantes

Java—java.sun.com
SQL Server—www.microsoft.com/sql
Oracle—www.oracle.com
MySQL—www.mysql.com
Tabledesigner—www.tabledesigner.com

Estándares

JDBC—java.sun.com/products/jdbc
y mmmysql.sourceforge.net
ODBC—www.liv.ac.uk/middleware/html/odbc.html
ODMG—www.odmg.org
SQL3—www.obis.com/x3h7/sql3.htm
Worldwide Web Consortium—www.w3.org
XML—www.xml.org

Libros y publicaciones

ANSI X3. *American National Standard for Information Systems—Database Language SQL*. ANSI, 1992.

Berson, Alex, & Stephen J. Smith. *Data Warehousing, Data Mining, and OLAP*. Nueva York: McGraw-Hill, 1997.

Boumphrey, Frank. *Professional Stylesheets for HTML and XML*. Chicago, IL: Wrox Press, 1998.

Bowen, Rich & Ken Coar. *Apache Server Unleashed*. Indianápolis, Indiana: SAMS, 2000.

Bruce, T. *Designing Quality Databases with IDEF1X Information Models*. Nueva York: Dorset House, 1992.

Chamberlin, D. D., et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control." *IBM Journal of Research and Development 20* (noviembre de 1976).

Chen, P. *Entity-Relationship Approach to Information Modeling*. E-R Institute, 1981.

Chen, P. "The Entity-Relationship Model: Toward a Unified Model of Data." *ACM Transactions on Database Systems 1* (marzo de 1976).

Coar, Ken A. L. *Apache Server for Dummies*. Foster City, California: IDG Books, 1997.

Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM 25* (febrero de 1970).

Codd, E. F. "Extending the Relational Model to Capture More Meaning." *Transactions on Database Systems 4* (diciembre de 1979).

Coffee, Peter. "No-sweat Database Design." *PC Week* (11 de marzo de 1996).

Corning, Michael, Steve Elfanbaum, & David Melnick. *Working with Active Server Pages*. Indianápolis, Indiana: Que Corporation, 1997.

Deitel, H. M, & P. J. Deitel. *Java: How to Program, 3rd Edition*. Upper Saddle River, Nueva Jersey: Prentice-Hall, 1999.

Embley, D. W. "NFQL: The Natural Forms Query Language." *ACM Transactions on Database Systems 14* (junio de 1989).

Eswaran, K. P., J. N. Gray, R. A. Lorie, & I. L. Traiger. "The Notion of Consistency and Predicate Locks in a Database System." *Communications of the ACM 19* (noviembre de 1976).

Fagin, R. "A Normal Form for Relational Databases That is Based on Domains and Keys." *Transactions on Database Systems 6* (septiembre de 1981).

Fagin, R. "Multivalued Dependencies and a New Normal Form for Relational Databases." *Transactions on Database Systems 2* (septiembre de 1977).

Feuerstein, Steven with Bioll Pribyl. *Oracle PL/SQL Programming*. Sebastopol, California: O'Reilly, 1997.

Fields, Duane K. & Mark A Kolb. *Web Development with Java Server Pages*. Greenwich CT: Manning Press, 2000.

Flanagan, David, Jim Farley, William Crawford, & Kris Magnusson. *Java Enterprise in a Nutshell*. Sebastapol, California: O'Reilly, 1999.

Fronckowiak, John W. *Microsoft SQL Server 7.0 Administration*. Microsoft Press, 1999.

Goldfarb, Charles F. & Paul Prescod. *The XML Handbook*. Upper Saddle River, Nueva Jersey: Prentice Hall, 1998.

Goodman, Danny. *Dynamic HTML: The Definitive Resource*. Sebastopol, California: O'Reilly and Associates, 1998.

Hall, Marty. *Core Servlets and Java Server Pages*. Upper Saddle River, Nueva Jersey: Sun Microsystems Press, 2000. ¡Recomendado ampliamente!

Hammer, M., & D. McLeod. "Database Description with SDM: A Semantic Database Model." *Transactions on Database Systems 6* (septiembre de 1981).

Harold, Elliotte Rusty. *XML: Extensible Markup Language*. Nueva York: IDG Books Worldwide, 1998.

Kay, Michael. *XSLT: Programmer's Reference*. Birmingham, Reino Unido, Gran Bretaña: WROX Press, 2000.

Keuffel, Warren. "Battle of the Modeling Techniques." *DBMS Magazine* (agosto de 1996).

Kroenke, David. "Waxing Semantic: An Interview." *DBMS Magazine* (septiembre de 1994).

Loney, Kevin & George Koch. *Oracle 8i, The Complete Reference*. Berkeley, California: Osborene/McGraw-Hill, 2000.

Loomis, M. E. S. *Object Databases, The Essentials*. Reading, Massachussets: Addison-Wesley, 1995.

McLaughlin, Brett. *Java and XML*. Sebastopol, California: O'Reilly, 2000.

Meyers, Nathan. *Java Programming on Linux*. Indianápolis, Indiana: Waite Group Press, 2000.

Monson-Haefel, Richard. *Enterprise Java Beans, 2nd Edition*. Sebastopol, California: O'Reilly, 2000.

Moriarty, T. "Business Rule Analysis." *Database Programming and Design* (abril de 1993).

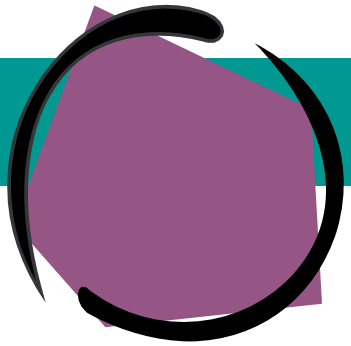
Muench, Steve. *Building Oracle XML Applications*. Sebastapol, California: O'Reilly, 2000.

Muller, Robert J. *Database Design for Smarties: Using UML for Data Modeling*. San Francisco, California: Morgan Kaufmann Publishers, 1999.

Nijssen, G., & T. Halpin. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. Englewood Cliffs, Nueva Jersey: Prentice-Hall, 1989.

Nolan, R. *Managing the Data Resource Function*. St. Paul: West Publishing, 1974.
Red Hat, Inc. *Linux 6.2 Getting Started Guide*. Durham, Carolina del Norte: Red Hat, Inc., 2000.
Ricart, Manuel Alberto. *The Complete Idiot's Guide to Linux, 2nd Edition*. Indianápolis, Indiana: Que Publishing, 2000.
Rogers, Dan. "Manage Data with Modeling Tools." *VB Tech Journal* (diciembre de 1996).
Schumate, John. *A Practical Guide to Microsoft OLAP Server*. Boston, Massachussets: Addison Wesley, 2000.
Sturm, Jake. *Data Warehousing with Microsoft SQL Server 7.0*. Redmond, WA: Microsoft Press, 2000.
Thakkar, Meghraj. *Teach Yourself Oracle 8i on Windows NT*. Indianápolis, Indiana: SAMS, 1999.
Thomsen, Erik. *OLAP Solutions: Building Multidimensional Information Systems*. Nueva York: John Wiley and Sons, 1997.
Weissinger, A. Keyton. *ASP in a Nutshell, 2nd Edition*. Sebastopol, California: O'Reilly, 2000.
Zloof, M. M. "Query by Example." *Proceedings of the National Computer Conference*, AFIPS 44 (mayo de 1975).

ÍNDICE



M

S

T

U

V

X